Часть 1. Основы создания программ в Си

Лабораторная работа.1. Простейшие программы и базовые операции ввода вывода

Цель выполнения лабораторной работы

Освоение создания простых программ языка Си. Освоение базовых операций ввода-вывода при работе с консолью.

Порядок выполнения работы

Обучаемые разрабатывают программы в соответствии с представленным описанием по выданным им индивидуальным заданиям.

Задание 1 Приложение «Hello World!»

Дается представление об элементарной функции.

С этого сакраментального восклицания начинают обучение все программисты. Вот код программы, который, если не лень, вы можете набрать в любом редакторе и сохранить (желательно в отдельном пустом каталоге, например каталоге D:\Student\ — это сейчас не имеет особого значения, но хорошо выработать у себя такую привычку) под именем hello.c.

Листинг Hello.c

```
#include <stdio.h>
#include <conio.h>
/*
** Печать простого сообщения.
*/
#pragma argsused
int  main(int   argc,   char*   argv[])
{
  printf("Hello World!\n");
  printf ("Press a key...\n");
  getch();
  return 0;
}
```

Далее полученный файл с кодом необходимо скомпилировать для получения исполняемого файла *.exe.

На первых лабораторных для компиляции будем использовать компилятор командной строки входящий в инструментарий среды разработки Visual Studio 2010.

Для использования компилятором воспользуется консолью ввода, в программах нужно найти группу Visual Studio и в инструментальных приложения следует найти консоль ввода - Command Promt (2010).

В консоли сначала следует проверить доступность вызова команд компилятора из командной строки. В командной строке выполните вызов «cl»:

```
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC>cl
```

В окне появится, например, такая информация:

```
  Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00.40219.01 for
80x86
  Copyright (C) Microsoft Corporation.  All rights reserved.

  usage: cl [ option... ] filename... [ /link linkoption... ]
```

Это означает, что компилятор доступен.

Возвращаясь к программе файле Hello.c, скомпилируем содержимое этого файла в исполняемый файл Hello.exe. Для начала необходимо установить текущую директорию в командной строке на каталог D:\Student, для этого воспользуемся командой смены текущего диска и его каталога “cd”:

Команда смены диска – это просто указание его имени:

```
  >D:
```

Команда смены каталога на текущем диске D:

```
  D:\>cd Student
```

Итак, находясь в текущем каталоге (D:\Student), можно вызывать команды компилятора для файлов Си кода из этой директории. Способ вызова команды имеет следующий обобщенный вид:

```
  >cl <имя_файла.c> /Fe<имя_исполняемого_файла.exe>
```

Для нашего примера получится следующая команда:

```
  >cl Hello.c /FeHelloApp.exe
```

После нажатия клавиши ENTER начнется работа компилятора. Если в нашей программе нет ошибок, то в файле HelloApp.exe будет создан исполнимый код программы. Для ее исполнения достаточно набрать в командной строке:

```
  >HelloApp.exe
```

Результат будет следующий:

```
  Hello World!
  Press a key...
```

Если в код закралась ошибка, компилятор сообщит о ней, например:

```
  >cl Hello.c /FeHelloApp.exe
  Hello.c(11) : error C2146: syntax error : missing ';' before identifier
'getch'
```

Это говорит о том что код не может быть разобран перед вызовом функции 'getch' на строчке 11.

В данном случае была пропущена точка с запятой после второго вызова функции ‘ printf ':

```
    printf("Hello World!\n");
    printf ("Press a key...\n");
```

По правилам синтаксиса языка: Каждый оператор в теле функции оканчивается точкой с запятой (;).

Элементы простейшей программы

Наша программа при запуске в Windows с консоли (в «окне MS-DOS») напечатала приветствие «Hello World!», на следующей строчке «Press a key...» и после нажатия произвольной клавиши возвратила управление системе. Таков алгоритм её работы. Как это происходит…

Первые две строки:

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

являются *директивами препроцессора.* Как и любые другие строки, начинающиеся со знака "#". *Препроцессор* в исторические времена был отдельной программой, выполнявшей т. н. препроцессорную обработку текста программы, т.е. процессорную — значит обработку кода до того, как его начнет обрабатывать собственно компилятор, т.е. программа, транслирующая текст на языке С в машинный или объектный код. В данном случае директивы #include означают, что каждую из них следует заменить на содержимое файла, имя которого указано в угловых скобках. Это необходимо для проверки корректности вызова библиотечных процедур. Вот те строчки (первая — из stdio.h, вторая — из conio.h), которые, собственно, нам здесь требуются:

```
int       _RTLENTRY _EXPFUNC printf (const char * __format, . ..);
int       _RTLENTRY _EXPFUNC getch ( void );
```

Это, как говорят, прототипы функций, которые позволяют компилятору проверить правильность передачи параметров (то, что в скобках) и возврат корректных типов значений (они указываются слева от имени функции; здесь обе функции, если не вдаваться в подробности, возвращают значения типа int, т.е. целые значения).

Далее идут три строки, являющиеся комментарием. В языке С комментарий — это любой фрагмент текста программы, начинающийся символами "/*" и заканчивающийся литерами "*/".

