Технология программирования задач с циклами.

Виды циклов

1. Общая характеристика операторов цикла
2. Счётный оператор цикла for
3. Оператор цикла с предусловием
4. Оператор цикла с постусловием
5. Типовые приемы программирования циклических процессов

В языке C++ имеется <u>три вида операторов цикла</u>:

- for - оператор цикла
  с параметром - счетчиком
  (счетный оператор цикла)
  (или арифметический оператор цикла).
- while - оператор цикла
  с предварительным условием (с предусловием);
- do-while - оператор цикла
  с последующим условием
  (с постусловием);

Оператор цикла for

применяется при заранее известном количестве повторений.

При этом некоторая переменная, называемая параметром цикла, должна последовательно принимать значения от начального до конечного.

Операторы цикла while и do -while

применяются в тех случаях, когда известно

условие выполнения цикла,

а количество повторений может быть заранее не известно.

Счетный (арифметический) оператор цикла и массивы.

1. Краткие теоретические сведения и программы с оператором for.

- Цикл ***for*** организуется с помощью специальной переменной, которая называется *параметром цикла*.
- *Параметр цикла* - это числовая переменная, которая управляет работой цикла. Она изменяется по закону арифметической прогрессии, что обеспечивает повторение цикла нужное количество раз.

Параметры цикла

- Для определения количества повторений заранее должны быть известны:
- начальное значение параметра - $t_{нач}$;
- конечное значение параметра - $t_{кон}$;
- шаг изменения параметра - Δt.
- Тогда *количество повторений цикла* определится по формуле:

$$n = \left[\frac{t_{кон} - t_{нач}}{\Delta t} \right] + 1$$

Структура цикла *for* на С++

имеет 4 блока, выполняющиеся в следующей последовательности:

1.- *блок инициализации* параметру цикла присваивается начальное значение;

2.- *условие* выхода из цикла (или, напротив - условие повторения цикла) - проверка параметра на конечное значение

3.- *тело цикла* основные действия, которые повторяются каждый раз, на каждом витке цикла;

4.- *блок изменения* параметра цикла на величину шага.

Блок-схема арифметического цикла и общий вид и работа цикла for

Параметр цикла

Начальное значение

Изменение параметра цикла

for(<*п.цк.*> = <н.з.>; <*условие выполнения цикла*>; <*изм. п.цк.*>)

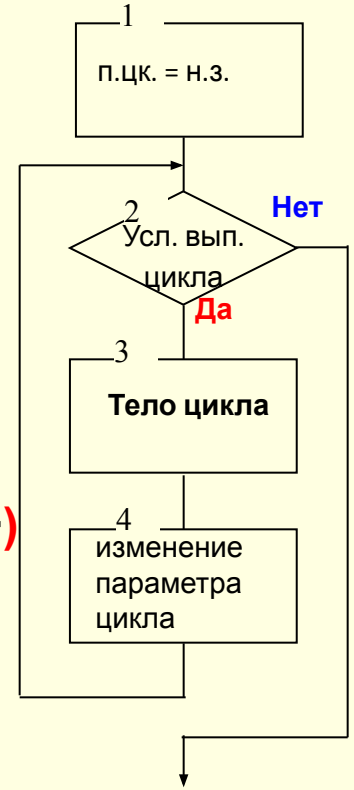
<*оператор*>; ← ***тело цикла***

Любой оператор

Блок инициализации

for(<*п.цк.*> = <н.з.>; <*условие выполнения цикла*>; <*изм. п.цк.*>)
{<*оператор1*>; <*оператор2*>; … <*операторn*>; }

тело цикла



***Пример 1*. Допустим, что в группе из N человек собираются профсоюзные взносы по m рублей. Рассчитать, какую сумму группа перечислит в профсоюзный фонд.**

Метод накопления суммы

```
#include<iostream>
using namespace std;
int main()
{int N=20,m=100,s=0;
for(int i=1; i<=N; i++)
s+=m;
cout<< "Summa = "<<s<<endl;
system("pause");
  return 0;
}
```

2. Программа для определения максимального элемента в массиве

Дано: массив (x_1, x_2, x_3, … x_{10}), x_i = [± xx,xx].

Найти: номер и значение максимального элемента массива.

Константа:

размер массива (кол-во эл-тов): n;

Входные величины:

массив x;

Параметр цикла: i;

Выходные величины:

номер максимального элемента: im

значение максимального элемента: x[im]

ФРАГМЕНТ ПРОГРАММЫ: **поиск минимакса по индексу**

```
const int n=10;
double x[n];
cout << "enter  " << n << "   numbers\n";
for ( int  i  = 0; i < n;  i++)
{
  cout << "x[" << setw(2) << i << "]: ";
  cin >> x[i];
}
cout  <<  endl  <<  "Massiv   x "  <<  endl;
for(int  i  = 0; i < n;  i++)
  cout << setw(4) << x[i];
cout  <<  endl;
int  im = 0;
for(int  i  = 1;  i < n;  i++)
  if(x[i] > x[im])
        im = i;

cout << "im = "  <<  im  <<  "   x[im]= " <<  x[im] <<  endl;
```

```
enter 10 numbers
x[ 0]: 3
x[ 1]: 2
x[ 2]: 5
x[ 3]: 7
x[ 4]: 6
x[ 5]: 9
x[ 6]: 8
x[ 7]: 1
x[ 8]: 4
x[ 9]: 6
```

```
Massiv x
```

```
   3   2   5   7   6   9   8   1   4   6
```

1	2	3	4	5	6	7	8	9
-	+	+	-	+	-	-	-	-
-	2	3	-	5	-	-	-	-

```
im = 5   x[im]= 9
```

3. Программа для определения первого по порядку следования элемента массива, значение которого равно заданной величине

Дано: целочисленный массив (f_1, f_2, f_3, … f_5),
заданное значение: h.

Найти: номер первого по порядку следования элемента со значением h.

