learnem.com

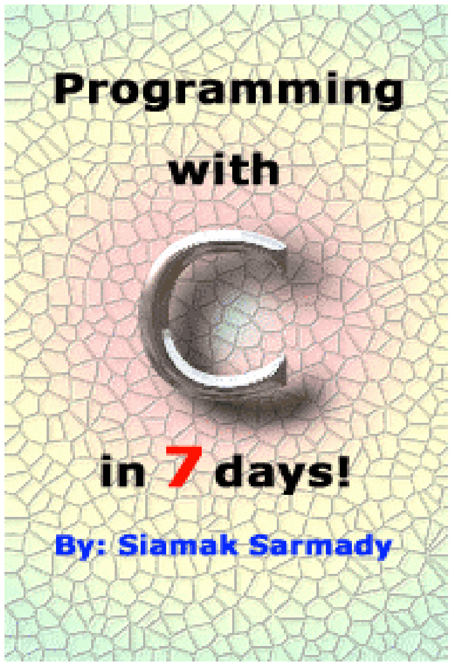
Learnem Group presents:

# Programming in C in 7 days !

Lessons 1- 7

By: Siamak Sarmady

Start Here



Course support : You can ask your course questions in Learnem support forums .

Please support us by visiting our support forums and asking your questions and answering questions of others.

Registration: You can also register for paid Programming in C email course. If you register you will benefit from strong one to one student support, personal tutor, more facilities, discount for other courses, access to students area and many more. Course fee for ***Programming in C email course*** is $20 only. Paid students will receive 13 advanced lessons in addition to 7 general lessons. You can register for paid course at:

Course Registration Here

# Programming in C

## Lesson 1

### 1-1 Introduction

This course is a quick tutorial on C Programming language. We assume that you are familiar with at least one of famous operating systems.

For this course you can use the following compilers or Programming Environments.

- cc in Unix or gcc in Linux operating systems
- Borland C or Turbo C in DOS operating system
- Visual C++ (create a simple win32 console application.) in Windows
- C++Builder (create a console application using console app wizard) in windows

We suggest a simple environment like Unix, Linux or DOS. Windows compilers are very complex for newbie programmers.

### 1-2 Your first C program

Let's write our first C program.

```
#include <stdio.h>
main()
{
printf("Hello World!\n");
}
```

First step for running this program is to make a text file containing above code. Be sure that the file is a pure text file. You must save the text file with .c extension.

Then you must compile the source code. The result of compile step is an executable file that you can run it.

To compile program under most of Unix operating systems you will use the command:

```
$ cc test.c
```

and under linux:

```
$ gcc test.c
```

The resulting executable file is a.out file. To run this executable you must type:

```
$./a.out
```

Program output must appear on your screen.

```
Hello World!
```

To compile the source code in other environments, you must use their Development Environment. You must consult your compiler User's Guide for this.

If you have problem working with your compiler you may ask your problem in our support forums. To reach our forums, visit:

http://www.learnem.com/forums/

---

## 1-3 Details of Test program

*#include <stdio.h>*

Tells C compiler to include the file "stdio.h" in this point of your C program before starting compile step. This "include file" contains several definitions , declarations etc.

*main()*

C program consist of one or more functions. Functions are building blocks of C programs. main() function is different from other functions by that it is the start point of program execution. Our program contains only function while complicated programs may contain thousands.

*{*

Opening brace marks the start of a block. Closing brace will mark its end. This one marks main () function start

*printf("Hello world!");*

This line of code prints the statement between quotation marks on your output screen. \n tells program to start a new line in output screen.

Each command line in C ends with ";" character. Control statements are exceptions. You will soon be able to
determine when you must use ; to end a line of code.

*}*

closes main() function. This program contains only one function while complicated programs may contain several functions.

---

## 1-4 Data types and variables

C uses several data types of data. These include characters, integer numbers and float numbers.

In C language you must declare a variable before you can use it.

Declaring a variable to be an integer or a character for example will let computer to allocate memory space for
storing and interpreting data properly.

---

## 1-5 Naming a variable

It's better that you use meaningful names for your variables even if this causes them to became long names. Also take this in mind that C is case sensitive. A variable named "COUNTER" is different from a variable named "counter".

Functions and commands are all case sensitive in C Programming language. You can use letters, digits and underscore _ character to make your variable names. Variable names can be up to 31 characters in ANSI C language.

