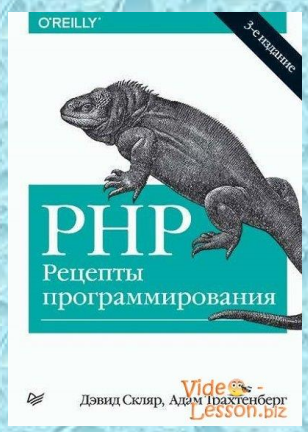








**Программирование** — это процесс создания (разработки) программы, который может быть представлен последовательностью следующих шагов:

1. Спецификация (определение, формулирование требований)
   Разработка алгоритма
2. Кодирование (запись алгоритма на языке программирования)
3. Отладка
4. Тестирование

Для решения любой задачи, необходимо выполнить следующие этапы:

1. Четко определить условия задачи, входные данные и какой результат должен быть получено после решения задачи.
2. Какие дополнительные данные необходимы для решения задачи.
3. Составить блок-схему решения задачи и записать ее в виду удобного описания.
4. Анализ всех возможных проблем и усовершенствование алгоритма.

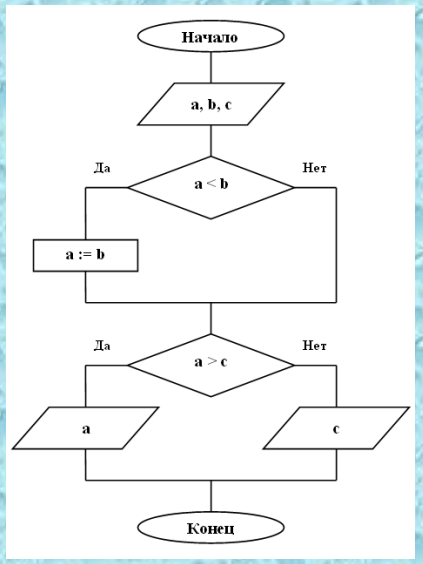
$$ax^2 + bx + c = 0$$

**Постановка**

- Входными данными для нашей задачи являются три коэффициента: a, b и c.
- Решение задачи предполагает вычисление возможных корней уравнения.
- Так как количество корней возможно от 0 до 2, то необходимо в качестве решения указать количество корней и собственно перечислить их.