В языке С++ введен новый стиль комментариев. Комментарий начинается двумя чертами дроби и продолжается до конца строи, например:

```
int   main (int   argc,   char   *argv[])  //Пример комментария C++
```

Стоит отметить, что использование редактора Notepad++ позволяет получить подсветку ключевых слов в синтаксисе языка Си в случае если имя файла имеет соответствующее расширение (*.c, *.h, *.cpp и др.).

Далее следует строка

```
#pragma argsused
```

Эта строка программы особого смысла не имеет. Вообще директивы *#pragma*, хотя и относятся формально к директивам препроцессора, на самом деле обращены к самому компилятору и не имеют к препроцессору особого отношения. Директив #pragma очень много и у разных компиляторов они разные. Данная директива должна означать, что компилятору не нужно выдавать предупреждающих сообщений (не сообщений об ошибке) о том, что параметры функции main внутри нее никак не используются.

Можно было бы написать заголовок функции main и так:

```
int main()
```

И это было бы правильно, поскольку функцию *main* мы можем здесь задавать как угодно. Те параметры, что указаны в первоначальном коде — это аргументы командной строки, опционально передаваемые программе или командой, или директивой операционной системы.

Несколько забегая вперед, скажем, что *int argc* в заголовке *main* — это число аргументов командной строки, а *char* argv* [ ] — это массив самых аргументов в символьном (*char**) виде. Однако функция *main* () дает нам возможность поговорить о функциях вообще.

Функция main()

После всех директив в программе расположено определение функции *main* (). Как уже говорилось, в строгом смысле любая программа на С содержит эту функцию, которая является ее входной точкой. Однако в среде Windows вместо *main* () часто используется *winMain* ().

Функция *main* () — это, конечно, частный случай функции вообще. Функции являются основными «строительными блоками» программы, или подпрограммами. Они, в свою очередь, строятся из операторов, составляющих тело функции. Каждый оператор оканчивается точкой с запятой (;).

В общем виде функция определяется таким образом:

возвращаемый_тип имя_функции(список_параметров)

```
 {
  // В фигурных скобках заключено тело функции,
  // составленное из отдельных операторов.
  тело_функции
}
```

В нашем случае тело функции состоит из четырех операторов, первые три из которых являются, в свою очередь, вызовами функций. Значения, возвращаемые функциями, здесь игнорируются. Применяемые здесь функции содержатся в стандартной (исполнительной) библиотеке С. Первая выводит сообщение и переводит строку на консоли, вторая — выводит подсказку (не переводя строки), а третья, getch( ), ждет, пока пользователь не нажмет какую-нибудь клавишу.

Когда это произойдет, getch( ) возвратит управление в функцию main(), а оператор return возвратить управление операционной системе:

```
return   0;
```

Переменные

Отдельная единица данных должна обязательно иметь определенный тип. Для ее хранения во время работы программы мы должны, во-первых, отвести соответствующее место в памяти, а во-вторых, идентифицировать ее, присвоив некоторое имя. Именованная единица памяти для хранения данных называется *переменной*.

Каждая переменная и каждое выражение (набор операций с переменными) обязаны иметь тип. Именно тип определяет операции, которые могут выполняться над ними. Например, в описании

int inch;

говорится, что переменная с именем *inch* имеет тип *int*, т.е. *inch* является целой переменной.

Объявление переменной - это оператор, который вводит имя в программу. В объявлении указывается тип переменной и её имя. Тип, в свою очередь, определяет, как правильно использовать переменную или строить выражения с нею.

```
short i;                // объявление короткой целой переменной
char quit = 'Q';  // инициализация символьной переменной
float f1, factor = 3.0, f2;  //  Три переменных типа float,
                             //  одна из которых инициализирована
```

Основные типы, наиболее приближенные к "аппаратной реальности" машины, таковы:

char
short
int
long

Они представляют целые числа. Следующие типы:

float
double
long double

представляют числа с плавающей точкой. Переменная типа *char* имеет размер, нужный для хранения одного символа на данной машине (обычно это один байт или 8 бит). Переменная int имеет размер, необходимый для целой арифметики на данной машине (обычно это одно слово 2 байта или 16 бит).

Встроенные типы данных представлены в Таблице 1 ниже:

Таблица 1 Встроенные типы данных

Тип данных	Размер (бит)	Диапазон
char	8	–128 – 127
signed char	8	–128 – 127
unsigned char	8	0 – 255
short	16	–32768 – 32767
unsigned short	16	0 – 65535
int	32	–2147483648 – 2147483647
unsigned int	32	0 – 4294967295
long	32	–2147483648 – 2147483647
unsigned long	32	0 – 4294967295
float	32	$3,4\times10^{-38} – 3,4\times10^{38}$
double	64	$1,7\times10^{-308} – 1,7\times10^{308}$
long double	80	$3,4\times10^{-4932} – 3,4\times10^{4932}$

Следующие арифметические операции можно использовать над любым сочетанием перечисленных типов:

+ (плюс, унарный и бинарный)
- (минус, унарный и бинарный)
* (умножение)
/ (деление)
% (остаток от деления)

То же верно для операций отношения:

== (равно)

!= (не равно)

< (меньше чем)

<= (меньше или равно)

>= (больше или равно)

Для операций присваивания и арифметических операций в C++ выполняются все осмысленные преобразования основных типов, чтобы их можно было неограниченно использовать любые их сочетания:

```
double d;
int i;
short s;
// ...
d = d + i;
i = s * i;
```

В таблице ниже представлен список операций языка C (см. Таблица 2.)

