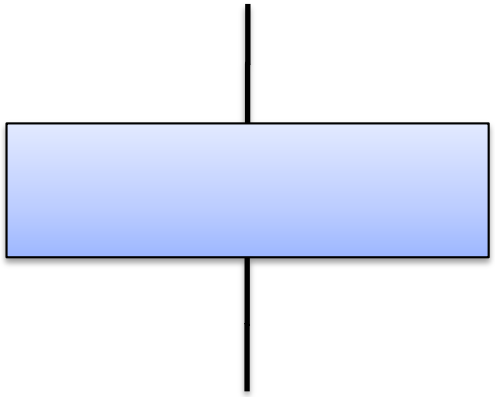
Лекция 5. Управляющие операторы Си: циклы.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

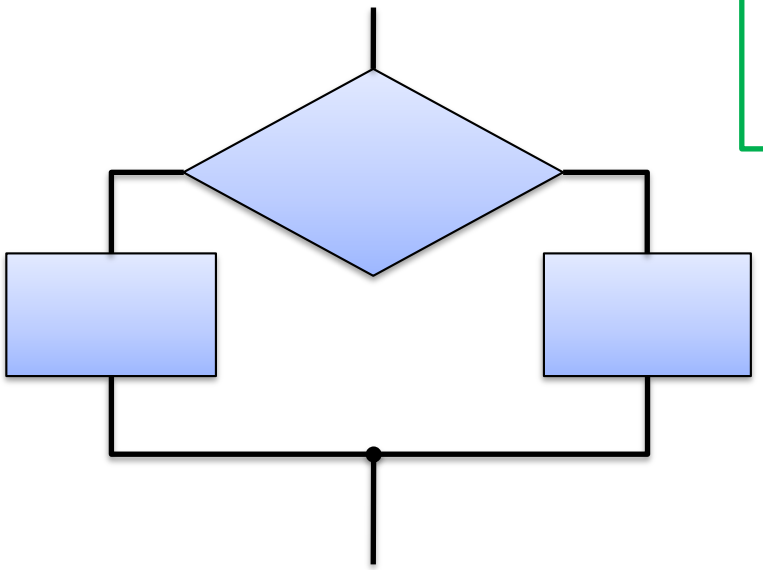
Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Базовые конструкции

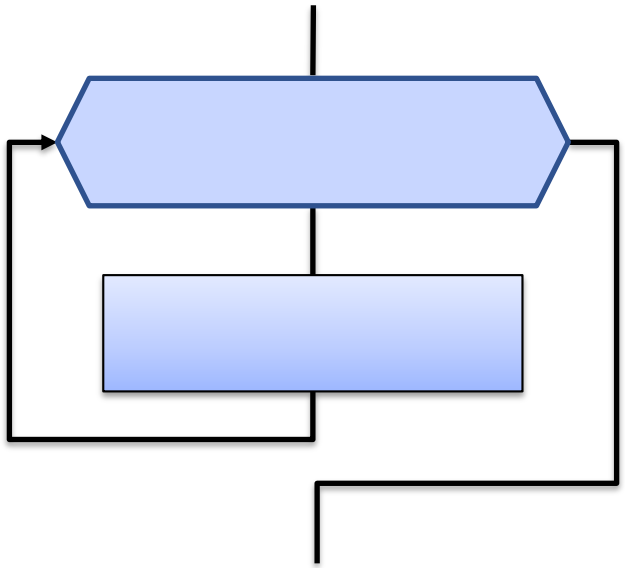
Следование



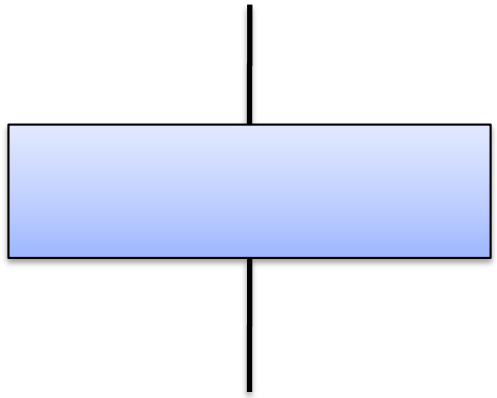
Ветвление



Цикл



Конструкция следование



Следование

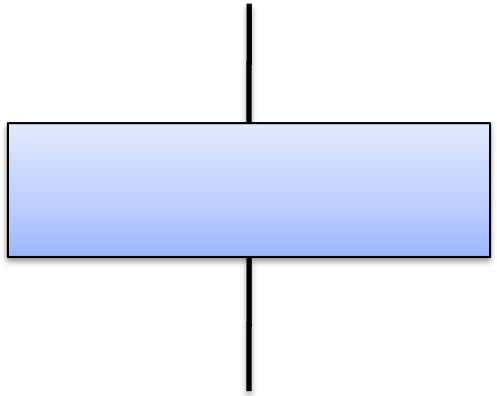
Операторы в языке Си выполняются *последовательно*. После завершения текущего оператора производится переход к следующему оператору по тексту программы.

