# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**Кафедра програмного забезпечення систем**

**ЗВІТ**

**Про виконання лабораторної роботи №5**

**«Основи 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**: Основи роботи з функціями в С++.

**Мета роботи:** здобути практичні навики створення та застосування функцій у мові С++.

## 1. ТЕОРЕТИЧНІ ВІДОМОСТІ

1. У процесі програмування реальних практичних задач дуже часто виникає ситуація коли на різних етапах розв’язання вихідної задачі доводиться неодноразово розв’язувати деяку часткову підзадачу. Причому алгоритм розв’язання цієї підзадачі передбачає виконання тої ж самої послідовності дій, але над різними наборами даних. Для підвищення ефективності програми та її простоти і наглядності, групу операторів, які реалізують таку послідовність дій, оформляють у вигляді самостійної програмної одиниці (підпрограми). У мові С++ такі підпрограми реалізуються у вигляді функцій.
2. Синтаксис мови С++ передбачає такий формат опису функції:

*[тип\_результату] ім’я\_функції(список\_параметрів)*

*{*

*тіло функції*

*}*

1. Поле *список\_параметрів* представляє собою список так званих параметрів для позначення вхідних даних функції. Тому ці параметри часто називають формальними параметрами. Якщо функція потребує декілька формальних параметрів, то вони задаються у списку послідовно, розділяючись комою, але кожний параметр у цьому списку описується за наведеним вище правилом (тобто для кожного параметру треба вказати свій тип, навіть якщо всі вони мають одинаковий тип). У випадку, коли функція не має параметрів, список формальних параметрів або задається ключовим словом void, або є порожнім (але круглі дужки у цьому випадку залишаються).
2. Існують два способи завершення виконання функції і повернення у точку програми, з якої здійснено виклик. Перший - завершення виконання функції з використанням оператора повернення return. Цей спосіб найчасітше застосовують тоді, коли функція має тип результату відмінний від void. Інший спосіб завершення виконання функції полягає у послідовному проходженні всіх операторів тіла функції до закриваючої фігурної дужки. Цей спосіб можливий лише для функцій з типом результату void, хоча як показує наступний приклад він не є єдино можливим способом виходу з функцій типу void.
3. Виклик функції здійснюється в потрібному місці програми за її іменем (аналогічно як і для стандартних бібліотечних функцій) у такому загальному форматі:

*ім’я\_функції([список\_аргументів])*

1. Поле список\_аргументів задає значення фактичних параметрів, які підставлються у відповідні формальні параметри. На відміну від списку формальних параметрів у списку аргументів не потрібно вказувати тип аргумента, достатньо записати лише ідентифікатор. В загальному випадку аргумент може задаватися виразом відповідного типу, тобто типу який співпадає з типом відповідного формального параметра в описі функції. Під час виклику функції у формальні параметри підставляються значення аргументів у тому порядку, в якому вони задані при виклику функції. Кількість формальних параметрів функції та аргументів повинні співпадати (за винятком випадку коли формальні параметри мають значення за замовчуванням). Якщо у функції відсутні формальні параметри, то при виклику такої функції повинен також бути відсутнім і список аргументів (але пара дужок () при виклику залишається).
2. Попереднє оголошення функції прийнято називати прототипом функції. Для визначення прототипу функції потрібно задати тип результату функції, її ім’я та список типів формальних параметрів у такому форматі:

*[тип\_результату] ім’я\_функції(список\_типів\_параметрів);*

Зауважимо, що прототип функції співпадає із заголовком в описі функції за виключенням того, що в прототипі відсутні імена формальних параметрів.

1. Суть приведення типів, полягає в наступному: якщо типи формального параметру і аргумента функції не співпадають (але вони відносяться до одного з вбудованих типів мови С++), то відбувається автоматичне приведення значення “нижчого” типу до значення “вищого” типу згідно ієрархії основних типів, яка наведена у табл.1 (в порядку від “вищого” типу до “нижчого”). Якщо аргумент (фактичний параметр) функції має “нижчий” тип ніж тип відповідного формального параметра, то таке перетворення типу відбувається без втрати значення аргумента. У протилежному випадку, високою є імовірність втрати значення аргумента, наприклад, якщо відбувається перетворення числа типу *float* у число типу *int*.

