# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**Кафедра програмного забезпечення систем**

**ЗВІТ**

**Про виконання лабораторної роботи №3**

**«Програмування циклічних процесів в С++»**

**з дисципліни «Основи програмування»**

Виконав:

слухач групи ПЗС-11

Гринчук Тарас

Прийняв:

доц. Макар В.М.

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

∑ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

ЛЬВІВ – 2013

**Тема роботи**: Програмування циклічних процесів в С++.

**Мета роботи:** Навчитися організовувати програми циклічної структури, які дозволяють повторювати певну групу операторів задану кількість разів.

## 1. ТЕОРЕТИЧНІ ВІДОМОСТІ

1. Алгоритм називається циклічним, якщо певна послідовність однотипних дій (тіло циклу) виконується багато разів. Однократне виконання тіла циклу називається ітерацією. В залежності від постановки задачі розрізняютарифметичні цикли (тобто цикли з наперед відомою кількістю ітерацій) та ітераційні цикли (число ітерацій такого циклу наперед невідомо). В арифметичних циклах кількість повторень відома до його початку і задається за допомогою лічильника повторень. В ітераційних циклах кількість ітерацій задається неявно, наприклад досягненням якоїсь змінної заданої точності, або заданого значення.
2. У мові С++ використовуються оператори циклу while, do…while та for.
3. В циклах з передумовою спочатку перевіряється умова, а тоді, залежно від того, істинна вона чи хибна, виконується або не виконується тіло циклу.

**Приклад.** Наступний цикл не виконається жодного разу, оскільки умова є хибною:

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

cout<<"begin"<<endl;

int k = 0;

while(k>0)

{

cout<<k<<endl;

k--;

}

cout<<"end"<<endl;

\_getch();

}

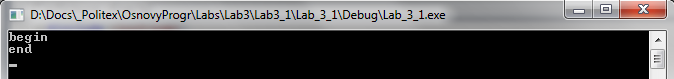


Рис. 1.1

**Приклад.** Наступний цикл пройде 5 ітерацій:

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

cout<<"begin"<<endl;

int k = 5;

while(k>0)

{

cout<<k<<endl;

k--;

}

cout<<"end"<<endl;

\_getch();

}

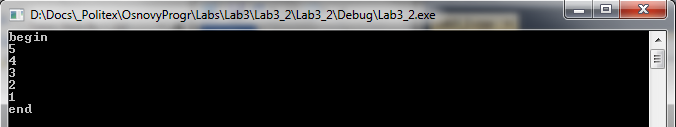


Рис. 1.2

**Приклад.** З клавіатури здійснити ввід довільної кількості чисел та порахувати їхнє середнє арифметичне. Після вводу числа користувачеві задають питання, чи він бажає продовжити (користувач натискає клавіші ‘y’ для продовження або довільну іншу клавішу для завершення вводу даних).

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

char answer = 'y'; int n; double suma = 0; int k = 0;

while (answer=='y'){

cout<<"Enter a number"<<endl;

cin>>n;

cout<<"Go on? y|some another key"<<endl;

cin>>answer;

suma = suma + n;

k++;

}

cout<<"Average: "<<(suma/k)<<endl;

\_getch();

}

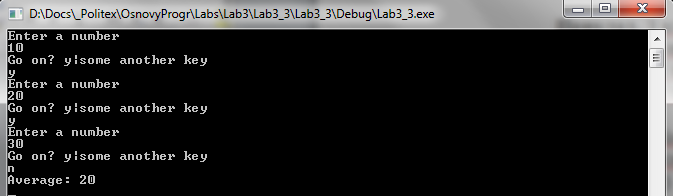


Рис. 1.3

**Приклад.** Порахуємо кількість цифр у введеному з клавіатури числі, за умови, що воно невід’ємне.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

int k = 0; double basenum = 10;

int n;

cout<<"Vvedit chyslo:"; cin>>n;

if(n < 0) cout<<"Chyslo vidyemne";

else {

while((pow(basenum,k))<=n) k++;

cout<<"K-st cyfr:"<<k<<endl;

}

\_getch();