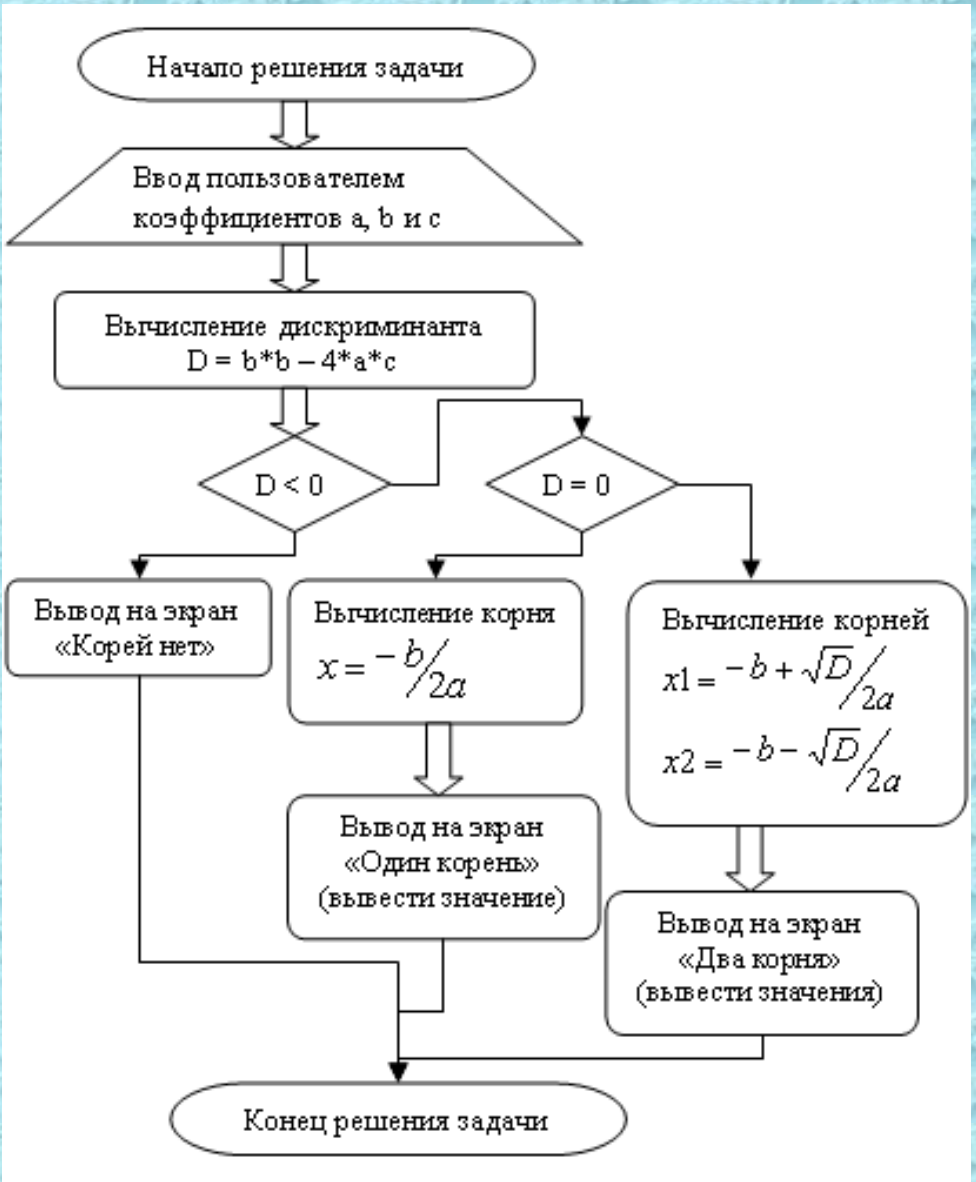
**Дополнительные данные**

- При решении квадратного уравнения необходимо вычислить значение дискриминанта. В нашей задаче, дискриминант является промежуточным результатом, необходимым для решения задачи.




**Описание блок-схемы может производиться с помощью различных средств и обозначений. Основной принцип заключается в наглядности шагов исполнения и однозначности переходов при ветвлении.**

$$ax^2 + bx + c = 0$$


Начало решения задачи
Ввод пользователем коэффициентов a, b и c
Вычисление дискриминанта D = b*b – 4*a*c
D < 0
D = 0
Вывод на экран «Корей нет»
Вычисление корня $x = -b/2a$
Вычисление корней $x1 = -b + \sqrt{D}/2a$ $x2 = -b - \sqrt{D}/2a$
Вывод на экран «Один корень» (вывести значение)
Вывод на экран «Два корня» (вывести значения)
Конец решения задачи

