# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**Кафедра програмного забезпечення систем**

**ЗВІТ**

**Про виконання лабораторної роботи №6**

**«Особливості роботи з функціями в С++»**

**з дисципліни «Основи програмування»**

Виконав:

слухач групи ПЗС-11

Гринчук Тарас

Прийняв:

доц. Макар В.М.

« »\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2013 р.

∑ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛЬВІВ – 2013

**Тема роботи**: Особливості роботи з функціями в С++.

**Мета роботи:** Поглиблене вивчення можливостей функцій у мові С++.

## 1. ТЕОРЕТИЧНІ ВІДОМОСТІ

1. У ролі аргументів функцій в С++ можуть використовуватися масиви. У цьому випадку достатньо вказати лише ім’я масиву без необхідності задавати квадратні дужки і розмірність масиву. Так, наприклад, якщо одновимірний масив array оголошений як *int array[25];* то передати його у функцію сортування можна так: *sort(array);*

Приклад програми з передачею масивів у функції. Результат на рис.1.1:

#include <iostream>

#include <conio.h>

using namespace std;

void sort(int[], int);

void display(int\*, int);

void main() {

const int n=10;

int array[n] = {3,5,8,10,1,4,15,2,4,7};

cout<<"Array before sort: ";

display(array, n);

cout<<"Array after sort: ";

sort(array, n);

display(array, n);

\_getch();

}

void display(int \*mas, int size) {

for(int i=0; i<size; i++) cout<<\*(mas+i)<<" ";

cout<<endl;

}

void sort(int mas[], int size) {

int k=1, temp;

for(int i; (i<size)&&(k); i++) {

k=0;

for(int j=0; j<size-1; j++) {

if(mas[j]>mas[j+1]) {

temp=mas[j];

mas[j]=mas[j+1];

mas[j+1]=temp;

k=1;

}

}

}

}

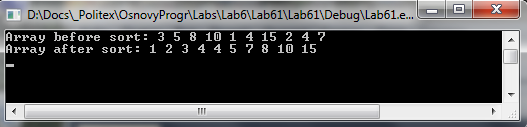


Рис 1.1

1. Для передачі двовимірних масивів у функції необхідно явно задавати розмірність другого виміру масиву. Це пов’язано з тим, що компілятор використовує це значення розмірності для правильного визначення комірок пам’яті елементів багатовимірного масиву, тобто компілятору потрібна інформація про те скільки елементів знаходиться в одному рядку двовимірного масиву для того щоб правильно розбити матрицю на окремі рядки. Наступний приклад програми демонструє можливості передачі двовимірних масивів у функції з їх подальшою обробкою (результат на рис 1.2):

#include <iostream>

#include <conio.h>

using namespace std;

const int n=3, m=4;

void sort(int [], int);

void display(int [][m], int);

void main()

{

int matr[n][m]={{3,5,8,10},{1,4,15,2}, {4,7,2,9}};

cout<<"Matrix before sort: \n";

display(matr,n);

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

sort(matr[i],m);

cout<<"Matrix after sort: \n";

display(matr,n);

\_getch();

}

void display(int mas[][m], int size)

{

for(int i=0; i<size; i++) {

for (int k=0; k<m; k++)

cout<<mas[i][k]<<" ";

cout<<endl;

}

cout<<endl;

}

void sort(int mas[], int size)

{

int k=1, temp;

for(int i=0; (i<size)&&(k); i++) {

k=0;

for(int j=0; j<size-1; j++)

if(mas[j]>mas[j+1]) {

temp=mas[j];

mas[j]=mas[j+1];

mas[j+1]=temp;

k=1;

