Київський національний університет імені Тараса Шевченка

КРЕНЕВИЧ А.П.

ОБВІНЦЕВ О.В.

С
У ЗАДАЧАХ І ПРИКЛАДАХ

Навчальний посібник
із дисципліни "Інформатика та програмування"

Рекомендовано
Міністерством освіти і науки, молоді та спорту України

КИЇВСЬКИЙ УНІВЕРСИТЕТ 2012

УДК 004.438С(075.8); 519.682
ББК 32.973.26-018я73
К79

Рецензенти:
д-р фіз.-мат. наук, проф. Ю.А. Бєлов,
д-р фіз.-мат. наук, проф. Ю.М. Онопчук,
д-р фіз.-мат. наук, проф. В.О. Капустян

Рекомендовано Міністерством освіти і науки, молоді та спорту України
як навчальний посібник для студентів вищих начальних закладів

Креневич, А.П.

К79 С у задачах і прикладах : навчальний посібник із дисципліни "Інформатика та програмування" / А.П. Креневич, О.В. Обвінцев. – К. : Видавничо-поліграфічний центр "Київський університет", 2011. – 208 с.

ISBN 978-966-439-488-5

Призначено для практичного опанування процедурного програмування із застосуванням мови С. Охоплює основні розділи, що викладаються у межах курсів програмування в багатьох вищих навчальних закладах.

Систематизовано подано короткі теоретичні відомості, типові приклади розв'язання задач і задачі для самостійної роботи.

Для студентів молодших курсів університетів і викладачів, що проводять практичні заняття з програмування.

УДК 004.438С(075.8); 519.682
ББК 32.973.26-018я73

Гриф надано Міністерством освіти і науки,
молоді та спорту України
(лист №1/1131 від 05.01.11)

ISBN 978-966-439-488-5

ВСТУП

Практичному програмуванню неможливо навчитися теоретично. Основний принцип вивчення будь-якої мови програмування полягає в необхідності розв'язання великої кількості різноманітних задач із застосуванням комп'ютера.

Основною метою посібника є засвоєння та закріплення теоретичного матеріалу, опанування технологій та прийомів написання програм і, що найголовніше, – отримання навичок, достатніх для самостійного розв'язання задач із програмування за допомогою комп'ютера.

Основною мовою програмування у посібнику є мова С [6, 10, 17, 18, 19, 20, 21, 22], визначена міжнародним стандартом ANSI C (1989). Для кращого сприйняття матеріалу, аби не відволікати читача зайвими умовностями та з метою зосередження уваги тільки на алгоритмах, у прикладах для введення та виведення даних здебільшого використано інструкції, визначені у наступнику мови С – мові С++.

С++ також використано у розділі 11, що присвячений основам об'єктно-орієнтованого програмування. Проте, слід зауважити: більшість задач є інваріантними щодо мови програмування, тому посібник може використовуватися як збірник задач із програмування будь-якою мовою.

Зауважимо, що п. 2.1 посібника присвячено основам алгебри висловлювань. Ця тема є розділом математичної логіки і, взагалі кажучи, має непряме відношення до програмування мовою С. Проте володіння її апаратом є невід'ємною частиною програмування та складовою освіти будь-якого програміста.

Посібник складається з 11 розділів, кожен з яких присвячений конкретній темі. Деякі з розділів поділені на пункти. Кожен розділ або пункт супроводжується теоретичним матеріалом в обсязі, достатньому для розв'язання задач, містить приклади розв'язання типових задач і перелік задач для самостійної роботи.

Приклади розв'язання задач супроводжуються детальними описами алгоритмів, а ключові моменти програм пояснено в коментарях.

Усі програми, наведені у посібнику, протестовані із застосуванням компілятора Borland C++.

Набір задач для самостійної роботи складений з урахуванням багаторічного досвіду викладання курсу програмування на механіко-математичному факультеті Київського національного університету імені Тараса Шевченка та частково базується на посібнику [7]. Використано також вправи з інших задачників [1, 8, 11, 13, 14, 15, 16].

Розв'язання задачі передбачає написання програми, налагодження її за допомогою комп'ютера з використанням компілятора С або С++ і тестування

щодо коректної роботи. Це стосується також випадку, коли у задачі вимагається написати тільки підпрограму: для розв'язання задачі потрібно написати підпрограму та програму, що її використовує.

Задачі та приклади деяких розділів можуть використовувати матеріал із попередніх розділів, що вимагає від читача послідовного опрацювання та засвоєння матеріалу.

Нумерація об'єктів у посібнику (прикладів, задач, рисунків тощо) складається з двох частин: номеру розділу та порядкового номеру об'єкта у розділі. Багато з представлених у посібнику задач мають підпункти, що позначено маленькими літерами українського алфавіту. При посиланні на задачу зазначається її номер, перед яким вказано слово "задача" (у потрібній формі). У посиланні також може бути позначений підпункт задачі.

Для запису абстрактних об'єктів (математичних і булевих змінних, векторів, матриць, рядків, абстрактних даних, математичних та алгоритмічних функцій тощо) використовується курсив. Для відображення фрагментів коду програм, інструкцій та ідентифікаторів, що використовуються при написанні програм, застосовується моноширинний шрифт, що полегшує процес сприйняття матеріалу.

Автори висловлюють щиру подяку доцентам Бублику В.В., Вакалу Є.С., Довгому Б.П., Попову В.В. за корисні поради, конструктивну критику та допомогу при створенні цього посібника.

Розділ 1
ЛІНІЙНІ ПРОГРАМИ

Означення 1.1. Точний перелік інструкцій, що описує порядок дій виконавця для розв'язання певної задачі за скінченну кількість операцій, називається **алгоритмом**.

Означення 1.2. Алгоритм, записаний за правилами деякої формалізованої знакової системи (**мови програмування**) та призначений для виконання обчислювальною машиною (**комп'ютером**), називається **програмою**.

Найпростіша програма мовою С має вигляд:

```
void main() { }
```

Вона визначає головну підпрограму `main`, що не має параметрів і не виконує жодних дій. Фігурні дужки використовують у С для групування операторів. У даному контексті вони визначають початок і кінець порожнього тіла підпрограми `main()`. Будь-яка програма мовою С має містити підпрограму `main()`, і робота програми починається з виконання саме цієї підпрограми.

Як правило, кожна програма є послідовністю інструкцій.

Означення 1.3. Інструкцією (або **оператором**) називається найменша автономна частина мови програмування.

Будь-яка інструкція (крім випадків, що буде оговорено окремо) у програмі, написаній мовою С, закінчується символом `';'` (крапка з комою).

Однією з базових конструкцій будь-якої мови програмування є змінна.

Означення 1.4. Змінною називається іменована область пам'яті, ім'я якої використовується для здійснення доступу до даних, що містяться у цій області пам'яті (змінній).

Ім'я змінної – це послідовність символів, що складається з латинських літер, цифр і символу `'_'`, причому першим символом не може бути цифра. Також слід пам'ятати, що у С великі та маленькі літери розрізняються, тому `a` та `A` – позначають різні змінні.

У С змінна характеризується певним типом, що дозволяє конкретизувати поведінку програми з такою змінною. Тому, перш ніж використовувати змінну, її потрібно **оголосити** (або **описати**). Для опису змінних використовується такий формат:

```
t v;
```

де `v` – ім'я змінної (або послідовність імен змінних, що перераховуються через кому), `t` – її (їхній спільний) тип.

Наразі використовуватимемо два числових типи:
`int` – цілочисельний тип;
`float` – дійснозначний тип.
Приклади оголошення змінних:

```
int n1, k2, _Max;
float a;
```

Над змінними числових типів виконують такі операції:
+, -, *, / – **арифметичні операції** – відповідно додавання, віднімання множення та ділення (для типу `int` ділення – це ділення націло);
% – для типу `int` – остача від ділення.
У мові С також визначено інструкції присвоєння, введення та виведення:
= – **присвоєння** – запис до змінної значення арифметичного виразу;
`cin` – **введення** – запис у змінну зчитаного з клавіатури значення;
`cout` – **виведення** – виведення значення змінної на екран;
`scanf` – **форматоване введення** з клавіатури;
`printf` – **форматоване виведення** на екран.
Приклади інструкції присвоєння й арифметичних операцій:

```
n1 = 1;                  // у змінну n1 записати 1
n1 = n1 + 3;             // збільшити n1 на 3
k2 = n1 * (n1 - 1) + 5;// дужки використовуються
                         // для зміни пріоритету
                         // арифметичних операцій
k2 = 999 % 100;          // k2 = 99
```

Зауваження. Інструкції `cin/cout` (введення/виведення) визначено в мові С++, що є нащадком мови С. Тому ці інструкції не можуть використовуватися для програм, написаних класичною С. Проте, оскільки правила їхнього використання значно простіші за інструкцій `scanf/printf`, то у посібнику вони частіше використовуються для демонстрації прикладів.
Для використання інструкцій `cin/cout` потрібно підключити бібліотеку `iostream.h`. Для цього на початку програми слід записати рядок

```
#include <iostream.h>
```

Правила використання інструкцій `cin/cout`:

```
cin >> variables;
cout << variables;
```

де `variables` – перелік змінних (які розділяються послідовністю символів ">>" для інструкції `cin` або "<<" – для інструкції `cout`), що беруть участь в інструкції введення/виведення.
Приклади інструкції введення:

```
cin >> n1;               // введення однієї змінної
cin >> a >> k2 >> _Max;// послідовне введення
                         // кількох змінних
```

Приклади інструкції виведення:

```
cout << a;              // виведення однієї змінної
cout << k2 << n1;       // послідовне виведення
                        // кількох змінних
cout << "_Max = ";      // виведення на екран
                        // повідомлення _Max =
```

Послідовність символів `"//"` використовується для позначення **однорядкових коментарів** у програмі. Послідовність будь-яких символів поточного рядка, що розміщується після початку коментаря, буде проігнорована компілятором під час компіляції програми. У C також використовують **багаторядкові коментарі**. Для створення багаторядкового коментаря використовується послідовність символів `"/*"` для початку коментаря та `"*/"` – для його закінчення. Таким чином, довільна послідовність символів (не обов'язково одного рядка), що розміщена між `"/*"` і `"*/"`, буде проігнорована під час компіляції.

Інструкції `scanf` і `printf` надають програмісту дещо ширші можливості для введення і виведення даних. Для використання операцій `scanf` і `printf` слід підключити бібліотеку `stdio.h`. Для цього на початку програми потрібно записати рядок

```
#include <stdio.h>
```

Правила використання інструкцій `scanf` і `printf`:

```
scanf("format", variables);
printf("format", variables);
```

де `format` – рядок формату – послідовність символів, серед яких можуть міститися керуючі послідовності. `variables` – перелік змінних (які розділяються комами), що беруть участь в інструкції введення/виведення. Для інструкції `scanf` перед кожною змінною із переліку ставиться символ `'&'`.

Під час виконання інструкції `scanf` користувач має ввести з клавіатури рядок формату, причому в місцях, де містяться керуючі послідовності, він має ввести значення відповідних змінних із переліку змінних. Під час виконання команди `printf` на екран виводиться рядок формату `format`, причому замість керуючих послідовностей на екран буде виведено значення відповідних змінних із переліку.

У спрощеній інтерпретації керуюча послідовність має вигляд:

`%[ширина][.точність][розмір]тип`

де % – обов'язковий символ, що вказує на початок керуючої послідовності. Параметри, `ширина`, `точність` і `розмір` не обов'язкові;

`ширина` – десяткове число – вказує мінімальну ширину поля (з урахуванням знаку для чисел), яку займає число;

`точність` – для дійсного типу вказує мінімальну кількість знаків, що має з'явитися після десяткової коми;

розмір – дозволяє вказати розмір даних, що вводяться або виводяться. Значення, яких набуває цей параметр, залежать від типу даних і будуть розглянуті нижче;

тип – специфікатор типу:

i – для змінної типу int;

f – для змінних типу float.

Наприклад, для введення двох змінних через кому, перша з яких – n1 – цілого типу, а друга – a – дійсного, можна скористатися рядком:

```
scanf("%i,%f", &n1, &a);
```

Якщо змінні потрібно ввести не через кому, а через символ пропуску, то вищенаведений рядок перетвориться на:

```
scanf("%i %f", &n1, &a);
```

Для прикладу роботи з операцією printf припустимо, що

```
a = 3.1415; n1 = 100;
```

тоді інструкція

```
printf("a = %4.2f, n1 = %i.", a, i);
```

виведе на екран рядок

```
a =    3.14, n1 = 100.
```

Для переведення програмою курсора на наступний рядок під час операції виведення використовують спеціальний символ '\n' – символ нового рядка. Таким чином, якщо попередню інструкцію модифікувати як

```
printf("a = %4.2f,\n n1 = %i.", a, i);
```

то програма на екран виведе

```
a =    3.14,
 n1 = 100.
```

Означення 1.5. Програма, складена тільки з інструкцій присвоєння, введення та виведення, називається **лінійною**.

Вищезгадані арифметичні операції та інструкція присвоєння є стандартними для всіх мов. У мові С та її наступниках визначено унікальний перелік операцій (табл. 1.1), які дозволяють зробити запис програми компактнішим.

Таблиця 1.1. Інкрементні та декрементні операції

Операція	Опис	Приклад	Еквівалент
++	збільшення змінної на 1	i++	i = i + 1
--	зменшення змінної на 1	I--	i = i – 1
+= n	збільшення змінної на *n*	i += n	i = i + n
-= n	зменшення змінної на *n*	i -= n	i = i – n
*= n	збільшення змінної в *n* разів	i *= n	i = i * n
/= n	зменшення змінної в *n* разів	i /= n	i = i / n

Приклади розв'язання задач

Приклад 1.1. Вивести на екран повідомлення "Hello, World!"
Розв'язок

```
#include <iostream.h>
void main(){
  cout << ”Hello, World!” << ’\n’;
}
```

Приклад 1.2. Знайти потенціальну енергію тіла масою m, піднятого на висоту h над поверхнею Землі.

Розв'язок. Шукана енергія визначається за формулою $U = mgh$, де $g = 9{,}8$ м/ с^2 – прискорення вільного падіння. Отже, програма має вигляд:

```
#include <iostream.h>
void main(){
  /* Оголошення дійсної сталої */
  const float g = 9.8; // м / с^2
  float m, h, U;
  cin >> m >> h;
  U = m * g * h;
  cout << "Потенціальна енергія =" << U << "Дж\n";
}
```

Якщо у програмі використати форматоване введення/виведення, то програма матиме вигляд:

```
#include <stdio.h>
void main(){
  const float g = 9.8; // м / с^2
  float m, h, U;
  scanf("%f %f", &m, &h);
  // Вводити змінні треба через символ пропуску
  U = m * g * h;
  printf("Потенціальна енергія =%2.2fДж\n", U);
}
```

Приклад 1.3. Обчислити значення многочлена для заданого значення x.

$$y = 5x^5 + 4x^4 + 3x^3 + 3x^2 + 4x + 5, \quad x = -2.$$

Розв'язок. Для оптимізації алгоритму перетворимо многочлен, використовуючи схему Горнера:

$$\begin{aligned} y &= x(5x^4 + 4x^3 + 3x^2 + 3x + 4) + 5 = \\ &= x(x(5x^3 + 4x^2 + 3x + 3) + 4) + 5 = \\ &= x(x(x(5x^2 + 4x + 3) + 3) + 4) + 5 = \\ &= x(x(x(x(5x + 4) + 3) + 3) + 4) + 5. \end{aligned}$$

Тоді програма матиме вигляд:

```
#include <iostream.h>
void main(){
  float x = -2, y;
  y = x*(x*(x*(x*(5*x+4)+3)+3)+4)+5;
  cout << "y(" << x << ") = " << y << '\n';
}
```

Задачі для самостійної роботи

1.1. Вивести на екран рисунки:

```
  а)            б)              в)
     *          * * * * *          *******
   * * *        * *   * *        *         *
 * * * * *      *   *   *      *   Hello   *
   * * *        * *   * *        *         *
     *          * * * * *          *******
```

1.2. Вивести на екран текст:

```
  а)            б)
  a   a   a     a-------a
    a   a       |   a   |
  a   a   a     a-------a
```

де a – введена з клавіатури цифра.

1.3. Вивести на екран таблицю

```
 x | 1 | 2 | 3 | 4 | 5
---+---+---+---+---+---
 y | 3 | 1 | 5 | 4 | 2
```

1.4. Зобразити на екрані декартову систему координат у вигляді

```
          ^ y
          |       x
    -------+-+----->
          | 1
          |
```

Зауваження. У цьому розділі під час написання програм вважати, що дані з клавіатури користувачем вводяться коректно.

1.5. Обчислити гіпотенузу c прямокутного трикутника за катетами a та b.

1.6. Обчислити площу трикутника S за трьома сторонами a, b, c.

1.7. Знайти довжини всіх медіан, бісектрис і висот трикутника, якщо відомі три сторони a, b, c.

1.8. Обчислити відстань від точки (x_0, y_0) до:

а) заданої точки (x, y);

б) заданої прямої $ax + by + c = 0$;

в) точки перетину прямих $x + by + c = 0$ і $ax + y + c = 0$, де $ab \neq 1$.

1.9. Знайти об'єм циліндра, якщо відомо його радіус основи та висоту.

1.10. Знайти об'єм конуса, якщо відомо його радіус основи та висоту.

1.11. Знайти об'єм тора з внутрішнім радіусом r і зовнішнім радіусом R.

1.12. Вважаючи, що Земля має форму сфери радіуса $R = 6350$ км, знайти відстань до лінії горизонту від точки із заданою висотою h над Землею.

1.13. Без попередніх алгебраїчних перетворень обчислити значення арифметичних виразів:

а) $y = \dfrac{c + \dfrac{a}{a^2 + b^2}}{a + \dfrac{b}{b^2 + c^2}}$; б) $y = 1 + \dfrac{1}{2 + \dfrac{1}{3 + \dfrac{1}{4}}}$.

1.14. Наближено визначити значення числа π, використовуючи ланцюговий дріб

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292}}}}.$$

1.15. Наближено визначити період обертання Землі навколо Сонця, використовуючи ланцюговий дріб

$$T = 365 + \cfrac{1}{4 + \cfrac{1}{7 + \cfrac{1}{1 + \cfrac{1}{3}}}}.$$

1.16. Скласти програми для розв'язання рівнянь:

а) $4.2343x + b = c$; б) $(5.23x + b)(d - 2.4y) = 0$;

в) $ax + b = cx + d$, $a - c \neq 0$,

де коефіцієнти a, b, c, d – вводяться з клавіатури.

1.17. Розв'язати квадратне рівняння $ax^2 + bx + c = 0$, де коефіцієнти a, b і c такі, що $b^2 - 4ac > 0$.

1.18. Дано дійсне число x. Користуючись тільки операцією множення, отримати:

а) x^4 за дві операції; б) x^6 за три операції;

в) x^9 за чотири операції; г) x^{15} за п'ять операцій;

ґ) x^{28} за шість операцій; д) x^{64} за шість операцій.

1.19. За найменшу кількість арифметичних операцій, обчислити значення многочлена для введеного з клавіатури значення x:

а) $y = x^4 + 2x^2 + 1$; б) $y = x^4 + x^3 + x^2 + x + 1$;

в) $y = x^5 + 5x^4 + 10x^3 + 10x^2 + 5x + 1$; г) $y = x^9 + x^3 + 1$;

ґ) $y = 16x^4 + 8x^3 + 4x^2 + 2x + 1$; д) $y = x^5 + x^3 + x$.

1.20. Скласти програму для обчислення значення многочлена від двох змінних для введеної з клавіатури пари чисел (x, y):

а) $f(x, y) = x^3 + 3x^2y + 3xy^2 + y^3$;

б) $f(x, y) = x^2y^2 + x^3y^3 + x^4y^4$;

в) $f(x, y) = x + y + x^2 + y^2 + x^3 + y^3 + x^4 + y^4$.

1.21. Скласти програму взаємного обміну значень цілих змінних x та y:

а) з використанням додаткової змінної;

б) без використання додаткової змінної.

1.22. Дано натуральне тризначне число. Знайти:

а) число одиниць, десятків і сотень цього числа;

б) суму цифр цього числа;

в) число, утворене при прочитанні заданого числа справа наліво.

1.23. Дано натуральне тризначне число, у якому всі цифри різні. Знайти всі числа, утворені при перестановці цифр заданого числа.

1.24. Від тризначного числа x відняли його останню цифру. Після ділення результату на 10 до частки ліворуч дописали останню цифру числа x та отримали число n. За заданим числом n знайти вихідне число x. Вважати, що $10 < n < 999$, а число десятків у n не дорівнює нулю.

1.25. У тризначному числі x закреслено першу цифру. Якщо отримане число помножити на 10 і добуток додати до першої цифри числа x, то буде отримано число n. За заданим числом n, $1 < n < 999$ знайти число x.

1.26. Тіло починає рухатися без початкової швидкості з прискоренням a. Обчислити:

а) відстань, яку воно пройде за час t від початку руху;

б) час, за який тіло досягне швидкості v.

1.27. Обчислити кінетичну енергію тіла масою m, що рухається зі швидкістю v відносно поверхні Землі.

Розділ 2
РОЗГАЛУЖЕНІ ПРОГРАМИ

2.1. Основи алгебри висловлювань

Розглянемо двоелементну множину $B = \{Хибність, Істина\}$ (скорочено $\{Хиб, Icm\}$). Вважатимемо, що кожне зі значень із множини B є запереченням іншого. Елементи множини B називатимемо **висловлюваннями**. Над елементами цієї множини визначимо такі операції:

$\vee$ – **диз'юнкція** (логічне "або") – бінарна операція, що діє за правилом: висловлювання $x \vee y$ хибне тоді й тільки тоді, коли хибні одночасно висловлювання x та y, та істинне в інших випадках;

$\&$ – **кон'юнкція** (логічне "і") – бінарна операція, що діє за правилом: висловлювання $x \,\&\, y$ істинне тоді й тільки тоді, коли істинні одночасно висловлювання x та y, і хибне – в інших випадках;

$\neg$ – **заперечення** (логічне "не") – унарна операція, що діє за правилом: висловлювання $\neg x$ є протилежним за змістом до висловлювання x.

Означення 2.1. Множина $B = \{Хиб, Ict\}$ з операціями диз'юнкція, кон'юнкція та заперечення називається **булевою алгеброю** або **алгеброю висловлювань.**

Означення 2.2. Логічним (**булевим**) **виразом** або **висловлюванням** називається твердження, якому завжди можна поставити у відповідність тільки одне з двох логічних значень: *Хибність* або *Істина*.

Поруч із трьома основними булевими операціями розглядають похідні від них логічні операції, серед яких імплікація, тотожність та альтернатива.

$\supset$ – **імплікація** (логічне "наслідування") – бінарна операція, що діє за правилом:

$$x \supset y = \begin{cases} y, & x = Ict; \\ Ict, x = Хиб. \end{cases}$$

$\equiv$ – **тотожність** – бінарна операція, що діє за правилом: висловлювання $x \equiv y$ істинне тоді й тільки тоді, коли висловлювання x та y одночасно істинні або одночасно хибні.

if – **альтернатива** – булева операція, що діє за правилом:

$$if(x, y, z) = \begin{cases} y, & x = Ict; \\ z, & x = Хиб. \end{cases}$$

Булеві вирази узагальнюють на впорядковані (напр., числові) множини, використовуючи поняття **відношення**. Відношення ставить у відповідність одній або кільком (зазвичай двом) величинам булеве значення. Існує стандартний набір відношень для елементів впорядкованих множин:

$=$ – **дорівнює** – бінарна операція, що діє за правилом: висловлювання $a = b$ істинне тоді й тільки тоді, коли значення виразів a та b; є однаковими;

$\neq$ – **не дорівнює** – бінарна операція, що діє за правилом: висловлювання $a \neq b$ істинне тоді й тільки тоді, коли значення виразів a та b є різними;

$<$ – **менше** – бінарна операція, що діє за правилом: висловлювання $a < b$ істинне тоді й тільки тоді, коли значення виразу a є меншим за значення виразу b;

$>$ – **більше** – бінарна операція, що діє за правилом: висловлювання $a > b$ істинне тоді й тільки тоді, коли значення виразу a є більшим за значення виразу b;

$\leq$ – **менше або дорівнює** – бінарна операція, що діє за правилом: висловлювання $a \leq b$ істинне тоді й тільки тоді, коли значення виразу a є меншим або дорівнює виразу b;

$\geq$ – **більше або дорівнює** – бінарна операція, що діє за правилом: висловлювання $a \geq b$ істинне тоді й тільки тоді, коли значення виразу a є більшим або дорівнює виразу b.

Означення 2.3. Булевий вираз, утворений за допомогою відношень і логічних операцій, називається **умовою**.

Зауваження. На відміну від булевих виразів, результат яких завжди є або істинним, або хибним, умова може бути невизначеною. Наприклад, умова $1/x > y$ невизначена при $x = 0$. Тому, розглядаючи умови, необхідно враховувати три випадки: істинність, хибність і невизначеність.

Умовним називається вираз

$$IF(F, a, b) = \begin{cases} a, & F = Icm; \\ b, & F = Xиб, \end{cases}$$

де F – умова, a, b – арифметико-логічні вирази.

Зауваження. Умовні вирази дозволяють компактно записувати деякі функції. Наприклад, функцію, що обчислює модуль дійсного числа можна описати так

$$abs(x) = IF(x \geq 0, x, -x).$$

Приклади розв'язання задач

Приклад 2.1. Довести властивість булевих операцій

$$(p \vee q) \& r \equiv (p \& r) \vee (q \& r).$$

Розв'язок. Оскільки множина B складається лише з двох елементів, то подібні тотожності часто доводять перебором. Згідно з основним правилом комбінаторики, матимемо 2^3 різних комбінацій (p, q, r). Заповнимо таблицю.

p	q	r	$(p \vee q) \& r$	$(p \& r) \vee (q \& r)$
* Icm*	* Icm*	* Icm*	* Icm*	* Icm*
Хиб	* Icm*	* Icm*	* Icm*	* Icm*
* Icm*	*Хиб*	* Icm*	* Icm*	* Icm*
Хиб	*Хиб*	* Icm*	*Хиб*	*Хиб*
* Icm*	* Icm*	*Хиб*	*Хиб*	*Хиб*
Хиб	* Icm*	*Хиб*	*Хиб*	*Хиб*
* Icm*	*Хиб*	*Хиб*	*Хиб*	*Хиб*
Хиб	*Хиб*	*Хиб*	*Хиб*	*Хиб*

Як бачимо, останні два стовпчики таблиці є еквівалентними. Це й доводить нашу властивість.

Приклад 2.2. Довести властивість булевих операцій

$$\neg p \supset \neg q \equiv q \supset p.$$

Розв'язок. Для доведення цієї властивості скористаємося властивістю імплікації $p \supset q \equiv \neg p \vee q$. Таким чином, використовуючи комутативність операції диз'юнкції, маємо:

$$\neg p \supset \neg q \equiv \neg(\neg p) \vee \neg q \equiv \neg q \vee p \equiv q \supset p.$$

Приклад 2.3. Спростити булевий вираз $(x > 0 \vee x \le 0)$.

Розв'язок. Користуючись властивостями булевих операцій та операцій відношень, отримаємо

$$(x > 0 \vee x \le 0) \equiv (x > 0) \vee (x < 0) \vee (x = 0) \equiv (x \ne 0) \vee (x = 0) \equiv Icm.$$

Приклад 2.4. Довести властивість умовного виразу $IF(F, a, a) \subset a$.

Розв'язок. Доведення проведемо перебором усіх варіантів

F	$IF(F, a, a)$	a
Хиб	a	a
* Icm*	a	a
Невизначеність	*Невизначеність*	a

Отже, як видно з таблиці, вираз $IF(F, a, a)$ є звуженням виразу a, що й треба було довести.

Задачі для самостійної роботи

2.1. Обчислити значення булевих виразів:

а) $\neg Icm \vee Хиб$; б) $\neg(Icm \,\&\, Хиб)$;

в) $(Icm \vee Хиб) \vee Icm$; г) $(Icm \,\&\, Хиб) \vee Icm$;

ґ) $\neg(Хиб \,\&\, Хиб) \,\&\, \neg Хиб$; д) $(\neg Хиб \vee Хиб) \,\&\, Icm$.

2.2. Обчислити значення булевих виразів при заданих значеннях величин:

а) $p \vee (\neg p \,\&\, q)$, $p = Хиб, q = Icm$;

б) $p \,\&\, (p \vee q)$, $p = Icm, q = Хиб$;

в) $(p \vee q) \,\&\, (\neg p \vee q)$, $p = Хиб, q = Хиб$;

г) $p \,\&\, q \vee (p \vee q) \vee p \,\&\, r$, $p = Хиб, q = Хиб, r = Хиб$;

ґ) $(p \vee q) \,\&\, (r \vee \neg t)$, $p = Icm, q = Хиб, r = Хиб, t = Icm$.

2.3. Довести властивості булевих операцій:

а) $p \vee q \equiv q \vee p$; б) $p \,\&\, q \equiv q \,\&\, p$;

в) $p \vee p \equiv p$; г) $p \,\&\, p \equiv p$;

ґ) $p \vee \neg p \equiv Icm$; д) $p \,\&\, \neg p \equiv Хиб$;

е) $p \vee Icm \equiv Icm$; є) $p \,\&\, Хиб \equiv Хиб$;

ж) $p \vee Хиб \equiv p$; з) $p \,\&\, Icm \equiv p$;

и) $\neg(\neg p) \equiv p$.

2.4. Довести властивості булевих операцій:

а) $(p \vee q) \vee r \equiv p \vee (q \vee r)$;

б) $(p \,\&\, q) \,\&\, r \equiv p \,\&\, (q \,\&\, r)$;

в) $\neg(p \vee q) \equiv \neg p \,\&\, \neg q$;

г) $\neg(p \,\&\, q) \equiv \neg p \vee \neg q$;

ґ) $p \vee (q \,\&\, r) \equiv (p \vee q) \,\&\, (p \vee r)$;

д) $(p \vee q) \,\&\, r \equiv (p \,\&\, r) \vee (q \,\&\, r)$.

2.5. Використовуючи наведені у попередніх задачах властивості булевих операцій, довести такі властивості:

а) $p \vee (p \,\&\, q) \equiv p$; б) $p \,\&\, (p \vee q) \equiv p$;

в) $p \vee (\neg p \,\&\, q) \equiv p \vee q$; г) $p \,\&\, (\neg p \vee q) \equiv p \,\&\, q$;

ґ) $(p \vee q) \,\&\, (\neg p \vee q) \equiv q$; д) $(p \,\&\, q) \vee (\neg p \,\&\, q) \equiv q$.

2.6. Довести властивості імплікації:

а) $p \supset q \equiv \neg p \vee q$; б) $p \supset (p \vee q) \equiv Icm$;

в) $\neg p \supset \neg q \equiv q \supset p$; г) $(p \,\&\, q) \supset r \equiv p \supset (q \supset r)$;

ґ) $p \supset (q \vee r) \equiv (p \supset q) \vee r)$.

2.7. Спростити висловлювання:

а) $(r \supset p \,\&\, q) \supset (r \supset p) \,\&\, (r \supset q)$;

б) $(p \supset (p \supset (p \supset (p \supset (p \supset q)))))$; в) $(p \supset q) \supset p \,\&\, q$.

2.8. Довести властивості альтернативи:

а) $if\,(Icm, q, r) \equiv q$; б) $if\,(Хиб, q, r) \equiv r$;

в) $if\,(p, q, q) \equiv q$; г) $if\,(\neg p, q, r) \equiv if\,(p, r, q)$;

ґ) $if\,(p_1 \,\&\, p_2, q, r) \equiv if\,(p_1, if\,(p_2, q, r), r)$.

2.9. Вважаючи доведеними властивості г) і ґ) попередньої задачі довести властивості альтернативи:

а) $if\,(p_1 \vee p_2, q, r) \equiv if\,(p_1, q, if\,(p_2, q, r))$;

б) $if\,(p_1 \supset p_2, q, r) \equiv if\,(p_1, if\,(p_2, q, r), q)$.

2.10. Обчислити значення умов при заданих значеннях параметрів:

а) $x^2 + y^2 \le 1 \,\&\, x^2 - y^2 \le 0$, при $x = 0.4, y = 0.5$;

б) $x^2 < 4 \vee y^3 > 8$, при $x = y = 2$;

в) $\neg(|x| + |y| \le 1) \vee \neg(x^2 + y^2 \ge 2)$, при $x = -1, y = 2$;

г) $x > 0 \,\&\, xy < 1 \vee x < 0 \,\&\, y < 4x^2$, при $x = -2, y = 1$.

2.11. Довести властивості умов:

а) $x \le y \equiv \neg(x > y) \equiv x < y \vee x = y$;

б) $x \ge y \equiv \neg(x < y) \equiv x > y \vee x = y$.

2.12. Нехай (x, y) – декартові координати точки на площині. Спростити умови:

а) $yx > 0 \,\&\, x \ge 0 \,\&\, y \ge 0$; б) $yx \ge 0 \,\&\, x \ge 0 \,\&\, y \le 0$;

в) $\neg(|x| + |y| > 1 \vee x^2 + y^2 < 2)$; г) $|x| + |y| < 1 \,\&\, x^2 + y^2 > 2$;

ґ) $xy = 0 \,\&\, x / y \ge 0$; д) $x + y = 2 \,\&\, x - y = 1$;

е) $x \ge 0 \,\&\, x \le 0$; є) $x \ge 0 \vee x \le 0$.

2.13. Довести властивості умовних виразів:

а) $IF(Icm, a, b) \equiv a$; б) $IF(Хиб, a, b) \equiv b$;

в) $IF(F, a, a) \subset a$; г) $IF(\neg F, a, b) \equiv IF(F, b, a)$;

ґ) $IF(F_1 \,\&\, F_2, a, b) \subset IF(F_1, IF(F_2, a, b), b)$.

2.14. Вважаючи доведеними властивості г) і ґ) попередньої задачі, довести властивості умовних виразів:

а) $IF(F_1 \vee F_2, a, b) \subset IF(F_1, a, IF(F_2, a, b))$;

б) $IF(F_1 \supset F_2, a, b) \subset IF(F_1, IF(F_2, a, b), a)$.

2.15. Використовуючи умовні вирази описати такі функції:

а) $sign(x)$ – визначення знаку дійсного числа x;

б) $max(x, y)$ – функція визначення більшого серед двох чисел x та y;

в) $min(x, y, z)$ – функція визначення найменшого серед чисел x, y і z.

2.2. Умови та булеве присвоювання

Мова програмування С має механізми для роботи з алгеброю висловлювань та умовами в цілому. Проте для булевих констант *Хиб* та *Ісm* не передбачено спеціальних імен. Не передбачено також і булевий тип даних. Натомість робота із алгеброю висловлювань моделюється за допомогою цілих чисел. Значення будь-якого (навіть арифметичного) виразу вважається істинним, якщо воно не дорівнює 0. Найчастіше у ролі *Істини* у С використовується значення 1.

Зауваження. Надалі, щоб уникнути плутанини, умови також називатимемо булевими виразами і вважатимемо, що вони завжди визначені.

Для запису умов у С використовуються:

- операції відношення (==, !=, <, >, >=, <=)
  - == – дорівнює;
  - != – не дорівнює;
  - < – менше;
  - > – більше;
  - <= – менше або дорівнює;
  - >= – більше або дорівнює;
- булеві операції (||, &&, !)
  - || – диз'юнкція;
  - && – кон'юнкція;
  - ! – заперечення.

Умовний вираз у мові С записується так

```
F ? a : b
```

де F – умова. Результатом умовного виразу буде значення виразу a, якщо умова F істинна, і b – якщо умова F хибна.

Приклади розв'язання задач

Приклад 2.5. Точка площини задана декартовими координатами (x, y). Перевірити чи належить вона третьому координатному квадранту.

Розв'язок. Результатом виконання алгоритму слід вважати значення деякого булевого виразу. При цьому вважатимемо, що значення 1 (*Ісm*) інтерпретується як "так, точка належить третьому квадранту", а значення 0 (*Хиб*) – як відповідь "ні, точка не належить третьому квадранту". Розглянемо булевий вираз $(x < 0 \,\&\, y < 0)$. Очевидно, що він буде істинним тоді й тільки тоді, коли точка (x, y) належить третьому координатному квадранту. Тоді програма матиме вигляд:

```
#include <iostream.h>
void main(){
  float x, y;
  int z;
  cin >> x >> y;
  z = (x < 0 && y < 0);
  // Програма виводить на екран число 1, якщо
  // умова істинна і 0 у протилежному випадку
  cout << z;
}
```

У програмі можна скористатися умовним виразом. Для цього рядок

```
  z = (x < 0 && y < 0);
```

потрібно замінити таким

```
  z = (x < 0 && y < 0) ? 1 : 0;
```

Приклад 2.6. Скласти програму перевірки можливості існування трикутника із заданими сторонами a,b,c.

Розв'язок. Аналогічно до попереднього прикладу, результатом виконання програми вважатимемо деякий булевий вираз. Причому значення 1 (*Істина*) інтерпретується як відповідь "так, трикутник із такими сторонами існує", а значення 0 (*Хиба*) – як відповідь "ні, трикутника з такими сторонами не існує". Якщо згадати, що в будь-якому трикутнику кожна сторона менша від суми двох інших сторін, можливість існування трикутника відповідає істинності виразу

$$(a + b > c)\ \&\ (a + c > b)\ \&\ (b + c > a).$$

Отримаємо програму:

```
#include <iostream.h>
void main(){
  float a, b, c;
  int z;
  cin >> a >> b >> c;
  z = (a + b > c) && (a + c > b) && (b + c > a);
  cout << z;
}
```

Задачі для самостійної роботи

Зауваження. У кожній задачі поточного пункту потрібно записати необхідну для розв'язання задачі умову в зручному аналітичному вигляді, а потім – створити програму, результатом виконання якої буде виведення на екран числа 1, якщо умова істинна, і 0 – у протилежному випадку.

2.16. Перевірити впорядкованість змінних a,b,c:

а) за зростанням їхніх значень;

б) за спаданням їхніх значень;

в) за зростанням чи спаданням їхніх значень.

2.17. Записати умови, що істинні тоді й тільки тоді, коли:

а) натуральне число n – парне;

б) остання цифра числа $n - 0$;

в) ціле число n кратне натуральному числу m;

г) натуральні числа n і k одночасно кратні натуральному числу m;

ґ) сума першої і другої цифри двозначного натурального числа n – двозначне число;

д) число x більше за число y не менше, ніж на 6;

е) принаймні одне з чисел x, y або z більше за 100;

є) тільки одне з чисел x, y або z менше за 1000.

Зауваження. У наступних задачах вважати, що межа заданої області самій області не належить.

2.18. Точка площини задана декартовими координатами (x, y). Перевірити, чи належить вона координатному квадранту:

а) першому; б) другому; в) третьому; г) четвертому.

2.19. Точка площини задана декартовими координатами (x, y). Перевірити, чи належить вона:

а) кільцю, з центром у початку координат, внутрішнім радіусом 1 і зовнішнім 2;

б) квадрату, діагоналі якого перетинаються у початку координат, а одна вершина розташована у точці $(3, 4)$;

в) внутрішності еліпса, фокуси якого розташовані на дійсній осі, велика піввісь еліпса 3, мала піввісь – 2.

2.20. Скласти програму перевірки належності точки (x, y) до зафарбованої області (рис. 2.1.)

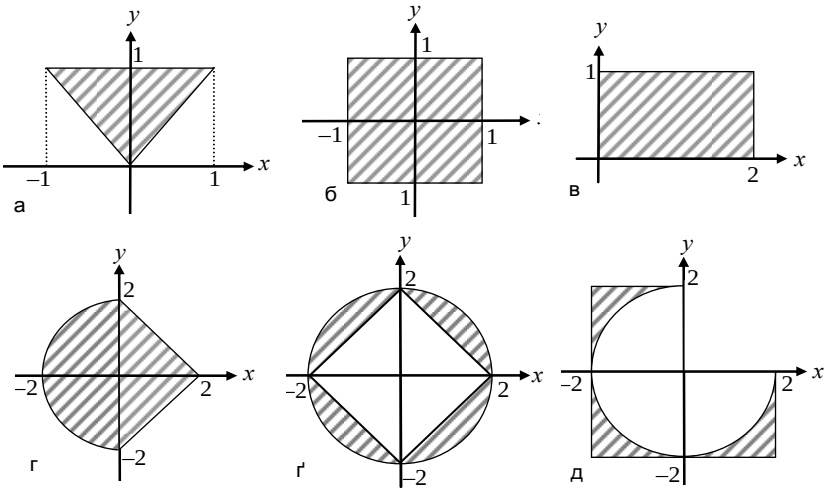


Рис. 2.1

2.21. Створити програму, яка перевіряє, чи належить початок координат трикутнику з вершинами $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$.

2.22. Точка площини задана декартовими координатами (x, y). Перевірити, чи належить вона:

а) трикутнику з вершинами $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$;

б) багатокутнику з вершинами $A(x_1, y_1)$, …, $A_n(x_n, y_n)$.

2.23. Точка простору задана декартовими координатами (x, y, z). Перевірити, чи належить вона кулі з радіусом R і центром у початку координат.

2.24. Точка простору задана декартовими координатами (x, y, z). Перевірити, чи належить вона циліндру, вісь якого збігається з віссю Oz, висота дорівнює h, а нижня основа лежить у площині Oxy та має радіус r.

2.25. Точка простору задана декартовими координатами (x, y, z). Перевірити, чи належить вона зрізаному конусу, вісь якого збігається з віссю Oz, висота дорівнює h, нижня основа лежить у площині Oxy та має радіус R, а верхня основа – радіус r.

2.26. Точка простору задана декартовими координатами (x, y, z). Перевірити, чи належить вона внутрішності тора, що утворюється в результаті обертання круга $(x-(a+r))^2 + z^2 \le r^2$ навколо осі Oz.

2.27. Точка простору задана декартовими координатами (x, y, z). Перевірити, чи належить вона тетраедру з вершинами у точках $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$, $C(x_3, y_3, z_3)$, $D(x_4, y_4, z_4)$.

2.3. Розгалуження

Означення 2.4. Структура керування, що виконує певну дію (перелік дій), залежно від істинності деякої умови F, називається **розгалуженням**.

Для програмування розгалужень у C реалізовано умовний оператор, що має такий синтаксис

```
if (F)
  {P}
else
  {Q}
```

де F – умова, P – інструкція (або набір інструкцій), що виконується у випадку істинності умови F, Q – інструкція (або набір інструкцій), що виконується у випадку хибності умови F. У випадку, якщо P або Q складається тільки з однієї інструкції, то операторні дужки `{}` можна опустити.

Зауваження. Блок розгалуження `else` є необов'язковим, його можна опускати, якщо у випадку хибності умови F програма не має виконувати жодних дій.

Приклади розв'язання задач

Приклад 2.7. Знайти більше з двох значень, введених із клавіатури.
Розв'язок

Спосіб 1. Використаємо умовний вираз. Дійсно, легко бачити, що

$$\max(x,y) = IF(x \geq y, x, y).$$

Тоді програма матиме вигляд:

```
#include <iostream.h>
void main(){
  float x, y, max;
  cin >> x >> y;
  max = (x >= y) ?  x : y;
  cout << max;
}
```

Спосіб 2. За означенням

$$\max(x,y) = \begin{cases} x, & x \geq y, \\ y, & x < y. \end{cases}$$

Тоді програма матиме вигляд:

```
#include <iostream.h>
void main(){
  float x, y, max;
  cin >> x >> y;
  if (x >= y)
    max = x;
  else max = y;
  cout << max;
}
```

Приклад 2.8. Скласти програму для знаходження найбільшого з трьох значень, введених з клавіатури.

Розв'язок. Нехай $\max(\ldots)$ визначає найбільше серед чисел, що стоять у дужках. Тоді легко бачити, що $\max(x,y,z) = \max(\max(x, y), z)$. Тому у програмі спочатку визначимо більше зі значень x та y, а вже потім – найбільше серед трьох чисел. Ураховуючи попередній приклад, отримаємо програму:

```
#include <iostream.h>
void main(){
  float x, y, z, max;
  cin >> x >> y >> x;
  if (x >= y)
    max = x;
  else
    max = y;
```

```
  if (z >= max)
    max = z;
  cout << max;
}
```

Задачі для самостійної роботи

2.28. Визначити більше та менше з двох чисел, введених з клавіатури.
2.29. Дано три дійсних числа. Скласти програму для знаходження числа:
а) найбільшого за модулем; б) найменшого за модулем.
2.30. Дано три дійсних числа x, y і z. Скласти програму для обчислення:
а) $\max(x+y+z, xy-xz+yz, xyz)$; б) $\max(xy, xz, yz)$.
2.31. Дано три дійсних числа x, y і z. Визначити кількість:
а) різних серед них; б) однакових серед них;
в) чисел, що є більшими за їхнє середнє арифметичне значення;
г) чисел, що є більшими за введене з клавіатури число a.
2.32. Обчислити значення функцій:
а) $f(x)=|x|$; б) $f(x)=||x|-1|-1$;
в) $f(x)=\operatorname{sign}(x)$; г) $f(x)=\sin|x|$;
ґ) $f(x)=\begin{cases}-x^2+1, & x\le -1,\\ 0, & |x|\le 1,\\ x^2-1, & x>1;\end{cases}$ д) $f(x)=\begin{cases}0, & x\le 0,\\ x^2, & 0<x\le 1,\\ x^4, & x>1.\end{cases}$
2.33. Обчислити значення функцій (рис. 2.2.).

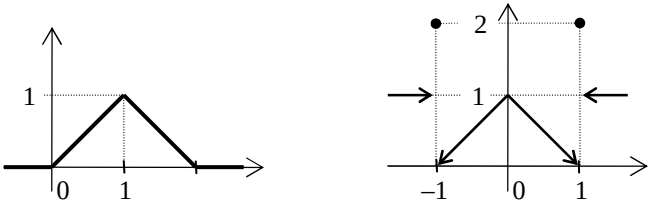


Рис. 2.2

2.34. Обчислити значення виразу:

$$z=\begin{cases}\max(x, y+5), & x>y,\\ \min(x+1, y, 3), & x\le y.\end{cases}$$

2.35. Обчислити значення $x=f(y)-6.3$, де $y=z+2$ та

$$f(y)=\begin{cases} y^2-0.3, & y<0,\\ 0, & 0\le y\le 1,\\ y^2+y, & y>1. \end{cases}$$

2.36. Перевірити, чи існує трикутник із заданими сторонами a,b,c. Якщо так, то визначити, який він:

а) гострокутний; б) прямокутний; в) тупокутний.

2.37. Визначити, скільки розв'язків має рівняння та розв'язати його:

а) $ax^2+bx+c=0$; б) $ax^4+bx^2+c=0$.

2.38. Визначити, скільки розв'язків має система рівнянь і розв'язати її:

а) $\begin{cases} a_1x+b_1y+c_1=0,\\ a_2x+b_2y+c_2=0; \end{cases}$ б) $\begin{cases} |x|+|y|=1,\\ ax+by+c=0. \end{cases}$

2.39. Задано два квадрати, сторони яких паралельні координатним осям. З'ясувати, чи перетинаються вони. Якщо так, то знайти координати лівого нижнього та правого верхнього кутів прямокутника, що є їхнім перетином.

2.40. Дано два прямокутники, сторони яких паралельні координатним осям. Відомо координати лівого нижнього та правого верхнього кутів кожного з прямокутників. Знайти координати лівого нижнього та правого верхнього кутів мінімального прямокутника, що містить задані прямокутники.

2.41. Дано два прямокутники, сторони яких паралельні координатним осям. Відомо координати лівого нижнього кута кожного з прямокутників і довжини їхніх сторін. Знайти координати лівого нижнього та правого верхнього кутів мінімального прямокутника, що містить задані прямокутники.