$$ax^2 + bx + c = 0$$

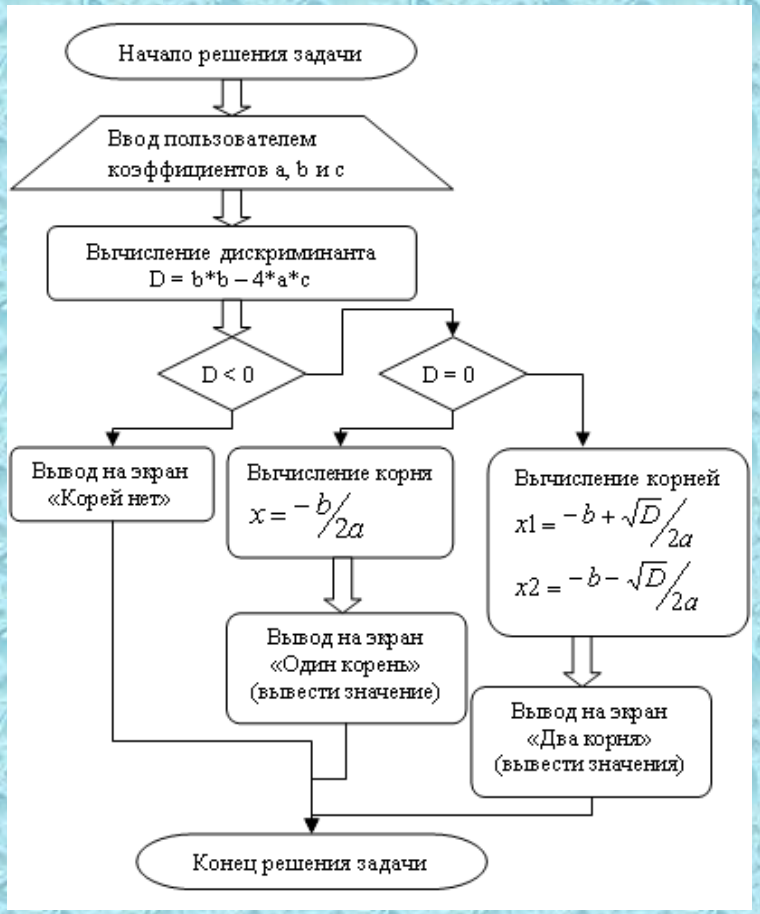
**Анализ созданной блок-схемы** решения задачи.
При этом необходимо ответить на два вопроса:

1. Будет ли схема корректно работать во всем диапазоне входных параметров?
2. Возможны ли оптимизации, которые позволят ускорить процесс выполнения задачи?

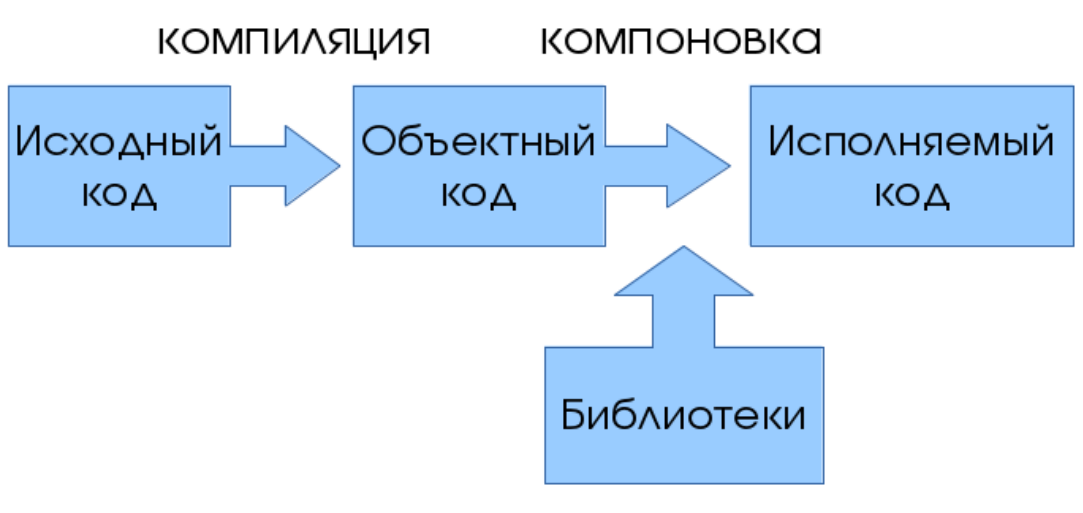
При анализе совершаемых действий, очевидно, что при значении коэффициента **a = 0**, происходит деление на **0!** В качестве решения данной ситуации можно предложить два основных метода:

1. Добавить проверку корректности введенных пользователем значений, и при вводе **a=0**, выдавать сообщение об ошибке: «Уравнение не является квадратным!».
2. Допустить возможность решения линейных уравнений, по сути, расширив диапазон применения вашей реализации. Для этого необходимо добавить такую проверку до вычисления дискриминанта, и идти описанным путем лишь при коэффициенте **a** отличном от 0, иначе добавить еще одну ветвь исполнения.

