Лабораторная работа №1

Тема: Изучение базовых понятий программирования и технологии создания программ. Программирование ветвящихся и циклических алгоритмов.

Цель работы:

1. Освоение построения алгоритмов линейной структуры и простейшего диалогового интерфейса.
2. Изучение логических операторов и операторов отношения, а также операторов ветвления и выбора. Получение навыков построения ветвящихся алгоритмов.
3. Получение навыков построения циклических алгоритмов.

Лабораторное задание

1. Изучить теоретическую часть лабораторной работы :
   - этапы трансляции программы
   - структура программы на языке Си.
   - директивы препроцессора #include и #define
   - встроенные типы данных;
   - операторы ветвления: условный оператор if, оператор выбора swich
   - операторы цикла: while, do-while, for
   - операторы break, continue
   - изучить функции ввода-вывода библиотеки stdio: printf и scanf
2. Разобрать примеры программирования и выполнить их на компьютере.
3. Выполнить три задания (номер варианта соответствует номеру компьютера).
   - Разработать и записать алгоритмы решения
   - Написать программы, реализующие разработанные алгоритмы, используя, операторы присваивания, if, switch, while, do-while, for и ввода-вывода данных.

Теоретическая часть

Действие – основное понятие программирования, оно должно приводить к желаемому результату. Действие всегда совершается над неким объектом (данными), в результате чего объект меняет свое состояние. В языках программирования описание действия называется *инструкцией,* а последовательность инструкций называется *программой*.

Алгоритм – это конечная последовательность точно определенных элементарных действий для решения поставленной задачи при всех допустимых вариантах исходных условий задачи.

Алгоритм, программа и данные - все три понятия тесно связаны между собой и играют в программировании важнейшую роль, упрощенно эту связь можно представить следующим образом :

Программа = Алгоритм + Данные

Алгоритм решения задачи

В основе решения любой задачи лежит построение алгоритма. Разработка алгоритма для компьютера включает в себя выделение этапов обработки данных и представления их в определенной форме, например в виде блок-схемы алгоритма. *Блок-схема - это графическое представление алгоритма программы с использованием стандартных графических элементов (прямоугольников, ромбов, трапеций и др.), обозначающих команды, действия, данные и т. п.*

Итак, для построения алгоритма этапы обработки данных необходимо представить в виде графических элементов (геометрических фигур или блоков) и соединить их линиями передачи управления. В блоках следует записывать последовательность действий.

Разработка алгоритма – один из важнейших этапов решения задачи, помогающий с одной стороны лучше понять поставленную задачу и с другой стороны наметить пути её решения. Стандартные формы геометрических фигур, используемых в блок-схемах, приведены на рис 1

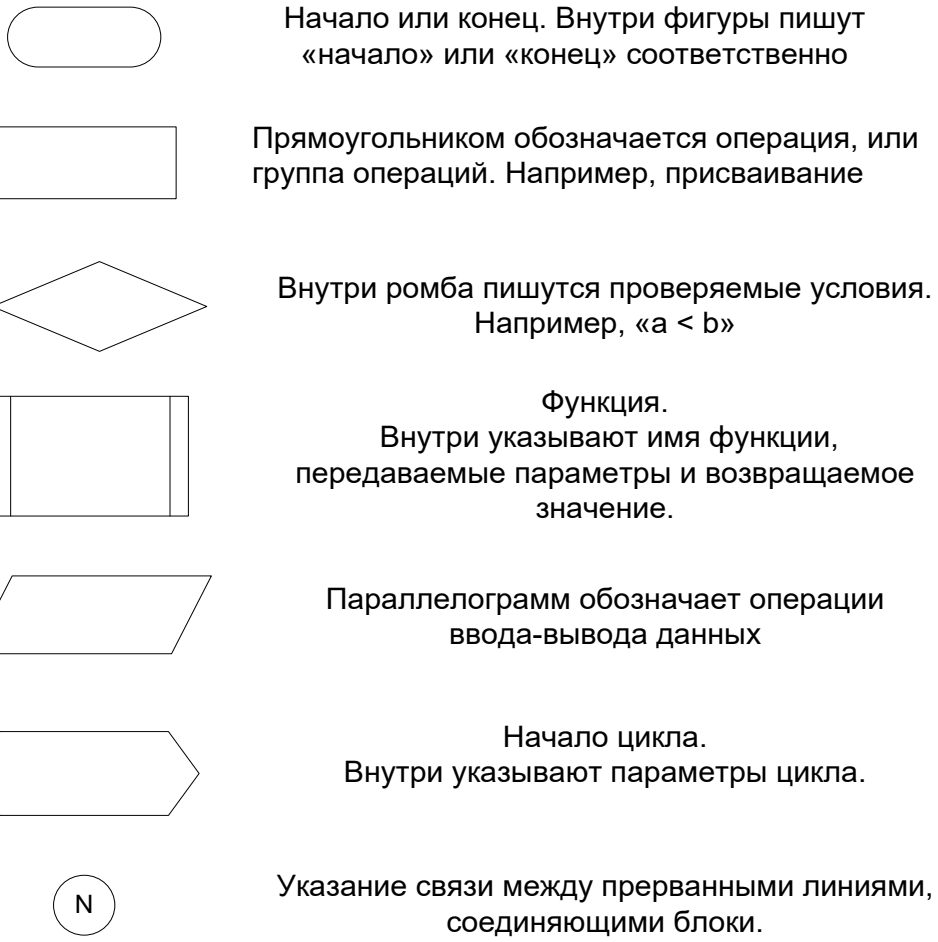


Рис. 1 Элементы блок-схемы

Этапы трансляции программы.

Превращение текста на языке высокого уровня в машинный код проходит в несколько этапов:
На первом этапе происходит препроцессорная обработка текста.
На втором этапе создается промежуточный (объектный) файл.
На третьем этапе несколько объектных файлов компонуются в единый исполняемый файл, который может быть загружен в память компьютера и выполнен.
Библиотечные файлы хранятся в объектном виде и присоединяются к программе пользователя на этапе компоновки. Ход трансляции приведен на рис. 2

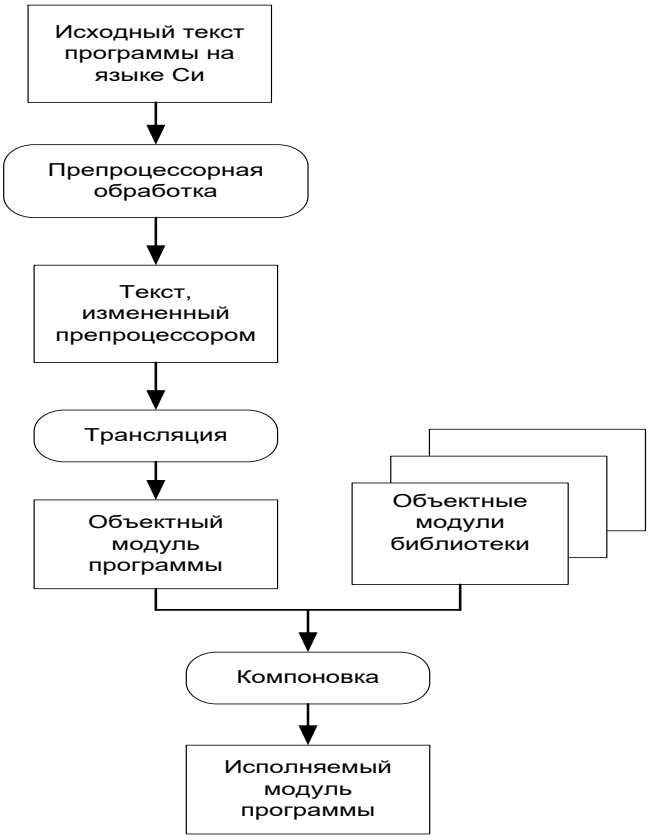


Рис. 2. Этапы трансляции текста программы.

После того, как программа оттранслирована, её можно выполнить, для чего используется специальная программа, называемая *загрузчиком.*

Препроцессор языка Си

Для того, чтобы исполнить программу на компьютере её нужно разместить в его памяти, но единственный язык, напрямую выполняемый процессором компьютера — это машинный код. Перевод исходного текста на языке высокого уровня, написанного программистом (текст программы), в машинный код выполняют специальные программы, называемые трансляторами или компиляторами, а процесс превращения исходного текста программы в машинные коды называется компиляцией.

Препроцессор - это составная часть компилятора языка С, реализующая *изменение текста программы перед компиляцией*. В рамках данной лабораторной работы рассмотрим две важные возможности препроцессора:

- включение файлов в текст программы (директива **#include**);
- изменение текста программы с помощью макроподстановок (директива **#define**).

Для того, чтобы обратиться к препроцессору, необходимо включить в текст программы специальные команды, или директивы препроцессора, начинающиеся со знака #. После директив препроцессора точка с запятой не ставятся.

Директива #include

#include – директива включения файла в исходный текст программы.
Имеется две её формы:
#include "имя файла"
#include <имя файла>

Если имя файла указано в кавычках, то поиск файла осуществляется в соответствии с заданным маршрутом, а при его отсутствии в текущем каталоге, то есть в том же каталоге, что и текст программы.