Константа:
размер массива (кол-во эл-тов): n;

Входные величины:
массив f;
значение h.

Параметр цикла: i;

Выходные величины: номер искомого элемента: ih;
значение найденного элемента: f[ih].

Метод применения «флага»

```
const int n=5;
int f[n];      int h, ih=-1; //ih – это флаг
cout << "enter  " << n << "   numbers\n";
for ( int i=0; i<n; i++)
{       cout << "f[" << setw(2) << i << "]: ";
  cin >> f[i];     }
  cout << endl << "Massiv f " << endl;
for(int i=0; i<n; i++)
  cout << setw(4) << f[i];
cout << endl;
cout << "h: "; cin >> h;
for(int i=0; i<n; i++)
  if(f[i] == h)
  {     ih = i;
        break;   }
if(ih != -1)
     cout << "ih = " << ih << "   h = " << h ;
else
     cout << " No " ;
cout << endl;
```

```
enter  5   numbers
f[ 0]: 3
f[ 1]: 2
f[ 2]: 5
f[ 3]: 7
f[ 4]: 5
```

```
Massiv f
```

```
   3   2   5   7   5
```

```
h: 5
```

0	1	2
-	-	+
-	-	2

```
ih = 2    h = 5
```

4. Программа для определения среднего арифметического положительных элементов массива

Дано: массив (b_1, b_2, b_3, … b_6), b_i = [± x,x].

Найти: среднее арифметическое положительных элементов массива.

Константа:

размер массива (кол-во эл-тов): n;

Входные величины:

массив b;

Параметр цикла: i;

Выходные величины:

кол-во полож. эл-тов: k;

ср. арифм. полож. эл-тов : s.

Метод включения счётчика

```
const int n=6;
double b[n];
cout << "enter  " << n << "   numbers\n";
for ( int i=0; i<n; i++)
{    cout << "b[" << setw(2) << i << "]: ";  cin >> b[i];  }
cout << endl << "Massiv b " << endl;
for(int  i = 0;  i < n;  i++)  cout  <<  setw(8)  << b[i];
cout << endl;
```

```
enter  6   numbers
b[ 0]: -1
b[ 1]: 2
b[ 2]: -3
b[ 3]: 4
b[ 4]: -5
b[ 5]: 6
```

```
Massiv b
```

```
    -1     2    -3     4    -5     6
```

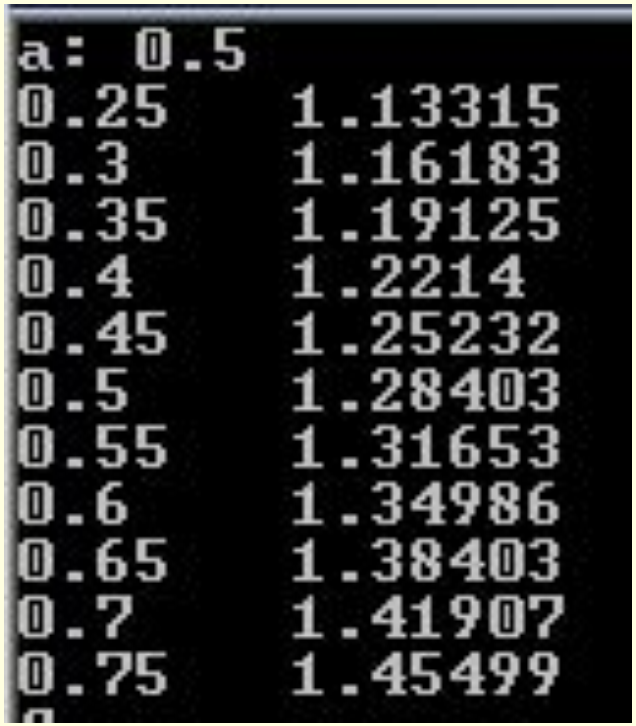
```
int   k=0;   double  s = 0;   //счётчик k и накопитель суммы s
    for(int i = 0; i < n; i++)
      if(b[i] > 0)
      {     s +=b[i];
            k++;   }
    if(k != 0)
    { s = s / k;
      cout  <<  "k = "  <<  k  <<  "   s = "  <<  s ;   }
    else
      cout  <<  " No " ;
    cout << endl;
```

0	1	2	3	4	5
-	+	-	+	-	+
-	2	-	6	-	12
-	1	-	2	-	3

```
k = 3    s = 4
```

5. Пример программы с использованием счетного оператора цикла

$y = e^{ax}$, $x \in [0{,}25; 0{,}75]$, $\Delta x = 0{,}05$



Кол-во циклов

$$n = \frac{\text{к.з.} - \text{н.з.}}{\text{шаг}} + 1 = \frac{0.75 - 0.25}{0.05} + 1 = 10 + 1 = 11$$

Программа с использованием счетного оператора цикла

$y = e^{ax}$, $x \in [0{,}25; 0{,}75]$, $\Delta x = 0{,}05$

```
int main( )
{
   double a, y; int n, i;
   cout << "a: ";  cin >> a;
   double x,  xn = 0.25, xk = 0.75, dx = 0.05;
    n = (xk - xn) / dx+1;
   cout <<  "\ni" << setw(5) << "x" << "          y\n\n";

   for (x=xn, i = 1 ; i < =n  ; i++, x += dx)
   {
      y = exp(a * x);

     cout << left <<  setw(5) << i << setw(7) << x << y << endl;

      x += dx;
   }
  return 0;
}
```

```
a: 0.5

i     x       y

1     0.25    1.13315
2     0.3     1.16183
3     0.35    1.19125
4     0.4     1.2214
5     0.45    1.25232
6     0.5     1.28403
7     0.55    1.31653
8     0.6     1.34986
9     0.65    1.38403
10    0.7     1.41907
11    0.75    1.45499
```

