РОЗДІЛ 5
Допоміжні функції, або підпрограми

5.1. Підпрограма – опис розв'язання підзадачі

Програма – це опис розв'язання деякої задачі. Практично в кожній задачі можна виділити окремі допоміжні *підзадачі*. Деякі підзадачі доводиться розв'язувати в багатьох різних задачах, наприклад, обчислення математичних функцій або введення й виведення даних. Для таких *стандартних підзадач* у кожній системі програмування є величезний набір *готових підпрограм*, зібраних у спеціальні набори – *бібліотеки*. Виконання такої бібліотечної підпрограми задається її *викликом*, в якому вказано вирази або змінні, що мають оброблятися під час її виконання.

У практичному програмуванні знати стандартні підпрограми корисно й необхідно, адже застосовувати готові деталі набагато легше, ніж створювати їх самому.

Проте запрограмувати розв'язання всіх можливих підзадач – утопія, тому програмістам доводиться постійно створювати власні підпрограми.

Використання підпрограм втілює загальний принцип "*розділяй і панує*", дозволяє створювати добре структуровані та зрозумілі програми, суттєво полегшує їх проектування й розробку.

5.2. Функція та її виклики

5.2.1. Приклад функції у програмі

Розглянемо засоби створення підпрограм мовою С++. У цій мові підпрограма називається **функцією**. Програма зазвичай містить головну функцію та кілька допоміжних, які описують розв'язання підзадач основної задачі. Ознайомимося зі створенням функцій на простому прикладі.

Приклад. Обчислити периметр трикутника на площині, заданого координатами його вершин.

Нехай (x_1, y_1), (x_2, y_2), (x_3, y_3) – координати вершин трикутника. Щоб обчислити його периметр, потрібні довжини сторін (відрізків із кінцями у вершинах)

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2},$$
$$\sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2},$$
$$\sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2}.$$

Вони обчислюються, по суті, однаково, лише з різними парами точок. Однак ці доволі громіздкі й дуже схожі вирази писати тричі не будемо, оскільки всі вони описують конкретні розв'язання такої *загальної підзадачі*: обчислити довжину відрізка за чотирма координатами його кінців. Довжина залежить від чотирьох величин – **параметрів** підзадачі. Коли розв'язується підзадача, параметри мають конкретні значення-координати – **аргументи** в цьому *конкретному розв'язанні*.

Опишемо розв'язання вказаної підзадачі у вигляді окремої функції з іменем **dist**, параметрами якої є чотири дійсні величини, позначені іменами. У головній же функції напишемо три **виклики** функції **dist**, в яких укажемо координати потрібних нам точок.

```
// обчислення периметра трикутника
#include <iostream>
#include <cmath>
using namespace std;
double dist(double a1, double b1,
```

```
            double a2, double b2)
//distance from (a1,b1) to (a2,b2)
{
  return sqrt((a1-a2)*(a1-a2)+(b1-b2)*(b1-b2));
}
int main()
{ double x1, y1, x2, y2, x3, y3;
  cout << "Enter three pairs of coordinates: ";
  // уведення конкретних координат
  cin>>x1>>y1>>x2>>y2>>x3>>y3;
  cout << "Perimeter = " <<
    dist(x1,y1,x2,y2)+  //виклик – перша сторона
    dist(x1,y1,x3,y3)+  //виклик – друга сторона
    dist(x2,y2,x3,y3);  //виклик – третя сторона
  system("pause"); return 0;
}
```

prog010.cpp

У наведеному тексті перші два рядки після інструкції `using` утворюють **заголовок функції**. Функція описує обчислення довжини відрізка – дійсного числа, яке *повертається з виклику* функції. Тип значень, що повертаються, указано в її заголовку перед іменем функції. Після імені в круглих дужках оголошено параметри функції – дійсні змінні з іменами `a1`, `b1`, `a2`, `b2`. Після заголовку записано коментар, який хоча й не обов'язковий, але дуже корисний: він повідомляє, що має обчислювати ця функція. Далі йде **тіло функції** – блок, що містить послідовність інструкцій. Тут лише одна інструкція, яка задає обчислення й повернення дійсного значення виразу – зарезервоване слово `return` означає "повернути".