Если имя файла задано в угловых скобках, то поиск файла производится в стандартных директориях операционной системы, задаваемых командой PATH (например, в каталоге файлов включений, прописанном в среде программирования).

Директива #include широко используется для включения в программу так называемых заголовочных файлов, содержащих прототипы библиотечных функций, и поэтому большинство программ на С начинаются с этой директивы.

Например #include <stdio.h> включает в программу пользователя заголовочный файл стандартной библиотеки ввода-вывода языка С.

Директива # define.

#define служит для замены часто использующихся констант, ключевых слов, операторов или выражений некоторыми идентификаторами. Например, идентификаторы, заменяющие текстовые или числовые константы, называются именованными константами и могут быть заданы с помощью директивы #define.

Формат директивы:
#define идентификатор текст

Эта директива заменяет все вхождения идентификатора на «текст». Текст может представлять собой любой фрагмент программы на С, а также может и отсутствовать. В последнем случае все экземпляры идентификатора удаляются из программы. Проводя аналогию с текстовым редактором, действия директивы #define аналогичны «поиску с заменой».

Пример:
```
#define QU 40
#define WEWQ (QU+15)
```

Эти директивы изменят в тексте программы каждое слово QU на число 40, а каждое слово WEWQ на выражение (40+15) вместе с окружающими его скобками. При выполнении данной директивы препроцессора исключение составляют *текстовые константы, заключенные в двойные кавычки*, внутри которых подстановки не производятся. Например, внутри строки "В данной строке QU записано один раз" слово QU изменяться не будет.

Структура программы на языке Си

Любая программа на языке Си состоит из одной или более функций.
Одна из этих функций (главная) должна иметь имя **main()** Отличительным признаком функции служат круглые скобки, а аргумент может и отсутствовать. Тело функции заключено в фигурные скобки и представляет собой набор операторов, каждый из которых оканчивается символом "точка с запятой".

При запуске программы пользователя, операционная система передает управление на функцию **main()** и тем самым начинается выполнение программы. От других функций существующих в программе функция **main()** отличается тем, что её нельзя вызвать изнутри программы, а ее параметры, если они есть, задаются операционной системой. Обычно, хотя это и не обязательно, **main()** бывает первой функцией программы.
Область директив препроцессора находится перед функцией main()

Пример :

```
/* Простейшая программа, выводящая приветствие на экран дисплея */
#include <stdio.h> // директива препроцессора
void main()
{
  printf("Hello, world\n");
}
```

Разбор программы :
Первая строка : /* Простейшая программа, выводящая приветствие на экран дисплея */ является *комментарием*, который в данном случае кратко объясняет, что делает программа. Все символы, помещенные между /* и */, игнорируются компилятором, и этим можно свободно пользоваться, чтобы сделать программу более понятной.

Инструкция printf("Hello, world\n");- это вызов функции **printf**, которая выполняет печать своего первого аргумента - текстовой строки. Функция **printf** находится в стандартной библиотеке **stdio.h** Результатом работы программа явится сообщение :

Hello, world

Рассмотрим подробнее функцию **printf()**, ей можно передать любое количество параметров, причем первый параметр обязательно должен быть текстовой строкой, описывающей формат вывода. При вызове функция печатает строку, стоящую первой. Если в этой строке встречаются специальные комбинации символов, начинающиеся с символа **%** , функция подставляет вместо них значения последующих параметров.
Приведем наиболее часто используемые комбинации:

%s	- печать текстовой строки
%c	- печать отдельного символа
%d, %i	- печать целого числа
%f, %e, %l	- печать вещественного (дробного) числа

Например, **з**апишем инструкцию, которая выводит в одной строке значения переменных a, b и c целого типа (int), в качестве разделителя между переменными будем использовать знак «:»

```
printf(" %d:%d:%d",a,c,b);
```

Этот оператор имеет 4 параметра, разделенных запятыми. Первый параметр – это текстовая строка, определяющая формат (форму) и типы данных, выводимых на экран.
Второй, третий и четвертый параметры – это имена переменных, выводимых на экран.
Если значения переменных a,b и c соответственно равны 5,10,25, то результат на экране дисплея выглядит следующим образом:
5:25:10

Функция **printf()** позволяет не только выводить любые данные (как числовые, так и текстовые), но и форматировать их, например, снабжать числовую информацию текстовыми комментариями, переводить строки, делать отступы и тому подобное.
Например, предыдущий пример можно отформатировать следующим образом:

```
printf("\na= %d:\tb= %d:\tc= %d\n",a,b,c);
```

Результат на экране дисплея выглядит следующим образом:
a=5: b=10: c=25

Функция **printf()** «перевела строку» (символ **\n**), перед выводом числа поставила комментарий «a=», после каждого числа вывела знак «табуляции» (символ **\t**).

Пассивная часть программы - данные : переменные и константы.

Любой программе необходимо работать с данными, с объектами, которые несут в себе информацию. Некоторые данные устанавливаются равными определенным значениям еще до того, как программа начинает выполняться, а после ее запуска сохраняются неизменными на всем протяжении работы программ. Такие данные называются *константами*.

Данные, которые могут изменяться, или же им могут быть присвоены значения во время выполнения программы, называются *переменными.* Различие между переменной и константой очевидно: во время выполнения программы значение переменной может быть изменено, значение же константы неизменно.
Кроме различия между переменными и константами существует различие между *типами* данных.

Тип данных определяет множество значений объекта программы плюс множество операций, которые можно выполнять над этими объектами.

Проверка выполнения требований, накладываемых типом, осуществляется на этапе компиляции. Например, ошибочное присваивание числовой переменной логического значения можно выявить без выполнения программы. Другой пример: операция сложения определена для вещественных и целых типов, но не определена для логического типа.

В языке Си предусмотрено использование основных (встроенных) типов данных, для этой цели используется семь ключевых слов:
int , long, short, double, double, char, bool
и два модификатора:
signed (со знаком), unsigned (без знака)

Классификация типов данных представлена на рис. 3

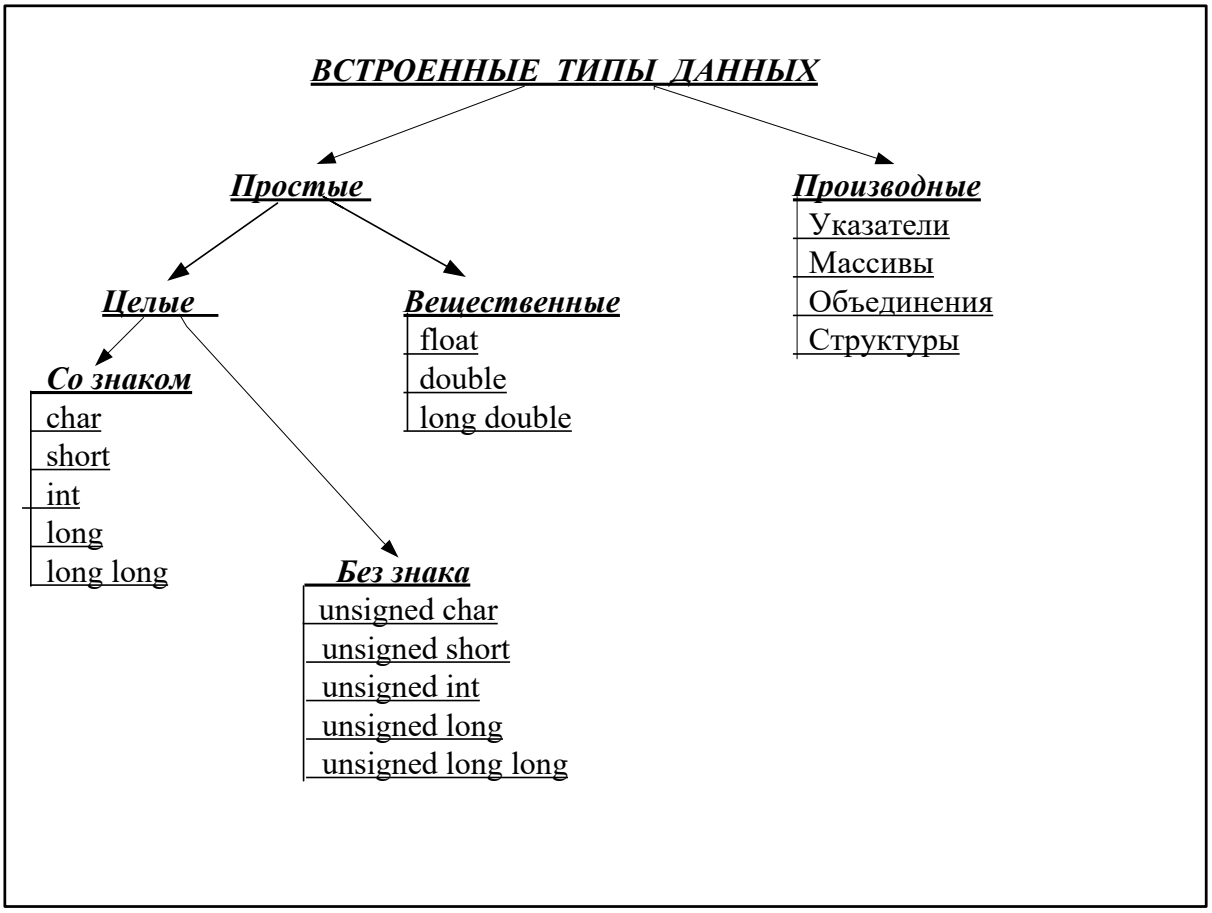


Рис. 3 Встроенные типы данных. Классификация.

