

Основы алгоритмизации и программирования

Лекция 7

Составление циклических алгоритмов

Понятие циклического кода

Цикл – это одно из фундаментальных понятий программирования. Под **циклом** понимается **организованное повторение** некоторой **последовательности операторов**

Любой **цикл** состоит из **кода цикла**, т.е. тех **операторов**, которые выполняются **несколько раз**, **начальных установок**, **модификации параметра цикла** и **проверки условия** продолжения выполнения цикла

Один проход цикла называется **шагом** или **итерацией**. Проверка условия продолжения цикла происходит на каждой итерации либо до выполнения кода цикла (**с предусловием**), либо после выполнения (**с постусловием**)

Разновидности операторов цикла языка Си

оператор цикла с
предусловием

оператор цикла с
постусловием

оператор цикла с
предусловием и коррекцией

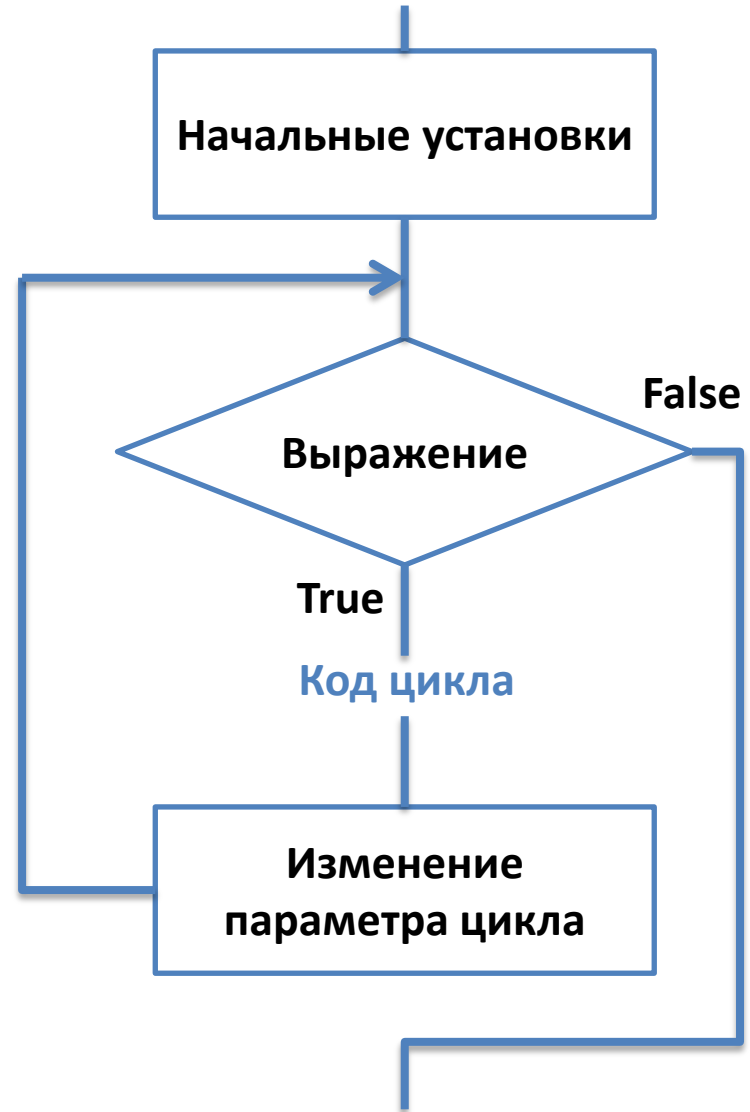
Оператор с предусловием while

while (*выражение*)
код цикла;

Выражение определяет условие повторения кода цикла, представленного простым или составным оператором

Код цикла может включать любое количество операторов, связанных с конструкцией **while**, которые нужно заключить в **фигурные скобки** (организовать блок), если их более одного

Переменные, изменяющиеся в коде цикла и используемые при проверке условия продолжения, **называются параметрами цикла**. Целочисленные параметры цикла, изменяющиеся с постоянным шагом на каждой итерации, называются **счетчиками цикла**



Оператор с предусловием **while**

Начальные установки могут явно не присутствовать в программе, их смысл состоит в том, чтобы до входа в цикл задать значения переменным, которые в этом цикле используются

Цикл завершается, если условие его продолжения не выполняется. Возможно принудительное завершение как текущей итерации, так и цикла в целом. Для этого используют оператор ***continue*** – переход к следующей итерации цикла и ***break*** – выход из цикла.