Что касается оптимизации, то основной ее смысл в удалении лишний действий, а также в недопустимости совершения дублирующий действий. На приведенной выше схеме, дублированным действием является извлечение квадратного корня из дискриминанта.



# Порядок создания программы




**1. Верстка исходного кода**

Для написания исходного кода (текста) программы можно использовать обычные редакторы. Некоторые редакторы имеют возможность подсветки синтаксиса и дополнительные расширения, которые превращают их в компактные IDE (Geany, KWrite, Gedit, Notepad++). Но лучшим решением будет использовать полноценные интегрированные среды разработки — IDE (Integrated Development Environment).

**2. Компиляция**

Используемый компилятор транслирует исходный код программы в объектный, машинно-ориентированный, код. Созданный в результате компиляции исполняемый модуль привязан как к данной платформе, так и к типу процессора. Компиляция завершится успешно, если в программе не будут обнаружены синтаксические ошибки, в противном случае, компилятор выдаст сообщение об ошибке и остановит свою работу.

**3. Компоновка**

Компоновщик (он же линкер) завершает сборку программы объединяя (связывая) объектный код программы с объектным кодом библиотек в финальный продукт — исполняемый код.

**4. Тестирование и отладка**

На этом этапе готовая программа подвергается всестороннему анализу на предмет обнаружения неверных результатов — алгоритмических ошибок. Для этих целей в IDE применяется специальный программный модуль — отладчик.

# 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.

| | |
|---|---|
| **#include** | вставляет текст из указанного файла |
| **#define** | задаёт макроопределение (макрос) или символическую константу |
| **#undef** | отменяет предыдущее определение |
| **#if** | осуществляет условную компиляцию при истинности выражения |
| **#ifdef** | осуществляет условную компиляцию при определённости |
| **#ifndef** | осуществляет условную компиляцию при неопределённости |
| **#else** | ветка условной компиляции при ложности выражения |
| **#elif** | ветка условной компиляции, образуемая слиянием else и if |
| **#endif** | конец ветки условной компиляции |
| **#line** | препроцессор изменяет номер текущей строки и имя файла |
| **#error** | выдача диагностического сообщения |
| **#pragma** | действие, зависящее от конкретной реализации компилятора. |

# Препроцессор

**Директива #include** позволяет включать в текст программы указанный файл.
Если файл является стандартной библиотекой и находится в папке компилятора, он заключается в ***угловые скобки***.
Если файл находится в текущем каталоге проекта, он указывается в кавычках. Для файла, находящегося в другом каталоге необходимо ***в кавычках*** указать полный путь.

```
#include <stdio.h>
#include "func.c"
```

**Директива #define** позволяет вводить в текст программы константы и макроопределения.
Идентификатор не заменяется, если он находится в комментарии, в строке или как часть более длинного идентификатора.

```
#define A 280U   // unsigned int
#define B 280LU  // unsigned long int
#define C 280    // int (long int)
#define D 280L   // long int
#define K 28.0   // double
#define L 28.0F  // float
#define M 28.0L  // long double
```

```
#define SIN(x) sin(PI*x/180)
```

# Основы языка С. Переменные

**Переменная** — это именованная область памяти, в которую могут быть записаны различные значения.

**Тип данных** – определяет значения:

- ***char*** — целое значение, 8 бит (диапазон от –128 до 127);
- ***int*** — целое значение, обычно 4 байта, зависит от платформы;
- ***float*** — вещественные числа;
- ***double*** — вещественные числа удвоенной точности.