The declaration of variables must take place just after the opening brace of a block. For example we can declare variables for main() function as below code:

```
main()
{
int count;
float sum,area;
.
.
.
}
```

First character in a variable name must be a letter or an underscore character. It cannot be a C programming
language-reserved word (i.e. Commands and pre defined function names etc)

An example for using variables comes below:

```
#include<stdio.h>
main()
{
int sum;
sum=12;
sum=sum+5;
printf("Sum is %d",sum);
}
```

General form for declaring a variable is:

```
Type name;
```

The line sum=sum+5; means: Increase value of sum by 5. We can also write this as sum+=5; in C programming language.

printf function will print the following:

```
Sum is 17
```

In fact %d is the placeholder for integer variable value that it's name comes after double quotes.

Common data types are:

```
int : integer
long : long integer
float : float number
double : long float
char : character
```

Other placeholders are:

```
%d : decimal integer
%ld : decimal long integer
%s : string or character array
%f : float number
%e : double ( long float)
```

printf () function used in this example contains two sections. First section is a string enclosed in double quotes. It is called a format string. It determines output format for printf function. Second section is "variable list" section.

We include placeholders for each variable listed in variable list to determine it's output place in final output text of printf function.

---

## 1-6 Control characters

As you saw in previous examples \n control character makes a new line in output. Other control characters are:

```
\n New line
\t tab
\r carriage return
\f form feed
\v vertical tab
```

---

## 1-7 Multiple functions

Look at this example:

```
#include<stdio.h>
main()
{
printf("I am going inside test function now\n");
test();
printf("\nNow I am back from test function\n");
}

test()
{
int a,b;
b=a+100;
printf("a is %d and b is %d",a,b);
}
```

In this example we have written an additional function. We have called this function from inside main function. When we call the function, program continues inside test () function and after it reached end of it, control returns just after test () function call in main ()

You see declaring a function and calling it, is an easy task.

Just pay attention that we used ";" when we called the function but not when we were declaring it.

We finish this lesson here. Now try to do lesson exercises and know the point that you will not learn anything if you do not do programming exercises.

## Exercises:

Course support:
Paid students must send exercises to their tutor. Tutor will return corrected exercise to the student. Others can ask their questions in Support forums in our web site.

http://www.learnem.com/forums/

1- What is the exact output result of this code.

```
#include <stdio.h>

main()
{
printf("Hi\nthere\nWhat is the output\n?");
}
```

2- Write a program that declares two floating numbers. Initialize them with float values. Then print
their sum and multiplication in two separate lines.

3- Write the output result of multiple function example in this lesson.

4- Why these variable names are not valid ?

```
test$var
my counter
9count
float
```

==========================================================

Back Next

# Programming in C

Lesson 2

## 2-1 Introduction

In previous lesson you learned about variables and printing output results on computer console. In this lesson we will learn how to get input values from console and output results after doing required calculations and processes.

## 2-2 Receiving input values from keyboard

Let's have an example that receives data values from keyboard.

Example 2-1:

```
#include<stdio.h>
main()
{
int a,b,c;
printf("Enter value for a :");
scanf("%d",&a);
printf("Enter value for b :");
scanf("%d",&b);
c=a+b;
printf("a+b=%d",c);
}
```

Output results:

```
Enter value for a : 10
Enter value for b : 20
a+b=30
```

* As scanf itself enters a new line character in receiving line we will not need to insert it in printf functions.

General form of scanf function is :

```
scanf("Format string",&variable,&variable,...);
```

Format string contains placeholders for variables that we intend to receive from keyboard. A '&' sign comes before each variable name that comes in variable listing. Character strings are exceptions from this rule . They will not come with this sign before them. We will study about character strings in this lesson.

You are not allowed to insert any additional characters in format string other than placeholders and some special
characters.

Entering even a space or other undesired character will cause your program not to work as expected. For now just insert placeholder characters in scanf format string.

The following example receives multiple variables from keyboard.

```
float a;
int n;
scanf("%d%f",&n,&a);
```

Pay attention that scanf function has no error checking capabilities built in it. Programmer is responsible for
validating input data (type, range etc. ) and preventing errors.

---

## 2-3 Variable arrays

Arrays are structures that hold multiple variables of the same data type. An array from integer type holds integer
values.

```
int scores[10];
```

The array "scores" contains an array of 10 integer values. We can use each member of array by specifying its index value.

Members of above array are scores[0],...,scores[9] so we can work with these variables like other variables:

```
scores[0]=124;
scores[8]=1190;
```

Example 2-2:

Receive 3 scores of a student in an array and finally
calculate his average.