Пример

Организация выхода из бесконечного цикла по нажатию клавиши **Esc**:

```
while (1) {    // Бесконечный цикл
    if (kbhit() && getch()==27 ) break;
}
```

Функция ***kbhit()*** возвращает значение **> 0**, если нажата любая клавиша, а функция ***getch()*** возвращает код нажатой клавиши.

Пример

Организации паузы в работе программы с помощью цикла, выполняющегося до тех пор, пока не нажата любая клавиша

...

```
while (!kbhit());
```

...

Оператор цикла с постусловием do – while

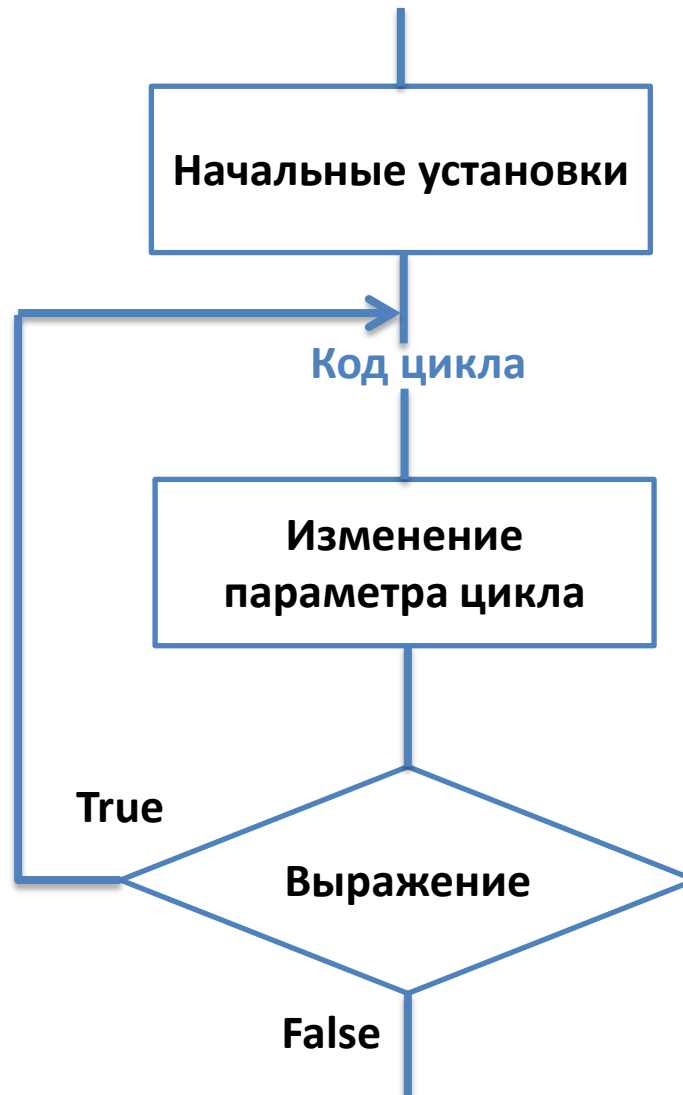
Общий вид записи

```
do  
код цикла;  
while (выражение);
```

Код цикла будет выполняться до тех пор, пока **выражение** истинно. Данный цикл всегда **выполняется хотя бы один раз**, даже если изначально выражение ложно. Здесь сначала выполняется код цикла, после чего проверяется, надо ли его выполнять еще раз.

