Структура программы и основные алгоритмические структуры языка Си

- В языке Си исходные файлы бывают двух типов:
- заголовочные, или h-файлы
- файлы реализации, или Си-файлы
- Имена заголовочных файлов имеют расширение ".h". Имена файлов реализации имеют расширения ".c".

Заголовочные файлы

Заголовочные файлы содержат только описания. Прежде всего, это прототипы функций (**Прототипом функции** в языке Си называется объявление функции, не содержащее тела функции, но указывающее имя функции, арность, типы аргументов и возвращаемый тип данных). Также в h-файлах описываются имена и типы внешних переменных, константы, новые типы, структуры и т.п. При трансляции заголовочных файлов, как правило, никакие объекты не создаются. Заголовочные файлы служат для передачи информации между модулями.

Заголовочные файлы

Заголовочный файл подключается с помощью директивы препроцессора #include. Директива #include является специальной командой компилятора. Она предписывает компилятору включить в программу содержимое определенного файла, как если бы вы сами ввели его в текущий исходный файл. Например, описания стандартных функций ввода-вывода включаются с помощью строки:

```
#include <stdio.h>
```

Имя h-файла записывается в угловых скобках, если этот h-файл является частью стандартной Си-библиотеки и расположен в одном из системных каталогов.

Файлы реализации

- **Файлы реализации** содержат тексты функций и определения глобальных переменных. Файлы реализации - это отдельные модули, которые разрабатываются и транслируются независимо друг от друга и объединяются при создании выполняемой программы. Файлы реализации могут подключать описания, содержащиеся в заголовочных файлах. Говоря упрощенно, Си-файлы содержат сами программы, а h-файлы - лишь информацию о программах.

Заголовочные файлы

- Всегда в описании каждой функции указывается, какой заголовочный файл необходим для её успешного использования. Это также делается с помощью директивы препроцессора #include.
- Имена h-файлов, созданных самим программистом в рамках разрабатываемого проекта и расположенных в текущем каталоге, указываются в двойных кавычках, например:
- #include "mylib.h"
- Директивы препроцессора содержат символ # в начале строки и используется в основном для подключения h-файлов.

Препроцессор

- Препроцессор — это специальная программа, являющаяся частью компилятора языка Си. Она предназначена для предварительной обработки текста программы. Препроцессор позволяет включать в текст программы файлы и вводить макроопределения. Работа препроцессора осуществляется с помощью специальных директив (указаний). Они отмечаются знаком решетка #. По окончании строк, обозначающих директивы в языке Си, точку с запятой можно не ставить.

Препроцессор

- ***Основные директивы препроцессора***
- #include — вставляет текст из указанного файла
  #define — задаёт макроопределение (макрос) или символическую константу
  #undef — отменяет предыдущее определение