}

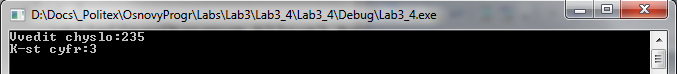


Рис. 1.4

Пояснимо роботу даного фрагмента коду. Нехай n = 235 (рис 1.4). Спочатку змінна k рівна 0, відповідно Перевіряється, чи 1 менше рівне за введене число n. Оскільки умова справджується, то змінна k збільшується на 1 і стає рівною 1. На другій ітерації обчислюється вираз-умова . Оскільки умова істинна, то змінна k збільшується ще на 1 і стає рівною 2. На третій ітерації обчислюється вираз . Оскільки , то k=3. На четвертій ітерації обчислюється вираз-умова. Оскільки умова хибна, то змінна k залишається рівною 3 і робиться висновок, що число n – тризначне.

1. У циклах з післяумовою спочатку виконується тіло, циклу, а тоді перевіряється умова, від істинності якого залежить виконання наступної ітерації циклу. Такий цикл завжди виконується хоча б один раз.

**Приклад.** Умова циклу є від початку хибною, проте цикл встигне виконатися один раз (на екран буде виведено число 0 (рис. 1.5).

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

int k = 0;

do

{

cout<<k<<endl;

k--;

}

while (k>0);

\_getch();

}

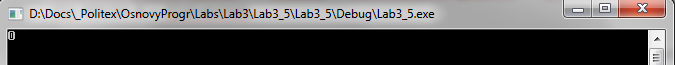


Рис. 1.5

**Приклад.** В наступному циклі робиться ввід символів з клавіатури, поки користувач не натисне символ ‘a’:

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

char ch;

do

cin>>ch;

while (ch!='a');

cout<<"End of cycle.";

\_getch();

}

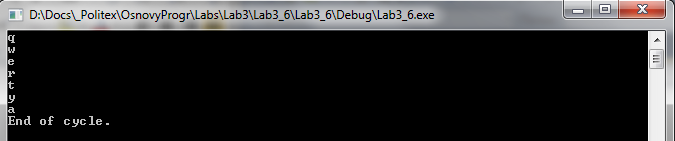


Рис. 1.6

1. Цикл for, як правило, застосовується при заздалегідь відомій кількості ітерацій – наприклад, для опрацювання всіх елементів масиву.

**Приклад.**

#include <iostream>

#include <conio.h>

int main()

{

using namespace std;

/\* Змінній циклу і присвоюється початкове значення 0. Перевіряється умова,

чи і менше 10. Якщо так, то на екран виводиться значення і (і=0). Змінна

циклу і збільшується на 1. Перевіряється умова, чи і менше 10. Виводиться

на екран поточне значення і (і=1). Цикл пройде 10 ітерацій, поки змінна і

не стане рівною 10. В результаті на екран будуть виведені десять цифр від 0

до 9 \*/

for(int i = 0; i != 10; i++) // змінну циклу можна ініціалізувати прямо

//в початковій\_інструкції

cout << i << endl;

\_getch();

return 0;

}

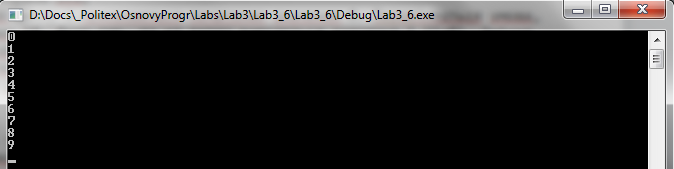


Рис. 1.7

Лічильник циклу може визначатися всередині його заголовка. Область видимості лічильника.

**Приклад.** Цикл може використовувати декілька змінних. В результаті виконання цього циклу буде виведено на екран цифри від 0 до 4, оскільки на 6-ій ітерації циклу змінна j стане рівною 5 і умова циклу перестане бути істинною (рис. 1.8).

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

for(int i = 0, j=10; i <= 10&&j!=5; i++,j--)

cout << i << endl;

\_getch();

}

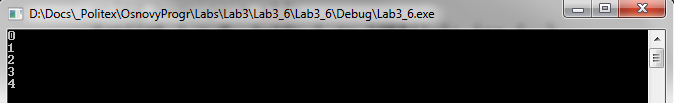


Рис. 1.8

**Приклад.** Наступний цикл виводить на екран всі натуральні парні числа від 1 до 100:

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

for(int i = 1; i <= 100; i++)

if (!(i%2))

cout << i << "\t";

\_getch();

}

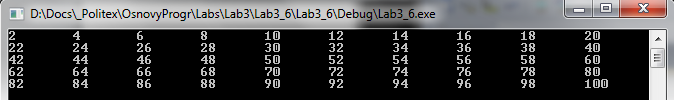
****

Рис. 1.9

Змінна циклу може бути не лише цілим числом, але й літерою.