Общие положения для выполнения задания 1:

Каждая переменная программы должна быть объявлена.

Объявления переменных обычно помещают в начале функции, сразу за заголовком. Следует обратить внимание на то, что хотя язык C++ допускает объявление переменных практически в любом месте функции, объявлять переменные лучше все-таки в начале функции, снабжая инструкцию объявления кратким комментарием о назначении переменной.

Инструкцию объявления переменной можно использовать для инициализации переменной.

В имени переменной допустимы буквы латинского алфавита и цифры (первым символом должна быть буква).

Компилятор C++ различает прописные и строчные буквы, поэтому, например, имена Sum и sum обозначают разные переменные.

Основными числовыми типами языка C++ являются *int* (целый) и *float* (дробный).

После инструкции объявления переменной рекомендуется поместить комментарий — указать назначение переменной.

Имя переменной должно отражать смысловое назначение

Индивидуальное задание №1

Пример задачи:

Объявить переменные, необходимые для вычисления площади прямоугольника.

Решение:

```
float w, l; // width - ширина и length - длина прямоугольника
float s;    // площадь прямоугольника
```

Список задач №1 для индивидуального решения:

1. Объявить переменные, необходимые для пересчета веса из фунтов в килограммы.

2. Определить исходные данные и объявить переменные, необходимые для вычисления дохода по вкладу.
3. Объявить переменные, необходимые для вычисления площади круга.
4. Объявить переменные, необходимые для вычисления площади кольца.
5. Объявить переменные, необходимые для вычисления стоимости покупки, состоящей из нескольких тетрадей, карандашей и линейки.
6. Объявить переменные, необходимые для вычисления объема цилиндра;
7. Объявить переменные, необходимые для вычисления объема куба;
8. Объявить переменные, необходимые для расчета времени в пути зависимости от средней скорости движения;
9. Объявить переменные, необходимые для расчета ускорения движения тела
10. Объявить переменные, необходимые для расчета sin, cos, и tg угла в прямоугольном треугольнике.

Операции присваивания

Операция присваивания (=) не представляет особых трудностей. При ее выполнении значением переменной в левой части становится результат оценки выражения справа. Как уже говорилось, эта операция сама возвращает значение, что позволяет, например, написать:

```
a = b = c =  someExpression;
```

После исполнения такого оператора все три переменных *a*, *b*, *c* получат значение, равное *someExpression*.

Что касается остальных десяти операций присваивания, перечисленных в таблице, то они служат для сокращенной нотации присваиваний определенного вида. Например,

```
s += i;
```

эквивалентно

```
s = s + i;
```

или

```
x *= 10
```

эквивалентно

```
x = x*10;
```

Таблица 2 Операции языка Си

Операция	Описание	Приоритет	Ассоциация
Первичные и постфиксные операции			
[]	индексация массива	16	слева направо
()	вызов функции	16	слева направо
.	элемент структуры	16	слева направо
->	элемент указателя	16	слева направо
++	постфиксный инкремент	15	слева направо
--	постфиксный декремент	15	слева направо
Одноместные операции			
++	префиксный инкремент	14	справа налево
--	префиксный декремент	14	справа налево
sizeof	размер в байтах	14	справа налево
(тип)	приведение типа	14	справа налево
~	поразрядное NOT	14	справа налево
!	логическое NOT	14	справа налево
-	унарный минус	14	справа налево
&	взятие адреса	14	справа налево
*	разыменование указателя	14	справа налево
Двухместные и трехместные операции:			
Мультипликативные			
*	умножение	13	слева направо
/	деление	13	слева направо
%	взятие по модулю	13	слева направо
Аддитивные			
+	сложение	12	слева направо
-	вычитание	12	слева направо
Поразрядного сдвига			
<<	сдвиг влево	11	слева направо
>>	сдвиг вправо	11	слева направо
Отношения			
<	меньше	10	слева направо
<=	меньше или равно	10	слева направо
>	больше	10	слева направо
>=	больше или равно	10	слева направо
==	равно	9	слева направо
!=	не равно	9	слева направо
Поразрядные			
&	поразрядное AND	8	слева направо
^	поразрядное XOR	7	слева направо
|	поразрядное OR	6	слева направо
Логические			
&&	логическое AND	5	слева направо
||	логическое OR	4	слева направо
Условные			
? :	условная операция	3	справа налево
Присваивания			
=	присваивание	2	справа налево
*=	присвоение произведения	2	справа налево
/=	присвоение частного	2	справа налево
%=	присвоение модуля	2	справа налево
+=	присвоение суммы	2	справа налево
-=	присвоение разности	2	справа налево
<<=	присвоение левого сдвига	2	справа налево
>>=	присвоение правого сдвига	2	справа налево
&=	присвоение AND	2	справа налево
^=	присвоение XOR	2	справа налево
|=	присвоение OR	2	справа налево
,	запятая	1	слева направо

Приведение типа

Если в операторе присваивания тип результата, полученного при оценке выражения в правой части, отличен от типа переменной слева, компилятор выполнит автоматическое *приведение типа* (по-английски typecast или просто cast) результата к типу переменной. Например, если оценка выражения дает вещественный результат, который присваивается целой переменной, то дробная часть результата будет отброшена, после чего будет выполнено присваивание. Ниже показан и обратный случай приведения:

```
int  p;
double pReal = 2.718281828;
p = pReal;    // p получает значение 2
pReal = p;    // pReal теперь равно 2.0
```

Возможно и принудительное приведение типа, которое выполняется посредством *операции приведения* и может применяться к любому операнду в выражении, например:

```
p = p0 + (int)(pReal +0.5);   // Округление pReal
```

Следует иметь в виду, что операция приведения типа может работать двояким образом. Во-первых, она может производить действительное преобразование данных, как это происходит при приведении целого типа к вещественному и наоборот. Получаются совершенно новые данные, физически отличные от исходных.