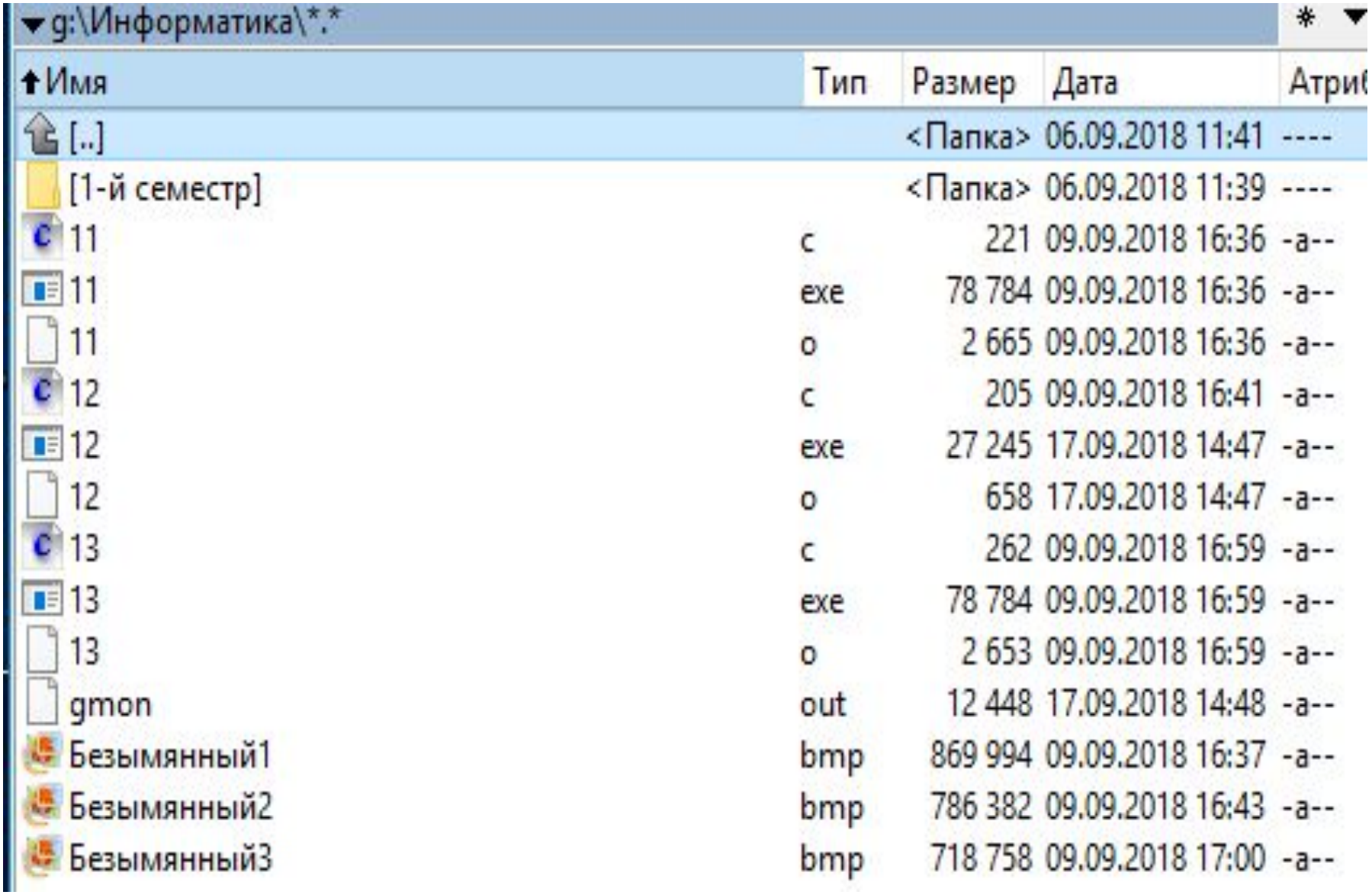
Директива #define

Директива **#define** позволяет вводить в текст программы константы и макроопределения.
Общая форма записи

#define ***Идентификатор*** **Замена**

Поля **Идентификатор** и **Замена** разделяются одним или несколькими пробелами.
Директива #define указывает компилятору, что нужно подставить строку, определенную аргументом **Замена**, вместо каждого аргумента **Идентификатор** в исходном файле. Идентификатор не заменяется, если он находится в комментарии, в строке или как часть более длинного идентификатора.

Компилирование



g:\Информатика*.*

Имя	Тип	Размер	Дата	Атри
[..]		<Папка>	06.09.2018 11:41	----
[1-й семестр]		<Папка>	06.09.2018 11:39	----
11	c	221	09.09.2018 16:36	-a--
11	exe	78 784	09.09.2018 16:36	-a--
11	o	2 665	09.09.2018 16:36	-a--
12	c	205	09.09.2018 16:41	-a--
12	exe	27 245	17.09.2018 14:47	-a--
12	o	658	17.09.2018 14:47	-a--
13	c	262	09.09.2018 16:59	-a--
13	exe	78 784	09.09.2018 16:59	-a--
13	o	2 653	09.09.2018 16:59	-a--
gmon	out	12 448	17.09.2018 14:48	-a--
Безымянный1	bmp	869 994	09.09.2018 16:37	-a--
Безымянный2	bmp	786 382	09.09.2018 16:43	-a--
Безымянный3	bmp	718 758	09.09.2018 17:00	-a--

Базовые алгоритмические структуры

В соответствии с теоремой о структурировании известно, что любой сколь угодно сложный алгоритм можно реализовать с помощью трех базовых алгоритмических структур: следование, ветвление и цикл. Эти три типа базовых алгоритмических структур имеют операторную поддержку во всех языках программирования. Они позволяют создавать однократно и линейно выполняемые участки программы, участки программы с разветвлением программного кода в зависимости от условия и участок программы повторяемый некоторое количества раз.

Линейная структура (следование)

Следование это самая простая алгоритмическая структура. В ней операторы располагаются в необходимой для выполнения последовательности и выполняются один раз.