Розділ 3
ЦИКЛІЧНІ ПРОГРАМИ

Означення 3.1. Структура керування, що здійснює повторення певної послідовності команд, називається **циклом**.

Цикли поділяють на два типи: **цикли з лічильником** і **цикли за умовою.**

3.1. Цикли з лічильником

Як правило, цикли з лічильником використовують для повторення певної послідовності команд наперед задану кількість разів.

Суттєвою відмінністю циклів із лічильником від циклів за умовою є наявність змінної, яка називається **лічильником**.

Синтаксис циклу з лічильником такий:

```
for (P; F; S){
  Q;
}
```

де P – ініціалізація змінної-лічильника, F – умова продовження циклу; S – операція зміни лічильника, що виконується після кожного проходження циклу; Q – тіло циклу.

Правило виконання циклу з лічильником:

1) виконується початкова ініціалізація змінної-лічильника P;

2) перевіряється істинність умови F.

Умова F хибна:

– відбувається перехід на наступний оператор після циклу.

Умова F істинна:

– виконується тіло циклу Q;

– виконується операція зміни лічильника S;

– правило повторюється, починаючи з пункту 2).

Наприклад, для виведення на екран усіх натуральних чисел від 1 до 5 у зворотному порядку можна використати такий цикл:

```
for (i = 5; i > 0; i--){
  cout << i;
}
```

Зауваження. Тіло циклу Q може використовувати змінну-лічильник i як параметр так, як показано на попередньому прикладі. Проте, не ре-

комендується у тілі циклу змінювати лічильник, оскільки це інколи призводить до неочікуваних результатів. Наприклад, цикл

```
for (i = 5; i > 0 ; i--){
  i++;
}
```

є нескінченним.

Приклади розв'язання задач

Приклад 3.1. Скласти програму для обчислення добутку двох натуральних чисел m і n, використовуючи тільки операцію додавання.

Розв'язок. Легко помітити, що

$$m \cdot n = \underbrace{n + ... + n}_{m \text{ разів}}.$$

Таким чином, програма має вигляд

```
#include <iostream.h>
void main(){
  int m, n, i, s = 0;
  cin >> m >> n;
  for (i = 1; i <= m; i++){
     s = s + n;
  }
  cout << s;
}
```

Приклад 3.2. Дано натуральне число n. Написати програму для обчислення значень многочлена

$$x^n + x^{n-1} + ... + x^2 + x + 1$$

при заданому значенні x.

Розв'язок. Розглянемо два способи розв'язання.

Спосіб 1. Позначимо $z_k = x^k, k \geq 0$. Тоді цикл

```
z = 1;
for (i = 1; i <= k; i++){
  z = z * x;
}
```

забезпечить послідовне обчислення у змінній z значень $z_0, z_1, ..., z_n$.

Таким чином, отримаємо програму

```
#include <iostream.h>
void main(){
  float x, y = 1, z = 1;
  int n, i;
  cin >> n >> x;
  for (i = 1; i <= n; i++){
     z = z * x;
     y = y + z;
```

```
  }
  cout << y;
}
```

Спосіб 2. Скористаємось схемою Горнера. Розставимо дужки так:

$$y = x^n + x^{n-1} + ... + x^2 + x + 1 = (...(x + 1)\,x + 1)\,x + ... + 1)\,x + 1.$$

Отримаємо програму

```
#include <iostream.h>
void main(){
  float x, y = 1;
  int n, i;
  cin >> n >> x;
  for (i = 1; i <= n; i++){
     y = y * x + 1;
  }
  cout << y;
}
```

Приклад 3.3. Задано натуральне число n і дійсні числа $a_1, a_2, \ldots a_n$. Знайти $\max(a_1, a_2, \ldots a_n)$.

Розв'язок. Очевидно, що

$$\max(a_1, a_2, ..., a_n) = \max(...(\max(\max(a_1, a_2), a_3), ...), a_n).$$

Нехай у змінній a міститься чергове число послідовності a_n, що вводиться з клавіатури. Оскільки кількість членів послідовності a_n відома, то для їхнього зчитування використаємо цикл з лічильником. Для визначення максимального числа використаємо змінну max. Таким чином, програма матиме вигляд:

```
#include <iostream.h>
void main(){
  int n, k, a, max;
  cout << "n = ";
  cin >> n;
  cout << "a(" << 1 << ")= " ;
  cin >> max;
  for (k = 2; k <=n; k++) {
    cout << "a(" << k << ")= " ;
    cin >> a;
    if (a > max) max = a;
  }
  cout << "max(a(1),...,a(" << n << "))= ";
  cout << max <<'\n';
}
```

Приклад 3.4. Дано натуральне число n і цілі числа $a_1, a_2, \ldots, a_n$. Знайти $\max(a_1, a_1a_2, \ldots, a_1a_2 \ldots a_n)$.

Розвязок. Використаємо змінну a для зчитування чергового члена послідовності. У змінній p обчислюватимемо добуток введених чисел $a_1, a_2,\ldots,$ що передують a, включаючи саме це число. Використаємо змінну max для визначення максимуму.

```
#include <iostream.h>
void main(){
  int n, k, a, p, max;
  cout << "n = ";
  cin >> n;
  cout << "a(" << 1 << ")= " ;
  cin >> max;
  p = max;
  for (k = 2; k <=n; k++) {
    cout << "a(" << k << ")= " ;
    cin >> a;
    p = p * a;
    if (p > max) max = p;
  }
  cout << "max(a(1), ..., a(1)*...*a(";
  cout << n << "))= ";
  cout << max <<'\n';
}
```

Приклад 3.5. Написати програму для обчислення подвійного факторіала натурального числа n: $y = n!!$

Розв'язок. За означенням

$$n!!=\begin{cases}1\cdot 3\cdot 5\cdot\ldots\cdot n, & n\text{ - непарне},\\ 2\cdot 4\cdot 6\cdot\ldots\cdot n, & n\text{ - парне}.\end{cases}$$

В обох випадках маємо як співмножники всі члени спадної арифметичної прогресії із різницею -2, які містяться між n та 1. Звідси програма

```
#include <iostream.h>
void main(){
  int n, k, p;
  cout << "n = "; cin >> n;
  p = 1;
  for (k = n; k >= 1; k = k - 2)
    p = p * k;
  // printf("%i!! = %i\n", n, p);
  cout << n << "!! = " << p << "\n";
}
```

розв'язує задачу. Зауважимо, що за непарного n останнім значенням k буде 1, а за парного – 2.

Приклад 3.6. Задано натуральне число n та дійсні числа y_1, y_2, ... y_n. Визначити $z_1 + z_2 + \ldots + z_n$, де

$$z_i = \begin{cases} y_i, & 0 < y_i < 10, \\ 1, & \text{в інших випадках.} \end{cases}$$

Розв'язок. У змінній S обчислюватимемо шукану суму, а у змінній y – значення чергового введеного числа. Подамо програму для розв'язку задачі в такому вигляді:

```
#include <iostream.h>
void main(){
  int n;
  float y, i, S = 0;
  cout << "n = ";
  cin >> n;
  for (i = 1; i <= n; i++) {
    cin >> y;
    if (y >= 10 || y <= 0) y = 1;
    S = S + y;
  }
  cout << S << "\n";
}
```

Задачі для самостійної роботи

3.1. Вивести на екран такий рядок:

```
n! = 1*2*3*4*5*...*n
```

де n – введене з клавіатури число.

3.2. Вивести на екран таблицю множення на 5:

```
1 x 5 = 5
2 x 5 = 10
...
9 x 5 = 45
```

3.3. Вивести на екран таблицю:

```
 1  |  2  |  3  |...|  n-1  |  n
------------------...--------------
a+1 | a+2 | a+3 |...| a+n-1 | a+n
```

де a, n – вводяться з клавіатури.

3.4. Написати програму обчислення добутку двох натуральних чисел, використовуючи лише операцію додавання.

3.5. Написати програму обчислення факторіала натурального числа.

3.6. Написати програму обчислення натурального степеня n від дійсного числа a.

3.7. Дано натуральне число n. Написати програми обчислення значень виразів:

а) $\sqrt{2+\sqrt{2+\ldots+\sqrt{2}}}$ (n коренів); б) $\sqrt{3+\sqrt{6+\ldots+\sqrt{3(n-1)+\sqrt{3n}}}}$;

в) $\cos\pi+\cos\frac{\pi}{2}+\cos\frac{\pi}{4}+\ldots+\cos\frac{\pi}{2^n}$.

3.8. Дано натуральне число n. Написати програми обчислення значень виразів при заданому значенні x:

а) $1+(x-1)+(x-1)^2+\ldots+(x-1)^n$;

б) $1+\frac{1}{x^2+1}+\frac{1}{(x^2+1)^2}+\ldots+\frac{1}{(x^2+1)^n}$;

в) $1+\sin x+\ldots+\sin^n x$.

3.9. Дано натуральне число n. Написати програму обчислення значень вказаних многочленів при заданому значенні x:

а) $x^n+x^{n-1}+\ldots+x^2+x+1$; б) $nx^n+(n-1)x^{n-1}+\ldots+2x^2+x$;

в) $(nx)^n+((n-1)x)^{n-1}+\ldots+(2x)^2+x$.

3.10. Задано натуральне число n і дійсні числа $a_1, a_2,\ldots, a_n$. Написати програми знаходження:

а) $\min(a_1,a_2,\ldots,a_n)$; б) $\max(|a_1|,\ldots,|a_n|)$;

в) $\max(a_2,a_4,\ldots)$; г) $\min(a_1,a_3,\ldots)$;

ґ) $\min(a_2,a_4,\ldots)+\max(a_1,a_3,\ldots)$; д) $\max(a_1,a_2,a_4,a_8,\ldots)$;

е) $\max(-a_1,a_2,-a_3,\ldots,(-1)^n a_n)$; є) $\max(a_1,2a_2,\ldots,na_n)$;

ж) $(\min(a_1,\ldots,a_n))^2-\min(a_1^2,\ldots,a_n^2)$; з) $\max(a_1+a_2,\ldots,a_{n-1}+a_n)$.

3.11. Дано натуральне число n, цілі числа $a_1, a_2,\ldots, a_n$. Написати програми знаходження кількості:

а) парних серед $a_1, a_2,\ldots, a_n$;

б) повних квадратів серед $a_1, a_2,\ldots, a_n$;

в) квадратів непарних чисел серед $a_1, a_2,\ldots, a_n$.

3.12. Дано натуральне число n. Написати програму обчислення факторіала $y = n!$, використовуючи цикл із лічильником:

а) суцільний зростаючий; б) суцільний спадний.

3.13. Написати програму обчислення подвійного факторіала натурального числа n: $y = n!!$

3.14. Написати програми обчислення факторіалів:

а) $y=(2n)!!$ б) $y=(2n+1)!!$ в) $y=n!n!!(n+1)!!$

3.15. Написати програму друкування таблиці значень функції $y = \sin(x)$ на відрізку $[0; 1]$ із кроком $h = 0{,}1$.

3.16. Написати програму визначення кількості тризначних натуральних чисел, сума цифр яких дорівнює n ($n \geq 1$). У програмі не використовувати операцію ділення!

3.17. Дано n цілих чисел. Створити програму, що визначає, яка кількість з них більша за своїх "сусідів", тобто за попереднє та наступне числа.

3.18. Задано натуральне число n, дійсні числа y_1, y_2, ... y_n. Скласти програму визначення:

а) $\max(|z_1|, .., |z_n|)$, де $z_i = \begin{cases} y_i, & |y_i| \leq 2, \\ 0.5, & \text{в інших випадках;} \end{cases}$

б) $\min(|z_1|, .., |z_n|)$, де $z_i = \begin{cases} y_i, & |y_i| \geq 1, \\ 2, & \text{в інших випадках;} \end{cases}$

в) $z_1 + z_2 + ... + z_n$, де $z_i = \begin{cases} y_i, & 0 < y_i < 10, \\ 1, & \text{в інших випадках.} \end{cases}$

3.2. Програмування рекурентних співвідношень

Означення 3.2. Кажуть, що послідовність $\{x_k : k \geq 0\}$ задано **рекурентним співвідношенням першого порядку**, якщо

$$\begin{cases} x_0 = a, \\ x_k = f(k, p, x_{k-1}), \ k \geq 1, \end{cases} \tag{3.1}$$

де a – числова стала, p – деякий сталий параметр або набір параметрів, f – функція, задана аналітично у вигляді арифметичного виразу, що складається з операцій, доступних для виконання компілятором С.

Приклад. Послідовність $\{x_k = k! : k \geq 0\}$ можна задати рекурентним співвідношенням першого порядку

$$\begin{cases} x_0 = 1, \\ x_k = k x_{k-1}, \ k \geq 1. \end{cases}$$

Означення 3.3. Кажуть, що послідовність $\{x_k : k \geq 0\}$ задано **рекурентним співвідношенням *m*-го порядку**, якщо

$$\begin{cases} x_0 = a,\ x_1 = b,\ ...,\ x_{m-1} = c, \\ x_k = f(k, p, x_{k-1}, ..., x_{k-m}),\ k \geq m, \end{cases} \tag{3.2}$$

де $a, b, ..., c$ – числові сталі, p – деякий сталий параметр або набір параметрів, f – функція, задана аналітично.

Приклад. Послідовність $\{F_k : k \geq 0\}$ чисел Фібоначчі зазвичай задають рекурентним співвідношенням другого порядку

$$\begin{cases} F_0 = F_1 = 1, \\ F_k = F_{k-1} + F_{k-2},\ k \geq 2. \end{cases}$$

Зауваження. Попередні два означення легко узагальнити на системи рекурентних співвідношень, якщо вважати, що кожен член послідовності $\{x_k\}$ є векторною величиною.

Для обчислення членів послідовностей, заданих рекурентними співвідношеннями, використовують цикли.

Нехай послідовність $\{x_k : k \geq 0\}$ задано рекурентним співвідношенням (3.1). Тоді після виконання фрагмента програми

```
x = a;
for (k = 1; k <= n; k++){
   x = f(k, p, x);
}
```

у змінній x буде міститися значення елемента x_n послідовності.

Зауважимо, що нумерація членів послідовності інколи починається не з 0, а з деякого натурального числа m: $\{x_k : k \geq m\}$. Припустимо, що рекурентне співвідношення для цієї послідовності має вигляд

$$\begin{cases} x_m = a, \\ x_k = f(k, p, x_{k-1}),\ k \geq m+1. \end{cases}$$

Тоді для отримання x_n необхідно замінити наведений вище фрагмент програми на такий

```
x = a;
for (k = m + 1; k <= n; k++)
   x = f(k, p, x);
}
```

Для обчислення елементів послідовності, заданої рекурентним співвідношенням вищого порядку, застосовують трохи інший підхід. Наприклад, нехай послідовність $\{x_k : k \geq 0\}$ задано рекурентним співвідношенням третього порядку

$$\begin{cases} x_0 = a,\ x_1 = b,\ x_2 = c, \\ x_k = f(k, p, x_{k-1}, x_{k-2}, x_{k-3}),\ k \geq 3, \end{cases}$$

Тоді після виконання фрагмента програми

```
u = a; v = b; w = c;
for (k = 3; k <= n; k++){
   x = f(k, p, w, v, u);
   u = v;
   v = w
   w = x;
}
```

у змінних x і w буде міститися x_n, у змінній v – x_{n-1}, у змінній u – x_{n-2}.

Запропонований підхід можна узагальнити для рекурентних співвідношень довільного порядку.

Приклади розв'язання задач

Приклад 3.7. Скласти програму обчислення елементів послідовності

$$x_k = \frac{x^k}{k!} \quad (k \geq 0). \tag{3.3}$$

Розв'язок. Складемо рекурентне співвідношення для заданої послідовності. Легко бачити, що кожен член послідовності x_k є добутком чисел. Ураховуючи це, обчислимо частку двох сусідніх членів послідовності. Таким чином, для $k \geq 1$ отримаємо співвідношення

$$\frac{x_k}{x_{k-1}} = \frac{x^k}{k!} \cdot \frac{(k-1)!}{x^{k-1}} = \frac{x}{k},$$

з якого випливає рівність

$$x_k = x_{k-1} \frac{x}{k}, \; k \geq 1.$$

Підставляючи до (3.3) $k = 0$, обчислимо $x_0 = 1$. Отже, рекурентне співвідношення має вигляд

$$\begin{cases} x_0 = 1, \\ x_k = x_{k-1} \dfrac{x}{k}, \; k \geq 1. \end{cases}$$

Скористаємося наведеним вище алгоритмом для обчислення відповідного члена послідовності заданої рекурентним співвідношенням першого порядку. Тоді програма матиме вигляд:

```
#include <iostream.h>
void main(){
  int n, k;
  float x, xn;
  cin >> x >> n;
  xn = 1;
  for (k = 1; k <= n; k++)
```

```
      xn = xn * x / k;
  cout << xn << '\n';
}
```

Приклад 3.8. Скласти програму обчислення суми:

$$S_k = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}. \qquad (3.4)$$

Розв'язок. Помітимо, що (3.4) має сенс тільки для $k \geq 1$. Складемо рекурентне співвідношення. Видно, що, на відміну від попереднього прикладу, кожен член послідовності S_k є сумою елементів вигляду $1/i$, де i змінюється від 1 до k. Отже, для побудови рекурентного співвідношення знайдемо різницю двох сусідніх членів послідовності S_k. Для $k \geq 2$:

$$S_k - S_{k-1} = 1/k.$$

Підставляючи до (3.4) $k = 1$, отримаємо $S_1 = 1$. Отже, рекурентне співвідношення має вигляд:

$$\begin{cases} S_1 = 1, \\ S_k = S_{k-1} + 1/k, \; k \geq 2. \end{cases}$$

Аналогічно до попереднього прикладу, враховуючи, що нумерація членів послідовності починається з 1, а не з 0, отримаємо програму.

```
#include <iostream.h>
void main(){
  int n, k;
  float S;
  cin >> n;
  S = 1;
  for (k = 2; k <= n; k++){
     S = S + 1 / float(k);
  }
  cout << S <<'\n';
}
```

Зауваження. Як вже відомо, ділення цілих чисел у С реалізовано як ділення націло, що необхідно враховувати при написанні програм. Наприклад, після виконання коду

```
float S = 1;
for (k = 2; k <= 5; k++)
   S = S + 1 / k;
```

у змінній S буде міститися число 1, оскільки 1 націло поділити на будь-яке ціле $k > 1$ завжди дає 0. Тому під час написання програми було використано операцію `float(k)`, що перетворює значення цілої змінної k до типу `float`.

Приклад 3.9. Скласти програму обчислення суми

$$S_n = \sum_{i=1}^{n} 2^{n-i} i^2. \qquad (3.5)$$

Розвязок. Складемо рекурентне співвідношення. Підставляючи до (3.5) $n = 1$, отримаємо $S_1 = 1$. Щоб отримати вираз для загального члена, розкриємо суму (3.5):

$$S_n = 2^n \left(\frac{1^2}{2^1} + \frac{2^2}{2^2} + \ldots + \frac{(n-1)^2}{2^{n-1}} + \frac{n^2}{2^n} \right) =$$

$$= 2 \cdot 2^{n-1} \left(\frac{1^2}{2^1} + \frac{2^2}{2^2} + \ldots + \frac{(n-1)^2}{2^{n-1}} + \frac{n^2}{2^n} \right) =$$

$$= 2 \cdot 2^{n-1} \left(\frac{1^2}{2^1} + \frac{2^2}{2^2} + \ldots + \frac{(n-1)^2}{2^{n-1}} \right) + 2 \cdot 2^{n-1} \frac{n^2}{2^n} = 2 \cdot S_{n-1} + n^2 .$$

Отже, рекурентне співвідношення для (3.5) матиме вигляд:

$$\begin{cases} S_1 = 1, \\ S_k = 2 \cdot S_{k-1} + k^2, \ k \geq 2. \end{cases}$$

Напишемо програму для визначення S_n.

```
#include <iostream.h>
void main(){
  int n, k, S = 1;
  cin >> n;
  for (k = 2; k <= n; k++)
     S = 2*S + k*k;
  cout << S <<'\n';
}
```

Приклад 3.10. Скласти програму для обчислення суми

$$S_k = \sum_{i=0}^{k} a^i b^{k-i}$$

Розв'язок. Рекурентне співвідношення можна побудувати двома способами.

Спосіб 1. Очевидно, що $S_1 = 1$. Розкриваючи суму та групуючи доданки, аналогічно до попереднього прикладу, дістанемо

$$\begin{cases} S_0 = 1, \\ S_k = b \cdot S_{k-1} + a^k, \ k \geq 1. \end{cases}$$

Введемо позначення $x_k = a^k$. Запишемо для послідовності $\{x_k : k \geq 0\}$ рекурентне співвідношення:

$$\begin{cases} x_0 = 1, \\ x_k = a \cdot x_{k-1}, \qquad k \geq 1. \end{cases}$$

Таким чином, отримаємо систему рекурентних співвідношень

$$\begin{cases} S_1 = x_1 = 1, \\ x_k = a \cdot x_{k-1}, \qquad k \geq 1. \\ S_k = b \cdot S_{k-1} + x_k, \end{cases}$$

Спосіб 2. Легко побачити, що послідовність S_k можна представити у вигляді

$$S_k = \frac{a^{k+1} - b^{k+1}}{a - b}.$$

Тоді система рекурентних співвідношень матиме вигляд

$$\begin{cases} x_1 = a, y_1 = b, \\ x_k = a \cdot x_{k-1}, \\ y_k = b \cdot y_{k-1}, \qquad k \geq 1. \\ S_k = \dfrac{x_k - y_k}{a - b}, \end{cases}$$

Отже, програма для знаходження S_n, заданого рекурентним співвідношенням, що отримано першим способом, має вигляд:

```
#include <iostream.h>
void main(){
  int n, k;
  float a, b, x, S;
  cin >> n >> a >> b;
  S = x = 1;
  for (k = 1; k <= n; k++){
     x = a * x;
     S = b * S + x;
  }
  cout << S <<'\n';
}
```

Приклад 3.11. Послідовністю чисел Фібоначчі називається послідовність $\{F_k : k \geq 0\}$, задана рекурентним співвідношенням другого порядку

$$\begin{cases} F_0 = F_1 = 1, \\ F_k = F_{k-1} + F_{k-2}, \; k \geq 2. \end{cases}$$

Написати програму для обчислення F_n.

Розв'язок. Оскільки послідовність Фібоначчі задана рекурентним співвідношенням другого порядку, то для того, щоб запрограмувати обчислення її членів, необхідно три змінних. Модифікувавши наведений вище алгоритм для обчислення відповідного члена послідовності зада-

ної рекурентним співвідношенням третього порядку на випадок рекурентного співвідношення другого порядку, отримаємо програму:

```
#include <iostream.h>
void main(){
  int n, k, f, f1, f2;
  cin >> n;
  f1 = 1; f2 = 1;
  for (k = 2; k <= n; k++){
     f = f1 + f2;
     f2 = f1;
     f1 = f;
  }
  cout << f1 <<'\n';
}
```

Приклад 3.12. Скласти програму для обчислення визначника порядку n:

$$D_n = \begin{vmatrix} 5 & 3 & 0 & 0 & \dots & 0 & 0 \\ 2 & 5 & 3 & 0 & \dots & 0 & 0 \\ 0 & 2 & 5 & 3 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 2 & 5 \end{vmatrix}.$$

Розв'язок. Легко обчислити, що

$$D_1 = 5;$$

$$D_2 = \begin{vmatrix} 5 & 3 \\ 2 & 5 \end{vmatrix} = 19.$$

Розкладаючи D_n за першим рядком, отримаємо співвідношення

$$D_n = 5D_{n-1} - 6D_{n-2}, n \geq 3.$$

Отже, аналогічно до попереднього прикладу, програма матиме вигляд:

```
#include <iostream.h>
void main(){
  int n, k, D, D1, D2;
  cin >> n;
  D1 = 5;
  D2 = 19;
  for (k = 3; k <= n; k++){
     D = 5 * D1 – 6 * D2;
     D2 = D1;
     D1 = D;
  }
  cout << D <<'\n';
}
```

Приклад 3.13. Обчислити суму, задану рекурентним співвідношенням

$$S_n = \sum_{k=1}^{n} \frac{a_k}{2^k}, \text{ де } a_1 = a_2 = a_3 = 1, \ a_k = a_{k-1} + a_{k-3}, \ k \geq 4.$$

Розв'язок. Звернемо увагу на те, що послідовність a_k задано рекурентним співвідношенням третього порядку. Введемо допоміжну послідовність $b_k = 2^k$, $k \geq 0$, для якої рекурентне співвідношення матиме вигляд $b_1 = 1, b_k = 2b_{k-1}, k \geq 1$. Тоді, поєднуючи алгоритми для визначення відповідних членів послідовностей, заданих рекурентними співвідношеннями першого і третього порядків, отримаємо програму

```
#include <iostream.h>
void main(){
  int n, k;
  float S;
  int a, b;
    // додаткові змінні для визначення ak
  int a1, a2, a3;
  cin >> n;
  S = 0;
  a1 = a2 = a3 = 1;
  b = 1;
  // обчислення перших трьох членів послідовності
  for (k = 1; (k <= 3 && k <=n); k++){
    b = b * 2;
    S = S + 1 / float(b);
  }
  for (k = 4; k <= n; k++){
     b = b * 2;
     a = a1 + a3;
     S = S + float(a) / float(b);
     a3 = a2;
     a2 = a1;
     a1 = a;
  }
  cout << S <<'\n';
}
```

Приклад 3.14. Обчислити суму, задану рекурентним співвідношенням

$$S_n = \sum_{k=0}^{n} \frac{a_k}{1+b_k}, \text{ де } \begin{cases} a_0 = 1, \\ a_k = a_{k-1} b_{k-1}, \end{cases} \begin{cases} b_0 = 1, \\ b_k = a_{k-1} + b_{k-1}, \end{cases} \quad k \geq 1.$$

Розв'язок. Послідовності $\{a_k\}$ і $\{b_k\}$ задано рекурентним співвідношеннями першого порядку, проте залежність перехресна. Використаємо по одній допоміжній змінній для кожної із послідовностей.

Отримаємо програму для знаходження S_n:

```
#include <iostream.h>
void main(){
  int n;
  float S;
  int a, b, aa, bb;
  cin >> n;
  S = 0.5;
  a = 1;
  b = 1;
  for (int k = 1; k <= n; k++){
     aa = a * b;
     bb = a + b;
     a = aa;
     b = aa;
     S = S + float(a) / float(1 + b);
  }
  cout << S <<'\n';
}
```

Приклад 3.15. Обчислити добуток, заданий рекурентним співвідношенням

$$P_n = \prod_{k=0}^{n} \frac{a_k}{3^k}, \text{ де } \begin{cases} a_0 = a_1 = 1,\ a_2 = 3, \\ a_k = a_{k-3} + \dfrac{a_{k-2}}{2^{k-1}} \end{cases}, \ k \geq 3.$$

Розв'язок. Послідовність $\{a_k\}$ задана рекурентним співвідношенням третього порядку. Тоді добуток P_n обчислюється за допомогою рекурентного співвідношення

$$\begin{cases} P_2 = 1/9, \\ P_k = P_{k-1} \cdot a_k / z_k, \ k \geq 3, \end{cases}$$

де z_k – k-й степінь числа 3, визначений рекурентним співвідношенням

$$\begin{cases} z_2 = 9, \\ z_k = 3z_{k-1}, \ k \geq 3. \end{cases}$$

Передбачивши змінну t для обчислення членів послідовності $\{t_k = 2^{k-1} : k \geq 3\}$, отримаємо програму

```
#include <iostream.h>
void main(){
  int n; double P;
  int a, z, t;
  // додаткові змінні для визначення ak
```

```
    int a1, a2, a3;
    cin >> n;
    P = 1.0 / 9; z = 9; t = 2;
    a1 = 3; a2 = a3 = 1;
    for (int k = 3; k <= n; k++){
       z = z * 3;
       t = t * 2;
       a = a3 + a2 / float(t);
       a3 = a2;
       a2 = a1;
       a1 = a;
       P = P * float(a) / float(z);
    }
    cout << P <<'\n';
}
```

Задачі для самостійної роботи

3.19. Створити програми обчислення елементів послідовностей:

а) $x_k = \frac{x^k}{k},\ k \ge 1;$ б) $x_k = \frac{(-1)^k (k-1) x^k}{k},\ k \ge 2;$

в) $x_k = \frac{x^k}{k!(k+1)!},\ k \ge 0;$ г) $x_k = \frac{(-1)^k x^k}{(k^2+k)!},\ k \ge 0;$

ґ) $x_k = \frac{kx^{2k}}{(2k)!},\ k \ge 1;$ д) $x_k = \frac{(-1)^k x^{2k}}{(2k)!},\ k \ge 0;$

е) $x_k = \frac{x^{2k+1}}{(2k+1)!},\ k \ge 0;$ є) $x_k = \frac{(-1)^k x^{2k+1}}{(2k+1)!},\ k \ge 0;$

ж) $x_k = \frac{x^{4k+1}}{(2k)!(2k+1)!},\ k \ge 0;$ з) $x_k = \frac{(-1)^k x^{4k+1}}{k!(2k+1)!},\ k \ge 0.$

3.20. Скласти програми обчислення сум:

а) $S_n = 1+2+3+\ldots+n;$ б) $S_n = 1-2+3+\ldots+(-1)^n n;$

в) $S_n = 1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n};$ г) $S_n = 1-\frac{1}{2}+\frac{1}{3}-\ldots+(-1)^{n-1}\frac{1}{n};$

ґ) $S_n = 1+\frac{1}{1!}+\frac{1}{2!}+\ldots+\frac{1}{n!};$ д) $S_n = 1-\frac{2}{1!}+\frac{4}{2!}+\ldots+\frac{(-2)^n}{n!};$

е) $S_n = \frac{1}{2}-\frac{2}{3}+\frac{3}{4}-\ldots+\frac{(-1)^n (n-1)}{n};$

є) $S_n = \frac{1}{1!!} + \frac{2}{3!!} + \frac{4}{5!!} + \ldots + \frac{2^n}{(2n+1)!!}$.

3.21. Створити програми для обчислення добутків:

а) $P_n = \prod_{i=1}^{n}\left(1+\frac{1}{i^2}\right) = \left(1+\frac{1}{1^2}\right)\left(1+\frac{1}{2^2}\right)\ldots\left(1+\frac{1}{n^2}\right)$;

б) $P_n = \prod_{i=2}^{n}\left(1-\frac{1}{i^2}\right) = \left(1-\frac{1}{2^2}\right)\left(1-\frac{1}{3^2}\right)\ldots\left(1-\frac{1}{n^2}\right)$;

в) $P_n = \prod_{i=1}^{n}\left(2+\frac{1}{i!}\right)$; г) $P_n = \prod_{i=1}^{n}\frac{i+1}{i+2}$; ґ) $P_n = \prod_{i=1}^{n}\frac{1}{1+i!}$;

д) $P_n = \prod_{i=1}^{n}\frac{1}{1+i^i}$; е) $P_n = \frac{2}{1}\cdot\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdot\ldots$, n – співмножників.

3.22. Створити програми для обчислення ланцюгових дробів:

а) $1+\cfrac{1}{1+\cfrac{1}{1+\ddots+\cfrac{1}{1}}}$; б) $1+\cfrac{1}{2+\cfrac{1}{2+\ddots+\cfrac{1}{2}}}$;

в) $n+\cfrac{1}{(n-1)+\cfrac{1}{(n-2)+\ddots+\cfrac{1}{2}}}$; г) $1+\cfrac{1}{3+\cfrac{1}{5+\ddots+\cfrac{1}{2n+1}}}$.

3.23. Числами трибоначчі називається числова послідовність $\{T_k : k \ge 0\}$, задана рекурентним співвідношенням третього порядку:

$$T_0 = 0, T_1 = T_2 = 1, T_k = T_{k-1} + T_{k-2} + T_{k-3},\ k \ge 3.$$

Скласти програму для обчислення T_n.

3.24. Послідовністю Падована називається числова послідовність $\{P_k : k \ge 0\}$, задана рекурентним співвідношенням третього порядку:

$$P_0 = P_1 = P_2 = 1, P_k = P_{k-1} + P_{k-3},\ k \ge 3.$$

Скласти програму для обчислення P_n. Для всіх $n \le N$ безпосередньою перевіркою переконатися, що послідовність Падована задовольняє рекурентним формулам:

а) $P_n = P_{n-1} + P_{n-5}$; б) $P_n = P_{n-2} + P_{n-4} + P_{n-8}$;

в) $P_n = 2P_{n-2} - P_{n-7}$; г) $P_n = 4P_{n-5} + P_{n-14}$.

3.25. Скласти програму обчислення суми перших n членів:

а) послідовності Фібоначчі; б) послідовності трибоначчі;

в) послідовності Падована.

3.26. Скласти програми обчислення визначників порядку n:

а) $\begin{vmatrix} 2 & 3 & 0 & \dots & 0 \\ 1 & 2 & 3 & \dots & 0 \\ 0 & 1 & 2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 2 \end{vmatrix}$; б) $\begin{vmatrix} 2 & 1 & 0 & \dots & 0 \\ 1 & 2 & 1 & \dots & 0 \\ 0 & 1 & 2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 2 \end{vmatrix}$; в) $\begin{vmatrix} 3 & 2 & 0 & \dots & 0 \\ 1 & 3 & 2 & \dots & 0 \\ 0 & 1 & 3 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 3 \end{vmatrix}$;

г) $\begin{vmatrix} 7 & 5 & 0 & \dots & 0 \\ 2 & 7 & 5 & \dots & 0 \\ 0 & 2 & 7 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 7 \end{vmatrix}$; ґ) $\begin{vmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ -1 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & -1 & 0 \end{vmatrix}$;

д) $\begin{vmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{vmatrix}$; е) $\begin{vmatrix} a+b & ab & 0 & 0 & \dots & 0 \\ 1 & a+b & ab & 0 & \dots & 0 \\ 0 & 1 & a+b & ab & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & a+b \end{vmatrix}$;

є) $\begin{vmatrix} 1 & 2 & 0 & 0 & 0 & \dots & 0 & 0 \\ 3 & 4 & 3 & 0 & 0 & \dots & 0 & 0 \\ 0 & 2 & 5 & 3 & 0 & \dots & 0 & 0 \\ 0 & 0 & 2 & 5 & 3 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & 5 & 3 \\ 0 & 0 & 0 & 0 & 0 & \dots & 2 & 5 \end{vmatrix}$; ж) $\begin{vmatrix} a & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & a & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & a & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 1 & a \end{vmatrix}$;

з) $\begin{vmatrix} 1+x^2 & x & 0 & 0 & \dots & 0 & 0 \\ x & 1+x^2 & x & 0 & \dots & 0 & 0 \\ 0 & x & 1+x^2 & x & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & x & 1+x^2 \end{vmatrix}$.

3.27. Створити програми обчислення сум:

а) $S_n = \sum_{k=1}^{n} 2^k a_k$, де $a_1 = 0,\ a_2 = 1,\ a_k = a_{k-1} + k a_{k-2},\ k \ge 3$;

б) $S_n = \sum_{k=1}^{n} \frac{3^k}{a_k}$, де $a_1 = 1,\ a_2 = 1,\ a_k = \frac{a_{k-1}}{k} + a_{k-2},\ k \ge 3$;

в) $S_n = \sum_{k=1}^{n} \frac{k!}{a_k}$, де $a_1 = 1,\ a_2 = 1,\ a_k = a_{k-1} + \frac{a_{k-2}}{2^k},\ k \ge 3$;

г) $S_n = \sum_{k=1}^{n} k! a_k$, де $a_1 = 0,\ a_2 = 1,\ a_k = a_{k-1} + \frac{a_{k-2}}{(k-1)!},\ k \ge 3$;

ґ) $S_n = \sum_{k=1}^{n} \frac{a_k}{2^k}$, де $a_1 = a_2 = a_3 = 1,\ a_k = a_{k-1} + a_{k-3},\ k \ge 4$;

д) $S_n = \sum_{k=1}^{n} \frac{2^k}{k!} a_k$, де $a_1 = 1,\ a_k = k a_{k-1} + \frac{1}{k},\ k \ge 2$.

3.28. Створити програми обчислення сум:

а) $S_n = \sum_{k=1}^{n} \frac{2^k}{a_k + b_k}$, де $\begin{cases} a_1 = 0, a_2 = 1, \\ a_k = \frac{a_{k-1}}{k} + a_{k-2} b_k, \end{cases} \begin{cases} b_1 = 1, b_2 = 1, \\ b_k = b_{k-1} + a_{k-1}, \end{cases} k \ge 3$;

б) $S_n = \sum_{k=1}^{n} \frac{a_k b_k}{(k+1)!}$, де $\begin{cases} a_1 = u, \\ a_k = 2b_{k-1} + a_{k-1}, \end{cases} \begin{cases} b_1 = v, \\ b_k = 2a_{k-1}^2 + b_{k-1}, \end{cases} k \ge 2$;

u, v – задані дійсні числа;

в) $S_n = \sum_{k=1}^{n} \frac{2^k}{\left(1 + a_k^2 + b_k^2\right) k!}$ де $\begin{cases} a_1 = 1, \\ a_k = 3b_{k-1} + 2a_{k-1}, \end{cases} \begin{cases} b_1 = 1, \\ b_k = 2a_{k-1} + b_{k-1}, \end{cases} k \ge 2$;

г) $S_n = \sum_{k=1}^{n} \left(\frac{a_k}{b_k} \right)^k$, де $\begin{cases} a_0 = 1, a_1 = 2, \\ a_k = a_{k-2} + \frac{b_k}{2}, \end{cases}, \begin{cases} b_0 = 5, b_1 = 5, \\ b^k = b_{k-2}^2 - a_{k-1}, \end{cases} k \ge 2$;

ґ) $S_n = \sum_{k=0}^{n} \frac{a_k}{1 + b_k}$, де $\begin{cases} a_0 = 1, \\ a_k = a_{k-1} b_{k-1}, \end{cases} \begin{cases} b_0 = 1, \\ b_k = a_{k-1} + b_{k-1}, \end{cases} k \ge 1$.

3.29. Створити програми для обчислення добутків:

а) $P_n = \prod_{k=0}^{n} \frac{a_k}{3^k}$, де $\begin{cases} a_0 = a_1 = 1,\ a_2 = 3, \\ a_k = a_{k-3} + \frac{a_{k-2}}{2^{k-1}} \end{cases}, k \ge 3$;

б) $P_n = \prod_{k=0}^{n} a_k b_k$, де $\begin{cases} a_1 = 1, \\ a_k = \left(\sqrt{b_{k-1}} + a_{k-1}\right) / 5 \end{cases}$, $\begin{cases} b_1 = 1, \\ b_k = 2b_{k-1} + 5a_{k-1}^2 \end{cases}$, $k \geq 2$.

3.3. Цикли за умовою

На відміну від циклу з лічильником, **цикл за умовою** виконується не наперед задану кількість разів, а доки істинною є деяка умова. У С є два типи циклів за умовою:

цикл з **передумовою**

```
while (F) {
  Q;
}
```

цикл з **післяумовою**

```
do {
  Q;
} while (F);
```

де F – умова, Q – тіло циклу. Обидва цикли виконують набір інструкцій Q, доки умова F залишається істинною. Різниця полягає лише в тому, що для циклу з передумовою перевірка умови F здійснюється до першого виконання тіла циклу Q. Для циклу ж з післяумовою перевірка умови F здійснюється вже після того, як виконається перший раз тіло циклу Q. Цикли з післяумовою здебільшого використовують у випадках, коли умова F є невизначеною до початку виконання циклу.

Зауважимо, що цикл з післяумовою легко отримується з циклу з передумовою:

```
do {                    Q; while (F){
  Q;          ≡           Q;
} while (F);            }
```

До циклів за умовою також належить і **цикл з виходом**. Окремої конструкції даний тип циклів у С не має. Його моделювання здійснюється за допомогою оператора `break`. Як тільки програмі трапляється цей оператор, вона припиняє роботу циклу та переходить до виконання наступного оператора, що йде після циклу. Як приклад використання цього оператора наведемо моделювання циклу з виходом з допомогою циклу з передумовою. Нагадаємо, що у С будь-яке ненульове значення арифметичного виразу означає *Істину*.

```
do {                        while ( 1 ){
  Q;              ≡           Q;
}                             if (!F) break;
while (F);                  }
```

Приклади розв'язання задач

Приклад 3.16. Задана непорожня послідовність ненульових цілих чисел, за якою йде 0. Визначити кількість змін знака в цій послідовності. Наприклад, у послідовності 1, –34, 8, 14, –5, 0 знак змінюється тричі.

Розв'язок. Оскільки кількість членів послідовності наперед невідома, то очевидно, що для введення членів послідовності необхідно використати цикл з умовою. Умовою продовження циклу є те, що введене з клавіатури число не є нулем. Тут зручніше використати цикл з післяумовою, оскільки до початку виконання циклу невідомо, яке саме число ввів користувач.

У програмі використаємо допоміжну змінну p, у якій будемо запам'ятовувати попередній член послідовності.

```
#include <iostream.h>
void main(){
  int a, p = 0, z = 0;
  do {
    cin >> a;
    if (a * p < 0) z++;
    p = a;
  } while (a != 0);
  cout << "кількість знакозмін" << z <<'\n';
}
```

Наведемо модифікацію попередньої програми з використанням циклу з виходом.

```
#include <iostream.h>
void main(){
  int a, p = 0, z = 0;
  do {
    cin >> a;
    if (a == 0) break;
    if (a * p < 0) z++;
    p = a;
  } while (1);
  cout << "кількість знакозмін" << z <<'\n';
}
```

Приклад 3.17. Послідовність задано рекурентним співвідношенням

$$\begin{cases} x_0 = 1, x_1 = 0, x_2 = 1, \\ x_n = x_{n-1} + 2x_{n-2} + x_{n-3}, n \geq 3. \end{cases}$$

Створити програму для знаходження номера найменшого члена цієї послідовності, який перевищує число a.

Розв'язок. Оскільки послідовність x_n задано рекурентним співвідношенням третього порядку, то для обчислення її довільного елементу

потрібні чотири змінні. Нехай змінна x пробігає послідовність $\{x_n : n \geq 1\}$, тоді умовою завершення циклу є умова x > a, отже, цикл має виконуватися, доки виконується умова x <= a. Таким чином, маємо програму

```
#include <iostream.h>
void main(){
  int n, x, x1, x2, x3, a;
  cin >> a;
  x1 = 1; x2 = 0; x3 = 1; n = 2;
  do {
     n++;
     x = x1 + 2*x2 + x3;
     x3 = x2;
     x2 = x1;
     x1 = x;
  } while (x <= a);
  cout << "n = " << n <<'\n';
}
```

Приклад 3.18. Створити програму для "обернення" (запису в оберненому порядку цифр) заданого натурального числа.

Розв'язок. Нехай у цілочисельній змінній n міститься натуральне число. Скористаємося рекурентним співвідношенням

$$\begin{cases} y_0 = 0; \\ y_k = 10 y_{k-1} + a_k, \end{cases}$$

де a_k – k-та цифра числа n при розгляді цифр справа наліво. Таким чином, програма матиме вигляд:

```
#include <iostream.h>
void main(){
  int n, k = 0;
  cin >> n;
  while (n > 0){
    k = 10 * k + n % 10;
    n = n / 10;
  }
  cout << k <<'\n';
}
```

Приклад 3.19. Маємо ціле $n > 2$. Знайти всі прості числа з діапазону $[2, n]$.

Розв'язок. Нехай m послідовно набуває значень цілих чисел із діапазону $[2, n]$. Тоді m буде простим числом, якщо воно не має дільників у діапазоні $[2, \sqrt{m}]$. Ураховуючи це, напишемо програму:

```
#include <iostream.h>
void main(){
```

```
  int p, k, n, m;
  do              // Операція захищеного введення.
    cin >> n;     // Запобігає неправильному
  while (n <= 2); // введенню даних.
  cout << 2 << ", ";
  m = 3;
  while (m <= n){
    p = 1;
    k = 3;
    while (k * k <= m) {
      if (m % k == 0) {
        p = 0;
        break;
      }
      k = k + 2;
    }
    if (p) cout << m <<", ";
    m = m + 2;
  }
  cout << "\n";
}
```

Приклад 3.20. Написати програму, яка визначає, чи є задане натуральне число n досконалим, тобто рівним сумі всіх своїх дільників, крім самого цього числа (напр., числа 6 і 28: $6 = 1 + 2 + 3$, $28 = 1 + 2 + 4 + 7 + 14$).

Розв'язок. Шукатимемо суму S усіх дільників заданого числа n. Якщо $S = n$, то число, яке перевіряємо, є досконалим. Програму можна написати, використовуючи одну з двох ідей. Перша полягає у знаходженні дільників числа n у діапазоні $[1, n/2]$. Відповідно до другої ідеї пошук проводиться у діапазоні $[1, \sqrt{n}]$, і якщо дільник знайдено, то до суми S додаються як дільник, так і частка.

Напишемо програму згідно з першою ідеєю. У змінній S цілого типу підрахуємо суму дільників, числа, що міститься у змінній n. Використаємо змінну k, яка пробігатиме всі цілі значення в проміжку від `1` до `n/2`.

```
#include <iostream.h>
void main(){
  int S, k, n;
  cin >> n;
  S = 0;
  k = 1;
  while (k <= n / 2) {
    if (n % k == 0) // якщо k - дільник
       S = S + k;   // до суми додаємо k
    k = k + 1;
  }
```

```
  if (S == n) cout << "Число є досконалим\n." ;
  else cout << "Число не є досконалим.\n";
}
```

Напишемо програму згідно з другою ідеєю. Цього разу змінна k пробігатиме всі цілі значення в проміжку від 1 до sqrt(n). У програмі слід врахувати окремо граничні випадки k = 1 і k=sqrt(n).

```
#include <iostream.h>
void main(){
  int S, k, n;
  cin >> n;
  S = 1; // До суми додаємо перший дільник – 1
  /* Пошук дільників здійснюємо у
              діапазоні від 2 до sqrt(n) - 1 */
  k = 2;
  while (k <= sqrt(n)) {
    if (n % k == 0) {// якщо k - дільник
       S += k;       // до суми додаємо дільник
       S += n / k;   // до суми додаємо частку
    }
    k = k + 1;
  }
  if (k * k == n) S += k; // випадок k = sqrt(n)
  if (S == n) cout << "Число є досконалим\n.";
  else cout << "Число не є досконалим.\n";
}
```

Слід зауважити, що програма, написана згідно з першою ідеєю, має коротший та очевидніший запис, проте друга – виконуватиметься значно швидше, оскільки діапазон пошуку дільників суттєво вужчий.

Приклад 3.21. За допомогою розвинення функції у ряд Тейлора обчислити з точністю $\varepsilon > 0$ її значення для заданого значення x.

$$y = e^x = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \ldots \cdot$$

Зауваження. У загальному випадку обчислення значень алгебраїчних функцій перебуває поза межами даного посібника та вимагає від читача додаткових знань із теорії чисельних методів [9].

Розв'язок. Зафіксуємо параметр x. Розглянемо послідовність

$$y_n = \sum_{k=0}^{n} \frac{x^k}{k!}.$$

Очевидно, що $y_n \to y$ якщо $n \to \infty$.

