

Составление циклических алгоритмов

Цикл – это повторение некоторой последовательности действий.

Любой цикл обычно состоит из:

- **кода**, т.е. тех инструкций, которые выполняются несколько раз;
- начальных установок;
- изменения параметра цикла;
- проверки условия продолжения цикла.

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

Для организации циклов используются операторы:

- 1) цикл с предусловием **while**;
- 2) цикл с постусловием **do – while**;
- 3) цикл с предусловием и коррекцией **for**.

1. Цикл с предусловием имеет вид:

while (*Выражение*) - *Заголовок цикла*
Код цикла

Выражение определяет условие повторения *Кода цикла*, который может быть простым или составным (*Блок*).

Если *Выражение* **Истинно** (не равно **0**), то выполняется *Код цикла*, иначе (*Выражение Ложно*) – происходит выход из цикла и выполняется оператор, следующий за **while**. Если *Выражение* изначально ложно, то цикл не выполнится ни разу.

Код цикла может содержать любое количество операторов, если их более одного создается **Блок**.

Переменные, изменяющиеся в цикле и используемые при проверке условия продолжения, называются **параметрами цикла**.

Целочисленные параметры цикла, изменяющиеся с постоянным шагом, называются **счетчиками**.

Начальные установки в программе могут явно не присутствовать, но их смысл в том, что до входа в цикл переменным, которые в нем используются, должны быть заданы начальные значения.

Цикл завершается, если **Выражение Ложно**, но можно принудительно закончить, как текущий шаг, так и цикл в целом. Для этого используют операторы: **continue** (*продолжить*) – переход к следующему шагу, **break** (*прекратить*) – выход из цикла.

Полезные примеры

1. Распространенный прием для организации выхода из бесконечного цикла по нажатию клавиши *Esc*

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

Функция ***kbhit*** имеет значение **> 0**, если нажата любая клавиша, ***getch*** - код нажатой клавиши (код ***Esc*** - **27**). При выполнении оператора ***if*** , если будет нажата клавиша ***Esc***, оператор ***break*** завершит цикл.

2. Организация паузы в работе программы с помощью цикла, выполняющегося до тех пор, пока не нажата любая клавиша:

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

2. Цикл с постусловием:

do

Код цикла

while (*Выражение*) ;

Код цикла будет выполняться до тех пор, пока **Выражение Истинно**.

Все, что говорилось для **while** аналогично, за исключением того, что **do-while** всегда выполняется хотя бы один раз, даже если изначально **Выражение Ложно**, т.е. сначала выполняется код цикла, а потом проверяется, надо ли его выполнять еще раз.

Следующий пример дает возможность создать в **Консольном приложении** процесс в стиле **Оконного**, т.е. выполнять программу до тех пор, пока будете вводить символ **Y** или **y** (**Yes**). Введя любой другой символ, цикл завершит свою работу и работу всей программы.

```
void main ()  
{  
char kod;
```

```
do {
```

```
    Ввод исходных значений,  
    ... все вычисления,  
    вывод результатов
```

```
    cout << " Repeat – ( y, Y ), Else – EXIT “ << endl;
```