- **Пример:** a = a + b;

  c = 15;

  d = d + c;

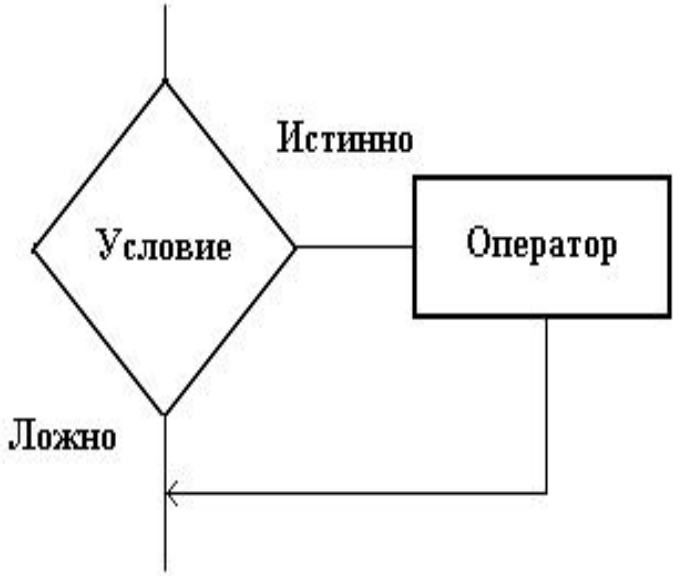
Условный оператор (оператор ветвления)

Условные операторы во многих языках программирования обычно бывают 3-х типов: простое условие (неполный оператор), условие с альтернативными действиями (полный оператор) и множественный выбор (оператор множественного выбора).

Условный оператор (оператор ветвления)

Первый тип постановки условия может быть изображен в виде блок-схемы, как показано на рис. Если условие верно, то выполняется блок из одного или более операторов, в противном случае они пропускаются и указатель команд переходит к оператору расположенному после условного оператора.

Этот частный случай ветвлен получил название "обход" и иногда рассматривается как самостоятельная алгоритмическая структура.



Условный оператор if

Синтаксис на языке Си выглядит следующим образом:

if (условие) {блок из одного или более операторов}

Пример: if (a > 0) {c++; b+=a;}

Условный оператор if

Определение совпадающих цифр в трехзначном

```
1   #include <stdio.h>
2   int main()
3   {   int a, e, d, s;
4       printf("a:   "); scanf("%d",&a);
5     s = a / 100;
6     d = (a % 100)/10;
7     e = a % 10;
8     if (s == d)
9         { printf(" equal numbers! \n"); }
10    if (s == e)
11        printf(" equal numbers! \n");
12    if (d == e)
13       printf(" equal numbers! \n");
14       printf("s:%d \t d:%d \t e:%d ",s, d, e);
15                  return 0;}
16
```

```
a:   433
 equal numbers!
s:4       d:3       e:3
Process returned 0 (0x0)   execution time : 4.309 s
Press any key to continue.
```

Операторы if - else

Второй тип это постановка полного условия с альтернативными операторами.

В этом случае некоторый набор действий выполняется и в случае истинности условия, и некоторый другой набор действий в случае его ложности. Если условие верно, то выполняется блок 1 из одного или более операторов, а если не верно, то блок 2.

Операторы if - else

Структура
if (условие) {блок 1 из одного или более операторов}
else {блок 2 из одного или более операторов}

Пример: if (a > 0) {c++; b+=a;}
else {c--; a = 0;}

Допускается использование вложенных операторов if. Оператор if может быть включен в конструкцию if или в конструкцию else другого оператора if. Чтобы сделать программу более читаемой рекомендуется группировать операторы и конструкции во вложенных операторах if, используя фигурные скобки. Если же фигурные скобки опущены, то компилятор связывает каждое ключевое слово else с наиболее близким if, для которого нет else.

Операторы if - else

Пример:

```
int main ( )
 {
   int t=2, b=7, r=3;
     if (t>b)
      {
         if (b < r)  r=b;
      }
     else r=t;
   return 0;
 }
```

```
1   #include <stdio.h>
2
3   int main()
4   {
5    int t=20, b=7, r=10;
6       if (t>b)
7            {
8                if (b < r)   r=b;
9                  }
10                else r=t;
11  printf ("r=%d\n",r);
12               return (0);
13               }
14
```

```
r=7

Process returned 0 (0x0)
```

В результате выполнения этой программы **r** станет равным 2.