Исключением являются: операторы, входящие в состав ветвления или цикла.

Например:

```
// Определение переменных
int a = 10, b = 5, c;
// операторы
c = a + b;
c = c * 2;
a = c / b;
b = a * 1.5;
```

Блок операторов



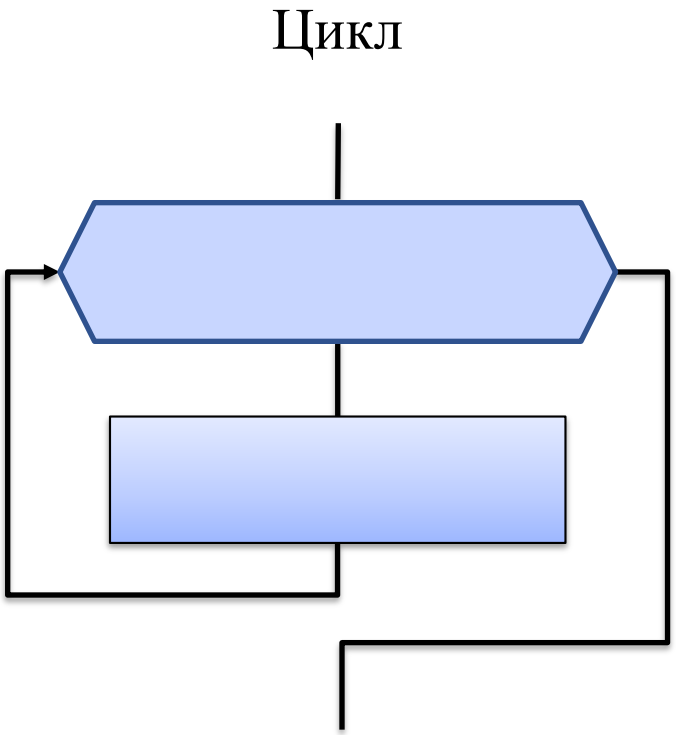
Следование

Определения переменных и набор операторов могут быть сгруппированы вместе при помощи фигурных скобок в составной оператор.

Например:

```
// Определение переменных
int a = 10, b = 5, c;
// Блочный оператор
{
c = a + b;
c = c * 2;
}
a = c / b;
b = a * 1.5;
```

Цикл



Цикл – разновидность управляющей конструкции в высокоуровневых языках программирования. Предназначена для организации *многократного* исполнения *однотипных инструкций* над *данными*.

В языке Си предусмотрено 3 циклических конструкции, которые являются *взаимозаменяемыми*:

1) с предусловием – **while**;
2) по счётчику - **for**
3) с постусловием – **do-while;**