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

```
maın()
{
int scores[3],sum;
float avg;

printf("Enter Score 1 : ");
scanf("%d",&scores[0]);
printf("Enter Score 2 : ");
scanf("%d",&scores[1]);
printf("Enter Score 3 : ");
scanf("%d",&scores[2]);
sum=scores[0]+scores[1]+scores[2];
avg=sum/3;

printf("Sum is = %d\nAverage = %f",sum,avg);
}
```

Output results:

```
Enter Score 1 : 12
Enter Score 2 : 14
Enter Score 3 : 15
Sum is = 41
Average = 13.000000
```

---

## 2-4 Character Strings

In C language we hold names, phrases etc in character strings. Character strings are arrays of characters. each
member of array contains one of characters in the string.

Look at this example:

Example 2-3:

```
main()
{
char name[20];

printf("Enter your name : ");
scanf("%s",name);

printf("Hello, %s , how are you ?",name);
}
```

Output Results:

```
Enter your name : Brian
Hello, Brian, how are you ?
```

If user enters "Brian" then the first member will contain 'B' , second will contain 'r' and so on.

C determines end of a string by a zero value character. We call this character as "NULL" character and show it with '\0' character. (It's only one character snd its value is 0, however we show it with two characters )

Equally we can make that string by assigning character values to each member.

```
name[0]='B';
name[1]='r';
name[2]='i';
name[3]='a';
name[4]='n';
name[5]=0; or name[0]='\0';
```

As we saw in above example placeholder for string variables is %s.

Also we will not use a '&' sign for receiving string values. we will understand the reason in future lessons.

---

## 2-5 Preprocessor

Preprocessor statements are those starting with '#' sign. An example is #include<stdio.h> statement that we used to include stdio.h header file into our programs.

Preprocessor statements are processed by a program called preprocessor before compilation step takes place. After preprocessor step, compiler starts its work.

---

## 2-6 ***#define*** preprocessor command

#define is used to define constants and aliases. Look at this example:

Example 2-4:

```
#include<stdio.h>

#define PI 3.14
#define ERROR_1 "File not found."
#define QUOTE "Hello World!"

main()
{
printf("Area of circle = %f * diameter", PI );
printf("\nError : %s",ERROR_1);
printf("\nQuote : %s",QUOTE);
}
```

Output results:

```
Area of circle = 3.140000 * diameter
Error : File not found.
Quote : Hello World!
```

As preprocessor step is performed before compilation step, compiler will see this program as below

```
#include<stdio.h>
main()
{
printf("Area of circle = %f * diameter", 3.14 );
printf("\error : %s","File not found.");
printf("\nQuote : %s","Hello World!");
}
```

In brief #define allows us to define symbolic constants . We usually use uppercase names for #define variables.

Pay attention that we do not use ';' after preprocessor statements.

---

## 2-7 Variable limitations

variable limit for holding variable values is related to the amount of memory space it uses in system memory.
Also in different operating systems and compilers different amount of memory is allocated for specific variable types . For example int type will use 2 bytes in DOS but 4 bytes in windows environment.

If you are not sure of limitations of variable types in your system you must see your compiler information.

Also we can use sizeof() function to determine size of a variable or variable type.

Example 2-5:

```
main()
{
int i;
float f;

printf("Integer type uses %d bytes of memory.", sizeof(i));
printf("float type uses %d bytes of memory.", sizeof(float));
}
```

You see we can use both a variable and a variable type as a parameter to sizeof() function.

In this table you will see limitations of Turbo C and Microsoft C in DOS operating system.

| | Bytes used | Range |
|---|---|---|
| char | 1 | 256 |
| int | 2 | 65536 |
| short | 2 | 65536 |
| long | 2 | 4 billion |
| float | 4 | 6 digits * 10e38 |
| double | 8 | 10 digits * 10e308 |

We have also two kinds of variables from above types in C programming language. Signed and unsigned. Signed variables support negative numbers too while unsigned values only support positive numbers.

If signed variable is used above range will be divided by two. For example signed int range is (-32768,+32767)

You can declare a variable as signed or unsigned by adding "signed" or "unsigned" keywords before type name.

Example:

```
signed int a;
unsigned int b;

a=32700;
b=65000;
```

We are allowed to assign values larger than 32767 to variable "b" but not to variable "a". C programming language may not complain if we do so but program will not work as expected.

Alternatively we can assign negative numbers to "a" but not to "b".

Default kind for all types is signed so we can omit signed keyword if we want a variable to be signed.

## Exercises:

Course support:
Paid students must send exercises to their tutor. Tutor will return corrected exercise to the student. Others can ask their questions in Support forums in our web site.

http://www.learnem.com/forums

1- Write a program that asks for work hours, wage per hour and tax rate and outputs payable money for a person.

