Лабораторна робота №3. Організація функцій в С++

Мета: засвоїти методи створення та організації функцій в С++.

Теоретичні відомості

Один з ефективних способів створення великих програм, технологія низхідного проектування, полягає в їх конструюванні за принципом «**розділяй і пануй**»: програма розглядається як набір маленьких фрагментів, кожний з яких виконує певну логічно завершену дію, може бути виконаний декілька разів та є більш керованим, ніж програма в цілому. Такий невеликий фрагмент програмного коду називається **підпрограмою**. Підпрограма позначається ідентифікатором, тобто має власне унікальне ім'я. Вказування в тексті програми імені підпрограми рівнозначне запису всіх їх операторів і називається викликом підпрограми.

Отже, **підпрограма** – це іменована частина програми, котра описує деякі обчислення і може бути викликана з будь-якого місця програми.

Таким чином, для багаторазового виконання деякого програмного коду достатньо записати його один раз у підпрограмі, і надалі, в разі потреби, вказувати лише ім'я підпрограми. Концепція програмування, що ґрунтується на використанні підпрограм як стандартних блоків для створення нових програм, отримала назву **повторне використання коду**. Мови, в яких реалізовано механізми використання підпрограм, називаються процедурно-орієнтованими. Мови С\С++ належать до таких мов.

У мовах С\С++ підпрограми існують у вигляді функцій. Виклик деяких функцій приводить до виконання дій, котрі в ній описані у вигляді операторів. Інші функції повертають деяке значення в точку її виклику.

Розрізнюють **два основні різновиди функцій:**

1) стандартні убудовані функції, які є складовою частиною мови програмування, наприклад: `sin()`, `pow()` тощо;

2) функції, які створюються безпосередньо користувачем для власних потреб.

Функції можна поділити на функції без параметрів, функції з параметрами та функції, що повертають значення. **Параметрами** називаються змінні, за допомогою яких відбувається передача даних до функції, що викликається, із програмного блоку, що здійснює виклик. Отже, функції без параметрів можуть бути незалежними від зовнішніх даних, а функції з параметрами обов'язково їх використовують.

Оголошення функції користувача здійснюють або у вигляді **прототипу** (розглядатиметься далі), або у вигляді **означення функції**, яке записують раніше оголошення функції `main()`.

Функції типу `void` без параметрів

Синтаксис означення функції `void` без параметрів є таким:

```
void <ім'я функції> ()
{
    <оголошення ідентифікаторів змінних>
    <оператори>
}
```

Тут `void` – тип значення, що повертає функція; `<ім'я функції>` – унікальний в межах програми ідентифікатор функції, за яким вона викликається; `<оголошення ідентифікаторів змінних>` – оголошення ідентифікаторів, що використовуватимуться лише в межах функції; `<оператори>` – оператори, які виконуються під час виклику функції, їх називають також **тілом функції**.

Тип void означає, що функція не повертає жодного значення в точку її виклику. В такому випадку виклик функції типу void без параметрів здійснюється за її ім'ям:

```
<ім'я функції ();>
```

Приклад 1. Введення двох чисел та обчислення НСД.

```
//example 3_1.cpp Програма вводу двох чисел та обчислення їх НСД
//Приклад 1 до лабораторної роботи №3
//Автор: Кудін О.В., 20.09.2017
#include "stdafx.h"                                  // Включення заголовних файлів
#include <iostream>
#include <cmath>
using namespace std;                          // Визначення простору імен
int a;                                        // Оголошення змінних
int b;

void input()                                  // Визначення функції типу void без параметрів
{
	cout << "Enter int a, int b" << endl;          // Вивід строки з запрошенням вводу
даних
	cin >> a >> b;                                // Ввід даних

}
int main()
{
	cout << "Hello, world"<<endl;                 // Вивести повідомлення
	input();                                      // Виклик функції
	if ((a == 0) && (b == 0))
	{
		cout << "error: a=0 and b=0 " << endl;   // Вивід строки з результатом
обчислень
		return 0;
	}
	for (int i = fmin(a, b); i > 0; i--)
	{
		if ((a%i == 0) && (b%i == 0))
		{
			cout << "GCD is: " << i << endl;  // Вивід строки з результатом
                                                  //обчислень
			return 0;
		}
	}
}
```