Пример

```
char answer;  
do {  
    puts(" Hello! => ");  
    scanf(" %c ", &answer);  
}  
while ((answer=='y') || (answer=='Y'));
```



Оператор цикла с предусловием и коррекцией for

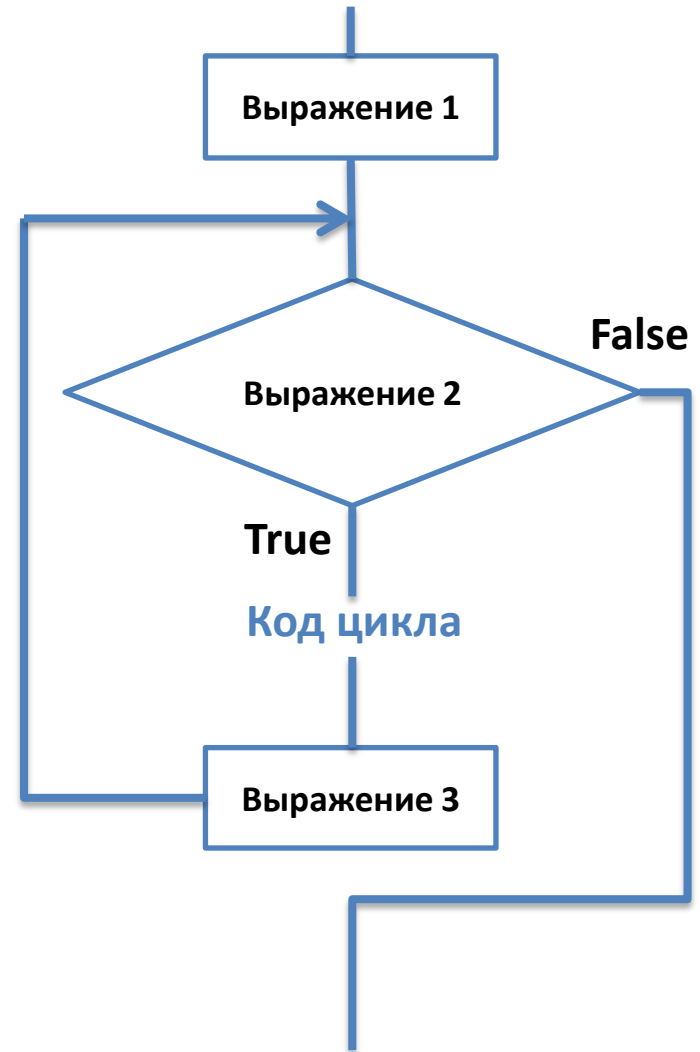
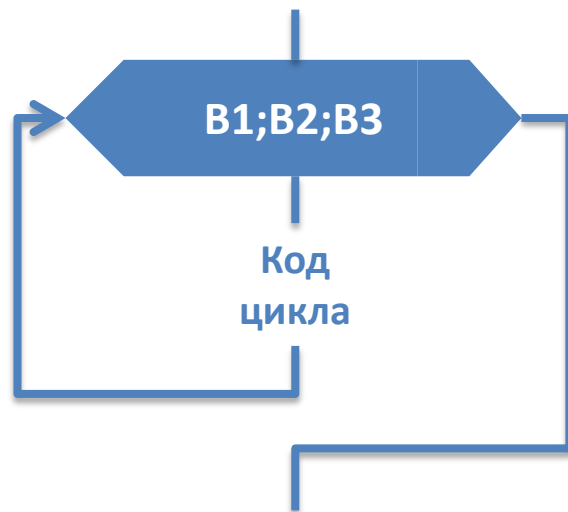
Общий вид оператора

```
for (выражение 1; выражение 2; выражение 3)  
код цикла;
```

выражение 1 – инициализация счетчика
(параметр цикла)

выражение 2 – условие продолжения счета

выражение 3 – коррекция счетчика



Оператор цикла с предусловием и коррекцией for

Инициализация используется для присвоения счетчику (параметру цикла) **начального значения**.