У головній функції виклики функції `dist` записано у виразі – значення, що повертаються з викликів, додаються, і ця сума виводиться. ◂

Оголошення параметрів функції, хоча й розташоване за межами блоку функції, діє в цьому блоці до його кінця. Параметри функції, імена яких оголошено в заголовку, в літературі часто називаються **формальними параметрами**, а вирази або імена змінних, записані у виклику, – **фактичними параметрами**.

Функція може не мати параметрів, але круглі дужки в її заголовку обов'язкові.

Виклик функції є виразом і може виступати операндом в інших виразах. Його типом вважається вказаний у заголовку тип значення, яке повертається.

Будь-яке виконання функції, що повертає значення, має закінчуватися виконанням інструкції **`return`** із виразом, тип якого може бути перетворений до типу значень, що повертаються. В іншому випадку виконання функції може мати непередбачувані наслідки.

Вправи

5.1. Написати функцію, що повертає максимальне зі значень двох її дійсних параметрів.

5.2. Написати функцію з цілим параметром n, яка повертає значення $(-1)^n$.

5.3. Написати функцію з дійсним параметром x, яка повертає:
а) -1, 0, 1, відповідно, якщо $x < 0$, $x = 0$, $x > 0$;
б) дробову частину x; в) результат округлення x до цілого.

5.4. Написати функцію, що за значеннями дійсних змінних x та y обчислює значення z за формулою $z = \begin{cases} x + 2, & \textit{якщо } 0 < x \le 3, \\ x - y, & \textit{якщо } x \le 0, \\ x + y, & \textit{якщо } x > 3. \end{cases}$

5.5. Написати функцію, яка за дійсними значеннями x та y обчислює значення $\log_x y$, якщо це можливо, інакше повертає 0.

5.6. Написати функцію, що перевіряє, чи можна утворити трикутник із трьох відрізків із заданими довжинами.

5.7. Відповідно до правил української граматики, після цілого числа, що задає вік людини, пишеться одне зі слів "рік", "роки", "років": 21 *рік*, 34 *роки*, 14 *років*. Написати функцію, яка за заданим цілим числом визначає й повертає результат, що зображує відповідне слово. Наприклад, за числа 21 результатом має бути 1, за 34 – 2, за 14 – 3.

5.2.2. Прототип функції

Ім'я функції необхідно оголосити в тексті програми до того, як воно буде використовуватися. Проте записувати всю функцію вище від її викликів не обов'язково – достатньо записати лише її заголовок.

Заголовок функції зі знаком "`;`" у кінці називається **прототипом функції**. Прототип є інструкцією оголошення функції й повідомляє компілятору, що в програмі є така функція. Після оголошення функцію все одно необхідно означити, тобто описати задані нею дії. **Означення функції** складається із заголовка й тіла.

Загальноприйнятою практикою програмування мовою C++ є запис прототипів функцій програми на її початку (зазвичай після зовнішніх оголошень імен констант, типів і змінних). Після прототипів зазвичай записують головну функцію, а за нею –оголошені та інші функції. Порядок розташування оголошених функцій може бути довільним. Поруч із прототипом функції варто вказати:

– призначення функції, тобто що саме вона виконує та яким є зміст її параметрів;

– **передумови** – умови, що мають справджуватися *перед* її викликом, зокрема умови для її аргументів (фактичних параметрів);

– **післяумови** – умови, що справджуються *після* її виклику, зокрема, як функція змінює глобальні змінні програми;

– поведінку функції за некоректних фактичних параметрів.