Операторы if - else

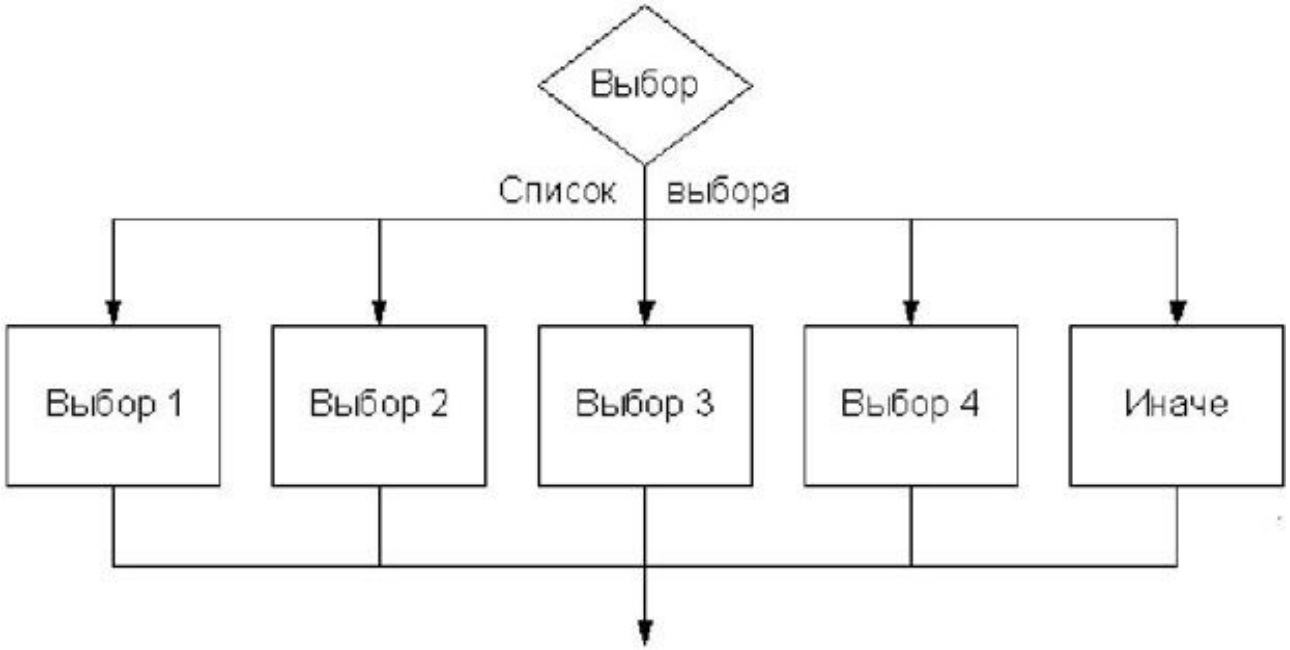
Если же в программе опустить фигурные скобки, стоящие после оператора
if, то программа будет иметь следующий вид:

```
int main  ( )
 {
  int t=2,b=7,r=3;
  if ( t>b )
      if ( b <r ) t=b;
      else       r=t;
  return 0;
 }
```

В этом случае r получит значение равное 3, так как ключевое слово else относится ко второму оператору if, который не выполняется, поскольку не выполняется условие, проверяемое в первом операторе if.

Оператор switch

Третий тип это оператор множественного выбора. Обычно его используют при необходимости совершать различные действия при определенных, но различных значениях некоторой переменной. Возможный вариант записи этого участка алгоритма в виде блок-схемы представлен на рис.



Оператор switch

В заголовке выбор размещается имя переменной относительно значения которой будет осуществляться проверка. Далее, в списке выбора, идут возможные значения этой переменной. При совпадении ее с указанным в списке значением, будут выполнены действия расположенные в этом блоке. Если среди списка не обнаружено совпадение со значением переменной, то могут быть выполнены действия указанные в блоке default. Но его может и не быть, если в этом нет необходимости.

```
switch (выражение)
{
case значение 1: {блок из одного или нескольких операторов 1; break;}
case значение 2: {блок из одного или нескольких операторов 2; break;}
case значение 3: {блок из одного или нескольких операторов 3; break;}
default: блок из одного или нескольких операторов n; break;
}
```

Выражением могут быть любое выражение, константа или переменная типа int или char. Оператор switch удобно использовать вместо нескольких операторов if else.

Оператор switch

Пример:

```
   switch (count)          // начало оператора switch
   {case 1: {c = a + b;   break; }    // если  count = 1, то
выполнить сложение
case 2: {c = a - b;    break; }    // если  count = 2, то
выполнить вычитание
case 3: {c = a * b;   break; }    // если  count = 3, то
выполнить умножение
case 4: {c = a / b;    break; }    // если  count = 4, то
выполнить деление
default:{c = 0;}        // если count равно любому
другому значению
   }
```

Оператор switch

Входящие в эту алгоритмическую структуры операторы имеют следующий смысл:

case - оператор выбора конкретного значения переменной. Таких операторов выбора может быть много, но работать будет именно тот оператор case, который будет равен нашему "определенному значению".

default - оператор выбора который будет выполняться, если ни один из операторов case не содержит нужного значения. Т.е. если ничего не совпало с "определенным значением", то будет выполняться оператор по умолчанию - default.

break - является оператором остановки проверки условий, т. е. с помощью него сразу завершается выполнение оператора выбора и продолжается дальнейшее выполнение программы. Этот оператор необходим в конце каждого блока операторов ветки case, иначе после совпадения с одном из вариантов выбора, будут выполняться действия из других веток множественного

Оператор switch

Выражение в круглых скобках, следующее за ключевым словом switch, может быть любым выражением, допустимым в языке СИ, значение которого должно быть целым.