**Приклад.** В наступній програмі, в обох випадках отримаємо значення 16, незважаючи на те, що в другому виклику функції square аргумент має значення типу double (рис 1.1)

#include<iostream>

#include<conio.h>

int square(int); // прототип функції

using namespace std;

void main() {

cout<<square(4)<<endl;// хочемо отримати квадрат від 4

// хочемо отримати квадрат від 4.7

cout<<square(4.7)<<endl;

\_getch();

}

int square (int x) {

return x\*x;

}

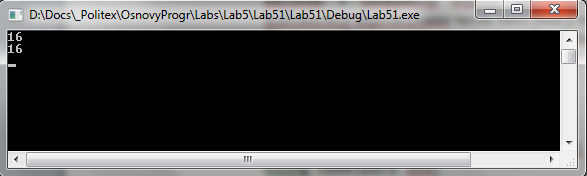


Рис 1.1

1. Мова С++ має два способи передачі аргументів: передача аргументів за значенням і передача аргументів за посиланням(адресою). Виклик функції з передачею значення передбачає, що значення аргументу копіюється у відповідний формальний параметр функції. Тому зміни значення цього параметра всередині функції не впливають на значення змінних, які використовувалися для виклику. При передачі параиетрів за адресою, передається не копія значення аргумента, а його адреса. Це означає, що всередині функції, яка викликається, ми маємо доступ до оригінала значення аргумента (через його адресу). Тому будь-яка модифікація відповідного формального параметра всередині функції означає й одночасну зміну аргумента.

**Приклад.** Виклик функції з передачею значення. Результат виконання на рис 1.2.

#include<iostream>

#include<conio.h>

int square(int); // прототип функції

using namespace std;

void main()

{

int p=5;

cout<<"p="<<p<<" p\*p="<<square(p)<<" p="<<p<<endl;

\_getch();

}

int square (int x)// опис функції

{x\*=x; return x; }

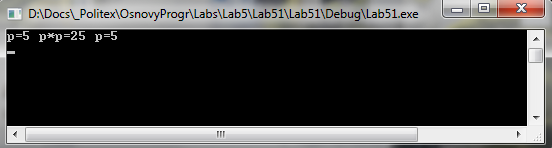


Рис 1.2

У цьому прикладі значення аргументу для функції square() (тобто 5) скопіювалося у параметр x. Коли відбулося присвоєння x\*=x, змінилося значення лише формального параметру x. Змінна p, яка використовувалася у ролі аргумента при виклику square(), і надалі має значення 5, тому на екран виведеться такий рядок: p=5 p\*p=25 p=5.

**Приклад.** Виклик функції з передачею адреси. Результат виконання на рис 1.3.

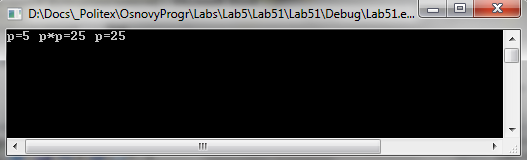


Рис 1.3

Тут, при виклику функції square() було передано адресу аргумента (тобто змінної p). Ця адреса тепер використовується для доступу до комірки пам’яті, в якій зберігається значення змінної p. Тому тепер присвоєння x\*=x змінює не лише значення формального параметру x, а й значення змінної p, яка використовується у ролі аргументу при виклику функції. Як наслідок, на екран виведеться такий рядок: p=5 p\*p=25 p=25.