6. Параметр цикла вещественного типа

$y = e^{ax}$, $x \in [0{,}25; 0{,}75]$, $\Delta x = 0{,}05$

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main( )
{
   double a, y;
   cout << "a: ";  cin >> a;
 double xn = 0.25, xk=0.75, dx=0.05;

for(double x = xn; x < = xk; x += dx)
   {
     y = exp(a * x);
     cout << left << setw(7) << x << y << endl;
    }
  return 0;
}
```

```
a: 0.5
0.25   1.13315
0.3    1.16183
0.35   1.19125
0.4    1.2214
0.45   1.25232
0.5    1.28403
0.55   1.31653
0.6    1.34986
0.65   1.38403
0.7    1.41907
Для продолжения нажм
```

Вещественные значения НЕЛЬЗЯ сравнивать на «равно»: в силу **приближённого** представления в цифровом ПК вещественных чисел.

$X_N = 0.25_{10} = 0.01_2 = 0.1 \cdot 2^{-1}$

$dX = 0.05_{10} = 0.0000\ \underline{1100}\ \underline{1100}_2 = 0.(1100) \cdot 2^{-4}$

$X_k = 0.75_{10} = 0.11_2 = 0.11 \cdot 2^{0}$

Нормализованный экспоненциальный формат

									2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216
X_N	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dX	0	1	0	0	0	1	0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
X_k	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{32} + \frac{1}{64} + \frac{1}{512} + \frac{1}{1024} + \frac{1}{8192} + \frac{1}{16384} + \frac{1}{131072} + \frac{1}{262144} + \frac{1}{2097152} + \frac{1}{4194304} = 0{,}049999758601189$$

Абсолютная погрешность 0,000000241398811 ~ $2.4 \cdot 10^{-7}$

X_n		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
$_dX$					1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
1		1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
2		1	0	1	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0
3		1	1	0	1	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	0	0
4		1	1	1	0	1	1	1	0	0	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0
5	1	0	0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
6	1	0	0	1	0	0	0	0	1	0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	0	0	0
7	1	0	0	1	1	0	0	1	0	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	0	1	0	0
8	1	0	0	1	1	0	1	0	0	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0
9	1	0	1	1	0	1	1	1	0	0	0	1	0	0	1	1	0	0	1	1	0	0	1	0	1	1	0	0
10	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
11	1	1	0	0	0	0	0	0	1	0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	0	1	0	0
X_k	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

<u>Кол-во циклов</u>

$$n = \frac{\text{к.з.} - \text{н.з.}}{\text{шаг}} + 1 = \frac{0.75 - 0.25}{0.05} + 1 = 10 + 1 = 11$$

6. Параметр цикла вещественного типа

$y = e^{ax}$, $x \in [0{,}25; 0{,}75]$, $\Delta x = 0{,}05$

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main( )
{
    double a, y;
    cout << "a: ";  cin >> a;
 double xn = 0.25, xk=0.75, dx=0.05;

for(double x = xn; x < xk + dx/2; x += dx)
    {
      y = exp(a * x);
      cout << left << setw(7) << x << y << endl;
     }
  return 0;
}
```

```
a: 0.5
0.25   1.13315
0.3    1.16183
0.35   1.19125
0.4    1.2214
0.45   1.25232
0.5    1.28403
0.55   1.31653
0.6    1.34986
0.65   1.38403
0.7    1.41907
0.75   1.45499
```

7. Использование нескольких переменных управления циклом

Цикл for является одним из наиболее гибких операторов, т. к. допускает большое кол-во разнообразных вариантов.

Например, допустимо использовать несколько переменных управления.

Пример:

```
for (x = 0, y = 10;   x <= y;   ++x, --y)
cout << x << ' ' << y <<'\n';
```

8. Пропущенные секции в операторе for

Пример (отсутствует секция приращения):
Цикл, который должен выполняться до тех пор, пока с клавиатуры не будет введено число 123.

```
int   x;

for (x = 0;  x != 123;    )
{
   cout << "enter number: ";
   cin  >> x;
}
```

Пример (отсутствуют секции инициализации и приращения):

```
x = 0;

for (  ;  x  < 10;    )
{
   cout << x << ' ';
   x++;
}
```

Пример (отсутствуют все секции – бесконечный цикл):

```
for (   ;    ;    )
{
   // …
}
```

Например: кроты запасли в своей норке **S** штук зерен. С определенной периодичностью они обновляют запасы своих норок **ds1= a | sin (3t+2) |** и поедают их с величиной **ds2 = 5·10^3**. Хитрые же мыши с другой периодичностью иногда обворовывают, а иногда и возвращают награбленное у кротов на величину **ds3 = b (sin (5t-4))**. Запустить процесс заполнения/опустошения норки.

Для выхода из такого цикла необходимо в теле цикла использовать оператор break, размещенный внутри условного оператора if.

```
double s=2.3e20; int t=0; float a=35,b=52;
for (   ;    ;    )
{t++; ds1 = a*abs(sin(3*t)+2;  ds2=5e3 ; ds3 = b *(sin (5t-4));
s+=ds1-ds2- ds3;
If (s<=0) break;
}
```

Пример (отсутствует тело цикла):
(бестелесые циклы весьма полезны)

```
int i;
int sum = 0;

// суммирование чисел от 1 до 10
for ( i = 1  ; i <= 10  ;  sum += i++  ) ;  // цикл без тела цикла
```

Чтобы понять смысл оператора sum += i++ (это обычная запись для C++)
необходимо разобрать его на составные части:

```
sum = sum + i;
i = i + 1;
```

2.3 Табулирование функции счетным оператором

Написать программу печати таблицы значений функции

$$y = \begin{cases} t, & x < 0 \\ tx, & 0 \le x < 10 \\ 2t, & x \ge 10 \end{cases}$$

для аргумента, изменяющегося в заданных пределах с заданным шагом. Если $t > 100$, должны выводиться целые значения функции.

