

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет “Львівська політехніка”



ПРОГРАМУВАННЯ ЦИКЛІЧНИХ ПРОЦЕСІВ В С

ІНСТРУКЦІЯ

до лабораторної роботи № 2 з курсу
“Основи програмування”
для базового напрямку “Програмна інженерія”

Затверджено
На засіданні кафедри
програмного забезпечення
Протокол № від

ЛЬВІВ – 2018

1. МЕТА РОБОТИ

Мета роботи – навчитися організовувати програми циклічної структури, які дозволяють повторювати певну групу операторів задану кількість разів.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1. Поняття про цикли

Алгоритм називається *циклічним*, якщо певна послідовність однотипних дій (тіло циклу) виконується багато разів. Однократне виконання тіла циклу називається *ітерацією*. В залежності від постановки задачі розрізняють *арифметичні* 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.

Отже, цикли дозволяють багатократно виконувати сукупність однотипних операцій. Це означає, що програмний код за рахунок циклів виглядатиме компактнішим. Умовою виходу циклу або ж умовою продовження циклу є вираз, від істинності якого залежить, буде виконуватися тіло циклу чи ні. Цикл може мати лічильник (можливо, не один), що зберігає номер ітерації циклу. Значення лічильника водночас може бути і умовою виходу з циклу. У мові C використовуються оператори циклу **while**, **do...while** та **for**. Усі три оператори циклу в C можуть використовуватися для реалізації як арифметичних, так й ітераційних циклічних процесів. Цикл **while** є циклом з *передумовою*, а цикл **do...while** – з *післяумовою*. В циклах з передумовою *спочатку* перевіряється умова, а *тоді*, залежно від того, істинна вона чи хибна, виконується або не виконується тіло циклу. У циклах з післяумовою *спочатку* виконується тіло циклу, а *тоді* перевіряється умова, від істинності якого залежить виконання *наступної* ітерації циклу. Одне з типових застосувань циклу **while** – перевірка, чи не досягнуто кінець файлу. Цикл **for**, як правило, застосовується при заздалегідь відомій кількості ітерацій – наприклад, для опрацювання всіх елементів масиву.

2.2. Конструкція while

Оператор **while** – це оператор циклу з передумовою. Синтаксис:

while (<умова>) <оператор>;

Цикл виконується, поки умова є істинною. Якщо умова початково є хибною (вираз, що задає умову має нульове значення), цикл не виконається жодного разу. У ролі <оператор>, який і є тілом циклу, може бути будь-який допустимий оператор мови C. Єдине обмеження, яке накладає тут мова C полягає в тому, що, структурно, тіло циклу має мати вигляд одного єдиного оператора, що змушує нас використовувати операторні дужки *тоді*, коли нам потрібно задати тіло циклу за допомогою декількох операторів (аналогічно як було в операторі умови **if**).

Приклад.

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

```
int k = 0;
while (k > 0)
{
    printf("%d", k);
    k--;
}
```

Цей цикл пройде 5 ітерацій:

```
int k = 5;
while (k > 0)
{
    printf("%d", k);
    k--;
}
```

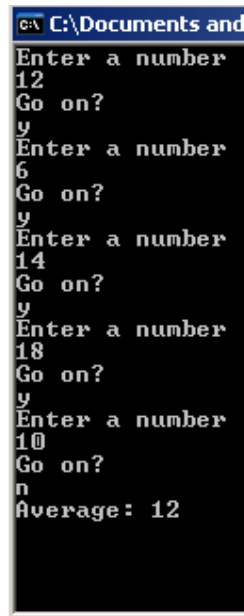
```
}
```

Слід не забувати змінювати змінну, яка використовується в умові циклу *while*, а інакше відбудеться зациклення!

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<stdio.h>
#include<conio.h>
int main(){
char answer = 'y'; int n=0; double suma = 0; int k = 0;
while (answer == 'y'){
    printf("Enter a number\n");
    scanf("%d", &n);
    printf("Go on?\n");
    scanf("%c", &answer);
    printf("%c", answer);
    suma = suma + n;
    k++;
}
printf("Average: %lf ", suma/k);
_getch();
return 0;
}
```

Результат показаний на рис. 1.