Під наближеним із точністю ε значенням границі послідовності y_n розумітимемо такий член y_N послідовності, що виконується співвідношення

$$| y_N - y_{N-1} | < \varepsilon \qquad (3.6)$$

Послідовність y_n можна задати рекурентним співвідношенням:

$$\begin{cases} y_0 = a_0 = 1, \\ a_k = a_{k-1} x / k, \quad k \geq 1. \\ y_k = y_{k-1} + a_k, \end{cases}$$

З огляду на рекурентне співвідношення бачимо, що умова (3.6) для y_n набуде вигляду $|a_k| < \varepsilon$. Члени послідовності y_n шукатимемо за допомогою циклу за умовою, яка є запереченням умови (3.6). Тоді програма:

```
#include <iostream.h>
#include <math.h>
void main(){
  const float eps = 0.000001;
  float y, x, a;
  int k;
  cin >> x;
  y = a = 1;
  k = 0;
  while (fabs(a) >= eps){
    k = k + 1;
    a = a * x / float(k);
    y = y + a;
  }
  cout << y << '\n';
}
```

У вищенаведеній програмі було підключено бібліотеку `math.h`, де міститься реалізація багатьох математичних функцій. Зокрема, там міститься функція `fabs`, яка обчислює модуль дійсного числа.

Зауваження. З огляду на попереднє зауваження, слід зазначити, що продемонстрований підхід для інших функцій може призводити до хибних результатів. Тому надалі для таких функцій у формулюванні задачі буде вказано, що потрібно розуміти під наближеним значенням.

Приклад 3.22. Знайти корінь рівняння $\mathrm{tg}x = x$ на відрізку $[0.001, 1.5]$ із заданою точністю ε, використовуючи метод ділення відрізка навпіл.

Розв'язок. Метод бісекції (або ділення відрізка навпіл) – один із найпростіших чисельних методів розв'язання нелінійних рівнянь вигляду

$$f(x) = 0,$$

де від функції $f(x)$ вимагається лише неперервність. Якщо відомо, що на деякому відрізку існує єдиний корінь цього рівняння, то очевидно, що на кінцях відрізка має бути функція з різними знаками. Поділивши відрізок навпіл, візьмемо ту половину, для якої на кінцях функція набуватиме значень різних знаків. Якщо середина відрізка виявиться нулем функції, то процес завершується. Якщо задано точність обчислення ε, то процедуру поділу слід виконувати, доки довжина відрізка не стане меншою за ε, а наближеним значенням вважають середину цього відрізка.

Отже, для розв'язання рівняння $\mathrm{tg}x = x$ розглянемо функцію

$$f(x) = tgx - x.$$

Тоді програма матиме вигляд:

```
#include <iostream.h>
#include <math.h>
void main(){
  const float eps = 0.000000000000001; // точність
  float x, l, r;
  l = 0.001;          // лівий кінець відрізка
  r = 1.5;            // правий кінець відрізка
  // tg(l) - l < 0, tg(r) - r > 0
   while (r - l >= eps) {
      x = (l + r) / 2; //середина відрізка [l,r]
      if (tan(x) - x < 0)
         l = x;    // [l,r] = [x,r]
      else
         r = x;    // [l,r] = [l,x]
  }
  // корінь - середина останнього відрізка [l,r]
  x = (l + r) / 2;
  cout << x << "\n";
}
```

Задачі для самостійної роботи

3.30. Для довільного цілого числа $m > 1$ знайти найбільше ціле k, за якого $4^k < m$.

3.31. Для заданого натурального числа n одержати найменше число вигляду 2^r, яке перевищує n.

3.32. Визначити, зі скількох від'ємних чисел починається задана послідовність чисел.

3.33. Дано непорожню послідовність різних натуральних чисел, за якою йде 0. Визначити порядковий номер найменшого з них.

3.34. Дано непорожню послідовність різних дійсних чисел, серед яких є принаймні одне від'ємне число, за якою йде 0. Визначити величину найбільшого серед від'ємних членів цієї послідовності.

3.35. Дано непорожню послідовність із натуральних чисел, за якою йде 0. Обчислити суму тих із них, порядкові номери яких – числа Фібоначчі.

3.36. Маємо дійсне число a. Скласти програму обчислення:

а) серед чисел $1, 1+1/2, 1+1/2+1/3,$ ... першого, більшого за a;

б) такого найменшого n, що $1+1/2+...+1/n > a$.

3.37. Скласти програми обчислення:

а) найбільшого числа Фібоначчі, яке не перевищує число a;

б) номера найменшого числа Фібоначчі, яке більше від числа a;

в) суми всіх чисел Фібоначчі, які не перевищують число a.

3.38. Послідовність задано рекурентним співвідношенням

$$\begin{cases} x_0 = x_2 = 1,\ x_1 = 0, \\ x_n = 2x_{n-1} + 3x_{n-3}, n \geq 3. \end{cases}$$

Створити програму для знаходження найбільшого члена цієї послідовності та його номера, який не перевищує число a.

3.39. Створити програму, яка з'ясовує, чи входить задана цифра до запису заданого натурального числа.

3.40. Натуральне число називається паліндромом, якщо його запис читається однаково зліва направо і справа наліво. Написати програму, що визначає, чи є введене з клавіатури число паліндромом.

3.41. Маємо ціле $n > 2$. Скласти програму для знаходження всіх простих чисел із діапазону $[2, n]$, які є

а) числами послідовності Фібоначчі; б) паліндромами;

в) числами вигляду $q^2 + 1$, де q – ціле число.

3.42. Скласти програму знаходження всіх простих дільників заданого натурального числа.

3.43. Число називається досконалим, якщо дорівнює сумі всіх своїх дільників, крім самого цього числа. Перевірити, чи існують досконалі числа з проміжку $[2, n]$, що є повними квадратами. Якщо так, то вивести на екран перше з них.

3.44. Знайти k-ту цифру послідовності:

а) 110100100010000 ... , у якій виписано поспіль степені 10;

б) 123456789101112 ... , у якій виписано поспіль усі натуральні числа;

в) 149162536 ... , де виписано поспіль квадрати всіх натуральних чисел;

г) 01123581321 ... , у якій виписані поспіль усі числа Фібоначчі.

3.45. За допомогою розвинення функції у ряд Тейлора обчислити з точністю $\varepsilon > 0$ її значення для заданого значення x:

а) $y = \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ...;$ б) $y = \cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - ...;$

в) $y = \operatorname{sh} x = x + \frac{x^3}{3!} + \frac{x^5}{5!} + ...;$ г) $y = \operatorname{ch} x = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + ...;$

ґ) $y = \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - ..., \ (|x| < 1);$

д) $y = \frac{1}{1+x} = 1 - x + x^2 - x^3 + ..., \ (|x| < 1);$

е) $y = \ln\frac{1+x}{1-x} = 2 \cdot \left(\frac{x}{1} + \frac{x^3}{3} + \frac{x^5}{5} + ... \right), \ (|x| < 1);$

є) $y=\dfrac{1}{(1+x)^2}=1-2x+3x^2-...,\ \ (|x|<1);$

ж) $y=\dfrac{1}{(1+x)^3}=1-\dfrac{2\cdot 3}{2}\cdot x+\dfrac{3\cdot 4}{2}\cdot x^2-\dfrac{4\cdot 5}{2}\cdot x^3+...,\ \ (|x|<1);$

з) $y=\dfrac{1}{1+x^2}=1-x^2+x^4-x^6+...,\ \ (|x|<1);$

и) $y=\sqrt{1+x}=1+\dfrac{1}{2}\cdot x-\dfrac{1}{2\cdot 4}\cdot x^2+\dfrac{1\cdot 3}{2\cdot 4\cdot 6}\cdot x^3-...,\ \ (|x|<1);$

і) $y=\dfrac{1}{\sqrt{1+x}}=1-\dfrac{1}{2}\cdot x+\dfrac{1\cdot 3}{2\cdot 4}\cdot x^2-\dfrac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\cdot x^3+...,\ (|x|<1);$

ї) $y=\arcsin x=x+\dfrac{1}{2}\cdot\dfrac{x^3}{3}+\dfrac{1\cdot 3}{2\cdot 4}\cdot\dfrac{x^5}{5}+...,\qquad (|x|<1).$

3.46. Для заданого значення $x\ \left(|x|<\left(\sqrt{5}-1\right)/2\right)$ з точністю $\varepsilon>0$ обчислити генератрису послідовності чисел Фібоначчі:

$$\frac{x}{1-x-x^2}=\sum_{n=0}^{\infty}F_n x^n,$$

де $\{F_n, n\ge 0\}$ – послідовність чисел Фібоначчі.

3.47. Дано дійсні числа x,ε $(x\ne 0,\varepsilon>0)$. Обчислити з точністю ε нескінченну суму та вказати кількість урахованих доданків:

а) $\sum\limits_{k=0}^{\infty}\dfrac{x^{2k}}{2k!}$; б) $\sum\limits_{k=0}^{\infty}\dfrac{(-1)^k x^k}{(k+1)^2}$; в) $\sum\limits_{k=0}^{\infty}\dfrac{x^{2k}}{2^k k!}$; г) $\sum\limits_{k=0}^{\infty}\dfrac{(-1)^k x^{2k+1}}{k!(2k+1)}$.

3.48. Методом ділення відрізка навпіл знайти корінь рівняння $\sin x=x/3$ із заданою точністю $\varepsilon>0$ на відрізку $[1.6,\ 3]$.

3.49. Із заданою точністю ε знайти корінь рівняння $x^3+4x^2+x-6=0$, що належить відрізку $[0,\ 2]$.

Указівка. Одним із методів розв'язування рівняння є метод хорд, який полягає в обчисленні елементів послідовності

$$u_0=a;$$

$$u_n=u_{n-1}-\frac{y(u_n-1)}{y(b)-y(u_{n-1})}\cdot(b-u_{n-1}),$$

до виконання умови $|u_n-u_{n-1}|<\varepsilon$. За умов нашої задачі

$$a=0,\ b=2,$$

$$y(x)=x^3+4x^2+x-6.$$

Розділ 4
ПРОСТІ ТИПИ ДАНИХ

Пам'ять комп'ютера є послідовністю 8-розрядних клітинок – **байтів**. Кожний розряд – його називають **бітом** – може набувати одне з двох значень: 0 або 1. Згідно з основним правилом комбінаторики, за допомогою одного байта можна закодувати $2^8 = 256$ різних значень. Наприклад, за допомогою одного байта можна записувати натуральні числа з діапазону від 0 до 255 або цілі числа з діапазону від –128 до 127. Очевидно, що цього вистачить тільки для найпростіших обчислень. Тому передбачене об'єднання байтів у групи по два, чотири або більше байтів для створення зручніших типів даних. У цьому розділі розглядаються загальноприйняті способи кодування цілих і дійсних чисел, символьних даних і типів даних, що створюються користувачем.

4.1. Цілі типи даних

Для роботи із цілочисельними даними у С використовується базовий цілочисельний тип `int`. За допомогою кваліфікаторів `short` (короткий), `long` (довгий), `unsigned` (без знака), `signed` (зі знаком) на базі типу `int` утворюються похідні цілочисельні типи. Наприклад,

```
signed int a, b, c;
signed short int d;
unsigned long int x, y;
```

Якщо вказано кваліфікатори, ключове слово `int` можна опустити. Якщо опущено кваліфікатор `signed` або `unsigned`, то вважається, що вказано кваліфікатор `signed`. Якщо опущено кваліфікатор `short` або `long`, то вважається, що використовується кваліфікатор `short`. Наприклад,

```
int a, b, c;
short d;
unsigned long x, y;
```

У С для змінних, що створюються на базі цілого типу з кваліфікатором `short`, відводиться 2 байти, а для змінних на базі цілого типу з кваліфікатором `long` – 4 байти. Таким чином, діапазон зміни значень змінних різних цілих типів можна записати у вигляді таблиці

Таблиця 4.1. Цілі типи даних

Тип даних	Кількість байтів	Діапазон
`unsigned short`	2	$0 \dots 2^{16}-1$
`unsigned long`	4	$0 \dots 2^{32}-1$
`signed short`	2	$-2^{15} \dots 2^{15}-1$
`signed long`	4	$-2^{31} \dots 2^{31}-1$

Над елементами `x`, `y` цілого типу визначено:

▪▪ операції:

`x + y` – додавання;
`x - y` – віднімання;
`x * y` – множення;
`x / y` – частка від ділення без остачі;
`x % y` – остача від ділення.

▪▪ функції обчислення:

`abs(x)` – модуля `x` (для `int x`);
`labs(x)` – модуля `x` (для `long x`).

Зауважимо, що для роботи з вищенаведеними функціями `abs` і `labs` необхідно підключити бібліотеку `math.h`.

Для простого введення/виведення цілих змінних користуються інструкціями (див. ст. 6) `cin/cout`. Для форматованого введення/виведення використовують інструкції (див. ст. 6) `scanf/printf`.

Специфікатори цілих типів для операцій `scanf/printf`:

`d, i` – десяткове знакове число. За замовчуванням записується з правим вирівнюванням, знак друкується тільки для від'ємних чисел;

`o` – вісімкове беззнакове число;

`u` – десяткове беззнакове число;

`x` і `X` – шістнадцяткове число, `x` використовує маленькі літери (`abcdef`), `X` – великі (`ABCDEF`);

Для довгих змінних (`long`) параметр `розмір` у відповідній керуючій послідовності має набувати значення `l`, наприклад для зчитування довгої беззнакової змінної `a` можна скористатися рядком:

```
scanf("%lu", &a);
```

і, відповідно, для виведення:

```
printf("Змінна a = %lu", &a);
```

Приклади розв'язання задач

Приклад 4.1. Написати програму для обчислення найбільшого спільного дільника двох цілих чисел за допомогою модифікованого алгоритму Евкліда.

Розв'язок

```
#include <iostream.h>
#include <math.h>
void main(){
  int N, M, U, V, P;
  cin >> N >> M;
  U = abs(N); V = abs(M);
  if (U < V){
    P = U; U = V; V = P;
  }
  while (V > 0) {
    P = V;
    V = U % V;
    U = P;
  }
  cout << U << '\n' ;
}
```

Приклад 4.2. Дано: натуральне число n. Визначити серед чисел $1,...,n$ усі такі, запис яких збігається з останніми цифрами запису їхніх квадратів (напр., $6^2 = 36$, $25^2 = 625$ тощо).

Розв'язок. Очевидно, що якщо запис числа x збігається з останніми цифрами його квадрата x^2, то різниця $x^2 - x$ ділиться на число $k = 10^p$, де p – кількість цифр у числі x. Ураховуючи це, запишемо програму:

```
#include <iostream.h>
void main(){
  unsigned long int n, k, x, y;
  cin >> n;
  for (x = 1; x <= n; x++) {
    // Знаходження числа k для поточного x
    y = x;
    k = 1;
    while (y != 0) {
      y = y / 10;
      k = k * 10;
    }
    if ( x * (x-1) % k == 0)
      cout << x << '\n' ;
  }
}
```

Приклад 4.3. Дано натуральне число n. Скласти програму знаходження всіх менших або рівних n натуральних чисел, які при піднесенні до квадрата дають паліндром.

Розв'язок. Натуральне число називається паліндромом, якщо його запис читається однаково зліва направо та справа наліво. Для чисел $x = 1,...,n$ порівнюватимемо значення $y = x^2$ із числом k, що є записом числа y у зворотному порядку.

```
#include <iostream.h>
void main(){
  unsigned long int n, k, x, y;
  cin >> n;
  for (x = 1; x <= n; x++) {
    y = x * x;
    // Знаходження оберненого запису для числа y
    k = 0;
    while (y > 0) {
      k = k * 10 + y % 10;
      y = y / 10;
    }
    if ( k == x * x)
      cout << x << '\n' ;
  }
}
```

Приклад 4.4. Задано натуральні числа n і p, цілі числа $a_1,\ldots,a_n$. Створити програму обчислення добутку членів послідовності, кратних p.

Розв'язок. У програмі використаємо булеву змінну k, що буде 0, якщо кратних до p серед чисел $a_1,\ldots,a_n$ не буде.

```
#include <iostream.h>
void main(){
  unsigned int n, p;
  long int s;
  int a, k, i;
  cin >> n >> p;
  s = 1; k = 0;
  for (i = 1; i <= n; i++) {
    cin >> a;
    if ( a % p == 0) {
      s = s * a;
      k = 1;
    }
  }
  if (k) cout << s <<'\n' ;
  else cout << "Таких чисел немає" <<'\n';
}
```

Задачі для самостійної роботи

4.1. Створити програми для обчислення найбільшого спільного дільника за допомогою класичного алгоритму Евкліда (тобто з використанням тільки операції віднімання):

а) двох натуральних чисел; б) двох цілих чисел

4.2. Використовуючи формулу

$$НСК(m,n)=\frac{m\cdot n}{НСД(m,n)},$$

де $НСД(m,n)$ – найбільших спільний дільник чисел m і n, скласти програму для обчислення найменшого спільного кратного двох натуральних чисел m і n.

4.3. Дано натуральні числа m і n. Знайти такі натуральні числа p і q, не виключаючи спільних дільників, що $p/q=m/n$.

4.4. Знайти всі прості нескоротні дроби, що містяться між 0 і 1, знаменники яких не перевищують 7 (дріб задається двома числами – чисельником і знаменником).

4.5. Дано натуральне число n. Знайти всі такі натуральні q, що n ділиться на q^2 і не ділиться на q^3.

4.6. Дано натуральні числа m і n. Написати програму для знаходження всіх їх натуральних спільних кратних, менших за $m\cdot n$.

4.7. Дано натуральні числа m і n. Скласти програму знаходження всіх їхніх спільних дільників.

4.8. Два натуральних числа називаються "дружніми", якщо кожне з них дорівнює сумі всіх дільників іншого, крім самого цього числа. Скласти програму знаходження всіх пар "дружніх" чисел, що лежать у діапазоні $[200, 300]$.

4.9. Дано натуральне число n. Скласти програму знаходження всіх піфагорових трійок натуральних чисел, кожне з яких не перевищує n, тобто всіх таких трійок натуральних чисел a, b, c, що $a^2+b^2=c^2$ $(a\le b\le c\le n)$.

4.10. Натуральне число із n цифр є числом Армстронга, якщо сума його цифр, піднесених до n-го степеня, дорівнює самому числу (напр., $153=1^3+5^3+3^3$). Скласти програму знаходження всіх чисел Армстронга, що містить дві, три та чотири цифри.

4.11. Дано натуральне число n. Скласти програму, що визначає, чи можна подати його у вигляді суми двох квадратів натуральних чисел. Якщо це можливо, то

а) указати пару a, b таких натуральних чисел , що $n=a^2+b^2$;

б) указати пару a, b таких натуральних чисел, що $n=a^2+b^2,\ a\ge b$.

4.12. Дано натуральне число n. Скласти програму, що дозволяє
а) переставити першу та останню цифри числа n;
б) приписати по одиниці до початку та кінця запису числа n;
в) одержати суму m останніх цифр числа n.

4.13. Дано натуральне число n. Скласти програму, що визначає серед чисел $1,...,n$ усі такі, запис яких збігається з останніми цифрами запису їхніх кубів (напр., $4^3 = 64$, $25^3 = 15625$ тощо).

4.14. Натуральне число називається паліндромом, якщо його запис читається однаково зліва направо та навпаки (напр., 1, 393, 4884). Скласти програму, що визначає, чи є задане натуральне число n паліндромом.

4.15. Дано натуральне число n. Написати програму для знаходження всіх менших за n паліндромів, які при піднесенні до квадрата дають також паліндром.

4.16. Задано натуральне число n і цілі числа $a, x_1,\ldots, x_n$. Знайти номер члена послідовності, який дорівнює a. Якщо такого члена послідовності немає, то результатом має бути число 0.

4.17. Задано натуральне число n і цілі числа $a, x_1,..., x_n$. Якщо в послідовності $x_1,\ldots, x_n$ є принаймні один член, що дорівнює a, то отримати суму всіх членів, які йдуть за таким членом послідовності, у протилежному випадку відповіддю має бути відповідне текстове повідомлення.

4.18. Задано натуральне число n і цілі числа $a_1,\ldots, a_n$. Отримати суму додатних, кількість від'ємних і число нульових членів послідовності $a_1,\ldots, a_n$.

4.19. Задано натуральні числа n і p і цілі числа $a_1,\ldots, a_n$. Скласти програму обчислення добутку членів послідовності, кратних p.

4.20. Дано натуральне число n. Визначити, чи можна подати $n!$ у вигляді добутку трьох послідовних чисел.

4.2. Дійсні типи даних

Найуживаніший формат представлення дійсних чисел у інформатиці – експоненціальний запис.

Означення 4.1. Експоненціальний запис (**формат**) – представлення дійсних (дробових) чисел у вигляді **мантиси** та **порядку**:

$$N = M \cdot n^p,$$

де N – число, M – мантиса, n – основа показникової функції, p – порядок.

В інформатиці основу показникової функції n беруть рівною 10, а комп'ютерну реалізацію чисел у експоненціальній формі називають

числами з плаваючою комою (**плаваючою крапкою**). Показник степеня в обчислювальних машинах прийнято відділяти від мантиси символами "E" або "e" (скорочення від "exponent"). Наприклад, число $1.234 \cdot 10^{-56}$ у більшості мов програмування записується так:

```
1.1234E-56
```

Слід зауважити, що представлення дійсного числа в показниковій формі є неоднозначним. Наприклад, число 0.0001 у показниковій формі можна записати багатьма способами, наприклад,

$$0.0001 = 10 \cdot 10^{-5} = 0.1 \cdot 10^{-3}.$$

Тому в інформатиці використовують нормалізовану форму запису чисел з плаваючою комою.

Означення 4.2. Нормалізованою формою запису числа з плаваючою комою називають представлення, у якому мантиса належить проміжку $[1,2)$. У такій формі будь-яке число (крім нуля) записується єдиним чином. Наприклад, вищенаведене число 0.0001 у нормалізованій формі матиме вигляд:

$$0.0001 = 1 \cdot 10^{-4}.$$

У мові С є три дійсні типи: `float`, `double` та `long double`. Діапазони значень, кількість цифр мантиси та кількість байтів для цих типів наведено в таблиці.

Таблиця 4.2. Дійсний тип даних

Тип	Байтів	Цифр мантиси	Діапазон точності
`float`	4	7– 8	`1.5E-45 ... 3.4E38`
`double`	8	15– 16	`5.0E-324 ... 1.7E308`
`long double`	10	19– 20	`3.4E-4932 ... 1.1E4932`

Нехай елементи `x` та `y` дійсного типу. Тоді для них визначено:

- операції:
  - `x + y` – додавання;
  - `x - y` – віднімання;
  - `x * y` – множення;
  - `x / y` – ділення.
- функції:
  - `fabs(x)` – обчислення модуля `x`;
  - `sqrt(x)` – обчислення квадратного кореня з `x`;
  - `sin(x)` – обчислення синуса `x`;
  - `cos(x)` – обчислення косинуса `x`;
  - `asin(x)` – обчислення арксинуса `x`;

`acos(x)` – обчислення арккосинуса `x`;
`atan(x)` – обчислення арктангенса `x`;
`log(x)` – обчислення натурального логарифма від `x`;
`log10(x)` – обчислення десяткового логарифма від `x`;
`exp(x)` – обчислення експоненти від `x`;
`pow(x, y)` – обчислення `x` у степені `y`;
`ceil(x)` – найменше ціле число, більше або рівне `x`;
`floor(x)` – найбільше ціле число, менше або рівне `x`.

Зауваження. Для використання вищенаведених функцій слід підключити бібліотеку `math.h`. Результатом усіх функцій буде значення типу `double`.

Для простого введення/виведення дійсних значень користуються інструкціями (див. ст. 6) `cin/cout`. Для форматованого введення/виведення використовують інструкції (див. ст. 6) `scanf/printf`.

Специфікатори дійсних типів для інструкцій `scanf/printf`:

`f` – дійсне число – за замовчуванням виводиться з точністю 6 знаків; якщо число за модулем менше за одиницю, то перед десятковою комою пишеться 0;

`e` та `E` – дійсне число в експоненціальній формі запису (тобто виду `1.1e+44`); `e` виводить символ `'e'` у нижньому регістрі, `E` – у верхньому (`3.14E+0`);

`a` та `A` — дійсне число в шістнадцятковому вигляді.

Для типу `long double` параметр `розмір` у відповідній керуючій послідовності набуває значення `l`.

Приклади розв'язання задач

Приклад 4.5. Скласти програму наближеного обчислення золотого перетину c, використовуючи:

а) границю

$$c = \lim_{n\to\infty} \frac{F_n}{F_{n-1}}, \text{ де } F_n \text{ – числа Фібоначчі;}$$

б) ланцюговий дріб

$$c = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \ddots}}.$$

Розв'язок

а) Розглянемо послідовність $c_n = F_n / F_{n-1}$, $n \geq 1$. Знайдемо рекурентне співвідношення для c_n. Очевидно, що $c_1 = 1$, далі для $n \geq 2$ отримаємо

$$c_n = \frac{F_n}{F_{n-1}} = \frac{F_{n-1} + F_{n-2}}{F_{n-1}} = 1 + \frac{1}{F_{n-1} / F_{n-2}} = 1 + \frac{1}{c_{n-1}}.$$

б) Розглянемо послідовність

$$c_n = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \ddots + \cfrac{1}{1}}},$$

що містить $n-1$ риску дробу. Очевидно, що для цієї послідовності рекурентне співвідношення буде таким, як у пункті а).

Під наближеним значенням золотого перетину будемо розуміти таке значення c_N, за якого

$$| c_N - c_{N-1} | \leq \varepsilon,$$

для деякого наперед заданого $\varepsilon > 0$.

Використаємо змінну c для обчислення поточного члену послідовності c_n і змінну c1, у якій запам'ятовуватимемо попередній член цієї послідовності. Тоді програма має вигляд

```
#include <iostream.h>
#include <math.h>
void main(){
  double const eps = 0.000001; // точність
  double c1, c;
  c = 1;
  do {
    c1 = c;
    c = 1 + 1 / c;
  } while ( fabs(c - c1) >= eps );
  cout << c << '\n';
}
```

Задачі для самостійної роботи

4.21. Скласти програму обчислення із заданою точністю границь послідовностей, утворених за законами:

а) $a_n = \dfrac{n}{\sqrt{n^2+1} + \sqrt{n^2-1}}$; б) $a_n = \left(1 - \dfrac{1}{2}\right) \cdot \left(1 - \dfrac{1}{3}\right) \cdot \ldots \cdot \left(1 - \dfrac{1}{n+1}\right)$;

в) $a_n = \left(1-\frac{1}{2!}\right)\cdot\left(1+\frac{1}{3!}\right)\cdot\ldots\cdot\left(1+\frac{(-1)^n}{(n+1)!}\right)$.

4.22. Для заданих $\varepsilon > 0$ і $x > 0$ скласти програму наближеного обчислення кореня k-го степеня $y(x) = \sqrt[k]{x}$ з числа x, використовуючи відповідні рекурентні співвідношення і границю $\lim\limits_{n\to\infty} x_n = \sqrt[k]{x}$.

а) корінь квадратний $y(x) = \sqrt{x}$, $k = 2$:

$$\begin{cases} x_0 = \dfrac{x}{2}, \\ x_n = \dfrac{1}{2}\left(x_{n-1} + \dfrac{x}{x_{n-1}}\right), n \ge 1; \end{cases}$$

б) корінь третього степеня $y(x) = \sqrt[3]{x}, k = 3$:

$$\begin{cases} x_0 = \dfrac{x}{3}, \\ x_n = \dfrac{1}{3}\left(2x_{n-1} + \dfrac{x}{x_{n-1}^2}\right), n \ge 1; \end{cases}$$

в) кореня четвертого степеня $y(x) = \sqrt[4]{x}, k = 4$:

$$\begin{cases} x_0 = \dfrac{x}{4}, \\ x_n = \dfrac{1}{4}\left(3x_{n-1} + \dfrac{x}{x_{n-1}^2}\right), n \ge 1; \end{cases}$$

г) кореня п'ятого степеня $y(x) = \sqrt[5]{x}, k = 5$:

$$\begin{cases} x_0 = \dfrac{x}{5}, \\ x_n = \dfrac{1}{5}\left(4x_{n-1} + \dfrac{x}{x_{n-1}^4}\right), n \ge 1. \end{cases}$$

Під наближеним з точністю $\varepsilon > 0$ значенням $y(x) = \sqrt[k]{x}$ вважати значення такого члена x_N послідовності, що

а) $|x_N - x_{N-1}| < \varepsilon$; б) $|x_N^k - x| < \varepsilon$.

Порівняти отримані результати зі значеннями відповідних функцій із бібліотеки `math.h`.

4.23. Розв'язати задачу **3.45** і порівняти отримані результати зі значеннями відповідних функцій із бібліотеки `math.h`.

4.24. Скласти програму наближеного обчислення із заданою точністю границі послідовності, заданої рекурентним співвідношенням

а) $a_0 = 0,\ a_i = \dfrac{a_{i-1}+1}{a_{i-1}+2},\ i \ge 1;$ б) $a_0 = 1,\ a_i = \dfrac{2-a_{i-1}^3}{5},\ i \ge 1.$

4.25. Задано дійсне число x. Послідовність $a_1 a_2, \ldots$ утворена за законом $a_1 = x$: далі для $i \ge 2$ виконано:

а) $a_i = \sqrt{\left|4 \cdot a_{i-1}^2 - 2x\right|};$ б) $a_i = \dfrac{16+x}{1+\left|a_{i-1}^3\right|};$

в) $a_i = 2 \cdot a_{i-1} + \dfrac{x}{4+a_{i-1}^2};$ г) $a_i = 3 + \dfrac{1}{2^i}\cos^2(a_{i-1} - x).$

Скласти програму для обчислення границі послідовності a_n із заданою точністю.

4.26. Задано рекурентні співвідношення

$$x_1 = a, y_1 = b,$$

$$x_i = \frac{1}{2}(x_{i-1} + y_{i-1}),\ y_i = \sqrt{x_{i-1} \cdot y_{i-1}},\ i \ge 2\ (a > b > 0).$$

Скласти програму для обчислення першого члена послідовності x_i такого, що $|x_i - y_i| < \varepsilon,\ (\varepsilon > 0)$.

4.27. Скласти програму наближеного обчислення числа π:

а) за формулою Грегорі

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots;$$

б) використовуючи добуток

$$\frac{2}{\pi} = \sqrt{\frac{1}{2}} \cdot \sqrt{\frac{1}{2} + \frac{1}{2} \cdot \sqrt{\frac{1}{2}}} \cdot \sqrt{\frac{1}{2} + \frac{1}{2} \cdot \sqrt{\frac{1}{2} + \frac{1}{2} \cdot \sqrt{\frac{1}{2}}}} \cdot \ldots;$$

г) за формулою Валліса

$$\frac{\pi}{2} = \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \cdot 8}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \cdot 9} \ldots.$$

Проаналізувати, який з методів дає кращий результат за меншу кількість ітерацій.

4.28. Дано n дійсних чисел. Скласти програму для знаходження того з них, що є найближчим до цілого числа.

4.29. Задано дійсні числа $x, y \ \ (x > 0, y > 1)$. Скласти програму обчислення цілого числа k (додатного, від'ємного або рівного нулю), що задовольняє умову $y^{k-1} \le x < y^k$.

4.30. Задано послідовність дійсних чисел $x_1, x_2, ..., x_n$ (n заздалегідь невідоме), за якою йде 0. Скласти програму для обчислення суми

$$y = nx_1 + (n-1)x_2 + ... + 2x_{n-1} + x_n.$$

4.31. Задано послідовність із n дійсних чисел $x_1, x_2, ..., x_n \ \ (n \ge 3)$. Скласти програму для обчислення:

а) $(x_1 + 2x_2 + x_3) \cdot (x_2 + 2x_3 + x_4) \cdot ... \cdot (x_{n-2} + 2x_{n-1} + x_n)$;

б) $(x_1 + x_2 + x_3) \cdot x_2 + ... + (x_{n-2} + x_{n-1} + x_n) \cdot x_{n-1}$.

4.32. Задано натуральне число n, дійсні числа $a_1, a_2, \ldots, a_n$. Скласти програму, що визначає в послідовності $a_1, a_2, \ldots, a_n$ кількість сусідств:

а) двох додатних чисел; б) двох чисел різних знаків.

4.33. Скласти програму обчислення суми

$$1 - \frac{1}{2} + \frac{1}{3} - ... + \frac{1}{9999} - \frac{1}{10000}$$

чотирма способами:

а) послідовно зліва направо;

б) послідовно зліва направо обчислюються окремо суми додатних і від'ємних доданків, далі ці суми складаються;

в) послідовно справа наліво;

г) послідовно справа наліво обчислюються окремо суми додатних і від'ємних доданків, далі ці суми складаються.

Порівняти результат із даним значенням суми (30 десяткових знаків):

$$0.693097183059945296917232371458...$$

Який спосіб дає найменшу похибку обчислень, чому?

4.3. Символьний тип даних

Серед задач, що розв'язують за допомогою комп'ютера, особливе місце належить задачам обробки текстів. До числа простих типів, розглянутих вище, додамо **символьний тип** даних, до якого входять латинські букви, арабські цифри, спеціальні математичні та інші символи, розділові знаки, а також, можливо, й букви кирилиці. Позначається символьний тип у С за допомогою ключового слова `char`:

```
char a, b;
```

Під змінну типу `char` у С відводиться 1 байт, тому зрозуміло, що за допомогою типу `char` можна закодувати 256 різних символів. Бієктивне

відображення натуральних чисел від 0 до 255 у множину символів називається **таблицею кодування** символів, а число, що відповідає конкретному символу, називається **кодом символу**. Під терміном **символьна константа** розумітимемо конкретний символ із таблиці кодування. Таблиць кодування існує багато, проте для всіх них існують спільні правила їхнього створення, наприклад,

- літери розташовуються у таблиці кодування в алфавітному порядку;
- арабські цифри розташовуються у таблиці кодування в порядку зростання.

Щоб відрізняти символьні константи від імен змінних, цифр та інших символьних знаків, символьні константи оточуються апострофами, наприклад,

```
a = 'a'; b1 = '1'; b2 = ';';
```

У С символьні константи, які неможливо задати явно (напр., символ нового рядка, символ апострофа), можна задавати їхніми кодами у системі числення за основою 8 (або 16). Для цього в апострофах записують \ і вісімковий код символу. Наприклад, записують `'\0'` для символу з кодом 0 (або `'\x0'`, де x вказує на те, що код задано у системі числення за основою 16). Крім цього, існує перелік так званих спеціальних символів, серед яких найуживаніші:

`'\n'` – символ нового рядка;
`'\t'` – символ табуляції;
`'\\'` – символ \ (back slash);
`'\''` – символ апострофа.

Тип `char` у С належить до цілих типів. Це означає, що у змінній символьного типу міститься число, що є кодом деякого символу. Таким чином, для даних типу `char` визначено всі операції і відношення, що й для цілих чисел.

Для простого введення/виведення символів користуються інструкціями (див. ст. 6) `cin/cout`. Для форматованого введення/виведення використовують інструкції (див. ст. 6) `scanf/printf` зі специфікатором символьного типу – `c`.

Для введення символів також використовують інструкції посимвольного зчитування з клавіатури: `getch` і `getchar`. Для використання інструкції `getch` потрібно підключити бібліотеку `conio.h`, а для використання інструкції `getchar` – `stdio.h`. Синтаксис використання обох інструкцій подібний.

Нехай змінна `c` типу `char`. Тоді кожна з інструкцій

```
c = getch();
c = getchar();
```

призупиняє роботу програми, доки з клавіатури не буде введено деякий символ, і присвоює його змінній `c`. Різниця між ними полягає у двох моментах:

1) інструкція getch зчитує з клавіатури символ одразу після натискання деякої клавіші, а інструкція getchar – після натискання клавіші очікує натиснення клавіші Enter;

2) на відміну від інструкції getchar, інструкція getch зчитаний символ на екран не виводить.

Для символів визначено функції:

- булеві:
  - isalnum(c) – визначає, чи є символ c літерою або цифрою;
  - isalpha(c) – визначає, чи є символ c літерою;
  - isdigit(c) – визначає, чи є символ c цифрою;
  - islower(c) – визначає, чи є символ c маленькою літерою;
  - isupper(c) – визначає, чи є символ c великою літерою;
- переведення літер до верхнього (нижнього) регістра:
  - tolower(c) – повертає маленьку літеру, що відповідає літері c;
  - toupper(c) – повертає велику літеру, що відповідає літері c;

Зауваження. Вищенаведені функції працюють вийнятково для літер латинського алфавіту. Для використання цих функцій треба підключити бібліотеку ctype.h.

Приклади розв'язання задач

Приклад 4.6. Скласти програму виведення великих латинських літер в алфавітному порядку разом із їхніми кодами.

Розв'язок. Оскільки символьний тип упорядкований, то для виведення всіх великих латинських літер використаємо цикл із лічильником. У нашому випадку змінна c у циклі пробігатиме всі літери від 'A' до 'Z'. Зауважимо, що, оскільки char є цілочисловим типом, то у змінній c буде міститися число. Але інструкція cout виводить на екран саме символ, а не його код. Тому в програмі для виведення кодів символів використаємо операцію перетворення типу char на тип int.

```
#include <iostream.h>
void main(){
  char c;
  for (c = 'A'; c <= 'Z'; c++)
    cout << c << " <-> " << int(c) << '\n';
}
```

Якщо використати інструкцію форматованого виведення printf, то програма набуде ще простішого вигляду:

```
#include <stdio.h>
void main(){
  for (char c = 'A'; c <= 'Z'; c++)
    printf("%c <-> %u\n", c, c);
```

```
}
```

Приклад 4.7. Визначити, чи є символ латинською літерою (великою або маленькою), цифрою або ні тим, ні іншим.

Розв'язок. Цю задачу легко розв'язати, використовуючи стандартні функції із бібліотеки `ctype.h`. Проте для глибшого розуміння роботи з символами напишемо програму без їхнього використання. Скористаємося тим, що літери у таблиці кодування розташовані в алфавітному порядку, а цифри – у порядку зростання. Для введення символу з клавіатури використаємо інструкцію `getch()`.

```
#include <iostream.h>
#include <conio.h>
void main(){
  char c;
  c = getch(); // зчитування символу с клавіатури
  cout << c;   // виведення зчитаного символу
  if (c >= 'a' && c <= 'z')
    cout << " – маленька літера" << '\n';
  else if (c >= 'A' && c <= 'Z')
    cout << " – велика літера" << '\n';
  else if (c >= '0' && c <= '9')
    cout << " - цифра" << '\n';
  else
    cout << " – ні літера, ні цифра" << '\n';
}
```

Приклад 4.8. Написати програму, що переводить маленькі латинські літери у відповідні великі (у верхній регістр).

Розв'язок. Розв'яжемо задачу, не використовуючи функцію `toupper(c)`. Аналогічно до попереднього прикладу скористаємося властивостями таблиці кодування. Помітимо, що різниця кодів між символами маленької і великої літери однакова та становить `'a' – 'A'`. Тоді програма набуде такого вигляду:

```
#include <iostream.h>
#include <conio.h>
void main(){
  char c;
  c = getch(); cout << c << '\n';
  if (c >= 'a' && c <= 'z')
    c = c - ('a' - 'A');
  cout << c << '\n' ;
  getch();
}
```

Приклад 4.9. Перевірити, чи правильно в заданому тексті розставлено круглі дужки (тобто, чи стоїть праворуч від кожної відкритої дужки

відповідна до неї закрита дужка, а ліворуч від кожної закритої – відповідна відкрита). Відповідь – "так" або "ні".

Розв'язок. Під терміном "текст" розумітимемо непорожню послідовність символів, що закінчується крапкою. Використаємо змінну k для підрахунку різниці кількості відкритих і закритих дужок. Додатково використаємо змінну r, що набуватиме значення 0 (*Хибність*), якщо закритій дужці не відповідає жодна із відкритих дужок, що їй передують. Наприклад, r = 0, якщо текст матиме вигляд

```
a * ( b + c ) ) * ( d
```

хоча при цьому k = 0, бо кількість відкритих і закритих дужок однакова.

```
#include <iostream.h>
#include <conio.h>
void main(){
  char c;
  int k = 0, r = 1;
  do {
    c = getch();
    cout << c;
    if (c == '(') k++;
    if (c == ')') k--;
    if (k < 0)    r = 0;
    /* Введення символів відбувається доти, доки
    не буде введено символ '.' або закриту дужку
    ')', якій не відповідає відкрита '('      */
  } while (c != '.' && r);
  if ( k == 0 && r )
    cout << "Дужки розставлені правильно\n";
  else
    cout << "Дужки розставлені неправильно\n";
}
```

Приклад 4.10. Надрукувати заданий текст, виключивши з нього всі знаки арифметичних операцій.

Розв'язок. Скористаємось тим, що при введенні з клавіатури кількох символів за допомогою інструкції cin, символи ставляться у чергу для обробки. Кількість символів у черзі не може перевищувати 128. Цей факт дає змогу обробляти текст посимвольно, попередньо повністю ввівши його з клавіатури.

```
#include <iostream.h>
void main(){
  char c;
  do {
    cin >> c;
    if (c!='+' && c!='-' && c!='*' && c!='/')
      cout << c;
  } while (c != '.');
```

```
  cout << '\n';
}
```

Приклад 4.11. Використовуючи тільки символьне введення, ввести непорожню послідовність цифр, перед якою може стояти знак плюс (`'+'`) чи мінус (`'-'`), і за якою стоїть крапка; отримавши відповідне ціле число, присвоїти його цілій змінній `m`.

Розв'язок. Легко побачити, що для того, щоб за символом-цифрою `c` отримати відповідну цифру, досить скористатись виразом `c - '0'`. Використаємо змінну `k`, у якій буде міститися знак числа.

```
#include <iostream.h>
#include <conio.h>
void main(){
  char c;
  int k = 1, m = 0;
  do {
    c = getch(); // зчитування символа
    cout << c;   // виведення зчитаного символа
    if (c == '-') k =-1;
    if (c >= '0' && c <= '9')
      m = m * 10 + c - '0'; // дописування цифри
  } while (c != '.');
  if (k == -1) m = m * k;
  cout << '\n' << m <<'\n';
}
```

Вищенаведена програма незахищена від неправильного введення даних. Напишемо програму, що реагуватиме на введення некоректних даних. Для цього використаємо змінну `err`, що набуватиме значення 1 (*Істина*) у випадку, якщо введений текст не є записом цілого числа.

```
#include <iostream.h>
#include <conio.h>
void main(){
  char c;
  int k = 0, m = 0, err = 0;
  do {
    c = getch();
    cout << c;
    if (c == '.') break;
    if (c == '+')
      if  (k == 0) k = 1;
      else err = 1;
    else if (c == '-')
      if  (k == 0) k =-1;
      else err = 1;
```

```
      else if ( c >='0' && c <='9')
        m = m * 10 + int(c - '0');
      else err = 1;
    } while (1);
    if (k == -1) m = m * k;
    if (err)
      cout << "Error\n";
    else
      cout << m << '\n';
}
```

Приклад 4.12. До заданого тексту входять тільки цифри та літери. Визначити, чи правда, що сума числових значень цифр, які входять до тексту, дорівнює довжині тексту.

Розв'язок. Використаємо змінну m для підрахунку суми цифр і змінну k – для визначення довжини тексту.

```
#include <iostream.h>
#include <conio.h>
void main(){
  char c;
  int k = 0, m = 0;
  do {
    c = getch();
    cout << c;
    if (c == '.') break;
    k++;
    if ( c >= '0' && c <= '9')
      m = m + int(c - '0');
  } while (1);
  if (k == m)
    cout << "Так\n";
  else
    cout << "Ні\n";
}
```

Задачі для самостійної роботи

4.34. Вивести на екран таблицю символів:

```
Aa Bb Cc Dd Ee Ff Gg Hh
Ii Jj Kk Ll Mm Nn Oo Pp
Qq Rr Ss Tt Uu Vv Ww Xx
```

Зауваження. У цьому розділі під "текстом" потрібно розуміти непорожню послідовність символів $c_1,\ldots,c_n$ (n заздалегідь невідоме), що передують символу '.' (до тексту крапка не входить).

4.35. Визначити, яка з двох заданих літер у даному тексті трапляється частіше.

4.36. Визначити, чи входить до даного тексту кожна з літер слова `"key"`.

4.37. Надрукувати заданий текст:

а) виключивши з нього всі цифри й подвоївши знаки `'+'` та `'–'`;

б) виключивши з нього всі знаки `'+'`, безпосередньо за якими стоїть цифра;

в) виключивши з нього всі літери *a*, безпосередньо перед якими стоїть літера *c*;

г) замінивши в ньому всі пари `"ph"` на літеру `'f'`;

ґ) виключивши з нього всі зайві пропуски (тобто з кількох, що йдуть підряд, залишити один).

4.38. Визначити, чи є заданий текст правильним записом цілого числа (можливо, зі знаком).

4.39. Дано текст, серед символів якого є принаймні одна кома. Знайти номер:

а) першої по порядку коми; б) останньої по порядку коми.

4.40. Виключити із заданого тексту групи символів, які стоять між `'('` та `')'`. Самі дужки теж мають бути виключені. Вважається, що дужки розставлено правильно (парами) та всередині кожної пари дужок немає інших дужок.

4.41. Дано текст, серед символів якого міститься двокрапка `':'`. Отримати всі символи, розміщені:

а) до першої двокрапки включно;

б) після першої двокрапки;

в) між першою та другою двокрапками. Якщо другої двокрапки немає, то отримати всі символи, розташовані після єдиної двокрапки.

4.42. Задано непорожню послідовність непорожніх слів із латинських літер. Словами називаються групи символів, розділені одним чи кількома пропусками, що не містять пропусків усередині. Визначити кількість слів, які:

а) містяться в цій послідовності;

б) починаються із заданої літери;

в) закінчуються заданою літерою;

г) починаються та закінчуються однією літерою;

ґ) містять принаймні одну задану літеру;

д) містять рівно три задані літери.

4.43. За умов попередньої задачі:

а) знайти довжину найкоротшого слова;

б) підрахувати кількість входжень заданої літери до останнього слова даної послідовності.

4.44. Заданий текст надрукувати рядками. Розуміти під рядком або наступні 60 символів, якщо серед них немає коми, або частину тексту до коми включно.

4.45. Використовуючи тільки символьне виведення, вивести на друк значення цілої змінної k (знак '+' не друкувати).

4.46. Використовуючи тільки символьне введення, ввести задане дійсне число зі знаком, записане у форматі з фіксованою крапкою, за яким стоїть символ '?'. Присвоїти його дійсній змінній x.

4.47. Використовуючи тільки символьне виведення, надрукувати дійсне число x у такій формі:

$$\pm 0.d_1d_2...d_9E \pm p_1p_2,$$

де d_i, p_j – цифри, причому $d_1 \neq 0$, якщо $x \neq 0$.

4.48. Задано послідовність символів вигляду

$$d_1 \pm d_2 \pm ... \pm d_n$$

(d_1 – цифри, $n > 1$), за якою стоїть крапка. Обчислити значення цієї алгебраїчної суми.

4.49. Задане натуральне число n. Надрукувати у трійковій системі числення цілі числа від 0 до n.

4.50. До заданого тексту входять тільки цифри та літери. Визначити, чи задовольняє він такі властивості:

а) текст є десятковим записом числа, кратного 9 (6, 4);

б) текст починається з деякої ненульової цифри, за якою стоять тільки літери, і їхня кількість дорівнює числовому значенню цієї цифри;

в) текст містить (крім літер) тільки одну цифру, причому її числове значення дорівнює довжині тексту;

г) сума парних числових значень цифр, які входять до тексте, дорівнює довжині тексту;

ґ) текст збігається з початковим (кінцевим, будь-яким) відрізком ряду 0123456789;

д) текст складається тільки з цифр, причому їхні числові значення складають арифметичну прогресію (напр., 3 5 7 9, 8 5 2, 2).

4.51. Знайти у даному тексті символ та довжину найдовшої послідовності однакових символів, що йдуть поспіль.