Функції типу `void` з параметрами

Повний синтаксис заголовку функції `void` з параметрами має вигляд:

`void <ім'я> (<тип ім'я>, …, <тип ім'я>)`

Параметри `(<тип ім'я>, …, <тип ім'я>)`, що вказуються в заголовку функції при її описі, називаються **формальними параметрами**.

Параметри, що вказуються при виклику функції, називаються **фактичними параметрами**.

Коректність передачі параметрів ґрунтується на їхньому порядку перерахування в заголовку функції і сумісності по присвоюванню між відповідними фактичними і формальними параметрами.

Багато функцій мають кілька параметрів. Задача програміста – переконатися, що параметри, що він указує при виклику (фактичні параметри), відповідають за змістом формальним параметрам. Компілятор може перевірити тільки очевидні випадки — неправильне число чи параметрів несумісність типів.

Приклад 2. Пошук простого числа за його номером.

```
//example 3_2.cpp Пошук простого числа за його номером
//Приклад 2 до лабораторної роботи №3
//Автор: Кудін О.В., 20.09.2017
#include "stdafx.h"               // включення заголовних файлів
#include <iostream>
#include <cmath>
using namespace std;              // визначення простору імен
int number,j,n;                   // номер шуканого простого числа;лічильник
                                  // простих чисел; число, що перевіряється на простоту
bool flag;                        // ознака того, що число є простим

void IsSimple(int a)              // визначення функції void з параметром (перевірка
                                  // числа на простоту)
{
      int k=2;                    // визначити перший дільник
      flag = true;                // вважати простим
      while (k <= (int)sqrt(a) && flag) // перебрати всі можливі дільники
      {
            if (a%k == 0)               // якщо k ділить число, що перевіряється
                  flag = false;         // то число складене і потрібно виключити flag
            k++;                        // наступний дільник
      }
}
int main()
{
      cout << "Search prime number program"<<endl;    // вивести повідомлення
      cout << "Enter a number " << endl;
      cin >> number;                            // вивести номер шуканого простого числа
      j = 1;                                    // задати лічильник простих чисел
      n = 1;                                    // задати число, що перевіряється на
                                                // простоту
      while (j < number)                        // поки лічильник не досягнув заданого
                                                // номера
      {
            n++;                                // перейти до наступного числа
            IsSimple(n);                        // перевірити його на простоту
            if (flag)                           // якщо число просте
                  j++;                          // збільшити лічильник простих чисел
      }
      cout << "j=" << j << " prime number=" << n << endl;
```

```
}
```

Функції, що повертають значення

Синтаксис оголошення функції, що повертає значення будь-якого типу:

```
<ім’я типу> <ім’я функції> (<тип ім’я>,…, <тип ім’я>)
{
    <оглошення імен>
    <оператори>
}
```

Тут `<ім’я типу>` – тип значення, що повертається функцією, яке не може бути `void`; `<ім’я функції>` – ідентифікатор функції, за яким здійснюється її виклик; `<тип ім’я>` – список змінних та їх ідентифікаторів; `<оглошення імен>, <оператори>` складають тіло функції.

Усі функції, які в своєму заголовку вказують тип значення, що повертається, окрім типу void, мають виконувати принаймні один оператор повернення, інакше компілятор видає повідомлення про помилку:

«… must return a value»

Синтаксис оператора повернення такий:

```
return <вираз>;
```

Тут `<вираз>` – послідовність операндів та операцій, тип значення виразу такий, який вказано у заголовку функції.

Під час виконання функції останнім має виконуватись оператор повернення `return`. Значення виразу, що повертається цим оператором, вважатиметься значенням функції. Якщо внаслідок використання конструкцій розгалуження виконання функції може завершуватися різними операторами, то операторів повернення значень виразів має бути декілька.