При объявлении переменной задается тип данных, который определяет объем выделяемой памяти и допустимые операции. На рис 4 представлена информация о диапазоне значений и размере занимаемой памяти для встроенных типов данных.

Тип данных	Память (в битах)	Диапазон Значений
bool	8	True, false
char	8	-128…127
unsigned char	8	0…255
short	16	-32768…32767
unsigned short	16	0…65535
int, long	32	-2 147 483 648…2 147 483 647
unsigned int, long	32	0…4 294 967 295
double	32	3.4E-38…3.4E+38
double	64	1.7E-308… 1.7E+308
long double	80	3.4E-4932…1.1E+4932

Рис 4 Встроенные типы данных. Размещение в памяти.

Обратите внимание на отличие типов ***short*** и ***unsigned short -*** данные занимают одинаковую область памяти, но имеют различные диапазоны значений.
О языке Си говорят, что он строго типизирован. Программист должен описать данные, явно указывая их типы. Объявляя переменную, следует задать ее тип, при этом происходит выделение необходимого объема памяти (говорят, что переменная размещается в памяти).
Например :

```
double var;        // переменная var - вещественного типа
int rez      ;     // переменная rez - целого типа
```

Язык Си позволяет одновременное объявление переменной и присваивание ей начального значения, в этом случае говорят, что переменная ***инициализирована.***

При инициализации происходит одновременное выделение памяти и присваивание ей начального значения, в нашем примере запись нуля в память, выделенную для переменной **r** целого типа.

Пример :

```
int r = 0;          // переменная r инициализирована,
double s;           // переменная s- только объявлена (выделена память)
```

Для константы тип можно задать как явно, указав его при описании, так и неявно.

Явное описание констант:

```
const int    B=10;      //целая константа
const double C=10;      //      вещественная константа (10.0)
```

Неявное задание основано на том, что компилятор может распознать тип константы по её внешнему виду, например, наличие десятичной точки говорит о том, что константа вещественная (дробное число), а её отсутствие указывает на то, что константа целая.

Например :

```
3.      // вещественная константа
3       // целая константа.
```

Добавление после константы знака L (или l) заставляет компилятор выделять под переменную двойной размер памяти, например:

```
// определение константы двойной точности
 3.141 592 653 589 793 2L
// определение константы обычной точности
 3.141 592
```

Активная часть программы. Выражения.

С точки зрения языка программирования выражение состоит из *операндов* (переменных, констант, функций), соединенных *знаками операций*.

Знак операции - это символ или группа символов, сообщающих компилятору о необходимости выполнения арифметических, логических или других действий.

Например: **a+b, a*10, a / sin(x)**

a, b,10, sin(x) - операнды

\+ * / - операции

Операции выполняются в строгой последовательности, определяемой их приоритетом, но порядок выполнения операций может быть изменен с помощью круглых скобок.

Все операции делятся на три группы:

- арифметические,
- логические и операции отношения.
- операции с битами.

Арифметические операции

Унарные арифметические операции (операции с одним операндом) :

++	инкремент (увеличение)
--	декремент (уменьшение)
-	изменение знака операнда

Бинарные арифметические операции (операции с двумя операндами):

+	сложение
-	вычитание
*	умножение
/	деление
%	деление по модулю (остаток от деления)

Примеры арифметических выражений :

X = A + B
W = (D*10+C)/(K+1)
A = sin(b)+10
X = 5 + 3 * 2

Выражение **X = 5 + 3 * 2** включает три операции * + и = (операции перечислены в соответствии с их приоритетами). Сначала вычисляется выражение в правой части (3*2)+5, а затем результат присваивается переменной в левой части (X=11).

Оператор присваивания.

Для работы с переменными, нужно научиться заносить в них различные значения. Как это делается при создании переменных с помощью знака «=» (инициализация), мы уже знаем. Но часто в уже созданную переменную нужно занести её значение, этом случае говорят записать значение.

Пример:

```
int p1, p2, p3;
p1=5;              // записать в переменную p1 константу 5
p2=p1;             // записать в переменную p2 значение из p1
p3 = p1+10;        // записать в переменную p3 результат вычислений
```

В последнем примере сначала вычисляется значение выражение p1+10, а затем результат присваивается операнду p3.

Следует отметить, что левый операнд оператора присваивания всегда должен быть *адресным выражением*, то есть объектом, размещенным в памяти компьютера. Примером адресного выражения является имя переменной. Примером не адресного выражения является, выражение a+b, а также константы, которые могут стоять только справа от символа присваивания.

Примеры неправильного использования оператора присваивания :

5 = p1;
p1 + p2 = p3;

Язык Си позволяет выполнять множественное присваивание, например можно написать такую конструкцию: var = p3 = 7;
Операторы присваивания обрабатываются справа налево, читать эту строку следует так: "записать в **p3** значение **7**, затем записать в var значение из **p3**".

Существует так называемая комбинированная операция присваивания, которая позволяет вместо выражения **x =x+a**, записать **x += a**
Вместо операции «+» может стоять любая их следующих бинарных операций:

+ - * / %

Примеры :

```
a += 2;      // эквивалентно    a = a+2
s *= a;      // эквивалентно    s = s*a
```

Преобразование типов в выражениях

Язык Си допускает выражения с операндами различных типов. Например, можно делить переменную типа **double** на константу типа **int,** прибавлять константу типа **int** к переменной типа **char** и тому подобное. Если выражение имеет операнды различных типов, более низкий тип всегда преобразуется к более высокому. Один тип ниже другого, если он занимает меньше памяти, например, целый тип ниже вещественного.

Пример:

```
double x,y;
char ch;
long in;
int i;
....
y = x*(i+ch/in);
```

В выражении **y = x*(i+ch/in)** преобразования типов данных будут выполняться в следующей последовательности:

- Операнд **ch** преобразуется к типу **long** (к большему операнду выражения ch/in), по этой же причине **i** преобразуется к **long**
- Результат операции, заключенной в круглые скобки будет иметь тип **long**.
- Затем он преобразуется к типу **double** , к большему операнду выражения **x*(i+ch/in)**
- Результат всего выражения будет иметь тип **double**

До сих пор преобразование типов выполнялось неявно, то есть по определенным правилам, заданным языком Си и реализованным в компиляторе. Но можно и явно указать какое преобразование нужно выполнить. Явное приведение значения одного типа к другому выполняется с помощью специальной операции. Допускается даже преобразование к типу данных с меньшим объемом памяти, но следует помнить, что в этом случае будет потеряна часть информации. Формат операции приведения типа: ***(имя_типа) операнд***

Операндом может быть :

- Переменная
- Константа
- Выражение в круглых скобках

Примеры явного преобразования переменных:

```
...
int a,b,c;
double x;
/* результат правой части выражения – целое, поэтому результат будет округлен до ближайшего целого (потеря дробной части числа)*/
x=(a+b)/c;
/* Следующие действия позволяют избежать потери результата */
x = ( (double)a + b) / c;
x = (double)(a+b) / c;
x = (a+b) / (double)c;
```

Логические операции и операции отношения

Наряду с арифметическими операциями, которые используются для всевозможных вычислений, в языках программирования есть и логические операции, которые используются для проверки условий. Логические операции называют операциями отношения, значение переменной или константы они сравнивают с литералом, или со значением другой переменной или константы.

Результат сравнения имеет логический тип (**bool**) :

true (истина или 1) либо false (ложь или 0).

Рассмотрим подробнее операции отношения, к ним относятся:

>	**больше;**
<	**меньше;**
>=	**больше или равно;**
<=	**меньше или равно;**
= =	**равно (проверка на равенство)**
!=	**не равно.**

Операции отношения по рангу младше арифметических операций, так что выражения типа :

i < lim+3 понимаются как **i < (lim+3)**

Приведем примеры проверки простых логических условий :

```
(i>7)       // результат 1(true,да) если i больше 7, и 0 (false,нет) – в противном случае
(i==j)      //результат 1 если i равно j
(x+1 != k)//результат 1 если x+1 не равно k
```

Чаще всего ошибки совершают при проверке на равенство, обратите внимание, что в этом случае необходимо ставить два знака «=»,

выражение подобное if (i=j) неверно!

но компилятор подобные ошибки не диагностирует , так как интерпретирует данное выражение следующим образом : if ((i=j)!=0), то есть сначала заносит значение j в переменную i, а затем сравнивает результат с нулем.