**Приклад.** В результаті виконання цього циклу на екран будуть виведені усі малі латинські літери (рис. 1.10).

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

for(char ch = 'a'; ch <= 'z'; ch++)

cout << ch << "\t";

\_getch();

}

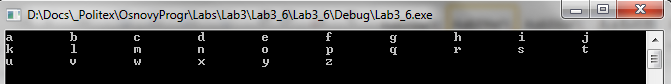
****

Рис. 1.10

**Приклад.** Результатом виконання наступного циклу будуть виведені на екран усі дільники введеного з клавіатури числа n, а також їхня загальна кількість (рис. 1.11).

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

int n;

cout<<"Enter n"<<endl;

cin>>n; int k = 0; /\* число дільників зберігатиметься у цій змінній \*/

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

if(!(n%i)) /\* якщо n ділиться на i без остачі, то збільшуємо кількість

дільників та виводимо i на екран \*/

{k++;

cout<<"Dil'nyk chysla "<<n<<" # "<<k<<": "<<i<<endl;

}

\_getch();

}

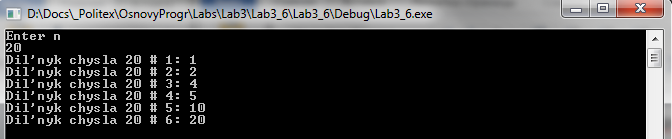
****

Рис. 1.11

1. Коли умова виходу не виконуються на всіх ітераціях циклу – наступає зациклення. Приклад циклів, які виконуються нескінченну кількість раз:

* for(;;){……….}
* while(1){……..}
* do{………}while(1);

1. Оператор break дозволяє здійснити “достроковий” вихід із циклу, коли умова циклу є істинною.

**Приклад.** Визначимо, чи введене з клавіатури число n є простим (число просте, якщо ділиться лише само на себе та на 1).

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

int n;

cout<<"Enter n"<<endl;

cin>>n;

int proste = 1; /\* застосуємо змінну, що служитиме ознакою, є число простим

чи ні. Спочатку вважатимемо, що число просте і у цю змінну запишемо 1 \*/

for(int i=2;i<n;i++) {

if(!(n%i)) /\* якщо знайшлося відмінне від 1 та від n число і, на яке

n ділиться без остачі, то ясно, що число n не є простим, а тому

продовжувати цикл недоцільно \*/

{proste=0; /\* міняємо значення змінної-ознаки. Вона буде

індиктором того, що вихід з циклу відбувся саме через оператор break, а не

завершився через те, що умова циклу стала хибною\*/

break;

}

}

if(proste) cout<<"Chyslo "<<n<<" proste "; /\* якщо змінна-ознака лишилася

незмінною, то число насправді просте \*/

\_getch();

}

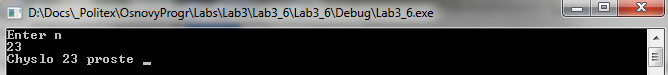


Рис. 1.12

1. Оператор continue дозволяє проігнорувати всі інструкції, записані у тілі циклу після нього, та перейти до наступної ітерації циклу.

**Приклад.** Виведемо тангенс кута angle, де angle перебігає множину всіх парних значень від 0 до 360 у градусах. Відомо, що функція тангенса не визначена для , де n – непарне число.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

double pi = 3.1415926535897932384626433832795;

for(int angle=0;angle <= 360; angle+=2) {

if(!(angle%90)&&((angle/90)%2)) /\* якщо кут ділиться на 90 без

остачі, причому результат цього ділення є непарним числом,

то переходимо до наступної ітерації циклу \*/

continue;

cout<<angle<<"\t"<<(tan(angle\*pi/180))<<"\n";

}

\_getch();