Значение этого выражения является ключевым для выбора из нескольких вариантов. Тело оператора switch состоит из нескольких операторов, помеченных ключевым словом case с последующим константным выражением.

Так как константное выражение вычисляется во время трансляции, оно не может содержать переменные или вызовы функций. Обычно в качестве константного выражения используются целые или символьные константы.

Оператор switch

Все константные выражения в операторе switch должны быть уникальны. Кроме операторов, помеченных ключевым словом case, может быть ещё (но обязательно один) фрагмент, помеченный ключевым словом default.

Список операторов может быть пустым, либо содержать один или более операторов. В операторе switch не требуется заключать последовательность операторов в фигурные скобки.

Отметим также, что в операторе switch можно использовать свои локальные переменные, объявления которых находятся перед первым ключевым словом case, однако в объявлениях не

Оператор switch

Схема выполнения оператора switch следующая:
- вычисляется выражение в круглых скобках;

- вычисленные значения последовательно сравниваются с константными выражениями, следующими за ключевыми словами case;

- если одно из константных выражений совпадает со значением выражения, то управление передается на оператор, помеченный соответствующим ключевым словом case;

- если ни одно из константных выражений не равно выражению, то управление передается на оператор, помеченный ключевым словом default, а в случае его отсутствия управление передается следующему после switch оператору.

Оператор switch

Рассмотрим пример, в котором иллюстрируется применение оператора switch для изменений в арифметических действиях.

```
#include <stdio.h>
int main()
{
    char ZNAC;
      int x,y,z;
   ZNAC = '-';
   y=5;
   z=3;
      switch (ZNAC)
      {
            case '+':  x = y + z;    break;
            case '-':  x = y - z;    break;
            case '*':  x = y * z;    break;
            case '/':  x = y / z;    break;
            default : x=13;
        }
printf ("x=%d\n",x);
                return 0;
}
```

```
x=2

Process returned 0 (0x0)
```

Циклическая структура

Цикл - форма организации действий, при которой одна и та же последовательность действий совершается несколько раз до тех пор, пока выполняется некоторое условие.

Циклы являются основной частью подавляющего большинства практических программ, поэтому организация цикла есть наиболее часто встречающаяся
задача программирования.

В алгоритме циклического типа можно выделить 2 части: условие, в зависимости от которого выполняется (или не выполняется) группа команд, и группа повторяющихся действий, называемая телом цикла.

