**Лабораторна робота № 12** Розробка програми з використанням власних функцій

**Мета:** навчитися складати програми з власними функціями**.**

**Завдання**

1. Ознайомтеся з теоретичною частиною.
2. Модифікуйте свою програму з лабораторної роботи №10 таким чином, щоб дії, які виконують при виборі (в середині оператору case) виконувались окремою функцією, а в операторі case встановіть виклик цієї функції.**.**
3. Результати у вигляді текстового файлу надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)

Файл повинен мати назву в такому форматі:

**ОП+АМ <Номер групи><Номер лабораторної><Прізвище англійською>**

Наприклад, 21-01Ivanov.cpp.

**Тему в заголовку листа записати**

**ОП+АМ <Номер групи>-><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ОП+АМ-Запитання-<Номер групи>-<Прізвище >**.

.

**Строк відсилки ЛР для ІПЗ-21 02.11.2022**

**ІПЗ-22 09.11.2022**

*Контрольні запитання для самоперевірки*.

1. Що ми називаємо функцією?
2. Які змінні називаються локальними?
3. Як задаються в функціях значення, що повертаються?
4. Які параметри називаються формальними, а які фактичними?
5. Розкажіть про порядок дій, які виконуються під час виклику функції.
6. Чи можна у викликаємій функції змінювати параметри при **виклику функції з передачею значень? Чи вплине така зміна значень параметрів на дії в головній функції, якщо ці параметри потім там використовуються?**
7. **Які правила встановленні для формування списку параметрів, коли** передача даних виконується за замовчуванням?

**ТЕОРЕТИЧНА ЧАСТИНА**

**Повторення**

**Функція** — це послідовність операторів для виконання певного завдання. Часто ваші програми будуть переривати виконання одних функцій заради виконання інших. Ви це постійно робите в реальному житті, наприклад, ви читаєте книгу і згадали, що повинні були зробити телефонний дзвінок. Ви залишаєте закладку в своїй книзі, берете телефон і набираєте номер. Після того, як ви вже поговорили, ви повертаєтеся до тієї сторінки в книзі, на якій ви зупинилися. Програми в C++ працюють схожим чином. Іноді, коли програма виконує код, вона може зіткнутися з викликом функції. **Виклик функції** — це **вираз**, який вказує процесору перервати виконання поточної функції і приступити до виконання іншої функції. Процесор “залишає закладку” в поточній точці виконання, а потім виконує функцію, що викликається. Коли виконання функції, що викликається, — завершено, то процесор повертається до “закладки” і відновлює виконання перерваної функції.

Функція, в якій знаходиться виклик, називається **викликаючою функцією** (англ. “**caller**“).

**Наприклад:**

1. **#include <iostream> // для std::cout і std::endl**
2. **// Оголошення функції doPrint(), яку ми будемо викликати**
3. **void doPrint() {**
4. **std::cout << "In doPrint()" << std::endl;**
5. **}**
6. **// Оголошення функції main()**
7. **int main()**
8. **{**
9. **std::cout << "Starting main()" << std::endl;**
10. **doPrint(); /\* перериваємо виконання main() викликом функції doPrint(). main(), в даному випадку, є викликаючою функцією\*/**
11. **std::cout << "Ending main()" << std::endl;**
12. **return 0;**
13. **}**

Результат виконання програми вище:

Starting main()  
In doPrint()  
Ending main()

Запитання

1. Що означає запис **std::?**
2. З якого рядка починається виконання програми?
3. Що відбувається, коли процесор завершує виконання doPrint()?
4. Для виклику функції потрібно вказати її ім’я і список параметрів в круглих дужках (), чому в прикладі круглі дужки порожні

**Правило: Не забувайте вказувати круглі дужки () при виклику функцій.**

**Значення, що повертаються**

Коли функція main() завершує своє виконання, вона повертає цілочисельне значення назад в операційну систему, використовуючи **оператор return**. Функції, які ми пишемо, також можуть повертати значення. Для цього потрібно вказати **тип повернення значення**. Він вказується при оголошенні функції, перед її ім’ям. Тип повернення не вказує, яке саме значення буде повертатися. Він вказує тільки тип цього значення. Після цього всередині функції, що викликається, ми використовуємо оператор return, щоб вказати фактичне **значення, що повертається**.