}

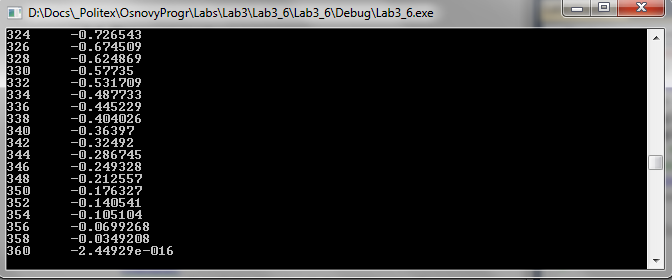


Рис. 1.13

**Приклад.** Напишемо цикл для обчислення наступного виразу:

10 ⋅ 3 + 8 ⋅ 6 + 6 ⋅ 9 + 4 ⋅ 12 + 2 ⋅ 15

Враховуючи, що це є сума добутків відповідних членів двох рядів, організуємо наступний цикл:

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

int n = 10; int m = 3; int i,j; int s = 0; /\* у змінній s зберігатимемо

поточне значення суми \*/

for(i=n,j=m;i>=2;i-=2,j+=3){

cout<<i<<endl;

cout<<j<<endl;

s = s + i\*j;

}

cout<<"Sum: "<<s<<endl;

\_getch();

}

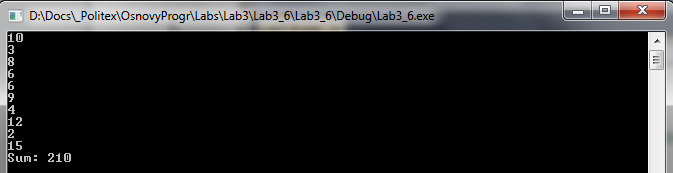


Рис. 1.14

1. Якщо один цикл знаходиться в тілі іншого, то його називають вкладеним. Якщо всередині внутрішнього циклу викликати оператор break, то він вийде тільки з тіла внутрішнього циклу, а зовнішній цикл продовжить виконуватись. Аналогічно, це стосується оператора continue – виконання перейде на наступну ітерацію внутрішнього циклу.

**Приклад.** З клавіатури ввести довільну кількість чисел (ввід числа супроводжується натисканням клавіші ‘y’ для продовження або ж довільної іншої клавіші для завершення вводу даних, як розглядалося вище). Знайти та вивести число з максимальною кількістю дільників, а також вивести саме максимальне число дільників. Потрібний програмний код мовою С++ можна утворити, застосувавши розглянуті вище фрагменти кодів – один для вводу довільної кількості чисел з клавіатури, а другий – для знаходження кількості дільників числа.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

char answer = 'y'; int n; int maxd = 0; int nummux = 0; /\* у змінній maxd

зберігатимемо поточне значення максимальної кількості дільників введеного

числа, а у змінній nummux – саме число, що володіє найбільшою кількістю

дільників \*/

while (answer=='y'){

cout<<"Enter a number"<<endl;

cin>>n;

cout<<"Go on?"<<endl;

cin>>answer;

int k = 0; /\* перед внутрішнім циклом слід не забути «скидати в

нуль» лічильник дільників, оскільки для кожного нового

розглядуваного числа

підрахунок починається наново \*/

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

if(!(n%i))

k++;

if(maxd<k) /\* якщо поточне число дільників виявилося більшим за

maxd, тоді у змінну maxd записуємо поточне число дільників, а

у змінну

nummux – відповідне число n \*/

{ maxd = k;

nummux = n;

}

}

cout<<"Najbil'she dilnykiv "<<maxd<<" maye chyslo: "<<nummux<<endl;

\_getch();

}

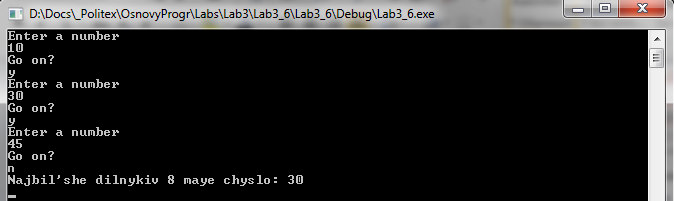


Рис. 1.15

**Приклад.** Роздрукувати таблицю множення.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

/\* Цей цикл виводить в ряд цифри від 0 до 9, після чого пропускаються

два рядки для полегшення читання \*/

for(int i=1;i<10;i++) cout<<"\t"<<i;

cout<<endl<<endl;

for(int i=1;i<10;i++) /\* у зовнішньому циклі “перебираємо цифри від 1

до 9” та виводимо кожне з них на початку рядка \*/

{cout<<i<<"\t";

for(int j = 1;j < 10; j++) /\* у внутрішньому циклі знову

“перебираємо” усі цифри від 1 до 9 та результат множення j на поточне

значення i виводимо на екран \*/

{

cout<<(i\*j)<<"\t";