**Имя переменной** – набор символов для обращения к значению переменной.

**Объявление переменной** – устанавливает свойства объекта: его тип (например, целый), размер (например, 4 байта) и т.д. Определение наряду с этим вызывает выделение памяти для хранения данных.

```
int a;          char ch;
float my, w;    double lenght;
```

! **Начальное значение не определено**

# Основы языка С. Переменные

**Инициализация переменных** — процесс присвоение первоначального значения

```
int a = 5;
```

```
float b;
b = 0.78;
```

**Варианты присвоения значений**

```
int a, b, c;
a = 15;
b = a;
c = a = b;
```

**Преобразование типов**

```
int a;
float b;
a = (int)b;
```

# Основы языка С. Понятия.

**Оператор** - это наименьшая исполняемая единица программы.

Для обозначения конца оператора в языке Си используется **точка с запятой**

**Блок** - набор логически связанных операторов, помещенных между открывающей **{** и закрывающей **}** фигурными скобками. После блока **;** не ставится

```
{
        int a = 5;
        int b, c;

        c = a + b;
        c = a - b;
        c = a * b;
        c = a / b;
}
```

! ***Объявление переменных происходит в начале блока! (*)***

! ***Оператора возведения в степень (^) в языке Си НЕ СУЩЕСТВУЕТ!***

! ***При делении учитываются типы переменных a и b (*)***

# Основы языка С. Вывод данных.

Функция **printf( )** (прототип содержится в файле stdio.h) обеспечивает форматированный вывод.

**printf ("управляющая строка", аргумент _1, аргумент _2,...);**

Управляющая строка содержит компоненты трех типов:

- ***обычные символы***, которые копируются в стандартный выходной поток (на экран дисплея);
- ***спецификаци***и преобразования;
- ***управляющие символьные константы***.

Каждая ***спецификация*** преобразования начинается со знака % и заканчивается некоторым символом, задающим преобразование:

- **c** - значением аргумента является символ
- **d или i** - значением аргумента является десятичное целое число
- **e** - значением аргумента является вещественное десятичное число в экспоненциальной форме вида 1.23e+2
- **f** - значением аргумента является вещественное десятичное число с плавающей точкой
- **s** - значением аргумента является строка

```
int a = 5;
printf("%d", a);
```

```
float b = 0.755;
printf("%f", b);
```

```
printf("Hello World");
```

# Основы языка С. Вывод данных.

**printf ("управляющая строка", аргумент _1, аргумент _2,...);**

Управляющая строка содержит компоненты трех типов:

- ***обычные символы***, которые копируются в стандартный выходной поток
- ***спецификации*** преобразования
- ***управляющие символьные константы***

Среди ***управляющих символьных констант*** наиболее часто используются следующие:

**\a** - для кратковременной подачи звукового сигнала
**\b** - для перевода курсора влево на одну позицию
**\n** - для перехода на новую строку
**\r** - для возврата каретки
**\t** - горизонтальная табуляция
**\v** - вертикальная табуляция
**** - вывод символа \

```
printf("Hello World\n");
```

```
int a = 5;
printf("\tMy number is\n%d\n", a);
```

# Основы языка С. Ввод данных.

Функция **scanf( )** (прототип содержится в файле stdio.h) обеспечивает форматированный ввод.

**scanf("управляющая строка", аргумент_1, аргумент_2,...);**

***Управляющая строка*** содержит спецификации преобразования и используется для установления количества и типов аргументов. В нее могут включаться:

- пробелы, символы табуляции и перехода на новую строку (все они игнорируются)
- ***спецификации преобразования***, состоящие из знака %, возможно, числа, задающего максимальный размер поля, и самого символа преобразования
- обычные символы, кроме %

***спецификации преобразования***:

- **c** - на входе ожидается появление одиночного символа
- **d** или **i** - на входе ожидается десятичное целое число (тип int)
- **f** - на входе ожидается вещественное число с плавающей точкой
- **s** - на входе ожидается появление строки символов