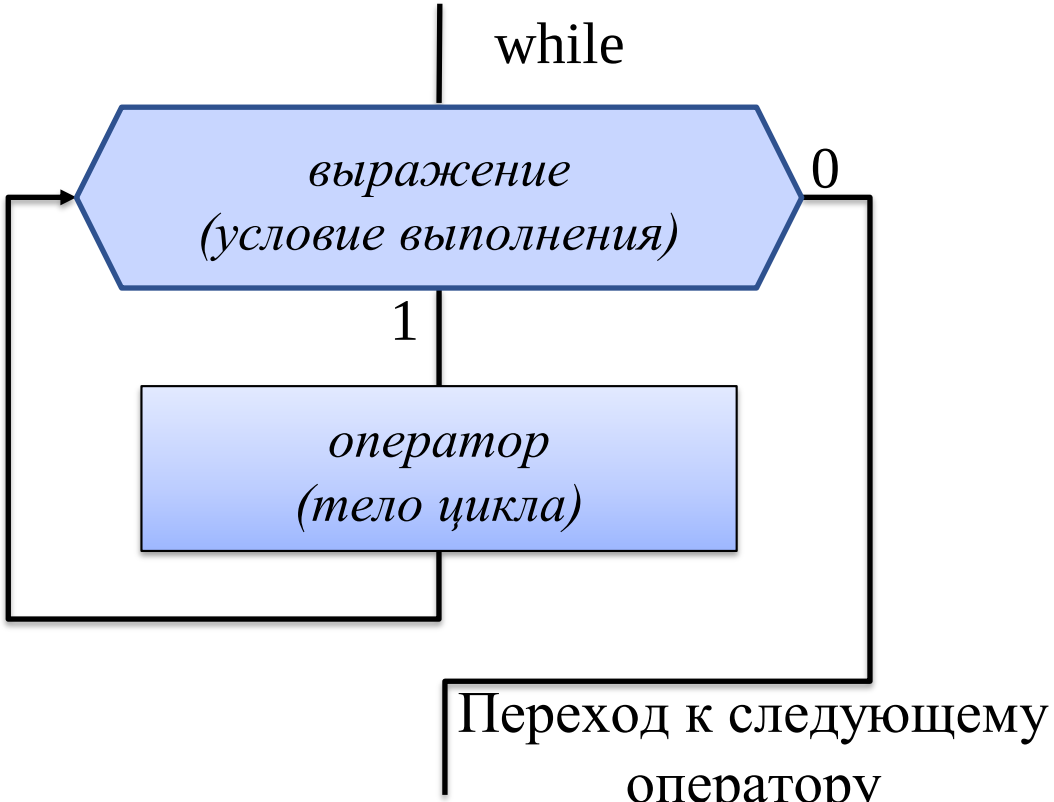
Оператор while

Общая форма цикла while имеет следующий вид:

while (*выражение*)

оператор

Где *оператор* может быть **простым** или **составным** оператором, заключенным в фигурный скобки (блочный оператор)



Если *выражение* **истинно**, то часть *оператор* выполняется один раз и затем *выражение* проверяется снова. И далее это происходит до тех пор, пока *выражение* не станет **ложным**.

Ложным считается нулевой результата (**0**).

Каждый последующий процесс проверки называется **итерацией.**

Оператор while

Цикл ***while***, называют циклом с **пред**условием. Таким образом его работа напрямую связана с выражением.

```
while(1)
{
  printf("Бесконечный цикл\n");
}
```

А вернее проверкой **истинности** выражения.

```
while(0)
{
  printf("Цикл без итераций\n");
}
```

Оператор while

При построение цикла ***while:***

1) Должен быть предусмотрен код, который позволяет изменить значение проверочного выражения.
2) Цикл должен быть конечным, в противном случае он никогда не закончится.

```
int index = 1;
while (index < 5)
   printf("Доброе утро!\n");
```

Что можно сказать об этом цикле?

```
int index = 1;
while (--index < 5)
   printf("Доброе утро!\n");
```

А об этом?

Особенности while

```
int index = 1 ;
while (index < 5)
   printf("Доброе утро!\n");
   index++;
```

Обратите внимание, что частью цикла является только *один оператор*, ***простой*** или ***составной.***

```
int index = 1 ;
while (index < 5)
{
   printf("Доброе утро!\n");
   index++;
}
```

Особенности while

Оператор ***while***, даже если в нем используется составной оператор, синтаксически считается одним оператором.

Начинается с ключевого слова ***while*** до первой точки с запятой или до закрывающей фигурной скобки при наличии составного оператора.

```
int index = 1 ;
while (index < 5);
{
   printf("Доброе утро!\n");
   index++;
}
```

Пустой оператор

Сравнение: операции и выражения отношений

Выражения написанные при помощи ***операторов отношений*** называются – ***выражениями отношений***.

Результатом выполнения ***операторов отношения*** являются значения **0** или **1**, где **0**=>***Ложно***, а **1**=>***Истина***

Операция	Описание
<	Меньше
<=	Меньше или равно
==	Равно
>=	Больше или равно
>	Больше
!=	Не равно

В поисках истины

Выражения написанные при помощи ***операторов отношений*** называются – ***выражениями отношений***.