Повторить – (y, Y), Иначе - Выход

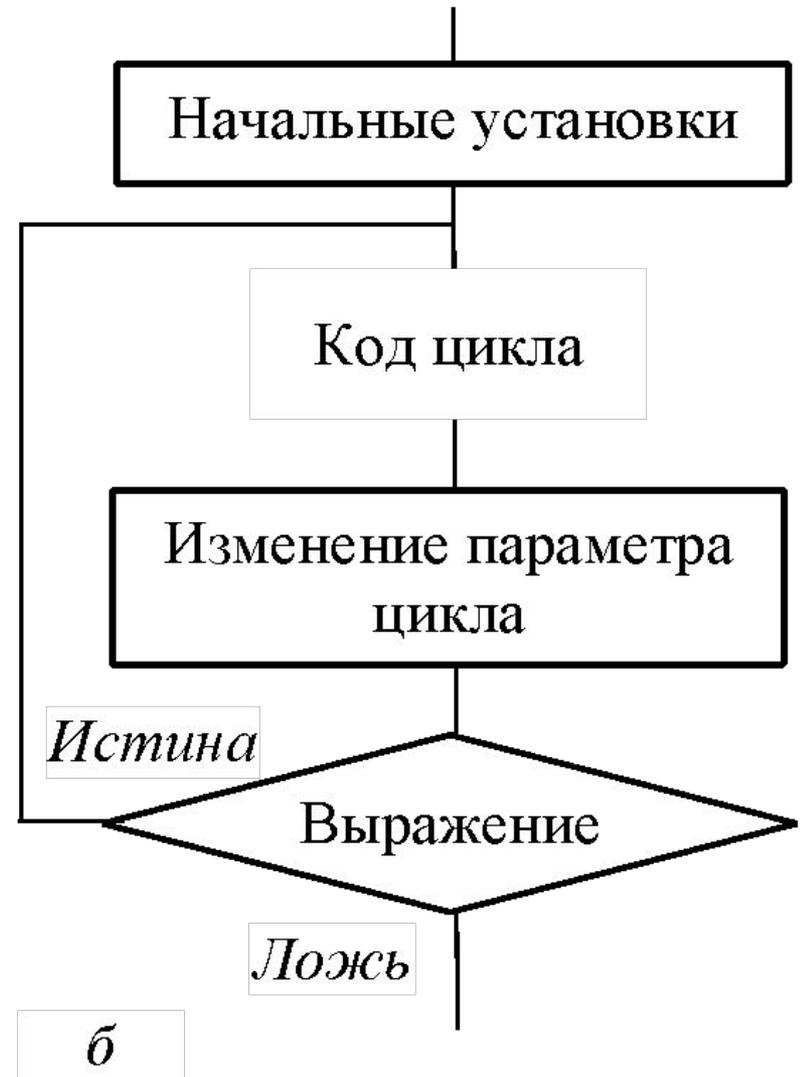
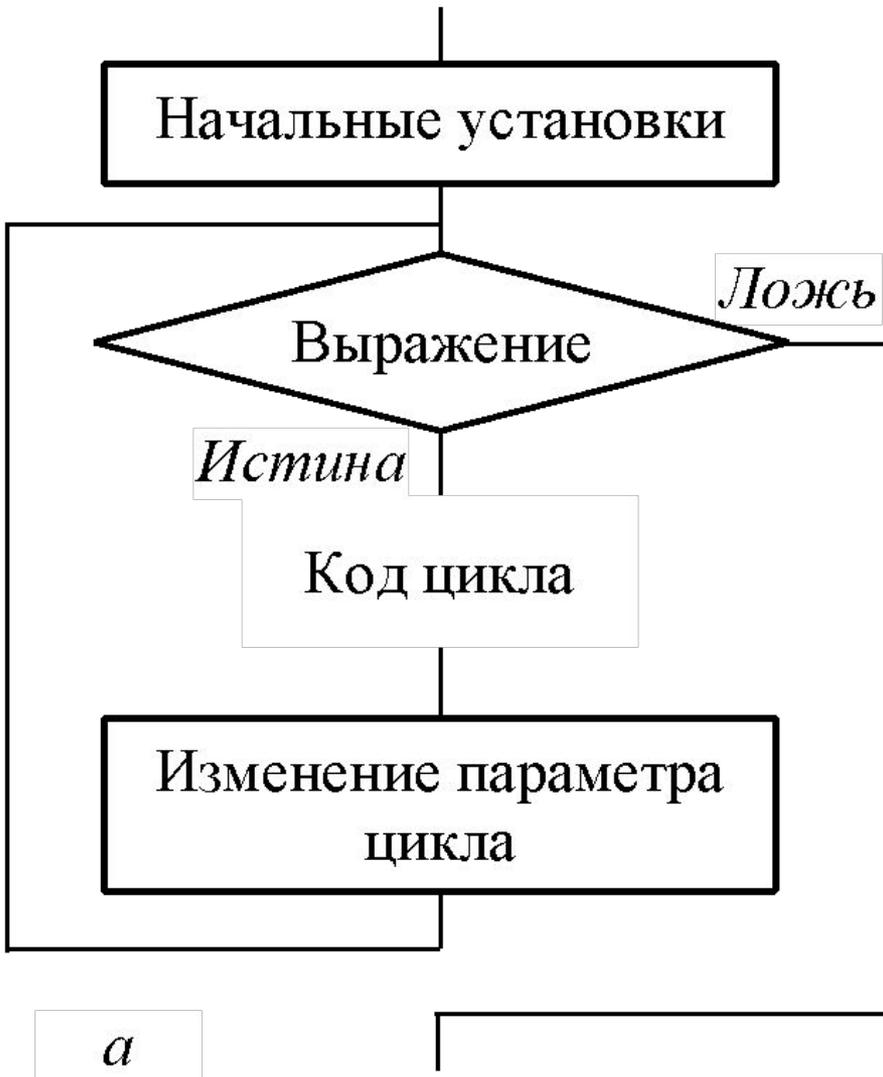
```
    cin >> kod;
```

```
}
```

```
while ( ( kod == 'y' ) || ( kod == 'Y' ) );
```

```
}
```