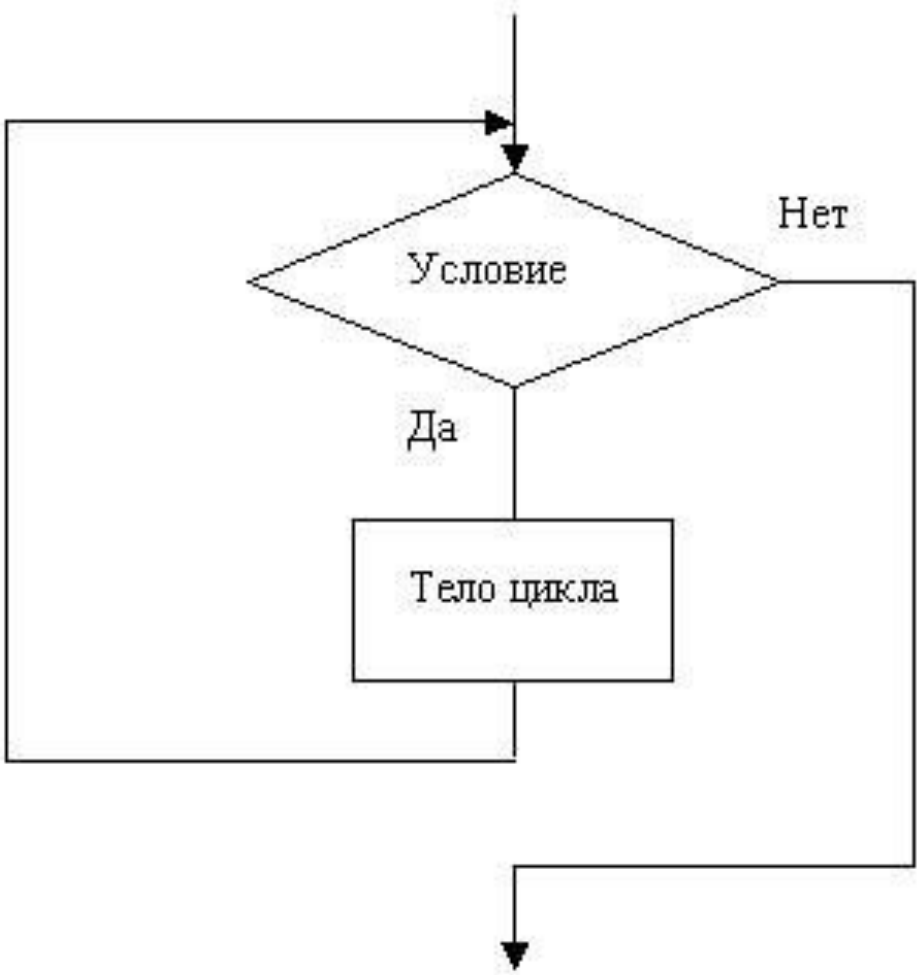
Циклическая структура

Рассмотрим случай, когда условие стоит перед телом цикла. Такая организация называется циклом с предусловием, или циклом "пока" (оператор while в СИ).

В цикле с предусловием оператор начала цикла и условие выхода из цикла размещают в начале этой алгоритмической структуры. Конец цикла определяется местом размещения закрывающей фигурной скобки. Если фигурные скобки не используются, то в цикле учавствует только один оператор, который размещен сразу после оператора начала цикла. Цикл данного типа это цикл с неизвестным числом повторений. Если условие истинно, то выполняется блок из одного или нескольких операторов и происходит возврат на

Циклическая структура

Схема цикла



Оператор while

Оператор цикла while имеет следующий формат:
while(выражение) тело ;
В качестве выражения допускается использовать любое выражение языка Си, а в качестве тела любой оператор, в том числе пустой или составной.

Схема выполнения оператора while следующая:
1. Вычисляется выражение.
2. Если выражение ложно, то выполнение оператора while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполняется тело оператора while.
3. Процесс повторяется с пункта 1.

Оператор while удобно использовать в ситуациях, когда тело оператора не всегда нужно выполнять.

Оператор while

Таблица умножения.

```
1   #include <stdio.h>
2   int main()
3   {
4       int i,y;
5       i=1;
6        while(i<11)
7       {
8         y=i*i;
9         i++;
10        printf("i*i=%d\n",y);
11      }
12         return 0;
13  }
```

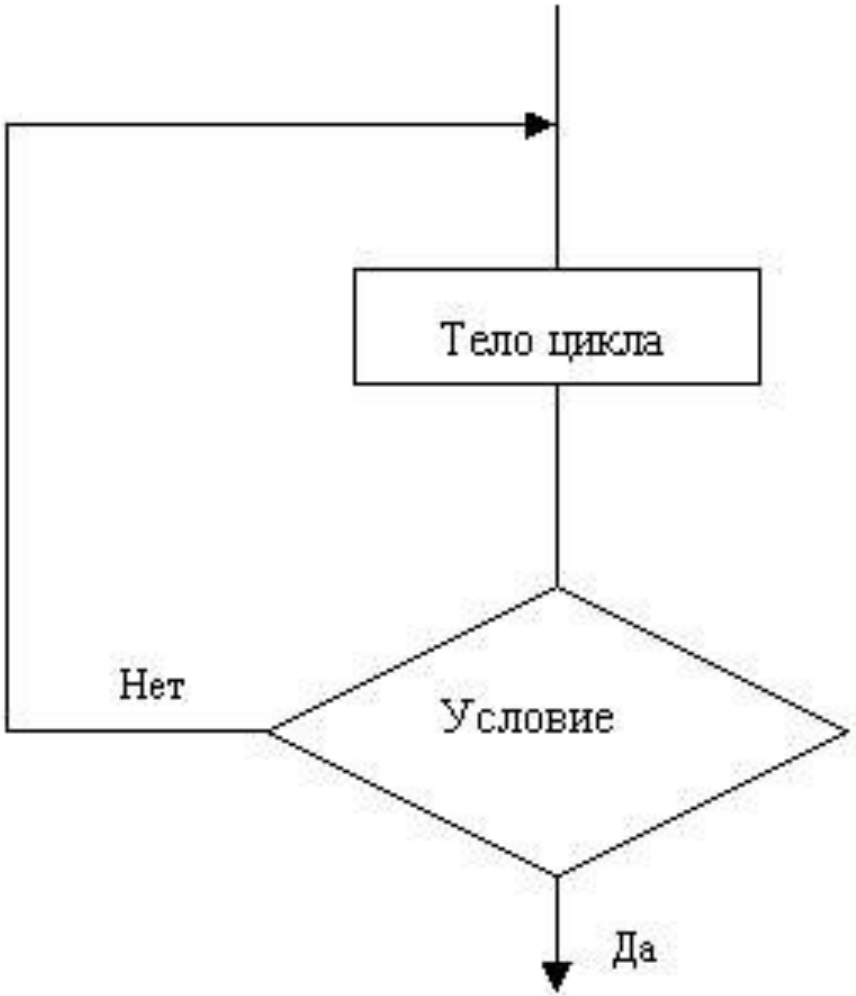
```
i*i=1
i*i=4
i*i=9
i*i=16
i*i=25
i*i=36
i*i=49
i*i=64
i*i=81
i*i=100

Process returned 0 (0x0)
```