Результатом выполнения ***операторов отношения*** являются значения **0** или **1**, где **0**=>***Ложно***, а **1**=>***Истина***.

```
int true_val, false_val;
true_val = (10 > 2) ;
false_val = (10 == 2);
printf("true = %d; false = %d \n", true_val,
false_val);
```

```
true = 1; false = 0
```

Что ещё может быть истиной?

```
int n = 3;
   while (n)
       printf ("%2d it's true\n", n--) ;
       printf("%2d it's false\n", n) ;
       n = -3 ;
       while (n)
       printf ("%2d it's true\n", n++);
       printf("%2d it's false\n", n);
   return 0;
```

```
 3 it's true
 2 it's true
 1 it's true
 0 it's false
-3 it's true
-2 it's true
-1 it's true
 0 it's false
```

А что не так с понятием истины

```
long num;
    long sum = 0L;
    int status;
    status = scanf("%ld", &sum) ;
    while (status = 1)
    {
        sum = sum + num;
        printf("Введите следующее целое число (или q для завершения программы): ") ;
        status = scanf("%ld", &num);
    }
```

А что не так с понятием истины?

```
long num;
    long sum = 0L;
    int status;
    status = scanf("%ld", &num) ;
    while (status = 1)
    {
        sum = sum + num;
        printf("Введите следующее целое число (или q для завершения программы): ") ;
        status = scanf("%ld", &num);
    }
```

Оператор присваивания

Оператор присваивания

Целочисленный тип данных _Bool

В стандарте C99 для переменных предназначенных для представления истинных и ложных значений был введён тип _Bool.

Назван в честь Джорджа Буля, английского математика, который разработал алгебраическую систему, позволяющую формулировать и решать логические задачи.

Переменная типа _Bool может иметь только значения 1 (“истина”) и 0 (“ложь”).

Стандарт C99 также предлагает заголовочный файл stdbool.h, в котором bool сделан псевдонимом типа _Bool и определены символические константы true и false для значений 1 и 0.

Приоритеты в операциях отношений

В контексте программы **операции отношений** имеют приоритет по аналогии с **арифметическими операциями**.

(X > Y + 2)

Следовательно, **операции отношений** имеют меньший приоритет, чем **арифметические операции.**

Z = X > Y + 2

Однако, имеют больший приоритет чем **операция присваивания.**

Операции отношений по приоритету делятся на две группы:

Группа с большим приоритетом: < , <=, >, >=

Группа с меньшим приоритетом: ==, !=

Таблица приоритетов (обновленная)

Операции (в порядке убывания приоритета)	Ассоциативность
`( )`	Слева направо
`-  +  ++  --  sizeof  (`*тип*`)` (все унарные)	Справа налево
`*  /  %`	Слева направо
`+  -`	Слева направо
`<  >  <=  >=`	Слева направо
`==  !=`	Слева направо
`=`	Справа налево

Условная классификация циклов

Неопределенный цикл – это означает что количество итераций цикла изначально не предопределено и зависит от того когда *выражение* станет ложным.

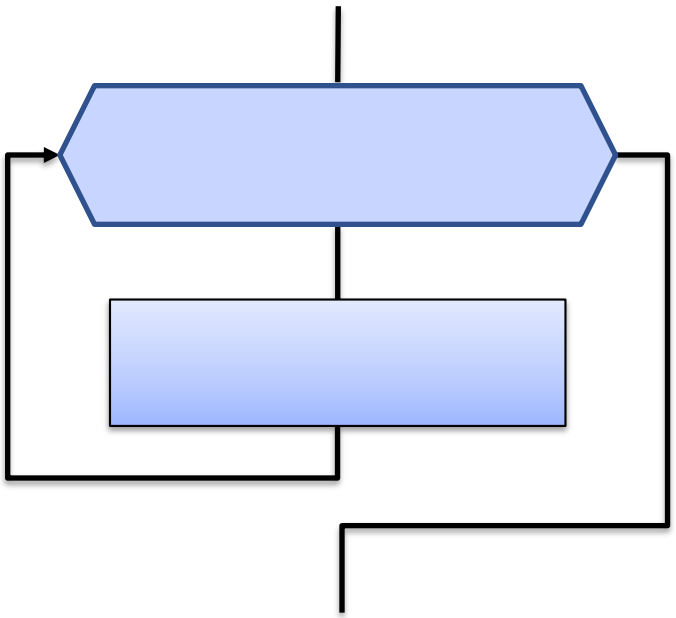
```
input_is_good = (scanf("%ld", &sum) == 1);
while (input_is_good)
```

Такие циклы применимы в случае если мы не знаем какой объём данных предстоит вычислить.