```
C:\Documents and Settings\user\My Documents
Enter a number
12
Go on?
y
Enter a number
6
Go on?
y
Enter a number
14
Go on?
y
Enter a number
18
Go on?
y
Enter a number
10
Go on?
n
Average: 12
```

Рис. 1 Результат виконання циклу для довільної кількості чисел, введених з клавіатури

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

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main(){
int k = 0, n=0; double basenum =10;
```

```

printf("Vvedit chyslo:");
scanf("%d",&n);
if(n < 0)
printf("Chyslo vidyemne");
else {
    while((pow(basenum,k))<=n) k++;
    printf("K-st cyfr: %d",k);
}

_getch();
return 0;
}

```

Пояснимо роботу даного фрагмента коду. Нехай $n = 235$. Спочатку змінна k рівна 0, відповідно $10^k = 10^0 = 1$. 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 ($10^k = 10^1 = 10$) $\leq n$. Оскільки умова істинна, то змінна k збільшується ще на 1 і стає рівною 2. На третій ітерації обчислюється вираз ($10^2 \leq n$). Оскільки $100 \leq 235$, то $k = 3$. На четвертій ітерації обчислюється вираз-умова $10^3 \leq 235$. Оскільки умова хибна, то змінна k залишається рівною 3 і робиться висновок, що число n – тризначне.

2.3. Конструкція do...while

Цикл **do...while** відрізняється від циклу **while** тим, що умова його виконання перевіряється після виконання тіла циклу. Це означає, що тіло циклу виконається принаймні один раз. Синтаксис цього оператора:

do <тіло циклу> while (<умова>)

Інша відмінність оператора циклу **do...while** полягає в тому, що тіло циклу може складатися з довільної кількості операторів, тобто використання операторних дужок не є обов'язковим.

Приклад.

```

int k = 0;
do
{
    printf("%d",k);
    k--;
}
while (k>0);

```

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

В наступному циклі робиться ввід символів з клавіатури, поки користувач не натисне символ 'a':

```

char ch;
do
    scanf("%c",&ch);
while (ch!='a');

```

2.4. Конструкція for

Цикл **for** управляється змінною, що називається лічильником циклу. Величину зміни лічильника називають кроком. Синтаксис циклу є наступний:

for(<початкова_інструкція>; <умова>; <вираз>)
<тіло циклу>;

Спочатку виконується початкова інструкція та перевіряється умова. Якщо вона істинна, то виконуються інструкції з тіла циклу, а тоді обчислюється вираз та управління передається в початок циклу з тією різницею, що початкова інструкція вже не виконується.

Приклад.

```

#include<stdio.h>
#include <conio.h>
int main()
{
    /* Змінній циклу i присвоюється початкове значення 0. Перевіряється умова,
    чи i менше 10. Якщо так, то на екран виводиться значення i (i=0). Змінна
    циклу i збільшується на 1. Перевіряється умова, чи i менше 10. Виводиться
    на екран поточне значення i (i=1). Цикл пройде 10 ітерацій, поки змінна i
    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
        printf("%d",i);
    _getch();
    return 0;
}

```

Лічильник циклу може визначатися всередині його заголовка. Область видимості лічильника сягає або кінця тіла циклу, або кінця функції. Значення лічильника можна змінювати всередині тіла циклу.

Цикл може використовувати декілька змінних:

```

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

```

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.

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

```

for(int i = 1; i <= 100; i++)
    if (!(i%2))
        printf("%d",i);

```

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

```

for(char ch = 'a'; ch <= 'z'; ch++)
    printf("%c",ch);

```

В результаті виконання цього циклу на екран будуть виведені усі малі латинські літери.

Результатом виконання наступного циклу будуть виведені на екран усі дільники введеного з клавіатури числа n, а також їхня загальна кількість

```

int n;
printf("Enter n");
scanf("%d",&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++;
        printf("Dil'nyk chysla %d #%d : %d",n,k,i);
    }

```

У циклі **for** можуть бути відсутні ініціалізація лічильника і/або умова виходу і/або приріст. Відсутність умови виходу інтерпретується як істинний вираз, тобто, цикл стає нескінченним. Якщо ініціалізація не виконується в заголовку циклу, то її слід здійснити раніше. Відсутність всіх виразів одночасно (for(;;) {}) означає нескінченний цикл. Згідно синтаксису C, оператор **for** може бути пустим.