Исходными данными являются начальное значение аргумента Xn, конечное значение аргумента Xk, шаг изменения агрумента dX и параметр t. Все величины – вещественные. Программа должна выводить таблицу, состоящую из двух столбцов – значений аргумента и соответствующих им значений функции.

Словесный алгоритм задачи

В словесной форме алгоритм можно сформулировать так:

1. Ввести исходные данные.
2. Взять первое из значений аргумента.
3. Определить, какому из интервалов оно принадлежит.
4. Вычислить значение функции y по соответствующей формуле.
5. Если t > 100, преобразовать значение y в целое.
6. Вывести строку таблицы.
7. Перейти к следующему значению аргумента.
8. Если оно не превышает конечное значение, повторить шаги 3–7, иначе закончить выполнение.

В каждый момент времени требуется хранить одно значение функции, поэтому для него достаточно завести одну переменную вещественного типа. Шаги 3–7 повторяются многократно, поэтому для их выполнения надо организовать цикл.

Решение задачи

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
        float xn, xk, dx, t , y;
        printf("Enter xn, xk, dx,t \n");// \n -переход на нов
                                // строку = endl
        cin>>xn>>xk>>dx>>t;
        for (float x=xn;x<=xk;x+=dx)
        {
                if(x<0) y=t;
                else if (x>=0 && x<10) y=t*x;
                else y=2*t;
          if (t>100) printf("%9.2f   %9d    \n", x,(int)y);
          else printf("%9.2f   %9.2f    \n", x, y);
        }
        system("pause");
        return 0;
}
```

```
Enter xn, xk, dx,t
-1 12.1 0.5 102
    -1.00          102
    -0.50          102
     0.00            0
     0.50           51
     1.00          102
     1.50          153
     2.00          204
     2.50          255
     3.00          306
     3.50          357
     4.00          408
     4.50          459
     5.00          510
     5.50          561
     6.00          612
     6.50          663
     7.00          714
     7.50          765
     8.00          816
     8.50          867
     9.00          918
     9.50          969
    10.00          204
    10.50          204
    11.00          204
    11.50          204
    12.00          204
Для продолжения нажмите
```

Замечание о внутренних переменных

- Переменная x описана ВНУТРИ цикла, после его завершения, переменная x УДАЛЯЕТСЯ из памяти и мы не можем получить доступ к ее значению!!!
- Область видимости этой переменной – только тело оператора for.

В программу введена вспомогательная переменная x, которая последовательно принимает значения от Xn до Xk с шагом dX. Она определена непосредственно перед использованием. Это является хорошим стилем, поскольку снижает вероятность ошибок (например, таких, как использование неинициализированной переменной).

Операторы цикла while и do -while

применяются в тех случаях, когда известно

условие выполнения цикла,

а количество повторений может быть заранее не известно.

2. Оператор цикла с предусловием

2.1. Общий вид

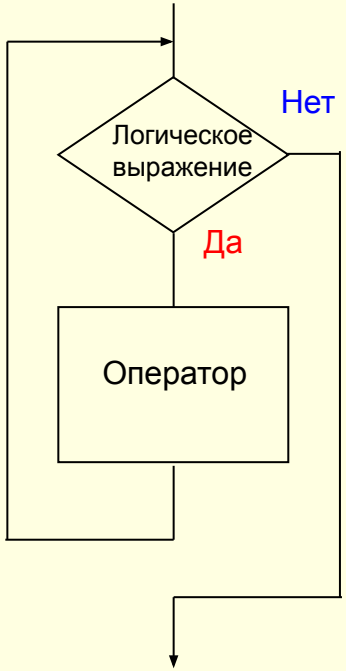
Пока

Условие выполнения цикла

while (<*логическое выражение*>)

<*оператор***>;** ← *тело цикла*

Любой оператор



- ***Пример 1***. Автомобиль движется со скоростью 5 км/ч и начинает наращивать скорость с ускорением 10 км/ч^2 до тех пор пока не будет достигнута скорость 60 км/час. Определить, за какое время эта скорость будет достигнута.

```
#include<iostream>
using namespace std;
int main()
{int speed = 5, time = 0, count=0;
    while ( speed < 60 )
    {
        cout << count <<"-speed = " << speed
<< " time= "<<time<<endl;
     speed += 10; // приращение скорости
          count++; // подсчёт повторений цикла
     time++;// наращивание времени
    }
   cout<< "final speed "<<speed<<" was reached
in "<< time<< " h"<<endl;
    system("pause");    return 0;}
```

Анализ программы

- *инициализация* трёх переменных (скорости *speed*, времени *time* и счётчика цикла *count* реализуется до начала цикла;
- *условие* **speed < 60** определяет возможность выполнения цикла, и пока скорость остаётся меньше 60, условие истинно и скорость будет нарщиваться;
- *управление условием* реализуется оператором **speed += 10**;
- *тело цикла* составляют операторы вывода на консоль и операторы приращения счётчика и времени.

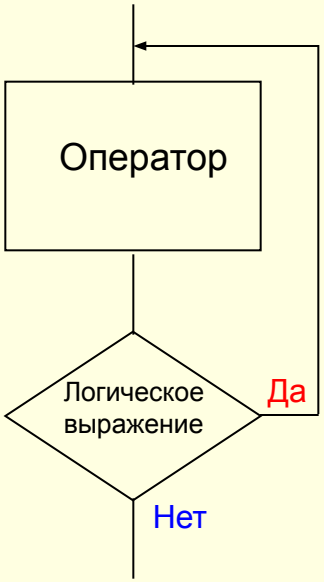
3. Оператор цикла с постусловием

3.1. Общий вид

Выполнять

Тело цикла