}

cout<<endl;

}

\_getch();

}

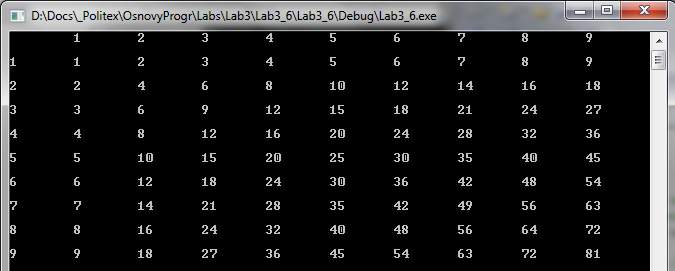


Рис. 1.16

1. В С++ є такі порозрядні операції:
   * Оператор побітового заперечення ~ інвертує всі розряди свого операнда (нулі стають одиницями, і навпаки). Основним призначенням цієї операції є інвертування логічних значень “істина” та “хибність”, а також перетворення додатних чисел у від’ємні і навпаки.
   * Порозрядний оператор зсуву бітів вліво (<<) переносить всі біти першого операнда на задану другим операндом кількість позицій вліво, при цьому позиції, що звільняються, заповнюються нулями. У двійковій системі числення зсув біта на 1 позицію вліво еквівалентний множенню на 2.
   * Аналогічно порозрядний оператор зсуву бітів вправо (>>) переносить усі біти на задану кількість позицій вправо, при цьому позиції, що звільняються справа, заповнюються нулями, а молодші біти зникають. Зсув біта на одну позицію вправо еквівалетне діленню на 2.
   * Порозрядна операція & (“I”). Операція “І” використовується для фільтрування одиниць, які входять у двійкове представлення числа. У цьому випадку другий операнд називається маскою.
   * Порозрядна операція | (“АБО”). Операція “АБО” дозволяє вставляти в результат потрібні розряди.
   * Порозрядна операція ^ (“виключаюче АБО”).
2. Умова, яка перевіряє чи 4-й біт (біти нумеруються починаючи з 0-го) натурального числа є одиницею, має вигляд: **if(n&(1<<4))**. Якщо змістити число 1 на 4 розряди вліво, то в двійковому вигляді получимо результат: 10000. Порозрядне & числа 10000 з будь-яким іншим числом n дасть додатнє значення, тільки в тому випадку, коли 4-й біт n буде одиницею, оскільки множення у решти бітах буде рівним 0 (1&0=0 та 0&0=0).

**Приклад.** Ввести з клавіатури число n. Перевірити, чи воно є степенем 2, крім .

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

int k = 0,n;

cout<<"n: ";

cin>>n;

for(int i = 1; i<=16;i++)

if(n==(1<<i))

{

k = 1;

cout<<"Chyslo ye stepenem 2";

break;

}

if(!k) cout<<"Chyslo ne ye stepenem 2";

\_getch();

}

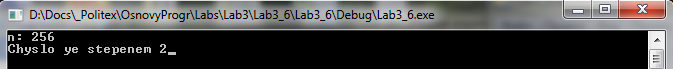
****

Рис. 1.17

Оскільки зсув вліво еквівалентний множенню на 2, то вираз 1<<i дорівнює числу . Тому у циклі for будуть розглянуті всі степені 2 від 2 до . Лічильник циклу перебігає значення від 1 до 16, за припущення, що число типу int займає в пам'яті 16 біт. Оператор break застосовується тому, що продовжувати цикл, коли вже знайдено збіг числа n з числом , немає змісту. Змінна k служить ознакою – спочатку вважаємо, що число не є степенем 2. Якщо в процесі виконання циклу з'ясується протилежне, то значення k міняється. Якщо ж значення k залишається рівним своєму початковому значенню (0), то виводимо повідомлення про те, що введене число не є степенем 2.