4.4. Тип даних перерахування. Оператор вибору

Структура символьного типу узагальнюється на випадок довільної скінченної множини значень. Таке узагальнення називатимемо **типом**

даних перерахування. Синтаксис опису нового типу, що має ім'я T і включає константи перерахування c_1, c_2, ..., c_n, такий:

```
typedef enum {
  c1, c2,..., cn
} T;
```

Приклад

```
typedef enum {
  plus, minus, multi, divide
} operator;
typedef enum {
  mon, tue, wed, thu, fri, sat, sun
} week;
```

Для цього типу виконуватимуться ті самі властивості, що й для символьного типу. Таким чином, змінні типу перерахування є цілими змінними. Для констант перерахування визначено номер (код), отже й порядок. Зазвичай нумерація констант починається з 0 і змінюється з кроком 1. Проте у C є можливість визначати довільні значення всіх або деяких констант перерахування, наприклад,

```
typedef enum {
  plus=1, minus, multi=4, divide=8
} operator;
```

Із типом даних перерахування зручно працювати, використовуючи оператор вибору, синтаксис якого

```
switch (x) {
  case a1 :  P1; break;
  ...
  case an :  Pn; break;
  default : Q;
};
```

де x – арифметичний вираз, що набуває значень типу T, a_1, ..., a_n – константи типу T, P_1, ..., P_n, Q – набори операторів.

Оператор вибору виконується в два етапи:

1) обчислюється значення виразу x, який називають **селектором**;

2) значення селектора порівнюється з константами a_i, i<=n. Якщо деяка константа вибору a_i дорівнює поточному значенню селектора x, то виконується інструкція P_i. Якщо жодна з констант a_i не дорівнює значенню селектора, то виконується інструкція Q. Після цього відбувається перехід до наступної команди, що йде після оператора вибору.

Зауважимо, що оператор вибору може не мати блоку `default`.

Якщо для кількох констант вибору потрібно виконувати одну й ту саму інструкцію, то константи можна групувати. Наприклад, якщо для констант a, b і c потрібно виконати інструкцію P, а для констант d та e – Q, то оператор вибору можна оформити так

```
switch (x) {
  case a :
  case b :
  case c : P; break;
  case d :
  case e : Q; break;
};
```

Приклади розв'язання задач

Приклад 4.13. Написати програму виведення назв днів тижня.

Розв'язок. У програмі визначимо новий тип week, що містить перелік усіх днів тижня. Для виведення необхідного дня тижня на екран скористаємось оператором вибору.

```
#include <iostream.h>
typedef enum {
  mon, tue, wed, thu, fri, sat, sun
} week;
void main(){
  week d;
  for (d = mon; d <= sun; d++) {
     cout << d;
     switch (d) {
       case mon: cout << "Понеділок\n"; break;
       case tue: cout << "Вівторок\n"; break;
       case wed: cout << "Середа\n"; break;
       case thu: cout << "Четвер\n"; break;
       case fri: cout << "П'ятниця\n"; break;
       case sat: cout << "Субота\n"; break;
       case sun: cout << "Неділя\n"; break;
     }
  }
}
```

Приклад 4.14. За назвою місяця визначити сезон (пору року), на який цей місяць припадає.

Розв'язок. У програмі визначимо два типи month і season. Тип month – перелік місяців року, season – пір року. Для визначення пори року, на яку припадає введений з клавіатури місяць, використаємо оператор вибору:

```
#include<iostream.h>
typedef enum {
  Jen = 1, Feb, Mart, Apr, May, Jun,
  Jul, Aug,  Sep,  Oct , Nov, Dec
} month;
typedef enum {
```

```
    winter, spring, summer, october
  } season;
  void main()
  { int n;
    month m;
    season s;
    cout << "Номер місяця =? ";
    cin >> n;          // Введення номера місяця
    // Перетворення номера місяця до типу month
    m = month(n);
    switch (m) {
         case Jen :
         case Feb :
         case Dec : s = winter;  break;
         case Mart:
         case Apr :
         case May : s = spring;  break;
         case Jun :
         case Jul :
         case Aug : s = summer;  break;
         case Sep :
         case Oct :
         case Nov : s = october; break;
    }
    switch (s) {
         case winter : cout << "winter";  break;
         case spring : cout << "spring";  break;
         case summer : cout << "summer";  break;
         case october: cout << "october"; break;
    }
  }
```

Приклад 4.15. Для натурального числа k від 1 до 99 надрукувати фразу "Мені k років", ураховуючи при цьому, що за деяких значень k слово "років" треба замінити на слово "рік" або "роки".

Розв'язок

```
  #include<iostream.h>
  void main(){
    int k;
    cout << "Скільки вам років"; cin >> k;
    cout << "Мені " << k << " ";
    if (k >= 11 && k<= 19) cout << "років";
    else switch (k % 10) {
         case 0:
         case 5:
```

```
        case 6:
        case 7:
        case 8:
        case 9: cout << "років"; break;
        case 1: cout << "рік";   break;
        case 2:
        case 3:
        case 4: cout << "роки";  break;
    }
    cout << '\n';
}
```

Приклад 4.16. Визначити день тижня за заданою датою.

Розв'язок. Для розв'язання задачі опишемо новий тип weekday – дні тижня. У програмі використаємо, що 1 січня 1920 року припадає на четвер.

```
#include <iostream.h>
typedef enum {
  mon, tue, wed, thu, fri, sat, sun
} weekday;
void main(){
  weekday wd;
  int y, m, d, disp, kv, day, nd, i;
  // Введення дати з клавіатури
  do {
     cout << "Рік [1920..2099]=? ";
     cin >> y;
  } while (y < 1920 || y > 2099);
  do {
     cout << "Місяць [1..12]=? ";
     cin >> m;
  } while (m < 1 || m > 12);
  // Визначення кількості днів у заданому місяці
  switch (m) {
    case 2:    nd = (y % 4 == 0 ? 29 : 28); break;
    case 4:
    case 6:
    case 9:
    case 11:   nd = 30; break;
    default:   nd = 31;
  }
  do {
    cout << "Число [1.." << nd << "]=?";
    cin >> d;
  } while (d < 1 || d > nd);
```

```
  // Визначення к-ті високосних років після 1920
  kv = (y == 1920 ? 0 : ((y - 1) - 1920) / 4 + 1);
  day = d;
  for (i = 1; i < m; i++)
  switch(i){
    case 2: day += (y % 4 == 0 ? 29 : 28); break;
    case 4:
    case 6:
    case 9:
    case 11: day += 30; break;
    default: day += 31;
  }
  /* Визначення кількості днів, що минуло з
     1 січня 1920 року до заданої дати      */
  disp = ((y - 1920) + kv + (day - 1)) % 7;
  wd = thu;  // 01.01.1920 – четвер
  // Визначення дня тижня за заданою датою
  for (i = 1; i <= disp; i++)
    if (wd == sun) wd = mon;
    else wd++;
  switch (wd){
    case mon: cout << "Понеділок\n"; break;
    case tue: cout << "Вівторок\n";  break;
    case wed: cout << "Середа\n";    break;
    case thu: cout << "Четвер\n" ;   break;
    case fri: cout << "П\'ятниця\n"; break;
    case sat: cout << "Субота\n";    break;
    case sun: cout << "Неділя\n";    break;
  }
}
```

Задачі для самостійної роботи

4.52. Скласти програму виведення назв:
а) місяців у році; б) кольорів спектру;
в) шахових фігур; г) років за східним календарем;
ґ) знаків зодіаку.

4.53. За назвою країни визначити назву її столиці.
Указівка. Розглянути типи:

```
typedef enum {
  Austria, Bulgaria, Italy, Ukraine, France
} country;
typedef enum {
  Vienna, Sofia, Rome, Kyiv, Paris
```

```
} capital;
```

4.54. За українською назвою мови програмування визначити її англійську назву.

4.55. За назвою країни визначити назву континенту, на якому вона розташована.

4.56. Довжину відрізка, задану в міліметрах, сантиметрах, дециметрах чи кілометрах, замінити на величину цієї довжини в метрах.

4.57. Надрукувати слово із заданого переліку: *стіл*, *біль*, *зошит*, *двері* у заданому відмінку й однині.

4.58. Корабель ішов за деяким курсом, потім його курс було змінено, згідно з наказом. Визначити новий курс корабля.

Указівка. Розглянути типи

```
typedef enum {
  north, east, south, west
} course;                          // Курс
typedef enum {
  forward, right, left, backward
} order;                           // Наказ
```

4.59. За літерою-цифрою d визначити назву цієї цифри.

4.60. Для натурального числа k надрукувати фразу "Ми знайшли k грибів у лісі", узгодивши закінчення слова "гриб" із числом k.

4.61. Визначити кількість днів у заданому місяці.

4.62. Визначити дату наступного дня.

4.63. Визначити дату k-го за рахунком дня високосного року.

4.64. Визначити порядковий номер того дня високосного року, який має задану дату.

Розділ 5
ПІДПРОГРАМИ

У С програма складається з глобальних описів і підпрограм. Серед останніх головне місце посідає підпрограма на ім'я `main()`. Виконання програми завжди починається з виконання цієї підпрограми.

Зауваження. Часто у програмуванні підпрограми розділяють на дві групи: **процедури** та **функції**. Усі підпрограми у С є функціями. Це означає, що кожна підпрограма по завершенню свого виконання повертає деяке значення.

Загальний вигляд опису функції у С:

```
t f(t1 x1, ..., tn xn){
   P;
}
```

де `f` – ім'я функції, `t` – тип результату, x_1, ..., x_n – параметри, t_1, ..., t_n – відповідно їхні типи, P – тіло функції. Рядок

```
t f(t1 x1, ..., tn xn){
```

називається **заголовком** підпрограми.

Для повернення результату функцією тіло P має містити інструкцію

```
return e;
```

де `e` – вираз типу `t`. Цей оператор завершує виконання функції і повертає значення `e` у місце її виклику.

Зауваження. Процедури у С розглядаються як функції, що повертають результат порожнього типу. У такому випадку тип функції позначається ідентифікатором `void` або взагалі не вказується.

Виклик функції у мові С може мати вигляд:

```
z = f(e1, ..., en);
```

або

```
f(e1, ..., en);
```

де e_1, ..., e_n – вирази типів t_1, ..., t_n. У другому випадку результат функції ігнорується. Такий виклик застосовується зазвичай для функцій типу `void`.

Параметри використовуються для контрольованого передавання даних у підпрограму. Параметри, описані в заголовку та тілі підпрограми, називають **формальними**, а ті, що вказуються під час виклику

підпрограми, – **фактичними**. Отже, у наведених описах параметри x_1, ..., x_n – формальні, e_1, ..., e_n – фактичні. Під час виклику підпрограми фактичні параметри, вказані у команді виклику, стають значеннями відповідних формальних параметрів, чим і забезпечується передавання даних у підпрограму.

Означення 5.1. Передавання параметрів у підпрограму, при якому формальному параметру присвоюється значення фактичного параметра, називається **передаванням параметрів за значенням**.

У мові С передавання параметрів реалізовано тільки як передавання за значенням. Слід звернути увагу на те, що при передаванні за значенням формальний параметр міститиме копію значення, що є у фактичному. Тому всі операції, що проводитимуться з формальними параметрами, не вплинуть на фактичні. Іншими словами, якщо за фактичний параметр буде використовуватися змінна, а всередині підпрограми значення відповідного формального параметра зміниться, то фактичний параметр не зміниться. Якщо ж потрібно, щоб підпрограма змінювала значення фактичних параметрів, то використовують механізм **вказівників**.

Вказівник задає посилання на деяку змінну, або, як ще кажуть, адресу змінної. Опис вказівника p на змінну типу t у С виглядає так:

```
t *p;
```

Для отримання змінної, на яку посилається вказівник, використовують операцію розіменування вказівника *. Наприклад, інструкція

```
*pn = 1;
```

присвоює змінній, на яку вказує pn, значення 1.

Операція & повертає вказівник на змінну (адресу змінної). Наприклад, &x є вказівником на змінну x.

Таким чином, якщо потрібно описати підпрограму, що змінюватиме значення деяких змінних, відповідні формальні параметри описуються як вказівники (адреси цих змінних). Наприклад, підпрограма

```
void increment(int *pk){
  *pk = *pk + 1;
}
```

призначена для збільшення значення змінної, на яку показує вказівник на 1. Її виклик може бути таким:

```
increment(&k);
```

де k – деяка змінна цілого типу.

Означення 5.2. Рекурсія – це спосіб визначення підпрограми, за якого попередньо вказується один або кілька (звичайно простих) базових випадків; далі на їхній основі задається правило побудови підпрограми, що явно або неявно посилається на ці базові випадки. Іншими словами, рекурсія – це спосіб загального визначення підпрограми через себе із застосуванням раніше заданих часткових визначень. Рекурсія використовується, коли можна визначити самоподібність задачі.

Рекурсія буває **прямою** та **непрямою**. Під час прямої рекурсії виклик підпрограми здійснюється з неї самої. За непрямої рекурсії виклик підпрограми здійснюється через інші функції, наприклад, функція A викликає функцію B, а функція B – функцію A. Кількість вкладених викликів підпрограми називається **глибиною** рекурсії.

Прикладом простої рекурсії є функція для визначення факторіала натурального числа

$$n! = n \times (n-1)! \text{ при } n \geq 1,$$
$$n! = 1 \text{ при } n = 0.$$

Іншим прикладом рекурсії є визначення послідовності чисел Фібоначчі $\{f_n \mid n \geq 0\}$

$$f_n = f_{n-1} + f_{n-2} \text{ при } n \geq 2,$$
$$f_0 = f_1 = 1.$$

Приклади розв'язання задач

Приклад 5.1. Задано непорожню послідовність дійсних чисел. Серед цих чисел знайти максимальне за модулем. Описати функцію обчислення модуля дійсного числа.

Розв'язок. Програма для задачі міститиме дві підпрограми. Перша, що має ім'я `d_abs`, використовується для визначення модуля дійсного числа. Друга – `main` – головна підпрограма програми. Для визначення кількості членів у послідовності використаємо змінну `n`. Змінну `a` використовуватимемо для введення з клавіатури поточного члена послідовності, а у змінній `max` буде міститися найбільший за модулем член послідовності серед введених раніше.

```
#include <iostream.h>
// підпрограма для обчислення модуля
double d_abs(double d){
  if (d >= 0)
    return d;
  else
    return -d;
}
void main(){
  double a;
  int n, i = 0, max;
  cout << "Введіть кількість членів";
  cin >> n;
  cout << "Введіть 1-й член: ";
  cin >> max;
  for (i = 2; i<=n; i++){
    cout << "Введіть " << i << "-й член: ";
    cin >> a;
    if (d_abs(max) < d_abs(a)) max = a;
```

```
  }
  cout << max << '\n';
}
```

Приклад 5.2. Визначити, чи можна введене з клавіатури число представити у вигляді суми двох простих.

Розв'язок. Для реалізації програми опишемо булеву функцію `prime`, яка визначатиме, чи є число простим.

```
#include <iostream.h>
int prime(int a){
  int i;
  /* is_prime - булева змінна, що набуває
     значення 1, якщо число a - просте,
     і 0 - у протилежному випадку */
  int is_prime = 1;
  // ділимо число a на всі числа від 2 до a-1
  for (i = 2; i <= a-1; i++) {
    if (a % i == 0) {
      /* якщо число a ділиться на i без остачі,
         то a не є простим */
      is_prime = 0;
      break;
    }
  }
  return is_prime;
}
void main(){
  int n, i,
  /* flag - булева змінна, вказує, чи можна число
     представити у вигляді суми двох простих */
  int flag = 0;
  cout << "Введіть число: ";
  cin >> n;
  for (i = 1; i <= n/2; i++){
    if (prime(i) && prime(n-i)){
      flag = 1;
      break;
    }
  }
  if (flag)  {
    cout << n << " можна представити у вигляді "
    cout << "суми двох простих, наприклад: \n";
    cout << n << "=" << i << "+" << n-i << '\n';
  }
  else
    cout << n << " не можна представити у "
```

```
    cout << "вигляді суми двох простих\n";
}
```

Приклад 5.3. Написати підпрограму обміну значень двох дійсних змінних.

Розв'язок. Як ми знаємо, параметри підпрограм, що відіграють роль модифікованих параметрів, мають бути описані як вказівники. Тому в підпрограмі, яку назвемо swap, використаємо змінні px і py, що міститимуть адреси змінних типу double.

```
void swap(double *px, double *py){
  double z; // допоміжна змінна z
  /* за вказівником (адресою) px
     отримуємо значення змінної
     і записуємо його у змінну z. */
  z = *px;
  /* записуємо значення змінної, на яку
     посилається вказівник py, у змінну,
     на яку посилається вказівник px. */
  *px = *py;
  /* у змінну, на яку посилається вказівник py,
     записуємо значення змінної z. */
  *py = z;
}
```

Виклик підпрограми swap такий

```
swap(&x, &y);
```

У підпрограмі swap, px і py – вказівники, яким під час виклику присвоюються значення адрес змінних x та y,тому підпрограма працює не із цими змінними, а з їхніми адресами. При цьому підпрограма, маючи адреси змінних, звертається безпосередньо до пам'яті, у якій змінні x та y зберігають свої значення. Це й забезпечує повернення результату підпрограми.

Приклад 5.4. Створити рекурсивну підпрограму обчислення факторіала натурального числа.

Розв'язок. Як відомо, факторіал натурального числа n можна представити у рекурсивному вигляді

$$n! = n \times (n-1)! \text{ при } n \geq 1,$$

$$n! = 1 \text{ при } n = 0.$$

Тоді функція fact для обчислення факторіала матиме вигляд:

```
long int fact(int n){
  if (n == 0)
    return 1;
  else
    return n * fact(n-1);
```

```
}
```

Приклад 5.5. Визначити рекурсивну процедуру зображення натурального числа Z у вісімковій системі числення.

Розв'язок. Перетворення числа Z на вісімкову систему числення здійснюємо шляхом ділення його на 8 і виведення залишків від ділення у зворотній послідовності:

```
void Convert8(int Z){
  if (Z > 1)
    Convert8(Z / 8);
  cout << Z % 8;
}
```

Приклад 5.6. Визначити рекурсивну функцію для обчислення біноміального коефіцієнта C_n^m, $0 \le m \le n$.

Розв'язок. Біноміальні коефіцієнти C_n^m задовольняють рекурентну формулу

$$\begin{cases} C_n^0 = C_n^n = 1; \\ C_n^m = C_{n-1}^m + C_{n-1}^{m-1},\ 0 < m < n. \end{cases}$$

Тому рекурсивна підпрограма матиме вигляд

```
long int C(unsigned int n, unsigned int m){
  if (m == 0 || m == n)
      return 1;
  else
      return C(n-1, m)+C(n-1, m-1);
}
```

Ця підпрограма для великих значень аргументів вимагає досить значних ресурсів пам'яті та часу. Саме тому наведемо інший алгоритм обчислення біноміальних коефіцієнтів.

Відомо, що біноміальні коефіцієнти обчислюються за формулою

$$C_n^m = \frac{n!}{m!(n-m)!},\ 0 \le m \le n.$$

Тоді

$$C_n^m = \frac{n}{m} \cdot \frac{(n-1)!}{(m-1)!((n-1)-(m-1))!} = \frac{n}{m} C_{n-1}^{m-1}, 0 \le m \le n.$$

Отже, біноміальні коефіцієнти задовольняють рекурентну формулу

$$C_n^m = \begin{cases} 1, & m = 0, \\ \dfrac{n}{m} C_{n-1}^{m-1}, & 1 \le m \le n. \end{cases}$$

Таким чином, рекурсивна підпрограма матиме вигляд

```
long int C(unsigned int n, unsigned int m){
```

```
  if (m == 0)
      return 1;
  else
      return n * C(n-1, m-1) / m;
}
```

Приклад 5.7. Обчислити з точністю ε значення золотого перерізу c, використовуючи нескінченний ланцюговий дріб

$$c = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{\ddots}}}.$$

Розв'язок. Визначимо дискретну функцію

$$c(n) = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cdots + \cfrac{1}{1}}},$$

де $n-1$ – риска дробу. Цю функцію можна подати у такому рекурсивному вигляді:

$$\begin{cases} c(1) = 1, \\ c(n) = 1 + \dfrac{1}{c(n-1)}, n \geq 2. \end{cases}$$

Наближеним значенням золотого перетину називатимемо таке значення функції $c(N)$, що

$$| c(N) - C(N-1) | < \varepsilon.$$

Тоді програма матиме вигляд:

```
#include <iostream.h>
#include <math.h>
// Рекурсивна підпрограма знаходження c(n)
double c(int n){
  if (n == 1)
    return 1;
  else
    return 1 + 1 / c(n-1);
}
void main(){
  double const eps = 0.000001;    // точність
  double cur = 1 // поточне значення:   c(n)
  double prev;   // попереднє значення: c(n-1)
  int n = 1;
  do {
```

```
      n++;
      prev = cur;
      cur = c(n);
    } while ( fabs(cur - prev) >= eps );
    cout << cur << '\n';
}
```

Задачі для самостійної роботи

5.1. Створити програму для обчислення добутку

$$p = f_0 \cdot f_1 \cdot \ldots \cdot f_n,$$

де $f_l = \frac{1}{l^2+1} + \frac{1}{l^2+2} + \ldots + \frac{1}{l^2+l+1}$.

5.2. Два простих числа називаються "близнюками", якщо вони відрізняються одне від одного на 2 (напр., числа 41 та 43). Скласти програму виведення на друк усіх пар "близнюків" із відрізка $[n, 2n]$, де n – задане ціле число, більше за 2.

5.3. Дано натуральне число n і послідовність натуральних чисел $a_1, a_2, \ldots a_n$. Показати всі елементи послідовності, які є:

а) повними квадратами; б) степенями п'ятірки;

в) простими числами.

Указівка. Визначити відповідні функції для перевірки, чи є число повним квадратом, степенем п'ятірки, простим числом.

5.4. Дано натуральне число n. Для чисел від 1 до n визначити всі такі, що можна зобразити у вигляді суми двох повних квадратів. Описати функцію, яка перевіряє, чи є число повним квадратом.

5.5. Щасливим називають таке шестизначне число, у якого сума перших трьох дорівнює сумі останніх трьох. Знайти всі щасливі числа. Описати функцію для визначення суми цифр тризначного числа.

5.6. Дано парне число $n > 2$. Перевірити для нього гіпотезу Гольдбаха: кожне парне число $n > 2$ можна зобразити у вигляді суми двох простих чисел. Визначити функцію, яка перевіряє, чи є число простим.

5.7. Описати функцію *НСД*(a, b), що за алгоритмом Евкліда визначає найбільший спільний дільник двох натуральних чисел a, b. Визначити найбільший спільний дільник трьох чисел *НСД*(a, b, c), ураховуючи, що *НСД*(a, b, c) = *НСД*(*НСД*(a, b), c).

5.8. Скласти програму обчислення величини

$$\frac{\sqrt[3]{a} - \sqrt[5]{a^2+1}}{\sqrt{a} + \sqrt[4]{3+a}}$$

для заданого дійсного числа $a > 0$. Визначити функції обчислення коренів $y = \sqrt[k]{x}$ із заданою точністю (див. задачу **4.22**).

5.9. Визначити підпрограми для обчислення тригонометричних і гіперболічних функцій, використовуючи їхні розвинення в ряд Тейлора (див. задачу **3.45**).

5.10. Дано координати вершин трикутника й точки всередині його. Використовуючи функцію для обчислення площі трикутника через три його сторони, визначити відстань від цієї точки до найближчої сторони трикутника.

Указівка. Урахувати, що площа трикутника обчислюється також через основу й висоту.

5.11. Визначити рекурсивні підпрограми зображення натурального числа:

а) у двійковій системі числення; б) у вісімковій системі числення:

в) у системі числення з основою N ($N > 1$).

5.12. Визначити рекурсивну функцію обчислення $НСД(n,m)$ натуральних чисел, яка ґрунтується на співвідношенні $НСД(n,m) = НСД(m,r)$, де r – залишок від ділення n на m.

5.13. Визначити рекурсивну функцію обчислення степеня дійсного числа з цілим показником x^n, згідно з формулою

$$x^n = \begin{cases} 1, & n = 0, \\ \dfrac{1}{x^{|n|}}, & n < 0, \\ x \cdot x^{n-1}, & n > 0. \end{cases}$$

5.14. Вивести на екран перші n рядків трикутника Паскаля. Реалізувати рекурсивний і нерекурсивний варіанти.

5.15. Визначити рекурсивну функцію для обчислення числа Фібоначчі F_n для заданого натурального n (див. приклад 3.11).

5.16. Для перших n чисел послідовності чисел Фібоначчі F_n переконатися у справедливості співвідношень:

а) $F_n^2 + F_{n+1}^2 = F_{2n+1}$; б) $F_{2n} = F_{n+1}^2 - F_{n-1}^2$;

в) $F_{3n} = F_{n+1}^3 + F_n^3 - F_{n-1}^3$; г) $F_{n+m} = F_{n-1}F_m + F_nF_{m+1}$;

ґ) $F_{(k+1)n} = F_{n-1}F_{kn} + F_nF_{kn+1}$; д) $F_n = F_lF_{n-l+1} + F_{l-1}F_{n-l}$.

5.17. Для перших n чисел послідовності чисел Фібоначчі F_n переконатися у справедливості формули

$$F_{n+1} = \sum_{k=0}^{[n/2]} C_{n-k}^k,$$

де C_m^k – біноміальні коефіцієнти.

5.18. Для перших n чисел послідовності чисел Фібоначчі F_n переконатися у справедливості твердження: F_n може бути простим тільки для простих m (за винятком $m = 4$).

5.19. Визначити рекурсивні функції:

а) обчислення суми цифр; б) кількості цифр;

в) максимальної цифри; г) мінімальної цифри;

ґ) перевірки входження заданої цифри до натурального числа.

5.20. Дано послідовність натуральних чисел, що завершується числом 0 (вважати, що 0 до послідовності не входить). Визначити рекурсивні функції для знаходження:

а) максимального числа серед членів послідовності;

б) мінімального числа серед членів послідовності;

в) суми всіх членів послідовності.

5.21. Дано послідовність дійсних чисел, що завершується числом 0 (вважати, що 0 до послідовності не входить). Визначити рекурсивні функції для знаходження:

а) суми додатних; б) суми від'ємних;

в) найбільшого серед додатних;

г) найменшого за модулем серед від'ємних членів послідовності.

5.22. Дано перший член і різницю арифметичної прогресії. Визначити рекурсивні функції для знаходження:

а) n-го члена арифметичної прогресії;

б) суми n членів арифметичної прогресії.

5.23. Дано перший член і знаменник геометричної прогресії. Визначити рекурсивні функції для знаходження:

а) n-го члена геометричної прогресії;

б) суми n членів геометричної прогресії.

5.24. Задано натуральні числа a, c, m. Визначити рекурсивну функцію для обчислення $f(m)$ за формулою

$$f(m)=\begin{cases} m, & 0 \le m \le 9, \\ g(m)\cdot f(m-1-g(m))+m, & \text{в інших випадках,} \end{cases}$$

де $g(m)$ – залишок від ділення $a \cdot m + c$ на 10.

5.25. Обчислити з точністю ε значення числа $\sqrt{2}$ використовуючи нескінченний ланцюговий дріб

$$\sqrt{2}=1+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{\ddots}}}.$$

5.26. Обчислити з точністю ε значення основи натурального логарифма e, використовуючи нескінченний ланцюговий дріб

$$e-1=1+\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{4+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{6+\ddots}}}}}}}}}.$$

Розділ 6
СКЛАДЕНІ ТИПИ ДАНИХ

6.1. Масиви

Означення 6.1. Упорядкований набір однотипних даних, об'єднаних під спільним ім'ям, називається **масивом**.

Характерні особливості масиву:

1) усі його компоненти одного типу;

2) кожний елемент масиву може бути позначений явно, і до нього є прямий доступ;

3) кількість компонент масиву визначається при його описі та в подальшому не змінюється.

На практиці найчастіше використовують одномірні масиви, доступ до елементів в яких здійснюється за допомогою номера елемента – **індексу**. Одномірні масиви у мові С описують як

```
t a[n];
```

де `t` – тип компонент масиву, `a` – ім'я масиву, `n` – кількість елементів масиву.

Нумерація елементів масиву в С завжди починається з 0. Отже, індекс змінюється від 0 до `n-1`.

У багатомірних масивах доступ до елементів масиву здійснюється за допомогою кількох індексів. Для опису багатомірного масиву кількість елементів за кожним індексом указують у дужках $[n_i]$:

$t\ a[n_1][n_2]\ldots[n_k];$

Елементи масиву розташовуються в пам'яті послідовно. Компоненти з меншими значеннями індексу зберігаються в більш низьких адресах пам'яті. Багатовимірні масиви розташовуються так, що найправіший індекс зростає першим.

Дій над масивами як цілісними структурами не передбачено. Обробка масивів проводиться поелементно. Після опису масиву до кожного його елемента можна звернутися, указавши ідентифікатор (ім'я) масиву та індекс елемента в квадратних дужках. Наприклад, запис `a[5]` дозволяє звернутися до шостого елемента масиву `a`. При роботі з двомір-

ним масивом указуються два індекси, наприклад `a[2][4]`, із n-вимірним масивом – n індексів.

Індексовані елементи масиву називаються індексованими змінними та можуть бути використані так само, як і прості змінні.

У С масиви є вказівниками на початок елементів масиву в пам'яті. Отже, для передавання масивів як аргументів, модифікованих параметрів або результатів достатньо вказати ім'я масиву та квадратні дужки, наприклад, `a[]`. Для багатовимірних масивів треба явно вказувати розміри (кількість елементів) для всіх індексів, крім першого, наприклад, `a[][5]`.

Для визначення кількості елементів у масиві зручно використовувати макрозаміну, що має синтаксис:

```
#define MacroName Expression
```

де `MacroName` – ім'я макрозаміни, `Expression` – код макрозаміни. Наприклад, якщо в код програми буде включено рядок

```
#define N 10
```

то скрізь у програмі компілятор замінить ідентифікатор N на число 10.

Приклади розв'язання задач

Приклад 6.1. Знайти суму елементів дійсного вектора.

Розв'язок. Оскільки вектор – це впорядкований набір однотипних даних, то задачу зручно розв'язувати, використовуючи одномірні масиви. Для роботи з масивами у С не визначено жодних операцій, тому в нашій програмі опишемо дві додаткові функції. Перша – `ReadVector` – буде використовуватися для зчитування вектора з клавіатури, друга – `sum` – для обчислення суми елементів дійсного вектора. Масив – це впорядкований набір даних, тому для поелементної роботи з масивом зручно використовувати цикли з лічильником, у яких змінна-лічильник пробігає всі можливі індекси масиву.

Таким чином, програма матиме вигляд

```
#include <iostream.h>
#define N 10
// функція для зчитування масиву з клавіатури
void ReadVector(double a[]){
  int i;
  // лічильник циклу i використовуємо
  // у якості індексу масиву для послідовного
  // зчитування всіх елементів масиву.
  for (i = 0; i <= N - 1; i++)
    cin >> a[i];
}
// функція для підрахунку суми елементів масиву
```

```
double sum(double a[]){
  double s;
  unsigned k;
  s = 0;
  for (k = 0; k < N; k++)
    s += a[k];
  return s;
}
void main(){
  // оголошення масиву a з N елементів.
  double a[N];
  // зчитуємо елементи з клавіатури у масив a.
  ReadVector(a);
  // виводимо суму елементів масиву a на екран.
  cout << sum(a);
}
```

Приклад 6.2. Визначити підпрограму пошуку найменшого серед найбільших елементів рядків квадратної дійсної матриці порядку n.

Розв'язок. Оформимо підпрограму у вигляді функції, аргументом якої є дійсна матриця. При цьому використаємо змінні `min` і `max`. У змінній `max` зберігатиметься максимальне серед чисел поточного рядка, а у змінній `min` – мінімальне серед максимальних.

```
float Min_Max(float A[][n]){
  float min,max;
  int i,j;
  for (i = 0; i <= n-1; i++){
    max = A[i][0];
    for (j = 1; j <= n-1; j++){
      if (max < A[i][j])
        max = A[i][j];
    }
    // змінну min ініціалізуємо як максимальний
    // елемент першого рядка матриці.
    if (i == 0)
      min = max;
    else if (min > max)
      min = max;
  }
  return min;
}
```

Приклад 6.3. Визначити функцію перевірки матриці на симетричність.

Розв'язок. Умова симетричності квадратної матриці має вигляд: $a_{ij}=a_{ji}$ для всіх i, j. Тому задачу можна розглядати як задачу пошуку таких i, j,

що $a_{ij} \neq a_{ji}$. Якщо таких індексів немає, то матриця симетрична. У підпрограмі використаємо змінну `res`, яка міститиме `1` (*Істину*), якщо не існує таких i, j, що $a_{ij} \neq a_{ji}$ і `0` (*Хибність*), якщо такі i, j буде знайдено.

```
int Is_Sym(float A[][n]){
  int res = 1; // припустимо, що матриця симетрична
  int i, j;
  for (i = 0; (i <= n-1) && res; i++)
    for (j = 0; (j <= n-1) && res; j++)
      if (A[i][j] != A[j][i])
        res = 0;
  return res;
}
```

Таким чином, булева підпрограма `Is_Sym` повертає `1` (*Істину*), якщо матриця симетрична, і `0` (*Хибність*) – у протилежному випадку.

Приклад 6.4. Створити програму обчислення абсолютної величини визначника квадратної матриці.

Розв'язок. Зведемо матрицю до трикутного вигляду. Нагадаємо, що при перестановці двох сусідніх рядків знак визначника змінюється на протилежний. Проте, оскільки у задачі вимагається обчислити абсолютну величину визначника, то при перестановці двох рядків зміну знака ігноруватимемо.

Зауважимо, що якщо в i-му рядку $a_{ii} = 0$, то шукатимемо такий рядок j, в якому $a_{ji} \neq 0$. Якщо такий рядок відсутній, то визначник дорівнює нулю; в іншому разі міняємо місцями i-й та j-й рядки.

```
#include<iostream.h>
#include<math.h>
#define n 3
// Підпрограма зчитування матриці з клавіатури
void ReadMatrix(float A[][n]){
  int i, j;
  for (i = 0; i <= n-1; i++)
    for (j = 0; j <= n-1; j++)
     cin >> A[i][j];
}
// Підпрограма перестановки i-го і j-го
// рядків матриці A
void SwapRow(float A[][n], int i, int j){
  float b;
  int k;
  for (k = 0; k <= n-1; k++) {
    b = A[i][k];
    A[i][k] = A[j][k];
```

```
    A[j][k] = b;
  }
}
// Підпрограма віднімання j-го рядка матриці A
// помноженого на число c від i-го рядка.
void SubtrRow(float A[][n], float c, int i, int j){
  int k;
  for (k = 0; k <= n-1; k++)
    A[i][k] = A[i][k] - c * A[j][k];
}
// Підпрограма обчислення визначника
float det(float A[][n]){
  float c,p;
  int i,j,k;
  p = 1;
  i = 0;
  while (p != 0 && i <= n-1){
    j = n-1;
    if (A[i][i] == 0){
      while (A[j][i] == 0 && j > i)
        j = j - 1;
      if (j > i)
        SwapRow(A,i,j);
      else {
        p = 0;
        break;
      }
    }
    for (k = i + 1; k <= n-1; k++){
      c = A[k][i]/A[i][i];
      SubtrRow(A, c, k, i);
    }
    p = p * A[i][i];
    i++;
  }
  return fabs(p);
}
void main(){
  float A[n][n];
  ReadMatrix(A);
  cout << det(A);
  getch();
}
```

Задачі для самостійної роботи

6.1. Визначити процедури для:

а) введення; б) виведення дійсного (натурального) вектора.

6.2. Дано масив, компоненти якого є натуральними числами. Визначити функцію для обчислення кількості:

а) парних компонент;

б) компонент, що діляться на 3 або 5;

в) нульових компонент;

г) компонент, що є простими числами.

6.3. Дано масив із числовими компонентами. Визначити функцію для:

а) знаходження максимального числа серед компонент;

б) знаходження мінімального числа серед компонент;

в) кількості максимальних компонент;

г) знаходження номера останнього максимального числа серед компонент;

ґ) кількості мінімальних компонент;

д) підрахунку кількості компонент, більших за задане число.

6.4. Визначити процедуру одночасного обчислення максимальної й мінімальної серед компонент із парними та непарними номерами.

6.5. Визначити функцію для обчислення суми компонент дійсного вектора, які розташовані між максимальною та мінімальною компонентами (всі компоненти вектора різні).

6.6. Визначити функції обчислення:

а) середнього арифметичного компонент дійсного вектора;

б) норми дійсного вектора;

в) відстані між двома точками у n-мірному евклідовому просторі;

г) скалярного добутку двох дійсних векторів.

6.7. Скласти підпрограми:

а) множення дійсного вектора на число;

б) нормування дійсного вектора;

в) знаходження суми двох дійсних векторів;

г) обміну значень двох дійсних векторів;

ґ) пошуку однакових компонент.

д) перестановки компонент вектора у зворотному порядку.

6.8. Дано два дійсних вектори розмірності n. Визначити процедуру пошуку найменшої серед тих компонент першого вектора, які входять до другого вектора.

6.9. Визначити процедуру пошуку спільної компоненти двох векторів.

6.10. Визначити процедуру пошуку в заданому векторі компонент, що є числами Фібоначчі.

6.11. Визначити процедуру обчислення кількості інверсій у заданому векторі (тобто таких пар компонент, у яких більше число стоїть ліворуч від меншого: $x_i > x_j$ при $i < j$).

6.12. Визначити функцію, яка перевіряє впорядкованість компонент вектора за зростанням.

6.13. Визначити процедуру пошуку спільної компоненти двох упорядкованих векторів.

6.14. Дано два впорядкованих за неспаданням вектори. Визначити процедуру об'єднання:

а) усіх компонент;

б) спільних компонент цих двох векторів у третій, щоб він знову став упорядкованим за неспаданням.

6.15. Розглядаючи вектори A та B як послідовності цифр десяткового запису деяких невід'ємних цілих чисел, отримати вектор C – аналогічне зображення для суми цих двох чисел.

6.16. Визначити процедуру перетворення дійсного вектора за правилом: усі від'ємні компоненти вектора перенести на його початок, а всі інші – у кінець, зберігаючи початкове взаємне розташування як серед від'ємних, так і серед інших компонент.

6.17. Циклічним k-зсувом вектора $(a_1, a_2, ..., a_n)$ ліворуч називається вектор $(a_{k+1}, ..., a_n, a_1, ..., a_k)$. Визначити підпрограму для його обчислення.

6.18. Визначити циклічний k-зсув вектора праворуч і підпрограму для його обчислення.

6.19. Визначити функцію обчислення вектора B за формулами

$$b_i = a_1 + a_2 + ... + a_n,\ i = 1,2,...,n,$$

де a_i – компоненти заданого вектора A.

6.20. Дано масив, компонентами якого є коефіцієнти многочлена

$$P_n(x) = a_0 x^n + a_1 x^{n-1} + ... + a_{n-1} x + a_n.$$

Визначити функції обчислення:

а) значення многочлена за заданого значення x;

б) похідної від многочлена за заданого значення x;

в) інтеграла від многочлена $P_n(x)$ на заданому відрізку.

6.21. Визначити функції для обчислення суми й добутку двох многочленів

$$A(x) = a_0 + a_1 x + ... + a_n x^n \text{ і } B(x) = b_0 + b_1 x + ... + b_m x^m .$$

6.22. Задано многочлен $P(x)$ із нульовим вільним членом. Знайти перші n коефіцієнтів розвинення $1/P(x)$ у ряд Тейлора в околі точки $x = 0$.

6.23. Скласти підпрограму обчислення частки $Q(x)$ і залишку $R(x)$ від ділення многочлена $P(x) = p_0 + p_1 x + ... + p_n x^n$ на многочлен $T(x) = t_0 + t_1 x + ... + t_m x^m$ за схемою $P(x) = T(x) \cdot Q(x) + R(x)$, $deg(R) < deg(T)$, де deg – степінь многочлена.

6.24. Визначити функцію для обчислення найбільшого спільного дільника двох многочленів.

6.25. Задано два вектори $X = \{x_i\}$, $Y = \{y_i\}$, $i = 0,..., n$, причому серед x_i немає однакових. Знайти в точці t значення інтерполяційного полінома:

а) у формі Ньютона (реалізувати програму без використання додаткових масивів):

$$P_n(t) = x_0 + (t - x_0)Y(x_0; x_1) + (t - x_0)(t - x_1)Y(x_0; x_1; x_2) + \ldots + \\ + (t - x_0)(t - x_1)\ldots(t - x_{n-1})Y(x_0; x_1; x_2; \ldots; x_n),$$

де введено позначення

$$Y(x_i; x_j) = \frac{y_i - y_j}{x_i - x_j};\ \ Y(x_i; x_j; x_k) = \frac{Y(x_i; x_j) - Y(x_j; x_k)}{x_i - x_k};\ \ \ldots.$$

б) у формі Лагранжа:

$$P_n(t) = y_0 L_0(t) + y_1 L_1(t) + \ldots + y_n L_n(t),$$

де введено позначення

$$L_k(t) = \frac{W_{n+1}(t)}{(t - x_k)W'_{n+1}(x_k)};\ \ W_{n+1}(t) = (t - x_0)(t - x_1)...(t - x_n).$$

6.26. Визначити функції для обчислення:

а) суми всіх елементів матриці, що належать головній діагоналі;

б) суми всіх елементів, що належать побічній діагоналі;

в) суми всіх недіагональних елементів матриці;

г) кількості нульових елементів матриці.

6.27. Визначити функції для обчислення норм матриці $A = (a_{ij})_{i,j=1,...,n}$:

а) $\| A \| = \max\limits_{i,j=1,...n} | a_{ij} |$;

б) $\| A \| = \sum\limits_{i,j=1}^{n} | a_{ij} |$;

в) $\| A \| = \max\limits_{i=1,...n} \sum\limits_{j=1}^{n} | a_{ij} |$;

г) $\| A \| = \max\limits_{j=1,...n} \sum\limits_{i=1}^{n} | a_{ij} |$;

ґ) $\| A \| = \sqrt{\sum\limits_{i,j=1}^{n} a_{ij}^2}$;

д) $\| A \| = \sqrt[p]{\sum\limits_{i,j=1}^{n} a_{ij}^p}$.

6.28. Визначити процедуру пошуку заданого елемента в матриці.

6.29. Визначити процедуру пошуку в матриці:

а) індексів усіх її ненульових елементів;

б) кількості всіх її різних елементів.

6.30. Визначити процедуру обчислення в матриці:

а) найменшого елемента; б) найбільшого елемента;

в) суми елементів рядка, в якому розташовано елемент із найменшим значенням;

г) суми найбільших значень елементів її рядків;

ґ) суми елементів рядків із від'ємним елементом головної діагоналі.

6.31. Елемент матриці називається "особливим", якщо:

1) він більший за суму інших елементів свого стовпчика;

2) у його рядку ліворуч від нього розташовані елементи, менші за нього, а праворуч – більші.

Визначити процедуру пошуку кількості "особливих" елементів матриці.

6.32. Визначити процедуру побудови за заданою матрицею A цілочислового вектора b, присвоївши його k-й компоненті значення 1, якщо виконується вказана нижче умова, і значення 0 – у протилежному випадку:

а) усі елементи k-го стовпчика матриці однакові;

б) елементи k-го рядка матриці впорядковані за спаданням;

в) k-й рядок матриці симетричний;

г) елементи k-го рядка матриці не перебільшують заданого числа x;

ґ) у k-му рядку матриці є принаймні один від'ємний елемент.

6.33. Задано дійсну матрицю розміру $m \times n$. Знайти вектор B, k-та компонента якого b_k – це:

а) сума абсолютних величин елементів k-го рядка матриці;

б) добуток елементів k-го рядка матриці;

в) значення середнього арифметичного елементів k-го рядка матриці;

г) число від'ємних елементів у k-му рядку матриці;

ґ) добуток квадратів тих елементів k-го рядка матриці, модулі яких належать відрізку $[1,1.5]$ (якщо таких елементів немає, то покласти $b_k = 1$);

д) значення першого за порядком додатного елемента k-го рядка матриці (якщо таких елементів немає, то покласти $b_k = 10$);

е) сума елементів, які містяться за першим від'ємним елементом у k-му рядку матриці (якщо таких елементів немає, то покласти $b_k = 100$).

6.34. Дано дійсну матрицю розмірності $m \times n$. Визначити підпрограму побудови вектора, компонентами якого є:

а) найменше значення елементів рядків;

б) різниці між найбільшим і найменшим значеннями елементів рядків;

в) найбільші зі значень елементів рядків від початку до головної діагоналі включно.

6.35. Скласти програму визначення послідовності чисел Фібоначчі F_n, використовуючи матричну властивість

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}, \; n \geq 0.$$

Для правильності рівності при $n = 0$ вважати, що $F_{-1} = 0$.

6.36. Визначити підпрограми:

а) транспонування матриці; б) множення матриці на вектор;

в) перестановки двох заданих рядків (стовпчиків) матриці;

г) перестановки заданого рядка квадратної матриці із заданим її стов-

пчиком;

ґ) побудови цілочислової квадратної матриці порядку 7, елементами якої є числа $1, 2, ..., 49$, розташовані в ній за спіраллю;

д) видалення з матриці заданого рядка й заданого стовпчика.

6.37. Визначити функцію для обчислення:

а) визначника квадратної матриці; б) рангу квадратної матриці.

6.38. Визначити функцію перевірки матриці на симетричність.

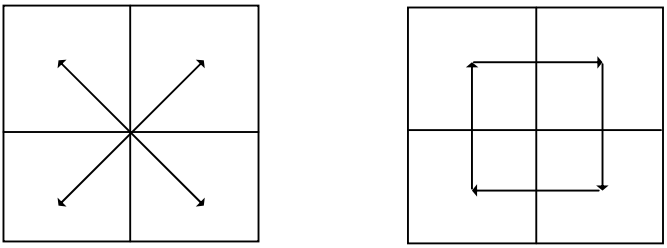
6.39. Нехай матриці A, B – симетричні. Для компактного подання інформації елементи матриць зберігаються у вигляді одновимірних масивів (тільки елементи верхньої трикутної частини за рядками). Визначити підпрограми:

а) добутку матриці A на заданий вектор b;

б) добутку матриці A на матрицю B.

6.40. Задано дійсну квадратну матрицю порядку $2n$. Побудувати нову матрицю, переставляючи її блоки розміру $n \times n$:

а) відповідно до рис. 6.1 а; б) відповідно до рис. 6.1 б.



а б

Рис. 6.1

6.41. Задано цілочислову квадратну матрицю порядку n і цілочисельний вектор розмірності m. Замінити нулями в матриці ті елементи, для яких існують рівні серед компонент даного вектора.

6.42. Задано натуральне число m, цілі числа $(a_1, a_2, ..., a_m)$ та цілочислова квадратна матриця порядку m. Рядок із номером i матриці назвемо відміченим, якщо $a_i > 0$, і невідміченим – у протилежному випадку. Необхідно:

а) усі елементи, розташовані у відмічених рядках матриці, перетворити за правилом: додатні замінити на -1, від'ємні – на 1, а нульові залишити без зміни;

б) підрахувати кількість від'ємних елементів матриці, розташованих у відмічених рядках.

6.43. Задано дійсні матриці A, B, C, N порядку m, причому N – нульова матриця. Побудувати за цими матрицями матрицю D розміру $2m \times 3m$ у вигляді

$$D = \begin{pmatrix} A & B & C \\ B & N & A \end{pmatrix}.$$

6.44. Визначити процедуру перестановки місцями рядка матриці, що містить елемент із найменшим значенням, із рядком, в якому є елемент із найбільшим значенням.

6.45. Задано координати n точок на площині. Знайти номери двох точок, відстань між якими найбільша (вважати, що така пара точок єдина).

6.46. Визначити процедуру перестановки рядків і стовпчиків матриці так, щоб елемент, який задовольняє умову $Q(x)$, опинився в її верхньому лівому куті.