```
do
    <оператор> ;
while (<логическое выражение>);
```



До тех пор, пока

Условие повторения цикла

3.2. Технология программирования задачи с оператором цикла do-while

<u>Составить программу:</u>

Вычислить с заданной точностью сумму членов бесконечного ряда:

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots = \sum_{n=1}^{\infty} \frac{1}{n}$$

<u>Условие прекращения вычислений</u>

$S_n - S_{n-1} = \frac{1}{n} < \delta$ − малое число больше нуля.

<u>Выбор идентификаторов:</u>

Входной *Параметр цикла* *Выходной*

δ → d, n → n, S → s

$$S = \boxed{1} + \boxed{\frac{1}{2}} + \boxed{\frac{1}{3}} + \frac{1}{4} + \ldots = \sum_{n=1}^{\infty} \frac{1}{n}$$

$$S_n - S_{n-1} = \frac{1}{n} < \delta$$

```
int  main( )                    // Заголовок функции
{                               // Начало кода функции
    cout   <<   "d: ";
    double d;
    cin >> d;
    double s = 0;
    double n = 1;
    do
    {
        s +=  1 / n;
        n ++;
    }  while (1 / n > d);
    cout << "n = " << n << "   s = " << s << endl;
}
```

0,3

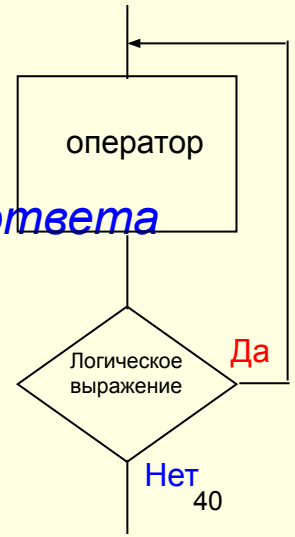
N	1/N	S
1	1	1.00
2	0.5	1.500
3	0.333	1.833
4	0.25	

0 + 1/1 + 1/2 + 1/3

1 + 1 + 1 + 1

// Вывод ответа

1.8333



Применение рекуррентных соотношений для программирования рядов

- Вычислить *сумму бесконечного ряда* с заданной точностью (суммировать до тех пор, пока модуль текущего элемента ряда не станет меньше заданной точности) двумя способами:
- 1) по общей формуле ряда;
- 2) по *рекуррентной формуле*.

$$\sum_{k=0}^{\infty}(-1)^k \frac{(k+1)!}{(2k+1)!};\ \text{точность}\ \varepsilon = 10^{-3} = 0{,}001.$$

Первый способ: применяем формулу общего члена

- Входные параметры:
- *eps* – точность вычислений ε
- *R* – значение первого элемента ряда при начальном значении *k* – номере 1-го элемента ряда
- Промежуточные параметры
- *P1* – интегратор для накопления произведения (*k*+1)!
- *P2* – интегратор для накопления произведения (2**k*+1)!
- *k* – номер итерации, совпадает с номером элемента ряда *R* – текущий (k-ый) элемент ряда.

- Выходные параметры
- *S* – сумма элементов ряда
- Определение начальных значений переменных
- Для переменной k начальное значение дано: *k* = 0 Подставим начальное значение *k* в формулу ряда

$$R_k = (-1)^k \frac{(k+1)!}{(2k+1)!}$$

- получим начальное значение переменной *R*:

$$R_0 = (-1)^0 \frac{(0+1)!}{(2\cdot 0+1)!} = 1$$

- Принимаем начальное значение сумматора *S* = 0, начальные значения интеграторов *P*1 = 1, *P*2 = 1.

- **Контрольный пример:**
- пусть ε=0,1:

- $k = 1$; $P1 = 1 \cdot 2$; $P2 = 1 \cdot 2 \cdot 3$;
  $R = -0.333 > \varepsilon$; $S = 0.667$;
- $k = 2$; $P1 = 1 \cdot 2 \cdot 3$; $P2 = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$;
  $R = +0.05 < \varepsilon$; $S = 0.687$;
- Выход из цикла.
- Ответ: S = 0.687

```
int main()
{int k=0,P1=1,P2=1;
```

```
Input accuracy eps:0.1
accuracy = 0.1
1 -0.333333 2 6

2 0.05 6 120
Sum = 0.666667 number of iterations = 2
Для продолжения нажмите любую клавишу . .
```

```
Input accuracy eps:0.001
accuracy = 0.001
Sum = 0.711905 number of iterations = 4
Для продолжения нажмите любую клавишу . .
```

```
}while(abs(R)>eps);
cout << "Sum = "<<s<< " number of iterations =
"<<k<<endl;
system("pause");
return 0;
}
```

```
Input accuracy eps:1e-5
accuracy = 1e-05
1 -0.333333 2 6
```

k R P_1 P_2

```
2 0.05 6 120

3 -0.0047619 24 5040

4 0.000330688 120 362880

5 -1.80375e-05 720 39916800

6 2.60862e-06 5040 1932053504
Sum = 0.712217 number of iterations = 6
Time spent = 0.006
Для продолжения нажмите любую клавишу .
```

```
Input accuracy eps:1e-6
accuracy = 1e-06
1 -0.333333 2 6

2 0.05 6 120

3 -0.0047619 24 5040

4 0.000330688 120 362880

5 -1.80375e-05 720 39916800

6 2.60862e-06 5040 1932053504

7 -2.01166e-05 40320 2004310016
```

k R P_1 P_2

```
8 -0.00125772 362880 -288522240
```

```
31 inf -2147483648 0

32 -inf -2147483648 0

33 -nan(ind) 0 0
Sum = -nan(ind) number of iterations = 33
Time spent = 0.055
Для продолжения нажмите любую клавишу . .
```

nan – not a number