}

}

}

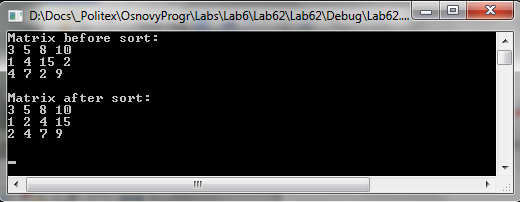


Рис 1.2

1. Інколи виникають ситуації, коли потрібно заборонити модифікувати елементи масиву всередині функції. Оскільки масив передається за адресою, то контролювати самостійно всі зміни масиву в тілі функції може бути не простою задачею. Краще тут викориcтати спеціальний специфікатор типу мови С++, а саме специфікатор const. Коли параметр типу масив описується за допомогою const, то елементи такого масиву в тілі функції стають константами і будь-яка спроба їх модифікувати буде приводити до помилки.
2. У C++ крім функцій з фіксованим числом параметрів є можливість також використо вувати функції зі змінною кількістю параметрів, під якими розуміють функції, в які можна передавати дані, не описуючи їх усіх у прототипі й, відповідно, у заголовку опису функції.
3. Список формальних параметрів для функцій із змінним числом параметрів складається з постійних параметрів та змінної частини, яка задається за допомогою конструкції еліпсису, що представляє собою три крапки … , причому спочатку перераховуються постійні параметри. У загальному випадку формат оголошення функції зі змінною кількістю параметрів має такий вигляд:

*[тип\_результату] ім’я\_функції(параметр1, параметр2, ...);*

**Приклад** (результат на рис. 1.3):

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Функція обчислення середнього арифметичного довільної кількості дійсних чисел \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <conio.h>

#include <stdarg.h>

double f(double , ...); // прототип функції зі змінним чисом параметрів

int main()

{

using namespace std;

cout<<"Seredne aryfmetychne 2,4,3 = "<<f(2.0,4.0,3.0,0.0)<<endl;

cout<<"Seredne aryfmetychne 2,14,8,16 = "<<f(2.0,14.0,8.0,16.0,0.0)<<endl;

\_getch();

return 0;

}

double f(double n, ...)

{

double \*p = &n;// вказівник на початок списку параметрів

double sum = 0;

int count = 0;

while (\*p)// поки в списку є ненульовий елемент

{

sum+=(\*p);

p++;// переходимо на наступний параметр зі списку

count++;

}

return ((sum)?sum/count:0);

}

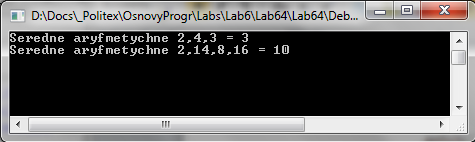


Рис 1.3

1. Існує два способи задання довжини змінного списку параметрів: в одному з постійних аргументів вказувати їх кількість або ж додати у кінець списку аргумент з унікальним значенням, що служитиме ознакою кінця списку. Наприклад: *func1(5,xl,x2,x3,x4,x5);* - тут зазначене число аргументів - 5; *func2( xl,x2,x3,x4,0);* - тут зазначена ознака кінця списку - 0.

**Приклад** (результат на рис. 1.4):

#include <iostream>

#include <conio.h>

#include <stdarg.h>

double f(int, ...);

using namespace std;

int main() {

cout<<f(3,2.0,4.0,3.0)<<endl;

cout<<f(4,2.0,14.0,8.0,16.0)<<endl;

cout<<f(10,1.5,1.5,2.8,3.4,2.0,14.0,8.0,16.0,4.5,2.3)<<endl;

\_getch();

return 0;

}

double f(int n, ...) {

double sum=0, term;

int count = 0;

va\_list p;

va\_start(p,n);

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

term = va\_arg(p,double);

cout<<"term #"<<(i+1)<<" = "<<term<<"\t";

sum+=term;

count++;

}

va\_end(p);

cout<<endl<<"Average = ";

return ((sum)?sum/count:0);

}

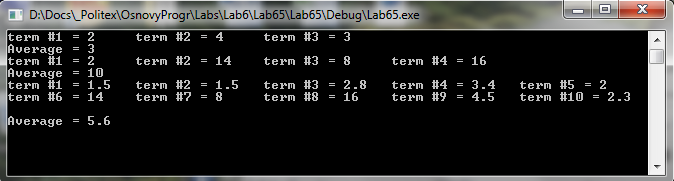


Рис 1.4

1. Передача аргументів у функції в С++ відбувається через спеціальну ділянку оперативної пам’яті, яка називається стек. У С/С++ фактичні параметри записуються в стек з кінця списку аргументів, причому ці дані поміщаються в стек відповідно до типу, який використовується при виклику функції. Оскільки принцип роботи зі стеком полягає в тому, що значення, яке було занесено до стеку останнім, буде використовуватися (витягуватися зі стеку) першим.
2. В С++ не можна створювати функції, щоб список параметрів цих функцій містив лише значення параметрів змінної дожини, тому що компілятор не буде мати інформації, як виділити пам’ять для стеку функції, не маючи ні типу параметрів, ні початкової точки входу у стек.
3. У стандарт мови С++ входять макроси для роботи зі списками параметрів змінної довжини. Ці макроси визначені у файлі stdarg.h. При їх використанні також доводиться вказувати у списку постійний параметр, оголосити та встановити на нього вказівник та переміщувати цей вказівник по списку. В кінці списку має бути NULL. Макроси мають наступний формат:

*void va\_start(va\_list prm, останній\_фіксований\_параметр);*

*тип va\_arg(va\_list prm, тип);*

*void va\_end(va\_list prm);*

1. Макрос *va\_arg* переміщає вказівник на наступний параметр. Синтаксис наступний: *va\_arg*(вказівник\_типу\_va\_list,тип\_чергового\_параметра).

У момент написання програми програміст повинен знати тип кожного з параметрів.

**Приклад** (результат на рис. 1.5):

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функція для обчислення добутку довільної кількості дійсних чисел \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <conio.h>

#include <stdarg.h>

double multiply(char\*, ...);

using namespace std;

int main(int argc, char \* argv[])

{

multiply("4.0\*6.0\*2.0 = ",4.0,6.0,2.0,-1.0);

multiply("2.0\*3.0\*5.0\*0.5 = ",2.0,3.0,5.0,0.5,-1.0);

multiply("This product is ",-1.0);

\_getch();

return 0;

}

double multiply(char \*str, ...)

{

double product = 1, term;

va\_list arg\_list;// оголошуємо вказівник типу va\_list

va\_start(arg\_list,str); // встановлюємо вказівник на початок змінної

// частини списку параметрів

int k = 0;

while((term = va\_arg(arg\_list, double))>0)

{

product \*=term;

k = 1;

}

if(!k) product = 0;

cout<<str<<product<<endl;

va\_end(arg\_list); // завершуємо оборобку списку параметрів

return product;

// прототип функції зі змінним чисом параметрів

}

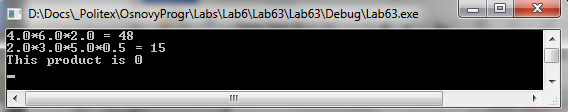


Рис 1.5

1. Список формальних параметрів функції main() є фіксованим, однак за його допомогою можна задавати список аргументів, кількість яких обмежується лише можливостями операційної системи. Аргументи функції main() повинні вказуватися у командному рядку одразу після імені виконуваного модуля (файла з розширенням .exe). С++ підтримує три параметри функції maіn(). Для їхнього позначення рекомендується використовувати загальноприйняті імена argc, argv, envp. Заголовок функції maіn() при використанні командного рядка має вигляд:

*іnt maіn(іnt argc, char \*argv[], char \*envp[])*

**Приклад** (результат на рис. 1.6):

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Застосування аргументів функції main\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <conio.h>

int solvelin(double,double);

using namespace std;

void main(int argc, char\* argv[])

{

cout<<"There are "<<argc<<" arguments"<<endl;

if(!(argc==4))

{

cout << "Error - not enough or too many arguments"<<endl;

\_getch();return;

}

solvelin(atof(argv[2]), atof(argv[3]));

\_getch();

}

int solvelin(double var1, double var2)

{

cout<<"The equation to be solved is:"<<"\t"<<(var1)<<"\*x = "<<(-var2)<<endl;

if(var1) cout<<"Result: "<<(-var2/var1);

else cout<<"There are no solutions";

return 0;

}

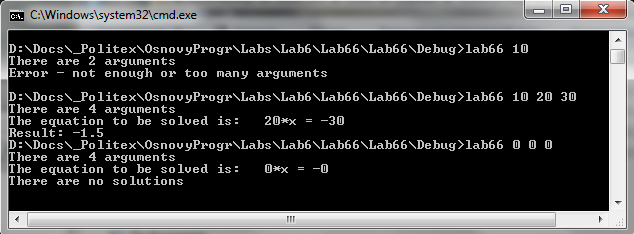


Рис 1.6

1. Перший параметр *argc* є цілим числом, що задає кількість аргументів, вказаних у командному рядку. Ім’я програми вважається першим аргументом. Тому у випадку, якщо користувач не задав жодного аргументу, значення argc становитиме 1. Параметр *argv* є вказівником на масив символьних вказівників, які посилаються на аргументи командного рядка. Слід мати на увазі, що всі аргументи командного рядка є рядками, а тому при потребі їх слід перетворювати у числа за допомогою відповідних функцій. Останнім елементом масиву вказівників є нульовий вказівник *(NULL).* Параметр *env* є масивом вказівників на рядки оточення операційної системи. Його реалізація залежить від конкретного середовища. Приклад:

## ХІД РОБОТИ

**Індивідуальне завдання №1 (варіант 5).** Написати функцію для додавання двох матриць. З її допомогою додати вихідну матрицю і транспоновану.