2- Using arrays write a program that receives tax rate, work hours and wage per hour for two persons in arrays and then calculates and outputs payable money for each of them.

3- Write a program that asks for your name and outputs 3 first characters of your name each in a separate line.
Use %c as placeholder of a single character in printf format string.

==================================================================

Back Next

# Programming in C

## Lesson 3

### 3-1 Operators

There are many kinds of operators in each programming language. We mention some of these operators here:

() Parentheses
+ Add
- Subtract
* Multiply
/ Divide

There are also some other operators which work differently:

% Modulus
++ Increase by one
-- Decrease by one
= Assignment

sizeof( ) return value is the size of a variable or type inside parentheses in bytes. It is actually the size that
variable takes in system memory.

Examples:

```
 c=4%3            c will be equal to 1 after execution of this command.
```

|  |  |
|---|---|
| `i=3;` `i=i*3;` | i will be equal to 9 |
| `f=5/2` | if f is integer then it will be equal to 2. If it is a float type variable its value will be 2.5 |
| `j++` | Increases the value of j by one. |
| `j--` | Decreases value of j by one |
| `sizeof(int)` | returned value is 2 in dos and 4 in windows |
| `int a=10;` `c=sizeof(a);` | c will be 2 in dos and 4 in windows as the size of integer is different in different Os. |

## 3-2 Loops

Sometimes we want some part of our code to be executed more than once. We can either repeat the code in our program or use loops. It is obvious that if for example we need to execute some part of code for a hundred times it is not acceptable to repeat the code.

Alternatively we can use our code inside a loop.

```
while(not a hundred times)
{
code
}
```

There are many kinds of loop commands in C programming language. We will see these commands in next sections.

## 3-3 while loop

while loop is constructed of a condition and a command or a block of commands that must run in a loop. As we have told later a block of commands is a series of commands enclosed in two opening and closing braces.

```
while( condition )
command;

while( condition )
{
block of commands
}
```

Loop condition is a Boolean expression. A Boolean expression has a value of 0 or 1 at any given time.

Example 3-1:

```
#include<stdio.h>
main()
```

```
{
int i=0;

while( i<100 )
 {
 printf("\ni=%d",i);
 i=i+1;
 }
}
```

In above example i=i+1 means: add 1 to i and then assign it to i or simply increase its value. As we saw later, there is a special operator in C programming language that does the same thing. We can use the expression i++ instead of i=i+1.

---

## 3-3 Type Conversion

From time to time you will need to convert type of a value or variable to assign it to a variable form other type. This type of conversions may be useful in other situations for example you can convert type of a variable to become compatible with a function with different type of arguments.

Some rules are implemented in C programming language for this purpose.

1- Automatic type conversion takes place in some cases. Char is automatically converted to int. Unsigned int will
be automatically converted to int.

2- If there are two different types in an expression then both will convert to better type.

3- In an assignment statement, final result of calculation will be converted to the type of variable that result is
being assigned to it.

For example if you add two values from int and float type and assign it to a double type variable, result will be
double.

---

## 3-4 Using loops in an example

Write a program to accept scores of a person and calculate sum of them and their average and print them.

Example 3-2 :

```
#include<stdio.h>
main()
{
int count=0;
float num=0,sum=0,avg=0;

printf("Enter score : ");
scanf("%f",&num);
while(num>=0)
 {
  sum=sum+num;
  count++;
  printf("Enter score : ");
  scanf("%f",&num);
 }

avg=sum/count;
printf("\nAverage=%f",avg);
printf("\nSum=%f",sum);
}
```

In this example we get first number and then enter the loop. We will stay inside loop until user enters a value smaller than 0. If user enters a value lower than 0 we will interpret it as STOP receiving scores.

Here are the output results of a sample run:

```
Enter score : 12
Enter score : 14
Enter score : -1

Average=13.000000
Sum=26.000000
```

When user enters -1 as the value of num, logical expression inside loop condition becomes false as num>=0 is not acceptable.

Just remember that, while loop will continue until the logical condition inside its parentheses is true.

---

## 3-4 for loop

As we told later, there are many kinds of loops in C programming language. We will learn about for loop in this section.

For loop is something like while loop but it is more complex. For loop is constructed from a control statement that
determines how many times the loop will run and a command section. Command section is either a single command or a block of commands.

```
for( control statement )
command;

for( control statement )
{
block of commands
}
```

Control statement itself has three parts:

```
for(initialization; test condition; run every time command)
```

Initialization part is performed only once at for loop start. We can initialize a loop variable here. Test condition is
the most important part of the loop. Loop will continue to run if this condition is True. If condition becomes false
then loop will terminate.