**Приклад**

**#include <iostream>**

**/\* int означає, що функція повертає цілочисельне значення в викликаючу функцію \*/**

**int return7()**

**{**

**/\* Ця функція повертає цілочисельне значення, тому ми повинні використовувати оператор return \*/**

**return 7; // повертаємо число 7 в викликаючу функцію**

**}**

**int main()**

**{**

**std::cout << return7() << std::endl; // на екран виведеться 7**

**std::cout << return7() + 3 << std::endl;**

**// на екран виведеться 10**

**return7(); /\* значення 7, що повертається, - ігнорується, так як main() нічого з ним не робить\*/**

**return 0;**

**}**

**Повернення значення функцією main()**

Коли програма виконується, операційна система робить виклик функції main() і починається її виконання. Оператори в main() виконуються послідовно. В кінці функція main() повертає цілочисельне значення (зазвичай 0) назад в операційну систему. Тому main() оголошується як **int main().**

Чому потрібно повертати значення назад в операційну систему? Справа в тому, що значення, що повертається функцією main(), є **кодом стану**, який повідомляє операційній системі про те, чи успішно було виконання програми чи ні. Зазвичай, значення 0 (нуль) означає що все пройшло успішно, тоді як будь-яке інше значення означає невдачу/помилку.

За стандартами C++ функція main() повинна повертати цілочисельне значення. Однак, якщо ви не вкажете return в кінці функції main(), то компілятор поверне 0 автоматично, якщо ніяких помилок не буде. Проте все ж рекомендується вказувати return в кінці main() і використовувати тип повернення int для функції main().

**Значення, що повертаються**

1. Якщо тип повернення функції не void, то вона повинна повертати значення зазначеного типу (використовуючи оператор return). Єдиний виняток — це функція main(), яка повертає 0, якщо не надано інше значення.
2. Коли процесор зустрічає в функції оператор return, він негайно виконує повернення значення назад в викликаючу функцію і точка виконання також переходить в викликаючу функцію. Будь-який код, який знаходиться за return-ом у функції — ігнорується.
3. Функція може повертати тільки одне значення через return в викликаючу функцію. Це може бути або число (наприклад, 7), або значення змінної, або вираз (який генерує результат), або певне значення з набору можливих значень.
4. Автор функції вирішує, що означає значення, що повертає функція. Деякі функції використовують повернені значення в якості кодів стану для надання інформації щодо результату виконання функції (успішне виконання чи ні). Інші функції повертають певне значення з набору можливих значень, ще інші функції взагалі нічого не повертають.

**Повторне використання функцій**

Одну і ту ж функцію можна викликати декілька разів, навіть в різних програмах, що дуже корисно:

**#include <iostream>**

**/\* getValueFromUser отримує значення від користувача, а потім повертає його назад в викликаючу функцію\*/**

**int getValueFromUser()**

**{**

**std::cout << "Enter an integer: ";**

**int x;**

**std::cin >> x;**

**return x;**

**}**

**int main()**

**{**

**int a = getValueFromUser();**

**int b = getValueFromUser();**

**std::cout << a << " + " << b << " = " << a + b << std::endl;**

**return 0;**

**}**

**Результат виконання програми вище:**

**Enter an integer: 4  
Enter an integer: 9  
4 + 9 = 13**

Запитання

1. Скільки разів переривається main() для виклику функції?
2. Чи зберігається в змінній x значення від користувача після повернення в main()?
3. Чи є main()єдиною функцією, яка може викликати інші функції?
4. Яким буде результат виконання в наступному прикладі?

Приклад.