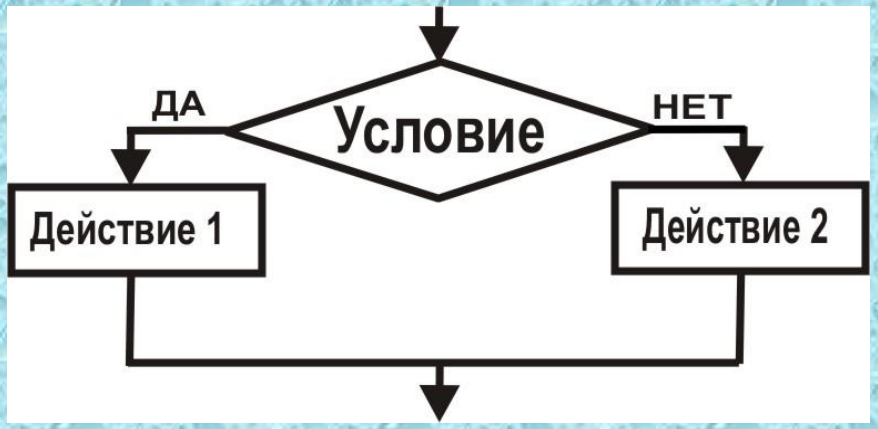
```
int a = 5;
scanf("%d", &a);
```

```
float b = 0.755;
printf("%f", &b);
```

**аргумент = & имя_переменной**

# Основы языка С. Условный оператор

**if (проверка_условия) оператор_1; else оператор_2;**




***Основные условные операторы*:**

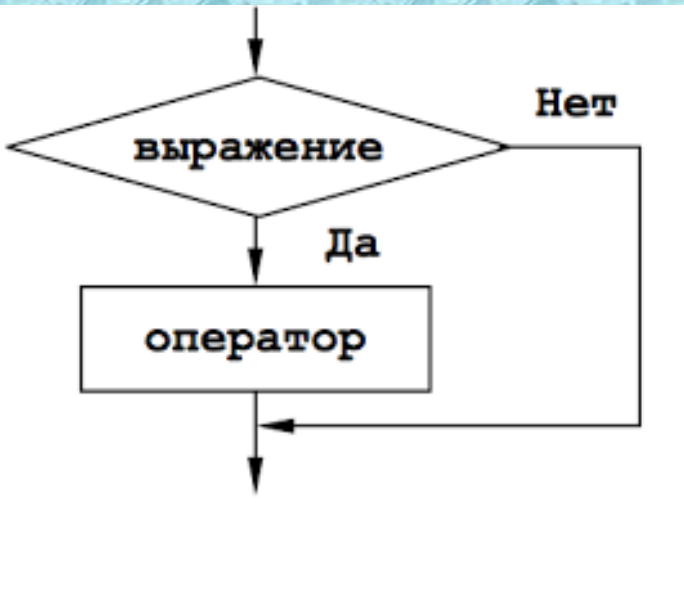
| | | |
|---|---|---|
| > | >= | == |
| < | <= | != |

**Форматирование кода – залог хорошей программы!**

```
if(a>b) c=a; else c = b;
```

```
if(a>b)
        c=a;
else
        c = b;
```

# Основы языка С. Условный оператор




```
if (проверка_условия)
        оператор_1
```

***Печать модуля введенного числа***

```
void main()
{
        int a;
        printf(“Input number”);
        scanf(“%d”, &a);
        if(a<0)
                a = a * (-1);
        printf(“Module = %d”, a);
}
```

***Корректные запросы и вывод информации - правило хорошего тона***

# Основы языка С. Переключатель

оператор **switch** используется для организации мультиветвления

```
switch (выражение)
{
        case Константа_1: операторы_1;
        case Константа_2: операторы_2;
        ….
        default: операторы;
}
```

При первом совпадении значения выражения с константой происходит выполнение операторов, помеченных данной меткой.
Если после их выполнения не предусмотрено никаких операторов перехода, то выполняются также все последующие операторы. То есть по сути, CASE является меткой, обозначающей место выполнения программы после SWITCH.