```
15 -2.71498 2004189184 738197504

16 0.134354 -288522240 -2147483648

17 inf -898433024 0

18 inf 109641728 0
```

Анализ программы

- Анализ решения показывает, что программа работает верно, т.к. результаты программы совпадают с контрольным примером.

- Для проверки **полноты программы**, получено решение и для другого значения точности.
- Анализ программы позволяет выяснить **её неэффективность**:
- Возведение в степень -1 неоправданно при большом количестве итераций, т.к. результат всегда будет или +1 или -1.
- Накопление произведения для вычисления двух факториалов тоже неэффективно, поскольку при большом количестве итераций будет приводить к недопустимо большим числам в числителе и знаменателе.
- При вычислении *R* используются операторы приведения типа (от целого к вещественному) для того, чтобы, во-первых, избежать целочисленного деления, и, во-вторых, чтобы не было лишних предупреждений от транслятора.

Оценка времени выполнения процесса

Если нужно замерить время выполнения фрагмента кода на C++, используется clock() из модуля ctime — она *возвращает число таков*, измеряемое процессором от начала выполнения программы.

Глобальная константа CLOCKS_PER_SEC хранит число тактов, выполняемое процессором в секунду. Соответственно, чтобы получить время работы программы в секундах достаточно результат работы функции разделить на эту константу:

```
clock() / CLOCKS_PER_SEC;
```

Для определения времени работы фрагмента программы нужно определить моменты времени до фрагмента и после него, а затем — посчитать разницу.

```
#include <ctime>
//        ...
   unsigned int start_time =  clock(); // начальное время
   // здесь должен быть фрагмент кода, время выполнения которого нужно измерить
   unsigned int end_time = clock(); // конечное время
   unsigned int search_time = (end_time - start_time)/CLOCKS_PER_SEC; // искомое время
```

Код с операторами оценки времени

Переменная begin класса clock_t

```
clock_t begin = clock();
/* here, do your time-consuming job */
do
{
s += R;
k = k + 1;
P1 *= (k + 1);
P2 *= (2 * k)*(2 * k + 1);
R = pow(-1., k)*(float)P1 / float(P2);
cout <<  k << ' ' << R << ' ' << P1 << ' ' << P2 << endl;
} while (abs(R) > eps);
cout << "Sum = " << s << " number of iterations = " << k << endl

clock_t end = clock();
double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
cout << "Time spent = " << time_spent << endl;
```

Второй способ

- Для начала *выведем* *рекуррентную формулу*, т. е. соотношение, которое позволяет получать любой *k*-й член через (*k*-1)-й:

$$R_k = Q \cdot R_{k-1}$$

- Вычислим значение коэффициента Q для нашего ряда:

$$Q = \frac{R_k}{R_{k-1}} = \frac{(-1)^k}{(-1)^{k-1}} \cdot \frac{(k+1)!}{k!} \cdot \frac{(2k-1)!}{(2k+1)!} = -\frac{k!(k+1)}{k!} \cdot \frac{(2k-1)!}{(2k-1)! \cdot 2k \cdot (2k+1)} =$$

$$= -\frac{k+1}{2k \cdot (2k+1)}.$$

ТАКОЙ СПОСОБ НЕОБХОДИМ ТОЛЬКО В СЛУЧАЯХ <u>НЕОЧЕВИДНОЙ</u> СВЯЗИ МЕЖДУ n-м И (n-1)-м членами. ОБЫЧНО РЕКУРРЕНТНЫЕ ОТНОШЕНИЯ НАХОДЯТСЯ БОЛЕЕ ПРОСТЫМ <u>МЕТОДОМ ПОДБОРА</u>

```
Input accuracy eps:0.1
accuracy = 0.1
 k=1  -0.333333  -0.333333

 k=2  -0.15  0.05

Sum = 0.666667 number of iterations = 2
Для продолжения нажмите любую клавишу . . .
```

```
cout<< "accuracy = "<<eps;
do
{s+=R;
k=k+1;
Q=-float(k+1)/float((2*k)*(2*k+1));
R*=Q;
cout<<"\n k="<<k<<"   " <<Q<< "   "<<R<<endl;
}while(abs(R)>eps);
cout << "\nSum = "<<s<< " number of iterations
= "<<k<<endl;
system("pause");
return 0;
}
```

```
Input accuracy eps:1e-5
accuracy = 1e-05
 k=1  -0.333333  -0.333333

 k=2  -0.15  0.05

 k=3  -0.0952381  -0.00476191

 k=4  -0.0694444  0.000330688

 k=5  -0.0545455  -1.80375e-05

 k=6  -0.0448718  8.09376e-07

Sum = 0.712217 number of iterations = 6
Time spent = 0.005
Для продолжения нажмите любую клавишу .
```