Во-вторых, операция может никак не воздействовать на имеющиеся данные, а только изменять их интерпретацию. Например, если переменную типа *short* со значением -1 привести к типу unsigned *short*, то данные останутся теми же самыми, но будут интерпретироваться по-другому: (как целое без знака), в результате чего будет получено значение 65535.

Смешанные выражения

В арифметическом выражении могут присутствовать операнды различных типом кик целые, так и вещественные, а кроме того, и те и другие могут иметь различную длину (*short* , *long* и т.д.). в то время как оба операнда любой арифметический операции должны иметь один и тот же тип, В процессе оценки таких выражений компилятор следует алгоритму т. н. возведения типов, который заключается в следующем.

На каждом шаге оценки выражения выполняется одна операция, и имеются два операнда. Если их тип различен, операнд меньшего «ранга экстенсивности» приводится к типу более «экстенсивного». Под экстенсивностью понимается диапазон значений, который поддерживается данным типом. По возрастанию экстенсивности типы следуют в очевидном порядке:

char
short
int, long
float
double
long double

Кроме того, если в операции участвуют знаковый и беззнаковый целочисленные типы, то знаковый операнд приводится к беззнаковому типу. Результат тоже будет беззнаковым. Во избежание ошибок нужно точно представлять себе, что при этом происходит, и при необходимости применять операцию приведения, явно преобразующую тот или иной операнд.

Поразрядные операции и сдвиги

Эти операции применяются к целочисленным данным. Последние рассматриваются просто как набор отдельных битов.

При поразрядных операциях каждый бит одного операнда комбинируется (в зависимости от операции) с одноименным битом другого, давая бит результата. При единственной одноместной поразрядной операции — отрицании (-) — биты результата являются инверсией соответствующих битов ее операнда.

При сдвиге влево биты первого операнда перемещаются влево (в сторону старших битов) на заданное вторым операндом число позиций. Старшие биты, оказавшиеся за пределами разрядной сетки, теряются; справа результат дополняется нулями.

Результат сдвига вправо зависит от того, является ли операнд знаковым или беззнаковым. Биты операнда перемещаются вправо на заданное число позиций. Младшие биты теряются. Если операнд — целое со знаком, производится расширение знакового бита (старшего), т.е. освободившиеся позиции принимают значение 0 в случае положительного числа и 1 — в случае отрицательного. При беззнаковом операнде старшие биты заполняются нулями.

Сдвиг влево эквивалентен умножению на соответствующую степень двойки, сдвиг вправо — делению. Например,

```
aNumber  =   aNumber<<4; //умножает aNumber на 16.
```

Инкремент и декремент

Одноместные операции инкремента (++) и декремента (--) соответственно увеличивают или уменьшают свой операнд (обязательно переменную) на единицу. Они изменяют значение самой переменной, т.е. являются скрытыми присваиваниями. Иногда эти операции применяют в качестве самостоятельного оператора:

```
i++;      или      ++i;
```

И то и другое эквивалентно

```
i  =  i  +  1;
```

Но эти операции могут использоваться и в выражениях:

sum = sum + x * -i;

Инкремент и декремент реализуются в двух формах: префиксной (++i) я постфиксной (i--). Префиксные операции выполняются перед тем, как их операнд будет участвовать в оценке. Постфиксные операции выполняются только после оценки.

Условная операция

Условная операция (?:) позволяет составить условное выражение, т.е. выражение, принимающее различные значения в зависимости от некоторого условия. Эта операция является трехместной. Если ее условие (первый операнд) истинно, оценкой выражения будет второй операнд; если ложно — третий. Классический пример:

```
max_ab  =  a > b? a : b;
```

Функции ввода и вывода в С

printf( ) является функцией стандартной библиотеки с переменным числом аргументов. Она всегда имеет по крайней мере один аргумент — *строку формата,* чаще всего строковый литерал. Строка может содержать *спецификаторы преобразования.* Функция сканирует строку и передает ее символы на *стандартный вывод* программы, по умолчанию консоль, пока не встретит спецификатор преобразования. В этом случае *printf( )* ищет дополнительный аргумент, который форматируется и выводится в соответствии со спецификацией. Таким образом, вызов *printf( )* должен содержать столько дополнительных аргументов, сколько спецификаторов преобразования имеется в строке формата.

Спецификация преобразования

Синтаксис спецификатора преобразования имеет такой вид:

%[флаги] [поле] [.точность] [размер] *символ_типа*

Как видите, обязательными элементами спецификатора являются только начальный знак процента и символ, задающий тип преобразования. Следующая таблица перечисляет возможные варианты различных элементов спецификации.