**#include <iostream>**

**void printO(){    std::cout << "O" << std::endl; }**

**void printK() {   std::cout << "K" << std::endl;}**

**// Функція printOK() викликає як printO(), так і printK()**

**void printOK(){   printO();   printK();}**

**int main() // Оголошення main()**

**{**

**std::cout << "Starting main()" << std::endl;**

**printOK();**

**std::cout << "Ending main()" << std::endl;**

**return 0;**

**}**

**Вкладені функції**

В С++ одні функції можуть бути оголошені всередині інших функцій (тобто бути вкладеними). Запитання. Чому наступний код спровокує помилку компіляції:

**#include <iostream>**

**int boo() // тепер уже не в main()**

**{   std::cout << "boo!";   return 0;}**

**int main()**

**{**

**boo();**

**return 0;**

**}**

Запитання.  **Чи є помилка в наступному коді?**

**#include <iostream>**

**int return5()**

**{**

**return 5;**

**int return8()**

**{**

**return 8;**

**}**

**}**

**int main()**

**{**

**std::cout << return5() + return8() << std::endl;**

**return 0;**

**}**

**#include <iostream>**

**int return6()**

**{**

**return 6;**

**}**

**int main()**

**{**

**std::cout << return6 << std::endl;**

**return 0;**

**}**

**Параметр функції** (або **“формальний параметр”**) — це змінна, яка створюється в оголошенні функції:

**void boo(int x); // оголошення (прототип функції). x - це параметр**

**void boo(int x) // визначення (також оголошення). x - це параметр**

**{**

**}**

**Аргумент** (або ***“фактичний параметр”***) — це значення, яке в функцію передає об’єкт, який викликає (caller) функцію:

boo(7); // 7 - це аргумент, який передається в параметр x

boo(y+1); // вираз y+1 - це аргумент, який передається в параметр x

Коли функція викликається, всі параметри функції створюються як змінні, а значення аргументів копіюються в параметри. При виклику функції boo() з аргументами 4 і 5, створюються параметри x і y функції boo() і їм присвоюються відповідні значення: 4 і 5. Результатом буде x = 4 і y = 5.

**Примітка:** У прикладі, наведеному вище, порядок обробки параметрів в функції boo() буде справа наліво, тобто спочатку створиться змінна y і їй присвоїться значення 5, а потім вже створиться змінна x і їй присвоїться значення 4. Порядок, в якому ініціалізуються параметри в круглих дужках функції, визначає кожен компілятор окремо, так як С++ явно не вказує цей порядок обробки. З параметрами-змінними це не настільки важливо і критично, але якщо ви будете використовувати в якості параметрів виклики інших функцій (що є поганою практикою і не рекомендується до використання), то результат може бути несподіваним.

Розглянемо наступну програму:

**#include <iostream>**

**int prinX()**

**{    std::cout << "x = 4\n";     return 0;}**

**int prinY()**

**{    std::cout << "y = 5\n";     return 0;}**

**void prinAll(int a, int b) {}**

**int main() {**

**prinAll(prinX(), prinY());  /\* в якості параметрів функції використовуються виклики функцій X() і Y()\*/**

**return 0;**

**}**

Результат виконання програми:

y = 5  
x = 4

Хоча параметри не оголошені всередині блоку функції, вони мають локальну область видимості. Це означає, що вони створюються при виконанні функції та знищуються, коли блок функції завершується:

**Передача по значенню**

За замовчуванням, аргументи в C++ передаються по значенню. Коли аргумент **передається по значенню**, то його значення копіюється в параметр функції. Наприклад:

**#include <iostream>**

**void boo(int y)**

**{     std::cout << "y = " << y << std::endl; }**

**int main()**

**{**

**boo(7); // 1-й виклик**

**int x = 8;**

**boo(x); // 2-й виклик**

**boo(x + 2); // 3-й виклик**

**return 0;**

**}**

У першому виклику функції boo() аргументом є **літерал** 7. При виклику boo() створюється змінна y, в яку копіюється значення 7. Потім, коли boo() завершує своє виконання, змінна y знищується.

У другому виклику функції boo() аргументом вже є змінна x = 8. Коли boo() викликається вдруге, змінна y створюється знову і значення 8 копіюється в y. Потім, коли boo() завершує своє виконання, змінна y знову знищується.

У третьому виклику функції boo() аргументом є вираз x + 2, який обчислюється в значення 10. Потім це значення передається в змінну y. При завершенні виконання функції boo() змінна y знову знищується.