Приклад 3. Обчислення довжини сторін трикутника.

```
//example 3_3.cpp Обчислення довжини сторін трикутника
//Приклад 3 до лабораторної роботи №3
//Автор: Кудін О.В., 20.09.2017
#include "stdafx.h"                 // включення заголовних файлів
#include <iostream>
#include <cmath>
using namespace std;                // визначення простору імен

float x_1, y_1, x_2, y_2, x_3, y_3;      // координати вершин трикутника
float d1, d2, d3;                        // довжини сторін трикутника
// обчислення відстані між двома точками
float distance(float a1, float b1, float a2, float b2)
{
      return sqrt(pow((a1 - a2), 2) + pow((b1 - b2), 2));
}
// введення координат вершин трикутника
void init()
{
      cout << "Enter triangle points coordinates" << endl;
      cout << "coordinates x1,y1 " << endl;
      cin >> x_1; cin >> y_1;
      cout << "coordinates x2,y2 " << endl;
```

```
	cin >> x_2; cin >> y_2;
	cout << "coordinates x3,y3 " << endl;
	cin >> x_3; cin >> y_3;
}
// обчислення довжин сторін трикутника
void solution()
{
	d1 = distance(x_1, y_1, x_2, y_2);
	d2 = distance(x_2, y_2, x_3, y_3);
	d3 = distance(x_1, y_1, x_3, y_3);
}
// вивід результатів обчислень
void browse()
{
	cout << "length1=" << d1 << endl;
	cout << "length2=" << d2 << endl;
	cout << "length3=" << d3 << endl;
}
// головна підпрограма
int main() {
	cout << "Calculate lengths of a triangle" << endl;
	init();
	solution();
	browse();
	return 0;
	}
```

Прототипи функцій

Програма, яка записана мовами C\C++, складається з функцій, оголошення або визначення яких передує їх виклику. Оголошення заголовка функції, яке містить тип значення, що повертає функція, її ім'я, список параметрів, називається **прототипом функції**.

Наприклад, оголошення (прототип) функції обчислення середнього арифметичного двох цілих чисел може мати вигляд:

```
float mean (int a, int b);
```

Тут `mean` – ім`я функції; `a` і `b` – формальні вхідні аргументи (параметри), які за виклику набувають значень фактичних параметрів; `float` – тип функції, який безпосередньо є типом результату виконання операторів функції. Результат повертається оператором `return`.

Крім оголошення, кожна функція повинна мати визначення (реалізацію). Визначення функції, окрім заголовку, містить тіло функції (команди, які виконує функція) і команду повертання результату `return`:

```
float mean (int a, int b)
{
	float sr;
	sr=(a+b)/2.0;
	return sr;
}
```

Приклад 4. Використання прототипу функції.

```
//example 3_4.cpp Використання прототипу функції
//Приклад 4 до лабораторної роботи №3
//Автор: Кудін О.В., 20.09.2017
#include "stdafx.h"		// включення заголовних файлів
```

```
#include <iostream>
#include <cmath>
using namespace std;              // визначення простору імен

void butler(void);                /* прототип функції в стандарте ISO/ANSI C */
int main(void)
{
       cout<<"Я вызываю дворецкого.<<\n";
       butler();
       cout<<"Да. Принесите мне чай и овсянку \n";
       return 0;
}

void butler(void)                 /* визначення (реалізація) функції */
{
       cout<<"Вы звонили, сэр?\n";
}
```

Правила організації функцій

1) Фактичними параметрами можуть бути константи, змінні чи вирази. В останньому разі спочатку обчислюється значення виразу, а потім результат передається у функцію.

2) Імена змінних, які є фактичними параметрами, можуть не збігатися з іменами формальних параметрів.

3) Типи змінних, які є фактичними параметрами, мають збігатися з типами відповідних формальних параметрів чи то мати можливість бути перетвореними до типів формальних параметрів.