‘Run every time command’ section will be performed every loop cycle. We use this part to reach the final condition for terminating the loop. For example we can increase or decrease loop value so that we no more have the condition to continue performing loop and therefore terminate the loop.

At this step we rewrite example 3-1 with for loop. Just pay attention that we no more need I=I+1 for increasing loop variable. It is inserted inside for condition phrase.

Example 3-3:

```
#include<stdio.h>
main()
{
int i=0;

for(i=0;i<100;i++ )
printf("\ni=%d",i);

}
```

Example 3-4:

Write a program that gets temperatures of a week and calculate average temperature for that week.

```
#include<stdio.h>
main()
{
int count=0;
float num=0,sum=0,avg=0;

for(count=0;count<7;count ++)
{
  printf("Enter score : ");
  scanf("%f",&num);
  sum=sum+num;
}

avg=sum/7;
printf("\nAverage=%f",avg);
printf("\nSum=%f",sum);
}
```

---

## 3-5 End

We learned most important types of loop commands in this lesson. In next lesson we will have more useful examples on loops.

## Exercises:

**Course support:**
Paid students must send exercises to their tutor. Tutor will return corrected exercise to the student. Others can ask their questions in Support forums in our web site.

http://www.learnem.com/forums

1- Using while loop, write a program that prints alphabets ‘a’ through ‘z’ in separate lines. Use %c in your format string. You can increment a character type variable with ++ operand.

2- Write a program that accepts time in seconds format and prints it in minutes and seconds. For example if you enter 89 it must print : 1:29

3- Using for loop, write a program that accepts net sales of a shop for each day. Then calculate sales for a month then deduct %5 taxes and finally calculate Average of sales after tax deduction for a day. Print results on the screen.

==========================================================

Back Next

# Programming in C

## Lesson 4

### 4-1 More about "for" loops

As we saw in previous lesson we can use "for" statement to create a loop for executing a command or a block of commands.

```
for(initialization; test condition; run every time command)
    command;
```

There are three parts inside condition phrase. Initialization statement you can initialize any variable including loop
counter or any other variable. In condition statement you can use any kind of logical statement that will specify the condition for loop execution. As we told later if this condition becomes false loop execution will terminate.

Last part is a statement that will be executed every time loop is being executed. In previous examples we used a statement like i++ and count++. These will increase the value of a variable each time loop is executed. Increase in this variable can change the loop condition section to false if the condition is based on this variable.

In this lesson we first see a more complex example of a "for" loop and then continue to new concepts.

Example 4-1:

Below example will print a multiplication chart (from 1*1 to 9*9). Run the program and see the results.

```
#include<stdio.h>
main()
{
int i,j;

for(i=1;i<10;i++)
 {
 for(j=1;j<10;j++)
     printf("%3d",i*j);
 printf("\n");
 }
}
```

---

## 4-2 "if" statement

Sometimes you will need a command or a block of commands to be executed when a condition exists or vice versa, when a condition does not exist.

```
if(condition)
   command;

if(condition)
{
   block of commands;
}
```

If statement is a branching statement because it provides a way to select a path from several paths in a program. If condition is true the command or block of commands will be executed.

Example 4-2:
What does this program do?

```
#include<stdio.h>
main()
{
 int n;
 printf("Enter a number: ");
 scanf("%d",&n);
 if(n>=0)
  printf("Number is positive !\n");
 if(n<0)
  printf("Number is negative !\n");
}
```

Now let's see a more complex example.

Example 4-3:
Write a program to solve a second-degree equation $ax^2+bx+c=0$.

```
#include<stdio.h>
#include<math.h>
main()
{
```

```
float delta,a,b,c,x1,x2;

printf("Enter a : ");
scanf("%f",&a);
printf("Enter b : ");
scanf("%f",&b);
printf("Enter c : ");
scanf("%f",&c);
delta=b*b-(4*a*c);

if(delta<0)
 {
 printf("Equation has no answer !\n");
 exit(0);
 }

if(delta==0)
 {
 x1=-b/(2*a);
 printf("Equation has two equal answers !\n");
 printf("x1=x2=%f",x1);
 exit(0);
 }

x1=(-b+sqrt(delta))/(2*a);
x2=(-b-sqrt(delta))/(2*a);
printf("\nX1=%f",x1);
printf("\nX2=%f\n",x2);
}
```

---

## 4-3 More complex "if" statements

Simple form of "if" statement gives you the choice of executing or skipping a command or block of commands. If in a program you decide to execute a command when condition is true and execute another command when it is false, you will need two simple "if" statements.

```
if(condition)
  command;
if(!condition)
  command
```

! Sign reverses the logical value of a Boolean expression. If it is true the result will become false with '!' sign and
vice versa.