Таблица 3 Элементы спецификатора преобразования

Элемент	Символ	Аргумент	Описание
флаг	-		Выровнять вывод по левому краю поля.
	0		Заполнить свободные позиции нулями вместо пробелов.
	+		Всегда выводить знак числа.
	пробел		Вывести пробел на месте знака, если число положительное.
	#		Вывести 0 перед восьмеричным или 0x перед шестнадцатеричным значением.
поле	число		Минимальная ширина поля вывода.
точность	.число		Для строк — максимальное число выводимых символов; для целых — минимальное число выводимых цифр; для вещественных — число цифр дробной части.
размер	h		Аргумент — короткое целое.
	l		Аргумент — длинное целое.
	L		Аргумент имеет тип long double.
символ типа	d	целое	Форматировать как десятичное целое со знаком.
	i	целое	То же. что и d.
	o	целое	Форматировать как восьмеричное без знака.
	u	целое	Форматировать как десятичное без знака.

Элемент	Символ	Аргумент	Описание
	x	целое	Форматировать как шестнадцатеричное в нижнем регистре. '
	X	целое	Форматировать как шестнадцатеричное в верхнем регистре.
	f	вещественное	Вещественное в форме (-)dddd.dddd.
	e	вещественное	Вещественное в форме (-)d.dddde(+1 -)dd.
	E	вещественное	То же, что и e, с заменой e на E.
	g	вещественное	Использовать форму f или e в зависимости от величины числа и ширины поля.
	G	вещественное	То же, что ид — но форма f или E.
	c	символ	Вывести одиночный символ.
	s	строка	Вывести строку.
	n	указатель	Аргумент —указатель на переменную типа int. В нее записывается количество выведенных к данному моменту символов.
	P	указатель	Вывести указатель в виде шестнадцатеричного числа XXXXXXXX.

Как видите, *флаги* задают «стиль» представления чисел на выводе, *поле* и *точность* определяют характеристики поля, отведенного под вывод аргумента, *размер* уточняет тип аргумента, и *символ типа* задает собственно тип преобразования. Следующий пример показывает возможности форматирования функции *printf* ( ). Советую не полениться и поэкспериментировать с этим кодом, меняя флаги и параметры поля вывода.

Задание 2 Возможности функции printf( )

Введите пример использования функции *printf ( )*, скомпилируйте и запустите приложение с именем PrintfTestApp.exe.

```
/*
/* Printf.c: Демонстрация форматирования вывода на'консоль **
             функцией printf().
*/
#include <stdio.h>
#include <conio.h>
#pragma argsused
int main(int  argc,   char  *argv[])
{
    double p = 27182.81828;
    int j = 255;
    char s[]= "Press any key...";
    /* Вывести 4 цифры; вывести обязательный знак: */
    printf("Test integer formatting: %13.4d %+8d\n", j, j);
    /* Вывести по левому краю ср знаком; заполнить нулями: */
    printf("More integer formatting: %-+13d % 08d\n", j, j) ;
    printf("Test octal and hex:      %#13o %#8.6x\n", j, j);
    printf("\nTest e and f conversion: %13.7e %8.2f\n", p, p);
    printf ("\n%s", s);    /* Вывести строку подсказки. */
    getch();
    return 0;
}
```

Команда для компиляции файла printf.c будет следующей:

```
>cl printf.c /FePrintfTestApp.exe
```

Результат работы должен быть следующим:

```
D:\Student>PrintfTestApp.exe
Test integer formatting:          0255     +255
More integer formatting: +255           0000255
Test octal and hex:               0377 0x0000ff

Test e and f conversion: 2.7182818e+04 27182.82

Press any key...
```

Escape-последовательности

В строках языка С для представления специальных (например, непечатаемых) символов используются *escape-последовательности,* состоящие из обратной дробной черты, за которой следует один или несколько символов. (Название появилось по аналогии с командами управления терминалом или принтером, которые действительно представляли собой последовательности переменной длины, начинающиеся с кода ESC.) В приведенных примерах функции printf () вы уже встречались с одной такой последовательностью — \n. Сама обратная косая черта называется *escape-символом.*

В таблице 4 перечислены возможные esc-последовательности:

Таблица 4 Escape-последовательности языка С

Последовательность	Название	Описание
\a	Звонок	Подает звуковой сигнал.
\Ь	Возврат на шаг	Возврат курсора на одну позицию назад.
\f	Перевод страницы	Начинает новую страницу.
\п	Перевод строки	Начинает новую строку.
\г	Возврат каретки	Возврат курсора к началу текущей строки.
\t	Табуляция	Переход к следующей позиции-табуляции.
\v	Вертикальная табуляция	Переход на несколько строк вниз.
\\		Выводит обратную дробную черту.
\’		Выводит апостроф (одинарную кавычку).
\”		Выводит кавычку (двойную).

Кроме того, esc-последовательности могут представлять символы в ASCII-коде — в восьмеричном или шестнадцатеричном формате:

\000	От одной до трех восьмеричных цифр после esc-символа.
\xHH ИЛИ \XHH	Одна или две шестнадцатеричных цифры после esc-символа.