2.5. Оператор break

Оператор **break** дозволяє здійснити “достроковий” вихід із циклу, коли умова циклу є істинною. Він використовується тоді, коли продовжувати цикл недоцільно.

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

```
int n;
printf("Enter n");
scanf("%d",&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 i, 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 індикатором того, що вихід з циклу відбувся саме через оператор break, а не завершився через те, що умова циклу стала хибною*/
        break;
    }
}
if(proste) printf("Chyslo %d proste ",n); /* якщо змінна-ознака лишилася незмінною, то число насправді просте */
```

2.6. Оператор continue

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 всі парних значень від 0 до 360 у градусах. Відомо, що функція тангенса не визначена для $\frac{\pi}{2}$, де n – непарне число.

Цикл виглядатиме так:

```
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;
    printf("%d \t %lf",angle, tan(angle*pi/180));
}
```

Приклад. Напишемо цикл для обчислення наступного виразу:

$$10 \cdot 3 + 8 \cdot 6 + 6 \cdot 9 + 4 \cdot 12 + 2 \cdot 15$$

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

```
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){
    printf("%d\n", i);
    printf("%d\n", j);
    s = s + i*j;
}
printf("Sum: %d",s);
```

2.7. Вкладені цикли

У мові C існують *вкладені* цикли. Тобто, один цикл (внутрішній) може знаходитися в тілі іншого циклу (зовнішнього). При цьому глибина вкладення (кількість вкладених циклів) не обмежується. Типове застосування вкладених циклів – для обробки матриць (двовимірних масивів). Цикл **for** може знаходитися у тілі циклу **while** (чи **do...while**), і навпаки.

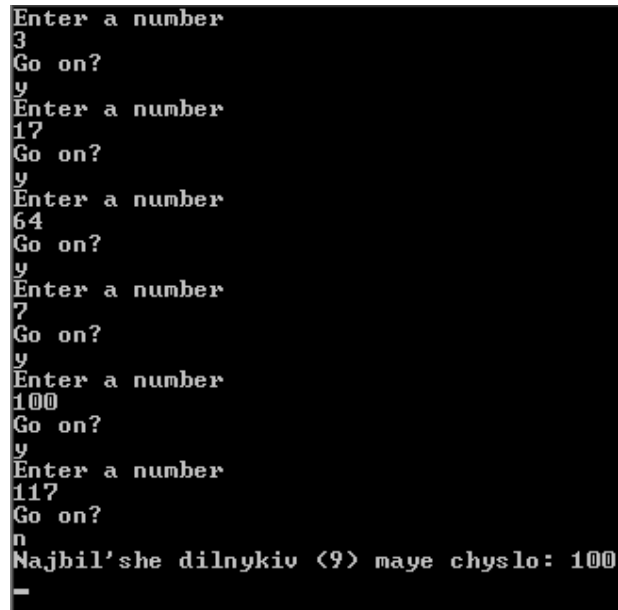
Приклад. З клавіатури ввести довільну кількість чисел (звід числа супроводжується натисканням клавіші 'у' для продовження або ж довільної іншої клавіші для завершення вводу даних, як розглядалося вище). Знайти та вивести число з максимальною кількістю дільників, а також вивести саме максимальне число дільників. Потрібний програмний код мовою C можна

утворити, застосувавши розглянуті вище фрагменти кодів – один для вводу довільної кількості чисел з клавіатури, а другий – для знаходження кількості дільників числа.

```
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'){
    printf("Enter a number");
    scanf("%d",&n);
    printf("Go on?");
    scanf("%c",&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 виявилось більшим за
maxd, тоді у змінну maxd записуємо поточне число дільників, а у змінну
nummux - відповідне число n */
    { maxd = k;
      nummux = n;
    }
}
printf("Najbil'she dilnykiv (%d) maye chyslo:%d",maxd, nummux);
```

Недоліком наведеного коду є те, що якщо два (і більше) числа мають однакову кількість дільників, то відображатиметься тим не менш лише одне число. Для усунення цього недоліку варто записувати всі пораховані значення в масив, а з масиву видобувати *всі* однакові найбільші значення.