Выражение 2 определяет условие выполнения цикла. Как и в предыдущих случаях, если его результат не нулевой («истина»), – то цикл **выполняется, иначе** – происходит **выход из цикла**.

Коррекция выполняется после каждой итерации цикла и служит для изменения параметра цикла.

Выражения 1, 2 и 3 могут отсутствовать (пустые выражения), но символы «**;**» **опускать нельзя**.



Пример

Суммирование первых N натуральных чисел:

```
sum = 0;
```

```
for ( i = 1; i<=N; i++) sum+=i;
```

В **выражении 1** переменную-счетчик можно **декларировать**.

```
for (int i = 1; i<=N; i++)
```

Областью действия такой переменной будет код цикла.

Оператор цикла с предусловием и коррекцией **for**

Если пропущено **выражение 2**, то цикл будет выполняться бесконечно, поскольку пустое условие всегда остается истинным. Бесконечный оператор:

for (; ;) код цикла; эквивалентен оператору **while (1) код цикла;**

В заголовке оператора **for** может использоваться операция «запятая». Она позволяет включать в его выражения несколько операторов.

Пример

Суммирование первых **N** натуральных чисел
for (sum = 0 , i = 1; i<=N; sum+= i , i++) ;

Оператор **for**
имеет
следующие
возможности

Можно вести подсчет с помощью символов, а не только чисел:
for (ch = 'a'; ch <= 'z'; ch++) ... ;

Можно проверить выполнение некоторого произвольного условия:
for (n = 0; s[i] >= '0' && s[i] < '9'; i++) ... ;
или
for (n = 1; n*n*n <= 216; n++) ... ;

Первое выражение необязательно должно инициализировать переменную. Необходимо только помнить, что первое выражение вычисляется только **один раз**, перед тем как остальные части начнут выполняться.

Оператор цикла с предусловием и коррекцией for

Переменные, входящие в **выражения 2** и **3**, можно изменять при выполнении кода цикла

```
for (n = 1; n < 10*k; n += delta) ... ;
```

Использование **условных выражений** позволяет во многих случаях значительно **упростить программу**, например:

```
for (i = 0; i < n; i++)  
    printf("%6d%c", a[i], (i%10==0) || (i==n-1) ? '\n' : ' ');
```

В этом цикле печатаются n элементов массива a по 10 в строке, разделяя каждый столбец одним пробелом и заканчивая каждую строку (включая последнюю) одним символом перевода строки. Символ перевода строки записывается после каждого десятого и n -го элементов. За всеми остальными – пробел.

Оператор цикла с предусловием и коррекцией for

Наиболее часто встречающиеся *ошибки* при создании циклов – это использование в коде цикла неинициализированных переменных и неверная запись условия выхода из цикла

Чтобы избежать ошибок, нужно



проверить, всем ли переменным, встречающимся в правой части операторов присваивания в коде цикла, присвоены до этого начальные значения (а также возможно ли выполнение других операторов)

проверить, изменяется ли в цикле хотя бы одна переменная, входящая в условие выхода из цикла

предусмотреть аварийный выход из цикла по достижении некоторого количества итераций

если в состав цикла входит не один, а несколько операторов, нужно заключать их в фигурные скобки

Операторы и функции передачи управления



Оператор безусловного перехода *goto*

goto метка ;

Он предназначен для **передачи управления оператору**, помеченному указанной **меткой**. Метка представляет собой **идентификатор**, оформленный по всем правилам идентификации переменных с символом «**двоеточие**» после него, например, пустой помеченный меткой ***m1*** оператор:

m1: ;

Область действия метки – функция, где эта метка определена. В случае необходимости можно использовать блок.