1. В ітераційних циклах кількість ітерацій задається неявно, наприклад досягненням якоїсь змінної заданої точності, або заданого значення. Багато задач у математиці передбачає використання числових методів, в яких обчислення проводиться до досягнення заданої точності. Так, наприклад, обчислення суми нескінченого ряду Тейлора проводиться до тих пір поки наступний член ряду за абсолютним значенням стане меншим за значення точності ε, де ε- достатньо мале число. При розв’язуванні таких задач наперед не відома кількість ітерацій циклу.

**Приклад.** Обчислити значення функції, заданої розкладом в ряд:

Оскільки обчислювати – на кожному кроці не ефективно, використаємо рекурентну формулу, тобто формулу знаходження наступного елемента за попереднім:

Програма мовою С++:

#include <iostream>

#include <conio.h>

#include <math.h>

#include <iomanip>

void main()

{

int i;

double x, s, si, eps, y;

using namespace std;

cout<<"Vvedit x i eps:";

cin>>x>>eps;

si=x; i=2; s=1+si;

while (fabs(si)>eps) {

si\*=x/i;

s+=si; i++;

}

y=exp(x); // точне значення

cout<<"x="<<x<<" y="<<y<<" s="<<s;

\_getch();

}

Запустимо програму на виконання з параметрами х=5 та eps=0.00001 (рис. 1.18). Як бачимо, при заданій точності, результат виконання співпадає.

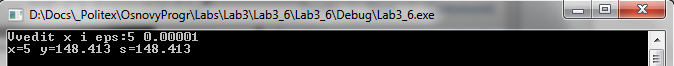


Рис. 1.18

## ХІД РОБОТИ

**Індивідуальне завдання №1 (варіант 5).** Знайти всі двозначні числа, рівні подвоєному добутку їхніх цифр.

Для вирішення цієї задачі, потрібно в циклі перебрати всі числа від 10 до 99 і вивести на екран ті, для яких справджується умова: 2\*x\*y==i, де i-число, х-його перша цифра, а у – друга. Запишемо дану умову як: 2\*х\*у-і==0. Для того, щоб розрахувати першу цифру числа, візьмемо цілу частину частки від ділення його на 10. Другу цифру отримуємо, як остачу від ділення цього числа на 10. Отже на мові С++, можна записати, що x=int(i/10), а y=i%10. Відповідно вираз, можна записати, як: 2\* int(i/10)\*( i%10)—i==0. Скориставшись оператором «!», умову виводу числа на екран, запишемо у більш характерному для С++ вигляді:

if(!(2\*int(i/10)\*(i%10)-i))

Блок-схема для даного алгоритму має вигляд (рис 2.1):

-

+

!(2\*int(i/10)\*(i%10)-i)

+

i=10

-

i<=99

Вивести і

i++

Рис 2.1

Програмна реалізація даного алгоритму на мові С++ має вигляд:

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

for(int i=10;i<=99;i++)

if(!(2\*int(i/10)\*(i%10)-i))

cout<<i<<endl;

\_getch();

}

Запустимо програму на виконання (рис. 2.2). В результаті того, що умова if(!(2\*int(i/10)\*(i%10)-i)) є істинною, для числа 36 (2\*3\*6=36) на екран з’являється відповідне повідомлення.



Рис. 2.2

**Індивідуальне завдання №2 (варіант 5).** З допомогою операторів циклу, протабулювати на відрізку від А до B з області визначення функцію, задану розкладом у ряд Тейлора. Для порівняння обчислити також у кожній точці табуляції значення функції задане формулою. Результати подати у виді таблиці з коментарями.

Для того, щоб не обчислювати на кожному кроці, оскільки це не ефективно, використаємо рекурентну формулу:

Для того, щоб протабулювати функцію на відрізку (-1;1), доцільно розбити його на n рівних відрізків (візьмемо n=20). За початкову точку візьмемо значення *х=-0,9* (оскільки точка *х=1* лежить поза областю визначення функції). Значення *х* будемо обчислювати за формулою:

В зовнішньому циклі обійдемо всі *n* значень *х* на проміжку від *a* до *b*, а у внутрішньому будемо очислювати суму ряду Тейлора *s*, поки абсолютне значення *і*-го елементу ряду буде більшим за вказану нами точність *eps*. Отриману суму ряду *s* порівняємо зі значенням функції *y* обчисленим штатними методами.