Оператором перехода может быть **break**, который осуществляет выход из тела (фрагмент кода, обозначенный фигурными скобками).
Переключатели чаще всего используются, когда количество ветвей больше 3, либо необходимо перебрать заданные константы.

# Основы языка С. Переключатель

```
#include <stdio.h>
void main()
{
        int a;
        printf("Input a = ");
        scanf("%d", &a);
        switch (a)
        {
                case 1: printf("%d - One\n", a);          break;
                case 2: printf("%d - Two\n", a);          break;
                case 3: printf("%d - Three\n", a);        break;
                case 4: printf("%d - Four\n", a);         break;
                case 5: printf("%d - Five\n", a);         break;
                case 6: printf("%d - Six\n", a);          break;
                case 7: printf("%d - Seven\n", a);        break;
                case 8: printf("%d - Eight\n", a);        break;
                case 9: printf("%d - Nine\n", a);         break;
                case 0: printf("%d - Zero\n", a);         break;
                default: printf("The value is not from 0 to 9");
        }
}
```

# Основы языка С. Циклы

Оператор **for** является, пожалуй базовым оператором для организации циклов. Цикл **for** — это параметрический цикл, то есть имеется возможность задавать параметры для выполнения цикла: начальные значение и условия.

***for (выражение_1; условие_цикла; выражение_2)***
***Тело цикла***

***выражение_1*** – определяет действия, выполняемые до начала цикла.
***условие_цикла*** – обычно логическое или арифметическое условие. Пока это условие истинно, выполняется цикл. Проверка происходит перед началом очередной итерации тела цикла.
***выражение_2*** — это действие, выполняемые в конце каждой итерации цикла.

```
int i, Max;
printf("Input Max = ");
scanf("%d", &Max);

for(i = 0; i <= Max; i++)
{
        if(i%2 == 0)
        printf("%d\t", i);
}
```

# Основы языка С. Циклы

***for (выражение_1; условие_цикла; выражение_2)***
***Тело цикла***

*Выражения в скобках оператора **for** не являются обязательными.*

```
int i, Max;
printf("Input Max = ");
scanf("%d", &Max);

for(i = 0; i <= Max; i++)
{
        if(i%2 == 0)
        printf("%d\t", i);
}
```

```
int i, Max;
printf("Input Max = ");
scanf("%d", &Max);

i = 0;
for(;i <= Max;)
{
        if(i%2 == 0)
        printf("%d\t", i);
        i++;
}
```

# Основы языка С. Циклы

*while (выражение условия цикла)*
*Тело цикла*

```
int i, Max;
printf("Input Max = ");
scanf("%d", &Max);

i = 0;
while(i <= Max)
{
        if(i%2 == 0)
        printf("%d\t", i);
        i++;
}
```

*do*
*Тело цикла*
*while (выражение условия цикла);*

```
int i, Max;
printf("Input Max = ");
scanf("%d", &Max);

i = 0;

do
{
        if(i%2 == 0)
        printf("%d\t", i);
        i++;
}
while(i <= Max);
```

# Форматирование кода

- Множество общепринятых стилей
  - K&R, Allman, Whitesmiths, GNU,…;
  - важно следовать единому стилю на протяжении всей программы
- Выравнивание
  - вертикальное;
  - длина строки –<80 символов
- Отступы
  - Вертикальные и горизонтальные
  - Каждый вложенный блок кода имеет отступ;
  - табуляция;
  - Выделение логических блоков пустыми строками
- Фигурные скобки
  - Перенос открывающейся скобки
  - Скобки требуют отдельных строк
- Пробелы
  - Помощь запятым int func(int a, int b, int c);
  - Разрежение математики (a + b)* (c –d)