Если нам необходимо проверить сложное условие, то есть объединить несколько простых логических выражений в единое сложное выражение, то понадобятся логические связки, так называемые логические операции.

К логическим операциям относятся:

&&	логическое И (конъюнкция), бинарная операция;
||	логическое ИЛИ (дизъюнкция), бинарная операция.
!	логическое НЕ (отрицание), унарная операция;

Так же как и у операций отношения, у логических операций результат логический (бинарный), либо 1 (true) либо 0 (false)

Примеры применения логических связок (сложных условий) :

Если a <b, то :

Чтобы записать логическое условие, соответствующее следующей математической записи **a<i<b**, потребуется логическая связка **&&** :

(i>a && i< b)

Чтобы записать логическое условие, соответствующее следующей математической записи **a>i>b**, потребуется логическая связка || :

(i<a || i> b)

Условный оператор if

Когда линейную часть программы необходимо разделить на две или более ветви (альтернативы) используют операторы условных и безусловных переходов. Это структурные операторы языка, которые позволяют менять порядок выполнения программы в зависимости от некоторых условий. Ветвление осуществляется путем анализа некоторого логического выражения, задающего условие выбора альтернативы.
Оператор if позволяет разветвить вычислительный процесс на два варианта в зависимости от значения некоторого условия.
Имеется две формы условного оператора:

- Полная схема выполнения : if **(выражение) оператор1** else **оператор2;**
- Сокращенная схема выполнения : if **(выражение) оператор1;**

Выполнение оператора if начинается с вычисления выражения в скобках.
Далее действует следующая схема:

1. если **(выражение)** имеет значение true (отлично от 0), то выполняется **оператор1**
2. если **(выражение)** имеет значение false (равно 0), то выполняется **оператор2**
3. если **(выражение)** имеет значение false и отсутствует конструкция со словом else, то выполнение оператора if завершается.

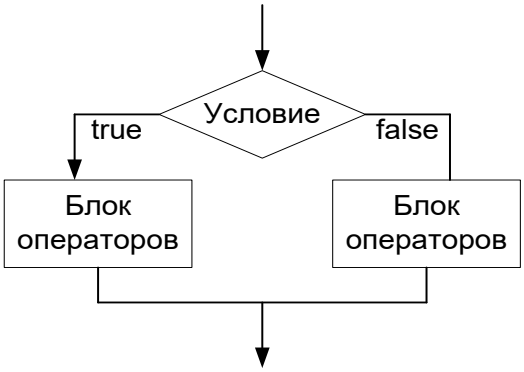


Рис 5 Полная блок-схема оператора if

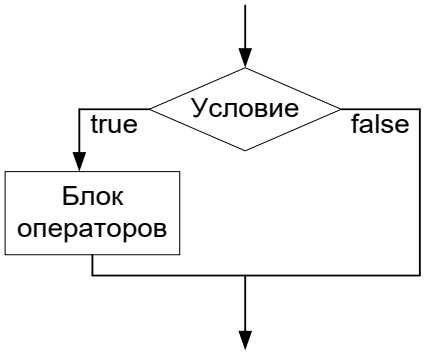


Рис 6 Сокращенная блок-схема оператора if

Примеры:

```
if (a > b) c = a - b; // выполняется, если a больше b
else c = b - a;                 // выполняется, если a меньше b
... ...
if (i < j) i++;
else {j = i - 3; i ++;} // блок операторов выполняется, если i больше j
```

Оператор выбора switch (селективный оператор).

Если в программе используются больше трех возможных вариантов, то следует использовать оператор switch Его часто называют селективным оператором, переключателем или оператором выбора. Оператор switch передает управление одному из нескольких помеченных специальными метками операторов в зависимости от значения целочисленного выражения. Специальные метки начинаются с ключевого слова case и являются целочисленными константами.

Оператор имеет следующий вид:

```
switch (целое_выражение )
{       [объявления]
        [case константное_выражение1:]
        [операторы группы_1]
        [case константное_выражение2: ]
        [операторы группы_2]

        [case константное_выражение n:]
        [операторы группы_n]
        [default:] [операторы default]
}
```

Результат выражения в круглых скобках должен быть целым. Оператор switch рассматривает значение выражения не как логический результат true либо false, а как "шаблон" для выбора подходящего варианта из заданного списка.

Схема выполнения оператора switch:

- Вычисляется выражение в круглых скобках (назовем его **селектором**).
- Значение селектора последовательно сравнивается с константными выражениями, записанными после ключевого слова case (case-метка), **е**сли селектор и case-метка равны, то управление передается оператору, помеченному данной меткой
- Если селектор не совпадает ни с одной меткой, то управление передается на оператор, помеченный словом default**.**
- Если default отсутствует, то управление передается следующему за switch оператору.

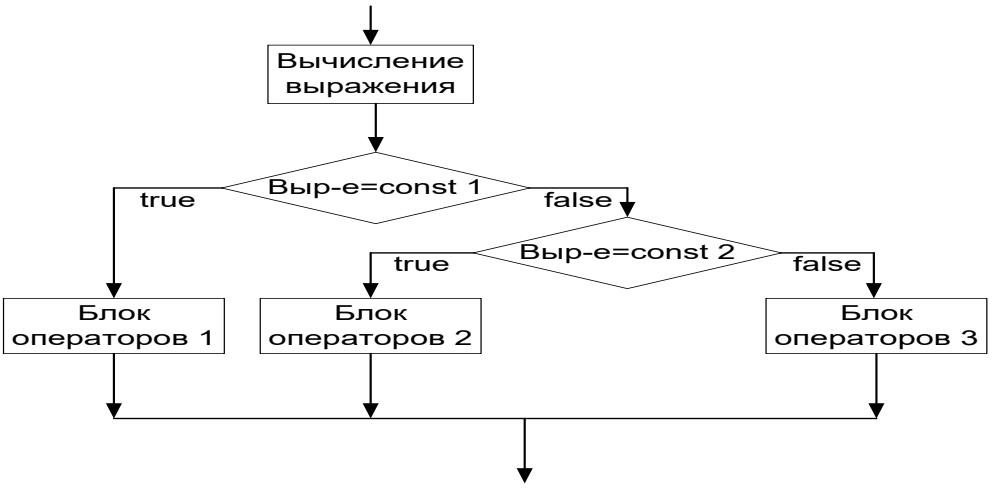


Рис 7. Блок-схема селективного оператора.

Пример: В качестве селектора выступает переменная x

```
int x;
printf ("x="); scanf ("%d",&x); // ввод числа с клавиатуры
switch (x)
{case 4:      printf("x равно 4\n");
             break;
  case 2:
  case 10:    printf("x равно 2 или 10");
             break;
default:     printf("x не равно 4, 2, 10\n");
}
```

В примере анализируется значение переменной x, и в зависимости от результата, передается управление на одну из ветвей оператора switch. Два варианта (x=2 и x=10) объединены, если x не равен 2, 4 или 10, то выполняется ветка default.
У всех вариантов выбора, кроме case **2:** наборы операторов заканчиваются ключевым словом break (у case **2:** вообще нет "своих" операторов). Дело в том, что оператор switch, выбрав один из своих вариантов, не заканчивает своё выполнение автоматически, дойдя до следующего варианта - он исполняет все операторы, идущие дальше, вплоть до конца всего блока, заканчивающегося закрывающейся «фигурной скобкой». Наличие ключевого слова break принудительно заканчивает каждую альтернативу и передает управление за пределы оператора swich. Если мы уберем, например, первый break, то при **x==4** увидим сначала сообщение от первого **printf(),** а затем и от второго.

Тернарная операция ветвления.

Тернарная операция имеет три операнда, её форма :

выражение1 ? выражение2 : выражение3

Последовательность действий следующая:

- выражение1 сравнивается с нулем,
- если выражение1 не равно нулю, то вычисляется **выражение**2 и его значение является результатом операции.
- если выражение1 равно нулю, то вычисляется **выражение3**, и его значение является результатом операции.

Заметим, что вычисляется один из операндов после знака ?, но не оба.

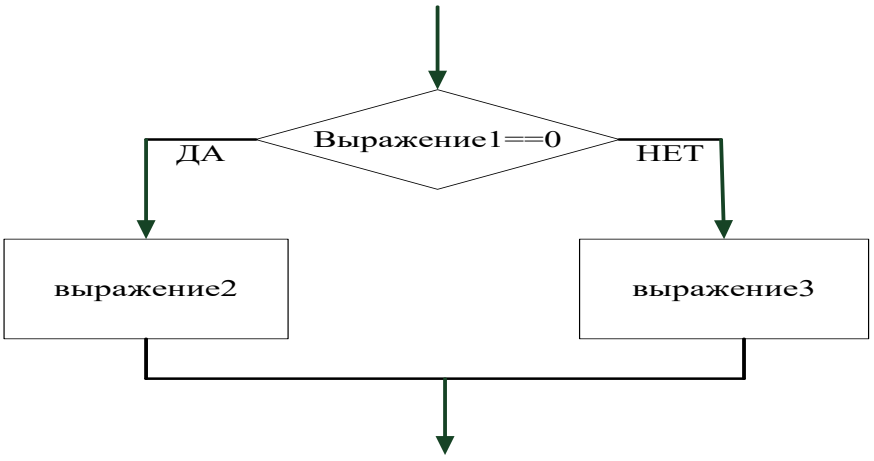


Рис 8. Блок-схема тернарного (условного) оператора.