1. Оскільки при передачі аргументів у функцію через адресу, здійснюється зміна фактичного параметра, в переданій адресі, у ролі такого параметра не можна задавати вираз, тому що вирази не мають статичної адреси в пам’яті, на відміну від змінних.
2. Усі змінні, оголошені всередині довільної функції є локальними змінними цієї функції. Це означає, що їх значення зберігаються тільки на час виконання цієї функції. Усі формальні параметри функції розглядаються також як локальні змінні цієї функції. Усі змінні, оголошені поза межами всіх функцій з яких складається програма, називаються глобальними. Доступ до таких змінних можливий у будь-якій функції. Іншими словами, область видимості глобальних змінних обмежена лише файлом у якому знаходиться вихідний код програми.

**Приклад.** Локальні та глобальні змінні. Результат виконання на рис 1.4.

#include <iostream>

#include <conio.h>

void a (void);

int x=1;

// прототип функції

// глобальна змінна

using namespace std;

void main()

{

int x = 5;// локальна змінна функції main

cout << "x=" << x <<endl;

{

int x=7; // локальна змінна внутрішнього блоку main

cout << "x=" << x <<endl;

}

cout << "x=" << x << "\n";

a();

\_getch();

}

void a(void)// опис функції a

{

cout << "x=" << x++ << "\n";

}

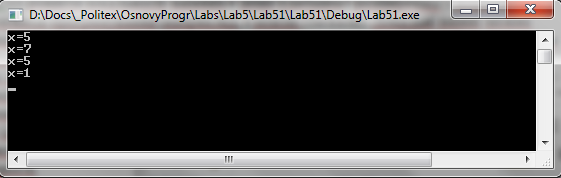


Рис 1.4

1. Локальні змінні можна оголошувати у довільному місці функції в межах так званого програмного блоку під яким розуміють довільну послідовність оголошень та операторів, яка знаходиться між парою фігурних дужок {}. Тіло довільної функції в С++ також розглядається як програмний блок. У цьому випадку змінна є локальною по відношенню до блоку в якому вона оголошена і її область видимості обмежена цим блоком. Більше того, в С++ дозволяється використовувати однакові ідентифікатори для позначення змінних в різних блоках. Це можливо завдяки вбудованому механізму перекриття імен мови С++.

## ХІД РОБОТИ

**Індивідуальне завдання (варіант 5).** Задано функцію. Написати програму для знаходження визначеного інтегралу з точністю для заданих границь [a,b] за допомогою трьох методів:

1. прямокутників;
2. трапецій;
3. Сімпсона.

В основу чисельного інтегрування покладене наближене обчислення площини під кривою, яка описується підінтегральною функцією інтеграла виду:

Визначений інтеграл *І* являє собою площину, обмежену кривою *f(x)*, віссю х та прямими *х = a; х = b*. Чисельне інтегрування основане на тому, що відрізок інтегрування *[a,b]* розбивають на множину менших відрізків, кожен з яких є основою геометричної фігури, знаходять наближено площу кожної такої фігури, та значення інтегралу *І* визначають як суму площин , тобто:

При цьому використовують два способи розбиття відрізка інтегрування на менші :

* Розбиття відрізка інтегрування проводиться раніше аналізу результатів інтегрування, до того ж завжди відрізки вибирають рівними (метод прямокутників, трапецій, Сімпсона);
* Місцезнаходження та довжина відрізків визначаються з умови досягти найбільшої точності чисельного інтегрування з заданим числом відрізків, а потім відповідно з цим визначають їхні межі (метод Гауса, Ньютона – Котеса, Чебишева).

**Оцінка похибки по правилу Рунге.** При програмуванні обчислення визначеного інтегралу завершують при досягненні заданої точності. Для оцінки точності використовують метод подвійного перерахунку, який заключається в наступному:

1. Обчислюється інтеграл з розбиванням на n інтервалів.
2. Збільшують кількість інтервалів в два рази і отримують нове наближення.

Щоб визначити як нове розраховане значення відрізняється від істинного значення застосовують правило Рунге:

Для методів прямокутників і трапеції: ;

Для метода Сімпсона: .