Приклад результату виконання показано на рис. 2.



```
Enter a number
3
Go on?
y
Enter a number
17
Go on?
y
Enter a number
64
Go on?
y
Enter a number
7
Go on?
y
Enter a number
100
Go on?
y
Enter a number
117
Go on?
n
Najbil'she dilnykiv <9> maye chyslo: 100
_
```

Рис. 2 Результат виконання програми

Приклад. Роздрукувати таблицю множення.

```
/* Цей цикл виводить в ряд цифри від 0 до 9, після чого пропускаються
два рядки дл полегшення читання */
for(int i=1;i<10;i++)
```

```

printf ("\t%d",i);
printf ("\n");
for(int i=1;i<10;i++) /* у зовнішньому циклі "перебираємо цифри від 1
до 9" та виводимо кожне з них на початку рядка */
{printf ("%d\t",i);
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 */
{
printf ("%d\t",i*j);
}
printf ("\n");
}

```

Для відокремлення чисел зручно використовувати символи табуляції.

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

Рис. 3 Результат роздруку таблиці множення

2.8. Ітераційні цикли

Багато задач у математиці передбачає використання числових методів, в яких обчислення проводиться до досягнення заданої точності. Так, наприклад, обчислення суми нескінченного 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. Крім того в більшості таких задач часто використовується *алгоритм накопичення добутку*:

$$S_{p_i} = S_{p_{i-1}} * p$$

Продемонструємо сказане на прикладі :

Обчислити значення функції, заданої розкладом в ряд:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + \dots$$

з точністю $\varepsilon=10^{-5}$, тобто $|S_i| < \varepsilon$, де S_i – елемент ряду.

Оскільки обчислювати $\frac{x^n}{n!}$ - на кожному кроці неефективно, використаємо, так звану *рекурентну формулу*, тобто формулу знаходження наступного елемента за попереднім:

$$S_1 = \frac{x}{1!}$$

$$S_2 = S_1 * p = \frac{x}{1!} * \frac{x}{2} = \frac{x^2}{2!}$$

$$S_3 = S_2 * p = \frac{x^2}{2!} * \frac{x}{3} = \frac{x^3}{3!}$$

.....

$$S_i = S_{i-1} * \frac{x}{i}$$

Блок-схема алгоритму з детальними поясненнями наведена на рис.4.

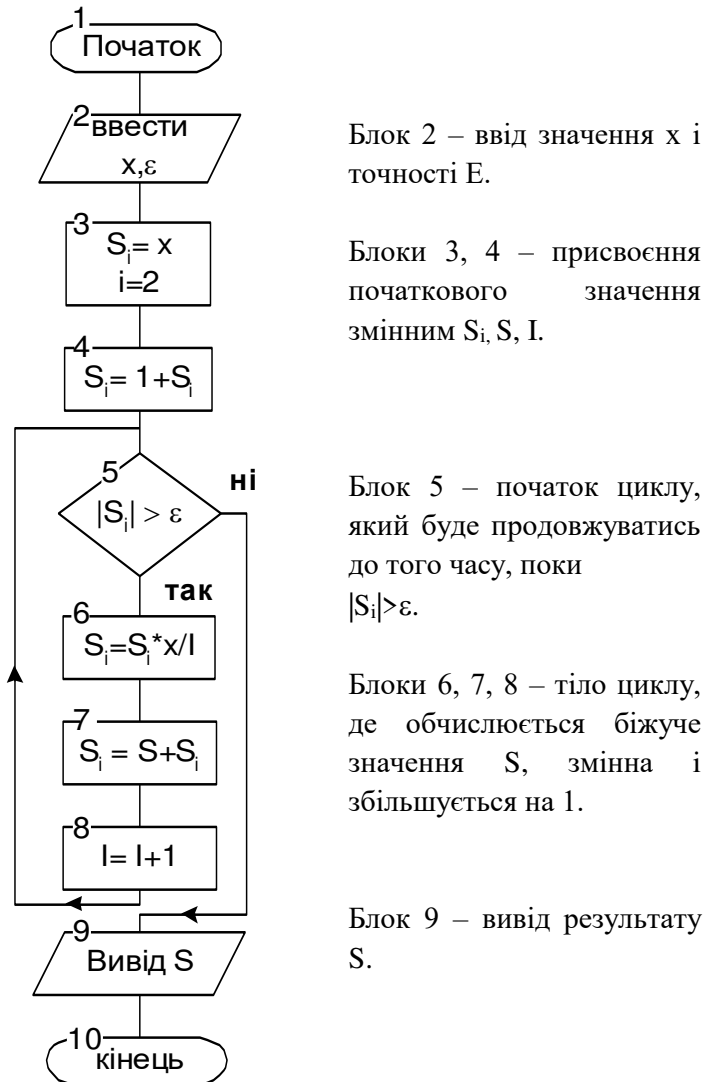


Рис.4. Блок-схема обчислення суми елементів нескінченного ряду Тейлора.