Таким чином, результат виконання програми:

y = 7  
y = 8  
y = 10

Оскільки в функцію передається копія аргументу, то початкове значення не може бути змінено функцією. Це проілюстровано в наступному прикладі:

**#include <iostream>**

**void boo(int y)**

**{**

**std::cout << "y = " << y << '\n';**

**y = 8;**

**std::cout << "y = " << y << '\n';**

**} // y знищується тут**

**int main()**

**{**

**int x = 7;**

**std::cout << "x = " << x << '\n';**

**boo(x);**

**std::cout << "x = " << x << '\n';**

**return 0;**

**}**

Результат:

x = 7  
y = 7  
y = 8  
x = 7

На початку функції main() змінна x дорівнює 7. При виклику boo() значення x (7) передається в параметр y функції boo(). Усередині boo() змінній y спочатку присвоюється значення 8, а потім у знищується. Значення x не змінюється, навіть якщо змінити y.

Параметри функції, передані по значенню, також можуть бути [**const**](https://acode.com.ua/urok-40-const-constexpr-i-symvolni-konstanty/). Тоді вже буде 100% гарантія того, що функція не змінить значення параметру.

**Плюси і мінуси передачі по значенню**

**Плюси передачі по значенню:**

1. Аргументи, передані по значенню, можуть бути змінними (наприклад, x), літералами (наприклад, 8), виразами (наприклад, x + 2), **структурами**, класами або **перерахуваннями** (тобто майже будь-чим).
2. Аргументи ніколи не змінюються функцією, в яку передаються, що запобігає виникненню **побічних ефектів**.

**Мінуси передачі по значенню**:

Копіювання структур і класів може призвести до значного зниження продуктивності (особливо, коли функція викликається багато разів).

**Коли використовувати передачу по значенню**:

При передачі фундаментальних типів даних і енумераторів, коли припускається, що функція не повинна змінювати аргумент.

**Коли не використовувати передачу по значенню**:

При передачі **масивів,** структур і класів.

У більшості випадків, передача по значенню — це найкращий спосіб передачі аргументів фундаментальних типів даних, коли функція не повинна змінювати вихідні значення. Передача по значенню є гнучкою і безпечною, а в разі фундаментальних типів даних — ще і ефективною.

**Масиви як параметри функцій**

Аргументами (параметрами) функцій можуть бути не тільки змінні, але й масиви. Можна використовувати як масиви фіксованого розміру, так і невизначеного (масиви змінної довжини). При застосуванні масивів фіксованої довжини в заголовку функції в списку формальних аргументів указується тип масиву і його розмір, наприклад:

**void sort (int mas[30]);**

Якщо описується функція з масивом змінної довжини, то в заголовку вказується тип масиву невизначеного розміру і обов’язково ще один параметр, за допомогою якого задається розмірність масиву, наприклад:

**void sort (int mas[ ],int n);**

Всі масиви у функції *передаються за адресою (як покажчики)*, тому у випадку зміни масивів у функції ці зміни зберігаються при поверненні у викликаючу функцію.

***Приклад 1.*** У масиві хm1[10],m2[15],m3[12] визначити мінімальний елемент та його індекс.

**/\* визначення мінімальних значень  масиву, використання глобальної змінної\*/**

**#include <iostream>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**int ind=0; //--- глобальна змінна**

**// функція введення елементів масиву vvod()**

**vvod(float mas[ ],int n)**

**{ for (int i = 0; i <n ;i++)**

**{ cout<<"Enter " <<i <<" item ";**

**cin>>mas [i]; }**

**cout<<endl; }**

**// функція визначення мінімального елемента**

**float fmin(float mas[ ],int n)**

**{ ind = 0; float min =mas[0];**

**for (int i = 1; i < n; i++)**

**if (mas[i] <min) { min = mas[i]; ind = i;}**

**return min; }**

**main()//---------------головна функція**

**{ system("color F0");**

**float ml[5];**

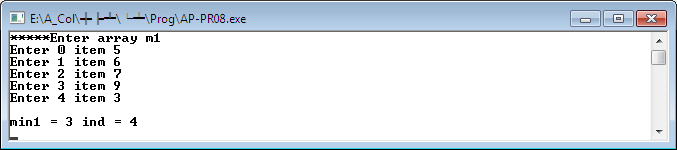
**cout<<"\*\*\*\*\*Enter array m1\n";**

**// виклик функції vvod()— введення масиву m1[]**

**vvod(ml,5);**

**cout<<"min1 = "<<fmin (ml,5); // виклик функції fmin()**

**cout<<" ind = "<<ind<<endl;**

**getch(); }**

**Результати обчислень:**

Програма, крім **головної функції main()**, має також функцію **vvod()** введення елементів деякого формального масиву та функцію **fmin()**визначення мінімального елемента цього масиву. У головній функції здійснюється виклик функцій для розв’язання необхідних обчислень кожного конкретного масиву, для передачі параметрів використовується глобальна змінна.