Пример: Переменной max присваивается максимальное значение из двух переменных a и b.

```
max = a<=b ?  b : a;
```

Согласно общим правилам языка Си сначала будет выполняться выражение справа от знака присваивания, то есть тернарный оператор **a<=b ? b : a** Результатом этого оператора будет максимальное значение из **b, a**, которое и будет занесено в переменную **max**.

Если в условной операции выражение2 и выражение 3 являются адресными выражениями, то тернарная операция может стоять слева от знака присваивания:

Пример:
a< b? a:b = c*x+d;

В данном примере значение выражения **c*x+d** присваивается меньшей из переменных **a , b**, если **a<b**, то вычисляется выражение **a = c*x+d**, иначе **b = c*x+d**

В заключении приведем таблицу приоритетов операций языка С. Операции расположены по убыванию их приоритетов. Самый высокий приоритет имеют операции **( ) [ ] -> .** , а самый низкий – операция «запятая»

Вид операции	Знак операции	Ассоциативность
Первичные	( ) [ ] -> .	слева направо
Унарные	! ~ - + ++ -- (type) * & sizeof	справа налево
Мультипликативные	* / %	слева направо
Аддитивные	+ -	слева направо
Сдвиги	<< >>	слева направо
Отношения	< <= > >= == !=	слева направо
Побитовые	& ^ |	слева направо
Логические	&& ||	слева направо
Тернарная (условная)	? :	справа налево
Присваивание	=	справа налево
Запятая	,	слева направо

Операторы цикла

При программировании часто возникает потребность выполнить одни и те же действия, с разными исходными данными. Конечно, если вы знаете заранее, сколько раз надо повторить те или иные операции, и таких повторов немного (например, программа всегда обрабатывает два набора данных), можно просто написать два раза соответствующие операторы. Но чаще оказывается, что повторов может быть много, и к тому же нельзя сказать заранее, сколько именно их будет. Чтобы решить эту проблему, программист организует цикл - многократное повторение какого-то участка кода (каждый такой повтор называют итерацией), при этом число повторений в той или иной форме задает условие выхода из цикла.

В языке Си имеются три оператора цикла - это, while, do-while и for. Кроме того, специально для расширения возможностей этих трех операторов, предусмотрены еще 2 специальных оператора - continue и break.
Оператор continue прерывает текущую и передает управление на следующую итерацию цикла, отсекая операторы, следующими за ним (досрочный переход на следующую итерацию).
Оператор break используется для прерывания текущей итерации и выхода из цикла (досрочный выход из цикла).
break также применяется в операторе switch, но суть действия его одинакова в обоих случаях – завершение текущего оператора.

Цикл с предусловием (оператор while)

Оператор цикла while называется циклом с предусловием, поскольку условие проверяется перед входом в цикл:

```
while (выражение) тело цикла ;
```

В качестве выражения допускается использовать любое выражение языка Си, а в качестве тела любой оператор, в том числе пустой или составной.

Последовательность действий оператора while :

1. Вычисляется выражение в скобках, затем проводится его проверка.
2. Если выражение ложно, то выполнение оператора while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполняется блок операторов (тело цикла while).
3. Процесс повторяется с пункта 1.

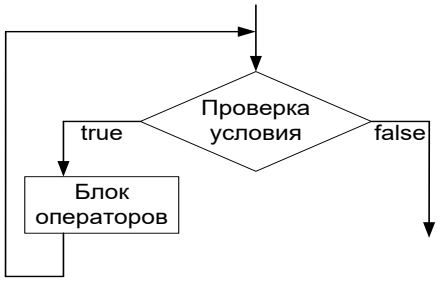


Рис 9. Блок-схема цикла с предусловием.

Таким образом, блок операторов, следующий за while будет выполняться пока проверка условия дает результат отличный от нуля, true (истина). Разумеется, после выхода по условию выполнение программы продолжится с оператора, следующего за циклом.
Рассмотрим следующий фрагмент программы:

```
int x=0;
while (x<10)
{       printf("%d\n",x);
        x++;
}
printf("The end ");
```

Такой цикл выполнится 10 раз, напечатав значения **x** от **0** до **9**, и только после этого управление перейдет на второй, стоящий за циклом **printf()**.

Цикл с постусловием (Оператор do … while)

По поведению он очень похож на предыдущий цикл **while()**, за тем исключением, что условие проверяется после выполнения тела цикла. Если условие окажется ложным, цикл на этом заканчивается, в противном случае – выполняется очередная итерация цикла. В этом цикле одна итерация выполняется при любых условиях. Формат оператора имеет следующий вид:
do тело цикла while (выражение);

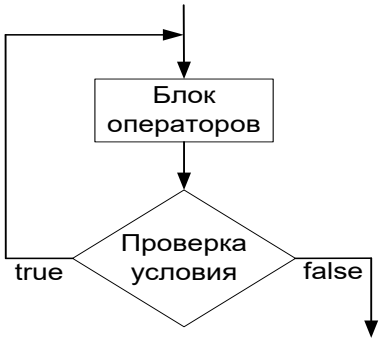


Рис 10. Блок-схема цикла с постусловием

Чтобы прервать выполнение цикла до того, как условие станет ложным, можно использовать оператор break. Напечатаем значения **x** от **0** до **9** с использованием цикла do ... while

```
int x=0;
do
{  printf("%d\n", x);
    x++;
} while (x<10);
printf("The end");
```

Пример: Этот цикл выполнится один раз, прежде чем определит, что x равно нулю.

```
int x=0;
do
{
  printf("x=%d\n", x);
} while (x!=0);
```

Цикл со счетчиком (оператор for)

Оператор for - это наиболее общий способ организации цикла. Он имеет следующий формат:
for **(выражение 1 ; выражение 2 ; выражение 3) тело цикла**

Выражение 1 обычно используется для установки начальных значений переменных, используемых в цикле («блок инициализации»).
Выражение 2 определяет условие, при котором тело цикла будет выполняться.
Выражение 3 определяет регулярные изменения переменных после каждого прохода тела цикла («блок модификации»).

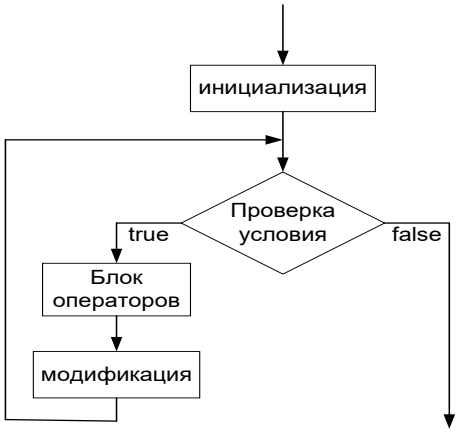


Рис 11. Блок-схема цикла со счетчиком

Схема выполнения оператора for:

1. Вычисляется выражение 1 ,один раз перед входом в цикл .
2. Вычисляется выражение 2 (перед каждым проходом цикла), если оно отлично от нуля – true (истина), то выполняется тело цикла, иначе (если выражение ложно) – цикл прекращается и управление передается оператору, следующему за оператором for.
3. Вычисляется выражение 3 (модификация данных после каждой итерации цикла), переход к пункту 2.

Существенно то, что проверка условия выполняется в начале цикла, а это значит, что тело цикла может ни разу не выполниться, если первый результат проверки будет **false**.

Пример : Эта программа, как и две предыдущие напечатает цифры от **0** до **9**

```
int x;
for (x=0; x<10; x++)  printf("%d\n",x);
```

Можно опустить любую из трех частей заголовка цикла, или даже все три, но ни одну точку с запятой опускать нельзя.

Пример: Вот как будет выглядеть тот же алгоритм с отсутствующими частями инициализации и модификации в цикле for:

```
int x=0;
for ( ; x<10 ; )
{printf("%d\n",x);
  x++;
}
```

Как видите, **x=0** теперь стоит перед циклом, а оператор **x++** стал последним в теле цикла.

Можно опустить и условие выхода, тогда программа будет считать, что условие всегда истинно и будет печатать строку "hello" до бесконечности.

```
for (;;)  printf("hello\n");
```

аналогичный бесконечный цикл while будет выглядеть так :

```
while (true)  printf("hello\n");
```

while (true) – это короткая запись оператора while (1!=0),так как проверка условия 1!=0 всегда будет давать результат – true

Бесконечные циклы применяются, когда условие выхода из цикла не известно заранее, для выхода из такого цикла требуется проверка дополнительного условия в теле цикла и оператор break.

Пример: Вычислять квадраты натуральных чисел от **1** до **k**, где **k** вводится с клавиатуры.