Сама програма мовою C:

```

void main()
{
    int i;
    double x, s, si, eps, y;

    printf("Vvedit x i eps:");
    scanf("%lf, %lf", &x, &eps);
    si=x; i=2; s=1+si;
    while (fabs(si)>eps) {
        si*=x/i;
        s+=si; i++;
    }
}
  
```

```

    }
    y=exp(x); // точне значення
    printf("x=%lf y=%lf s=%lf", x, y, s);
    _getch();
}

```

2.9. Побітові операції

Оскільки числа в комп'ютері представляються у двійковій системі числення і комп'ютер виконує обчислення саме над двійковими числами, то в мові C передбачені побітові операції, які дозволяють перевіряти та встановлювати задані розряди чи множину розрядів у ціло-чисельних змінних. Часто побітові операції використовуються для швидкого множення на 2.

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:

$$3 << 4 = 3 * 2 * 2 * 2 * 2 = 3 * 16 = 48$$

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 еквівалентне діленню на 2:

$$32 > 4 = 32 / (2 * 2 * 2 * 2) = 32 / 16 = 2.$$

Порозрядна операція **& (“І”)** є бінарною та задається таблицею істинності:

Біт1	Біт2	Біт1 & Біт2
0	0	0
0	1	0
1	0	0
1	1	1

Операція “І” використовується для фільтрування одиниць, які входять у двійкове представлення числа. У цьому випадку другий операнд називається маскою.

Порозрядна операція **| (“АБО”)** є бінарною та задається таблицею істинності:

Біт1	Біт2	Біт1 Біт2
0	0	0
0	1	1
1	0	1
1	1	1

Операція “АБО” дозволяє вставляти в результат потрібні розряди.

Порозрядна операція **^ (“виключаюче АБО”)** є бінарною та задається таблицею істинності:

Біт1	Біт2	Біт1 ^ Біт2
0	0	0
0	1	1
1	0	1
1	1	0

Приклад. Ввести з клавіатури число n. Перевірити, чи воно є степенем 2, крім 2⁰.

```

int k = 0, n=0;
printf("Vvestu chyslo");
scanf("%d", &n);
for(int i = 1; i<=16;i++)

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

```

```

    {
        k = 1;
        printf("Chyslo ye stepenem 2");
        break;
    }
    if(!k) printf("Chyslo ne ye stepenem 2");

```

Оскільки зсув вліво еквівалентний множенню на 2, то вираз $1 \ll i$ дорівнює числу 2^i . Тому у циклі for будуть розглянуті всі степені 2 від 2 до 2^{16} . 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 2^i , 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.

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке цикл? Яка різниця між ітераційними та арифметичними циклами?
2. Які оператори мови C використовуються для організації циклічного процесу?
3. Які особливості виконання оператора циклу з передумовою?
4. Які Ви знаєте особливості виконання оператора циклу з післяумовою?
5. Як виконується оператор циклу for?
6. Що таке зациклення? Наведіть хоча б два приклади операторів циклу мови C, які породжують нескінченний цикл.
7. Для чого призначений оператор break? Наведіть приклад його використання.
8. Яке призначення оператора continue? Чим він відрізняється від оператора break?
9. Що таке вкладені цикли? Яким буде результат використання оператора break у внутрішньому циклі: достроково завершиться тільки внутрішній цикл чи повністю увесь вкладений цикл?
10. Які порозрядні операції мови C Ви знаєте? Яка операція зсуву (вправо чи вліво) еквівалентна множенню на 2?
11. Наведіть приклад, як за допомогою побітової операції множення можна перевірити чи четвертий молодший біт заданого натурального числа містить значення 1.
12. Які особливості ітераційних циклічних процесів? В яких випадках вони часто використовуються?