4) Якщо кількість чи типи фактичних параметрів не збігаються з формальними параметрами, це призведе до помилки з відповідним повідомленням.

5) Послідовність змінних, які є фактичними параметрами, має збігатися з послідовністю формальних параметрів.

6) У прототипі функції необов’язково задавати імена формальних параметрів, достатньо їх оголосити, наприклад у такий спосіб:

```
float mean(int, int);
```

7) Якщо функція не отримує жодного параметра, слід у заголовку функції ставити порожні дужки (опустити дужки не можна) чи записати у дужках `void`:

```
double func();
double func(void);
```

8) Тип значення, яке повертає функція, може бути яким-завгодно, але не може бути масивом чи функцією. Наприклад, таке оголошення є помилкове:

```
int [10] func(); // Помилка!
```

Локалізація імен

Кожний ідентифікатор у програмі характеризується областю дії імені або областю видимості. Область видимості ідентифікатора – це область програми, в якій можна посилатися на даний ідентифікатор. На деякі ідентифікатори можна посилатися в будь-якому місці програми, де синтаксисом мови це не заборонено, на інші – тільки в певний частинах програми. У мова C\C++ припускається довільна послідовність і кількість блоків, в яких іменуються ті чи інші ідентифікатори. Існують такі області дії ідентифікатора: блок, функція, файл і прототип функції. Взагалі область дії імен поширюється від точки, де ідентифікатор було оголошено, до кінця блоку, в якому це оголошення відбулося. Кінцем блоку вважається права фігурна дужка «}».

Отже, функції можуть містити власні оголошення констант і змінних. Усі ідентифікатори, оголошені всередині функції, є локалізованими в ній, тобто вони є невидимими зовні підпрограми. Такі ідентифікатори називаються локальними. Зокрема, локальними є будь-які параметри функції. Мета локалізації змінних полягає в тому, щоб надати доступ лише до тих даних, які потрібні для формулювання задач, що їх розв’язують функції, і зробити в такий спосіб постановки обчислювальних задач незалежними від методів їх розв’язань.

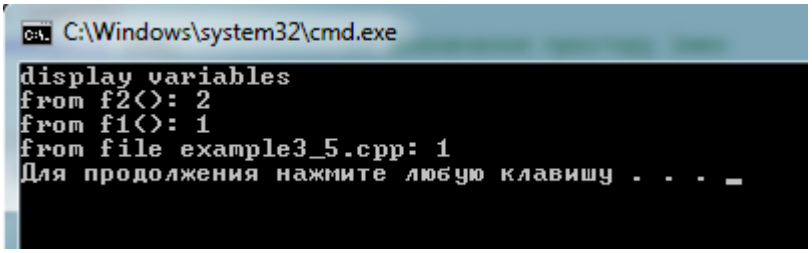
Крім власних локальних імен всередині функції видимими є й ідентифікатори верхнього рівня, тобто ідентифікатори, що їх було оголошено поза будь-якої функції на початку файла. Ці ідентифікатори називаються глобальними.

Приклад 5. Локалізація імен.

```
//example 3_5.cpp Локалізація імен
//Приклад 5 до лабораторної роботи №3
//Автор: Кудін О.В., 20.09.2017
#include "stdafx.h"        // включення заголовних файлів
#include <iostream>
#include <cmath>
using namespace std;       // визначення простору імен

int i;                     // глобальна змінна, її область дії – весь файл
void f2() {
       int i = 2;          // локальна змінна, її область дії – функція f2()
       cout << "from f2(): " << i << endl;
}
void f1() {
       i = 1;              // присвоєння глобальній змінній
       f2();
       cout << "from f1(): " << i << endl;
}
int main() {
       cout << "display variables " << endl;
       i = 3;              // присвоєння глобальній змінній
       f1();
       cout << "from file example3_5.cpp: " << i << endl;
       return 0;
       }
```

Результат роботи наведеної програми наступний



Процес виклику підпрограми. Програмний стек

Під час виклику підпрограми їй передається керування. По завершені роботи вона повертає курування програмі, що її викликала, в ту точку, з якої виклик було здійснено. Перша команда підпрограми називається **точкою входу**, а адреса такої команди – адресою точки входу. Оператор, що продовжує виконання програми по завершенні роботи підпрограми, називається **точкою повернення** із підпрограми. Точка повернення із функції – це наступний за її викликом оператор.