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
int _tmain(int argc, _TCHAR* argv[])
{
 int x=0,b,k;
printf ("k=");// вывод текста "k=" на экран («приглашение» перед вводом с клавиатуры)
scanf("%d",&k);                    // ввод данных с клавиатуры в переменную k
for (;;)
  {     x=x+1;
        b=x*x;
        printf ("\nx=%d\tx*x=%d\n",x,b);
        if (x==k) break;           // выход из цикла по условию
  }
system("pause");
return 0;
}
```

Усложним условие задачи, пусть необходимо в начале цикла задать начальные значения не одной, а двум переменным (допустим **x** и **y**), а в цикле изменить значения их обеих.
Можно сделать это следующим образом:

```
int y, x=0;
for (y=10; x<y; x+=2 )
{  printf("x=%d\ty=%d\n", x, y);
  y--;
}
```

Благодаря оператору « , » (запятая) можно упростить текст программы:

```
int y, x=0;
for (y=10; x<y; x+=2, y--) printf("x=%d\ty=%d\n", x, y);
```

Операторы break и continue

break может быть использован в операторах циклов и в операторе switch, он прерывает текущий и передает управление следующему оператору.
В циклах оператор break используется для прерывания текущей итерации цикла, затем управление передается оператору, следующему за последним оператором тела цикла. Если циклы вложенные, то он обеспечивает прекращение выполнения самого внутреннего из них. Подчеркнем, что break нельзя использовать для выхода из нескольких вложенных циклов, а составной оператор, состоящий из двух операторов break, эквивалентен одному break**.** Для выхода из вложенных циклов используйте оператор goto**.**

Оператор continue используется исключительно в циклах. Он прерывает текущую и передает управление на следующую итерацию цикла, отсекая операторы следующие за ним. Также как и оператор break, он прерывает самый внутренний из вложенных циклов.

Пример: В примере проводится подсчет суммы нечетных значений, начиная от **1** до **k**.
Оператор **%** (деление по модулю) дает целочисленный остаток от деления двух операндов, таким образом, нулевой результат получается, если первый операнд кратен второму. В примере оператор проверяет число **a** на четность, если число четное, то оно без остатка делится на **2**, управление передается на очередной проход цикла, в противном случае подсчитывается сумма.

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#define k 5
int _tmain(int argc, _TCHAR* argv[])
{
  int a,sum=0;
  for (a=1; a<=k; a++)
       {  if (a%2==0) continue;
              sum += a;    // эквивалентно sum=sum+a;
        }
printf("sum=%d\n", sum);
system("pause");
return 0;
}
```

Оператор безусловного перехода goto

Использование оператора goto всегда вызывало много споров, многие опытные практики программирования не рекомендуют его использовать, так как он затрудняет понимание программ, а также их модификацию.

Тем не менее, применение goto в некоторых случаях можно оправдать, например, когда необходимо выйти из вложенных циклов.

Оператор безусловного перехода имеет вид : goto метка;

Пример №18.

Примеры программирования

Объявление переменных.

Повторим, что при объявлении переменной присваивается **имя, тип и выделяется память.**

Пример 1: Объявление 3-х переменных вещественного типа (дробные числа), необходимых для вычисления площади прямоугольника.

```
double a, b;   // ширина и длина прямоугольника
double s;      // площадь прямоугольника
```

Пример 2: Объявление переменных, необходимых для вычисления площади кольца.

```
double rl, ri;   // внешний радиус и радиус отверстия
double s;              // площадь кольца
```

Пример 3: Объявление переменных, необходимых для вычисления стоимости покупки, состоящей из нескольких книг, ручек и блокнота.

```
  // переменные целого типа
int Kol_book;         // количество книг
int Kol_pen;          // количество ручек
// переменные вещественного типа двойной точности
double C_book;              // цена одной книги
double C_pen;               // цена одной ручки
double C_notebook;          // цена блокнота
double Summa;               // стоимость всей покупки
```

Присваивания и вычисления

Пример 4: Запишите инструкции, которые:

- присваивают переменной summa нулевое значение,
  summa = 0;
- увеличивают на единицу значение переменной p.
  p++;
- увеличивают значение переменной x на величину dx
  x = x + dx; или **x += dx;**
- пересчет расстояния из километров в версты (одна верста — это 1066,8 м).
  versta = km*1066.8;

Ввод данных с клавиатуры и вывод на экран дисплея

Пример 5.
Написать инструкцию вывода значений целых переменных a, b и c. Значение каждой переменной должно быть выведено в отдельной строке, в начале вывода разместить заголовок.

```
printf ("\nВывод переменных a,b и c: \n  a=%d\n b=%d\n c=%d\n",a,b,c);
```

Допустим a,b и c равны соответственно 1,2 и 3, тогда на экране увидим:

```
Вывод переменных  a, b и c:
a=1
b=2
c=3
```

Пример 8. Написать инструкцию вывода дробных значений переменных, которые определяют высоту (h) и длину (l) прямоугольника. Перед значением переменных должен быть пояснительный текст (h=, l=), а после — единица измерения – сантиметры (sm), между переменными поставить знак табуляции.

```
printf("\nH=%f sm\tL=%f sm\n",h,l);
```

Допустим h и l равны соответственно 3,5 и 9,12. На экране увидим:

```
H=3.5 sm        L=9.12 sm
```

Пример 9.
Написать инструкцию, обеспечивающую ввод с клавиатуры значения вещественной переменной rad

```
double rad;
scanf ("%le",&rad);
```

Пример 10.
Написать инструкции, которые обеспечивают ввод и последующий вывод значений вещественных переменных r и d. Предполагается, что пользователь после набора каждого числа будет нажимать какой-либо разделитель (<пробел>,<табуляцию> или <Enter>), а оканчивать ввод клавишей <Enter>.

```
double r,d;
printf ("\ninput r,d :");
scanf("%le %le",&r,&d);
printf ("\nr=%f\td=%f\n",r,d);
```

Пример 12*:* Ввести с клавиатуры значение угла (в радианах) и вычислить его синус.

```
/* программа вычисления синуса */
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>          // библиотека математических функций
int _tmain(int argc, _TCHAR* argv[])
{
double result , x;         // определение переменных result , x (выделение памяти)
printf("\nx=");            // вывод «приглашения» на терминал
scanf("%le",&x);           //ввод с клавиатуры числа в переменную x
                           // (формат числа: «длинное вещественное»)
result = sin (x);          // вычисление синуса
                       // вывод результата на терминал
printf("The sin() of %f is %f\n",x,result);
system("pause");   // задержка в программе для просмотра результатов
return 0;          // возвращаемое значение функции main()
}
```

Можно изменить формат вывода вещественного числа, явно указав общее число выводимых символов и количество символов после запятой.
Например вместо %f задать %10.2f , что означает вывести вещественное число в формате XXXXXXX.XX
В примере измените формат вывода результата, обратите внимание на разницу в выводе данных: printf("The sin() of %2.3f is %2.3f\n",x,result);
printf("The sin() of %e is %e\n",x,result);

Пример 11. Написать программу для вычисления площади круга. Данные для вычисления ввести с клавиатуры.
Потребуются переменные r для радиуса и s для результата вычисления.

```
// подключение библиотек
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#define PI 3.14                    // определяем константу PI
int _tmain(int argc, _TCHAR* argv[])
{int r;
double s;
printf ("\nr=");       // «приглашение» для ввода радиуса
scanf("%d",&r);      // ввод радиуса с клавиатуры
s=PI*r*r;            // вычисление и запоминание (запись) результата в переменной s
printf ("s=%10.4f\n",s);   // вывод результата вычислений
system("pause");
return 0;
}
```

Пример 12. Написать программу для вычисления площади круга. Данные для вычисления ввести с клавиатуры, проконтролировать корректность ввода данных.

Алгоритм решения

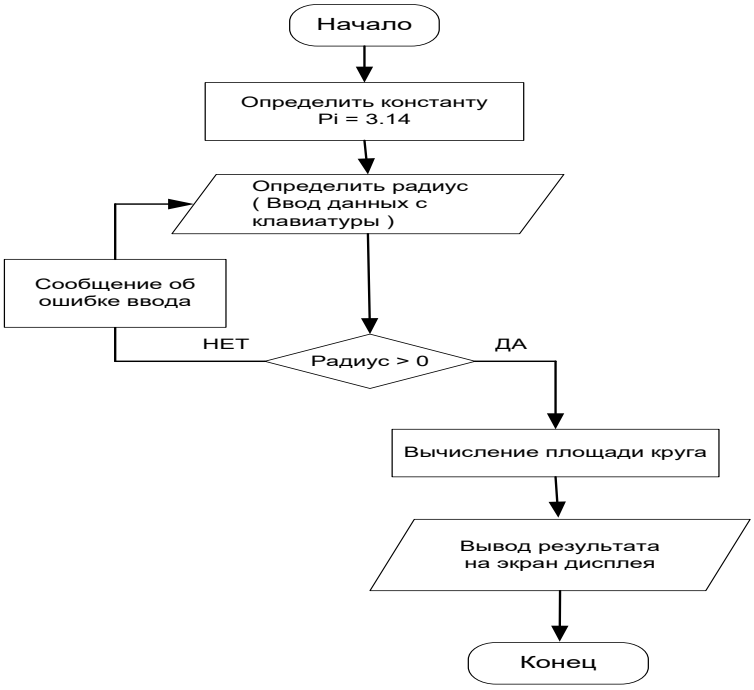
Для построения алгоритма решения выделим этапы обработки данных :

первый этап – ввод данных в компьютер (значение радиуса **r** и константы **PI**);

второй этап – вычисление по формуле $s=PI*r^2$

третий этап – вывод результата вычислений на экран дисплея.