Пам'ятайте: прототип функції пишеться не тільки для компілятора, але й для програміста, який має використовувати функцію. Отже, прототип слід писати так, щоб, не читаючи тіла функції, можна було зрозуміти, як нею користуватися.

Запишемо прототип функції обчислення відстані зі с. 88.

```
double dist(double a1, double b1,
            double a2, double b2);
//distance from (a1,b1) to (a2,b2)
```

У прототипі, на відміну від справжнього заголовка функції, можна не вказувати імен параметрів. Зокрема, прототип функції `dist` із погляду синтаксису може мати вигляд

```
double dist(double, double, double, double);
```

Проте він не дає інформації про призначення параметрів функції. Тому це, скоріше, приклад того, як *не слід* писати прототип функції, адже вдало названі параметри й сама функція дозволяють уникнути зайвих пояснень щодо її призначення. Замість скороченого імені **dist** краще було б узяти **distance** (*відстань*), але воно означене в бібліотеках мови C++, тому залишимо **dist**.

Кожен елемент програми (змінна, функція тощо) повинен мати ім'я, яким він позначається. *Перш ніж користуватися елементом, необхідно його оголосити*. Оголошення імені елемента лише описує його, але не створює сам елемент. Проте, щоб використовувати елемент програми, необхідно спочатку створити його в пам'яті програми. Створення елемента задається його означенням.

Функція, складена заголовком і тілом, є означенням, оскільки дії, задані функцією, у вигляді машинних команд записуються в певну ділянку пам'яті. Прототип же лише повідомляє про функцію й не описує жодних дій, тому є оголошенням її імені. Означення й оголошення змінних розглядаються в підрозд. 9.2.

5.2.3. Функція, що не повертає значень

Приклад. Точку площини задано двома її дійсними координатами. Напишемо функцію виведення координат точки у вигляді **(x, y)**, наприклад **(1.5, 3.12)**.

Тут немає значення, яке потрібно повернути, тому функція нічого не повертає. У заголовку таких функцій замість імені типу записується слово **void** (вільний, порожній). Отже, функція виведення координат точки виглядає так:

```
void outPoint(double x, double y)
{ cout << "(" << x << ", " << y << ")"; }
```

Замість рядкової константи **"("** можна було б вивести символ **'('**. Однак, якщо далі виникне потреба вивести, наприклад, ще пробіл після дужки, то запис **'('** призведе до непередбачуваних наслідків під час виконання, а запис **"("** – ні. Отже, обрано варіант, безпечніший для програміста. ◂

Виклик **void**-функції записується в окремій *інструкції виклику функції*, а не як операнд у виразі. Наприклад, виклик функції **outPoint** міг би бути таким:

```
...; outPoint(1.5, 2.3); ...
```

У **void**-функції не може бути інструкцій повернення значення виразу, але можуть бути інструкції повернення без виразу, що мають вигляд **return;**.

Вправи

5.8. Написати функцію, що за дійсними коефіцієнтами a, b рівняння $ax+b=0$ виводить рівняння на екран.

5.9. Написати функцію, що за кількістю розв'язків n і самим розв'язком x рівняння $ax+b=0$ виводить повідомлення про розв'язки рівняння на екран (див. приклад 4.1).

5.10. Написати функцію, що виводить логічне значення у вигляді **Yes** (для **true**) або **No** (для **false**).

5.3. Параметри-значення й параметри-посилання

5.3.1. Два різновиди параметрів

Інструкція **return** здатна повернути з функції *одне значення*. А що робити, коли у функції треба змінити *кілька значень*? Наприклад, у задачі визначення за трьома точками, чи утворюють вони трикутник, потрібно ввести три точки. Ці повторювані дії доцільно оформити функцією, яка має вводити, а отже, і змінювати значення двох координат точки.