There is another option. "if" statement has more complex forms. Below you see another form of "if" statement.

```
if(condition)
  command;
else
  command
```

In above statement there is an additional "else" section. When condition is true first command (or block of commands) is executed otherwise "else" section will be executed.

Example 4-3:

```
#include<stdio.h>
main()
{
int n;
printf("Enter a number: ");
scanf("%d",&n);
if(n>=0)
 printf("Number is positive !\n");
else
 printf("Number is negative !\n");
}
```

---

## 4-4 A useful example and more complex "if" statement

Example 4-5:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int choice;

while(1)
{
 printf("\n\nMenu:\n");
 printf("1- Math Program\n2- Accounting Program\n");
 printf("3- Entertainment Program\n4- Exit");
 printf("\n\nYour choice -> ");
 scanf("%d",&choice);

 if(choice==1)
   printf("\nMath Program Runs. !");
 else if(choice==2)
   printf("\nAccounting Program Runs. !");
 else if(choice==3)
   printf("\nEntertainment Program Runs. !");
 else if(choice==4)
   {
   printf("\nProgram ends.\n");
   exit(0);
   }
 else
   printf("\nInvalid choice");
 }
}
```

Above example is a very interesting example of what is used in most of menu driven programs. A loop that continues forever prints menu items on screen and waits for answer. Every time an answer is entered, proper action is done and again menu appears to accept another choice.

Loop continues forever unless you enter 4 as the answer for menu question. When this takes place the 'exit (0)' function terminates the program.

A more complex form of "if" statement is used here.

```
if(choice==1)
   command;
else if(choice==2)
   command;
else if(choice==3)
   command;
else if(choice==4)
  {
  block of commands;
  }
else
  command;
```

This kind of "if" command is used in cases that you need multiple actions in case of different values or states. At the end of if statement there is an else section again. So you can do whatever you want when answer does not matches any of your conditions.

---

## 4-5 End

With this lesson you must be able to write many useful programs.

As there are many commands and programming techniques you will not be able to remember all of them. So you must start programming. Start with lesson exercises and continue with more programs otherwise all your efforts will be useless in a while.

I always tell this sentence in my programming classes

*" No one becomes a programmer without programming "*

---

## Exercises:

Course support:
Paid students must send exercises to their tutor. Tutor will return corrected exercise to the student. Others can ask their questions in Support forums in our web site.

http://www.learnem.com/forums

---

1- Write a program that accepts 10 scores between 0 and 20 for each student. (use "for" loop) Then calculate average for the student. We want to determine an alphabetical score for each student according below table.

| | |
|---|---|
| A | 16-20 |
| B | 12-16 |
| C | 10-12 |
| D | 7-10 |

E 4-7
F 0-4

Print both average score in numerical and alphabetical form.

2- Rewrite example 4-5 to do the following tasks:

1- Add two numbers
2- Subtract two numbers
3- Multiply two numbers
4- Exit

Write necessary program inside a block to accept two numbers and perform necessary action for each menu choice. Program execution must continue until user enters 4 as menu choice.

==========================================================

Back Next

# Programming in C

## Lesson 5

### 5-1 "switch ... case" structure

In previous lesson we saw how we can use "if" statement in programs that want to choose a way among several alternatives. We can use "if" statement yet but it is better to use "switch" statement which is created for situations that there are several choices.

```
switch(...)
{
case ... : command;
           command;
           break;

case ... : command;
           break;

default :
           command;
}
```

In the above switch command we will be able to run different series of commands in case of each alternative
state.

Example 5-1:

Rewrite example 4-5 of previous lesson and use switch command instead of "if" statement.

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int choice;

while(1)
 {
 printf("\n\nMenu:\n");
 printf("1- Math Program\n2- Accounting Program\n");
 printf("3- Entertainment Program\n4- Exit");
 printf("\n\nYour choice -> ");
 scanf("%d",&choice);


 switch(choice)
  {
  case 1 : printf("\nMath Program Runs. !");
           break;

  case 2 : printf("\nAccounting Program Runs. !");
           break;

  case 3 : printf("\nEntertainment Program Runs. !");
           break;

  case 4 : printf("\nProgram Ends. !");
           exit(0);

  default:
           printf("\nInvalid choice");

   }
  }
}
```

In "switch" command each 'case' acts like a simple label. A label determines a point in program which execution must continue from there. Switch statement will choose one of 'case' sections.