Все эти три этапа в той или иной форме присутствуют в любой программе, причем если первый этап (ввод данных) связан с человеком (как в нашем случае – ввод с клавиатуры), то в алгоритм следует ввести контроль ошибок ввода. Например, в нашем примере радиус не должен принимать отрицательные значения.




```
// Версия программы с контролем ошибок ввода
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#define PI 3.14                    // определяем константу PI
int _tmain(int argc, _TCHAR* argv[])
{int r;
 double s;
 do
 {printf ("\nr="); scanf("%d",&r);
if (r<0)                           // проверка введенного значения r
{printf ("error: r<0 !!! \n");// ветвь «да»:  сообщение об ошибке
        continue;                  // досрочный переход на следующую итерацию
}
else  break;                       // ветвь «нет» : выход из цикла
 }  while (true);                  // бесконечный цикл (нет проверки условия)
s=PI*r*r;
printf ("s=%f\n",s);               // вывод результата вычислений
system("pause");
return 0;
}
```

Проверка условий. Ветвление программы: 2 ветви

Пример 13. Написать программу нахождения действительных корней квадратного уравнения общего вида **ax2+bx+c=0**

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int _tmain(int argc, _TCHAR* argv[])
{
int a, b, c;            //  Коэффициенты a,b,c
double d,x1,x2;
// приглашение для ввода и ввод переменных a, b, c
printf ("a= "); scanf("%d",&a);
printf ("b= "); scanf("%d",&b);
printf ("c= "); scanf("%d",&c);
d=b*b-4*a*c;                        //  вычисление дискриминанта
if ( d>0)
{ x1=(-b+sqrt(d))/(2.0*a);          // вычисление корней
   x2=(-b-sqrt(d))/(2.0*a);
   printf ("x1=%f\n", x1);          //  Печать корней
   printf ("x2=%f\n", x2);
}
//   Если корней нет, то вывод сообщения " no solution"
else  printf (" no solution\n");
system("pause");
return 0;
}
```

Ветвление программы: много ветвей

Пример 14. Написать программу, которая по введенному с клавиатуры номеру дня недели выводит на экран название этого дня.

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <clocale>
int _tmain(int argc, _TCHAR* argv[])
{       setlocale(LC_ALL, "Russian"); // русификация текстов
 int num;
 printf ("\nВведите номер дня недели >"); //  Приглашение и ввод данных
 scanf("%d",&num);
switch (num)                        //  Выбор варианта
 {case 1: printf ("Понедельник\n ");          break;
case 2: printf ("Вторник\n ");                break;
case 3: printf ("Среда\n ");                  break;
case 4: printf ("Четверг\n ");                break;
case 5: printf ("Пятница\n ");                break;
case 6: printf ("Суббота\n ");                break;
case 7: printf ("Воскресенье\n ");            break;
default: printf ("Номер неверен\n ");
}
system("pause");
return 0;
}
```

Циклические программы

Пример 15: Суммировать целые числа (через одно число) пока сумма <100, начальное число для суммирования ввести с клавиатуры.

```
// программа ведет диалог с оператором
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <clocale>
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");              // русификация текстов
int summa=0, x,c;
printf ("Суммирование чисел  через два (пока сумма < 100) \n ");
printf ("*****************************\n");
do
{printf ("Введите начальное число ->");       scanf ("%d",&x);
        printf ("Слагаемые :\n");
        summa=0;
        while (summa<100)
        {   summa=summa+x;              // или summa += x;
                printf ("%d\n",x);
                x = x+2;                        // или x+=2;
        }
        printf ("\nСумма =%d\n",summa);
printf ("Продолжить выполнение? (0-нет, 1-да)\n");
scanf ("%d",&c);
}
while (c ==1);
system("pause");
return 0;
}
```

Пример 16: Вычислить квадраты чисел от **1** до **9**.

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <clocale>
int _tmain(int argc, _TCHAR* argv[])
{       setlocale(LC_ALL, "Russian"); // русификация текстов
int i,b;
printf ("Вычисление квадратов чисел от 1 до 9\n");
printf ("---------------------------------\n");
for (i=1; i<10; i++)
        {       b=i*i;
  printf ("x=%d\t x*x=%d\n",i,b);
        }
system("pause");
return 0;
}
```

Пример 17: Ввести с клавиатуры **x** и **y**, суммировать ряд чисел начиная от **x** с шагом **y** пока сумма меньше **K**, подсчитать количество итераций суммирования. Повторять процесс ввода **x, y** и суммирования пока **x** находится в диапазоне от **A** до **B**, если число итераций больше **10**, прервать вычисление суммы.

Алгоритм решения данной задачи содержит два вложенных цикла. Во внешнем цикле проверяется условие, при котором начинается процесс суммирования (**x** должен быть в диапазоне **A…B**). Во внутреннем цикле проводится суммирование. Обратите внимание на то, что из внутреннего цикла можно выйти в двух случаях :

- естественное завершение цикла (сумма превысила **K**);
- если количество итераций превысило **10** (завершение по **break**).

Данная задача дает пример разветвленного алгоритма, когда различие во вводимых данных приводит к различию в выполнении программы. В этом случае интерфейс программы (диалог с оператором) должен максимально помогать человеку корректно вводить данные и делать выводы.

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <clocale>
#define A 10
#define B 100
#define K 200
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian"); // русификация текстов
int x, y, i, sum;
printf ("\nСуммирование натурального ряда чисел\n");
printf ("суммировать пока сумма < 200)\n");
printf ("-----------------------------------------\n");
while (true ) //бесконечный цикл, выход по break
{printf ("\nВведите начальное значение для суммирования (от 10 до 100): ");
scanf ("%d",&x);    // ввод данных с клавиатуры
if (x>=A && x<=B)
{printf ("\n Введите шаг суммирования :");   scanf ("%d",&y);
        printf ("-----------------------------------------\nСлагаемые :\n");
        for (i=1,sum=0; sum<K; x+=y, i++)
        {       printf ("%d\n",x);
                sum += x;
                if (i>10)       // полная форма записи (i>10)!=0
                {printf ("Количество итераций суммирования>10\n");
                 sum=0;
                break;          // выход из цикла for
                }
        }
if (sum !=0) {printf ("сумма=%d\n",sum);
                 printf ("\n-----------------------------------------------");
                }
}
else { printf ("Начальное значение вне допустимого диапазона, конец вычислений\n");
        break; // выход из бесконечного цикла
      }
}
system("pause");
return 0;
}
```

При выполнении условия (**i>10)** завершится выполнение только внутреннего цикла for, а выполнение внешнего цикла while продолжится, т.к. break прерывает текущий оператор, т.е. цикл for
Если нужно выйти их обоих вложенных циклов, нужно использовать оператор goto (смотри следующий пример)

Пример18: В программе осуществляется преобразование шести значений температуры, находящихся в пределах от **32** до **365** (преобразование температуры по Фаренгейту в температуру по Цельсию). Значения вводятся с клавиатуры. Если оператор ошибся более **5** раз, происходит аварийное завершение программы (Выход сразу из нескольких вложенных циклов).