Для реалізації задачі, перш за все ввести дві квадратні матриці (розміром *N* рядків та *N* стовбців) з клавіатури, позначимо їх *A* та *B*. Результатом задачі повинна бути матриця:

Ввід доцільно зробити за допомогою функції. Прототип цієї функції має вигляд:

void input(int [][N], int);

Першим параметром є матриця, яка вводиться, другим – її розмір. Також нам потрібно функцію, яка додає дві матриці і результат зберігає у першу матрицю:

void add(int [][N], int [][N], int);

За допомогою даної функції отримаємо:

*A = A + B.*

Наступним кроком нам потрібно транспонувати матрицю А і результат помістити у матрицю B. Транспонування оформимо у вигляді функції:

void trans(int [][N], int [][N], int);

Перший параметр – вхідна матриця, другий – транспонована, третій – розмір.

Отримаємо:

Тепер залишилось додати матриці *A* та *B*:

*A = A + B.*

Для додавання, знову скористаємось функцією add, яка була описана раніше.

Виведемо результуючу матрицю А на екран. Вивід реалізуємо функцією:

void display(int [][N], int);

Першим параметром даної функції буде матриця, а другим – її розмір.

**Блок-схема для даного алгоритму показана на рисунках 2.1-2.5.**

1

N=2

input(a,N)

input(b,N)

add(a,b,N)

trans(a,b,N)

**Рис. 2.1 Блок-схема алгоритму функція main()**

trans(a,b,N)

add(a,b,N)

display(a,N)

1

**Рис. 2.1 Блок-схема алгоритму функція main()**

-

+

i<N

i=0

j<N

-

+

i++

j=0

Ввести a[і][j]

j++

**Рис. 2.2. Блок-схема для функції *input(a,N)***

-

+

i<N

i=0

j<N

-

+

i++

j=0

j++

a[i][j]+=b[i][j]

**Рис. 2.3** **Блок-схема для функції *add(a,N)***

-

+

i<N

i=0

j=0

2

1

3

**Рис. 2.4 Блок-схема для функції *trans(a,b,N)***

j<N

-

+

i++

j++

b[i][j]=a[j][i]

2

1

3

**Рис. 2.4 Блок-схема для функції *trans(a,b,N)***

-

+

i<N

i=0

j<N

-

+

i++

j=0

Вивести a[і][j]

j++

**Рис. 2.5 Блок-схема для функції *display(a,N)***

Програмна реалізація даного алгоритму має вигляд:

#include <iostream>

#include <conio.h>

using namespace std;

const int N = 2;

void input(int [][N], int);

void display(int [][N], int);

void trans(int [][N], int [][N], int);

void add(int [][N], int [][N], int);

void main() {

int a[N][N],b[N][N];

//введення а та b

cout<<"\nInput matrix a:\n";

input(a, N);

cout<<"\nInput matrix b:\n";

input(b, N);

//виведемо матрицю а у вигляді таблиці на екран

cout<<"\nMatrix a:\n";

display(a, N);

//виведемо матрицю b у вигляді таблиці на екран

cout<<"\nMatrix b:\n";

display(b, N);

//a = a + b

add(a, b, N);

cout<<"\nMatrix (a+b):\n";

display(a, N);

//b = trans(a+b)

trans(a, b, N);

cout<<"\nMatrix trans(a+b):\n";

display(b, N);

//a = (a+b) + trans(a+b)

add(a, b, N);

cout<<"\nMatrix (a+b) + trans(a+b):\n";

display(a, N);

\_getch();

}

//ввелдення матриці з клавіатури

void input(int a[][N], int size) {

for(int i = 0; i < N; i++)

for(int j = 0; j < N; j++) {

cout<<"Element "<<i+1<<", "<<j+1<<": "; //нумерацію почнемо з 1,1;

cin>>a[i][j];