After entering case portion, execution continues until it reaches a break statement.

"break" statements have vital rule in switch structure. If you remove these statements, program execution will continue to next case sections and all commands until the end of "switch" block will be executed.

As we told, This is because each 'case' acts exactly as a label. When program execution is transferred to a case
section it will continue running to the end of switch block. The only way to end execution of statements in switch block is using break statements at the end of each section.

In one of case sections we have not used "break". This is because we have used a termination command "exit(0)" and a break statement will not make sense.

"default" section will be executed if none of the case sections match switch comparison.

Parameter inside switch statement must be of type int (or char) .

Using a variable in case sections is not allowed. This means that you are not allowed to use a statement like below in your switch block.

```
case i: something;
        break;
```

---

## 5-2 break statement

We used "break" statement in switch...case structures in previous part of lesson. We can also use "break" statement inside loops to terminate a loop and exit it.

Example 5-2:

```
while (num<20)
{
 printf("Enter score : ");
 scanf("%d",&scores[num]);
 if(scores[num]<0)
 break;
}
```

In above example loop execution continues until either num>=20 or entered score is negative.

Example 5-3:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int choice;


while(1)
  {
  printf("\n\nMenu:\n");
  printf("1- Math Program\n2- Accounting Program\n");
  printf("3- Entertainment Program\n4- Exit");
  printf("\n\nYour choice -> ");
  scanf("%d",&choice);

  switch(choice)
    {
    case 1 : printf("\nMath Program Runs. !");
             break;

    case 2 : printf("\nAccounting Program Runs. !");
             break;

    case 3 : printf("\nEntertainment Program Runs. !");
             break;
    case 4 : printf("\nProgram Ends. !");
             break;

    default:
             printf("\nInvalid choice");

    }

  if(choice==4) break;

  }
}
```

In above example we have used a break statement instead of exit command used in previous example.

As a result of this change, we must use a second break statement inside while loop and outside switch block.

If the choice is 4 then this second break command will terminate while loop and we reach the end of main function and when there is no more statements left in main function program terminates automatically.

---

## 5-3 getchar()

getchar() function is an alternative choice when you want to read characters from input. This function will get characters from input and return it to a variable or expression in our program.

```
ch=getchar();
```

There is a function for sending characters to output too.

```
putchar(ch);
```

Example 5-3:

```
#include<stdio.h>
#include<conio.h>
main()
{
 char ch;

 while(ch!='.')
  {
  ch=getchar();
  putchar(ch);
  }
}
```

First look at output results and try to guess the reason for results.

Console Screen:

```
test             <--This is typed by us
test              <--output of putchar after we pressed enter
again.testing     <--Typed string includes a '.' character
again.            <--Loop terminates when it reaches '.' char
```

Above program reads any character that you have typed on keyboard until it finds a '.' character in input characters.

Each key press will be both shown on console and captured to a buffer. This buffer will be delivered to 'ch' variable after pressing enter key (not before this).

So input characters are buffered until we press enter key and at that moment program execution continues running statements that follow getchar() function (These are loop statements).

If there is a '.' character in buffered characters, loop execution continues sending characters to console with putchar() function until it reaches '.' and after that stops the loop and comes out of while loop.

If it does not encounter '.' in characters it returns to the start of loop, where it starts getchar() function
again and waits for user input.

First 'test' string in output is the result of our key presses and second 'test' is printed by putchar statement after pressing enter key.

In some operating systems and some C Programming language compilers, there is another character input function "getch". This one does not buffer input characters and delivers them as soon as it receives them.

Borland C for Dos as an example, supports this function and It is defined in conio.h header file in both Borland C and Turbo C compilers.

With this function we will be able to check validity of each key press before accepting and using it. If it is
not valid we can omit it.

Just pay attention that some compilers do not support getch().

Look a this below example.

Example 5-4:

```
//code works on borland c
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
main()
{
char choice;


while(1)
{
 printf("\n\nMenu:\n");
 printf("1- Math Program\n2- Accounting Program\n");
 printf("3- Entertainment Program\n4- Exit");
 printf("\n\nYour choice -> ");
 choice=getch();

 switch(choice)
   {
   case 1 : printf("\nMath Program Runs. !");
            break;

   case 2 : printf("\nAccounting Program Runs. !");
            break;

   case 3 : printf("\nEntertainment Program Runs. !");
            break;

   case 4 : printf("\nProgram Ends. !");
            exit(0);