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <clocale>
#define d1 32
#define d2 365
#define n 6
int _tmain(int argc, _TCHAR* argv[])
{int  count, errc=0;
double farin,celsius;
setlocale (LC_ALL,"Russian");
printf ("Перевод значений: температура по Фаренгейту –> температура по Цельсию\n");
printf ("++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n");
for (count = 1; count <= 6; count++)
{printf ("Введите значение температуры  от %d  до %d\n(значение по Фаренгейту):",d1,d2);
 scanf ("%le",&farin);
 while (farin < d1 || farin > d2)
        {if (errc<5)
                {errc++;
                printf ("Повторите ввод, ошибка № %d  :  ",errc);
                scanf ("%d",&farin);
                }
                else    goto ex0;               // выход из двух вложенный циклов
        }//-------------- конец цикла while
celsius =  (farin - 32)/1.8;
printf ("%f (по Фаренгейту) = %f (по Цельсию)",farin,celsius);
printf ("\n------------------------------------------------------------\n");
}//--------- конец цикла for
        printf ("\nНормальное завершение \n");
        system("pause");
        return 0;
ex0:  printf ("\nАварийное завершение : оператор ошибся более 5\ раз!!!\n");
system("pause");
return 1;
}
```

Вопросы.

1. В чем различие между константой и переменной?
2. Приведите примеры объявления констант и переменных.
3. Как изменить значение константы?
4. Дайте определение термину "инициализация переменной"
5. Перечислите стандартные типы данных языка Си.
6. Перечислите основные характеристики стандартных типов данных.
7. Какой тип данных нужно использовать для записи стоимости товара (в рублях и копейках)?
8. Какова структура программы на языке Си?
9. Что такое препроцессор?
10. Какие функции выполняет директива #include.
11. Какие функции выполняет директива #define
12. Какие этапы трансляции Вы знаете?
13. Какие операторы позволяют организовать ветвление программы?
14. Что такое «логическое выражение»?
15. Какие операции отношений вы знаете?
16. Для чего нужны логические связки?
17. Какие логические связки вы знаете?
18. Нарисуйте в виде блок-схемы две формы условного оператора.
19. Приведите примеры использования тернарного оператора.
20. Что общего у операторов if и switch?
21. Какие различия между операторами if и switch?
22. Напишите общую форму оператора for
23. Нарисуйте схему выполнения цикла с предусловием;
24. Нарисуйте схему выполнения цикла с постусловием;
25. Как будет ли работать оператор for, и если отсутствует условие выхода из цикла;
26. Какой из операторов следует использовать для выхода из вложенных циклов: break, goto или return
27. Для какой цели используют оператор continue?

Номер компьютера	**Варианты задания**
1,17	Подсчитать площадь круга по введенному с клавиатуры радиусу. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: нулевое и отрицательное значение радиуса.
	Определить название числа от 0 до 10, например, при вводе числа 5 писать «пять», если число >10, писать «число должно быть меньше 10»
	Ввести с клавиатуры X и m, вычислить S, выдать сообщение об ошибке, если m<0 или X=0 S= $\sum_{i=1,3,5,\ldots}^{2*m+1} \frac{1}{i * X}$
2,18	Вычислить значение x= a*b/c a , b, c, d – целые числа, x – вещественное, a , b и c ввести с клавиатуры. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: c=0
	Определить название месяца по введенному номеру, если число >12 или <1, написать «некорректный номер»
	Ввести с клавиатуры 10 целых положительных чисел, подсчитать количество нулей. Отрицательные числа игнорировать и сообщать об ошибке ввода.
3,19	Подсчитать периметр прямоугольника по введенным с клавиатуры сторонам. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: нулевые и отрицательные значения.
	Определить и вывести на экран название дня недели по введенному номеру, если число >7 или <1, писать «некорректный номер дня»
	Ввести с клавиатуры 10 чисел (как положительных, так и отрицательных), подсчитать среднее арифметическое всех отрицательных, нули игнорировать и сообщать об ошибке ввода.
4,20	Вычислить значение x= a+b/c+d a , b, c, d – целые числа, x – вещественное, a , b, c, d ввести с клавиатуры. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: деление на ноль
	Вводить различные символы до тех пор, пока не введен «ENTER». При вводе символа "{" или "}" выводить сообщение "фигурная скобка", при вводе "[" или"]" - "квадратная скобка"; при вводе "(" или ")" - "круглая скобка", в остальных случаях - сообщение "не скобка". Для решения задачи можно использовать таблицу кодов ASCII
	Ввести с клавиатуры целое трехзначное число. Найти сумму цифр введенного числа.
5,21	Подсчитать площадь треугольника по введенным с клавиатуры основанию и высоте. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: нулевые и отрицательные значения.
	Вводить числа, пока не будет введен ноль. Определять количество цифр в числе, выводить надпись «одна цифра», «две цифры» и т.п.
	По введенному с клавиатуры n найти n!
6,22	Для целого числа k от 1 до 130 вывести фразу «Мне k лет», учитывая при этом, что при некоторых значениях k слово «лет» надо заменить словом «год» или «года»
	По введенному числу (от 3 до 8) определить название фигуры: треугольник, квадрат, пятиугольник и т.д., если число < 3 или >8 писать «неизвестная фигура»
	Ввести с клавиатуры два числа:a,b (b>a). Ввести с клавиатуры 15 чисел, подсчитать количество чисел в диапазоне от a до b.

7,23	Вычислить значение x= 1/(a+b) a , b – целые числа, x – вещественное, a , b ввести с клавиатуры Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: a+b=0
	Ввести с клавиатуры номер месяца и вывести сообщение о номере квартала. Программа должна проверять правильность исходных данных. Выйти из программы при вводе 999.
	Ввести с клавиатуры X и m, вычислить S, выдать сообщение об ошибке, если m<0 или X-1=0 $\sum_{i=2,4,6,...}^{m} \frac{i*(X+5)}{X-1}$
8,24	Для целого числа d от 1 до 1000, обозначающего денежную единицу, дописать слово «рубль» в правильной форме.
	Вводить с клавиатуры различные символы и выводить слова: "цифра", если введена цифра; "латинская буква" если введена латинская буква, все остальные случаи считать ошибкой ввода. Для решения задачи можно использовать таблицу кодов ASCII
	Ввести число k (от 0 до 9). Вводить пятизначные числа, при выводе удалять все цифры равные k. Например при k =0 число 10045 преобразуется в 145.
9,25	Вычислить значение x= a /c+b a , b, c – целые числа, x – вещественное, a , b и c ввести с клавиатуры. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: деление на ноль
	Реализовать «калькулятор». При вводе с клавиатуры символа: + вывести сообщение "сложение", – вывести сообщение "вычитание", * вывести сообщение "умножение", / вывести сообщение "деление" с указанием формулы и примера с конкретными значениями, введенными с клавиатуры. Программа должна проверять правильность исходных данных при выполнении команды "деление".
	Найти сумму всех целых чисел, дающих при делении на 7 в остатке 3, из отрезка [A,B]. A,B ввести с клавиатуры.
10,26	Подсчитать периметр прямоугольника по введенным с клавиатуры сторонам. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: нулевые и отрицательные значения.
	Составить программу, которая при вводе символа "{" или "}" выводит сообщение "фигурная скобка";. при вводе "[" или"]" - "квадратная скобка"; при вводе "(" или ")" - "круглая скобка"; в остальных случаях - сообщение "не скобка". Для решения задачи использовать таблицу кодов ASCII
	Ввести трехзначное число и вывести его на экран в обратном порядке (например 123 -> 321).
11,27	Вычислить значение x= a/b+c/d a , b, c, d – целые числа, x – вещественное. a , b, c, d ввести с клавиатуры. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: b=0, d=0
	Написать программу, которая запрашивает у пользователя номер месяца и выдает на экран сообщение о времени года и названии введенного месяца. Например: 1 - январь, зима. Программа должна проверять правильность исходных данных и выводить сообщение об ошибке.
	Вводить целые числа, пока не будет введено число 0. Подсчитать сумму чисел из диапазона a,b (b>a). a,b ввести с клавиатуры

12,28	Для целого числа d от 1 до 1000, обозначающего денежную единицу, дописать слово «копейка» в правильной форме.
	Написать программу, которая запрашивает у пользователя номер дня недели и выводит одно из сообщений: "Рабочий день", "Суббота" или "Воскресенье". Программа должна проверять правильность исходных данных и выводить сообщение об ошибке.
	Найти сумму всех целых чисел, кратных 5, из отрезка [A,B]. A,B ввести с клавиатуры.
13,29	Подсчитать площадь прямоугольника по введенным с клавиатуры сторонам. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: нулевые и отрицательные значения.
	Ввести координаты точки x, y. Определить какой координатной четверти принадлежит данная точка. Вывести надпись «первая четверть», «вторая четверть» и т.п.
	Ввести пятизначное целое число, подсчитать количество четных цифр. Например: в числе 12345 две четные цифры (2 и 4)
14,30	Подсчитать площадь цилиндра по введенным с клавиатуры радиусу и высоте. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: нулевые и отрицательные значения.
	Ввести число от 2 до 5, вывести оценку в виде слова: 5 - отлично, 4 - хорошо, 3 - удовлетворительно, 2 – неудовлетворительно. Программа должна проверять правильность исходных данных и выводить сообщение об ошибке.
	Ввести с клавиатуры трехзначное целое число. Найти сумму цифр введенного числа.
15,31	Поле шахматной доски определено парой натуральных чисел (значения от 0 до 8). Ввести координаты 2-х полей k,l и m,n. Определить являются ли они полями одного цвета.
	Ввести числа a, b, c, определить, возможно ли построить треугольника со сторонами a, b, c? Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: нулевые и отрицательные значения.
	Найти сумму всех целых чисел, дающих при делении на 7 в остатке 4, из отрезка [A,B]. A,B ввести с клавиатуры.
16,32	Вычислить значение x= (1+a)/b a , b – целые числа, x – вещественное, a , b ввести с клавиатуры. Реализовать контроль ошибки ввода, вывод сообщения об ошибке и повторный ввод данных. Ошибка: b=0
	Дано трехзначное число a. Определить, составляют ли цифры числа возрастающую последовательность.
	Ввести с клавиатуры X и m, вычислить S, выдать сообщение об ошибке, если m<0 или X=0 S = $\sum_{i=1,2,3,\ldots}^{m} \frac{i * X - 10}{X}$