Обратите внимание, в данном случае ***getch*** не нужен!



Схемы операторов: а) **while**, б) **do-while**

3. Оператор с предусловием и коррекцией **for** (цикл с параметром):

for (*Выражение_1*; *Выражение_2*; *Выражение_3*)

Код цикла

Выражение_1 – инициализация параметра цикла;

Выражение_2 – условие продолжения цикла;

Выражение_3 – коррекция параметра.

В **Выражении 1** обычно параметру цикла присваивается начальное значение.

Выражение_1 не обязательно должно инициализировать переменную, но необходимо помнить, что **ОНО** вычисляется вначале **только один раз**.

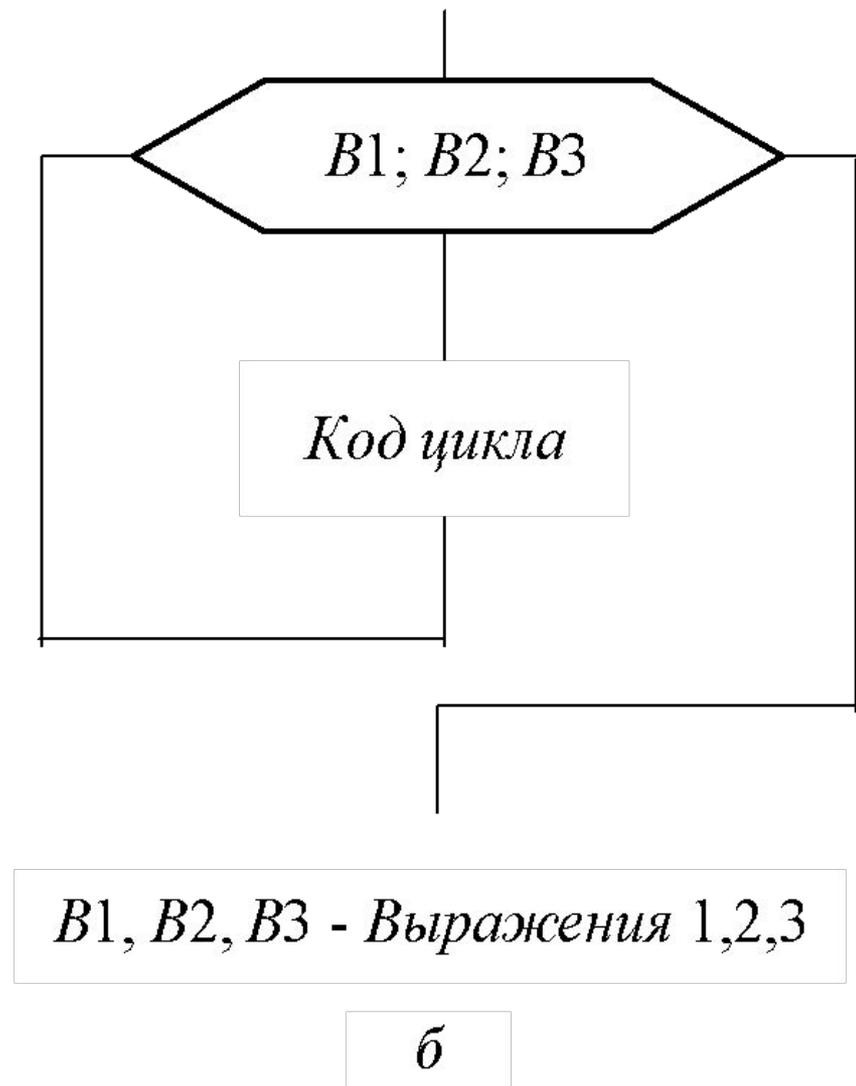
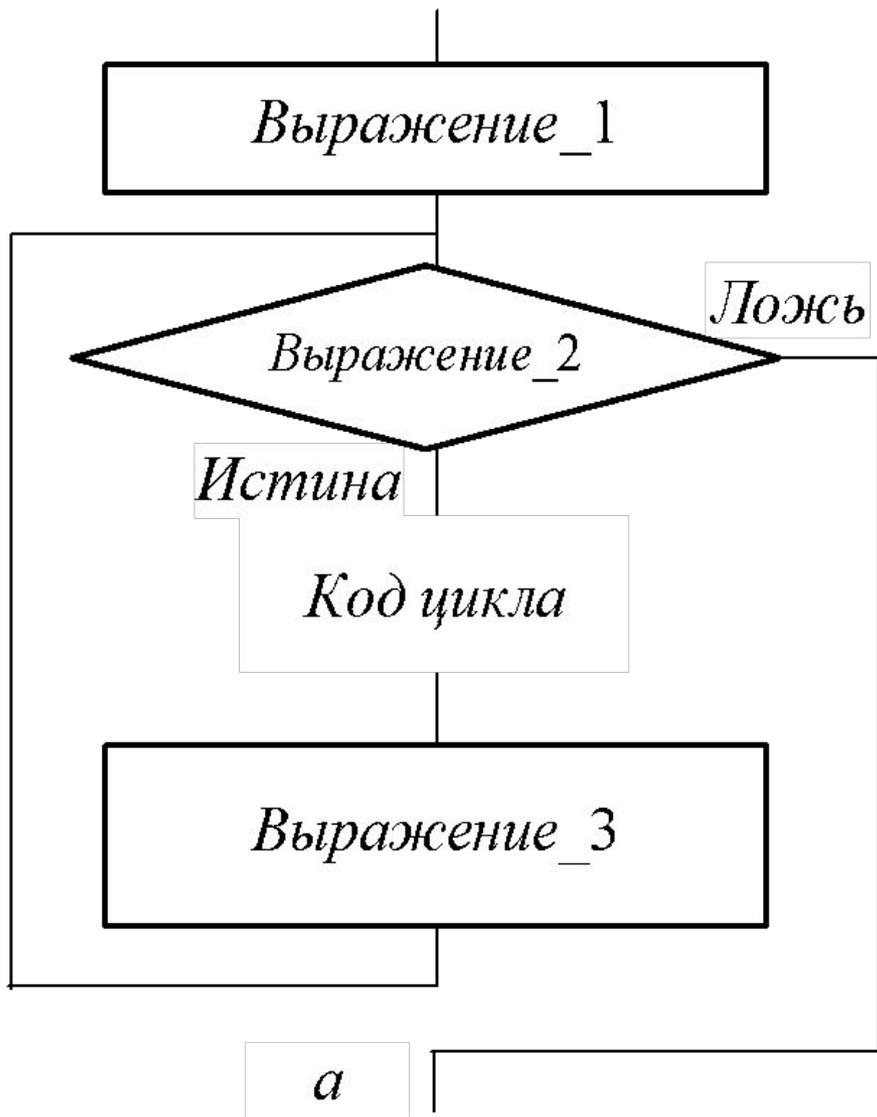


Схема работы (а), блок-схема (б) оператора **for**

В **Выражении 2** определяется условие продолжения цикла, если его результат не нулевой (**Истина**), то цикл выполняется.