Приступая к решению задач этого раздела, следует вспомнить, что:

1. Функция *printf* обеспечивает вывод на экран монитора сообщений и значений переменных.
2. Первый параметр функции printf — строка вывода, определяющая выводимый текст и формат отображения значений переменных, имена которых указаны в качестве остальных параметров функции.
3. Формат вывода значений переменных задается при помощи спецификатора преобразования — последовательности символов, начинающейся с %.
4. Чтобы после окончания работы программы ее окно не было сразу закрыто, в конец программы нужно поместить следующие инструкции:

```
printf ("Для завершения нажмите <Enter>\n");
getch();
```

Индивидуальное задание №2

1. Написать программу, которая выводит на экран ваше имя и фамилию, год и дату рождения, возраст. Дату и возраст программа должна брать из созданных и проинициализированных переменных data_day, data_month, data_year и age.
2. Написать программу, которая выводит на экран ваше имя, отчество и фамилию (каждую часть имени с новой строки). ФИО родителей.
3. Написать программу, которая выводит на экран приведенное далее четверостишие. Между последней строкой стихотворения и именем автора должна быть пустая строка.

Унылая пора! Очей очарованье!
Приятна мне твоя прощальная краса —
Люблю я пышное природы увяданье,
В багрен и золото одетые леса.

Пушкин А.С.

4. Написать инструкцию вывода значений переменных a, b и c (типа float) с пятью цифрами в целой части и тремя — в дробной. Значения должны быть выведены в виде: a = значение, b = значение, c = значение.
5. Написать инструкцию вывода значений переменных h и w (типа float), которые содержат значения высоты и длины прямоугольника. Перед значением переменной должен быть пояснительный текст (высота =, ширина =), а после — единица измерения (см).
6. Написать инструкцию, которая выводит в одной строке значения переменных n и m целого типа (int).
7. Написать инструкцию вывода значений целых переменных a, b и c. Значение каждой переменной должно быть выведено в отдельной строке.

8. Написать инструкции вывода значений дробных переменных xl и x2. На экране перед значением переменной должен быть выведен поясняющий текст, представляющий собой имя переменной, за которым следует знак "равно".

Функции ввода строки — *scanf*() и *gets*()

В языке С для ввода имеется «зеркальный двойник» *printf* ( ) — функция *scanf*( ). Функция читает данные со *стандартного ввода,* по умолчанию — клавиатуры. Она так же, как и *printf*(), принимает строку формата с несколькими спецификаторами преобразования и несколько дополнительных параметров, которые должны быть *адресами* переменных, куда будут записаны введенные значения.

Примером вызова *scanf* () может служить следующий фрагмент кода:

```
int  age;
printf("Enter your age: ");   // Запросить  ввод возраста пользователя.
scanf("%d", &age);            //  Прочитать  введенное число.
```

Функция возвращает число успешно сканированных полей, которое в приведенном фрагменте игнорируется. При необходимости вы можете найти полную информацию по *scanf* () в оперативной справке компилятора. Однако следует сказать, что программисты не любят эту функцию и пользуются ей очень редко. Причина в том, что опечатка при вводе (скажем, наличие буквы в поле, предполагающем ввод числа и т.п.) может привести к непредсказуемым результатам. Контролировать корректность ввода и обеспечить адекватную реакцию программы на ошибку при работе со *scanf* () довольно сложно. Поэтому часто предпочитают прочитать, целиком всю строку, введенную пользователем, в некоторый буфер, а затем самостоятельно декодировать ее, выделяя отдельные лексемы и преобразуя их в соответствующие значения. В этом случае можно контролировать каждый отдельный шаг процесса преобразования.

Ввод строки с клавиатуры производится функцией *gets* ():

```
char s[80];
gets(s);
```

Для преобразования строк, содержащих цифровое представление чисел, в численные типы данных могут применяться функции *atoi*(), *atol*() и *atof* (). Они преобразуют строки соответственно в целые, длинные целые и вещественные числа (типы *int*, *long* и *double*). Входная строка может содержать начальные пробелы; первый встреченный символ, который не может входить в число, завершает преобразование. Прототипы этих функций находятся в стандартном заголовочном файле *stdlib.h*.

Функции *getch*() и *getche*()

Это низкоуровневые функции ввода одиночного символа с консоли. Первую из них вы уже видели; она возвращает значение введенного символа (типа *int*), не отображая его на экране. Функция *getche*() отличается от нее только тем, что отображает введенный символ.

Задание 3 Пример функции ввода информации пользователем

Теперь мы напишем небольшую программу, которая проиллюстрирует все существенные моменты создания функции; в программе применяются некоторые из функций для работы со строками, описанных выше.

Пользователю предлагается ввести имя (в произвольной форме — только имя, имя и фамилию и т.п.), а затем номер телефона, просто как 7-значное число без пробелов или дефисов. После этого программа распечатывает полученные данные, выводя номер телефона в более привычном формате:

```
/**  Convert.c: Пример функции, преобразующей число
**              в строку специального вида.
*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
/* Прототип функции */
void convert(char *buffer, long num);
#pragma argsused
int main(int argc, char* argv[])
{
    long number; char s[80], name[80] ;
    printf("Enter name: ");
    gets (name);
    printf("Enter phone number:");
    gets (s) ;
    number = atol(s);
    /* Преобразовать номер в обычную форму. */
    convert(s, number);
    /* Вывести результат. */
    printf("\n%-30s %10s\n", name, s);
    getch();
    return 0;
}
/* Определение функции */
void convert(char *buffer, long num)
{
    int grpl, grp2, grp3;
    grp3 = num % 100;   //  Две  последние  цифры. num = 100;
    grp2 = num % 100;   //  Две  средние цифры,
    grpl = num / 100;   //  Три  старшие  цифры.
    /*  Преобразовать  в   строку. */
    sprintf(buffer, "%03d-%02d-%02d", grpl, grp2, grp3);
}
```

Введите пример, скомпилируйте и запустите приложение с именем Введите пример использования функции *printf*( ), скомпилируйте и запустите приложение с именем PrintfTestApp.exe.