Определенный цикл – такой цикл в котором изначально известное количество итераций.

1. Инициализировать счетчик.
2. Сравнить показание счетчика с некоторой граничной величиной.
3. Инкрементировать значение счетчика на каждом проходе цикла.

Цикл



Цикл – разновидность управляющей конструкции в высокоуровневых языках программирования. Предназначена для организации *многократного* исполнения *однотипных инструкций* над *различными данными*.

В языке Си предусмотрено 3 циклических конструкции, которые являются *взаимозаменяемыми*:

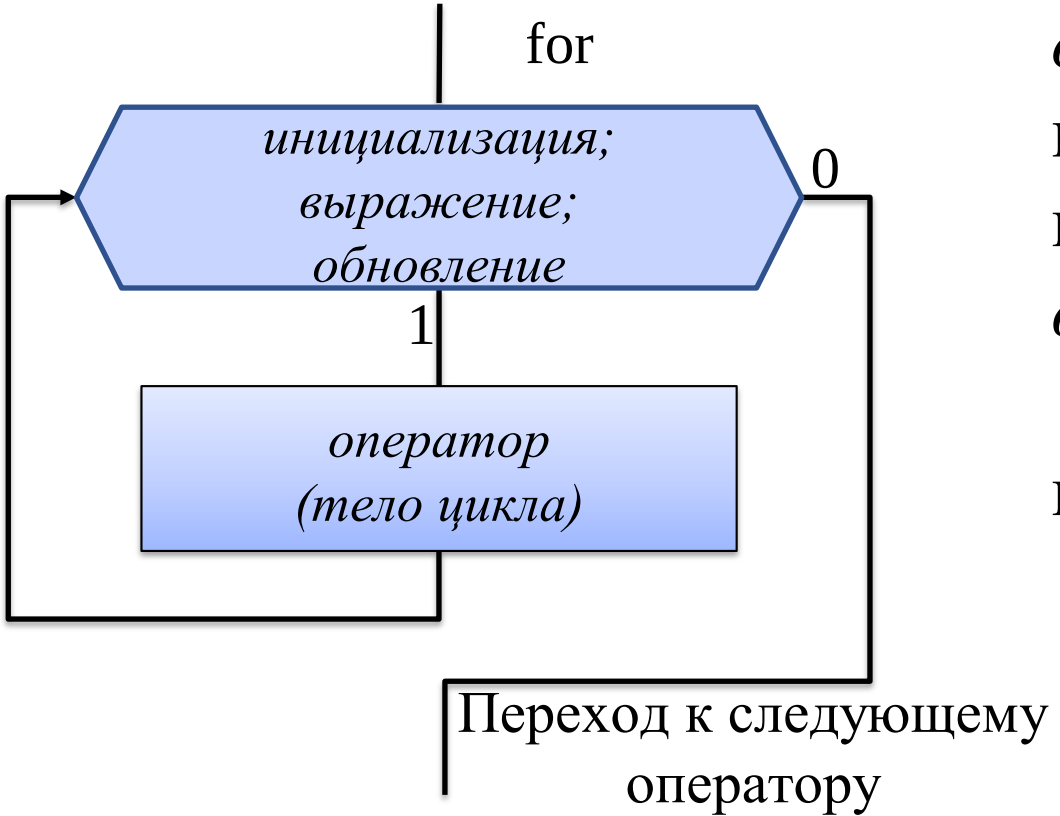
1) с предусловием – **while**;
2) по счётчику - **for**
3) с постусловием – **do-while;**

Оператор for

Общая форма цикла for имеет следующий вид:

for (*инициализация*; *выражение*; *обновление*)
оператор

Где *инициализация* вычисляется до выполнения любых операторов внутри цикла; Затем проверяется *выражение,* если оно **истинно**, то часть *оператор* выполняется один раз; Далее вычисляется выражение *обновление,* после чего возвращается к проверке *выражения*.



И далее это происходит до тех пор, пока *выражение* не станет **ложным**.

Использование оператора for

В Си возможности цикла for или цикла по счётчику значительно выше чем в других языках программирования.

- Можно реализовывать обратный отсчёт.

```
for (int secs = 5; secs > 0; secs--)
```

- При желании можно считать двойками, десятками и т.д.

```
for (int secs = 5; secs > 0; secs= secs+10)
```

- Можно делать подсчет п о символам, а не числам.