```
Input accuracy eps:1e-7
accuracy = 1e-07
 k=1  -0.333333  -0.333333

 k=2  -0.15  0.05

 k=3  -0.0952381  -0.00476191

 k=4  -0.0694444  0.000330688

 k=5  -0.0545455  -1.80375e-05

 k=6  -0.0448718  8.09376e-07

 k=7  -0.0380952  -3.08334e-08

Sum = 0.712218 number of iterations = 7
Time spent = 0.007
Для продолжения нажмите любую клавишу . . .
```

Анализ программы

- Преимущества этого метода очевидны: нет необходимости в неоправданно больших умножениях, бессмысленном возведении в степень (-1), этот алгоритм требует меньше процессорного времени.

Генерация псевдослучайных чисел средствами языка C++

Функции работы со случайными числами

- *Случайные числа на языке программирования* C^{++} могут быть сгенерированы функцией rand() из стандартной библиотеки **cstdlib.**
- Функция **rand()** генерирует числа в диапазоне от **0** до **RAND_MAX**.
- **RAND_MAX** — ***это константа***, определённая в библиотеке <cstdlib>.
- Для Microsoft Visual Studio: **RAND_MAX = 32767,** но оно может быть и больше, в зависимости от компилятора.

Функции работы со случайными числами

- Чтобы функция **rand()** всегда возвращала ***разные числа***, её нужно использовать в паре с функцией **srand(unsigned int арг).** Аргумент **арг** задаёт то стартовое число, на базе которого и создаётся база случайных чисел.
- Для того, чтобы автоматически всегда брать разные стартовые числа, рекомендуется использовать функцию
  (unsigned int) time(NULL)
  которая возвращает ***число секунд, прошедших с 00:00:00 UTC 1 января 1970 года*** (Unix-время равно нулю).

Особенности работы функции srand()

- Чаще всего в качестве передаваемой величины в функцию **srand()** используют системное время в секундах, т.к. это число будет всегда разным, а соответственно, мы будем получать на выходе из **rand()** разные случайные числа.
- Чтобы передать в функцию srand() текущее системное время, можно использовать библиотечную функцию **time(),** описанную в библиотеке <**ctime**>.
- Для того, чтобы эта функция возвращала текущее время в секундах (секунды отсчитываются от 00:00:00), нужно вызывать ее с параметром **NULL**: srand(time(NULL)); или **srand(time(0));.**
- Если нет необходимости в формировании всегда разных случайных чисел, то функцию **srand** можно задать с любой константой или вовсе не включать её в программу.

Пример 1. Инициализация массива случайными числами в заданном диапазоне значений: от -10 до 10.

tf[i] = rand( ) % (r_max - r_min+1) + r_min;

```
d:\КГЭУ\Lectures\Программирование\II семестр\Лаб_раб\2018_осень\simple_for\Debug\simple...
Inpiut min  and max limit values:-10 10
        -4      -7      8       -9      1       -8      5       4       -5
-8
Для продолжения нажмите любую клавишу . . .
```

```
Inpiut min  and max limit values:-10 11
 -4 5 -8 -3 8 -4 9 3 2 1
Для продолжения нажмите любую клавишу . . .
```

```
#include <iostream>
#include <ctime>
using namespace std;

// функция инициализации массива случайными числами
int main()
{
int tf [10], nf, r_min, r_max;
nf = 10;
cout<< “Inpiut min  and max limit values:”; cin>>r_min>>r_max;
   srand( (unsigned int) time( NULL ) );   // рандомизация генератора
   for (int  i = 0;   i < nf;   i++ )  // n - количество чисел
     tf[i] = rand( ) % (r_max - r_min+1)  +  r_min;  // формирование
случайного числа
for (int  i = 0;
return 0;
}
```

Создание базы случайных чисел на основе Time (NULL)

Функция ***rand*** генерирует случайные числа, возвращает псевдослучайное целое число в диапазоне от 0 до ***RAND_MAX***.

RAND_MAX это константа, определенная в <cstdlib>. По умолчанию её значение может изменяться, в зависимости от реализации, но, как правило, макрос RAND_MAX меньше значения 32767 не бывает.

Пример 1. Определить количество цифр в числе *N*, заданным случайным образом.

```
#include<iostream>
#include<cmath>
#include<ctime>
using namespace std;
int main()
{int Number,M,N, count;
srand(10);
cout<< " What is the maximal value of the
Number?";
cin>>M;
N=rand()%M;
Number=N;
```

```
// Метод - цикл с делением
count = (Number == 0) ? 1 : 0;
        while (Number != 0)
    {
            count++;
            Number /= 10;
        }
cout<<"\nCounts of digits in the number
"<<N<<" is equal "<<count<<endl;
```

```
// Метод - десятичный логарифм и округление
// хорош для очень больших чисел.
N=rand()%M;
Number=N;
count=(Number == 0) ? 1 :
(int)ceil(log10(abs(Number) + 0.5));
cout<<"\nCounts of digits in the number "<<N<<"
is equal "<<count<<endl;
system("pause");
return 0;
}
```

```
What is the maximal value of the Number?10000

Counts of digits in the number 71 is equal 2

Counts of digits in the number 6899 is equal 4
Для продолжения нажмите любую клавишу . . .
```

Пример 2. Паук находится на плоскости в точке с координатами x=50 и y=50. Каждую секунду он делает шаг влево, вправо, вниз или вверх с равной вероятностью. Смоделируйте движение паука с помощью генератора случайных чисел.

Координаты x(t) и y(t) сохраните в одномерных массивах. Напечатайте траекторию паука в виде таблицы, которая содержит в клеточке с координатами x и y символ “.”, если там паук не был, “+”, если паук там побывал 1 раз, “*”- для двух раз, “#” - для трёх и символ “@” -, если он побывал больше раз.

Блок инициализации