Оператор do-while

В цикле с постусловием условие выхода из цикла размещается в конце алгоритмической структуры. Конец цикла определяется местом размещения оператора с условием. Цикл данного типа так же является циклом с неизвестным числом повторений. При достижении циклической структуры программы, прежде всего, выполняются операторы, входящие в тело цикла. После этого проверяется условие выхода из цикла. Если условие истинно, то вновь выполняется блок тела цикла из одного или нескольких операторов. Если условие ложно, происходит выход из цикла. Поскольку условие проверяется в конце цикла, то тело цикла выполняется как минимум один раз.

Оператор do-while. Схема



Оператор do-while

Формат оператора имеет следующий вид:

do

{блок из одного или нескольких операторов}

while (условие)

Пример: do

```
    {a+=0.01; b+=a;}
while (a < 3.14)
```

Схема выполнения оператора do while:

1. Выполняется тело цикла (которое может быть составным оператором).
2. Вычисляется выражение.
3. Если выражение ложно, то выполнение оператора do while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполнение оператора продолжается с пункта 1.

Оператор do-while

Пример программы с оператором do while

```
 1    #include <stdio.h>
 2   int main()
 3   {
 4        int i,y;
 5        i=1;
 6        do
 7        {
 8          y=i*i;
 9          i++;
10          printf("i*i=%d\n",y);
11        }
12           while(i<11);
13
14           return 0;
15   }
```

```
i*i=1
i*i=4
i*i=9
i*i=16
i*i=25
i*i=36
i*i=49
i*i=64
i*i=81
i*i=100

Process returned 0 (0x0)
```

Циклы с предусловием и постусловием особенно удобны, когда количество повторений цикла заранее неизвестно.

Цикл с параметром

- **Цикл с параметром** (со счетчиком) используется в случаях, когда заранее известно число повторений тела цикла. Этот тип цикла частный случай цикла с предусловием, поскольку проверка условия выполняется до выполнения операторов тела цикла. По этой причине возможен вариант постановки условия, когда операторы тела цикла не будут выполнены ни разу. Конец цикла определяется местом размещения закрывающей фигурной скобки. Блок-схема данного типа цикла совпадает с таковой для цикла с предусловием.

Схема реализации. Оператор for

for (начальное выражение; контрольное выражение (условие); счетчик)
{блок из одного или нескольких операторов}
Пример: for (i = 0; i >= b; i++)
{a+=0.01; c+=a;}
В качестве начального выражения можно использовать несколько операторов, их нужно разделять запятыми. **Например:**
for (i=a, s=0; i>=b; i++) s+=i;
Существенно то, что проверка условия всегда выполняется в начале цикла.
Это значит, что тело цикла может ни разу не выполниться, если условие выполнения сразу будет ложным.

Оператор for

В примере вычисляются квадраты чисел от 1 до 9.

```
1   #include <stdio.h>
2   int main()
3   {
4       int i,y;
5           for (i=1; i<10; i++)
6           {
7           y=i*i;
8       printf("i*i=%d\n",y);
9           }
10      return 0;
11  }
12
13
```

```
i*i=1
i*i=4
i*i=9
i*i=16
i*i=25
i*i=36
i*i=49
i*i=64
i*i=81

Process returned 0 (0x0)
```

Оператор for

Оператор for может быть вложенным (пример – таблица умножения).

```
1     #include <stdio.h>
2     int main()
3     {
4         int i,j,y;
5           for (i=1; i<10; i++)
6           {
7
8            for (j=1; j<10; j++)
9            {
10               y=i*j;
11               printf("i*j=%d\n",y);
12           }
13           }
14          return 0;
15    }
16
17
```

```
i*j=1
i*j=2
i*j=3
i*j=4
i*j=5
i*j=6
i*j=7
i*j=8
i*j=9
i*j=2
i*j=4
i*j=6
i*j=8
i*j=10
i*j=12
i*j=14
i*j=16
```

Оператор continue

- Оператор **continue** предназначен для использования только в циклах. Он прерывает выполнение очередного шага цикла и заставляет компьютер начать выполнять новый шаг цикла. Если поместить оператор *continue* в цикл *for* или *while*, компьютер будет игнорировать все операторы, следующие в теле цикла после *continue*. Это оператор без параметра. Обычно оператор *continue* используется, если данные, обрабатываемые в теле цикла неверны, выходят за допустимые пределы или имеют непредвиденные значения. Вместо обработки некорректных данных, бывает лучше вернуться к началу цикла, чтобы получить другое значение.