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Ознайомитися з теоретичним матеріалом викладеним вище в даній інструкції і виконати приклади програм.
2. Одержати індивідуальне завдання з Додатку 1.
3. Розробити алгоритм розв'язання індивідуального завдання і подати його у вигляді блок-схеми.
4. Скласти програму на мові C у відповідності з розробленим алгоритмом.
5. Виконати обчислення по програмі.
6. Одержати індивідуальне завдання з Додатку 2.
7. Розробити алгоритм розв'язання індивідуального завдання і подати його у вигляді блок-схеми.
8. Скласти програму на мові C у відповідності з розробленим алгоритмом.
9. Виконати обчислення по програмі при різних значеннях точності і порівняти отримані результати.
10. Підготувати та здати звіт про виконання лабораторної роботи.

5. СПИСОК ЛІТЕРАТУРИ

1. Керниган Б., Ритчи Д. Язык программирования С. - М. - Финансы и статистика. - 1992. – 272 с.
2. Уэйт М., Прата С., Мартин Д. Язык С. Руководство для начинающих. - М. - Мир. - 1988. – 512 с.
3. К. Джемса. Учимся программировать на языке С. М.: Мир, 1997. – 320 с
4. Герберт Шилдт. Полный справочник по С++. М. – С.-П.-К., Вильямс. – 2003. – 800 с.
5. Демидович Е. М. Основы алгоритмизации и программирования. Язык Си. (Учебное пособие). – Санкт-Петербург: “БХВ Петербург”. – 2006. – 439 с.

6. ІНДИВІДУАЛЬНІ ЗАВДАННЯ

ДОДАТОК 1

Використовуючи цикли:

1. Надрукувати в зростаючому порядку усі тризначні натуральні числа, в записі яких немає однакових цифр.
2. Надрукувати у спадному порядку усі чотиризначні натуральні числа, в записі яких є дві однакові цифри.
3. Задано натуральне число. Надрукувати усі прості дільники цього числа.
4. Порахувати, скільки разів зустрічається кожна цифра у введенному з клавіатури числі.
5. Знайти всі двозначні числа, рівні подвоєному добутку їхніх цифр.
6. Знайти і вивести всі цілі числа в діапазоні від 2 до n , для яких у двійковому представленні числа кількість нулів менша за кількість одиниць.
7. Надрукувати всі чотиризначні цілі додатні числа, у запису яких є дві цифри 5.
8. Визначити і надрукувати номери і значення двох чисел Фібоначі, в проміжок між якими потрапляє введенне з клавіатури натуральне число.
9. Надрукувати всі чотиризначні цілі додатні числа, у запису яких немає цифри 7
10. Обчислити кількість точок з цілочисельними координатами (x,y) , які потрапляють у внутрішню область, описану колом з радіусом R та з центром в початку координат.
11. Надрукувати усі тризначні натуральні числа, сума цифр яких дорівнює 16.
12. Знайти і вивести всі цілі числа в діапазоні від 2 до n , для яких у двійковому представленні числа одиниці знаходяться тільки в парних розрядах.
13. Надрукувати усі п'ятизначні натуральні числа, сума цифр яких перевищує 18.
14. Задано послідовність додатніх цілих чисел, яка закінчується значення -1. Визначити, чи ці числа утворюють зростаючу послідовність.
15. Надрукувати усі числа, утворені діленням кожного тризначного натурального числа, в якого не повторюються цифри, на сума його цифр, помножену на 2.
16. З клавіатури ввести границі числового проміжку. Знайти і надрукувати всі парні числа з цього проміжку, які не містять цифри 7. Знайти їх кількість або вивести повідомлення про їх відсутність.
17. Надрукувати усі чотиризначні натуральні числа, сума цифр яких не перевищує 23.
18. Знайти і вивести всі цілі числа в діапазоні від 2 до n , для яких значення молодшого байта менше за значення старшого байта в двійковому представленні числа.
19. Надрукувати усі числа, утворені діленням кожного чотиризначного натурального числа, серед цифр якого є обов'язково 1, на корінь квадратний з потроєної суми цифр цього числа.
20. Скласти програму, яка читає натуральне число і визначає чи воно дорівнює сумі квадратів яких-небудь 3 натуральних чисел.
21. Скласти програму, яка читає чисельник та знаменник дробу і друкує їх після скорочення.
22. Надрукувати всі числа Фібоначі, які потрапляють у проміжок, заданий двома введенними з клавіатури натуральними числами.
23. Знайти і вивести всі цілі числа в діапазоні від 2 до n , для яких значення молодшого байта більше за значення старшого байта в двійковому представленні числа.