Для ілюстрації розглянемо простішу програму з викликом функції, яка має ввести координати точки та присвоїти їх двом своїм дійсним параметрам.

```
#include <iostream>
using namespace std;
void inPoint(double x, double y)
{ cout<<"Enter point (real x and y): ";
  cin >> x >> y;
}
```

```
void outPoint(double x, double y)
{ cout << "(" << x << ", " << y << ")"; }
int main(){
  double x=0, y=0;
  cout << "\"Old value\": ";
  outPoint(x,y); cout<<endl;
  inPoint(x,y);
  cout << "\"New value\": ";
  outPoint(x,y); cout<<endl;
  return 0;
}
```

prog011.cpp

Головна функція спочатку виводить значення координат, якими їх проініціалізовано. Потім, під час виконання виклику функції **inPoint**, координати мали б отримати значення. Мали б, але *не отримують* – про це свідчать ті самі значення координат після виклику. Якщо ввести числа **1** та **1**, то текст у вікні програми буде таким:

```
"Old value": (0, 0)
Enter point (real x and y): 1 1
"New value": (0, 0)
```

А тепер – увага! У заголовку функції перед іменами параметрів **x** та **y** додамо символ **&**.

```
void inPoint(double &x, double &y)
```

Після цього, якщо ввести ті самі числа, нова програма дає інший вихід.

```
"Old value": (0, 0)
Enter point (real x and y): 1 1
"New value": (1, 1)
```

Як бачимо, під час виконання виклику функції **inPoint** змінні **x** та **y** отримали нове значення. Причина в тому, що зі знаком **&** параметри функції стали параметрами *іншого різновиду*.

Параметри, оголошені зі знаком **&** перед іменем, називаються **параметрами-посиланнями**, а без нього – **параметрами-значеннями**.

Знак **&** можна записувати як окремо від імен навколо нього (**double & x**), так і разом із ними: **double&x**, **double& x** або **double &x**.

Отже, у новій функції **inPoint** параметри **x** та **y** є параметрами-посиланнями.

Якщо є вибір, використовувати параметри-посилання чи глобальні змінні, то віддавайте перевагу параметрам-посиланням.

5.3.2. Повернення значень за допомогою параметрів

Розглянемо функцію, яка має ввести й повернути ціле число із заданого діапазону [*min*; *max*]. Якщо користувач уведе значення за межами діапазону, то його потрібно замінити значенням лівої межі діапазону. На перший погляд здається, що цю функцію можна написати з прототипом

```
int f(int min, int max);
```

Утім, коли введене значення заміняється на **min**, про це *корисно повідомити*. Ураховуючи це, змінимо прототип функції на такий (необхідні коментарі до нього напишіть самостійно):

```
bool f(int min, int max, int &res);
```

Отримане число повертається через параметр-посилання **res**, а функція повертає ознаку того, що були помилки під час уведення числа (**true**, якщо користувач задав число поза діапазоном, інакше **false**).

```
bool f(int min, int max, int &res)
{ cout << "Enter integer in [" << min <<
          ", " << max << "]: ";
  cin >> res;
  if(min<=res && res<=max) return false;
  res=min;
  return true;
}
```

Зазначимо, що після виконання інструкції **return false;** подальші інструкції з тексту функції вже не виконуються, тому писати для них секцію **else** не потрібно. Це відсікання оброблених варіантів дуже часто дозволяє уникати великої вкладеності інструкцій розгалуження **if**.

Отже, у функції **f** параметри **min** і **max** є параметрами-значеннями, **res** – параметром-посиланням. Наведемо фрагмент головної функції з викликом функції **f**.

```
int main()
{ int a, b, c;
  bool isERROR;
  a=1; b=9; isERROR=false;
  isERROR=f(a,b,c);
  ...
  return 0;
}
```

5.3.3. Підстановка аргументів на місце параметрів

У цьому пункті розглянемо, що відбувається з параметрами-значеннями й параметрами-посиланнями під час виконання виклику функції.