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

- Можно выполнять проверку условия, отличного от количества итераций.

```
for (num = 1; num*num*num <= 216; num++)
```

Использование оператора for (продолжение)

- Можно позволить некоторой величине расти не в арифметической, а в геометрической прогрессии;

```
for (float debt = 100.0; debt < 150.0; debt = debt * 1.1)
```

- В качестве третьего выражения можно использовать любое допустимые выражение. Чтобы вы здесь не поместили, это будет обновляться на каждой итерации.

```
for (int x = 1; y <= 75; y = ( ++x * 5) + 50)
```

- Можно даже оставить одно или несколько выражений пустыми (но не опускайте точки с запятой).

```
for (int n = 3; ans <= 25; )
```

еще пример

```
for(;;)
```

Использование оператора for (продолжение)

- Первое выражение не обязательно должно инициализировать переменную.

```
for (printf( "Продолжайте вводить числа!\n"); num != 6; )
```

- Параметры выражений цикла могут изменяться с помощью действий внутри тела цикла. Например, предположим, что цикл определен следующим образом:

```
for (int n = 1;n < 10000; n = n + delta)
  {
    delta=n%3;
    …
  }
```

Короткие операции присваивания

scores += 20 — то же, что и scores = scores + 20

dimes -= 2 — то же, что и dimes = dimes - 2

bunnies *= 2 — то же, что и bunnies = bunnies * 2

time /= 2.73 — то же, что и time = time / 2.73

reduce %= 3 — то же, что и reduce = reduce % 3

Более сложные примеры использования:

x *= 3 * y +12 — то же, что и x = x * (3* y +12)

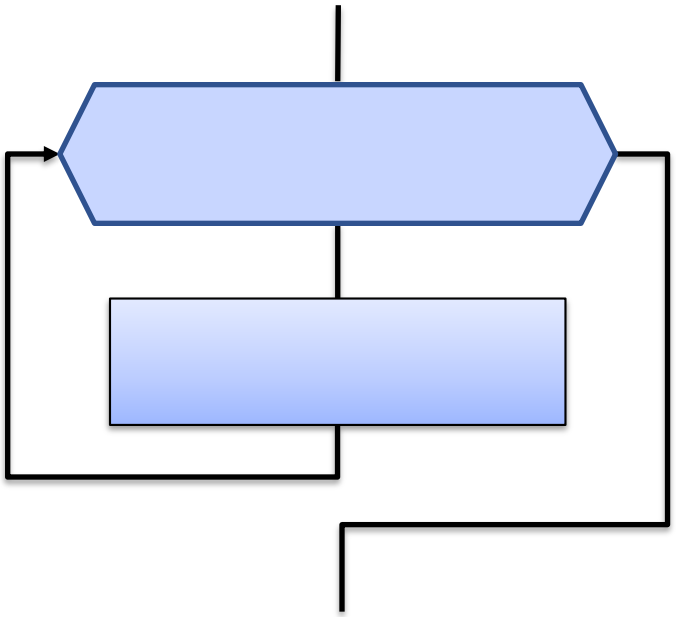
Операция запятой в цикле for

Оператор запятой в **операторе for,** позволяет расширить количество значений *инициализации*, *выражений* и *обновлений*.

```
for (int ounces=l, cost=FIRST_OZ; ounces <= 16; ounces++,cost += NEXT_OZ)
        printf("%5d $%4.2f\n", ounces, cost/100.0);
```

```
for (int time=0, power_of_2=1, t_ct=l; t_ct <= limit; t_ct++, power_of_2 *= 2.0)
```

Цикл



Цикл – разновидность управляющей конструкции в высокоуровневых языках программирования. Предназначена для организации *многократного* исполнения *однотипных инструкций* над *различными данными*.

В языке Си предусмотрено 3 циклических конструкции, которые являются *взаимозаменяемыми*:

1) с предусловием – **while**;
2) по счётчику - **for**
3) с постусловием – **do-while;**