24. Знайти число, яке отримується шляхом виписування в зворотньому порядку цифр заданого цілого числа n .
25. Визначити чи є задане натуральне число паліндромом, тобто таким числом, десятковий запис якого читається однаково зліва направо і справа наліво.

ДОДАТОК 2

З допомогою операторів циклу, протабулювати на відрізьку від А до В 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.

1. $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} \pm \dots; \quad |x| < \infty$
2. $\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + (-1)^n \frac{x^{2n}}{(2n)!} \pm \dots; \quad |x| < \infty$
3. $\ln x = 2 \sum_{k=1}^{\infty} \frac{1}{2k+1} \cdot \left(\frac{x-1}{x+1} \right)^{2k+1} = 2 \left[\frac{x-1}{x+1} + \frac{(x-1)^3}{3(x+1)^3} + \frac{(x-1)^5}{5(x+1)^5} + \dots + \frac{1}{2n+1} \cdot \left(\frac{x-1}{x+1} \right)^{2n+1} + \dots \right]; \quad x > 0$
4. $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots + (-1)^{n+1} \frac{x^n}{n} + \dots; \quad -1 < x \leq 1$
5. $\ln \left(\frac{1+x}{1-x} \right) = 2 \sum_{k=1}^{\infty} \frac{1}{2k+1} x^{2k+1} = 2 \left[\frac{x}{1} + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \dots + \frac{x^{2n+1}}{2n+1} + \dots \right]; \quad |x| < 1$
6. $\arcsin x = x + \frac{x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} + \frac{1 \cdot 3 \cdot 5 x^7}{2 \cdot 4 \cdot 6 \cdot 7} + \dots + \frac{1 \cdot 3 \cdot 5 \dots (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \dots (2n)(2n+1)} + \dots; \quad |x| < 1$
7. $\operatorname{arctg} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots + (-1)^n \frac{x^{2n+1}}{2n+1} + \dots; \quad |x| < 1$
8. $(1+x)^{1/2} = 1 + \frac{x}{2} - \frac{1 \cdot 1}{2 \cdot 4} x^2 + \frac{1 \cdot 1 \cdot 3}{2 \cdot 4 \cdot 6} x^3 - \frac{1 \cdot 1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6 \cdot 8} x^4 + \dots; \quad |x| \leq 1$
9. $(1+x)^{1/3} = 1 + \frac{x}{3} - \frac{1 \cdot 2}{3 \cdot 6} x^2 + \frac{1 \cdot 2 \cdot 5}{3 \cdot 6 \cdot 9} x^3 - \frac{1 \cdot 2 \cdot 5 \cdot 8}{3 \cdot 6 \cdot 9 \cdot 12} x^4 + \dots; \quad |x| \leq 1$
10. $(1+x)^{1/4} = 1 + \frac{x}{4} - \frac{1 \cdot 3}{4 \cdot 8} x^2 + \frac{1 \cdot 3 \cdot 7}{4 \cdot 8 \cdot 12} x^3 - \frac{1 \cdot 3 \cdot 7 \cdot 11}{4 \cdot 8 \cdot 12 \cdot 16} x^4 + \dots; \quad |x| \leq 1$
11. $(1+x)^{-1} = 1 - x + x^2 - x^3 + x^4 - \dots; \quad |x| < 1$
12. $(1+x)^{-2} = 1 - 2x + 3x^2 - 4x^3 + 5x^4 - \dots; \quad |x| < 1$
13. $(1+x)^{-3} = 1 - \frac{1}{1 \cdot 2} (2 \cdot 3x - 3 \cdot 4x^2 + 4 \cdot 5x^3 - 5 \cdot 6x^4 + \dots); \quad |x| < 1$