Параметр-значення. На початку виконання виклику утворюється *нова змінна*, яку надалі позначає ім'я параметра-значення. Аргументом, що відповідає параметру-значенню, може бути *довільний вираз*. Значення аргументу обчислюється та присвоюється цій новій змінній.

Описаний спосіб передачі даних у функцію називається підстановкою аргументу на місце параметра **за значенням**.

Коли виконуються інструкції з тіла функції, усі зміни, що стосуються параметра-значення, відбуваються в його власній змінній і не впливають на пам'ять, пов'язану з аргументом у виклику. Отже, у функції `f` із прикладу імена `min` і `max` позначають власні ділянки пам'яті, ніяк не пов'язані зі змінними `a` та `b` головної функції.

Параметр-посилання. У виклику аргумент, що відповідає параметру-посиланню, має позначати *змінну того самого типу, що й у параметра*. У найпростішому випадку це ім'я змінної, тип якої збігається з типом параметра. На початку виконання виклику визначається посилання на цю змінну (її адреса) і передається до функції. Під час виконання виклику функції ім'я параметра-посилання позначає змінну, задану аргументом, і всі *дії з параметром-посиланням насправді виконуються над змінною, яка відповідає аргументу*.

Описаний спосіб передачі даних у підпрограму називається підстановкою аргументу на місце параметра **за посиланням**.

Коли виконуються інструкції з тіла функції, зміни параметра-посилання відбуваються в пам'яті, що відповідає аргументу, тобто *є змінами аргументу*. Отже, у другому варіанті функції `f` під час виконання її виклику параметр `res` позначає змінну `c` головної функції, завдяки чому вона отримує нове значення. У

подібних випадках кажуть, що *значення повертається за допомогою параметра-посилання*.

Приклад. Різницю між параметрами-значеннями й параметрами-посиланнями функції `f` схематично проілюстровано на рис. 5.1.

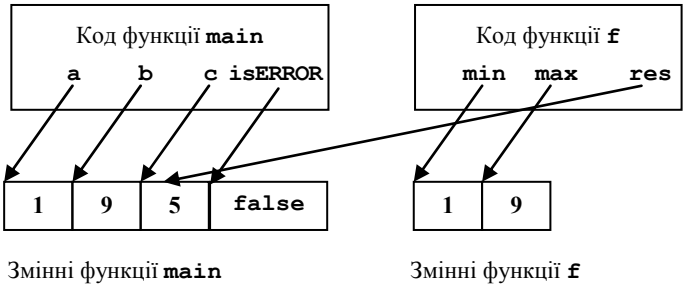

Рис. 5.1. Параметри-значення й параметри-посилання

Проілюструємо суттєві моменти виконання наведеної вище програми за умови введення числа `33`. Стовпчики таблиці зображують змінні, тобто ділянки пам'яті, послідовні рядки – зміни стану пам'яті. Позначимо змінні, що є параметрами функції, додавши її ім'я: `f.min`, `f.max`, `f.res`. Оскільки `f.res` позначає ділянку пам'яті змінної `c`, то для спрощення в таблиці імені `c` і параметру `f.res` відповідає один і той самий стовпчик.

Що виконується	**Усі наявні змінні**					
Утворення змінних головної функції	`a`	`b`	`c`	`isERROR`		
`a=1; b=9; isERROR=false;`	1	9	?	false		
Виклик `f(a,b,c)`	1	9	?	false		
Утворення змінних функції `f`	1	9	`f.res`	false	`f.min`	`f.max`
Присвоювання `min` і `max`	1	9	?	false	1	9
`cin>>res;`	1	9	33	false	1	9
`if(...)...`	1	9	1	false	1	9
Закінчення виклику `f`	1	9	1	false		
`isERROR=f(a,b,c)`	1	9	1	true		

Зауважимо: якби в тілі функції було змінено **min** або **max**, це б не вплинуло на значення **a** або **b**, оскільки іменам **min** і **max** відповідають власні змінні.