Функция *convert*() объявлена как *void* и не возвращает значения, вследствие чего в ее теле можно опустить оператор *return*. (именно этот оператор служит для возврата значения функции в вызывающую программу). Она преобразует переданный ей телефонный номер (второй параметр) и записывает его в указанный строковый буфер (первый параметр). Центральный момент преобразования — разбиение номера на группы — является довольно характерным примером применения операций деления с остатком.

Для преобразования полученных групп в строку вызывается функция sprintf (). Она совершенно аналогична функции *printf*( ) за исключением того, что вместо вывода на консоль записывает результат в строковый буфер, указанный первым параметром.

В основной программе, т.е. в функции *main*( ), использована функция *atol*( ), преобразующая строку в длинное целое. Ее прототип объявлен в заголовке *stdlib.h*.

В верхней части файла мы поместили прототип функции *convert*(). Определение функции мы поместили после *main*(), поэтому прототип в данном случае необходим — без него компилятор не сможет корректно генерировать вызов *convert*().

Подытожим некоторые правила относительно прототипов и определении функций

Функция может возвращать значение практически любого типа (включая определяемые пользователем) или не возвращать его вообще. В последнем случае функция описывается как void.

Функция может не иметь параметров. В этом случае на месте списка параметров в прототипе или определении также ставится ключевое слово void; в вызове функции на месте списка аргументов не пишется вообще ничего (однако скобки необходимы).

В прототипе, в отличие от определения, нет необходимости указывать имена параметров; список параметров может состоять из перечисления только их типов, разделенных запятыми, например:

void convert(char*, long);

Тем не менее, обычно имена параметров указывают, так как если имя дано осмысленно, оно должно пояснять читателю назначение параметра.

Прототип не обязателен, если определение функции расположено в тексте программы выше того места, где она вызывается (точнее говоря, в этом случае прототипом служит само определение функции).

Индивидуальное задание №3

Приступая к решению задач этого раздела, следует вспомнить, что:

Для ввода исходных данных с клавиатуры предназначена функция *scanf*.

Первым параметром функции *scanf* является управляющая строка, которая определяет формат вводимых данных. Остальные параметры задают переменные, значения которых должны быть введены с клавиатуры. Перед именем переменной нужно ставить символ & (фактически, в инструкции ввода указывают адреса переменных).

Управляющая строка представляет собой заключенный в двойные кавычки список спецификаторов:

- %i — для ввода целых чисел со знаком;
- %u — для целых беззнаковых чисел;
- %f — для дробных чисел;
- %c —для ввода символа;
- %s — для ввода строки.

Отсутствие знака & перед именем переменной, указанной в качестве параметра функции *scanf*, является типичной ошибкой начинающих программистов (следует обратить внимание, что компилятор эту ошибку не обнаруживает!).

Варианты заданий

1. Написать инструкцию, которая обеспечивает ввод с клавиатуры переменной *kol* целого типа.
2. Написать инструкцию, обеспечивающую ввод с клавиатуры значения переменной radius типа float.
3. Написать инструкции, которые обеспечивают ввод значений дробных переменных u и r (тип float). Предполагается, что пользователь после набора каждого числа будет нажимать клавишу <Enter> (каждое число вводить в отдельной строке).
4. Объявить необходимые переменные и написать инструкции ввода исходных данных для программы вычисления дохода по вкладу. Предполагается, что процентную ставку программа определяет на основе данных о сумме и сроке вклада.
5. Объявить необходимые переменные и написать инструкции ввода исходных данных для адресной книги.
6. Объявить необходимые переменные и написать инструкции ввода исходных данных для описания контактов человека: ФИО, mail, телефон, ICQ и т.п.
7. Написать инструкцию, обеспечивающую ввод с клавиатуры значений переменных для расчета sin, cos, и tg угла в прямоугольном треугольнике.
8. Написать инструкцию, обеспечивающую ввод с клавиатуры значений переменных для расчета ускорения движения тела;
9. Написать инструкцию, обеспечивающую ввод с клавиатуры значений переменных расчета времени в пути зависимости от средней скорости движения;
10. Написать инструкцию, обеспечивающую ввод с клавиатуры значений переменных для вычисления объема цилиндра.

Задание 4. Создание программы с линейной структурой

Приступая к решению задач этого раздела, следует вспомнить, что:

Программы с линейной структурой являются простейшими и используются, как правило, для реализации несложных вычислений по формулам.

В программах с линейной структурой инструкции выполняются последовательно, одна за другой.