Указівка. Знаходимо номер рядка k і номер стовпчика l елемента матриці, що задовольняє умову $Q(x)$. Далі міняємо місцями 1-й і k-й рядки, 1-й і l-й стовпчики.

6.47. Елемент матриці назвемо сідловою точкою, якщо він є найменшим у своєму рядку й водночас – найбільшим у своєму стовпчику. Для заданої цілочислової матриці визначити процедуру пошуку індексів усіх сідлових точок.

6.48. Дано дійсну квадратну матрицю, усі елементи якої – різні. Визначити процедуру обчислення скалярного добутку рядка, в якому міститься найбільший елемент матриці, на стовпчик із найменшим елементом.

6.49. Визначити функцію, яка перевіряє, чи є задана цілочислова квадратна матриця:

а) ортонормованою, тобто такою, в якій скалярний добуток кожної пари різних рядків дорівнює 0, а скалярний добуток кожного рядка на себе дорівнює 1;

б) магічним квадратом, тобто такою, в якій суми елементів в усіх рядках і стовпчиках однакові.

6.2. Рядки

Означення 6.2. Довільна послідовність символів називається **рядком**.

Щоб відрізняти рядки від ідентифікаторів і ключових слів, послідовність символів рядка обмежують подвійними лапками, наприклад, `"string"`.

У мові С тип даних **рядок** не входить до числа простих типів, а реалізується за допомогою масиву символів:

```
char s[maxls];
```

тут s – ім'я рядка, maxls – максимальна кількість символів, що може міститися у рядку s.

Рядок у пам'яті займає maxls байтів. Нумерація символів у рядку починається з 0. При цьому останній символ рядка має номер maxls-1.

Порівняння рядків проводиться лексикографічно.

Означення 6.3. Казатимемо, що послідовність $a = \{a_1 a_2 ... a_n\}$ лексикографічно менша (або просто менша) за послідовність $b = \{b_1 b_2 ... b_n\}$, якщо існує таке натуральне k, що для всіх $i \le k$ виконується рівність $a_i = b_i$, а для $i = k+1$ справедлива нерівність $a_{k+1} < b_{k+1}$. Якщо для всіх $i \le n$ виконується рівність $a_i = b_i$, то казатимемо, що послідовності a та b лексикографічно рівні (або просто рівні). Наприклад, послідовність $\{012\}$ менша за послідовність$\{021\}$. Також, якщо згадати, що символи можна порівнювати, порівнюючи їхні номери в алфавіті, то послідовність символів $\{AAA\}$ менша за послідовність $\{ABC\}$.

Зауваження. Вищенаведене означення можна узагальнити на випадок послідовностей, до яких входить неоднакова кількість членів. Для цього вважатимемо, що порожній член послідовності (тобто його відсутність) менший за будь-який наявний.

Із зауваження можемо зробити висновок, що, наприклад послідовність символів $\{ABC\}$ менша за послідовність символів $\{ABCD\}$.

Означення 6.4. Лексикографічний порядок послідовностей – порядок, за якого послідовності відсортовані за (лексикографічним) зростанням.

Прикладом лексикографічного порядку є послідовність слів у словнику.

Означення 6.5. Нуль-термінований рядок – це спосіб зображення рядків у пам'яті комп'ютера, при якому кінець рядка визначає спеціальний нуль-символ.

Нуль-терміновані рядки є стандартом у C. Для нуль-символа використовується позначення '\0'. Наприклад, нуль-термінований рядок "String" може бути представлений так:

S	t	r	i	n	g	\0	$	2	F

Тобто рядком вважається послідовність символів, що передує нуль-символу. Символи, що йдуть у рядку після нуль-символа, називаються **сміттям** – часто це випадкові дані, що могли лишитися у буфері від попередніх рядків або від іншого використання пам'яті.

Основні операції для роботи з рядками:

а) звернення до i-го символа рядка s

```
s[i]
```

Наприклад,

```
char s[20] = "abcdef"; // s = "abcdef"
```

```
s[2] = 'X';            // s = "aXcdef"
```

б) операція

```
&s[i]
```

повертає підрядок рядка s, починаючи з символа з номером i до кінця рядка.

```
char s[20] = "abcdef"; // &s[2]= "cdef"
```

Основні функції з бібліотеки string.h для роботи з рядками:

а) strlen(s) – довжина рядка s.

```
char s[20] = "abcdef";
n = strlen(s); // n = 6
```

б) strcmp(s1, s2) – порівняння рядків s1 з s2. Вона повертає від'ємне значення, якщо s1 < s2; нуль, – якщо s1 = s2; додатне значення, якщо s1 > s2.

в) strcpy(s, s1) – копіювання рядка s1 у рядок s. Ця функція є аналогом інструкції присвоєння для рядків.

```
char s[20], s1[20] = "abcdef";
strcpy(s, s1);    // s = "abcdef"
strcpy(s, "abc"); // s = "abc"
```

г) strncpy(s, s1, n) – копіювання n символів рядка s1 у рядок s.

```
char s[20], s1[20] = "abcdef";
strncpy(s, s1, 3); // s = "abc"
```

ґ) memmove(s, s1, n) – при роботі з рядками ця інструкція рівносильна вищенаведеній операції strncpy.

```
char s[20], s1[20] = "abcdef";
memmove(s, s1, 3); // s = "abc"
```

Зауваження (про коректну роботу функцій копіювання рядків). Інструкції strcpy, strncpy і memmove не додають до рядка, у який відбувається копіювання, символ кінця рядка '\0'. Це може призводити до некоректної роботи програми. Наприклад, якщо

```
s1 = "abcdef" // s1 = “abcdef\0xyz”;
s  = "s"      // s  = "s\0qERROR\0"
```

то після виконання інструкції

```
strncpy(s, s1, 3);
```

у змінній s буде

```
s = "abcERROR" // s  = "abcERROR\0"
```

Тут у коментарях вказано, який вигляд мають рядки s і s1 у пам'яті комп'ютера. Таким чином, відповідальність за коректну роботу інструкцій strcmp, strncpy і memmove повністю покладається на програміста.

д) strcat(s, s1) – приписування рядка s1 праворуч до рядка s.

```
char s[20]= "abc", s1[20] = "def";
strcat(s, s1);  // s = "abcdef"
```

У підпрограмі для strcat обидва параметри мають бути рядками. Проте часто виникає ситуація, коли потрібно до рядка дописати символ. Це можна реалізувати, перемістивши нуль-символ на одну позицію праворуч, а на його попередню позицію записати потрібний символ:

```
char s[20]= "abc", c = 'D';
int n = strlen(s);
s[n + 1] = '\0'; s[n] = c; // s = "abcD"
```

Для введення та виведення рядків можна використовувати стандартні потоки введення/виведення cin/cout із бібліотеки iostream.h. Проте слід зауважити, що потік cin символ пропуску розуміє як кінець рядка. Це призводить до проблем, коли необхідно вводити текст з клавіатури, що містить символи пропуску.Тому для введення/виведення рядків природніше використовувати функції із бібліотеки stdio.h:

а) gets(s) – введення рядка s із клавіатури;

б) puts(s) – виведення рядка s на екран.

Приклади розв'язання задач

Приклад 6.5. Скласти програму перевірки входження заданого символа до заданого рядка.

Розв'язок. Оскільки рядок – це впорядкований набір символів, то для пошуку заданого символу послідовно порівнюватимемо його з кожним символом у рядку. У програмі використаємо змінну is_char, що буде набувати значень *Хибності* (тобто 0), якщо рядок не містить заданий символ та *Істини* (тобто 1) – у протилежному випадку. Константа N визначатиме максимальну довжину рядка. Отже, програма матиме вигляд

```
#include<iostream.h>
#include<string.h>
#include<stdio.h>
#include<conio.h>
#define N 255
void main(){
  int is_char = 0;
  int i;
  char c, s[N];
  cout << "Введіть рядок: ";
  gets(s);
  cout << "Введіть символ: ";
  c = getch();
  cout << c << '\n';
  for (i = 0; i <= strlen(s)-1; i++)
    if (s[i] == c){
       is_char = 1;
       break;
```

```
    }
  if (is_char)
    cout << c << " ВХОДИТЬ У " << s << '\n';
  else
    cout << c << " НЕ ВХОДИТЬ У " << s << '\n';
}
```

Приклад 6.6. Вивести на друк лише великі латинські літери, що входять до заданого рядка.

Розв'язок. Використаємо те, що літери в таблиці кодування розташовуються в алфавітному порядку. Таким чином, якщо символ розміщується між символами `'A'` і `'Z'`, то він є великою латинською літерою. Отже, програма матиме вигляд:

```
#include<iostream.h>
#include<string.h>
#include<stdio.h>
#define N 255
void main(){
  int i;
  char s[N];
  cout << "Enter string: ";
  gets(s);
  for (i = 0; i<=strlen(s) - 1; i++)
    if (s[i] >= 'A' && s[i] <= 'Z')
      cout << s[i];
  cout << '\n';
}
```

Приклад 6.7. У заданому рядку всі символи `'0'` замінити на `'1'`, а символи `'1'` – на `'0'`.

Розв'язок. Посимвольно пробігатимемо рядок. Якщо поточний символ буде `'0'` або `'1'`, то змінюватимемо його на `'1'` або `'0'`, відповідно.

```
#include<iostream.h>
#include<string.h>
#include<stdio.h>
#define N 255
void main(){
  int i;
  char s[N];
  cout << "Введіть рядок: ";
  gets(s);
  for (i = 0; i <= strlen(s) - 1; i++){
    if (s[i] == '0')
      s[i] = '1';
    else if (s[i] == '1')
```

```
        s[i] = '0';
    }
    cout <<"Змінений рядок: " << s << '\n';
}
```

Приклад 6.8. Створити програму перетворення рядка, замінивши в ньому кожну крапку трьома крапками.

Розв'язок. Оскільки у задачі вимагається замінити один символ кількома, то підхід, використаний у попередньому прикладі, не можна застосувати до поточної задачі.

Використаємо допоміжний рядок `s1`, у якому зберігатиметься результат. На початку програми покладемо рядок `s1` порожнім. Далі, посимвольно пробігаючи вихідний рядок, дописуватимемо до рядка `s1` символи рядка `s`. При цьому, якщо поточний символ рядка буде крапкою, то до рядка `s` допишемо додатково дві крапки. Оскільки аргументами функції конкатенації `strcat` можуть бути лише рядки, то у програмі використаємо додаткову рядкову змінну `a`, що складається з двох символів – на нульову позицію рядка `a` записуватимемо поточний символ вихідного рядка, а на першу – нуль-символ. Отже, програма матиме вигляд

```
#include<iostream.h>
#include<string.h>
#include<stdio.h>
#define N 255
void main(){
  int i;
  char s[N], s1[N];
  char a[2];
  strcpy(s1, "");
  a[1] = '\0';
  cout << "Введіть рядок: ";
  gets(s);
  for (i = 0; i <= strlen(s) - 1; i++){
    a[0] = s[i];
    strcat(s1, a);
    if (s[i] == '.')
      strcat(s1, "..");
  }
  cout << s1 << '\n';
}
```

Приклад 6.9. Скласти функцію побудови інверсії заданого рядка.

Розв'язок. Інверсією рядка називається рядок, записаний тими самими символами у зворотному порядку.

Спосіб 1. Для отримання інверсії у вихідному рядку мінятимемо місцями рівновіддалені від початку і кінця рядка символи.

```
void Inverse(char s[]){
  char c;
```

```
    int i;
    for (i = 0; i <= (strlen(s)-1) / 2; i++){
      c = s[i];
      s[i] = s[strlen(s)-1-i];
      s[strlen(s)-1-i] = c;
    }
  }
```

Спосіб 2. Для побудови інверсії вихідного рядка використаємо допоміжний рядок `s1`. Спочатку покладемо рядок `s1` порожнім. Далі послідовно дописуємо символи рядка `s` до рядка `s1`, пробігаючи `s` від останнього до першого символа.

```
  void Inverse(char s[]){
    char s1[N], a[2];
    strcpy(s1, "");
    a[1] = '\0';
    int i;
    for (i = strlen(s)-1; i >= 0; i--){
      a[0] = s[i];
      strcat(s1, a);
    }
    strcpy(s, s1);
  }
```

Спосіб 3. Опишемо підпрограму інверсії рекурсивно. Рекурсивну підпрограму `Inverse(s)` можна задати так:

а) інверсія порожнього рядка або рядка, що складається з одного символа, – це вихідний рядок. Тому, якщо довжина рядка `s` є не більшою від одого символа, то `Inverse(s)` залишає рядок `s` без змін;

б) якщо довжина рядка `s` більша за 1, то у вихідному рядку міняємо місцями перший та останній символи та викликаємо підпрограму `Inverse` для підрядка `s1`, що є рядком `s` без першого й останнього символів. Отже, підпрограма:

```
  void Inverse(char s[]){
    if (strlen(s) > 1){
      char c_first[2], c_last[2], s1[N];
      /* у змінні c_first і c_ last копіюємо
         1-ий і останній символи рядка s */
      c_first[0] = s[0];
      c_last[0] = s[strlen(s)-1];
      c_first[1] = c_last[1] = '\0';
      /* у рядок s1 копіюємо рядок s без
         1-го і останнього символів */
      strncpy(s1, &s[1], strlen(s)-2);
      s1[strlen(s)-2] = '\0';
```

```
    Inverse(s1);     // Рекурсивний виклик
    strcpy(s, c_last);
    strcat(s, s1);
    strcat(s, c_first);
  }
}
```

Рекурсивну підпрограму за вищенаведеним алгоритмом можна записати простіше, якщо використати два додаткових формальних параметри, які визначатимуть межі рядка (тобто з якого по який символ), над яким потрібно проводити операцію інвертування.

```
void Inverse(char s[], int l, int r){
  char c;
  if (l < r){ // довжина рядка <= 1
    c = s[l];
    s[l] = s[r];
    s[r] = c;
    Inverse(s, l+1, r-1);
  }
}
```

Єдиною незручністю, порівняно з попереднім прикладом, є те, що початковий виклик підпрограми `Inverse` для рядка `s` має бути таким:

```
Inverse(s, 0, strlen(s)-1);
```

Проте цю незручність можна подолати, якщо описати іншу підпрограму, що викликатиме підпрограму `Inverse`:

```
void Inv(char s[]){
  Inverse(s, 0, strlen(s)-1);
}
```

Задачі для самостійної роботи

6.50. Скласти функцію підрахунку загальної кількості входжень символів `'+'`, `'-'`, `'*'` у рядок.

6.51. Скласти функцію, яка визначає позицію першого (останнього) входження заданого символа до заданого рядка.

6.52. Рядок містить арифметичний вираз, у якому використовуються круглі дужки. Перевірити, чи правильно в ньому стоять дужки.

6.53. Скласти функцію, що повертає кількість різних символів у рядку.

6.54. Знайти символ, який входить до рядка S найбільшу кількість разів.

Вказівка. Визначити масив з індексами символьного типу та елементами натурального типу. Обчислити та зберегти кількість входжень кожного символа до рядка S. Знайти максимальний елемент цього масиву.

6.55. Дано рядок A, що містить принаймні одну цифру. Знайти:

а) максимальну цифру, що міститься у рядку;

б) цифру, що входить до рядка найбільшу кількість разів;

в) суму цифр, що входять до рядка.

6.56. Визначити, чи є заданий текст правильним записом цілого числа.

6.57. Дано текст, що має вигляд:

$$d_1 \pm d_2 \pm ... \pm d_n,$$

де d_i, $1 \le i \le n$ – цифри, $n > 1$. Визначити записану в тексті алгебраїчну суму.

6.58. Розв'язати попередню задачу за припущення, що d_i – натуральні числа.

6.59. Дано число n із проміжку від 0 до 10^9. Надрукувати його українськими словами (напр., 234 = двісті тридцять чотири).

6.60. Скласти підпрограму, яка, починаючи із заданої позиції, збільшує всі парні цифри, що містяться у рядку, на 1.

6.61. Скласти підпрограму, яка всі цифри, що містяться у рядку, замінює їх квадратами.

6.62. Створити підпрограму перетворення рядка A, замінивши у ньому всі знаки оклику `'!'` знаками запитання `'?'`, а кожен знак запитання `'?'` – двома знаками `"??"`. Програму створити двома способами:

а) з використанням додаткової рядкової змінної;

б) без використання додаткової рядкової змінної.

6.63. Скласти програму перетворення рядка A, видаливши в ньому кожний символ `'*'` і подвоївши кожний символ, відмінний від `'*'`.

6.64. Дано рядок A. Скласти підпрограми, що видаляють з нього:

а) символи, що стоять на парних місцях;

б) символи, що дорівнюють символу c;

в) цифри;

г) цифри, що трапляються у рядку більше одного разу;

ґ) символи, що не є цифрами;

д) символи, що належать відрізку $[c1, c2]$.

6.65. Скласти програму видалення з рядка всіх входжень заданої групи символів.

6.66. Скласти програму перетворення рядка A видаленням із нього всіх ком, які передують першій крапці, і заміною у ньому знаком `'+'` усіх цифр `'3'`, які трапляються після першої крапки.

6.67. Скласти програму, яка за рядком A та символом s будує новий рядок, отриманий заміною кожного символа, що йде за s, заданим символом c.

6.68. Рядок називається симетричним, якщо його символи, рівновіддалені від початку та кінця рядка, збігаються. Порожній рядок вважається симетричним. Скласти функцію перевірки заданого рядка на симетричність.

6.69. Перевірити, чи складається рядок із:

а) двох симетричних підрядків;

б) n симетричних підрядків.

6.70. Скласти програму підрахунку найбільшої кількості цифр, що йдуть поспіль у рядку A.

6.71. Задано рядок, що складається з великих літер латинського алфавіту. Скласти програму перевірки впорядкованості цих літер за алфавітом.

6.72. Рядок називається монотонним, якщо складається зі зростаючої або спадної послідовності символів. Скласти програму перевірки монотонності рядка.

6.73. Виділити з рядка найбільший монотонний підрядок, коди послідовних символів якого відрізняються на 1.

6.74. Скласти програму підрахунку кількості входжень до рядка A заданої послідовності літер.

6.75. Скласти програму знаходження кількості входжень до рядка A символів, перелічених у рядку V. Знайти символ, кількість входжень якого до рядка A:

а) максимальна; б) мінімальна.

6.76. Виділити з рядка A найбільший підрядок, перший та останній символи якого збігаються.

6.77. Видалити з рядка симетричні початок і кінець. Наприклад, рядок `"abcdefba"` має перетворитися на рядок `"cdef"`.

6.78. Замінити всі пари однакових символів рядка, які йдуть поспіль, одним символом. Наприклад, рядок `"aabcbb"` має перетворитися на рядок `"abcb"`.

6.79. Скласти процедуру "стискання" рядка: кожний підрядок, який складається з кількох входжень деякого символа, замінюється на цей символ.

6.80. Видалити з рядка всі повторні входження символів.

6.81. Побудувати рядок S із рядків $S1$, $S2$ так, щоб до S входили:

а) ті символи $S1$, які не входять до $S2$;

б) усі символи $S1$, які не входять до $S2$, і всі символи $S2$, які не входять до $S1$.

6.82. Скласти підпрограму, яка з першого заданого рядка видаляє кожний символ, що належить і другому заданому рядку.

6.83. Скласти програму виведення на друк усіх цифр, які входять до заданого рядка, та окремо – решту символів, зберігаючи при цьому взаємне розташування символів у кожній із цих двох груп.

6.84. Скласти програму виведення на друк тільки маленьких літер латинського (українського) алфавіту, які входять до заданого рядка.

6.85. Вивести на друк усі символи рядка A, що складається з маленьких літер латинського алфавіту, відповідними великими літерами.

6.86. Скласти програму виведення на друк в алфавітному порядку всіх різних маленьких латинських (кириличних) літер, які входять до даного рядка.

6.87. Перевірити, чи є даний рядок ідентифікатором, натуральним числом, чи ні тим, ні іншим. Скласти функцію, що визначає: чи є заданий символ літерою; чи є даний символ цифрою.

6.88. Скласти підпрограму для обчислення значення натурального числа за заданим рядком символів, який є записом цього числа в системі числення за основою b $(2 < b < 16)$. Використати функцію, яка за заданим символом повертає відповідну цифру в системі числення за основою b.

6.89. Скласти підпрограму для отримання за заданим натуральним числом рядка символів, що є записом цього числа в системі числення за основою b $(2 < b < 16)$. Використати функцію, яка за заданою цифрою у системі числення за основою b повертає символ, що відповідає цій цифрі.

6.90. Використовуючи тільки поелементну роботу з рядками, описати функції, які реалізують такі дії:

а) знищення n символів рядка S, починаючи з позиції k;

б) вставку рядка A до рядка B, починаючи з позиції k;

в) виділення із рядка S підрядка R довжиною n символів, починаючи з позиції k;

г) перетворення дійсного числа d на рядок S;

ґ) перетворення рядка S на дійсне число d з кодом перетворення i: $i = 0$ у випадку успішного перетворення або i дорівнює номеру першого помилкового символа рядка;

д) виділення з рядка A підрядка довжиною n символів, починаючи з кінця рядка.

6.91. Визначити рекурсивні функції:

а) перевірки заданого рядка на симетричність;

б) побудови рядка, інвертованого відносно до заданого;

в) заміни у вихідному рядку всіх входжень даного символа даним рядком;

г) перевірки, чи є один рядок початком іншого;

ґ) перевірки на входження одного рядка до іншого.

Зауваження. У наступних задачах "словами" називаються підрядки, розділені одним чи кількома пропусками, що не містять пропусків усередині.

6.92. Знайти найкоротше й найдовше слово заданого рядка.

6.93. Дано рядок A, що містить послідовність слів. Скласти підпрограми, що визначають:

а) кількість усіх слів;

б) кількість слів, що починаються із заданого символа c;

в) кількість слів, що закінчуються заданим символом c;

г) кількість слів, що починаються й закінчуються заданим символом *c*;

ґ) кількість слів, що починаються й закінчуються однаковим символом.

6.94. Скласти процедури, які виділяють:

а) перше слово рядка, залишаючи рядок без першого слова;

б) *n*-те слово рядка.

Використати одну з цих процедур для:

а) підрахунку кількості слів рядка;

б) отримання найдовшого слова;

в) отримання найкоротшого слова;

г) отримання всіх слів, які є паліндромами (симетричними);

ґ) отримання всіх слів, які є ідентифікаторами;

д) отримання всіх слів, які є натуральними числами.

6.95. Скласти підпрограми, що виводять на друк:

а) усі слова рядка у зворотному порядку;

б) усі слова, які трапляються у рядку по одному разу;

в) цей самий рядок, але з видаленням із нього повторних входжень слів;

г) усі слова рядка у порядку зростання;

ґ) перше слово рядка, що починається заданим символом *c*;

д) останнє слово рядка, що закінчується символом *c*;

е) усі слова, що складаються з різних символів, довжина яких більша за один символ;

є) усі симетричні слова.

6.96. Дано два рядки, кожен з яких містить послідовність слів. Описати підпрограми, що друкують:

а) усі слова першого рядка, що не входять до другого рядка;

б) усі різні слова першого рядка, що входять до другого рядка;

в) усі різні слова, що входять до обох рядків;

г) усі слова, що входять до обох рядків тільки по одному разу;

ґ) усі різні слова, що містяться тільки в одному з рядків.

6.97. Скласти програму додавання "до стовпчика" двох чисел, записаних у вигляді рядків, що є позиційними записами цих чисел у десятковій системі числення. У програмі описати та використати підпрограми:

а) *GetDigit*(*c*) – повертає цифру, що відповідає символу *c*, і – 1, якщо *c* не є символом-цифрою;

б) *GetSymbol*(*d*) – повертає символ, що відповідає цифрі *d*;

в) *AddDigit*(*n*1, *n*2, *p*, *n*) – підпрограма додавання двох цифр *n*1, *n*2 з урахуванням перенесення *p* й отриманням останньої цифри результату в *n*;

г) *AddColumn*(*S*, *S*1, *S*2) – підпрограма додавання двох рядків *S*1 і *S*2 до стовпчика із записом результату в рядку *S*.

6.98. Скласти програму множення "у стовпчик" двох чисел, записаних у вигляді рядків, що є позиційними записами цих чисел у десятковій системі числення. У програмі описати та використати підпрограми:

а) *GetDigit*(*c*) – повертає цифру, що відповідає символу *c*, і – 1, якщо *c* не є символом-цифрою;

б) *GetSymbol*(*d*) – повертає символ, що відповідає цифрі *d*;

в) *MulDigit*(*n*1, *n*2, *p*, *n*) – множення двох цифр *n*1, *n*2 з урахуванням перенесення *p* та отриманням останньої цифри результату в *n*;

г) *MulStrChar*(*S*, *c*) – підпрограма множення рядка *S* на символ *c*;

ґ) *Add Str*(*S*, *S*1, *S*2, *n*) – підпрограма додавання двох рядків *S*1 і *S*2 у стовпчик зі "зсувом" другого рядка на *n* позицій ліворуч із записом результату в змінній *S*.

6.3. Структури та об'єднання

Найзагальніший метод отримання складених типів полягає в об'єднанні елементів довільних типів. Причому самі ці елементи можуть бути у свою чергу складеними.

Означення 6.6. Структура (або **запис**) – складений тип даних у програмуванні, змінні якого складаються з набору (загалом кажучи, різнотипних) даних, так званих **полів** структури.

Найуживаніший приклад структури з математики – комплексні числа, що складаються з двох дійсних чисел. Інший приклад – координати точки, що складаються з двох або більше чисел (залежно від розмірності простору). Приклад структури, поля якої різних типів, – відомості про особу, що описуються за допомогою кількох відповідних характеристик на зразок імені, прізвища, дати народження, статі та сімейного стану.

Структуру в С можна оголосити у два способи:

```
struct T_Struct {
  T1 s1;
  T2 s2;
  ...
  Tn sn;
};
```

або

```
typedef struct {
  T1 s1;
  T2 s2;
  ...
  Tn sn;
} T_Struct;
```

де `T_Struct` – ім'я структури, s_1, s_2,..., s_n – поля структури, відповідно типів T_1, T_2,..., T_n.

У першому випадку змінні відповідного типу описують

```
struct T_Struct x;
```

у другому, –

```
T_Struct x;
```

Наприклад, опис

```
struct date {
  unsigned day;
  enum {jan, feb, mar, apr, may, jun,
        jul, aug, sep, oct, nov, dec} month;
  unsigned year;
};
```

оголошує структуру `struct date`, складену з трьох полів – `day`, `month` і `year` – число, місяць і рік відповідно. Опис

```
typedef struct {
  double re, im;
} complex;
```

оголошує комплексний тип даних `complex`, складений із двох полів `re` (дійсна частина) та `im` (уявна частина) типу `double`.

Наступний фрагмент програми оголошує структуру `avto`, змінні якої містять відомості про автомобіль і його власника

```
typedef struct {
  char nom[8];     // номер автомобіля
  char marka[21];  // марка автомобіля
  unsigned color;  // колір автомобіля
  char name[41];   // ПІБ власника
  char adress[61]; // адреса власника
} avto;
```

Оголошення змінних типу **структура** здійснюється аналогічно до оголошення звичайних змінних. Наприклад, оголошення змінних для раніше визначених типів `date`, `complex` та `avto` має вигляд:

```
struct date d;
complex z, z1, z2;
avto a;
```

При роботі зі структурами для доступу до конкретного поля ім'я змінної відділяють від імені поля символом `'.'` (крапка). Наприклад,

```
z.re = z1.re + z2.re;
cout << a.marka;
```

Означення 6.7. Об'єднанням називається структура, усі члени якої розміщуються за однією й тією самою адресою, тобто у будь-який момент об'єднання містить значення тільки одного з його членів.

Розмір об'єднання дорівнює розміру його найбільшого члена.

Оголошення об'єднання подібне до оголошення звичайних структур. При цьому замість ключового слова `struct` використовують ключове слово `union`.

Наприклад, об'єднання

```
union int_or_long {
```

```
    int  i;
    long l;
  };
```

містить або ціле число (поле `i`), або довге ціле число (поле `l`).

Оскільки об'єднання – це вид структури, то в С працюватимемо з об'єднанням як зі звичайною структурою. При цьому, аналогічно до структури, символ `'.'` використовується для відокремлення імені змінної, що є об'єднанням, від відповідного поля об'єднання. Приклад фрагмента коду для роботи з визначеним об'єднанням `union int_or_long`:

```
union int_or_long a;
/* вводиться з клавіатури значення поля i,
            при цьому поле l є невизначеним */
cin >> a.i;
cout << a.i;
/* вводиться з клавіатури значення поля l,
    при цьому вже поле i стає невизначеним  */
cin >> a.l;
cout << a.l;
```

Означення 6.8. Об'єднання, що має механізм контролю за поточним типом об'єднання під час виконання програми, називається **розміченим об'єднанням**.

Як правило, механізм контролю за поточним типом здійснюється за допомогою окремого поля, що називається **дискримінантом** об'єднання. Об'єднання у С не мають стандартної конструкції для поля дискримінанта. Отже, не можна перевірити, до якого типу належить об'єднання у кожний момент виконання програми. Контроль за поточним типом об'єднання покладається на програміста. Реалізувати тип розміченого об'єднання з дискримінантом можна за допомогою комбінації типів запису та об'єднання. Наприклад, координати точки на площині, залежно від використаної системи координат, можна описати так:

```
typedef enum {cart, polar} coord_mode;
typedef struct {
  coord_mode cm;              // поле дискримінанта
  union {
    struct { double x, y;  } c; // декартові к-ти
    struct { double r, phi;} p; // полярні к-ти
  } t;
} point;
point a, b;
```

Приклади розв'язання задач

Приклад 6.10. Введення, виведення та додавання комплексних чисел.
Розв'язок

```
#include <iostream.h>
#include <math.h>
typedef struct {
  double re, im;
} complex;
/* підпрограма введення комплексного числа */
void get_c(complex *pz){
  cout << "Дійсна частина числа = ? ";
  cin >> (*pz).re;
  cout << "Уявна частина числа = ? ";
  cin >> (*pz).im;
}
/* підпрограма виведення комплексного числа */
void put_c (complex z){
  cout << z.re;
  if (z.im > 0)
    cout << "+";
  else
    cout << "-";
  cout << fabs(z.im) << "i" ;
}
void suma(complex z1, complex z2, complex *pz){
  (*pz).re = z1.re + z2.re;
  (*pz).im = z1.im + z2.im;
}
void main(){
  complex z1,z2,z;
  cout << "Введіть перше число:\n";
  get_c(&z1);
  cout << "Введіть друге число:\n";
  get_c(&z2);
  cout << "Сума комплексних чисел дорівнює\n";
  suma(z1, z2, &z);
  put_c(z);
}
```

Для обчислення суми використовується процедура, а не програмована функція, оскільки функція у С не може мати результат складеного типу `complex`. Використання дужок у позначеннях вигляду `(*pz).re` зумовлене тим, що операція звернення до відповідного поля (`"."`) має вищий пріоритет, ніж розіменування вказівника (`"*"`).

Приклад 6.11. Описати функцію для обчислення аргумента комплексного числа.

Розв'язок. Якщо вважати, що аргумент комплексного числа $z = x + iy$ лежить у проміжку $[0, 2\pi)$, то знайти його можна за формулою

$$\arg z = \begin{cases} \operatorname{arctg}(y/x), & x > 0, \\ \pi/2, & x = 0, y > 0, \\ 3\pi/2, & x = 0, y < 0, \\ \pi + \operatorname{arctg}(y/x), & x < 0, \\ \text{н/в} & x = 0, y = 0. \end{cases}$$

Припустимо, що у програмі визначено комплексний тип `complex` так само, як у попередньому прикладі. Для спрощення вважатимемо, що аргумент числа $z = 0$ дорівнює нулю. Тоді підпрограма матиме вигляд:

```
double arg(complex z){
  if (z.re > 0)
    return atan(z.im/z.re);
  else if (z.re == 0 && z.im > 0)
    return M_PI_2;
  else if (z.re == 0 && z.im < 0)
    return 3 * M_PI_2;
  else if (z.re < 0)
    return M_PI + atan(z.im/z.re);
  else
    return 0;
}
```

Зауваження. Константи `M_PI` і `M_PI_2` використовуються для визначення чисел π і $\pi/2$, відповідно. Для використання цих констант слід підключити бібліотеку `math.h`.

Приклад 6.12. Знайти відстань d між двома точками a та b.

Розв'язок. Вважатимемо, що кожна точка площини може бути задана як у прямокутній декартовій, так і у полярній системах координат. Тому для опису точок опишемо структуру `point`, одне з полів якої є об'єднанням. Як дискримінант об'єднання використаємо поле `coord_mode` структури `point`, у якому міститиметься стала `cart` або `polar`, залежно від того, в якій системі координат задано точку.

```
#include <iostream.h>
#include <math.h>
/* Distance */
typedef enum {cart, polar} coord_mode;
typedef struct {
  coord_mode cm;
  union {
    struct { double x, y; } c;
    struct { double r, phi; } p;
  } t;
} point;
```

```
void get_point (point *p) {
  int k;
  cout << "Система координат [1-прям, 2-поляр]:";
  cin >> k;
  if (k == 1){
      (*p).cm = cart;
      cout << "Введіть x, y: ";
      cin >> (*p).t.c.x >> (*p).t.c.y;
  }
  else {
      (*p).cm = polar;
      cout << "Введіть r,phi: ";
      cin >> (*p).t.p.r >> (*p).t.p.phi;
  }
}
double dist(point a, point b){
  double d;
  switch (a.cm){
    case cart:
      switch (b.cm) {
        case cart:
/* Обидві точки задаються у прямокутній системі
   координат. Відстань шукаємо як відстань між
   точками площини, тобто як корінь квадратний із
   суми квадратів різниць координат  */
          d = sqrt((a.t.c.x - b.t.c.x) *
                   (a.t.c.x - b.t.c.x) +
                   (a.t.c.y - b.t.c.y) *
                   (a.t.c.y - b.t.c.y));
          break;
        case polar:
/* Перша точка задається у прямокутній системі
   координат, а друга – у полярній. Для
   знаходження відстані координати другої точки
   перетворюємо у прямокутну систему */
         d = sqrt(
         (a.t.c.x - b.t.p.r * cos(b.t.p.phi)) *
         (a.t.c.x - b.t.p.r * cos(b.t.p.phi)) +
         (a.t.c.y - b.t.p.r * sin(b.t.p.phi)) *
         (a.t.c.y - b.t.p.r * sin(b.t.p.phi)));
         break;
      }
      break;
    case polar:
```

```
        switch (b.cm){
          case cart:
  /* Перша точка задається у полярній системі
     координат, а друга – у прямокутній. */
             d = sqrt(
             (a.t.p.r * cos(a.t.p.phi) - b.t.c.x) *
             (a.t.p.r * cos(a.t.p.phi) - b.t.c.x) +
             (a.t.p.r * sin(a.t.p.phi) - b.t.c.y) *
             (a.t.p.r * sin(a.t.p.phi) - b.t.c.y));
             break;
           case polar:
  /* Обидві точки задаються у полярній системі.
     Відстань шукаємо за теоремою косинусів */
             d = sqrt(
             a.t.p.r * a.t.p.r + b.t.p.r * b.t.p.r -
             2 * a.t.p.r * b.t.p.r *
             cos(a.t.p.phi - b.t.p.phi));
             break;
        }
        break;
    }
    return d;
  }
  void main(){
    point a, b;
    cout << "Введіть 2 точки \n";
    get_point(&a);
    get_point(&b);
    cout << "Відстань між ними дорівнює ";
    cout << dist(a, b);
  }
```

Задачі для самостійної роботи

6.99. Описати тип зображення комплексних чисел в алгебраїчній формі. Визначити процедури:

а) введення; б) виведення комплексного числа.

6.100. Визначити функції для обчислення:

а) суми; б) різниці; в) добутку; г) частки двох комплексних чисел.

6.101. Визначити функції для обчислення:

а) аргумента; б) модуля комплексного числа.

6.102. Визначити функцію для обчислення степеня комплексного числа з натуральним показником.

6.103. Визначити тип "комплексне число у тригонометричній формі" й підпрограми для операцій та інструкцій із завдань 6.99, 6.100, 6.102.

6.104. Визначити функції переведення комплексного числа з алгебраїчної форми в тригонометричну, і навпаки.

6.105. Визначити процедуру розв'язання квадратного рівняння із заданими комплексними коефіцієнтами.

6.106. Визначити функцію обчислення значень квадратного тричлена з комплексними коефіцієнтами в заданій комплексній точці.

6.107. Визначити підпрограми для наближених обчислень значень комплексних функцій, використовуючи їхні розвинення у відповідні ряди Тейлора:

а) $e^z = 1 + \frac{z}{1!} + \frac{z^2}{2!} + ... + \frac{z^n}{n!} + ...;$

б) $\operatorname{sh} z = z + \frac{z^3}{3!} + \frac{z^5}{5!} + ... + \frac{z^{2n+1}}{(2n+1)!} + ...;$

в) $\operatorname{ch} z = 1 + \frac{z^2}{2!} + \frac{z^4}{4!} + ... + \frac{z^{2n}}{(2n)!} + ...;$

г) $\sin z = z - \frac{z^3}{3!} + \frac{z^5}{5!} - ... + (-1)^n \frac{z^{2n+1}}{(2n+1)!} + ...;$

ґ) $\cos z = 1 - \frac{z^2}{2!} + \frac{z^4}{4!} - ... + (-1)^n \frac{z^{2n}}{(2n)!} + ...;$

д) $\ln(1+z) = z - \frac{z^2}{2} + \frac{z^3}{3} - ... + (-1)^{n-1} \frac{z^n}{n} + ... \quad (|z| < 1);$

е) $\operatorname{arctg} z = z - \frac{z^3}{3} + \frac{z^5}{5} - ... + (-1)^n \frac{z^{2n+1}}{2n+1} + ... \quad (|z| < 1).$

6.108. Визначити типи запису для зображення таких понять:

а) ціна (гривні, копійки);

б) час (година, хвилина, секунда);

в) дата (число, місяць, рік);

г) адреса (місто, вулиця, будинок, квартира);

ґ) семінар (предмет, викладач, № групи, день тижня, години занять, аудиторія);

д) бланк вимоги на книгу (відомості про книгу: шифр, автор, назва; відомості про читача: № читацького квитка, прізвище; дата замовлення);

е) поле шахової дошки (напр., $a5$, $b8$);

є) коло (радіус, координати центра).

6.109. Визначимо тип `Card` (*Карта*) таким чином:

```
typedef enum {
  spade, clubs, diamonds, hearts
```

```
} suit;            // масть карти
typedef enum {
  six, seven, eight, nine, ten,
  knave, queen, king, ace
} value;           // значення карти
typedef struct {   // структура "Карта"
  suit  s;
  value v;
} Card;
```

Описати булеву функцію, яка перевіряє, чи "б'є" карта $K1$ карту $K2$, ураховуючи, що масть KM є козирною.

6.110. Використовуючи тип *Поле* (задача **6.108** е), описати булеву функцію, яка перевіряє, чи може ферзь за один хід перейти з одного заданого поля шахової дошки на інше задане поле.

6.111. Визначимо тип `Rational` (*Раціональне число*) як:

```
typedef struct {
  int          numerator;   // чисельник
  unsigned int denominator; // знаменник
} Rational;
```

Визначити функції для:

а) обчислення суми двох раціональних чисел;

б) обчислення добутку двох раціональних чисел;

в) порівняння двох раціональних чисел;

г) зведення раціонального числа до нескоротного виду.

6.112. Використовуючи опис типу *Дата* (задача **6.108** в), визначити функції обчислення:

а) дати вчорашнього дня;

б) дня тижня за його датою в поточному році.

6.113. Визначити універсальний комплексний тип, який допускає як алгебраїчне, так і тригонометричне зображення:

```
typedef enum {alg, trig} compl_mode;
typedef struct {
  compl_mode cm;
  union {
    struct { double re, im; } a;
    struct { double r, phi; } t;
  } c;
} complex;
```

Визначити основні функції для універсального комплексного типу (задачі **6.99–6.102**). Для двомісних операцій передбачити всі можливі випадки.

6.114. Визначити тип *Плоска_Фігура*, який включає трикутник, прямокутник, трапецію та круг. Побудувати функції обчислення периметра та площі плоских фігур.

6.115. Визначити універсальний тип, який допускає зображення точки на площині у прямокутній або полярній системі координат. Побудувати функцію обчислення площі трикутника з вершинами A, B, C.

6.4. Організація пошуку та вибору інформації

Пошук – у широкому сенсі – намагання досягти чогось або знайти щось. **Пошук даних** – розділ інформатики, що вивчає способи й алгоритми для пошуку та обробки інформації як серед структурованих, так і неструктурованих даних.

Алгоритми ефективного пошуку існували задовго до появи комп'ютерів і застосовувалися, наприклад для відшукання книг у бібліотеці чи слів у словнику. Розглянемо найпростіший вид пошуку серед структурованих даних – лінійний пошук.

Означення 6.9. Лінійним або **послідовним пошуком** називається алгоритм відшукання елемента серед заданого набору шляхом послідовного перебору всіх елементів.

Зауважимо, що у посібнику раніше вже використовувалися механізми лінійного пошуку, наприклад при визначенні максимального елемента серед введених із клавіатури, а також для відшукання заданих літер у рядку тощо.

Нехай у масиві a

```
t a[n]; // t – заданий тип елементів масиву
```

міститься деякий набір даних. Тоді найпростіший алгоритм знаходження елемента, що задовольняє наперед визначену ознаку

```
P(e) // вираз e типу t
```

такий:

```
isElement = 0;      // елемент не знайдено
i = 0;
while (i <= n - 1 && !isElement){
  if (P(a[i]))
    isElement = 1; // елемент знайдено
  else
    i++;           // перехід до наступного
}
// якщо isElement = 1, то a[i] – шуканий елемент
```

Тут `isElement` – булева змінна – відповідає за наявність шуканого елемента. Масив пробігається послідовно, починаючи з нульового елемента, у напрямку збільшення індексів. Якщо поточний елемент масиву задовольняє ознаку P, то змінна `isElement` набуває значення 1 (*Істина*) й пошук припиняється. У протилежному випадку відбувається

перехід до наступного елемента масиву. Якщо серед елементів масиву не знайдеться жодного елемента, що задовольняє ознаку P, то у змінній `isElement` лишиться значення 0 (*Хибність*).

Цей алгоритм зручний, якщо потрібно відповісти на запитання, чи є такий елемент, що задовольняє деяку ознаку? Дійсно, якщо у масиві a є кілька елементів, що задовольняють ознаку P, то буде знайдено лише перший із них, а запитання "Скільки таких елементів?" лишиться відкритим. Тому, якщо потрібно знайти кількість елементів, що задовольняють ознаку P, то вищенаведений алгоритм модифікують до вигляду

```
findEl = 0;     // лічильник, знайдено 0 елементів
i = 0;
while (i <= n - 1){
  if (P(a[i]))  // елемент знайдено
    findEl++;   // збільшуємо лічильник на 1
  i++;          // перехід до наступного
}
```

По завершенні циклу в змінній `findEl` міститиметься кількість елементів масиву, що задовольняють ознаку P.

Вищенаведені алгоритми не зможуть розв'язати задачі, у яких ознака P змінюється залежно від вже опрацьованих елементів, наприклад задачі про знаходження мінімуму або максимуму. Тому, якщо задача поставлена так, що заздалегідь відомо, що результат пошуку визначить принаймні один елемент серед заданих (напр., задача про знаходження мінімуму), то використовують алгоритм

```
Element = a[0];
nElement = 0;
for (i = 1; i <= n - 1; i++){
  if ( P(a[i], Element) ){
    Element = a[i];
    nElement = i;
  }
}
```

По завершенні цього фрагмента програми у змінній `Element` знаходитиметься шуканий елемент, а у змінній `nElement` – його номер у масиві.

Комбінуючи та модифікуючи наведені алгоритми, можна розв'язувати значно складніші задачі.

Приклади розв'язання задач

Приклад 6.13. Серед елементів масиву a цілих чисел знайти число x.

Розв'язок. Оскільки немає додаткової інформації про розташування чисел у масиві, то єдиний очевидний спосіб пошуку заданого елемента

– послідовне перебирання всіх значень елементів масиву, доки не буде знайдено шуканий елемент або масив не буде вичерпано. Отже, опишемо цілу функцію, що повертатиме номер шуканого елемента у масиві $(a_0, a_1, ..., a_{N-1})$ або число -1, якщо такого елемента в масиві не знайдено. Використаємо змінну a для зберігання значень масиву $(a_0, a_1, ..., a_{N-1})$, а змінну x – для шуканого елемента. Тоді підпрограма лінійного пошуку матиме вигляд

```
int LinearSearch(int a[], int x){
  int i = 0, num = -1;
  while (i <= N – 1 && num == -1)
    if (a[i] == x)
      num = i;
    else
      i++;
  return num;
}
```

Зауваження. Тут і далі при написанні підпрограм вважатимемо, що масив містить N елементів, а програма містить рядок макрозаміни на кшталт

```
#define N 5
```

Приклад 6.14. Задано N комплексних чисел. Знайти серед них найменше за модулем комплексне число.

Розв'язок. Вважатимемо, що комплексні числа – це структури типу `complex`:

```
typedef struct {
  double re, im;
} complex;
```

Опишемо допоміжну підпрограму, що визначатиме модуль комплексного числа

```
float absC(complex z){
  return sqrt(z.re*z.re + z.im*z.im);
}
```

Нехай задані комплексні числа містяться у масиві

```
complex z[N];
```

Тоді підпрограма, що знаходить серед них найменше за модулем:

```
int SearchMinComplex(complex z[]){
  int i, minN = 0;
  for (i = 1; i <= N – 1; i++)
    if (absC(z[i]) < absC(z[minN]))
      minN = i;
  return minN;
}
```

Зауваження. Якщо у наборі буде кілька мінімальних за модулем комплексних чисел, то підпрограма SearchMinComplex поверне номер останнього з них. Опишемо додатково підпрограму, що знаходить всі мінімальні за модулем комплексні числа та виводить їх на екран.

```
void ShowMinComplex(complex z[]){
  int i, Last_minN;
  Last_minN = SearchMinComplex(z);
  for (i = 0; i <= N – 1; i++)
    if (absC(z[i]) == absC(z[Last_minN]))
      printf("z[%i]=(%f,%f)", i, z[i].re, z[i].im);
}
```

Задачі для самостійної роботи

6.116. Задано вектор розмірності N, компонентами якого є записи, що містять відомості про вершини гір. У відомостях про кожну вершину вказуються назва гори та її висота. Визначити процедуру пошуку найвищої вершини.

6.117. Відомо вартість і "вік" кожної з N моделей легкових автомобілів. Визначити середню вартість автомобілів, вік яких більший за 5 років.

6.118. Відомо інформацію про ціну та наклад кожного з N журналів. Знайти середню вартість журналів, наклад яких менший за 10000 примірників.

6.119. Відомі дані про масу й об'єм N предметів, виготовлених із різних матеріалів. Знайти предмет, густина матеріалу якого найбільша.

6.120. Відомі дані про чисельність населення (у мільйонах жителів) та площі N держав. Знайти країну з мінімальною щільністю населення.

6.121. Задано вектор C розмірності N, компонентами якого є відомості про мешканців деяких міст. Інформація про кожного мешканця містить його прізвище, назву міста, місцеву адресу у вигляді вулиці, будинку, квартири. Визначити процедуру пошуку двох будь-яких жителів, що мешкають у різних містах за однаковою адресою.

6.122. Відомо дані про вартість кожного з N найменувань товарів: кількість гривень, кількість копійок. Скласти підпрограми пошуку:

а) найдешевшого товару в магазині;

б) найдорожчого товару в магазині;

в) товару, вартість якого відрізняється від середньої вартості товару в магазині не більш ніж на 5 гривень:

г) усіх товарів, вартість яких вища за середню.