Якщо ознака помилки, яку повертає виклик **f(a,b,c)**, далі не потрібна, то інструкцію присвоювання з викликом **isERROR=f(a,b,c);** можна замінити інструкцією виклику функції **f(a,b,c);**. Значення, яке повертає функція, далі просто ігнорується. ◂

Вправи

5.11. Що буде надруковано за програмою?

а)

```
#include <iostream>
using namespace std;
int f(int a)
{ a+=1; return a; }
int g(int &x)
{ x=2; return x; }
int main(){
  int a=3, b=44;
  cout << f(a);
  cout<<' '<< g(b);
  cout<< ' ' << a <<
        ' '<< b;
  system("pause");
  return 0;
}
```

б)

```
#include <iostream>
using namespace std;
int f(int& a)
{ a+=4; return a; }
int g(int x)
{ x=3; return x; }
int main(){
  int a=2, b=1;
  cout << f(a);
  cout << ' ' << g(b);
  cout << ' ' << a <<
        ' '<< b;
  system("pause");
  return 0;
}
```

5.12. Що буде надруковано за програмою?

а)

```
#include <iostream>
using namespace std;
int a=1, b=1, c;
int f(int x, int &y)
{ x+=1; y+=2; c=a+b;
  return x*y; }
int main(){
  cout<< f(a,b);
  cout<<' '<< a <<' '<<
       b <<' ' << c;
  system("pause");
  return 0;
}
```

б)

```
#include <iostream>
using namespace std;
int a=2, b=2, c;
int f(int & x, int y)
{ x+=1; y+=2; c=a+b;
  return x+y; }
int main(){
  cout<< f(a,b);
  cout<<' '<< a << ' '<<
       b << ' ' << c;
  system("pause");
  return 0;
}
```

5.3.4. Функція обмінює місцями два значення

Розглянемо задачу: увести три цілих числа й вивести їх, упорядкувавши за неспаданням.

Уведемо числа в змінні **a**, **b**, **c**, за потреби обміняємо місцями їх значення так, щоб справджувалися нерівності $a \le b$ та $b \le c$, і виведемо ці значення. Уточнимо дії з упорядкування значень.

```
if(a>b) обміняти місцями значення a та b; // a ≤ b
if(b>c) обміняти місцями значення b та c;
// b≤c, a≤c, тобто значення c - найбільше,
// але a≤b може стати хибним, тому:
if(a>b) обміняти місцями значення a та b. // a ≤ b
```

У цьому алгоритмі неважко побачити підзадачу "обміняти місцями значення двох змінних", яка розв'язується тричі з різними парами змінних. Для цієї задачі напишемо функцію **swap** із двома параметрами-посиланнями.

```
#include <iostream>
using namespace std;
void swap(int&, int&); // прототип функції обміну
int main(){
  int a, b, c;
  cout << "Enter three integers: ";
  cin>>a>>b>>c;
  if(a>b) swap(a,b);
  if(b>c) swap(b,c);
  if(a>b) swap(a,b);
  cout<<a<<' '<<b<<' '<<c<<'\n';
  system("pause"); return 0;
}
void swap(int& x, int& y)
{ int t=x; x=y; y=t; }
```

prog012.cpp

Коли виконуються перший і третій виклики функції **swap**, її імена **x** та **y** позначають змінні **a** та **b**, а коли другий – **b** та **c**.

5.3.5. Параметр-посилання чи параметр-значення?

Наведемо міркування, що дозволяють визначати потрібний різновид параметра функції – параметр-значення чи параметр-посилання.

Якщо після виклику підпрограми використовується *старе значення* аргументу або аргументом може бути довільний вираз, то йому має відповідати *параметр-значення*.

Якщо після виклику підпрограми має використовуватися *нове значення* аргументу, отримане саме під час виконання виклику, то параметр має бути *параметром-посиланням*.