Блок-схема алгоритму наведена на рис. 2.3.

-

+

a=-0.9

j<n

Ввести eps

b=1.0

n=20

j=0

x=a+j\*(b-a)/n

si=2\*x

s=si

i=2

fabs(si)>eps

-

+

si\*=x\*x\*(2\*i-3)/(2\*i-1)

s+=si

i++

y=log((1+x)/(1-x))

2

1

3

Вивести x,y,s

j++

2

1

3

Рис. 2.3

Програмна реалізація даного алгоритму має вигляд:

#include <iostream>

#include <conio.h>

#include <math.h>

void main() {

const double a=-0.9,b=1.0; //(a;b) - проміжок табулювання

const int n=20; //кількість точок табулювання

int i;

double x, s, si, eps, y;

using namespace std;

cout<<"eps: ";

cin>>eps;

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

//отримуємо наступне значення x на проміжку табуляції

x=a+j\*(b-a)/(double) n;

//перший елемент ряду

si=2\*x; s=si;

i=2;

/\* збережемо значення x^2 у змінну хх,

щоб не витрачати час на обчислення в циклі while \*/

double xx = pow(x,2);

//виконуємо, поки не досягнуто бажаної точності

while(fabs(si)>eps) {

//обчислимо i-ий елемент ряду Тейлора

si\*=xx\*(2\*i-3)/(2\*i-1);

//збільшимо суму ряду

s+=si;

//збільшимо лічильник

i++;

}

//отримаємо значення ф-ї штатними методами

y=log((1+x)/(1-x));

cout<<"x="<<x<<" y="<<y<<" s="<<s<<endl;

}

\_getch();

}

Запустимо програму на виконання, зі значеннями точності: eps=0.001 (рис. 2.4), eps=0.00001 (рис. 2.5), eps=0.000001 (рис. 2.6).