Для получения доступа к математическим функциям *sin*() и *cos*() и константы M_PI числа ПИ используйте заголовочный файл *math.h* для подключения математических функций

```
#include   "math.h"       //   sin и константа M_PI  - число  "ПИ"
```

Алгоритм программы с линейной структурой может быть представлен следующим образом:

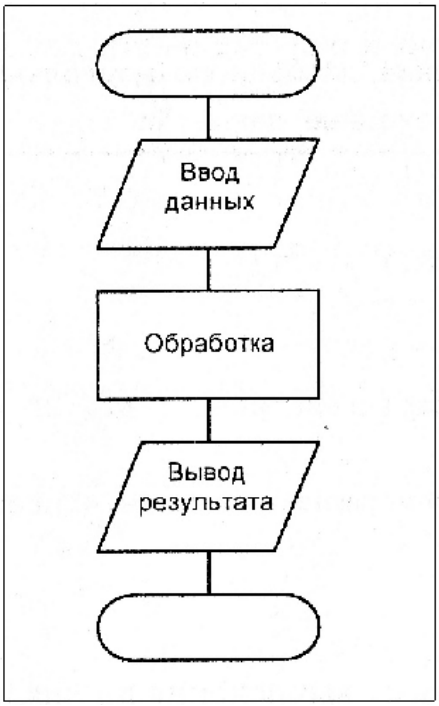

Рис. 1 Алгоритм программы с линейной структурой

Индивидуальное задание №4

Варианты заданий:

1. Написать программу вычисления площади прямоугольника. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление площади прямоугольника
Введите исходные данные:
Длина (см) -> 9
Ширина (см) ->7.5
Площадь прямоугольника: 67.50 кв. см.
```

2. Написать программу вычисления площади параллелограмма.
3. Написать программу вычисления объема параллелепипеда. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление объема параллелепипеда
Введите исходные данные:
Длинна (см) -> 9
Ширина (см) ->7.5
Высота (см) -> 5
Объем: 337.50 куб. см.
```

4. Написать программу вычисления площади поверхности параллелепипеда.

5. Написать программу вычисления объема куба. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление объема куба
Введите длину ребра (см) и нажмите <Enter> > 9.5
Объем куба: 857.38 куб. см.
```

6. Написать программу вычисления объема цилиндра. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление объема цилиндра
Введите исходные данные:
Радиус основания (см) -> 5
Высота цилиндра (см) -> 10
Объем цилиндра 1570.80 см. куб.
Для завершения нажмите <Enter>
```

7. Написать программу вычисления стоимости покупки, состоящей из нескольких тетрадей и карандашей. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление стоимости покупки
Введите исходные данные:
Цена тетради (руб.) -> 2.75
Количество тетрадей -> 5
Цена карандаша (руб.) -> 0.85
Количество карандашей -> 2
Стоимость покупки: 15.45 руб.
```

8. Написать программу вычисления стоимости покупки, состоящей из нескольких тетрадей и такого же количества обложек к ним. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление  стоимости покупки
Введите  исходные  данные:
Цена  тетради   (руб.)   ->  2.75
Цена  обложки   (руб.)   ->  0.5
Количество  комплектов   (шт.)   ->  7
Стоимость   покупки:   22.75   руб.
```

9. Написать программу вычисления стоимости некоторого количества (по весу), например яблок. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление стоимости покупки
Введите исходные данные:
Цена за килограмм (руб.) -> 8.5
Вес яблок (кг) -> 2.3
Стоимость покупки: 19.55 руб.
```

10. Написать программу вычисления площади треугольника, если известна длина основания и высота. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление площади треугольника
Введите исходные данные:
Основание (см) -> 8.5
Высота (см) -> 10
```

```
Площадь треугольника 42.50 кв. см.
```

11. Написать программу вычисления площади треугольника, если известны длины двух его сторон и величина угла между этими сторонами. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление площади треугольника
Введите (через пробел) длины сторон треугольника
-> 25 17
Введите величину угла между сторонами треугольника
->  30
Площадь треугольника: 106.25 кв. см.
```

12. Написать программу вычисления сопротивления электрической цепи, состоящей из двух параллельно соединенных сопротивлений R1 и R2 (R = R1R2/(R1 + R2)). Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление сопротивления электрической цепи при параллельном соединении
элементов. Введите исходные данные:
Величина первого сопротивления (Ом) -> 15
Величина второго сопротивления (Ом) -> 20
Сопротивление цепи: 8.57 Ом
```

13. Написать программу вычисления силы тока в электрической цепи I = U/R. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление силы тока в электрической цепи
Введите исходные данные: Напряжение (вольт) -> 36
Сопротивление (Ом) -> 1500
Сила тока: 0.024 Ампер
```

14. Написать программу вычисления расстояния между населенными пунктами, изображенными на карте. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление расстояния между населенными пунктами
Введите исходные данные:
Масштаб (количество километров в одном сантиметре) -> 120
Расстояние между точками (см) -> 3.5
Расстояние между точками 420 км.
```

15. Написать программу вычисления стоимости поездки на автомобиле. Исходные данные: расстояние (км); количество бензина (в литрах), которое потребляет автомобиль на 100 км пробега; цена одного литра бензина. Ниже приведен рекомендуемый вид экрана программы (данные, введенные пользователем, выделены полужирным).

```
Вычисление стоимости поездки на автомобиле
Расстояние (км) -> 67
Расход бензина (литров на 100 км пробега) -> 8.5
Цена литра бензина (руб.) -> 19.20
Поездка обойдется в 109.34 руб.
```