Вправи

5.13. Написати функцію, що вводить коефіцієнти квадратного рівняння $ax^2+bx+c=0$. (Зверніть увагу: має виконуватись умова $a \neq 0$.)

5.14. Написати функцію, що вводить коефіцієнти a, b, c рівняння прямої $ax+by+c=0$. (Коефіцієнти a та b не повинні одночасно бути нульовими.)

5.4. Переозначення функцій

Приклад. Припустимо, у програмі потрібні дві функції: одна має обчислювати максимум значень двох цілих параметрів, інша – трьох. Можна написати функції з іменами, наприклад **max2** і **max3**. Проте мова C++ дозволяє дати їм одне й те саме ім'я **max**, що виглядає природніше. Розглянемо ці функції в такій програмі:

```
#include <iostream>
using namespace std;
int max(int, int);      // два різних прототипи
int max(int, int, int); // однойменних функцій
int main()
{ int a, b, c;
  cout << "Enter three integers>";
  cin >> a >> b >> c;
  cout << max(a,b) << " " << max(a,b,c) << endl;
  system("pause"); return 0;
}
int max(int x, int y) // два параметри
{return x>y ? x : y;}
int max(int x, int y, int z) // три параметри
{int t=max(x,y); return t>z ? t : z;}
```

prog013.cpp ◀

Оголошення різних об'єктів програми з тим самим ім'ям називається **переозначенням імені**.

У мові C++ *дозволені переозначення функцій* і *заборонені переозначення змінних*.

Однойменні функції повинні мати різні кількості параметрів або різні послідовності типів значень, що повертаються, і параметрів. При цьому відмінність таких функцій лише за типом значень, що повертаються, є помилкою.

```
int f(double, double);
double f(double, double); // помилка!
```

У викликах функції типи аргументів можуть відрізнятися від типів параметрів. Для виконання виклику компілятор обирає ту з однойменних функцій, кількість і типи параметрів якої збігаються з кількістю й типами аргументів у виклику. Якщо такої немає, то обирається функція, прототип якої утворюється шляхом автоматичних перетворень типів аргументів у виклику. Якщо цей вибір неоднозначний, то компілятор повідомляє про помилку. Наприклад, за наявності прототипів

```
int f(double, int);
int f(int, double);
```

виклик `f(1,2)` не дозволяє компілятору визначити функцію для виконання виклику, і для компілятора це *помилка*. Подібних ситуацій слід уникати.

Контрольні запитання

5.1. Що таке підпрограма й функція в мові C++?
5.2. Що таке виклик функції?
5.3. Що таке прототип функції?
5.4. Які елементи заголовка функції можуть бути відсутні в її прототипі?
5.5. Що таке формальні й фактичні параметри (параметри та аргументи)?
5.6. Назвіть особливості в записі й викликах функцій, що не повертають значень.
5.7. Чим відрізняється підстановка аргументів на місце параметрів-значень і на місце параметрів-посилань?

5.8. Чи можна використовувати ім'я, оголошене у функції, в інших функціях, записаних після неї?
5.9. В яких ситуаціях параметр функції має бути параметром-посиланням?
5.10. Що таке переозначення імені?
5.11. Чи можуть заголовки переозначених функцій відрізнятися лише типом значень, що повертаються?

Задачі

5.1. Написати програму, що за трьома точками площини визначає, чи утворюють вони гостро-, прямо- або тупокутний трикутник. (*Вказівка*: можна використати елементи задачі 4.2, с. 85).
5.2. Модифікувати програму задачі 4.3 (с. 85) так, щоб для введення рівняння, знаходження його розв'язків і повідомлення результату використовувалися функції.
5.3. Написати програму, що для прямої $ax+by+c=0$ визначає, чи лежать точки з координатами (x_1, y_1) і (x_2, y_2) у різних півплощинах відносно неї.
5.4. Написати програму, що визначає, чи мають дві прямі на площині спільні точки.