**Параметри функцій - багатовимірні масиви**.

У цьому випадку використовуються масиви як фіксованої розмірності, так і невизначеної довжини. У заголовку функції під час роботи з багатовимірним масивом фіксованого розміру, наприклад матриці mat(7,10),вказуються розмірності масиву:

**void fun1(intmat[7][10]);**

Якщо застосовується багатовимірний масив невизначеної довжини, то невизначеним може бути тільки один вимір розмірності, наприклад:

**void fun2 (int mat[][10], int rows, int col);**

***Приклад 2.*** Для заданої матриці зробити обчислення середнього значення кожного її стовпця з використанням функції введення розмірності матриці, функції введення матриці і функції одержання середнього значення її стовпців.

**Матрицею** розміру n×m називається прямокутна таблиця спеціального вигляду, що складається із n рядків та m стовпців, заповнених числами.

**// Обчислення середнього значення стовбців матриці**

**#include <iostream>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**const int mincol = 1;**

**const int maxcol = 4;**

**const int minrow = 2;**

**const int maxrow = 4;**

// **функція getnum()для введення кількості рядків та стовбців столбців**

**int getnum (const char \*elemtype, int low, int high)**

**{int n;**

**do**

**{cout<<"Enter quantity " <<elemtype<< "n from["<<low<<"] to[" <<high<< "] : ";**

**cin >>n; }**

**while (n <low ||n > high); // OR**

**return n; }**

**//--функція inmatr() введення елементів матриці**

**void inmatr (float matr[][maxcol], int rows, int cols)**

**{ for (int i = 0; i < rows; i++)**

**{ cout <<"Enter " <<i << " matrix row " <<endl;**

**for (int j= 0;j<cols; j++)**

**cin >>matr [i][j];}**

**cout <<endl; }**

**//функція srcols() визначення середніх значень стовбців**

**void srcols (float matr [ ] [maxcol], int rows, int cols)**

**{ float sum, sr;**

**for(int j= 0;j<cols;j++)**

**{ sum= 0.0;**

**for (int i= 0;i<rows; i++) sum+=matr [i][j];**

**sr = sum/rows;**

**cout.precision(3);**

**cout <<"Column Average "<<j<<" = "<<sr<< endl; }**

**}**

**int main( ) //-------------------** **головна функція**

**{ system("color F0");**

**float matr [maxrow][maxcol]; int rows, cols;**

**//**

**rows= getnum ("Row", minrow, maxrow);**

**cols = getnum ("Column", mincol, maxcol);**

**inmatr (matr, rows, cols); //--** **введення матриці**

**cout<<endl;**

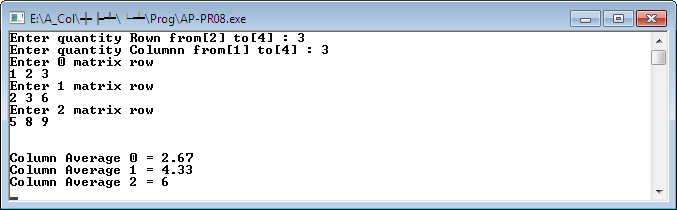
**// обчислення середнього значення стовбців матриці srcols(matr,rows,cols);**

**getch();**

**return 0;**

**}**

Результати обчислень:



Програма, крім **головної функції main()**, має функцію getnum (const char \*elemtype, int low, int high) для введення значень кількості рядків або стовбців. Параметр const char \*elemtype описує рядкову змінну з назвою, яка задається в **main()** літералом "Row" або "Column". Цей літерал передається через покажчик. Покажчики ми не вивчали, тому візьміть до відома цю конструкцію. Окрім того, в програмі є функції inmatr() введення матриці та srcols() визначення середніх значень стовбців .

У головній функції здійснюється виклик функцій для розв’язання необхідних обчислень кожного конкретного масиву, для передачі параметрів використовується глобальна змінна.