Оператор do…while

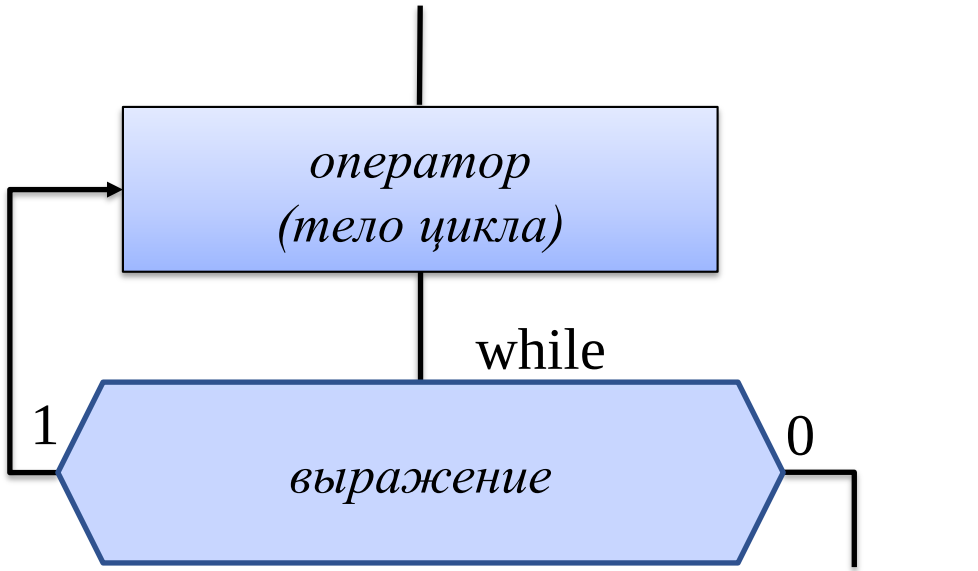
Общая форма цикла **do…while** имеет следующий вид:

do

оператор

…while (*выражение*);

Оператор выполняется один раз, после чего проверяется *выражение*. Если *выражение* **истинно**, то *оператор* выполняется ещё раз. Затем *выражение* проверяется снова. И далее это происходит до тех пор, пока *выражение* не станет **ложным**.



Гарантированно оператор выполнится один раз!!!

```
do
    scanf("%d", &number);
while (number !< 20);
```

Оператор do…while

Цикл ***do…while***, называют циклом с **пост**условием.

```
do{
    printf("Одна итерация точно будет\n");
}
while(0);
```

Поведение цикла определяется его *условием-выражением*.

Вложенные циклы

Вложенный цикл – представляет собой вложение одного цикла в другой.

```
int row;
char ch;
for (row = 0; row < ROWS; row++)
{
     for (ch = 'A'; ch < ('A' + CHARS); ch++)
     printf ("%c", ch) ;
    printf("\n");
}
```

```
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
```

Вложенные циклы (продолжение)

Вложенный цикл – представляет собой вложение одного цикла в другой.

```
int row;
char ch;
for (row = 0; row < ROWS; row++)
{
     for (ch = ('A'+row); ch < ('A' + CHARS); ch++)
     printf ("%c", ch) ;
    printf("\n");
}
```

```
ABCDEFGHIJ
BCDEFGHIJ
CDEFGHIJ
DEFGHIJ
EFGHIJ
FGHIJ
```

Строковый массив данных

Вместо специального типа данных в Си для хранения строковых данных применяются **массив** типа **char**.
Каждый символ по отдельности хранится в своей ячейки памяти в естественном виде, также как он и расположен в строке.
Из вышесказанного следует, что память для хранения строки конечного размера выделяется последовательно и единственным “куском”.
На окончание строки указывает символ ‘\0’

Массивы

Массив — это совокупность значений одного и того же типа, такая как 10 значений **char** или 15 значений **int**, которые хранятся в памяти последовательно.

Массив целиком носит свое имя, а доступ к его отдельным *элементам* осуществляется с применением целочисленного индекса.

Рис. 6.7. Массивы `char` *и* `int` *в памяти*

Массивы (продолжение)

int nannies[22]; массив для хранения 22 целых чисел
char actors [26]; массив для хранения 26 символов
long big [500] ; массив для хранения 500 целых чисел типа long

__тип_данных__ имя_переменной [N];

Следует учитывать что значение N должно быть **константным**.

int num[10] = {31, 32, 33, 34, 35, 36, 37, 38, 39, 40}
[0], [1], [2], [3], [4], [5], [6], [7] , [8], [9]

Задачи на практику

Читать: Главу 6. Управляющие операторы С: циклы.

- Акцент на использование циклов в зависимости от задачи.
- Разобраться с функциями возвращающими значения.

Смотреть вопросы для самоконтроля (240 стр, page 236)
Упражнения по программированию (243-246 стр, page 270-274)

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. Кафедры вычислительных систем

Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)