Оператор continue

Пример вычисления четных сумм чисел от 1 до a. Когда сумма чисел от 1 до a становится нечетной, оператор **continue** передает управление на очередную итерацию цикла for, не выполняя операторы обработки четных

```
1   #include <stdio.h>
2   int main()
3   {
4   int a,b;
5       for (a=1,b=0; a<20; b+=a,a++)
6                   { if (b%2) continue;
7                   printf("b=%d\n",b);
8                   }
9           return 0;
10  }
11
```

```
b=0
b=6
b=10
b=28
b=36
b=66
b=78
b=120
b=136

Process returned 0 (0x0)
```

Оператор break

- Команда **break** используется для выхода из текущего цикла. Она может находиться в любом месте программы, однако обычно *break* ставиться в теле операторов *switch*, *while или do... while*. Часто этот оператор используют без параметра.
- Оператор break обеспечивает прекращение выполнения самого внутреннего из объединяющих его операторов switch, do, for, while. После выполнения оператора break управление передается оператору, следующему за прерванным.

Оператор exit()

В Си предусмотрена возможность досрочного выхода из программы до ее нормального завершения с помощью функции *exit*(). Эта функция, находящаяся в стандартной библиотеке, вызывает немедленное окончание работы программы. Поскольку функция *exit*() останавливает выполнение программы и форсирует возврат в операционную систему, она используется для управления устройствами, и ее использует подавляющее большинство программ. Функция имеет следующий вид: *exit*(статус); где статус - переменная или константа типа int - код завершения. Часто этот оператор ставиться в теле оператора *if*, чтобы закончить программу, в зависимости от результата проверки условия. При использовании *exit* необходимо указывать

Стандартные математические функции

Стандартные математические функции хранятся в библиотеке math.h. Тригонометрические функции используют радианы. Все функции, кроме указанных, принимают один аргумент типа double. Возвращают так же число типа double.

Форма записи различных математических функций в языке Си

Математическая функция	Название	Функция на языке Си
arccos(x)	Аркосинус	acos(x)
arcsin(x)	Арксинус	asin(x)
arctan (x)	Арктангенс	atan(x)
ceil(x)	Округление в большую сторону	ceil(x)
cos(x)	Косинус	cos(x)
exp(x)	*e* в степени *x*	exp(x)
|x|	Модуль	fabs(x)
floor(x)	Округление в меньшую сторону	floor(x)
mod(x, y)	Остаток от деления x на y	fmod(x, y)
ln(x)	Натуральный логарифм	log(x)
lg(x)	Десятичный логарифм	log10(x)
x^y	Возведение основания ***x*** в степень ***y***	pow(x, y)
sin(x)	Синус	sin(x)
$\sqrt{x}$	Квадратный корень	sqrt(x)
tg(x)	Тангенс	tan(x)

Стандартные математические функции

```c
#include <stdio.h>
#include <math.h>
int main()
{
double x,y,z,pi,t,p,m;
pi=3.141592653589793;
x=73.5;
y=sin(x*pi/180);
z=asin(y);
t=sqrt(x);
p=pow(x,0.5);
m=log(x)/log(3);
       printf("y=%f\nz=%f\nt=%f\np=%f\nm=%f\n",y,z*180/pi,t,p,m);
        return 0;
}
```

```
y=0.958820
z=73.500000
t=8.573214
p=8.573214
m=3.911558

Process returned 0 (0x0)
```

Стандартные математические функции

Тип **float** и тип **double**

```
1   #include <stdio.h>
2   #include <math.h>
3   int main()
4   {
5   double x,y;
6   x=1000;
7   y= pow(x,4)+1;
8   printf("x=%.2f y=%.1f \n",x,y)
9   return 0;
10  }
11
```

```
x=1000.00 y=1000000000001.0

Process returned 0 (0x0)
```

```
1   #include <stdio.h>
2   #include <math.h>
3   int main()
4   {
5   float x,y;
6   x=1000;
7   y= pow(x,4)+1;
8   printf("x=%.2f y=%.1f \n",x,y);
9   return 0;
0   }
```

```
=1000.00 y=999999995904.0

rocess returned 0 (0x0)
```