Коррекция (**Выражение_3** - изменение параметра цикла) выполняется после каждого шага цикла.

Выражения 1,2,3 могут отсутствовать, но символы «;» опускать нельзя, например бесконечный цикл

for (; ;) Код цикла

Пример. Найти сумму первых **N** натуральных чисел:

```
sum = 0;
```

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

Заметим, что в **Выражении 1** переменную можно декларировать:

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

Областью действия такой переменной будет *Код цикла*.

Очевидно, что **for** эквивалентен последовательности:

```
    Выражение_1;  
    while (Выражение_2) {  
        ...  
        Выражение_3;  
    }
```

Если в цикле **for** отсутствует **Выражение_2**, то цикл будет выполняться бесконечно.

Бесконечный оператор: `for (; ;) Код цикла;`

эквивалентен оператору `while (1) Код цикла;`

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

Рассмотренный пример суммирования натуральных чисел можно записать:

```
for (sum = 0, i = 1; i <= N; sum += i, i++);
```

Или еще короче

```
for (sum = 0, i = 1; i <= N; sum += i++ );
```

В Си нет ограничения на тип параметра цикла.

Пример. Вывести символы-буквы и их коды:

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

```
cout << " Letter " << ch << " kod " << (int)ch << endl;
```

Наиболее часто встречающиеся **ошибки** при создании циклов:

- использование в *Коде цикла* неинициализированных переменных;

- неверная запись условия выхода из цикла;

- запись «;» после заголовка цикла **for**.

Чтобы избежать ошибок, нужно стараться:

- проверить, всем ли переменным **правой части** операторов присваивания присвоены значения;

- проверить, изменяется ли в цикле переменная, управляющая выходом из него.

Операторы и функции передачи управления

Формально простыми операторами передачи управления являются:

- безусловный переход **goto** (перейти на . . .);
- переход к следующему шагу цикла **continue**;
- **break** – выход из цикла и оператора **switch**;
- оператор возврата из функции **return**.

Оператор безусловного перехода

goto Метка ;

передает управление оператору с указанной **Меткой**.

Метка – идентификатор, записанный по синтаксису языка Си с символом «двоеточие» после него. Например, пустой оператор «;» с меткой **m1** :

m1: ;

Область действия **Метки** – функция, где она определена.

Имена меток не декларируются.

Циклы и ***switch*** могут быть вложены вдруг в друга и ***наиболее оправданный случай*** использования оператора ***goto*** – выход во вложенной структуре.

Например, выход из двух (или более) вложенных операторов ***for*** при возникновении грубых ошибок:

```
for (...) {  
    for (...) {  
        ...  
                if (Ошибка) goto error;  
    }  
    ...  
}
```

error : - Устранение ошибки

Второй случай: переход из нескольких мест функции в одно, например, когда перед завершением работы функции необходимо сделать одни и те же действия.

Оператор ***continue*** (*продолжить*) используется во всех циклах, выполняя 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*** – переход к проверочной части (***Выражение*** в заголовке). В цикле ***for*** управление передается на шаг коррекции, т.е. к ***Выражению_3***.

Оператор ***break*** (*прекратить*) выполняет выход из цикла или оператора ***switch*** и передает управление оператору, следующему за текущим, т.е. ***break*** обеспечивает переход в точку кода программы, находящуюся за оператором, внутри которого он (***break***) находится.

Оператор ***return*** (*возврат*) выполняет выход из текущей функции в точку ее вызова. Он, так же возвращает значение результата функции:

return *Выражение*;

Правила использования данного оператора будут рассмотрены позже.

Функции exit и abort

Функция ***exit*** (*выход*) прерывает программу и используется для завершения работы программы, например, при делении на ноль.

Функция описана файле ***stdlib.h*** и выглядит:

```
void exit ( int exit_code );
```

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

Для завершения работы программы также может использоваться функция (*прерывание*)

```
void abort ( void );
```

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