```
#include<iostream>
#include<ctime>
#include<iomanip>
using namespace std;
int main()
{
cout << " Случайное блужданиепо плоскости." << endl;
const int N=61, K=1550, J=1;
char buf[N+1]; ];// массив символов ~ количество посещени
int xy[N][N];// массив для хранения количества посещений
клетки N,N
int x=N/2;// ставим курсор в середину консоли
int y=N/2;
```

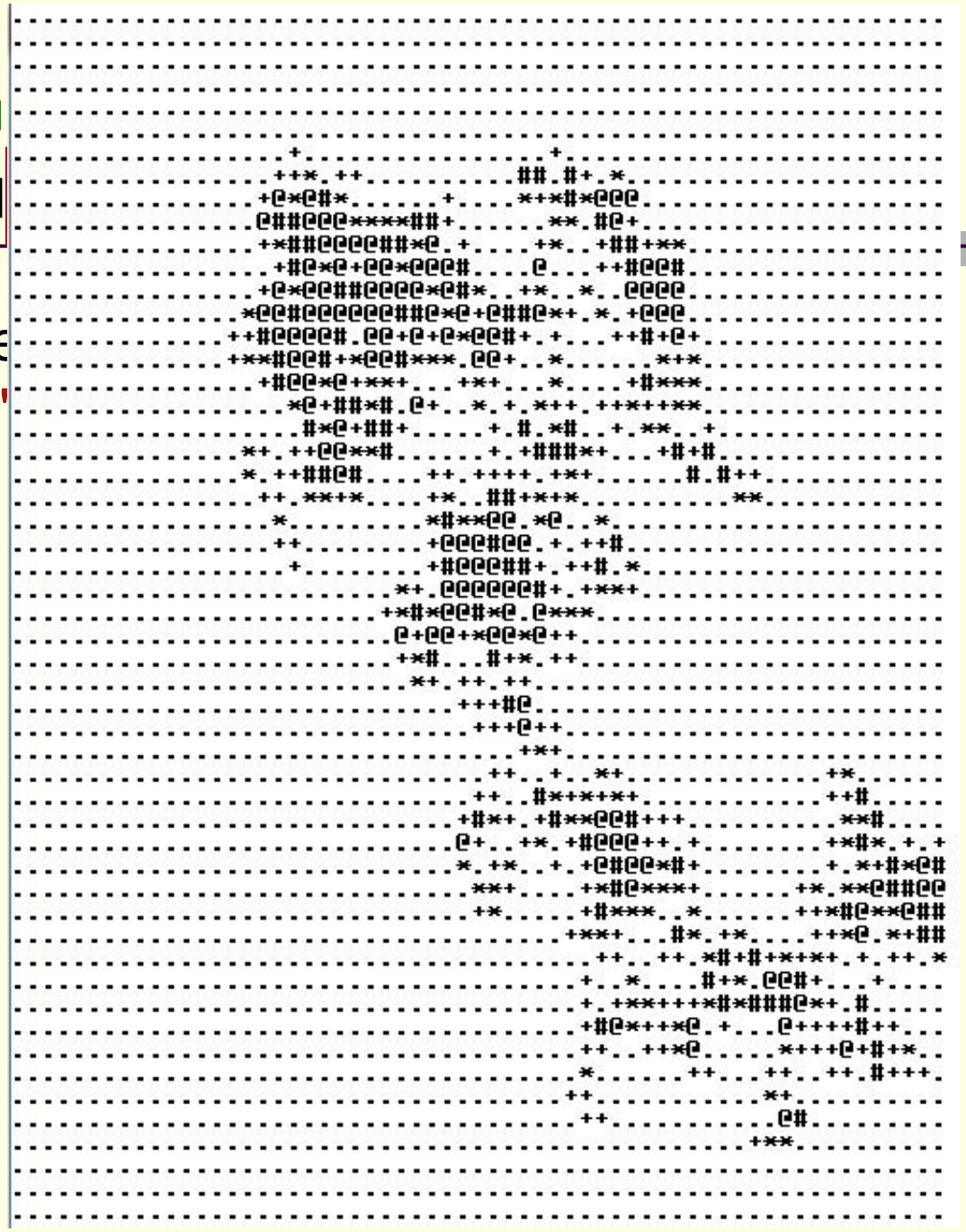
// Заполнение двумерного массива значениями при
// моделировании движения паука

```
for(int m=0; m<N; m++)
        for(int n=0; n<N; n++)
        xy[m][n]=0;
    xy[x][y]=1;
    // запись координат движения паука в массив
    for(int k=1; k<K; k++)
    {
        for(int j=0; j<J; j++)
      x+=rand()%3-1;//формирование сл.чисел от -1 до 1
        for(int j=0; j<J; j++)
          y+=rand()%3-1;
```

```
//проверка выхода к границам
    if(x<0)x=0;
    if(x>N-1)x=N-1;
    if(y<0)y=0;
    if(y>N-1)y=N-1;
    xy[x][y]+=1; // отметка о посещении точки в массиве
    }
```

```
// Заполнение символьного массива
  for(int m=0; m<N; m++)
  {
      for(int n=0; n<N; n++)
          switch(xy[m][n])
          {
          case 0: buf[n]='.'; break;
          case 1: buf[n]='+'; break;
          case 2: buf[n]='*'; break;
          case 3: buf[n]='#'; break;
          default: buf[n]='@';
          }
      buf[N]='\0';// ставим конец строки
```

```
// Выводим
        cou
        }
    cout<< e
    system("
return 0;
}
```



Краткие итоги

- **Оператор for на** C++ состоит из четырёх секций: инициализации, условия, тела цикла, приращение. Любая из секций может быть опущена с соблюдением синтаксиса, позволяя гибко строить алгоритм, используя этот компактный и многофункциональный оператор.
- **Условные операторы while и do-while** по эффективности эквиваленты оператору **for,** удобны при описании условных алгоритмов.
- **Оператор for** необходим при работе с элементами массивов, при вычислении сумм конечных рядов, при любых алгоритмах, где используется счётчик при заранее известном количестве итераций.
- **При алгоритмизации задач с бесконечными рядами** можно использовать любые типы циклов, однако **while** и **do-while** многими полагаются как более наглядные.
- **Рекуррентные формулы** при вычислении сумм длинных рядов не только позволяют избегать операций прерывания, но и существенно экономят процессорное время.
- **Для оценки времени** удобно пользоваться функций **clock.**
- **Формула для формирования числа в заданном диапазоне значений**

```
tf[i] = rand( ) % (r_max - r_min+1)  +  r_min;
```

Контрольные задания. Ответы обосновать.

1. Сколько итераций сделает данный цикл? :

for(int k = 9, s = -3; k > s; k /= 1.5; s *= -1,5);

Можно ли узнать значение **s** после завершения цикла?

2. Чему будет равно значение **c** после выполнения операторов:

c=044; while(~(0x7|0xc)^c) c << 2; cout << c << endl;

3. Проанализируйте фрагмент программы и напишите, что будет выведено на консоль:

int k=0,z=0xCAF; for(int m=5,k=2;m>2; m--)

{z << k; k++; cout << z << '\t' << k << endl;} cout << k;

4. Какой результат выведет этот оператор?

cout << sizeof(2.*35/7e0) << endl;

5. В каком диапазоне значений будет сформировано число по оператору **x+=rand()%5 - 2; ?**