6.123. Задано вектор P розмірності N, компонентами якого є записи, що містять анкети службовців деякого закладу. У кожній анкеті вказується прізвище та ім'я службовця, його стать, дата народження у вигляді числа, місяця, року. Визначити підпрограми пошуку:

а) посади, яку обіймає найбільша кількість співробітників;

б) співробітників з однаковими іменами;

в) співробітників, прізвища яких починаються із заданої літери;

г) найстаршого з чоловіків цього закладу;

ґ) співробітників, вік яких менший за середній по організації;

д) пенсійного віку (урахувати, що пенсійний вік чоловіків і жінок – різний).

6.124. Задано вектор P, компонентами якого P_i є записи, що містять дані про людину на ім'я i з указаного списку. Кожне дане складається зі статі людини та її зросту. Визначити підпрограми для:

а) обчислення середнього зросту жінок;

б) пошуку найвищого чоловіка;

в) перевірки, чи є дві людини, однакові на зріст.

6.125. Задано вектор розмірності N, компоненти якого містять інформацію про студентів деякого вишу. Відомості про кожного студента містять дані про його прізвище, ім'я, по батькові, стать, вік, курс. Визначити процедуру пошуку:

а) найпоширеніших чоловічих і жіночих імен;

б) прізвищ та ініціалів усіх студентів, вік яких є найпоширенішим.

6.126. Задано вектор розмірності N, компонентами якого є відомості про складання іспитів студентами деякого вишу. Інформація про кожного студента задана в такому вигляді: прізвище, номер групи, оцінка_1, оцінка_2, оцінка_3. Визначити процедуру пошуку:

а) студентів, що мають заборгованості принаймні з одного з предметів;

б) предмета, складеного найуспішніше;

в) студентів, що склали всі іспити на 5 і 4.

6.127. Відомо кількість очок, отриманих кожною з N команд, що взяли участь у чемпіонаті з футболу. Жодна пара команд не набрала однакової кількості очок. Визначити:

а) команду, що стала чемпіоном;

б) команди, що посіли друге й третє місця;

в) команду, що посіла останнє місце.

6.128. Відома інформація про багаж (кількість речей і загальну вагу багажу) N пасажирів. Знайти:

а) кількість пасажирів, що мають більш ніж дві речі;

б) кількість пасажирів, у яких кількість речей більша за середню кількість речей усіх пасажирів;

в) пасажирів, багаж яких складається з однієї речі, вага якої не більша за 25 кг;

г) загальну вагу та кількість усіх речей.

6.129. Таблицю футбольного чемпіонату задано квадратною матрицею порядку n, в якій усі елементи головної діагоналі дорівнюють нулю, а кожний елемент, що не належить головній діагоналі, дорівнює числу очок, набраних у грі: 3 – виграш, 1 – нічия, 0 – програш. Скласти процедуру:

а) знаходження кількості команд, що мають перемог більше, ніж поразок;

б) визначення номерів команд, що пройшли чемпіонат без поразок;

в) пошуку команд, що виграли більше половини ігор.

6.130. Задано булеву матрицю C, елементи якої $C_{i,j} = Істина$, якщо країни i,j мають спільний кордон; $C_{i,j} = Хибність$ – у протилежному випадку. Скласти процедуру пошуку країни, що має найбільшу кількість сусідів серед перелічених країн.

6.131. Задано дійсну матрицю T, елементи якої $T_{i,m}$ означають середньомісячну температуру на острові i у місяці m. Скласти процедуру, що визначає, який місяць і на якому острові серед перелічених островів є найхолоднішим.

6.132. Задано несуперечливу таблицю спорідненості C, елементи якої $C_{i,j}$ можуть набувати значень із набору (*син*, *дочка*, *батько*, *мати*, *ні*). Наприклад, $C_{i,j} = ні$, якщо j не є ні одним із батьків, ні родичем, ні дитиною i; $C_{i,j}$=*син*, якщо j – син i тощо. Скласти процедуру пошуку

а) будь-кого з онучок;

б) будь-кого з дядьків;

в) кількості двоюрідних братів і сестер;

г) людини на ім'я n із заданого набору імен.

Розділ 7
ФАЙЛИ

Означення 7.1. Файл (англ. file – папка) – іменований блок інформації, який зберігається на носії інформації.

Файл має такі ознаки:

- фіксоване ім'я (назва файла) – послідовність символів, що однозначно характеризує файл;
- певне логічне зображення та відповідні йому операції читання/запису;
- розмір файла характеризується розміром інформації, що в ньому міститься.

Файл є найменшою одиницею збереження інформації на носії. У файлах можуть зберігатися програми, дані, тексти документів, інструкцій, закодовані зображення тощо. На відміну від змінної, файл може існувати за межами конкретної програми.

В інформатиці, крім загальних **двійкових** (бінарних) файлів, окремо розглядають **текстові**. У загальному визначенні двійковий файл – це послідовність довільних байтів; текстовий файл – різновид файлів, що містять символьні дані, організовані у рядки змінної довжини. Текстові файли виділяють окремо через значну кількість практичних задач, пов'язаних з обробкою текстів.

7.1. Двійкові файли

На відміну від багатьох мов програмування, файли у С є нетипізованими. Це означає, що у файлі міститься лише деяка послідовність байтів, а механізми коректної роботи з цією послідовністю повністю покладаються на програму, яку використовуватиме файл. Проте такий підхід може призводити до непередбачуваних наслідків, тому завжди при написанні програм обробки файлів потрібно пам'ятати та враховувати тип інформації, що міститься у них. Надалі штучно вважатимемо, що файл поділено на записи (компоненти), розмір кожного з яких відповідає розміру відповідного типу даних, що містяться або передбачаються у файлі. Цей тип даних вважатимемо типом файла. Наприклад, якщо передбачається, що файл містить дані типу `int`, то казатимемо, що маємо файл типу `int` і вважатимемо, що файл поділено на записи розміром два байти кожний.

Основні константи, структури та методи роботи з файлами описані в бібліотеці `stdio.h`.

Опис файлової змінної є описом вказівника на структуру `FILE`:

```
FILE *fp;
```

тут `fp` – вказівник, який надалі називатимемо файловою змінною.

Для використання файлів програмою необхідно виконати зв'язування змінної типу файл (файлової змінної) з тим чи іншим файлом на носії інформації. Таке зв'язування звичайно передує будь-яким діям із файлом. Крім цього, файлова змінна містить інформацію про поточне положення у файлі, так званий **маркер**. Фактично маркер містить номер поточного байта. Нумерація байтів, а відповідно й записів, у файлі починаються з нуля. При відкритті файла маркер зазвичай встановлюється на нульовий запис. Інструкція

```
fp = fopen(file_name, mode);
```

пов'язує файлову змінну `fp` із файлом на зовнішньому пристрої, що має ім'я `file_name`, і відкриває його для обробки, згідно з режимом, заданим у рядку `mode`. Рядок `mode` може мати одне із значень:

```
"rb", "wb", "ab", "r+b", "w+b", "a+b",
```

де `r` – (від "read") відкриття існуючого файла для читання; `w` – (від "write") створення файла для запису (перезапис існуючого); `a` – (від "append") відкриття існуючого файла для доповнення (маркер встановлюється у кінець файла); `b` – (від "binary") указує на те, що одна з перелічних операцій проводиться із двійковим файлом; `+` – означає можливість заміни записів у файлі. Наприклад,

```
fp = fopen("rez.dat", "wb");
fp = fopen("mod.dat", "r+b");
```

Наприкінці роботи з файлом його завжди закривають, що гарантує збереження записаної чи зміненої інформації. Інструкція

```
fclose(fp);
```

закриває файл `fp`.

Для видалення файла з носія інформації використовують функцію

```
remove(file_name);
```

яка повертає значення `0` у випадку успішного видалення файла з ім'ям `file_name` з носія інформації.

Для зберігання проміжних результатів функціонування програми або передавання даних з однієї програми до іншої користуються тимчасовими файлами. Інструкція

```
ftmp = tmpfile();
```

створює та відкриває тимчасовий файл і пов'язує файлову змінну `ftmp` із цим файлом. Після виклику для тимчасового файла інструкції `fclose` тимчасовий файл знищується.

Для читання та запису даних у файлі використовують інструкції:

```
fread(&p, sizeof(p), 1, fp);
```
читання з файла fp запису, на який указує маркер, у змінну p. Після закінчення процедури маркер указує на наступну компоненту файла.
```
fwrite(&p, sizeof(p), 1, fp);
```
запис у файл. Значення змінної p присвоюється тій компоненті файла fp, на яку вказує маркер. Якщо маркер встановлено у кінці файла, запис p додається до файла. Після виконання процедури маркер вказує на наступну компоненту файла.

Зауваження. Підпрограми fwrite/fread є булевими функціями, що повертають значення *Хибності* (тобто 0) у випадку, якщо під час запису/зчитування виникла помилка.

Інструкція
```
fseek(fp, n*sizeof(p), SEEK_SET);
```
встановлює маркер у файлі fp на запис з номером n (починаючи з 0). Значення третього параметра, крім константи SEEK_SET, що означає відлік позиції маркера від початку файла, може також бути SEEK_CUR і SEEK_END, що означає зсув позиції відносно поточного положення маркера та кінця файла відповідно.

Булева функція
```
feof(fp)
```
повертає значення істини, якщо маркер стоїть у кінці файла.

Функція
```
ftell(fp)
```
визначає позицію маркера у файлі, починаючи з 0. Слід зауважити, що ftell повертає фактично номер поточного байта, а не поточного запису. Для того, щоб знайти номер поточного запису, можна використати вираз:
```
ftell(fp)/sizeof(p)
```

Приклади розв'язання задач

Приклад 7.1. Створити програму запису у файл усіх чисел Фібоначчі, які не перевищують натуральне число n.

Розв'язок. Для розв'язання задачі опишемо допоміжну функцію Fib, що обчислює відповідний k-й член послідовності Фібоначчі. Функція WriteFibToFile записує у файл з ім'ям fname усі числа Фібоначчі, що не перевищують задане число n. Функція ReadFile виводить вміст файла fname на екран.
```
#include <stdio.h>
#include <string.h>
#include <iostream.h>
/* обчислення k-го члена послідовності Фібоначчі*/
```

```
int Fib(int k){
  int F, F1, F2;
  int i = 1;
  F1 = F2 = F = 1;
  for (i = 2; i <= k; i++){
    F  = F1 + F2;
    F2 = F1;
    F1 = F;
  }
  return F;
}
/* підпрограма запису у файл */
void WriteFibToFile(char fname[], int n){
  FILE *fp;
  fp = fopen(fname, "wb"); // відкриття для запису
  int F;
  int i = 0;
  do {
    F = Fib(i);
    /* якщо член послідовності більший за задане
       число, то припиняємо запис у файл */
    if (F > n) break;
    fwrite(&F,sizeof(F),1,fp); // запис у файл
    i++;
  } while (1);
  fclose(fp);
}
/* підпрограма зчитування записів з файла */
void ReadFile(char fname[]){
  FILE *fp;
  // відкриття для читання
  fp = fopen(fname, "rb");
  int b; // змінна контролю коректності читання
  int F;
  while ( !feof(fp) ){
    b = fread(&F, sizeof(F), 1, fp);
    /* зчитування пройшло коректно b = 1 */
    if (b)
      /* виводиться на екран F */
      cout << F << " ";
  }
  cout << '\n';
  fclose(fp);
}
```

```
void main(){
  char fname[81]; // змінна для імені файла
  int n;
  cout << "Введіть ім\'я файла: "; gets(fname);
  cout << "n = ? "; cin >> n;
  WriteFibToFile(fname, n);
  ReadFile(fname);
}
```

Приклад 7.2. Дано символьний файл. Необхідно здійснити інверсію записів цього файла.

Розв'язок

Спосіб 1. Для здійснення інверсії файла переставлятимемо місцями рівновіддалені від початку та кінця файла записи.

Оформимо розв'язок задачі у вигляді підпрограми `InverseFile`.

```
void InverseFile(char fname[]){
  FILE *fp;
  /* відкриваємо файл для читання з можливістю
     зміни записів */
  fp = fopen(fname, "r+b");
  int size = 0;
  int i;
  char c, c1;
  /* у змінній size підраховуємо кількість
     символів у файлі fp */
  while (!feof(fp))
    if (fread(&c, sizeof(c), 1, fp))
      size++;
  for (i = 0; i <= (size-1)/2;  i++){
    // i-й запис зберігаємо у змінну c
    fseek(fp, i*sizeof(char), SEEK_SET);
    fread(&c, sizeof(c), 1, fp);
    // (size-i-1)-й запис зберігаємо у змінну c1
    fseek(fp, (size-i-1)*sizeof(char), SEEK_SET);
    fread(&c1, sizeof(c1), 1, fp);
    // змінну c1 записуємо у i-й запис
    fseek(fp, i*sizeof(char), SEEK_SET);
    fwrite(&c1, sizeof(c1), 1, fp);
    // змінну c записуємо у (size-i-1)-й запис
    fseek(fp, (size-i-1)*sizeof(char), SEEK_SET);
    fwrite(&c, sizeof(c), 1, fp);
  }
  fclose(fp);
}
```

Спосіб 2. Для реалізації інверсії використаємо допоміжний файл gp, до якого записуватимемо символи файла fp, беручи їх послідовно з кінця файла. Таким чином, на відміну від підпрограми, описаної у першому способі, підпрограма InverseFile2 матиме вже два аргументи – fname і gname – відповідно імена вихідного й результуючого файлів.

```
void InverseFile2(char fname[], char gname[]){
  FILE *fp, *gp;
  // відкриваємо для читання
  fp = fopen(fname, "rb");
  // відкриваємо для запису
  gp = fopen(gname, "wb");
  int size = 0;
  char c;
  // визначаємо розмір вихідного файла
  while (!feof(fp))
    if (fread(&c, sizeof(c), 1, fp))
      size++;
  for (int i = size-1; i>=0; i--){
    // переставляємо маркер у fp на i-й символ
    fseek (fp, i*sizeof(char), SEEK_SET);
    fread (&c, sizeof(c), 1, fp);
    fwrite(&c, sizeof(c), 1, gp);
  }
  fclose(fp);
  fclose(gp);
}
```

Приклад 7.3. Описати підпрограму видалення з файла запису з номером *k*.

Розв'язок. Для видалення запису з файла використаємо тимчасовий файл, до якого перепишемо всі записи вихідного файла, крім того, який треба видалити. Далі перезапишемо вихідний файл, заповнюючи його записами з тимчасового файла.

```
void DelFromFile(char fname[], int k){
  FILE *fp, *gp;
  fp = fopen(fname, "rb");
  gp = tmpfile(); // створюємо тимчасовий файл
  int i = 0;
  char c;
  while (!feof(fp)){
    if (fread(&c, sizeof(c), 1, fp)){
      if (i != k)
        fwrite(&c, sizeof(c), 1, gp);
    }
```

```
      i++;
    }
    fclose(fp);
    // переходимо на початок файла gp
    fseek(gp, 0*sizeof(char), SEEK_SET);
    // перезаписуємо вихідний файл
    fp = fopen(fname, "wb");
    while (!feof(gp))
      if (fread(&c, sizeof(c), 1, gp))
        fwrite(&c, sizeof(c), 1, fp);
    fclose(fp);
    fclose(gp);
  }
```

Приклад 7.4. Дано символьний файл F. У файлі G побудувати множину символів файла F.

Розв'язок. Поставлена задача рівносильна такій: побудувати файл G, до якого кожен символ файла F входить рівно один раз.

Для розв'язання поставленої задачі опишемо дві допоміжні функції: `inFile` і `AddToFile`.

Функція `inFile` перевіряє входження символу `c` до файла з ім'ям `fname`. Результатом її виконання буде 1 (*Істина*) у випадку, якщо символ `c` входить до файла `fname` і 0 (*Хибність*) – у протилежному разі.

```
  int inFile(char fname[], char c){
    int b = 0;
    char c1;
    FILE *fp;
    fp = fopen(fname, "rb");
    while (!feof(fp)){
      if (fread(&c1, sizeof(c1), 1, fp))
        if (c == c1){
           b = 1;
           break;
        }
    }
    fclose(fp);
    return b;
  }
```

Підпрограма `AddToFile` додає символ `c` до файла з ім'ям `fname`.

```
  void AddToFile(char fname[], char c){
    FILE *fp;
    // відкриваємо файл fname для дописування
    fp = fopen(fname, "ab");
    fwrite(&c, sizeof(c), 1, fp);
```

```
  fclose(fp);
}
```

Далі, використовуючи вищенаведені підпрограми `inFile` та `AddToFile`, опишемо підпрограму `SetFile`, що за файлом `fname` будує файл `gname`, до якого всі символи файла `fname` входять тільки один раз.

```
void SetFile(char fname[], char gname[]){
  FILE *fp;
  fp = fopen(fname, "rb");
  char c;
  while (!feof(fp)){
    if (fread(&c, sizeof(c), 1, fp))
      if (!inFile(gname, c))
         AddToFile(gname, c);
  }
  fclose(fp);
}
```

Задачі для самостійної роботи

7.1. Скласти програму для обчислення значень многочлена, його похідної, використовуючи файл його коефіцієнтів.

7.2. Нехай множина цілих чисел задана у файлі. Визначити:

а) процедуру введення множини;

б) процедуру виведення множини;

в) процедуру доповнення множини;

г) процедуру видалення елемента з множини;

ґ) функцію, що дає відповідь, чи входить елемент до множини;

д) функцію, що дає відповідь, чи порожня множина;

е) функцію, що знаходить максимальний елемент множини;

є) функцію, що знаходить мінімальний елемент множини;

ж) процедуру об'єднання множин;

з) процедуру різниці множин;

и) процедуру перетину множин;

і) функцію обчислення ваги множини;

ї) функцію обчислення діаметра множини;

й) функцію, що за множиною A знаходить підмножину всіх таких її елементів, для яких справедлива умова $Q(x)$, $x \in A$;

к) функцію, що з'ясовує, чи є множина A підмножиною множини B;

л) функцію, що з'ясовує, чи дорівнює множина A множині B.

7.3. Дано файл, компоненти якого є записи ($koef$, st) – коефіцієнт і степінь членів полінома ($koef \neq 0$). Визначити підпрограми для виконання таких дій над поліномом:

а) введення полінома; б) друк полінома;
в) обчислення похідної від полінома;
г) обчислення невизначеного інтеграла від полінома;
ґ) упорядкування за степенями елементів полінома;
д) приведення подібних серед елементів полінома;
е) додавання, віднімання двох поліномів;
є) множення двох поліномів;
ж) знаходження частки та залишку від ділення двох поліномів;
з) знаходження полінома за лінійної заміни змінної $x = dx + c,\ d \neq 0$;
и) знаходження полінома за заміни змінної $x = d/x,\ d \neq 0$;
і) знаходження степеня полінома;
ї) з'ясування, чи має поліном корені, рівні нулю, і визначення їхньої кратності;
й) знаходження максимального за умовою $Q(t)$ коефіцієнта серед коефіцієнтів полінома, які задовольняють умову $G(t)$;
к) знаходження мінімального за умовою $Q(t)$ коефіцієнта серед коефіцієнтів полінома, які задовольняють умову $G(t)$;
л) знаходження значення полінома в заданій точці.

7.4. Використовуючи підпрограми з попередньої задачі, визначити для алгебраїчного рівняння

$$a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n = 0$$

а) нижню та верхню границі розташування коренів x_k^*:

$$1/(1 + B/|a_n|) \le |x_k^*| \le 1 + A/a_0,$$

де $A = \max(|a_1|, |a_2|, \ldots, |a_n|)$, $B = \max(|a_0|, |a_1|, \ldots, |a_{n-1}|)$, $a_0 \neq 0, a_n \neq 0$;

б) верхню границю додатних коренів x_k^* (формула Маклорена):

$$x_k^* \le 1 + (C/a_0)^{1/m},$$

де $a_0 > 0$; $C = \max(|a_i|)$; $a_i < 0$; a_m – перший серед $a_i < 0$;

в) нижню границю від'ємних коренів x_k^*. Виконати заміну змінної $x = -x$ і скористатись формулою Маклорена;

г) нижню границю додатних коренів x_k^*. Виконати заміну змінної $x = 1/x$ і скористатись формулою Маклорена;

ґ) верхню границю від'ємних коренів x_k^*. Виконати заміну змінної $x = -1/x$ і скористатись формулою Маклорена.

7.5. Дано файл, компоненти якого є дійсними числами. Скласти підпрограми для обчислення:

а) суми компонент файла;
б) кількості від'ємних компонент файла;
в) останньої компоненти файла;
г) найбільшого зі значень компонент файла;
ґ) найменшого зі значень компонент файла з парними номерами;

д) суми найбільшого та найменшого зі компонент;

е) різниці першої й останньої компоненти файла;

є) кількості компонент файла, менші за середнє арифметичне всіх його компонент.

7.6. Дано файл, компоненти якого є цілими числами. Скласти підпрограми для обчислення:

а) кількості парних чисел серед компонент;

б) кількості квадратів непарних чисел серед компонент;

в) різниці між найбільшим парним і найменшим непарним числами з компонент;

г) кількості компонент у найдовшій зростаючій послідовності компонент файла.

7.7. За масивом A, елементи якого утворені за законом

а) $a_i = i$; б) $a_i = i^2$ в) $a_i = i!$;

г) $a_i = 2^{i+1}$; ґ) $a_i = 2^i + 3^i$; $(i = 1,2,...,n)$.

побудувати файл, який містить елементи масиву A.

7.8. Послідовність $x_1, x_2, ...$ утворена за законом

$$x_i = \frac{i - 0.1}{i^3 + |\mathrm{tg}\, 2i|} \ (i = 1,2,...).$$

Дано дійсне $\varepsilon > 0$. Побудувати файл, що містив би всі члени послідовності $x_1, x_2, ...$ перед першим членом, для якого виконано: $|x_i| < \varepsilon$.

7.9. Скласти процедуру запису до файла:

а) усіх непарних чисел Фібоначчі, які не перевищують натуральне число n;

б) усіх чисел Фібоначчі, що є простими числами та не перевищують натуральне число n;

в) перших n чисел послідовності трибоначчі;

г) усіх чисел послідовності Падована, що належать відрізку $[n, m]$.

7.10. Дано файл F, компоненти якого є цілими числами. Побудувати файл G, який містив би всі компоненти файла F:

а) що є парними числами; б) що діляться на 3 і на 5;

в) що є точними квадратами; г) записані у зворотному порядку;

ґ) за винятком повторних входжень одного й того самого числа.

7.11. Використовуючи файл F, компоненти якого є цілими числами, побудувати файл G, що містить усі парні числа файла F, і файл H – усі непарні. Послідовність чисел зберігається.

7.12. Задано натуральне число n та файл F, компоненти якого є цілими числами. Побудувати файл G, записавши до нього найбільше значення перших n компонент файла F, потім – наступних n компонент тощо. Розглянути два випадки:

а) кількість компонент файла ділиться на n;

б) кількість компонент файла не ділиться на n. Остання компонента файла g має дорівнювати найбільшій із компонент файла F, які утворюють останню (неповну) групу.

7.13. Дано файл F, компоненти якого є цілими числами. Файл містить рівне число додатних і від'ємних чисел. Використовуючи допоміжний файл H, переписати компоненти файла F до файла G так, щоб у файлі G:

а) не було двох сусідніх чисел одного знака;

б) спочатку йшли додатні, потім – від'ємні числа;

в) числа йшли таким чином: два додатних, два від'ємних тощо (припускається, що число компонент у файлі F ділиться на 4).

7.14. Назвемо файл, компоненти якого є символами, символьним. За рядком S побудувати файл, що містить символи цього рядка.

7.15. Дано символьний файл F. Побудувати файл G, утворений із файла F:

а) зміною всіх його великих літер однойменними малими;

б) записом його компонент у зворотному порядку.

7.16. Дано символьний файл, що складається не менш ніж із 2 компонент. Визначити, чи є два перших символи файла цифрами. Якщо так, то виявити, чи є число, утворене цими цифрами, парним.

7.17. Задано символьні файли $F1$ і $F2$. Переписати зі збереженням послідовності компоненти файла $F1$ у $F2$, а компоненти файла $F2$ – до $F1$. Використати допоміжний файл H.

7.18. Задано символьні файли F і G. Записати до файла H спочатку компоненти файла F, потім – файла G зі збереженням порядку.

7.19. Дано символьний файл. Скласти підпрограми для:

а) додавання в його кінець заданого символу;

б) додавання в його початок заданого символу;

в) підрахунку кількості входжень до файла заданого символу;

г) визначення входження до файла заданої комбінації символів;

ґ) вилучення заданого символу;

д) вилучення інших входжень кожного символу.

7.20. Скласти функцію перевірки рівності файлів, виконаної за один перегляд їхнього змісту.

7.21. Дано файл F, компонентами якого є записи (структури) вигляду

```
struct T {
  Type_key  Key;   // ключ
  Type_data Data;  // дані
};
```

Такий файл називатимемо впорядкованим за ключами, якщо записи в ньому розташовуються в порядку зростання (спадання) ключів. Скласти процедуру пошуку запису за ключем у впорядкованому файлі.

7.22. Скласти процедуру вилучення запису із заданим ключем:

а) з упорядкованого файла; б) з невпорядкованого файла.

7.23. Дано файл F. Скласти підпрограму, що будує множину записів файла F і записує її до файла G.

7.24. Файл H називатимемо злиттям файлів F і G, якщо він містить усі записи файлів F і G, причому кожен запис входить до H рівно один раз. Скласти підпрограму злиття двох упорядкованих файлів F і G (див. задачу **7.21**) у новий упорядкований файл H.

7.25. Багаж пасажира характеризується номером пасажира, кількістю речей і їхньою загальною вагою. Дано файл пасажирів, який містить прізвища пасажирів, і файл, що містить інформацію про багаж кількох пасажирів (номер пасажира – це номер запису у файлі пасажирів). Скласти процедури для:

а) знаходження пасажира, у багажі якого середня вага однієї речі відрізняється не більш ніж на 1 кг від загальної середньої ваги речей;

б) визначення пасажирів, які мають більше двох речей, і пасажирів, кількість речей у яких більша за середню кількість речей;

в) видачі відомостей про пасажира, кількість речей у багажі якого не менша, ніж у будь-якому іншому багажі, а вага речей – не більша, ніж у будь-якому іншому багажі із цією самою кількістю речей;

г) визначення, чи мають принаймні два пасажири багажі, які не відрізняються за кількістю речей і відрізняються вагою не більш ніж на 1 кг (якщо такі пасажири є, то показати їхні прізвища);

ґ) визначення пасажира, багаж якого складається з однієї речі вагою не менше 30 кг.

7.26. Відомості про учня складаються з його імені, прізвища та назви класу (рік навчання та літери), в якому він вчиться. Дано файл, який містить відомості про учнів школи. Скласти підпрограми, які дозволяють:

а) визначити, чи є в школі учні з однаковим прізвищем;

б) визначити, чи є учні з однаковим прізвищем у паралельних класах;

в) визначити, чи є учні з однаковим прізвищем у певному класі;

г) відповісти на питання а)-в) стосовно учнів, у яких збігаються ім'я та прізвище;

ґ) визначити, в яких класах налічується більше 35 учнів;

д) визначити, на скільки учнів у восьмих класах більше, ніж у десятих;

е) зібрати у файл відомості про учнів 9-10-х класів, розташувавши спочатку відомості про учнів класу 9 а, потім – 9 б тощо;

є) отримати список учнів даного класу за зразками:

Прізвище Ім'я
Прізвище І.
І.Прізвище.

7.27. Дано файл, який містить ті самі відомості про учнів школи, що й в попередній задачі, і додатково оцінки, отримані учнями на іспитах із заданих предметів. Скласти процедури для:

а) визначення кількості учнів, які не мають оцінок, нижче 4;

б) побудови файла, який містить відомості про кращих учнів школи, що мають оцінки, не нижче 4;

в) друкування відомостей про учнів, які мають принаймні одну незадовільну оцінку, у вигляді прізвища та ініціалів, назви класу, предмета.

7.28. Відомості про автомобіль складаються з його марки, номера й прізвища власника. Дано файл, який містить відомості про кілька автомобілів. Скласти процедури знаходження:

а) прізвищ власників номерів автомобілів певної марки;

б) кількості автомобілів кожної марки.

7.29. Дано файл, який містить різні дати, кожна з яких – це число, місяць і рік. Скласти процедури пошуку:

а) року з найменшим номером; б) усіх весняних дат;

в) найпізнішої дати.

7.30. Дано файл, який містить відомості про книжки. Відомості про кожну книгу – це прізвище автора, назва та рік видання. Скласти процедури пошуку:

а) назв книг певного автора, виданих із 1960 р.;

б) книг із заданою назвою. Якщо така книжка є, то надрукувати прізвища авторів і рік видання.

7.31. Дано файл, який містить номери телефонів співробітників установи: вказуються прізвище співробітника, його ініціали та номер телефону. Визначити процедуру пошуку телефону співробітника за його прізвищем та ініціалами.

7.32. Дано файл з відомостями про кубики: розмір кожного (довжина ребра у см), його колір (червоний, жовтий, зелений, синій) і матеріал (дерев'яний, металевий, картонний). Скласти процедури пошуку:

а) кількості кубиків кожного з перелічених кольорів, їх сумарний об'єм;

б) кількості дерев'яних кубиків із ребром 3 см і металевих кубиків із ребром, більшим за 5 см.

7.33. Дано файл, який містить відомості про речовину: назва речовини, її питома вага, провідність (провідник, напівпровідник, ізолятор). Скласти процедури для:

а) знаходження питомої ваги та назви всіх провідників;

б) вибору даних про напівпровідники й таке їх упорядкування, за якого кожний наступний був із меншою питомою вагою, ніж попередній.

7.34. Дано файл, що містить відомості про експортовані товари. Указано назву товару, країну-імпортера та обсяг партії у штуках. Скласти процедуру пошуку країн, до яких експортується цей товар, і загальний обсяг його експорту.

7.35. Задано два файли $F1$ і $F2$, перший з яких – інвентарний – містить відомості про кількість виробів і видів продукції, що зберігається на складі (вид продукції задається його порядковим номером). Файл $F2$ – допоміжний, що містить відомості про те, наскільки зменшилась чи збільшилась кількість виробів за деякими видами продукції. Допоміжний

файл може містити деякі відомості щодо продукції одного виду або не містити жодних відомостей. Скласти процедуру оновлення інвентарного файла за допоміжним, побудувавши новий файл G.

7.36. Дано файл, який містить відомості про іграшки: указано назву іграшки (напр., м'яч, лялька, конструктор тощо), її вартість у гривнях і вікові межі для дітей, яким іграшка призначається (напр., для дітей від двох до п'яти років). Скласти процедури:

а) пошуку назв іграшок, вартість яких не перевищує 40 грн, призначених дітям п'яти років;

б) пошуку назв іграшок, призначені дітям і чотирьох, і десяти років;

в) пошуку назв найдорожчих іграшок (ціна яких відрізняється від ціни найдорожчої іграшки не більш ніж на 50 грн);

г) визначення ціни найдорожчого конструктора;

ґ) визначення ціни всіх кубиків;

д) пошуку двох іграшок, що призначені дітям трьох років, сумарна вартість яких не перевищує 20 грн;

е) пошуку конструктора ціною 22 грн, призначеного дітям від п'яти до десяти років. Якщо такої іграшки немає, то занести відомості про її відсутність до файла.

7.37. Дано файл, який містить відомості про прямокутники: указано номер прямокутника у файлі, координати верхнього лівого кута, нижнього правого кута прямокутника. Скласти процедуру пошуку прямокутника з найбільшою площею й визначення цієї площі.

7.38. У двох файлах міститься таблиця футбольного турніру, у першому – записано назви команд; у другому – результати матчів, що зберігаються у записах типу `T_Match`

```
typedef struct {
  unsigned int n1, n2;
  unsigned int b1, b2;
} T_Match;
```

Тут у структурі типу `T_Match` поля `n1`, `n2` – номери першої і другої команд (тобто номери назв команд у файлі команд); `b1`, `b2` – кількість м'ячів, забитих першою та другою командами, відповідно.

Кожній команді за перемогу нараховується 3 очки, за нічию – 1, за поразку – 0.

Із двох команд, які мають однакову кількість очок, першою вважається та, що

1) має кращу різницю забитих і пропущених м'ячів;

2) за однакової різниці має більше забитих м'ячів;

3) за всіма однаковими попередніми показниками визначається жеребкуванням (для жеребкування використати генератор випадкових чисел).

Знайти команду, яка є лідером.

Указівка. Описати підпрограми створення файлів команд і матчів, додавання результату матчу, визначення лідера.

7.2. Текстові файли

Текстові файли у мові С описують так само, як двійкові файли. Указати, що файл текстовий можна під час відкриття файла за допомогою інструкції `fopen`:

```
fp = fopen(file_name, mode);
```

де `file_name` – ім'я текстового файла, а рядок `mode` для текстових файлів може мати одне зі значень:

```
"r", "w", "a",
```

де `r` – означає відкриття існуючого файла для читання; `w` – створення файла для запису; `a` – відкриття існуючого файла для доповнення. Наприклад,

```
fp = fopen("rez.txt", "w");
fp = fopen("mod.txt", "r");
```

Для текстових файлів залишаються також визначеними підпрограми `fclose` і `feof`.

Читання текстової інформації із текстового файла здійснюється за допомогою підпрограм:

```
c = fgetc(fp);
```

прочитати з файла `fp` поточний символ у змінну `c`.

```
fgets(str, n, fp);
```

читати з файла `fp` не більше `n-1` символів і присвоїти результат рядку `str`. Якщо до кінця рядка файла залишилось менше `n-1` символів, то буде прочитано тільки символи до кінця рядка та символи кінця рядка.

Запис у текстовий файл текстової інформації здійснюється підпрограмами

```
fputc(c, fp);
fputs(str, fp);
```

Перша із них записує у файл `fp` символ `c`, друга – рядок `str`.

Для читання/запису з файла/у файл інформації, що є не обов'язково текстовою, використовують функції `fscanf`/`fprintf`:

```
fscanf(fp, "format", variables);
fprintf(fp, "format", variables);
```

Ці функції є аналогами інструкцій `scanf`/`printf`. Правила використання також аналогічні до `scanf`/`printf` (див. ст. 6). Єдина відмінність полягає у тому, що інструкція `scanf` зчитує змінні (з переліку) з клавіатури, а інструкція `fscanf` – з файла `fp`. Відповідно `printf` виводить змінні (з переліку) на екран, а `fprintf` – у файл `fp`.

Приклади розв'язання задач

Приклад 7.5. Замінити у текстовому файлі всі символи *c* рядком *a*.

Розв'язок. У програмі визначимо константи MAXLEN і MAXREPL. Перша з них відповідає за максимальну довжину рядків файла, друга – за максимальну довжину рядка *a*. Глобальні змінні fname і gname призначено для імен вхідного і результуючого файлів.

```
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#define MAXLEN 256
#define MAXREPL 10
char fname[81], gname[81];
/* Підпрограма, що замінює у рядку s всі символи c
   рядком a і результат записує у рядок r */
void replace(char s[],char a[],char c,char r[]){
  unsigned i;
  strcpy(r, "");
  for (i=0; i < strlen(s); i++)
    if (s[i] == c) strcat(r,a);
    else {
       r[strlen(r) + 1] = '\0';
       r[strlen(r)] = s[i];
    }
}
/* Підпрограма, що замінює у вхідному файлі
   всі символи c рядком a і результат записує
   у результуючий файл */
void replace_text(char c, char a[]){
  FILE *fp, *gp;
  char s[MAXLEN], s1[MAXLEN*MAXREPL];
  fp = fopen(fname, "r");
  gp = fopen(gname, "w");
  while(1) {
     fgets(s,MAXLEN,fp);
     if (feof(fp)) break;
     replace(s,a,c,s1);
     fputs(s1,gp);
  }
  fclose(fp); fclose(gp);
}
void main(){
  char c, a[MAXREPL];
  cout << "Ім'я вхідного файла: ";
```

```
  gets(fname);
  cout << "Ім'я результуючого файла: ";
  gets(gname);
  cout << "Рядок заміни: ";
  gets(a);
  cout << "Символ для заміни: ";
  c = getchar(); cout << "\n";
  replace_text(c,a);
}
```

Задачі для самостійної роботи

7.39. Дано текстовий файл. Групи символів, що відокремлені пропусками (одним або кількома) і не містять пропусків усередині, називатимемо словами. Скласти підпрограми для:

а) знаходження найдовшого слова у файлі;

б) визначення кількості слів у файлі;

в) вилучення з файла зайвих пропусків і всіх слів, що складаються з однієї літери;

г) вилучення всіх пропусків на початку рядків, у кінці рядків і між словами (крім одного);

ґ) вставки пропусків до рядків рівномірно між словами так, щоб довжина всіх рядків (якщо в них більше 1 слова) була 80 символів і кількість пропусків між словами в одному рядку відрізнялась не більш ніж на 1 (вважати, що рядки файла мають не більш ніж 80 символів).

Результат записати до файла H.

7.40. Визначити процедуру, яка утворює текстовий файл із 9 рядків, у першому з яких одна літера `'1'`, у другому – дві літери `'2'`, ... , у дев'ятому – дев'ять літер `'9'`.

7.41. Визначити процедуру, яка за заданою послідовністю символів формує текстовий файл із рядками по 40 літер (в останньому рядку літер може бути й менше).

7.42. Визначити функцію, яка:

а) підраховує кількість порожніх рядків;

б) обчислює максимальну довжину рядків текстового файла.

7.43. Визначити процедуру виведення:

а) усіх рядків текстового файла;

б) рядків, які містять більше 60 символів.

7.44. Визначити функцію, що визначає кількість рядків текстового файла:

а) починаються із заданого символу;

б) закінчуються заданим символом;

в) починаються й закінчуються одним і тим самим символом;

г) що складаються з однакових символів.

7.45. Визначити процедуру, яка переписує до текстового файла G усі рядки текстового файла F:

а) із заміною в них символа `'0'` на `'1'`, і навпаки;

б) в інвертованому вигляді.

7.46. Визначити процедуру пошуку найдовшого рядка в текстовому файлі. Якщо таких рядків кілька, знайти перший із них.

7.47. Визначити процедуру, яка переписує компоненти текстового файла F до файла G, вставляючи до початку кожного рядка один символ пропуску. Порядок компонент не має змінюватися.

7.48. Визначити процедуру, яка друкує за рядками зміст текстового файла, вставляючи до початку кожного рядка його порядковий номер і символ пропуску.

7.49. Визначити процедуру пошуку в текстовому файлі рядків, фрагментом яких є заданий рядок.

7.50. Вважаючи, що довжина рядка текстового файла F не перевищує 80, визначити процедуру, яка, доповнюючи короткі рядки файла F пропусками праворуч, формує текстовий файл G, усі рядки якого мають довжину 80.

7.51. Визначити процедуру, яка, виключаючи пропуски, що стоять на початку рядків текстового файла F, формує текстовий файл G.

7.52. У текстовому файлі записано непорожню послідовність дійсних чисел, які розділяються пропусками. Визначити функцію обчислення найбільшого з цих чисел.

7.53. У текстовому файлі F записано послідовність цілих чисел, які розділяються пропусками. Визначити процедуру запису до текстового файла g усіх додатних чисел із F.

7.54. У текстовому файлі кожний рядок містить кілька натуральних чисел, які розділяються пропусками. Числа визначають вигляд геометричної фігури (номер) та її розміри. Прийнято такі домовленості:

- відрізок прямої задається координатами своїх кінців і має номер 1;
- прямокутник задається координатами верхнього лівого й нижнього правого кутів і має номер 2;
- коло задається координатами центра й радіусом і має номер 3.

Визначити процедури обчислення:

а) відрізка з найбільшою довжиною;

б) прямокутника з найбільшим периметром;

в) кола з найменшою площею.

7.55. Скласти програму для перевірки правильності розстановки фігурних дужок у текстовому файлі, що є файлом програми, написаної С.

Розділ 8
СОРТУВАННЯ ТА ШВИДКИЙ ПОШУК

У практиці програмування дуже часто виникає необхідність упорядкування елементів структур даних за деякою ознакою, що називають "ключем". У цьому випадку кажуть, що необхідно відсортувати множину елементів, використовуючи задане відношення порядку.

Означення 8.1. **Сортуванням** називають операцію впорядковування елементів деякої структури даних, на якій визначено відношення порядку, за ключами.

Залежно від того, чи містяться елементи структур даних у внутрішній (оперативній) або у зовнішній (на зовнішніх пристроях) пам'яті, розрізняють **внутрішнє** та **зовнішнє сортування**.

Найбільш поширені алгоритми внутрішнього сортування – це:

- сортування включенням або вставкою;
- сортування вибором;
- обмінне сортування, відоме під назвою "метод бульбашки";
- сортування злиттям;
- швидке сортування з використанням рекурсії.

Серед алгоритмів зовнішнього сортування найуживанішим є алгоритм сортування злиттям з використанням двох-трьох допоміжних файлів.

Інший клас важливих алгоритмів, що використовуються на практиці, – це алгоритми, які реалізують різні методи пошуку заданого елемента серед множини компонент визначеної структури даних. Раніше ми розглядали лінійний пошук, що не накладає жодних умов на елементи, серед яких здійснюється пошук, за винятком існування механізму послідовного доступу до них. Саме це робить лінійних пошук непридатним для використання з великими обсягами даних.

Додаткову інформацію про механізми сортування і пошуку можна отримати з додаткової літератури, наприклад, [12].

Розглянемо на прикладах основні алгоритми сортування масивів і файлів, а також механізм швидкого бінарного пошуку.

Зауваження. Нагадаємо, що нумерація елементів масиву в С завжди починається з 0.

Приклади розв'язання задач

Зауваження. Тут і далі при написанні підпрограм вважатимемо, що масив містить N елементів, і кожна програма містить рядок макрозаміни на кшталт

```
#define N 5
```

Приклад 8.1. Дано масив a цілих чисел, упорядкованих за зростанням. Серед елементів цього масиву знайти число x.

Розв'язок. Оскільки елементи масиву впорядковані, то для пошуку необхідного елемента можна скористатися, наприклад алгоритмом **бінарного пошуку**. Ідея алгоритму: на кожному кроці шукаємо елемент x серед елементів масиву $(a_0, a_1, ..., a_{N-1})$ від верхньої до нижньої границі. Спочатку нижня границя індексів – 0 (перший елемент масиву), а верхня $N-1$ (останній елемент).

На кожному кроці коло елементів, серед яких шукаємо x, зменшуємо вдвічі. Для цього порівнюємо x із середнім елементом частини масиву від нижньої до верхньої границі та змінюємо значення нижньої або верхньої границі індексу. Отже, підпрограма бінарного пошуку матиме вигляд:

```
int BinarySearch(int a[], int x){
  int l, r, m;
  l = 0; r = N - 1;
  while (l < r) {
    m = (r + l) / 2; // вибираємо середній елемент
    if (a[m]  < x)
       l = m + 1;   // змінюємо нижню границю
    else
       r = m;       // змінюємо верхню границю
  }
  if (a[l] == x)
     return l;
  else
     return -1;
}
```

Приклад 8.2. Упорядкувати методом включення масив цілих чисел за зростанням елементів.

Розв'язок. **Сортування включенням** (**вставкою**) передбачає $(N-1)$ кроків. Алгоритм сортування:

а) припускаємо, що для деякого i, починаючи з другого елемента ($i = 1$), у послідовності елементів $(a_0, a_1, ..., a_{i-1}, a_i, ..., a_{N-1})$ підпослідовність $(a_0, a_1, ..., a_{i-1})$ вже відсортовано;

б) вибираємо i-й елемент a_i і переносимо його до вже відсортованої послідовності $(a_0, a_1, ..., a_{i-1})$, вставляючи його на своє місце.

в) проробляємо вищеописані кроки а) та б) для всіх i від 1 до $N-1$.

Залежно від того, як відбувається включення елемента a_i до підпослідовності $(a_0, a_1, ..., a_{i-1})$, розрізняють **пряме** та **бінарне** включення.

При сортуванні прямим включенням елемент a_i послідовно порівнюється з елементами $(a_0, a_1, ..., a_{i-1})$ доки не буде знайдено такий елемент a_j, що $a_j \le a_i$ ($j = 0, ..., i-1$). При цьому всі елементи масиву (від $j + 1$ до $i - 1$) зсуваються на одну позицію праворуч, а на j+1 позицію вставляється елемент a_i.

Наступна підпрограма `StraightInsertionSort` здійснює сортування масиву a методом прямого включення:

```
void StraightInsertionSort(int a[]){
  int i, j, x;
  for (i = 1; i <= N-1; i++){
    x = a[i];
    j = i-1;
    // наступний цикл шукає елемент a[j] <= a[i]
    while (j >= 0){
      if (a[j] <= x)
        break; // a[j] знайдено - припиняємо пошук
      else
        a[j+1] = a[j]; // зсуваємо елементи масиву
      j = j--;
    }
    a[j+1] = x; // на j+1-шу позицію ставимо a[i]
  }
```

При сортуванні бінарним включенням для пошуку елемента a_j серед елементів $(a_0, a_1, ..., a_{i-1})$ такого, що $a_j \le a_i$ використовується бінарний пошук.

Підпрограма `BinaryInsertionSort` здійснює сортування масиву a методом бінарного включення:

```
void BinaryInsertionSort(int a[]){
  int i, j, l, r, m, x;
  for (i = 1; i <= N-1; i++){
    x = a[i];
    l = 0; r = i;
    // здійснюємо бінарний пошук
    while (l < r){
      m = (l + r) / 2;
      if (a[m] <= x)
        l = m + 1;
      else
        r = m;
    }
```

```
      // зсув елементів масиву на один вправо
      for (j = i; j >= r+1; j--)
        a[j] = a[j-1];
      // вставка a[i] у потрібну позицію
      a[r] = x;
    }
  }
```

Приклад 8.3. Методом сортування вибором впорядкувати масив цілих чисел за зростанням елементів.

Розв'язок. При **сортуванні вибором** на кожному кроці для всіх i від 0 (першого елемента) до $N-2$ (передостаннього) знаходимо елемент a_j – найменший серед елементів підпослідовності $(a_i,...,a_{N-1})$. Після цього міняємо місцями елементи a_i та a_j.

Підпрограма `SelectionSort` здійснює сортування масиву a вибором:

```
void SelectionSort(int a[]){
  int i, j, k, x;
  for (i = 0; i <= N - 2; i++){
    // пошук мінімального серед a[i],... ,a[N-1]
    j = i;
    for (k = i+1; k <= N-1; k++)
      if (a[k] <= a[j])
        j = k;
    // міняємо місцями a[i] і a[j]
    x = a[j]; a[j] = a[i]; a[i] = x;
  }
}
```

Приклад 8.4. Упорядкувати масив цілих чисел, використовуючи обмінне сортування.

Розв'язок. Алгоритм **обмінного сортування** базується на порівнянні пар сусідніх елементів масиву. За необхідності елементи у парі міняються місцями. Так продовжується до впорядкування всіх елементів. Цей метод також відомий як метод "бульбашки", оскільки за один крок зовнішнього циклу найменший ("найлегший") елемент серед розглянутих елементів масиву переміщується до початку масиву (немов би "спливає" нагору).

```
void BubbleSort(int a[]){
  int i, j, x;
  for (i = 1; i <= N-1; i++)
    for (j = N-1; j >= i; j--)
      if (a[j-1] > a[j]){
        x = a[j-1]; a[j-1] = a[j]; a[j] = x;
      }
```

```
}
```

Приклад 8.5. Використовуючи сортування злиттям, упорядкувати масив цілих чисел.