Коли здійснюється виклик функції, точка повернення з неї запам'ятовується і зберігається до завершення роботи цієї функції. Для збереження точки повернення використовується ділянка оперативної пам'яті. Крім того, під час виклику функції певні ділянки пам'яті зіставляються з її параметрами та локальними змінними. Сукупність усіх цих ділянок пам'яті називається локальною пам'яттю функції. До її локальної пам'яті додається ділянка для збереження значення, яке функція повертає. Ця ділянка ставиться у відповідність до імені функції.

Перш ніж розпочнеться виконання операторів тіла функції, для неї виділяється локальна пам'ять і в цій пам'яті записується точка повернення. Потім обчислюються значення тих аргументів, що відповідають параметрам-значенням. Ці значення копіюються в локальну пам'ять, таке копіювання називається підстановкою аргументів за значенням.

Можна сформулювати наступний алгоритм виклику функції.

1. Для функції виділяється локальна пам'ять.
2. В локальній пам'яті обчислюється та запам'ятовується точка повернення з функції.
3. Обчислюються значення аргументів, що відповідають параметрам-значенням.
4. Виконуються оператори тіла функції. У локальній пам'яті запам'ятовується значення, яке функція повертає.
5. Здійснюється повернення з функції. З локальної пам'яті функції копіюється значення, яке функція повертає. Управління передається команді, що адресується точкою повернення з функції.
6. Ділянка оперативної пам'яті, що була задіяна під локальну пам'ять, вважається вільною.

Виконання основної програми починається після завантаження її коду в оперативну пам'ять комп'ютера. При цьому відбувається виділення пам'яті для її змінних. Ці змінні доступні з програми протягом усього часу її

виконання, і тому називаються **статичними**. Область пам'яті, що виділяється під програму та її змінні, також називається статичною. Із локальними іменами, як правило, зіставляються ділянки іншої області пам'яті – **автоматичної**. Під час виконання викликів функції пам'ять виділяється та звільняється автоматично, без явних вказівок у програмі.

Виділення та звільнення ділянок пам'яті під час виконання викликів функції відбувається за принципом «останнім прийшов – першим пішов» (Last In – First Out, LIFO). Якщо складати книжки в стопку і брати їх тільки зверху, то книжка, що потрапила у стопку останньою, забирається першою. Така стопка називається стеком, а кладуть і забирають книжки за принципом LIFO. Тому, автоматична пам'ять, що виділяється для функцій програми, називається ще **програмним стеком**. У програмному стеку, як і у стопці книжок, доступним є завжди тільки один елемент – верхній, що зветься **вершиною стеку**.

Завдання до лабораторної роботи

Розробити алгоритм (у вигляді блок-схеми) та програму переводу натуральних чисел з десяткової системи числення до двійкової і навпаки. Програма повинна задовольняти наступним вимогам:

1. Передбачити введення даних з клавіатури та виведення результатів на екран консолі.
2. Передбачити діалогове меню для вибору напряму перетворення системи числення (з десяткової в двійкову або з двійкової в десяткову).
3. Для кожної окремої дії програми (введення чисел, виведення результату, перевід в двійкову систему, перевід в десяткову систему) передбачити окрему функцію.
4. Обов'язковим є використання коментарів (відповідно до прикладів 1-5) та правил вибору ідентифікаторів в програмі (див. лаб. №2).

Контрольні запитання

1. Які види підпрограм підтримує мова C\C++?
2. Алгоритм виклику функції.
3. Що таке прототип функції?
4. Поясніть відмінність між локальними та глобальними змінними.
5. Що таке програмний стек?