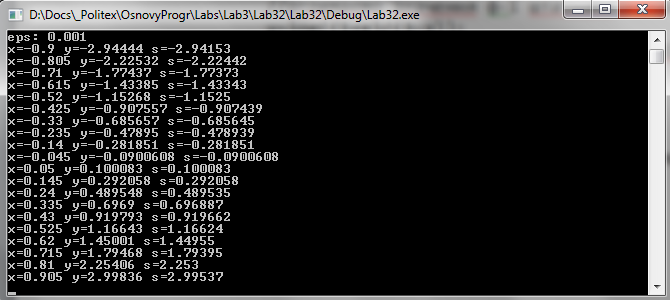


Рис. 2.4

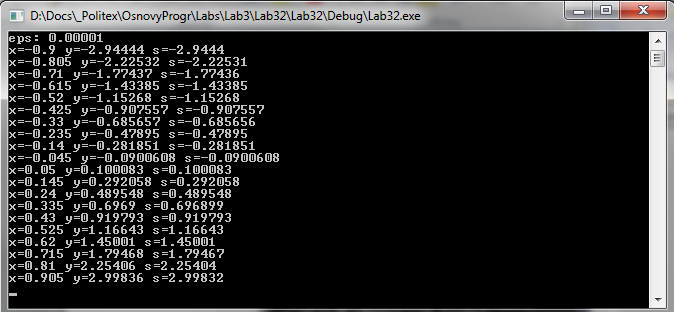


Рис. 2.5

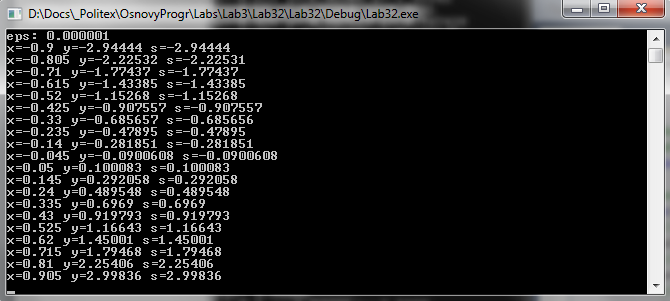


Рис. 2.6

Для порівняння, отримані результати зобразимо у вигляді таблиці (рис. 2.7):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **x** | **y** | **eps=0,001** | | **eps=0,00001** | | **eps=0,000001** | |
| **s** | ***│y-s*│** | **s** | ***│y-s│*** | **s** | ***│y-s│*** |
| 1 | -0,9 | -2,94444 | -2,94153 | ***0,00291*** | -2,94444 | ***0*** | -2,94444 | ***0*** |
| 2 | -0,805 | -2,22532 | -2,22442 | ***0,0009*** | -2,22531 | ***1E-05*** | -2,22531 | ***1E-05*** |
| 3 | -0,71 | -1,77437 | -1,77373 | ***0,00064*** | -1,77436 | ***1E-05*** | -1,77437 | ***0*** |
| 4 | -0,615 | -1,43385 | -1,43343 | ***0,00042*** | -1,43385 | ***0*** | -1,43385 | ***0*** |
| 5 | -0,52 | -1,15268 | -1,1525 | ***0,00018*** | -1,15268 | ***0*** | -1,15268 | ***0*** |
| 6 | -0,425 | -0,0907557 | -0,907439 | ***0,8166833*** | -0,907557 | ***0,816801*** | -0,090756 | ***0*** |
| 7 | -0,33 | -0,685657 | -0,685645 | ***1,2E-05*** | -0,685656 | ***1E-06*** | -0,685656 | ***1E-06*** |
| 8 | -0,235 | -0,47895 | -0,478939 | ***1,1E-05*** | -0,47895 | ***0*** | -0,47895 | ***0*** |
| 9 | -0,14 | -0,281851 | -0,281851 | ***0*** | -0,281851 | ***0*** | -0,281851 | ***0*** |
| 10 | -0,045 | -0,0900608 | -0,090061 | ***0*** | -0,0900608 | ***0*** | -0,090061 | ***0*** |
| 11 | 0,05 | 0,100083 | 0,100083 | ***0*** | 0,100083 | ***0*** | 0,100083 | ***0*** |
| 12 | 0,145 | 0,292058 | 0,292058 | ***0*** | 0,292058 | ***0*** | 0,292058 | ***0*** |
| 13 | 0,24 | 0,489548 | 0,489535 | ***1,3E-05*** | 0,489548 | ***0*** | 0,489548 | ***0*** |
| 14 | 0,335 | 0,6969 | 0,696887 | ***1,3E-05*** | 0,696899 | ***1E-06*** | 0,6969 | ***0*** |
| 15 | 0,43 | 0,919793 | 0,919662 | ***0,000131*** | 0,919793 | ***0*** | 0,919793 | ***0*** |
| 16 | 0,525 | 1,16643 | 1,16624 | ***0,00019*** | 1,16643 | ***0*** | 1,16643 | ***0*** |
| 17 | 0,62 | 1,45001 | 1,44955 | ***0,00046*** | 1,45001 | ***0*** | 1,45001 | ***0*** |
| 18 | 0,715 | 1,79468 | 1,79395 | ***0,00073*** | 1,79467 | ***1E-05*** | 1,79468 | ***0*** |
| 19 | 0,81 | 2,25406 | 2,253 | ***0,00106*** | 2,25404 | ***2E-05*** | 2,25406 | ***0*** |
| 20 | 0,905 | 2,99836 | 2,99537 | ***0,00299*** | 2,99832 | ***4E-05*** | 2,99836 | ***0*** |

Рис. 2.7

З отриманих результатів, можемо зробити висновок, що при зменшенні значення необхідної точності обчислення ***eps***, сума ряду Тейлора ***s*** заданої функції, наближається до значення функції ***у*** обчисленного, за допомогою штатних механізмів С++, при ***eps=0,001***, абсолютне значення похибки ***│y-s│*** рівне нулю, тільки у чотирьох точках, відрізка. При ***eps=0,00001***: значення y,s співпадають у 12-ти точках, а при ***eps=0,000001***: не співпадають значення, лише у 2-х точках.

## ВИСНОВКИ

На даній лабораторній роботі я навчився організовувати програми циклічної структури, які дозволяють повторювати певну групу операторів задану кількість разів, зокрема правильно реалізовувати конструкції: for, while, do while, вивчив особливості застосування операторів continue та та break в циклах, отримав знання про порозрядні операції в С++ та особливості їх застосування.