**Метод прямокутників.** Геометрична інтерпретація цього метода зводиться до знаходження визначеного інтегралу як суми площ *N* прямокутників (з висотою *f(x)* та основою ),  отриманих шляхом розбиття відрізка *[а, b]*на *N* рівних частин. В цьому випадку розділити на прямокутники можна або зліва на право (рис. 2.1), тоді отримаємо формулу лівих прямокутників: або розділити на *N* прямокутників справа наліво (рис. 2.2), тоді отримаємо формулу правих прямокутників:

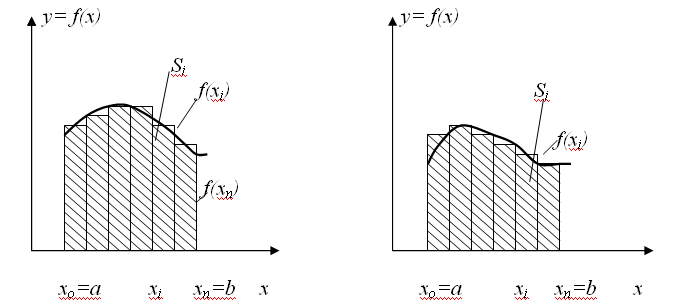


Рис. 2.1 Рис. 2.2

**Блок-схема для алгоритму прямокутників показана на рис 2.3.**

b=7

3

a=0

n=1

E=0.001

n\*=2

s=0

h=(b-a)/n

s0=s

i=0

x=a

i<n

-

+

2

4

1

Рис. 2.3

x+=h

3

s+=pow(x+1,-2/3)

i++

s\*=h

fabs(s-s0)>E

-

+

2

4

1

Вивести s

Рис. 2.3

**Метод трапецій.** Суть методу трапецій полягає в тому, що інтеграл обчислюється по-іншому, відрізок інтегрування поділяється на *N* рівних відрізків, всередині яких підінтегральна крива *f(x)* замінюється кусково-лінійною функцією *,* отриманою стягуванням ординат *N* відрізків хордами. Інтеграл знаходиться як сума площ прямокутних трапецій (pис. 2.4).

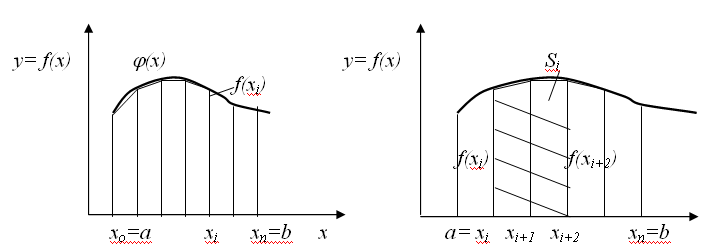


Рис. 2.4

Площа кожної такої трапеції визначається як:

**Блок-схема для алгоритму трапецій показана на рис 2.5.**

b=7

3

a=0

n=1

E=0.001

n\*=2

s=0

h=(b-a)/n

s0=s

fb= pow(b+1,-2/3)

fa= pow(a+1,-2/3)

i<n

-

+

2

4

1

s= (fa-fb)/2

x=a+h

i=0

Рис. 2.5

x+=h

3

s+=pow(x+1,-2/3)

i++

s\*=h

fabs(s-s0)>E

-

+

2

4

1

Вивести s

Рис. 2.5

**Метод Сімпсона.** У цьому методі інтегрування проводиться шляхом поділу відрізка інтегрування *[а, b]* на множину відрізків (*N* пар відрізків). Однак, з метою збільшення точності наближеного інтегрування на кожному відрізку підінтегральної функції *f(x)* замінюють квадратичною параболою (рис. 2.6), обчислення визначеного інтеграла зводиться до обчислення суми *N* криволінійних трапецій . Площа кожної такої трапеції визначається за формулою Сімпсона:

Тоді

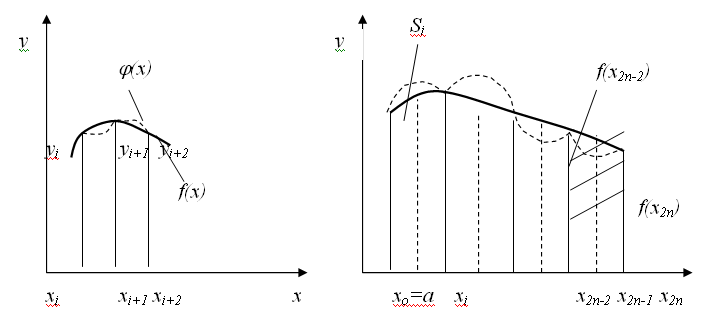


Рис. 2.6

**Блок-схема для методу Сімпсона показана на рис 2.7.**

b=7

3

a=0

n=1

E=0.001

n\*=2

s=0

h=(b-a)/n

s0=s

fb= pow(b+1,-2/3)

fa= pow(a+1,-2/3)

i<n

-

+

2

4

1

s= fa+fb

i=1

Рис. 2.7

s+=2\*(1+i%2)\*fx

3

fx=pow(a+i\*h+1,-2/3)

i++

s\*=h/3

fabs(s-s0)>E

-

+

2

4

1

Вивести s

Рис. 2.7

Програмна реалізація розглянутих алгоритмів має вигляд:

#include <iostream>

#include <conio.h>

#include <math.h>

using namespace std;

double rectangles(double, double, double);

double trapezium(double, double, double);

double simpson(double, double, double);

double f(double);

double const a=0.0, b=7.0, E=0.001;

void main() {

cout<<"1. Rectangles method f(x) = "<<rectangles(a,b,E)<<endl;

cout<<"2. Trapezium method f(x) = "<<trapezium(a,b,E)<<endl;

cout<<"3. Simpson method f(x) = "<<simpson(a,b,E)<<endl;

\_getch();

}

//Метод прямокутників

double rectangles(double a, double b, double E) {

int n = 1;

double h, x, s0 ,s = 0;

do{

n \*= 2;

s0=s;

h=(b-a)/n;

x=a;

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

s += f(x);

x += h;

}

s \*= h;

}

while(fabs(s-s0)>E);

return s;

}

//Метод трапецій

double trapezium(double a, double b, double E) {

int n = 1;

double h, x, s0, s = 0;

do{

n \*= 2;

s0 = s;

h = (b-a)/n;

s = (f(a)-f(b))/2;

x = a+h;

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

s += f(x);

x += h;

}

s \*= h;

}

while(fabs(s-s0)>E);

return s;

}

//Метод Сімпсона

double simpson(double a, double b, double E) {

int n = 1;

double h, s0, s = 0;

do{

n \*= 2;

s0 = s;

h = (b-a)/n;

s = f(a)+f(b);

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

s += 2\*(1+i%2)\*f(a+i\*h);

s \*= h/3;

}

while(fabs(s-s0)>E);

return s;

}

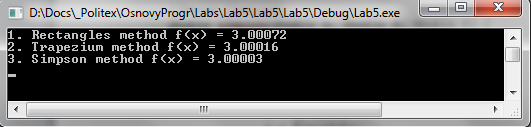
//Підінтегральна функція

double f(double x) {

return pow(x+1,-2.0/3.0);

}

Запустимо програму на виконання (рис 2.7).



(Рис 2.7)

Для перевірки результатів виконання програми, обчислимо інтеграл, використавши теорему Ньютона-Лейбніца, згідно якої:

Обчислимо первісну для нашої функції :

Підставляємо у формулу Ньютона-Лейбніца наші дані, а саме первісну і межі інтегрування:

Як бачимо, програма для всіх трьох методів обчислення визначеного інтеграла, дала результати дуже близькі до точного. Найточніше значення ми отримали методом Сімпсона: 3,00003.

## ВИСНОВКИ

На даній лабораторній роботі я навчився використовувати функції у мові С++. Зокрема вивчив особливості опису формальних та фактичних параметрів функції, локальних та глобальних змінних, а також використанню механізму перекриття імен змінних у С++.