$$14. (1+x)^{-4} = 1 - \frac{1}{1 \cdot 2 \cdot 3} (2 \cdot 3 \cdot 4x - 3 \cdot 4 \cdot 5x^2 + 4 \cdot 5 \cdot 6x^3 - 5 \cdot 6 \cdot 7x^4 + \dots); \quad |x| < 1$$

$$15. (1+x)^{-1/2} = 1 - \frac{1}{2}x + \frac{1 \cdot 3}{2 \cdot 4}x^2 - \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6}x^3 + \frac{1 \cdot 3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6 \cdot 8}x^4 - \dots; \quad |x| < 1$$

$$16. (1+x)^{-1/3} = 1 - \frac{1}{3}x + \frac{1 \cdot 4}{3 \cdot 6}x^2 - \frac{1 \cdot 4 \cdot 7}{3 \cdot 6 \cdot 9}x^3 + \frac{1 \cdot 4 \cdot 7 \cdot 10}{3 \cdot 6 \cdot 9 \cdot 12}x^4 - \dots; \quad |x| < 1$$

$$17. (1+x)^{-1/4} = 1 - \frac{1}{4}x + \frac{1 \cdot 5}{4 \cdot 8}x^2 - \frac{1 \cdot 5 \cdot 9}{4 \cdot 8 \cdot 12}x^3 + \frac{1 \cdot 5 \cdot 9 \cdot 13}{4 \cdot 8 \cdot 12 \cdot 16}x^4 - \dots; \quad |x| < 1$$

$$18. chx = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \dots + \frac{x^{2n}}{(2n)!} \pm \dots; \quad |x| < \infty$$

$$19. (1-x)^{1/2} = 1 - \frac{x}{2} - \frac{1 \cdot 1}{2 \cdot 4}x^2 - \frac{1 \cdot 1 \cdot 3}{2 \cdot 4 \cdot 6}x^3 - \frac{1 \cdot 1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6 \cdot 8}x^4 - \dots; \quad |x| \leq 1$$

$$20. (1-x)^{1/3} = 1 - \frac{x}{3} - \frac{1 \cdot 2}{3 \cdot 6}x^2 - \frac{1 \cdot 2 \cdot 5}{3 \cdot 6 \cdot 9}x^3 - \frac{1 \cdot 2 \cdot 5 \cdot 8}{3 \cdot 6 \cdot 9 \cdot 12}x^4 - \dots; \quad |x| \leq 1$$

$$21. (1-x)^{-2} = 1 + 2x + 3x^2 + 4x^3 + 5x^4 + \dots; \quad |x| < 1$$

$$22. (1-x)^{-3} = 1 + \frac{1}{1 \cdot 2} (2 \cdot 3x + 3 \cdot 4x^2 + 4 \cdot 5x^3 + 5 \cdot 6x^4 + \dots); \quad |x| < 1$$

$$23. (1-x)^{-4} = 1 + \frac{1}{1 \cdot 2 \cdot 3} (2 \cdot 3 \cdot 4x + 3 \cdot 4 \cdot 5x^2 + 4 \cdot 5 \cdot 6x^3 + 5 \cdot 6 \cdot 7x^4 + \dots); \quad |x| < 1$$

$$24. (1-x)^{-1/2} = 1 + \frac{1}{2}x + \frac{1 \cdot 3}{2 \cdot 4}x^2 + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6}x^3 + \frac{1 \cdot 3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6 \cdot 8}x^4 + \dots; \quad |x| < 1$$

$$25. (1-x)^{-1/3} = 1 + \frac{1}{3}x + \frac{1 \cdot 4}{3 \cdot 6}x^2 + \frac{1 \cdot 4 \cdot 7}{3 \cdot 6 \cdot 9}x^3 + \frac{1 \cdot 4 \cdot 7 \cdot 10}{3 \cdot 6 \cdot 9 \cdot 12}x^4 + \dots; \quad |x| < 1$$

$$26. (1-x)^{-1/4} = 1 + \frac{1}{4}x + \frac{1 \cdot 5}{4 \cdot 8}x^2 + \frac{1 \cdot 5 \cdot 9}{4 \cdot 8 \cdot 12}x^3 + \frac{1 \cdot 5 \cdot 9 \cdot 13}{4 \cdot 8 \cdot 12 \cdot 16}x^4 + \dots; \quad |x| < 1$$