Розв'язок. Для початку розглянемо задачу злиття двох упорядкованих масивів $a=(a_0,a_1,...,a_n)$ та $b=(b_0,b_1,...,b_m)$ у впорядкований масив $c=(c_0,c_1,...,c_{n+m})$, що містить усі елементи масивів a та b. Цю задачу розв'язує фрагмент програми:

```
i = j = k = 0;
while (i <= n && j <= m) {
  if (a[i] < b[j]){
       c[k] = a[i]; i++;
  }
  else {
    c[k] = b[j]; j++;
  }
  k++;
}
while (i <= n) {
  c[k] = a[i]; i++;
  k++;
}
while (j <= m) {
  c[k] = b[j]; j++;
  k++;
}
```

Для кожного конкретного випадку другий або третій цикл `while` не виконується жодного разу, залежно від того, який масив вичерпується раніше a чи b.

При **сортуванні злиттям** на i-му кроці масив a розбивається на впорядковані підмасиви (послідовності елементів масиву, що йдуть поспіль) довжини $l=2^{i-1}$. Пари сусідніх підмасивів зливаються у впорядковані підмасиви довжиною $2l$ у допоміжному масиві b. Після присвоєння масиву a значення масиву b і збільшення l удвічі, злиття повторюється для підмасивів більшого розміру. Процес завершується, коли l стає більшим або рівним $N-1$. Оскільки при $i=1$ підмасиви, що складаються з одного елемента, упорядковані за означенням, то у результаті отримуємо впорядкований масив a.

Для реалізації цього алгоритму опишемо допоміжну підпрограму `Merge`, яка зливає пари впорядкованих підмасивів довжини `Size` масиву `a` та результат записує у масив `b`. У ній `r1` – права границя індексів першого підмасиву пари, `r2` – права границя індексів другого підмасиву пари. У підпрограмі врахуємо, що, оскільки `Size` набуває значення

степені 2, а $N - 1$ може не бути кратним степені 2, то довжина підмасивів останньої пари може бути меншою за Size.

```
void Merge(int a[], int Size, int b[]){
  int i, j, k, r1, r2;
  k = 0;
  while (k <= N-1){
    //визначення границь підмасивів
    i = k;
    r1 = i + Size - 1;
    if (r1 > N-1) r1 = N-1;
    j = r1 + 1;
    r2 = j + Size - 1;
    if (r2 > N-1) r2 = N-1;
    // злиття пари підмасивів
    while (i <= r1 && j <= r2) {
      if (a[i] < a[j]){
          b[k] = a[i]; i++;
      }
      else {
        b[k] = a[j]; j++;
      }
      k++;
    }
    while (i <= r1) {
      b[k] = a[i]; i++;
      k++;
    }
    while (j <= r2) {
      b[k] = a[j]; j++;
      k++;
    }
  }
}
```

Оскільки у С немає механізмів копіювання масивів як цілісних структур, то опишемо допоміжну підпрограму ArrayCopy, що копіює масив b у масив a.

```
void ArrayCopy(int a[], int b[]){
    for (int j = 0; j <= N-1; j++)
       a[j] = b[j];
}
```

Використовуючи вищенаведені допоміжні підпрограми, можна записати підпрограму сортування у простому вигляді:

```
void MergeSort(int a[]){
```

```
    int Size;
    int b[N];
    Size = 1;
    do {
      Merge(a, Size, b);
      ArrayCopy(a, b);
      Size *= 2;
    } while (Size < N-1);
  }
```

Зауважимо, що копіювання масиву підпрограмою ArrayCopy за великих розмірів масивів відбиратиме багато часу під час сортування. Тому модифікуємо вищенаведену підпрограму. Для цього введемо лічильник i викликів підпрограми Merge. Тоді на непарних кроках виконуватимемо злиття підпослідовностей з масиву a у масиві b, а на парних – навпаки. Тоді підпрограма матиме вигляд:

```
  void MergeSort(int a[]){
    int i, Size;
    int b[N];
    i = 0; Size = 1;
    do {
      i++;
      if (i % 2 == 1)
        Merge(a, Size, b);
      else
        Merge(b, Size, a);
      Size *= 2;
    } while (Size < N-1);
    if (i % 2 == 1) ArrayCopy(a,b);
  }
```

Приклад 8.6. Описати рекурсивну підпрограму швидкого сортування масиву.

Розв'язок. Перш ніж описати підпрограму сортування, розглянемо процес поділу масиву. Виберемо деякий елемент масиву x і знайдемо ліворуч від нього елемент $a_i > x$, а праворуч – $a_j < x$. Продовжимо перегляд та обмін значень елементів, доки нарешті у лівій частині масиву не опиняться всі елементи, менші за x, а у правій – більші за x.

Швидке сортування полягає у застосуванні поділу спочатку до всього масиву, потім – до лівої і правої частин масиву, далі – для частин цих частин тощо, доки кожна із частин не складатиметься тільки з одного елемента (отже, буде відсортована). Як бачимо, цей процес можна описати з використанням рекурсії. Рекурсивна підпрограма Sort сортує елементи масиву a, починаючи з елемента з номером l і закінчуючи елементом з номером r.

```
void Sort(int a[], int l, int r){
  int i, j, x, y;
  i = l;
  j = r;
  x = a[(l + r) / 2];
  while (1){
    while (a[i] < x)
      i = i + 1;
    while (a[j] > x)
      j = j - 1;
    if (i > j) break;
    y = a[i];
    a[i] = a[j];
    a[j] = y;
    i = i+1;
    j = j-1;
  }
  if (l<j)
    Sort(a,l,j);
  if (i<r)
    Sort(a,i,r);
}
```

Щоб відсортувати весь масив, опишемо підпрограму `QuickSort`, що викликає підпрограму `Sort` для всіх елементів масиву `a`.

```
void QuickSort(int a[]) {
  Sort(a, 0, N-1);
}
```

Приклад 8.7. Дано файл цілих чисел. Упорядкувати його за зростанням.

Розв'язок. Файл, як правило, має достатньо великий розмір, отже його не можна розмістити в оперативній пам'яті у вигляді масиву, щоб відсортувати одним із методів із попередніх прикладів. Для сортування файлів застосовують методи зовнішнього сортування. Найчастіше для сортування файлів застосовують **метод природного злиття**, що використовує для сортування файла два допоміжних файли.

Нехай необхідно відсортувати файл $f1$. Його розбивають на два файли $f2$ і $f3$, які складаються зі зростаючих підпослідовностей елементів файла $f1$. Першу зростаючу підпослідовність записують до файла $f2$, другу – до файла $f3$, третю – знову до $f2$ і т.д. Після завершення розбиття виконують злиття впорядкованих підпослідовностей із файлів $f2$ і $f3$ до файла $f1$. Спочатку зливають першу пару підпослідовностей з $f2$ і $f3$, потім – другу і т.д. Злиття відбувається аналогічно до злиття для масивів (див. приклад **8.5**) за винятком того, що довжина послідовностей нефіксована. Після завершення злиття знову виконують розбиття $f1$ на

*f*2 і *f*3, потім – знову злиття. Процес завершують, коли у файлі *f*1 утворюється одна зростаюча підпослідовність.

Надалі вважатимемо, що змінні f1, f2 і f3 містять імена файлів *f*1, *f*2 та *f*3.

Розбиття файла f1 на файли f2 і f3 оформимо у вигляді підпрограми Divide, де змінні C, A, B – файлові змінні для файлів f1, f2 і f3, відповідно. Змінна x – попередній запис файла C, y – поточний запис, ToA – змінна, що вказує, у який файл (A чи B) проводити запис.

```
void Divide(char f1[], char f2[], char f3[]){
  int x, y,
  int ToA = 1; // Спочатку запис робимо у файл A
  FILE *A, *B, *C;
  C = fopen(f1, "rb");
  A = fopen(f2, "wb");
  B = fopen(f3, "wb");
  // файл C непорожній, тому можемо читати
  fread(&x, sizeof(x), 1, C);
  while (!feof(C)){
    if (ToA)
      fwrite(&x, sizeof(x), 1, A);
    else
      fwrite(&x, sizeof(x), 1, B);
    if (!fread(&y, sizeof(y), 1, C))
       break;
    if (y < x)
// завершилась зростаюча підпослідовність}
      ToA = !ToA;
    x = y;
  }
  fclose(A); fclose(B); fclose(C);
}
```

Злиття файлів A та B у n послідовностей у файлі C оформимо у вигляді процедури Merge, де x – останній внесений до файла C запис; y – поточний запис, що необхідно писати до файла C; xa – останній прочитаний запис файла A; xb – останній прочитаний запис файла B; EndOfA, EndOfB – булеві змінні, які вказують на те, що не тільки файл (A або B) закінчився, а й усі записи цього файла оброблено та записано до файла C; FromA – змінна, що вказує, з якого файла слід взяти наступний запис для файла C. Якщо обидва файли не оброблено, треба брати запис файла A в одному із трьох випадків:

1) якщо він менший за відповідний запис файла B і більший – за останній, записаний у C;

2) якщо він менший за відповідний запис файла B і відповідний запис

файла B менший за останній, записаний у C (означає кінець впорядкованої послідовності);

3) якщо він більший за останній, записаний у C, а відповідний запис файла B менший за останній, записаний у C.

Такий підхід дозволяє отримати у файлі C зростаючі послідовності максимально можливої довжини.

```
void Merge(char f1[],char f2[],char f3[],int *n){
  int xa,xb,x,y;
  int EndOfA, EndOfB, FromA;
  FILE *A, *B, *C;
  C = fopen(f1, "wb");
  A = fopen(f2, "rb");
  B = fopen(f3, "rb");
  *n = 1;
  // читаємо з файлів A та B перші записи
  EndOfA = !fread(&xa, sizeof(x), 1, A);
  EndOfB = !fread(&xb, sizeof(x), 1, B);
  x = xa;
  if (!EndOfB && xb < xa)
      x = xb;
// зливаємо пари впорядкованих послідовностей
// з файлів A та B у файл C
  while (!EndOfA || !EndOfB){
    if (EndOfA)
      FromA = 0;
    else if (EndOfB)
      FromA = 1;
    else
      FromA = xa <= xb && xa >= x ||
              xa <= xb && xb < x  ||
              xa >= x  && xb < x;
    if (FromA){
      y = xa;
      EndOfA = !fread(&xa, sizeof(x), 1, A);
    }
    else {
      y = xb;
      EndOfB = !fread(&xb, sizeof(x), 1, B);
    }
    if (y < x)
       *n = *n+1;
    fwrite(&y, sizeof(x), 1, C);
    x = y;
```

```
  }
  fclose(A); fclose(B); fclose(C);
}
```

Опишемо ще одну допоміжну функцію `IsEmptyFile` – це булева функція, що перевіряє, чи є файл `f` порожнім.

```
int IsEmptyFile(char f[]){
  FILE *A;
  int a, ferr;
  A = fopen(f, "rb");
  ferr = !fread(&a, sizeof(a), 1, A);
  fclose(A);
  return ferr;
}
```

Використовуючи вищенаведені підпрограми `IsEmptyFile`, `Divide` та `Merge`, можна описати підпрограму `SortFile` впорядкування файла `f1`.

```
void SortFile(char f1[]){
  char f2[81] = "f2.dat"; // 1-й допоміжний файл
  char f3[81] = "f3.dat"; // 2-й допоміжний файл
  int n;
  if ( !IsEmptyFile(f1) ){
    do {
      Divide(f1, f2, f3);    // розбиваємо f1
      Merge(f1, f2, f3, &n); // зливаємо f2 та f3
    } while (n > 1);
    // видаляємо допоміжні файли
    remove(f2); remove(f3);
  }
}
```

Задачі для самостійної роботи

8.1. Дано вектор a з n цілих компонент. Знайти зростаючу підпослідовність компонентів вектора найбільшої довжини.

8.2. Визначити процедуру впорядкування рядків дійсної матриці за:

а) неспаданням їхніх перших елементів;

б) незростанням сум їхніх елементів;

в) неспаданням їхніх найменших елементів;

г) незростанням їхніх найбільших елементів.

Використати методи обмінного сортування та сортування злиттям.

8.3. Дано масив з n точок площини, заданих своїми координатами. Упорядкувати точки за неспаданням відстані від початку координат.

8.4. Дано текстові файли F і G. Відомо, що кількість слів у файлі F небільша за n, де n – відома константа. Перевірити, чи всі слова з файла G входять до файла F.

Указівка. Використати масив рядків A з n елементів, до якого записати всі різні слова файла F, упорядковані за зростанням. Використати бінарний пошук для пошуку слова з файла G у масиві A.

8.5. За умов попередньої задачі визначити кількість входжень кожного слова файла F до файла G.

Указівка. У масиві A із попередньої задачі зберігати, разом зі словом, кількість його входжень до файла G.

8.6. Дано текстовий файл F. Відомо, що кількість слів у файлі не більша від n, де n – відома константа. Побудувати за файлом F пару файлів G і H так, щоб у компонентах файла G зберігались усі різні слова файла F, упорядковані за алфавітом, а у файлі H – цілі числа. Причому i-те число файла H – це номер i-го слова з файла F у файлі G.

Указівка. Використати масив рядків A з n елементів, до якого записати всі різні слова файла F, упорядковані за зростанням.

8.7. Нехай файли F і G, компонентами яких є цілі числа, упорядковано за неспаданням. Отримати у файлі H усі числа з файлів F і G без повторень. Файл H має бути впорядкований за зростанням.

8.8. Дано файл F, компонентами якого є цілі числа. Отримати у файлі G усі непарні числа, що входять до F. Числа у файлі G мають бути розташовані у порядку:

а) незростання;

б) спадання без повторень.

8.9. Дано текстовий файл F. Записати до файла G усі різні слова файла F у порядку зростання.

Розділ 9
МОДУЛЬНА СТРУКТУРА ПРОГРАМ

Один з основних принципів у сучасному програмуванні, що дозволяє спростити розробку та підвищити універсальність програм – принцип модульності.

Означення 9.1. Модульність – принцип, згідно з яким програмний засіб розділяється на окремі іменовані блоки, що називаються модулями.

Означення 9.2. Модулем у програмуванні називається функціонально завершений фрагмент програми, оформлений у вигляді окремого файла з вихідним кодом і призначений для використання в інших програмах.

Модулі дозволяють розділяти складні задачі на дрібніші. Вони проектуються так, щоб надати програмісту зручний для багаторазового використання функціонал (інтерфейс) у вигляді набору підпрограм, констант, змінних, типів тощо.

Характерною рисою модуля є **інкапсуляція** – приховування певної частини модуля від використання ззовні. Для цього модуль поділяється на дві частини: видиму (інтерфейсну) та частину реалізації. Усі описи, указані в інтерфейсній частині модуля, доступні ззовні. Частина ж реалізації модуля є внутрішньою та недоступною для безпосереднього використання.

У мові С на рівні синтаксису модулі не виділяються. Реалізація модулів можлива з використанням роздільної компіляції та файлів заголовків.

Означення 9.3. Роздільна компіляція – це обробка компілятором кожного файла програми окремо.

Програма у С може складатись із кількох файлів, кожний з яких містить підпрограми та описи. Головним файлом вважається той, що містить функцію `main()`.

Файли заголовків мають розширення імені "`.h`" і містять описи глобальних констант, типів, змінних, підпрограм. Описи констант, типів, змінних не відрізняються від розглянутих у попередніх розділах. Для підпрограм у файлах заголовків наводять тільки заголовок, після якого ставлять "`;`", наприклад,

```
extern void n(t1 x1, ..., tn xn);
```

або

```
extern t f(t1 x1, ..., tn xn);
```

де ключове слово extern указує на те, що підпрограма реалізована у зовнішньому файлі.

Для реалізації модульного принципу для кожного модуля у С створюють два файли з однаковими іменами та розширеннями ".h" і ".c" (або ".cpp" для програми на С++). Файл заголовка з розширенням ".h" відіграє роль інтерфейсної частини модуля, а файл із розширенням ".c" – частини реалізації модуля. Для використання модуля слід підключити відповідний файл заголовка за допомогою директиви #include.

Наприклад, директива

```
#include "myfile.h"
```

указує компілятору, що треба підключити до програми файл заголовків myfile.h.

Зауваження. Слід звернути увагу на те, що, на відміну від підключення файлів заголовків стандартних бібліотек, імена файлів заголовків, створених програмістом, обмежуються подвійними лапками.

Директива #include дає команду включити текст відповідного файла заголовка на місце #include й далі виконати компіляцію.

Зауваження. Для виконання програми С, що використовує модулі, треба створити файл проекту. Він має розширення ".prj" і містить перелік імен файлів, що утворюють програму.

Приклади розв'язання задач

Приклад 9.1. Описати модуль і програму для роботи з раціональними числами, що мають вигляд пари (чисельник, знаменник дробу).

Розв'язок. Програму представимо у складі трьох файлів: модуль раціональних чисел – rat.h і rat.cpp – і головна програма – rattst.cpp.

Файл заголовків rat.h:

```
/* Раціональні числа rat.h*/
typedef struct {
    int nom;
    unsigned den;
} rat;
extern void add_rat(rat a, rat b, rat* pc);
extern int  eq_rat(rat a, rat b);
extern void read_rat(rat* pr);
extern void write_rat(rat r);
```

Файл реалізації rat.cpp:

```
/* Раціональні числа rat.cpp */
#include <iostream.h>
#include <stdlib.h>
```

```
#include <conio.h>
#include <math.h>
#include "rat.h"
// підпрограма визначення НСД двох цілих чисел
static int NSD(int N, int M){
  int U, V, P;
  U = abs(N);
  V = abs(M);
  if (U < V){
    P = U; U = V; V = P;
  }
  while (V > 0) {
    P = V;
    V = U % V;
    U = P;
  }
  return U;
}
// п-ма зведення числа до нескоротного дробу
static void reduce(rat* pr){
   unsigned k;
   if ((*pr).nom == 0) (*pr).den = 1;
   else {
      k = NSD((*pr).den, (*pr).nom);
      (*pr).nom /= k;
      (*pr).den /= k;
   }
}
// п-ма додавання раціональних чисел
void add_rat(rat a, rat b, rat* pc){
   (*pc).nom = a.nom * b.den + b.nom * a.den;
   (*pc).den = a.den*b.den;
   reduce(pc);
}
// булева ф-ція порівняння двох раціональних чисел
int eq_rat(rat a, rat b){
   reduce(&a); reduce(&b);
   return a.nom * b.den == b.nom * a.den;
}
// п-ма зчитування раціонального числа
void read_rat(rat* pr){
  do {
    cin >> (*pr).nom >> (*pr).den;
  } while ((*pr).den == 0);
```

```
  reduce(pr);
}
// п-ма виведення числа на екран
void write_rat(rat r){
     cout << r.nom;
   if (r.den != 1)
     cout << "/" << r.den;
}
```

Ключове слово `static` перед підпрограмою NSD указує на те, що підпрограма є внутрішньою для файла `rat.cpp` та недоступна ззовні.

Головна програма, що міститься у файлі `rattst.cpp`:

```
/* rattst.cpp */
#include <iostream.h>
#include "rat.h"
void main(){
  rat a, b, c;
  cout << "Введіть 1-е число\n";
  read_rat(&a);
  cout << "Введіть 2-е число \n";
  read_rat(&b);
  cout << "Сума =  ";
  add_rat(a, b, &c);
  write_rat(c); cout << "\n";
  if ( eq_rat(a,b) )
    cout << "Числа рівні \n";
  else
    cout << "Числа не рівні \n";
  }
```

Файл проекту `rattst.prj` міститиме назви двох файлів:

```
rat.cpp
rattst.cpp
```

Задачі для самостійної роботи

9.1. Створити модуль, в якому описати функції для обчислення значень елементарних математичних функцій (див. задачу **3.45**). Використовуючи цей модуль, знайти значення арифметичного виразу:

$$\frac{\ln 4 + e^3 - \sin 3^{3/2}}{\cos 2 + \mathrm{ch}\, 4}.$$

9.2. Описати модуль для реалізації універсального комплексного типу (задача **6.113**). Реалізувати операції, відношення та інструкції:

1) взяти комплексне число;

2) показати комплексне число;

3) з'ясувати, чи рівні два комплексних числа;
4) сума двох комплексних чисел;
5) різниця двох комплексних чисел;
6) добуток двох комплексних чисел;
7) частка від ділення двох комплексних чисел;
8) модуль комплексного числа;
9) піднесення комплексного числа до натурального степеня;
10) добуток комплексного числа та дійсного числа;
11) переведення комплексного числа до алгебраїчної форми;
12) переведення комплексного числа до тригонометричної форми.

З використанням модуля знайти корені рівняння

$$az^2 + bz + c = 0$$

з комплексними коефіцієнтами a, b, c.

9.3. Використовуючи модуль задачі **9.2**, написати програми для обчислення значень елементарних функцій комплексного аргументу (див. задачу **6.107**).

9.4. Описати модуль роботи з відрізками на числовій осі. Тип відрізка представити у вигляді структури:

```
typedef struct {
  float a, b;
  int empty;
} segment;
```

де a, b – межі відрізка, `empty` – ознака того, що відрізок порожній.

Реалізувати дії над відрізками:

1) зробити відрізок t порожнім;
2) з'ясувати, чи порожній відрізок t;
3) покласти відрізок t рівним $[a, b]$;
4) покласти відрізок t рівним перетину відрізків $t1$, $t2$.

З використанням модуля скласти програму розв'язання системи квадратних нерівностей вигляду

$$x^2 + p_i x + q_i < 0, \quad i = 1,...,n.$$

Пари коефіцієнтів нерівностей p_i, q_i зчитуються з пристрою введення.

9.5. Описати модуль роботи з точками та відрізками на площині. Типи точки та відрізка подати у вигляді записів:

```
typedef struct { double x, y; } point;
typedef struct {  point a, b; } segment;
```

Реалізувати дії над точками:

1) взяти точку t; 2) покласти точку t рівною (x, y); 3) показати точку t.

Реалізувати дії над відрізками:

1) взяти відрізок s; 2) показати відрізок s;
3) покласти відрізок s рівним $[a, b]$; 4) довжина відрізка s;
5) з'ясувати, чи лежить точка t на одній прямій із відрізком s;

6) з'ясувати, чи лежить точка t усередині відрізка s;

7) площа трикутника, утвореного точкою t і відрізком s.

9.6. У файлі записано послідовність точок. З використанням модуля роботи з точками та відрізками на площині (див. задачу **9.5**) знайти:

а) трикутник із найбільшою площею, утворений точками послідовності;

б) коло найменшого радіуса, усередині якого лежать усі точки послідовності;

в) відрізок, на якому лежить найбільша кількість точок послідовності;

г) коло, на якому лежить найбільша кількість точок послідовності.

9.7. Реалізувати модуль для роботи з поліномами $P_n(x)$ степеня n. Поліном реалізувати як масив коефіцієнтів. У модулі реалізувати операції:

1) зчитати поліном $P_n(x)$;

2) показати поліном$P_n(x)$;

3) визначити значення полінома $P_n(x)$ для заданого x;

4) знайти похідну $P_n'(x)$ від полінома $P_n(x)$;

5) знайти первісну $F_n(x)$ полінома $P_n(x)$, що задовольняє умову $F_n(0) = c$, де c – задане дійсне число;

6) знайти суму двох поліномів;

7) знайти добуток двох поліномів;

8) реалізувати операцію ділення з остачею двох поліномів.

Використовуючи описаний модуль, розв'язати задачі:

а) відстань, яку пройшла матеріальна точка за час t, визначається за законом $s(t) = P_n(t)$. Визначити її швидкість у момент часу t;

б) швидкість руху точки в момент часу t визначається за законом $v(t) = P_n(t)$; визначити, яку відстань пройде точка за відрізок часу $[t_1, t_2]$;

в) знайти похідну порядку k від полінома $P_n(x)$;

г) визначити найбільший спільний дільник двох поліномів $P_n(x)$ і $R_n(x)$;

ґ) визначити найменше спільне кратне двох поліномів;

д) розв'язати задачу Коші для звичайного диференціального рівняння

$$y^{(k)} = P_n(x),\ x \ge x_0,\ \ y(x_0) = y_0, \ldots, y^{(k-1)}(x_0) = y_0^{k-1}.$$

9.8. Описати модуль, у якому реалізувати алгоритми сортування для масивів, які наведено в попередньому розділі. Використовуючи модуль, провести порівняльну характеристику методів сортування для невідсортованих і частково відсортованих масивів (що складаються з двох або трьох упорядкованих підмасивів)

9.9. Оформити модуль для роботи з векторами та матрицями. У модулі описати :

▪▪ операції введення/виведення:

1) зчитування вектора з клавіатури;

2) зчитування матриці з клавіатури;

3) зчитування вектора з текстового файла;

4) зчитування матриці з текстового файла;
5) виведення вектора на екран;
6) виведення матриці на екран;
7) запис вектора у текстовий файл;
8) запис матриці у текстовий файл;

▪▪ операції роботи з векторами та матрицями:

1) перестановка *i*-го і *j*-го рядків матриці;
2) додавання до *i*-го рядка матриці *j*-й рядок, помножений на число *a*;
3) множення *i*-го рядка матриці на число *a*;
4) множення матриці на вектор;
5) множення матриці на матрицю.

Використовуючи цей модуль, методом Гауса знайти розв'язок системи лінійних алгебраїчних рівнянь. Урахувати випадки, коли система:

а) має єдиний розв'язок; б) має безліч розв'язків;
в) не має жодного розв'язку.

Якщо система має єдиний розв'язок, то перевірити результат множенням вихідної матриці на знайдений вектор.

9.10. Використовуючи модуль задачі **9.9**, знайти обернену матрицю до заданої.

9.11. Використовуючи модуль задачі **9.9**, знайти експоненту матриці

$$e^A = E + A + \frac{A^2}{2!} + \frac{A^3}{3!} + ...,$$

де A – задана матриця, E – одинична матриця, урахувавши при обчисленні n доданків суми у правій частині.

9.12. Оформити модуль, що складається з функцій, описаних у підпунктах а) – г) і а) – ґ) задач **6.97** і **6.98**, відповідно. Використовуючи цей модуль, скласти програми для додаванні та множення "у стовпчик" двох чисел, записаних у вигляді рядків, що є позиційними записами цих чисел у десятковій системі числення.

Розділ 10
РЕКУРСИВНІ СТРУКТУРИ ДАНИХ

Для розв'язання задач, у яких розмір даних суттєво залежить від умов, що визначаються тільки під час виконання програми, використовують **динамічні структури даних**.

Зауваження. Реалізація динамічних структур даних залежить від мови програмування.

Розглянемо реалізацію динамічних структур даних з використанням вказівників.

Означення 10.1. Вказівником називають змінну, значенням якої є адреса пам'яті.

Найчастіше у С вказівники використовуються для:

а) реалізації модифікованих параметрів у підпрограмах;

б) роботи з динамічною пам'яттю (**купою**).

Оголошення вказівника

```
t *p;
```

де p – вказівник, t – тип вказівника.

Наведемо основні операції і дії з вказівниками.

а) Розіменування вказівника. Отримання значення, на яке посилається вказівник, називається **розіменуванням вказівника**. Операція

```
*p
```

за вказівником p повертає значення (змінну), на яке посилається вказівник.

б) Отримання адреси змінної. Операція

```
&x
```

повертає адресу змінної x.

в) Присвоєння. Якщо p1 і p2 – вказівники на змінні одного типу (або блоки пам'яті однакового розміру), то після виконання інструкції

```
p1 = p2;
```

вказівник p1 указуватиме на той самий блок пам'яті, що й p2. Присвоєння для змінних, на які посилаються вказівники, записують з використанням розіменування:

```
*p1 = *p2;
```

г) Збільшення та зменшення вказівника на одиницю. Операції

```
p++;
p--;
```

відповідно збільшують або зменшують адресу вказівника p на один елемент типу t.

ґ) Арифметичні операції. Вирази

```
p+i
p-i
```

повертають адресу, яка відповідно більша або менша від p на i елементів типу t.

д) Відношення. Для вказівників визначено стандартні відношення

```
=, !=, >, <, >=, <=.
```

е) Виділення (захоплення) пам'яті. Функція `malloc(n)` виділяє у динамічній пам'яті блок розміром n байтів і повертає адресу першого байта з цього блоку. Наприклад, інструкція

```
p = malloc(sizeof(int));
```

виділяє у динамічній пам'яті блок пам'яті розміром 2 байти та встановлює вказівник p на початок цього блока пам'яті.

є) Звільнення пам'яті. Інструкція

```
free(p);
```

звільняє блок пам'яті, на який указував вказівник p.

Зауваження. Для використання функцій `malloc` і `free` потрібно підключити бібліотеку `stdlib.h`.

Означення 10.2. Нульовий (порожній) **вказівник** – спеціальне значення, яке використовується для того, щоб показати, що змінна-вказівник не вказує на жодну клітину пам'яті.

У С порожній вказівник представлено константою NULL.

Розглянемо детальніше динамічні структури даних.

Означення 10.3. Стек – динамічна структура даних із принципом доступу до елементів "останнім прийшов – першим пішов" (англ. LIFO – last in, first out). Усі операції у стеку проводяться тільки з одним елементом, що називається **верхівкою стека**.

Базові дії при роботі зі стеком:

1. Почати роботу.
2. Операція визначення, чи є стек порожнім.
3. Вштовхнути (додати) елемент до стека.
4. Взяти верхівку стека.

Зауваження. Тут і далі інструкція "Почати роботу" означає операцію створення відповідної порожньої динамічної структури. Отже, у цьому випадку інструкція "Почати роботу" означає створення порожнього стека елементів.

Стек можна визначити рекурсивно:

1. Порожній стек.
2. Верхівка стека; стек.

Означення 10.4. Черга – динамічна структура даних із принципом доступу до елементів "першим прийшов – першим пішов" (англ. FIFO – first in, first out). У черги є **початок** (**голова**) та **кінець** (**хвіст**). Еле-

мент, що додається до черги, опиняється в її хвості. Елемент, що береться (тобто видаляється) з черги, розташований у її голові.

Базові операції для роботи з чергою:

1. Почати роботу.
2. Операція визначення, чи є черга порожньою.
3. Додати елемент до кінця черги.
4. Взяти елемент з початку черги.

Чергу можна визначити рекурсивно:

1. Порожня черга.
2. Перший елемент; черга.

Означення 10.5. Дек (від англ. double ended queue – двобічна черга) – динамічна структура (однотипних) даних, елементи якої можуть додаватись як на початок, так і в кінець.

Базові операції для роботи з деком:

1. Почати роботу.
2. Операція визначення, чи є дек порожнім.
3. Додати елемент до початку дека.
4. Взяти елемент з початку дека.
5. Додати елемент до кінця дека.
6. Взяти елемент з кінця дека.

Дек можна визначити рекурсивно:

1. Порожній дек.
2. Перший елемент; дек.
3. Дек; останній елемент.

Означення 10.6. Класичний (лінійний однобічний) список – це динамічна структура даних, що складається з елементів одного типу, пов'язаних між собою у строго визначеному порядку. При цьому визначено перший та останній елементи списку, і кожен елемент списку вказує на наступний елемент списку.

Базові операції для роботи з класичним списком:

1. Почати роботу.
2. Операція визначення, чи порожній список.
3. Додати елемент у початок списку.
4. Голова списку. Операція для отримання першого елемента списку (без зміни всього списку).
5. Хвіст списку. Операція для доступу до списку, що складається з усіх елементів вихідного списку, крім першого.

Класичний список можна визначити рекурсивно:

1. Порожній список.
2. Перший елемент; список.

Означення 10.7. Список із поточним елементом – різновид класичного списку – динамічна структура даних, що складається з елементів одного типу, пов'язаних між собою, і структури керування, що вказує на поточний елемент структури.

Базові операції зі списками з поточним елементом:
1. Почати роботу.
2. Операція визначення, чи порожній залишок списку.
3. Зробити поточним перший елемент списку.
4. Перейти до наступного елемента.
5. Отримати поточний елемент (список при цьому не змінюється).
6. Вставити новий елемент у список перед поточним.
7. Видалити поточний елемент у списку.

Список з поточним елементом можна визначити так:
1. Порожній список.
2. Список; поточний елемент; список.
3. Список.

Означення 10.8. Кільцевий список – різновид списку з поточним елементом, для якого не визначено перший та останній елементи. Усі елементи зв'язані у кільце, відомий лише порядок слідування.

Базовий набір дій над кільцевими списками:
1. Почати роботу.
2. Довжина списку (кількість елементів у списку).
3. Перейти до наступного елемента.
4. Отримати поточний елемент (список при цьому не змінюється).
5. Вставити новий елемент у список перед поточним.
6. Видалити поточний елемент у списку.

Кільцевий список можна визначити рекурсивно:
1. Порожній список.
2. Список; поточний елемент; список.

Означення 10.9. Двобічно зв'язаний (двозв'язний) список – динамічна структура даних, що складається з елементів одного типу, зв'язаних між собою у строго визначеному порядку. При цьому визначено перший та останній елементи у списку, а кожен елемент списку вказує на наступний і попередній елементи у списку.

Базовий набір дій над двозв'язними списками:
1. Почати роботу.
2. Операція визначення, чи порожній список.
3. Операція визначення, чи порожній початок списку.
4. Операція визначення, чи порожній кінець списку.
5. Зробити поточними перший елемент списку.
6. Зробити поточними останній елемент списку.
7. Перейти до наступного елемента.
8. Перейти до попереднього елемента.
9. Отримати поточний елемент.
10. Вставити новий елемент перед поточним.
11. Вставити новий елемент після поточного.
12. Видалити поточний елемент.

Приклади розв'язання задач

Приклад 10.1. Описати модуль для роботи зі стеком символів, у якому реалізувати стандартні операції роботи над стеком. Використовуючи цей модуль, написати програму, що виводить задану послідовність символів у зворотному порядку.

Розв'язок. Опишемо модуль для роботи зі стеком у складі двох файлів – файл заголовків `stack.h` і файл реалізації `stack.cpp`.

```
/* Стек символів stack.h */
typedef struct selem *stack;  // Стек
struct selem {                // Елемент стека
            char d;
            stack next;
            };
/* Почати роботу */
extern void init(stack *ps);
/* Функція визначення, чи є стек порожнім */
extern int empty(stack s);
/* Вштовхнути елемент у стек */
extern void push(stack *ps, char ch);
/* Взяти верхівку стека */
extern char top(stack *ps);
```

Файл реалізації:

```
/* Стек символів stack.cpp */
#include <stdlib.h>
#include "stack.h"
void init(stack *ps){
  *ps = NULL;
}
int empty(stack s){
  return s == NULL;
}
 void push(stack *ps, char ch){
  stack p;
  // Створюємо у купі елемент стека
  p = (stack) malloc(sizeof(struct selem));
  // робимо p першим елементом стека
  p -> d = ch;
  p -> next = *ps;
  *ps = p;
}
char top(stack *ps){
  if (empty(*ps)) return 0;
  else {
```

```
    stack p;
    char ch;
    p = *ps;
    ch = p -> d;
    *ps = p -> next;
    free(p);
    return ch;
  }
}
```

Зробимо деякі пояснення до файла реалізації. Інструкція

```
p = (stack) malloc(sizeof(struct selem));
```

виділяє пам'ять під елемент стека розміром `sizeof(struct selem)` та явно приводить вказівник до типу `stack`.

Позначення `"->"` використовується у С як альтернативне для вибору поля структури, на яку посилається вказівник. Таким чином, у програмі замість `(*p).d` використовується еквівалентний виклик `p -> d`.

Щоб виконати завдання, досить записати послідовність символів до стека, а потім – вивести цю послідовність зі стека на екран. Таким чином, програма, що використовує вищеописаний модуль, матиме вигляд:

```
#include <conio.h>
#include <iostream.h>
#include "stack.h"
void main(){
  stack st; // Оголошуємо стек символів
  char ch;  // Змінна для поточного символа
  init(&st);// Ініціалізуємо стек
  // Вводимо символи з клавіатури у стек
  do {
    ch = getch(); // Зчитуємо символ з клавіатури
    // Якщо введений символ відповідає клавішам
    // Esc або Enter, то припиняємо введення
    if (ch == 27 || ch == 13) break;
    cout << ch;
    push(&st, ch); // Додаємо символ у стек
  } while (1);
  cout << "\n";
  // Послідовно беремо символи зі стека, поки він
  // непорожній та виводимо їх на екран
  while (!empty(st)) cout << top(&st);
}
```

Приклад 10.2. Описати модуль для роботи з чергою цілих чисел, у якому реалізувати базові операції роботи з чергою. Використовуючи цей модуль, написати програму, що формує динамічну структуру, яка є по-

слідовністю (чергою) випадкових чисел (від 0 до 100). Вважати, що послідовність закінчується 0, яке генерується на випадковому кроці.

Розв'язок. Опишемо модуль для роботи з чергою у складі двох файлів – файл заголовків queue.h і файл реалізації queue.cpp.

Чергою буде двоелементна структура, перший елемент якої (bg) буде вказівником на голову черги, а другий (en) – на хвіст черги. Елементи черги, подібно до стека, опишемо рекурсивно.

```
/* queue.h */
// Вказівник на елемент черги
typedef struct qelem *qref;
// Елемент черги
struct qelem {
   int d;
   qref next;
};
// Черга
typedef struct {
   qref bg, en;
} queue;
// Почати роботу
extern void init(queue *pq);
// Визначення, чи є черга порожньою
extern int  empty(queue q);
// Додати елемент до кінця черги
extern void add(queue *pq, int n);
// Взяти елемент з початку черги
extern int  take(queue *pq);
```

Файл реалізації модуля:

```
/* queue.cpp */
#include <stdlib.h>
#include <iostream.h>
#include "queue.h"
void init(queue *pq){
  pq -> bg = NULL;
  pq -> en = NULL;
}
int empty(queue q){
  return (q.bg == NULL);
}
void add(queue *pq, int n){
  qref p;
  p = (qref) malloc(sizeof(struct qelem));
  p -> d = n;
```

```
  p -> next = NULL;
  if (empty(*pq))
    pq -> bg = p;
  else
    pq -> en -> next = p;
  pq -> en = p;
}
int take(queue *pq){
  if (empty(*pq)) {
    cout << "Queue is empty\n";
    return 0;
  }
  else {
    qref p;
    int n;
    p = pq -> bg;
    n = pq -> bg -> d;
    pq -> bg = pq -> bg -> next;
    if (pq -> bg == NULL) pq -> en = NULL;
    free(p);
    return n;
  }
}
```

Програма, що виконує завдання:

```
#include <stdlib.h>
#include <iostream.h>
#include "queue.h"
void main(){
  queue q;     // Черга
  int n;       // Змінна для поточного числа
  randomize();
  init(&q);    // Ініціалізація черги
  do {
    // Генерація випадкових чисел у змінну n
    n = random(100);
    // Якщо випадкове число дорівнює нулю, то
    // припиняємо додавання елементів до черги
    if (n == 0) break;
    add(&q, n);
  } while (1);
  // Поки черга залишається непорожньою,
  // виводимо елементи черги на екран
```

```
  while (!empty(q))
    cout << take(&q) << " ";
  cout << "\n";
}
```

Приклад 10.3. Описати модуль для роботи з деком цілих чисел, у якому реалізувати базові операції роботи з деком. Використовуючи цей модуль, реалізувати задачу: з клавіатури ввести послідовність цілих чисел, що завершується 0. На базі послідовності побудувати динамічну структуру, в якій спочатку йтимуть від'ємні числа, а потім – додатні.

Розв'язок. Модуль для роботи з деком складається з двох файлів – файл заголовків `deque.h` і файл реалізації `deque.cpp`.

```
/* deque.h */
// Вказівник на елемент дека
typedef struct delem *dref;
// Елемент дека
struct delem {
  int d;
  dref next, prev;
};
// Дек
typedef struct{
  dref bg, en;
} deque;
/* Почати роботу */
extern void init(deque *pd);
// Функція визначення, чи є дек порожнім
extern int empty(deque d);
// Додати елемент до початку дека
extern void put_bg(deque *pd, int n);
// Взяти елемент з початку дека
extern int get_bg(deque *pd, int *pn);
// Додати елемент до кінця дека
extern void put_en(deque *pd, int n);
// Взяти елемент з кінця дека
extern int get_en(deque *pd, int *pn);
```

Файл реалізації:

```
/* deque.cpp*/
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include "deque.h"
void init(deque *pd){
  pd -> bg = NULL;
```

```
  pd -> en = NULL;
}
int empty(deque d){
  return (d.bg == NULL);
}
void put_bg(deque *pd, int n){
  dref p;
  p = (dref) malloc(sizeof(struct delem));
  p -> d = n;
  p -> prev = NULL;
  p -> next = pd -> bg;
  if (pd -> bg == NULL) pd -> en = p;
  else pd -> bg -> prev = p;
  pd -> bg = p;
}
int get_bg(deque *pd, int *pn) {
  dref p;
  if (empty(*pd)){
    cout << "get_bg: Дек порожній\n";
    // якщо дек порожній, get_bg повертає 0
    return 0;
  }
  p = pd -> bg;
  *pn = p -> d;
  pd -> bg = pd -> bg -> next;
  if (pd -> bg == NULL) pd -> en = NULL;
  else pd -> bg -> prev = NULL;
  free(p);
  return 1;
}
void put_en(deque *pd, int n){
  dref p;
  p = (dref) malloc(sizeof(struct delem));
  p -> d = n;
  p -> next = NULL;
  p -> prev = pd -> en;
  if (empty(*pd)) pd -> bg  = p;
  else pd -> en -> next = p;
  pd -> en = p;
}
int get_en(deque *pd, int *pn) {
  dref p;
  if (empty(*pd)){
    cout << "get_en: Дек порожній\n";
```

```
    // якщо дек порожній, get_en повертає 0
    return 0;
  }
  p = pd -> en;
  *pn = p -> d;
  pd -> en = pd -> en -> prev;
  if (pd -> en == NULL) pd -> bg = NULL;
  else pd -> en -> next = NULL;
  free(p);
  return 1;
}
```

Для реалізації завдання потрібно аналізувати введене з клавіатури число: якщо воно від'ємне, то додавати його у початок дека, а якщо додатне – у кінець дека. Отже, програма матиме вигляд:

```
#include <iostream.h>
#include "deque.h"
void main(){
  deque q;
  int n;
  init(&q);
  do {
    cin >> n;
    if (n == 0)
      break;
    else if (n < 0)
      put_bg(&q, n); // додаємо у початок дека
    else
      put_en(&q, n); // додаємо у кінець дека
  } while (1);
  while (!empty(q)){
    get_bg(&q, &n);
    cout << n << " ";
  }
}
```

Приклад 10.4. Описати модуль, у якому реалізувати базові операції для роботи з класичним списком.

Розв'язок. Опишемо модуль у складі двох файлів – файл заголовків `classlst.h` і файл реалізації `classlst.cpp`.

```
/* classlst.h */
typedef struct lelem *list;   /* Список */
struct lelem {               /* Елемент списку */
       int d;
       list next;
       };
```

```
/* Почати роботу */
extern void init(list *pl);
/* Операція визначення, чи порожній список */
extern int  empty(list l);
/* Додати елемент у початок списку */
extern void add(list *pl, int n);
/* Голова списку */
extern int  head(list l);
/* Хвіст списку */
extern list tail(list l);
```

Файл реалізації

```
/* classlst.cpp */
#include <stdlib.h>
#include "classlst.h"
void init(list *pl){
  *pl = NULL;
}
int empty(list l){
  return l == NULL;
}
void add(list *pl, int n){
  list p;
  p = (list) malloc(sizeof(struct lelem));
  p -> d = n;
  p -> next = *pl;
  *pl = p;
}
int  head(list l){
  if (!empty(l))
    return l -> d;
  else
    return 0;
}
list tail(list l){
  if (empty(l))
    return NULL;
  else
    return l -> next;
}
```

Як тестову програму реалізуємо таку задачу. У список записати n випадкових чисел, вивести їх на екран, після чого знищити список.

```
/* uselst.cpp */
#include <iostream.h>
#include <stdlib.h>
```

```
#include "classlst.h"
// Заповнення списку n випадковими числами
void RandomFillList(list *pl, int n){
  randomize();
  for (int i = 1; i <= n; i++)
    add(pl, random(10));
}
// Рекурсивна підпрограма
// виведення списку на екран
void ShowList(list l){
  if (!empty(l)){
    cout << head(l) << " ";
    ShowList(tail(l));
  }
}
void main(){
  list l; int n;
  init(&l);       // Ініціалізація списку
  cout << "n=? "; cin >> n;
  RandomFillList(&l, n);
  ShowList(l);
}
```

Зауваження. Класичний список може одночасно використовуватися кількома динамічними структурами. Саме тому список має існувати до завершення виконання програми. Через це у програмі не описано підпрограму для видалення списку з пам'яті. Якщо ж у цьому стане потреба, то підпрограму видалення списку з пам'яті можна описати так:

```
// Підпрограма видалення списку з пам'яті
void Remove(list *pl){
  list p;
  while (!empty(*pl)){
    p = *pl; *pl = p -> next; free(p);
  }
```

Приклад 10.5. Описати модуль, у якому реалізувати базові операції для роботи зі списком з поточним елементом.

Розв'язок. Опишемо модуль у складі двох файлів – файл заголовків `listcur.h` і файл реалізації `listcur.cpp`. У задачі розглядатимемо список цілих чисел.

```
/* listcur.h */
// Вказівник на елемент списку
typedef struct lelem *lref;
// Елемент списку
struct lelem {
  int d;
```

```
  lref next;
};
// Список - складається з двох полів:
// beg - вказівник на перший елемент списку
// cur – вказівник на поточний елемент списку
typedef struct {
  lref beg, cur;
} list;
// Почати роботу
extern void init(list *pl);
// Визначення, чи порожній залишок списку
extern int emp_end(list l);
// Зробити поточним перший елемент списку
extern void first(list *pl);
// Перейти до наступного елемента
extern void next(list *pl);
// Отримати поточний елемент
extern int current(list l);
// Вставити новий елемент у список перед поточним
extern void insert(list *pl, int n);
// Видалити поточний елемент у списку
extern void del_cur(list *pl);
```

Файл реалізації

```
/* listcur.cpp */
#include <iostream.h>
#include <stdlib.h>
#include "listcur.h"
void init(list *pl){
  pl -> beg = NULL;
  pl -> cur = NULL;
}
int emp_end(list l){
  return l.cur == NULL;
}
void first(list *pl){
  pl -> cur = pl -> beg;
}
void next(list *pl){
  if (!emp_end(*pl)){
    pl -> cur = pl -> cur -> next;
  }
}
int current(list l){
  if (!emp_end(l))
```

```
    return l.cur -> d;
  else
    return 0;
}
// Допоміжна функція, що повертає елемент
// списку, що стоїть перед поточним
lref find_prev(list l){
  lref p;
  if (l.beg == l.cur) p = NULL;
  else {
    p = l.beg;
    while (p -> next != l.cur)
      p = p -> next;
  }
  return p;
}
void insert(list *pl, int n){
  lref p;
  p = (lref) malloc(sizeof(struct lelem));
  p -> d = n;
  p -> next = pl -> cur;
  if(pl -> beg == pl -> cur)
    pl -> beg = p;
  else
    find_prev(*pl) -> next = p;
}
void del_cur(list *pl){
  lref p;
  if(pl -> cur == NULL){
    cout << "del_cur: список порожній\n";
  }
  else {
    p = pl -> cur;
    if (pl -> beg == p)
      pl -> beg = pl -> cur -> next;
    else
      find_prev(*pl) -> next = pl -> cur -> next;
    pl -> cur = pl -> cur -> next;
    free(p);
  }
}
```

Приклад 10.6. Описати модуль, у якому реалізувати базові операції для роботи з кільцевим списком.