Операторы и функции передачи управления

Циклы и переключатели можно вкладывать друг в друга и наиболее характерный оправданный случай использования оператора **goto** – выполнение прерывания (организация выхода) во вложенной структуре. Например, при возникновении грубых неисправимых ошибок необходимо выйти из двух (или более) вложенных структур (где нельзя использовать непосредственно оператор **break**, т.к. он прерывает только самый внутренний цикл)

Пример

```
for (...)  
    for (...) {  
...  
        if (ошибка) goto error;  
    }  
...  
error: операторы для устранения ошибки;
```

Оператор **goto** можно использовать для организации переходов из нескольких мест функции в одно, например, когда перед завершением работы функции необходимо сделать одну и ту же операцию

Операторы и функции передачи управления

Операторы *continue*, *break* и *return*

Оператор ***continue*** может использоваться во всех типах циклов (но не в операторе-переключателе ***switch***). Наличие оператора ***continue*** вызывает пропуск «оставшейся» части итерации и переход к началу следующей, т.е. досрочное завершение текущего шага и переход к следующему шагу

В циклах ***while*** и ***do-while*** это означает непосредственный переход к проверочной части. В цикле ***for*** управление передается на шаг коррекции, т.е. модификации ***выражения 3***.

Оператор ***continue*** часто используется, когда последующая часть цикла оказывается слишком сложной, так что рассмотрение условия, обратного проверяемому, приводит к слишком высокому уровню вложенности программы.

Операторы и функции передачи управления

Операторы `continue`, `break` и `return`

Оператор ***break*** производит досрочный выход из цикла или оператора-переключателя ***switch***, к которому он принадлежит, и передает управление первому оператору, следующему за текущим оператором. То есть ***break*** обеспечивает переход в точку кода программы, находящуюся за оператором, внутри которого он (***break***) находится

Оператор ***return*** производит **досрочный выход** из текущей функции.

Он также возвращает значение результата функции:

return *выражение*;

Выражение должно иметь скалярный тип

Операторы и функции передачи управления

Функции `exit` и `abort`

Функция **`exit`** выполняет **прерывание программы** и используется для нормального, корректного **завершения работы** программы при возникновении какой-либо **внештатной ситуации**, например, ошибка при открытии файла. При этом записываются все буферы в соответствующие файлы, закрываются все потоки и вызываются все зарегистрированные стандартные функции завершения.

Прототип этой функции приведен в заголовочном файле **`stdlib.h`** и выглядит так:

`void exit (int exit_code);`

Параметр данной функции – ненулевое целое число, передаваемое системе программирования (служебное сообщение о возникшей внештатной ситуации).

Для завершения работы программы также может использоваться функция

`void abort (void);`

действия которой аналогичны функции **`exit(3)`**

Рекомендации по программированию

Выражение, стоящее в круглых скобках операторов *if*, *while* и *do-while*, вычисляется по правилам стандартных приоритетов операций

Если в какой-либо ветви вычислений условного оператора или в цикле требуется выполнить **два (и более) оператора**, то они при помощи фигурных скобок объединяются в **блок**

Проверка вещественных величин на **равенство**, как правило, из-за ограниченной разрядности **дает неверный результат**

Чтобы получить **максимальную читаемость** и **простоту структуры программы**, надо правильно **выбирать способ** реализации **ветвлений** (с помощью *if*, *switch*, или **условных операций**), а также наиболее подходящий оператор цикла

Выражение в операторе *switch* и константные выражения в *case* должны быть **целочисленного** или **символьного** типов

Рекомендуется использовать в операторе *switch* ветвь **default**

После каждой ветви для передачи управления на точку кода за оператором *switch* используется оператор **break**

При построении любого цикла надо не забывать тот факт, что в нем всегда явно или неявно присутствуют четыре основных элемента: **начальные установки**, **код цикла**, **модификация параметра цикла** и **проверка условия** на продолжение цикла

Если количество повторений цикла **заранее не известно** (реализуется **итерационный процесс**), необходимо предусмотреть **аварийное завершение** цикла при получении достаточно **большого** количества итераций

При использовании **бесконечного цикла** обязательно необходима организация **выхода из цикла по условию**