}

}

//вивід матриці на екран

void display(int arr[][N], int size) {

for(int i = 0; i < size\*size; i++)

cout<<\*(\*arr+i)<<((i+1)%size ? "\t" : "\n"); //якщо стрічка не нова: табуляція, інакше перехід на новий рядок

cout<<endl;

}

//Додавання двох матриць a= a+b

void add(int a[][N], int b[][N], int size) {

for(int i = 0; i < size\*size; i++)

\*(\*a+i) += \*(\*b+i);

}

//транспонування матриці a. b - результуюча матриця

void trans(int a[][N], int b[][N], int size) {

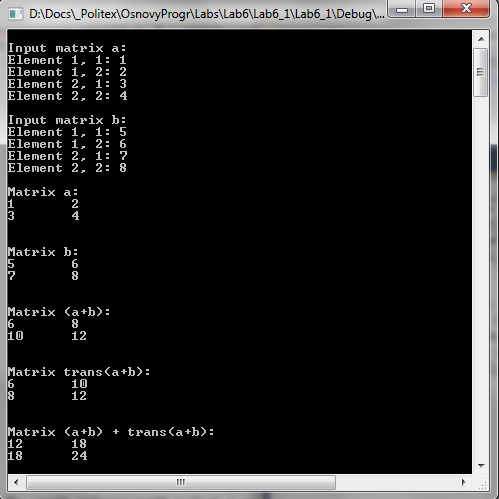
for(int i = 0; i < size; i++)

for(int j = 0; j < size; j++)

b[i][j]=a[j][i];

}

Запустимо програму на виконання. Результати на рис. 2.6.

****

(Рис. 2.6)

**Індивідуальне завдання №2 (варіант 5).** У функцію зі змінним числом параметрів надходять додатні цілі числа, кінець списку – значення -1. Порахувати, скільки разів зустрічається кожна цифра у заданому числі.

Опишемо вказану функцію, як:

void calculate(int n, ...)

В тілі цієї функції, будемо перебирати список аргументів, поставивши у вказівник адресу першого параметра функції, поки не отримаємо значення -1. Введемо допоміжний масив розміром 10 елементів, в який будемо записувати кількість кожної цифри числа.

**Блок-схема алгоритму функції зображена на рис. 2.7.**

-

+

\*p>0

x>0

-

+

x=(x-j)/10

x=\*p

digs[j]++

j=x%10

n

p=&n

digs={0,0,0,0,0,0,0,0,0,0}

Вивести \*p

i=0

2

1

3

Рис. 2.7

i<10

-

+

i++

digs[i]=0

2

1

3

Вивести digs[i]

p++

Рис. 2.7

Програмна реалізація даного алгоритму має вигляд (результат на рис. 2.8):

#include <iostream>

#include <conio.h>

using namespace std;

void calculate(int , ...); // прототип функції зі змінним чисом параметрів

void main()

{

calculate(100451,55555,320121,444222,-1);

\_getch();

}

void calculate(int n, ...) {

int \*p = &n;// вказівник на початок списку параметрів

int digs[10] = {0,0,0,0,0,0,0,0,0,0}; //масив цифр

while(\*p>0) {// поки в списку є додатній елемент

cout<<"\nNumber "<<\*p<<":\n";

int j, x=\*p;

while(x>0) {

j=x%10; //остання цифра числа х

digs[j]++;

x=(x-j)/10; //"відкидаємо" останню цифру від числа

}

for(int i=0; i<10; i++) {

cout<<"["<<i<<"]: "<<digs[i]<<";\t";

digs[i]=0; //обнулимо масив цифр

}

p++;// переходимо на наступний параметр зі списку

}

}

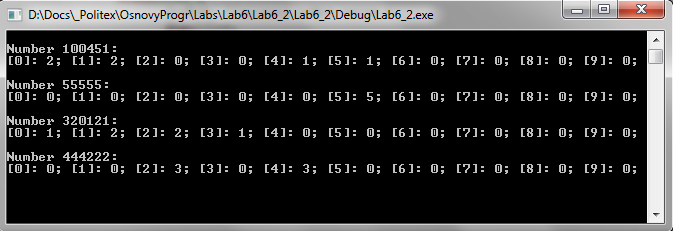
****

Рис. 2.8

## ВИСНОВКИ

На даній лабораторній роботі я навчився використовувати функції у мові С++. Зокрема вивчив особливості передачі масивів у функції, написанню функцій з невизначеною кількістю параметрів, отримувати доступ до списку параметрів за допомогою вказівників та макросів.