Розв'язок. Опишемо модуль у складі двох файлів – файл заголовків `rlist.h` і файл реалізації `rlist.cpp`.

```
/* rlist.h */
typedef struct relem *rlist;  /* Список */
struct relem {                /* Елемент списку */
  int d;
  rlist next;
};
// Почати роботу
extern void init(rlist *pr);
// Довжина списку
extern int len(rlist r);
// Перейти до наступного елемента
extern void next(rlist *pr);
// Поточний елемент
extern int current(rlist r);
// Вставити новий елемент у список перед поточним
extern void insert(rlist *pr, int n);
// Видалити поточний елемент у списку
extern void del_cur(rlist *pr);
```

Файл реалізації:

```
/* rlist.cpp */
#include <iostream.h>
#include <stdlib.h>
#include "rlist.h"
void init(rlist *pr){
  *pr = NULL;
}
int len(rlist r){
  int i = 0;
  rlist p;
  p = r;
  if (p != NULL)
    do {
      p = p -> next;
      i++;
    } while (p != r);
  return i;
}
void next(rlist *pr){
  if (*pr != NULL)
    *pr = (*pr) -> next;
  else
    cout << "next: список порожній\n";
}
```

```
int current(rlist r){
  if (r != NULL)
    return r -> d;
  else {
    cout << "current: список порожній\n";
    return 0;
  }
}
// Допоміжна підпрограма, що повертає попередній
// елемент у списку
rlist find_prev(rlist r){
  rlist p;
  if (r == NULL) p = NULL;
  else {
    p = r;
    while (p -> next != r)
      p = p -> next;
  }
  return p;
}
void insert(rlist *pr, int n){
  rlist p;
  p = (rlist) malloc(sizeof(struct relem));
  p -> d = n;
  p -> next = *pr;
  if(len(*pr) == 0)
    p -> next = p;
  else
    find_prev(*pr) -> next = p;
  *pr = p;
}
void del_cur(rlist *pr){
  rlist p;
  if(*pr  == NULL)
    cout << "del_cur: список порожній\n";
  else {
    p = *pr;
    if (len(*pr) == 1)
      *pr = NULL;
    else
      find_prev(*pr) -> next = (*pr) -> next;
    *pr = (*pr)  -> next;
    free(p);
  }
}
```

Приклад 10.7. Описати модуль, у якому реалізувати базові операції для роботи з двозв'язним списком.

Розв'язок. Модуль роботи із двозв'язним списком складається з двох файлів: `dlist.h` і `dlist.c`.

```
/* dlist.h */
/* Вказівник на елемент списку */
typedef struct dlelem *dlref;
/* Елемент списку */
struct dlelem {
            int d;
            dlref next, prev;
          };
/* Список */
typedef struct {
                  dlref bg, cur, en;
                } dlist;
/* Почати роботу */
extern void init(dlist *pdl);
// визначення, чи порожній список
extern int empty(dlist dl);
// визначення, чи порожній початок списку
extern int emp_beg(dlist dl);
// визначення, чи порожній кінець списку
extern int emp_end(dlist dl);
// Зробити поточними перший елемент списку
extern void first(dlist *pdl);
// Зробити поточними останній елемент списку
extern void last(dlist *pdl);
// Перейти до наступного елемента
extern void next(dlist *pdl);
// Перейти до попереднього елемента
extern void previous(dlist *pdl);
// Отримати поточний елемент
extern int current(dlist dl);
// Вставити новий елемент перед поточним
extern void ins_before(dlist *pdl, int n);
// Вставити новий елемент після поточного
extern void ins_after(dlist *pdl, int n);
// Видалити поточний елемент
extern void del_cur(dlist *pdl);
```

Зауважимо, що функції подібні до `init`, `emp_beg`, `emp_end`, `first`, `last`, `next`, `previous`, `current`, `ins_after`, неодноразово наводилися у попередніх задачах і без особливих труднощів можуть

бути описані для даної задачі. Тому у файлі реалізації наведемо тільки реалізацію для функцій ins_before і del_cur.

```
/* dlist.c*/
#include <stdio.h>
#include <stdlib.h>
#include "dlist.h"
/* У цьому файлі також має бути реалізація для
   підпрограм init, emp_beg, emp_end, first,
   last, next, previous, current, ins_after */
void ins_before(dlist *pdl, int n){
   dlref p;
   p = (dlref) malloc(sizeof(struct dlelem));
   p -> d = n;
   p -> next = pdl -> cur;
   if (pdl -> cur == NULL){
     /*порожній список*/
     p -> prev = NULL;
     pdl -> bg = pdl -> en = pdl -> cur = p;
   }
   else {
     if (pdl -> cur == pdl -> bg) {
        /*вставка перед першим елементом*/
        p -> prev = NULL;
        pdl -> bg = p;
     }
     else {
        /*вставка всередині списку*/
        p -> prev = pdl -> cur -> prev;
        pdl -> cur -> prev -> next = p;
     }
     pdl -> cur -> prev = p;
   }
}
void del_cur(dlist *pdl){
  if(pdl -> cur == NULL)
      cout << "delete: список порожній\n");
  else {
    dlref p;
    p = pdl -> cur;
    if (pdl -> cur == pdl -> bg &&
              pdl -> cur == pdl -> en){
      /*список з одного елемента*/
      pdl -> bg = pdl -> cur = pdl -> en = NULL;
```

```
    }
    else if (pdl -> cur != pdl -> en){
      /*поточний елемент неостанній*/
      pdl -> cur = pdl -> cur -> next;
      pdl -> cur -> prev = p -> prev;
      if (p == pdl -> bg)
        /*поточний елемент перший*/
        pdl -> bg = pdl -> cur;
      else
        /*поточний елемент не перший*/
        p -> prev -> next = pdl -> cur;
    }
    else {
      /*поточний елемент останній*/
      pdl -> cur = pdl -> cur -> prev;
      pdl -> cur -> next = NULL;
      pdl -> en = pdl -> cur;
    }
    free(p);
  }
}
```

Задачі для самостійної роботи

Зауваження. Усі рекурсивні структури даних у цьому розділі реалізовувати на базі вказівників та динамічної пам'яті.

10.1. Описати модуль для реалізації стека символів. Передбачити виконання дій над стеком:

1) почати роботу; 2) з'ясувати, чи порожній стек;
3) уштовхнути елемент у стек; 4) верхівка стека;
5) забрати верхівку стека.

Використовуючи цей модуль, розв'язати задачу: на вхід подається послідовність символів. Упорядкувати цю послідовність за зростанням.

Указівка. Для впорядкування використати два стеки.

10.2. Використовуючи модуль для реалізації стека символів (задача **10.1**), скласти підпрограми:

а) довжина стека; б) змінити верхівку стека;
в) інверсія стека; г) забрати n елементів стека.

10.3. Описати модуль для реалізації черги цілих чисел. Передбачити виконання дій над чергою:

1) почати роботу; 2) з'ясувати, чи порожня черга;
3) додати елемент до кінця черги; 4) взяти елемент із початку черги.

Використовуючи цей модуль, скласти підпрограму обчислення довжини черги. Запобігти знищенню черги після виклику підпрограми. Передбачити рекурсивний і нерекурсивний варіанти.

10.4. Використовуючи модуль для реалізації черги цілих чисел (задача **10.3**), розв'язати таку задачу. У черзі є m чисел. Проводять n випробувань, унаслідок кожного з яких отримують випадкові числа 0 або 1. Якщо отримано 0, то треба взяти елемент із початку черги й показати його на екрані. Якщо отримано 1, то ввести число з клавіатури та додати до кінця черги. Після завершення випробувань показати залишок черги.

Указівка. Використати генератор випадкових чисел (підпрограми `randomize()` – ініціалізації генератора та `random(k)` – отримання випадкового натурального числа в діапазоні від `0` до `k – 1`. Для використання цих підпрограм потрібно підключити бібліотеку `stdlib.h`).

10.5. Використовуючи модуль для реалізації черги цілих чисел (задача **10.3**), розв'язати задачу "лічилка": по колу розташовано n гравців, що мають номери від 1 до n. У лічилці m слів. Починають лічити з першого гравця. m-ий гравець вибуває, після чого знову починають лічити з гравця, що є наступним за тим, що вибув, тощо. Показати послідовність номерів, що вибувають, за заданих значень n і m.

10.6. Використовуючи модуль для реалізації черги цілих чисел (задача **10.3**), розв'язати задачу: у магазині стоїть черга з m покупців. Час обслуговування покупця з черги – випадкове ціле число в діапазоні від 1 до t_1. Час додавання нового покупця до черги – випадкове ціле число в діапазоні від 1 до t_2. Промоделювати стан черги (показати час виникнення подій – обслуговування та додавання покупця) за період часу T ($T >> t_1$, $T >> t_2$). Показати залишок черги.

10.7. Використовуючи модуль для реалізації черги цілих чисел (задача **10.3**), скласти підпрограми:

а) інверсія черги; б) забрати n елементів черги;

в) конкатенація двох черг; г) порівняти дві черги.

10.8. Описати модуль для реалізації дека цілих чисел. Передбачити виконання дій над деком:

1) почати роботу; 2) з'ясувати, чи порожній дек;

3) додати елемент до початку дека; 4) взяти елемент із початку дека;

5) додати елемент до кінця дека; 6) взяти елемент із кінця дека.

Використовуючи цей модуль, скласти підпрограми: обчислення довжини дека, присвоєння для деків. Запобігти знищенню дека після виклику підпрограми обчислення його довжини.

10.9. Використовуючи модуль для реалізації дека цілих чисел (задача **10.8**), реалізувати стек цілих чисел на базі дека. Реалізувати стек на базі дека – означає описати модуль і реалізувати дії над стеком, викликаючи відповідні підпрограми, що реалізують дії над деком. Для реалі-

зованого стека розв'язати задачу інвертування вхідної послідовності цілих чисел.

10.10. Використовуючи модуль для реалізації дека цілих чисел (задача **10.8**), реалізувати чергу на базі дека (див. задачу **10.9**). Для реалізованої черги розв'язати задачу обчислення довжини черги.

10.11. Розв'язати задачу "лічилка" (задача **10.5**), використовуючи модуль для реалізації дека цілих чисел (задача **10.8**).

10.12. Використовуючи модуль для реалізації дека цілих чисел (задача **10.8**), розв'язати задачу **10.4**, передбачивши чотири можливих результати кожного випробування (випадкові числа від 0 до 3):

1) 0 – взяти елемент із початку дека та показати його на екрані;

2) 1 – увести число з клавіатури та додати його до початку дека;

3) 2 – взяти елемент із кінця дека та показати його на екрані;

4) 3 – увести число з клавіатури та додати його до кінця дека.

10.13. Використовуючи модуль для реалізації дека цілих чисел (задача **10.8**), розв'язати задачу **10.6**, передбачивши, що через випадковий час від 1 до t_3 до початку черги додається "пільговий" покупець, який обслуговується першим, а через випадковий час від 1 до t_4 не витримує та йде з черги останній покупець.

10.14. Використовуючи модуль для реалізації дека цілих чисел (задача **10.8**), скласти підпрограми:

а) інверсія дека; б) конкатенація двох деків;

в) порівняти два деки; г) забрати n елементів з початку дека;

ґ) забрати n елементів з кінця дека;

д) замінити початок дека; е) замінити кінець дека.

10.15. Описати модуль для реалізації класичного списку цілих чисел. Передбачити виконання дій над списком:

1) почати роботу; 2) з'ясувати, чи порожній список;

3) додати елемент; 4) голова списку;

5) хвіст списку.

Використовуючи цей модуль, скласти підпрограми для:

а) обчислення довжини списку та пошуку в списку елемента, рівного заданому числу;

б) конкатенації двох списків та інверсії списку;

в) сортування списку;

г) перевірки, чи є один список початком іншого та чи входить один список до іншого;

ґ) заміни всіх входжень до списку елемента m числом n.

10.16. Використовуючи модуль для реалізації класичного списку цілих чисел (задача **10.15**), скласти підпрограми для:

а) перевірки списку на симетричність;

б) виділення зі списку L n елементів, починаючи з елемента з номером m, у новий список;

в) видалення n елементів списку L, починаючи з елемента з номером m;

г) побудови нового списку L_2 з елементів списку L_1, що задовольняють умову $F(x)$, де x – поточний елемент списку L_1;

ґ) видалення зі списку L елементів, що задовольняють умову $F(x)$, де x – поточний елемент списку L;

д) додавання до списку L_1 елементів списку L_2, що задовольняють умову $F(x)$, де x – поточний елемент списку L_2;

е) видалення рівних елементів списку (вважати, що елементи списку L упорядковані за неспаданням, тобто виконується умова $F(x_1) \le F(x_2)$, де x_1, x_2 – два послідовні елементи списку L);

є) створення впорядкованого списку L із елементів списків L_1 і L_2 (тут вважати, що елементи кожного списку впорядковані за неспаданням, тобто $F(x_1) \le F(x_2)$, де x_1, x_2 – два послідовні елементи списку).

10.17. Використовуючи модуль для реалізації класичного списку цілих чисел (задача **10.15**), скласти підпрограми для реалізації додаткових дій над списком:

а) дописати елемент у кінець списку;

б) повернути останній елемент списку;

в) повернути початок списку без останнього елемента.

10.18. Описати модуль для реалізації списку цілих чисел із поточним елементом. Передбачити виконання дій над списком:

1) почати роботу; 2) з'ясувати, чи порожній залишок списку;

3) встати до початку списку; 4) перейти до наступного елемента;

5) поточний елемент; 6) вставити елемент;

7) видалити елемент.

Використовуючи цей модуль, скласти підпрограми:

а) пошуку в списку елемента, рівного заданому числу;

б) сортування списку; в) конкатенації двох списків;

г) присвоєння для списків та обчислення довжини списку;

ґ) перевірки, чи є один список початком іншого та чи входить один список у інший;

д) інверсії списку.

10.19. Використовуючи модуль для реалізації списку цілих чисел із поточним елементом (задача **10.18**), скласти підпрограми для:

а) заміни поточного елемента списку з поточним елементом L числом n;

б) перевірки списку на симетричність;

в) виділення зі списку L n елементів, починаючи з елемента з номером m у новий список L_1;

г) видалення n елементів списку L, починаючи з елемента з номером m;

ґ) друкування двох "сусідів" поточного елемента;

д) вставки елемента з числом n після поточного елемента.

10.20. Описати модуль для реалізації кільцевого списку цілих чисел. Передбачити виконання дій над списком:

1) почати роботу;
2) довжина списку;
3) перейти до наступного елемента;
4) поточний елемент;
5) вставити елемент;
6) видалити елемент.

Використовуючи цей модуль, розв'язати задачі:

а) "лічилка" (задача **10.5**);

б) пошук у списку елемента, рівного заданому числу;

в) присвоєння для списків;

г) знаходження послідовності однакових елементів списку, що йдуть поспіль, максимальної довжини;

ґ) видалити зі списку всі повторні входження елементів;

д) знайти пару елементів списку, різниця між якими є максимальною за абсолютною величиною для всіх пар елементів списку.

10.21. Використовуючи модуль для реалізації кільцевого списку цілих чисел (задача **10.20**), скласти підпрограми для:

а) заміни поточного елемента списку L числом n;

б) перевірки списку на симетричність;

в) виділення n елементів зі списку L, починаючи з елемента з номером m до нового списку L_1;

г) видалення n елементів списку L, починаючи з m-го по відношенню до поточного елемента кільцевого списку.

10.22. Описати модуль реалізації двозв'язного списку цілих чисел. Передбачити виконання дій над списком:

1) почати роботу;
2) з'ясувати, чи порожній список;
3) з'ясувати, чи порожній початок списку;
4) з'ясувати, чи порожній кінець списку;
5) встати до початку списку;
6) встати до кінця списку;
7) перейти до наступного елемента;
8) перейти до попереднього елемента;
9) поточний елемент;
10) вставити елемент перед поточним;
11) вставити елемент після поточного;
12) видалити елемент.

Використовуючи цей модуль, розв'язати задачі:

а) пошуку в списку елемента, рівного заданому числу;

б) обчислення довжини списку;

в) сортування списку;

г) конкатенації двох списків;

ґ) присвоєння для списків;

д) перевірки, чи є один список початком іншого;

е) перевірки, чи входить один список до іншого.

10.23. Використовуючи модуль для реалізації двозв'язного списку цілих чисел (задача **10.22**), реалізувати на базі двозв'язного списку:

а) стек (розв'язати для стека задачу інвертування послідовності символів);

б) чергу (розв'язати для черги задачу "лічилка" (задача **10.5**));

в) дек (розв'язати для дека задачу "лічилка" (задача **10.5**));

г) список із поточним елементом (розв'язати для списку задачу пошуку елемента в списку).

10.24. Використовуючи модуль для реалізації двозв'язного списку цілих чисел (задача **10.22**), скласти підпрограми для:

а) заміни поточного елемента списку L числом n;

б) виділення n елементів зі списку L, починаючи з елемента з номером m, до нового списку L_1;

в) видалення n елементів списку L, починаючи з m-го.

Зауваження. Розглянуті структури даних є лінійними або одновимірними структурами даних. Прикладами плоских або двовимірних структур є **дерева** та **графи**. Відомості про дерева та графи в інформатиці можна отримати з додаткової літератури, наприклад [5, 12, 13].

Розділ 11
ОСНОВИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Об'єктно-орієнтоване програмування (ООП) – одна з парадигм програмування, що розглядає програму як множину "об'єктів", які взаємодіють між собою.

Означення 11.1. Об'єктом називають деяку сутність, що має визначені властивості, поведінку та стан. Множину об'єктів з однаковими властивостями та поведінкою називають **класом**.

Зауваження. Об'єкти, часом, називають **екземплярами класу**.

Кожен клас має **атрибути** (**властивості**) та **операції** (**методи**). Атрибути визначають і зберігають значення властивостей об'єктів класу, а операції – поведінку об'єктів класу. Властивості та методи класу разом називаються його **членами**.

Три основні принципи, на яких базується ООП, – інкапсуляція, наслідування та поліморфізм.

Означення 11.2. Інкапсуляція (**приховування даних**) полягає у приховуванні від зовнішнього користувача (прикладного програміста, іншого об'єкта тощо) деталей реалізації об'єкта, натомість надаючи механізм взаємодії з об'єктом.

Інкапсуляція служить передусім для того, щоб не давати можливості змінювати внутрішній стан об'єкта без його відома. Натомість для взаємодії об'єкта з іншими об'єктами надається набір "офіційних" (загальнодоступних) методів. Цей набір називається **інтерфейсною** частиною або **інтерфейсом** класу. Таким чином, користувач може взаємодіяти з об'єктом тільки через інтерфейс.

Для реалізації інкапсуляції члени класу позначаються як **загальнодоступні** (публічні – англ. public), **захищені** (англ. protected) та **приховані** (**приватні** – англ. private), визначаючи, чи доступні вони для використання відповідно всім (користувачам, об'єктам інших класів), підкласам або лише класу, в якому їх визначено.

Використання інкапсуляції зменшує ймовірність помилок через приведення екземпляра класу до непередбаченого стану.

Означення 11.3. Наслідування – це успадкування властивостей і методів одного класу іншим.

З точки зору наслідування виділяють **клас-предок** (суперклас або батьківський клас) і **клас-нащадок** (підклас). Наслідування може бути одинарним або множинним. За одинарного наслідування клас наслідує властивості та методи тільки одного класу; за множинного – клас може наслідувати властивості та методи кількох класів.

Означення 11.4. Поліморфізмом називають властивість, що дозволяє одне й те саме ім'я використовувати для розв'язання двох або більше схожих, але технічно різних задач.

В ООП поліморфізм означає, що об'єкти двох або більше класів можуть реагувати по-різному на однакові команди.

Зауваження. Класична мова С не є об'єктно-орієнтованою мовою програмування, тому в подальшому робота здійснюватиметься у межах мови програмування С++, яка у свій час розроблялася як об'єктно-орієнтований аналог мови С.

Опис класу в С++ має такий синтаксис:

```
class Cls_name : public Cls_parent {
  private:
    // оголошення прихованих членів класу
    def_private;
  protected:
    // оголошення захищених членів класу
    def_protected;
  public:
    // оголошення загальнодоступних членів класу
    def_public;
};
```

де `Cls_name` – ім'я класу, `Cls_parent` – ім'я класу-предка, `def_private`, `def_protected` , `def_public` – оголошення відповідно прихованих, захищених і загальнодоступних членів класу. Якщо клас-предок відсутній, то команду `": public Cls_parent"` не вказують.

Оголошення членів класу складається з оголошення властивостей і методів класу. Оголошення властивостей не відрізняється від оголошення звичайних змінних, наприклад інструкція

```
t name_property;
```

оголошує властивість `name_property` типу `t`.

Оголошення методу – оголошення заголовку функції-методу. Наприклад, інструкція

```
t name_method(t1 a1,...,tn an);
```

оголошує метод `name_method` типу `t` з аргументами a_1 типу t_1, ..., a_n типу t_n.

Оголошення об'єктів нічим не відрізняється від оголошення звичайних змінних:

```
Cls_name obj, a;
```

Для того, щоб звернутися до властивості чи методу об'єкта, подібно до структур, ім'я об'єкта та відповідного члена розділяють крапкою:

```
obj.name_method(...);
```

Як приклад наведемо оголошення класу Person (*Особа*):

```
class Person {
  private:
    char Name[20];     // Ім'я
    char Surname[30]; // Прізвище
    int BYear;         // Рік народження
  public:
    void Input();      // Зчитати дані
    void Output();     // Надрукувати дані
  };
```

Цей клас має приховані властивості Name, Surname та BYear і загальнодоступні методи Input і Output зчитування даних про особу з клавіатури та виведення даних про особу на екран, відповідно. Оголошений клас Person не має предка. Наступний опис

```
class Student: public Person {
  private:
    unsigned course;
    unsigned session[EX_NUM];
  public:
    void input();
    void output();
    double average();
};
```

визначає клас Student, що є нащадком Person. Клас Student успадковує від класу Person усі властивості класу Person і додає свої властивості course, session і метод average. Клас Student перевизначає (інколи кажуть перевантажує) методи input та output класу Person.

Реалізація методів класу має бути здійснена в тому самому файлі (модулі), де оголошено клас. У реалізації методу його імені передує префікс – ім'я класу та два символи ":", наприклад для вищенаведеного класу Person реалізація методу output виглядатиме так:

```
void Person::output(){
  // Реалізація
}
```

Перевантаження методів input та output є прикладом **статичного поліморфізму** (статичного зв'язування), оскільки рішення про те, яка з функцій викликатиметься – Student::output() або

`Person::output()`, визначається під час компіляції, тобто до початку виконання програми.

Для того, щоб краще зрозуміти статичний поліморфізм, наведемо ще один приклад. Розглянемо клас A та клас B, що є його нащадком:

```
class A {
  ...
  void S(...);
  void Q(...);
  ...
};
class B: public A {
  ...
  void S(...);
  ...
};
```

Як бачимо, клас B наслідує метод Q і перевантажує метод S класу A. Припустимо, що метод Q використовує метод S:

```
void A::Q(...){
      ...
      S(...);
      ...
}
```

Якщо об'єкт X з класу A, а об'єкт Y – з класу B, тоді як під час виклику методу Q з класу A

```
X.Q(...);
```

так і під час виклику методу Q з класу B

```
Y.Q(...);
```

викликатиметься метод S класу A, оскільки, як було сказано раніше, вибір методів відбувається на стадії компіляції.

Часто виникає ситуація, коли потрібно, щоб вибір методу проводився під час виконання програми. Для цього в C++ є **динамічний поліморфізм** (динамічне зв'язування). Для динамічного зв'язування методи-члени мають бути **віртуальними**.

Віртуальні методи позначаються ключовим словом `virtual`, яке стоїть перед описом методу. Таким чином, якщо в описі вищенаведених класів A і B, метод S буде оголошено як віртуальний

```
virtual void S(...);
```

то під час виклику методу `X.Q(...)` використовуватиметься метод S із класу A, а під час виклику методу `Y.Q(...)` – метод S із класу B.

Об'єкти можуть розміщуватися як у статичній, так і в динамічній пам'яті. Казатимемо, що у першому випадку об'єкт статичний, а у другому – динамічний.

Робота з динамічними об'єктами вимагає коректної ініціалізації і знищення об'єкта. Тому, як правило, кожен клас має спеціальний метод, що називається **конструктором** і відповідає за ініціалізацію об'єкта.

Основні правила при використанні конструктора:

а) ім'я конструктора збігається з ім'ям класу;

б) клас може мати скільки завгодно конструкторів, які відрізняються кількістю та типами параметрів;

в) конструктор викликається кожного разу, коли створюється об'єкт даного класу;

г) конструктори у C++ не можуть бути віртуальними.

Інший спеціальний метод, який відповідає за коректне знищення об'єкта з пам'яті, називається **деструктором**. Як і конструктор, деструктор має спеціальне ім'я, що складається з імені класу, перед яким стоїть символ "~". Кожний клас має тільки один деструктор, що не має параметрів.

```
class Cls_name {
  ...
  public:
    Cls_name();      // конструктор без параметрів
    Cls_name(int a); // конструктор з параметром a
    ~Cls_name();     // деструктор
  ...
};
```

Опис статичних об'єктів, для яких визначено конструктори, має відразу включати виклик одного з конструкторів, наприклад,

```
Cls_name obj = Cls_name(100);
```

або скорочено

```
Cls_name obj(100);
```

Вказівники на об'єкти описують як звичайні вказівники, а динамічні об'єкти створюються за допомогою ключового слова `new`. Нехай

```
Cls_name *p_obj;
```

тоді інструкція

```
p_obj = new Cls_name(100);
```

створює новий об'єкт класу `Cls_name`, викликає для нього конструктор, повертає вказівник на новостворений об'єкт і присвоює значення цього вказівника `p_obj`.

Звільнення пам'яті, раніше виділеної під динамічний об'єкт, здійснюється за допомогою інструкції `delete`, Отже,

```
delete p_obj;
```

звільняє пам'ять, що займає об'єкт, на який вказує `p_obj`. Деструктор `~Cls_name` неявно викликається під час знищення динамічного об'єкта.

Приклади розв'язання задач

Приклад 11.1. Описати клас `Student` (*Студент*) на базі класу `Person` (див. ст. 191) і реалізувати програму обчислення розміру стипендії за результатами сесії.

Розв'язок

```
/* Розрахунок стипендії */
#include <stdio.h>
#include <iostream.h>
class person {
  private:
    char name[30];    // Ім'я
    char surname[30]; // Прізвище
    int  byear;       // Рік народження
  public:
    void input();     // Взяти дані про особу
    void output();    // Показати дані про особу
};
void person::input() {
  cout << "Прізвище       : "; gets(surname);
  cout << "Ім'я           : "; gets(name);
  cout << "Рік народження : "; cin >> byear;
}
void person::output() {
  cout << surname << ' ' << name << ' ' << byear;
}
#define MAX_COURSE 5    // максимальний курс
/* кількість іспитів у сесію */
#define EX_NUM 4
#define TRUE 1
class student: public person {
  private:
    unsigned course;          // курс
    unsigned session[EX_NUM]; // оцінки у сесію
  public:
    void input();    // взяти дані про студента
    void output();   // показати дані про студента
    double average();// підрахувати середній бал
};
void student::input() {
  int i;
  person::input();
  cout << "Курс           : ";
```

```
  cin >> course;
  cout << "Оцінки         :\n";
  for (i = 0; i < EX_NUM; i++)
    cin >> session[i];
}
void student::output() {
  int i;
  person::output();
  cout << ' ' << course;
  for (i = 0; i < EX_NUM; i++)
    cout << ' ' << session[i];
}
double student::average() {
  int i, b;
  double s;
  s = 0; b = TRUE;
  for (i = 0; i < EX_NUM; i++){
    s += session[i];
    b = b && (session[i] != 2);
  }
  if (b)
    return s / EX_NUM;
  else
    return 0;
}
#define MAX_STUD 20   // кількість студентів
/* кількість градацій для нарахування стипендії */
#define MAX_COEFF 3
#define MIN_FEE 500  // мінімальна стипендія
/* коефіцієнти для нарахування стипендії */
double coeff[] = {0.00,1.00,1.20};
/* відповідний середній бал */
double ball[] = {0.00,4.00,5.00};
double stipend(double av) {
  int i;
  i = MAX_COEFF;
  do {
    i--;
  } while (av < ball[i]);
  return MIN_FEE * coeff[i];
}
void main() {
  student stud[MAX_STUD];  // масив студентів
  int  i;
```

```
    double av, stip;
    cout << "Введіть студентів:\n";
    for(i = 0; i < MAX_STUD; i++)
      stud[i].input();
    cout << "Результат\n";
    for(i = 0; i < MAX_STUD; i++) {
      av = stud[i].average();
      stip = stipend(av);
      stud[i].output();
      cout << ' ' << av << ' ' << stip << '\n';
    }
  }
```

Приклад 11.2. На базі класу `Auto` (*Авто*) описати класи `Car` (*Легковик*) і `Ttuck` (*Вантажівка*). Використовуючи динамічний поліморфізм, написати програму розрахунку витрат пального для кожного типу автомобілів. У програмі врахувати, що:

а) витрата пального залежить від кілометражу, який пройшов автомобіль, і зростає на 1 % від базової витрати пального через кожні 1000 км шляху;

б) максимальна кількість пасажирів у легковому авто – 5 осіб;

в) витрата пального для легкового авто залежить від кількості пасажирів – 5 % від базової витрати для кожного пасажира;

г) максимальний вантаж для вантажівки – 10 т;

ґ) витрата пального для вантажівки залежить від вантажу та зростає на 10 % від базової витрати для кожної наступної тонни.

Розв'язок

```
#include <iostream.h>
#include <conio.h>
class Auto {
  private:
    int counter;          // Лічильник кілометражу
    double consumption100;// Пального на 100 км
    double fuel;          // Пального у баці
  public:
    Auto(double c, double f); // Конструктор
    // ShowFuel повертає кількість пального у баці
    double ShowFuel();
    // AddFuel збільшує властивість fuel на f
    void   AddFuel(double f);
    double ShowCounter(); // Значення лічильника
    void   PrintData();   // Вивести всі дані
    /* consumption1 - віртуальна функція
       розрахунку витрати пального на 1 км  */
```

```
    virtual double consumption1();
    void   Go(int dist);  // Метод "їхати"
  };
Auto::Auto(double c, double f){
 consumption100 = c;
 fuel = f;
 counter = 0;
}
double Auto::ShowFuel() {
  return fuel;
}
void Auto::AddFuel(double f) {
  fuel += f;
}
double Auto::ShowCounter() {
  return counter;
}
void Auto::PrintData(){
  cout << "Пального  : " << ShowFuel()    << "\n";
  cout << "Лічильник : " << ShowCounter() << "\n";
}
double Auto::consumption1(){
  // враховується п.а) задачі
  return (1 + (counter/1000)) *
              consumption100 / 100;
}
void Auto::Go(int dist) {
  /* у змінній f підраховується кількість
     пального, необхідного для автомобіля,
     щоб пройти відстань dist.
     Оскільки метод consumption1 є віртуальним,
     то (успадкований) метод Go буде правильно
     працювати для всіх дочірніх класів.      */
  double f = consumption1() * dist;
  if (fuel >= f) {
    counter += dist;
    fuel    -= f;
  }
  else
    cout << "Пального недостатньо!\n";
}
class Car : public Auto{
  private:
    int passenger;      // Кількість пасажирів
```

```
  public:
    Car(double c, double f);        // Конструктор
    Car(double c, double f, int p); // Конструктор
    int  ShowPass();    // Повертає к-ть пасажирів
    void PrintData();   // Друкує всі дані
    virtual double consumption1();
  };
/* Для конструкторів Car викликається конструктор
   батьківського класу Auto.                    */
Car::Car(double c, double f) : Auto(c,f){
  passenger = 1;
}
Car::Car(double c, double f, int p): Auto(c,f){
  // враховується п. б) задачі
  if (p == 0)
    passenger = 1;
  else if (1 <= p <= 5)
    passenger = p;
  else
    passenger = 5;
}
int Car::ShowPass(){
  return passenger;
}
void Car::PrintData(){
  cout << "Легковик:\n";
  Auto::PrintData();
  cout << "Пасажирів : " << ShowPass() << "\n";
}
double Car::consumption1(){
  // враховується п. в) задачі
  return Auto::consumption1() *
              (1 + (passenger - 1) * 0.05);
}
class Truck: public Auto {
  private:
    int luggage;        // Вага вантажу
  public:
    // конструктори
    Truck(double c, double f);
    Truck(double c, double f, int l);
    int  ShowLugg();  // Повертає вагу вантажу
    void PrintData(); // Друкує всі дані
    virtual double consumption1();
```

```
  };
  Truck::Truck(double c, double f) :Auto(c,f){
    luggage = 0;
  }
  Truck::Truck(double c, double f, int
l):Auto(c,f){
    // враховується п. г) задачі
    if (l <= 0)
      luggage = l;
    else if (1 <= l <= 10)
      luggage = l;
    else
      luggage = 10;
  }
  int Truck::ShowLugg(){
    return luggage;
  }
  void Truck::PrintData(){
    cout << "   Вантажівка:\n";
    Auto::PrintData();
    cout << "Вантаж: " << ShowLugg() << "\n";
  }
  double Truck::consumption1(){
    // враховується п. ґ) задачі
    return Auto::consumption1() *
                   (1 + luggage*0.1);
  }
  #define fc 5  // Пального на 100 км для легковика
  #define ft 20 // Пального на 100 км для вантажівки

  /* Головна програма */
  void main(){
    int d, p, l;
    double f = 100; // Початкова кількість пального
    char a;
    do {
      // Обираємо транспорт
      cout << "Легковик чи Вантажівка (c/t)?\n";
      a = getch();
    } while (a != 'c' && a != 't');
    if (a == 'c'){
      cout << "Задайте кількість пасажирів: ";
      cin >> p;
      Car c(fc, f, p);
```

```
      do {
        cout << "Задайте дистанцію (0 - Стоп): ";
        cin >> d;
        if (d == 0) break;
        c.Go(d);
        c.PrintData();
        cout << "Поповнити пальне (y/n)?\n";
        if (getch() == 'y'){
          cout << "Скільки залити пального (л)? ";
          cin  >> f;
          c.AddFuel(f);
        }
      } while ( d != 0);
    }
    else {
      cout << "Задайте вагу вантажу (т): ";
      cin >> l;
      Truck c(ft, f, l);
      do {
        cout << "Задайте дистанцію (0 - Стоп): ";
        cin >> d;
        if (d == 0) break;
        c.Go(d);
        c.PrintData();
        cout << "Поповнити пальне (y/n)?\n";
        if (getch() == 'y'){
          cout << "Скільки залити пального (л)? ";
          cin  >> f;
          c.AddFuel(f);
        }
      } while ( d != 0);
    }
  }
```

Приклад 11.3. Описати клас Deque – дек довільних об'єктів. Використовуючи його, розв'язати задачу "лічилка": по колу розташовано n гравців, що мають номери від 1 до n. У лічилці m слів. Починають лічити з першого гравця. m-й гравець вибуває, після чого знову починають лічити з гравця, що є наступним за тим, що вибув і т. д. Треба показати послідовність номерів, що вибувають, за заданих значень n і m.

Розв'язок. Розіб'ємо програму на три частини – модуль роботи з довільними об'єктами, модуль роботи з деком довільних об'єктів і програма, що розв'язує задачу.

Модуль роботи з довільними об'єктами:

```
/* tobject.h */
class TObject {     // Клас довільних об'єктів
  public:
    TObject(){};    // Порожній конструктор
    ~TObject(){};   // Порожній деструктор
};
```

Опис дека майже повністю повторює опис дека цілих чисел (див. приклад **10.3**), розглянутий раніше. Проте сам дек оформимо у вигляді об'єкта, що має властивості `bg`, `en`, методи `isEmpty`, `PutBg`, `PutEn`, `GetBg`, `GetEn`, конструктор `Deque` та деструктор `~Deque`. Модуль для роботи з деком складається з двох частин – файла заголовків `deque.h` і файла реалізації `deque.cpp`.

```
/* deque.h */
#include "tobject.h"
// Посилання на елемент дека
typedef struct delem *dref;
// Елемент дека
struct delem {
  TObject *data;
  dref next, prev;
};
class Deque {               // Клас Дек
  private:
    dref bg, en; // Перший і останній елементи
  public:
    Deque();                // Конструктор
    ~Deque();               // Деструктор
    int isEmpty();          // Чи порожній дек?
    // Додати елемент у початок
    void PutBg(TObject *a);
    // Додати елемент у кінець
    void PutEn(TObject *a);
    // Взяти елемент з початку
    TObject *GetBg();
    // Взяти елемент з кінця
    TObject *GetEn();
};
```

Файл реалізації:

```
/* deque.cpp */
#include <stdlib.h>
#include "deque.h"
Deque::Deque(){
  bg = en = NULL;
```

```
}
int Deque::isEmpty() {
  if (bg == NULL && en == NULL)
    return 1;
  else
    return 0;
}
void Deque::PutBg(TObject *a){
   dref p;
   p = (dref) malloc(sizeof(struct delem));
   p -> data = a;
   p -> next = bg;
   p -> prev = NULL;
   if (isEmpty()) {
     en = p;
   }
   else
     bg -> prev = p;
   bg = p;
};
void Deque::PutEn(TObject *a){
   dref p;
   p = (dref) malloc(sizeof(struct delem));
   p -> data = a;
   p -> next = NULL;
   p -> prev = en;
   if (isEmpty()) {
     bg = p;
   }
   else
     en -> next = p;
   en = p;
};
TObject *Deque::GetBg(){
   if (isEmpty()) return NULL;
   TObject *a;
   dref p;
   a = bg -> data;
   p = bg;
   if (bg == en) {
     bg = en = NULL;
   }
   else {
     bg = p -> next;
```

```
      bg -> prev = NULL;
    }
    free(p);
    return a;
}
TObject *Deque::GetEn(){
  // Даний метод отримується з методу GetBg,
  // якщо скрізь у описі зробити взаємозаміну
  // bg на en і prev  на next.
}
Deque::~Deque(){
  // Деструктор не тільки видаляє дек з пам'яті,
  // а й всі об'єкти, що містяться у деку.
  TObject *a;
  while (!isEmpty()) {
    a = GetBg();
    delete a;
  }
}
```

У головній програмі визначимо клас TPlayer (гравець) як нащадок класу TObject – довільних об'єктів. Цей клас має одну властивість num – номер гравця, конструктор TPlayer – створити гравця з номером a, метод Show – показати гравця та порожній деструктор ~TPlayer. Об'єкти класу TPlayer зберігатимемо у деку довільних об'єктів. Спочатку в кінець дека додаються n об'єктів класу TPlayer з номерами від 1 до n. При цьому new TPlayer(i) створює новий об'єкт з класу TPlayer, викликає для нього конструктор TPlayer(i) та повертає вказівник на новостворений об'єкт. Цей вказівник передається у метод GetEn об'єкта pd як параметр-аргумент. Зауважимо, що метод GetEn як параметр очікує вказівник типу TObject, але, оскільки TPlayer є нащадком класу TObject, то ця операцію буде виконано коректно.

Після цього, поки дек не спорожніє, перекладаємо $m - 1$ разів елементи із початку дека у його кінець, а m-й – за ліком елемент виводимо на екран.

```
/* counter.cpp */
#include <iostream.h>
#include "deque.h"
class TPlayer : public TObject{  // Клас гравців
  private:
    int num;        // Номер гравця
  public:
    TPlayer(int a); // Конструктор
```

```
    ~TPlayer();      // Деструктор
    int Show();      // Повертає номер гравця
};
TPlayer::TPlayer(int a)
  { num = a;}
TPlayer::~TPlayer(){}
int TPlayer::Show(){
  return num;
}
void main(){
  int i;
  int n = 15; // кількість гравців
  int m = 11; // кількість слів у лічильці
  Deque *pd;
  TPlayer *p;
  pd = new Deque();
  for (i = 1; i <= n; i++){
    p = new TPlayer(i);
    pd -> PutEn(p);
  }
  while (!pd -> isEmpty()) {
    for (i = 1; i <= m-1; i++){
      /* pd -> GetBg() є вказівником на змінну
         типу TObject, а p – TPlayer, що є
         нащадком класу TObject.
         Тому використовуємо перетворення типів */
      p = (TPlayer *)  pd -> GetBg();
      pd -> PutEn(p);
    }
    p =  (TPlayer *) pd -> GetBg();
    cout << p -> Show() << " " ;
    delete p;
  }
  delete pd;
  cout << "\n";
}
```

Задачі для самостійної роботи

11.1. Описати клас `Familiar` (*Знайомий*) на базі класу `Person`

```
class Person {
  private:
    char Name[20];    // Ім'я
```

```
    char Surname[30]; // Прізвище
    int BYear;        // Рік народження
  public:
    void Input();     // Зчитати дані
    void Output();    // Надрукувати дані
  };
```

та реалізувати програму пошуку телефону в телефонній книжці.

11.2. Описати клас `Employees` (*Співробітник*) на базі класу `Person` (задача **11.1**) і реалізувати програму розрахунку прибуткового податку.

11.3. Описати клас `Guest` (*Гість*) на базі класу `Person` (задача **11.1**) і реалізувати програму розрахунку платні за проживання у готелі.

11.4. Описати клас `Passenger` (*Пасажир*) на базі класу `Person` (задача **11.1**) та реалізувати програму розрахунку платні за квиток, залежно від відстані маршруту.

11.5. Описати клас `Book` (*Книжка*) та реалізувати програму пошуку книжки за всіма можливими атрибутами у каталозі.

11.6. Описати клас `Polynom` (*Поліном*) та реалізувати методи взяття похідної і добутку поліномів.

11.7. Описати клас `Polynom` (*Поліном*) і реалізувати методи ділення поліномів з остачею та відшукання *НСД* і *НСК* двох поліномів.

11.8. Описати клас `Word` (*Слово*) і реалізувати програму пошуку однокореневих слів у файлі.

11.9. Описати клас `Rational` (*Раціональне_Число*) (див. задачу **6.111**).

11.10. Описати клас `ObjString`, що є об'єктно-орієнтованим аналогом рядкового типу. Реалізувати у ньому всі можливі операції над рядками (задача **6.90**). На базі цього класу розв'язати задачу **6.58**.

11.11. Побудувати клас `ObjComplex` – об'єктно-орієнтований аналог універсального комплексного типу (див. задачу **6.113**).

11.12. Описати клас точок площини `Point`. Описати клас `Dynamic_List`, що є динамічним списком, який містить точки площини. Реалізувати всі необхідні методи для роботи з цими класами та методи обчислення площі та периметра багатокутника, утвореного точками, що містяться у списку.

11.13. Описати клас `Dynamic_Array` (*Динамічний_Масив*), реалізувати методи створення та видалення масиву, читання та зміни елемента. Із використанням динамічних масивів розв'язати задачу: у двох масивах містяться коефіцієнти поліномів степенів m і n, відповідно. Отримати коефіцієнти добутку цих поліномів.

11.14. Описати клас `Stack` (*Стек*) із довільних об'єктів і реалізувати дії над стеками (задача **10.1**). Описати клас `Symbol` (*Символ*) на базі початкового об'єкта та виконати інвертування послідовності символів.

11.15. Описати клас `Queue` (*Черга*) із довільних об'єктів і реалізувати дії над чергами (задача **10.3**). Описати клас `Purchaser` (*Покупець*) на базі початкового об'єкта та розв'язати задачу **10.6**.

11.16. Описати клас `List` (*Список*) із довільних об'єктів і реалізувати дії над списками (задача **10.18**). Описати клас `Element` (*Елемент*) на базі початкового об'єкта та розв'язати задачу **10.18** д).

11.17. Побудувати об'єктно-орієнтований проект для виведення на екран графіків функцій однієї змінної.

11.18. Побудувати об'єктно-орієнтований проект для виведення на екран та переміщення фігур і реалізувати програму розміщення меблів.

11.19. Побудувати об'єктно-орієнтований проект для виведення на екран, переміщення та обертання об'ємних тіл.

11.20. Побудувати об'єктно-орієнтований проект для реалізації двозв'язного списку рядків і реалізувати перегляд текстових файлів.

11.21. Побудувати об'єктно-орієнтований проект для графічного зображення функцій двох змінних.

11.22. Реалізувати об'єктно-орієнтований проект для побудови опуклої оболонки точок тривимірного простору.

Список літератури

1. Абрамов, С.А. Задачи по программированию / С.А. Абрамов, Г.Г. Гнездилова, Е.Н. Капустина, М.И. Селюн. – М. : Наука, 1988. – 224 с.
2. Бауэр, Ф.Л. Информатика. Вводный курс / Ф.Л. Бауэр, Г. Гооз. – М. : Мир, 1976. – 488 с.
3. Буч, Г. Объектно-ориентированное проектирование с примерами применения / Г. Буч. – К. : Диалектика, 1992. – 519 с.
4. Вирт, Н. Систематическое программирование. Введение / Н. Вирт. – М. : Мир, 1977. –184 с.
5. Вирт, Н. Алгоритмы + структуры данных = программы / Н. Вирт. –М. : Мир, 1985. – 406 с.
6. Дейтел, Х.М. Как программировать на С++ / Х.М. Дейтел, П.Дж. Дейтел.– М. : Бином, 2001. –1024 с.: ил.
7. Збірник задач з дисципліни "Інформатика і програмування" / Є.С. Вакал, В.В. Личман, О.В. Обвінцев, В.В. Бублик, Б.П. Довгий, В.В. Попов. –2-ге вид., випр. та доп. –К. :ВПЦ "Київський університет", 2006.– 94 с.
8. Златопольский, Д.М. Сборник задач по программированию / Д.М. Златопольский. – 2-е изд., перераб. и доп. – СПб. : БХВ-Петербург, 2007. –240 с.: ил.
9. Калиткин, Н.Н. Численные методы / Н.Н. Калиткин. – Наука, 1978. – 512 с.
10. Керниган, Б. Язык программирования Си / Б. Керниган, Д. Ритчи. – 3-е узд. – СПб. : Невский диалект, 2001. – 352 с.
11. Керниган, Б. Язык программирования Си. Задачи по языку Си / Б. Керниган, Д. Ритчи, А. Фьюэр. – М. : Финансы и статистика, 1985. – 279 с.
12. Кнут, Д. Искусство программирования для ЭВМ : В 3 т. / Д. Кнут. Т.3. Сортировка и поиск. – М. : Мир, 1978. – 846 с.
13. Крячков, А.В. Программирование на С и С++. Практикум : учеб. пособие / А.В. Крячков, И.В. Сухинина, В.К. Томшин. – 2-е изд., испр.– М. : Горячая линия–Телеком, 2000. – 344 с.: ил.
14. Методические указания и задания к лабораторным занятиям по курсу "Вычислительные машины и программирование" для студентов механико-математического факультета / В.В. Бублик – К. : КГУ, 1986. –60 с.
15. Пильщиков, В.Н. Сборник упражнений по языку Паскаль : учеб. пособие для вузов / В.Н. Пильщиков. – М. : Наука, 1989. – 160 с.
16.Проскуряков, И.В. Сборник задач по линейной алгебре / И.В. Проскуряков. – 11-е изд., стереотип. – СПб. : Лань, 2008. – 480 с.
17. Проценко, В.С. Техніка програмування мовою Сі / В.С. Проценко, П.Й. Чаленко, А.Б. Ставровський. – К. : Либідь, 1993. – 224 с.
18. Подбельский, В.В. Язык программирования Си++ / В.В. Подбельский. – М. : Финансы и статистика, 2002. – 560 с.: ил.
19. Страуструп, Б. Язык программирования С++ / Б. Страуструп. – Вторая редакция. Ч1. – К. : Диасофт, 1993. –264 с.
20. Страуструп, Б. Язык программирования С++ / Б. Страуструп. – Вторая редакция. Ч2. – К .: Диасофт, 1993. – 296 с.
21. Уинер, Р. Язык Турбо Си / Р. Уинер.– М. : Мир, 1991. – 384 с.
22. Уэйт М. Язык Си / М. Уэйт, С. Прата, Д. Мартин. – М. : Мир, 1988. – 512 с.

Зміст