   }
 }

}
```

In above example we have rewritten example 5-1. This time we have used getch() function instead of scanf() function.

If you test scanf based example you will see that it does not have any control on entered answer string. If user inserts invalid choices or string it can corrupt the screen.

In getch function user can insert one character at a time. Program immediately gets the character and tests it to see if it matches one of the choices.

In this example we have omitted optional "default" section in switch...case .

If user presses an invalid key while loop will continue without entering any of "case" sections. So every invalid
key press will be omitted and only valid key presses will be accepted.

---

## 5-3 "continue" statement

Continue statement can be used in loops. Like break command "continue" changes flow of a program. While it does not terminate the loop.

It just skips the rest of current iteration of a loop and returns to start point of loop.

Example 5-5:

```
while((ch=getchar())!='\n')
{
if(ch=='.')
   continue;
putchar(ch);
}
```

In above example, program accepts inputs and omits '.' character. If you run this example you will see that results will not appear until program execution is complete or in fact when enter key "\n" is pressed.

As we told later this delay is because getchar() function is a buffered input function.

---

## 5-5 End

Once again we reach the end of our lesson. In next lesson we will work on more complex input/output functions. We will also see how we can write our own functions.

---

## Exercises:

Course support:
Paid students must send exercises to their tutor. Tutor will return corrected exercise to the student. Others can ask their questions in Support forums in our web site.

http://www.learnem.com/forums

---

1- Write a program that reads input until enter key is pressed ('\n' is found in input) and prints the number of alphabets and also number of spaces in input string. Use getchar() for reading input characters from console.

2- Write a program that reads input and replaces below characters with the character that comes in front of them. Then writes the output on your screen.

a -> c
f -> g
n -> l
k -> r

Use getchar() and switch...case to write the program.

==========================================================

Back Next

# Programming in C

## Lesson 6

### 6-1 Introduction

As we told in lesson 1, a C program consists of one or more functions. main() is a special function because program execution starts from main() function.

A function is combined of a block of code that can be called or used anywhere in a program by calling function name. Body of a function starts with '{' and ends with '}' . As you see this is the same as main function in our previous programs.

Example below shows how we can write a simple function.

Example 6-1:

```
#include<stdio.h>

/*Function prototypes*/
myfunc();

main()
{
 myfunc();
}

myfunc()
{
 printf("Hello, this is a test\n");
}
```

In above function we have put a section of program in a separate function. Function body can be very complex.

After creating a function we can call it using its name. Functions can call each other. Even a function can call itself. This is used in recursive algorithms. But must be sure that this will not enter our program in an infinite loop.

Most of C programming language commands are functions. For example “printf” is a function that accepts one or more arguments and do printing.

---

## 6-2 Reasons for using functions

There are many reasons for using functions.

a. A part of code may be reused many times in different parts of program.

b. Program will be divided to separate blocks. Each block will do a special job. Understanding and design of programs will be simplified in this way.

c. A block of code can be executed with different numbers of initial parameters. These parameters are passed to
function with arguments.

Assume we want to read scores of students from a file, calculate their average and print results. If we want the
program to be written just inside main() function block we will have a large main() function. But if we spread
the program to different functions, we will have a small main() function.

```
main()
{
readscores();
calculate()
writetofile();
printsresults();
}
```

Now let’s look at this example:

Example 6-2:

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

add();
subtract();
multiply();

main()
{
 int choice;
 while(1)
   {
   printf("\n\nMenu:\n");
   printf("1- Add\n2- Subtract\n");
   printf("3- Multiply\n4- Exit");
   printf("\n\nYour choice -> ");
   scanf("%d",&choice);


   switch(choice)
    {
    case 1 : add();
             break;

    case 2 : subtract();
             break;

    case 3 : multiply();
             break;

    case 4 : printf("\nProgram Ends. !");
             exit(0);

    default:
             printf("\nInvalid choice");

    }
  }
}

add()
{
 float a,b;

 printf("\nEnter a:");
 scanf("%d",&a);
 printf("\nEnter b:");
 scanf("%d",&b);

 printf("a+b=",a+b);
}

subtract()
{
 float a,b;

 printf("\nEnter a:");
 scanf("%d",&a);
 printf("\nEnter b:");
 scanf("%d",&b);

printf("a-b=",a-b);
}


multiply()
```

```
{
float a,b;

 printf("\nEnter a:");
 scanf("%d",&a);
 printf("\nEnter b:");
 scanf("%d",&b);

printf("a*b=",a*b);
}
```

---

## 6-2 Function arguments

Functions are able to accept initial values for some variables. These values are accepted as function arguments.

Example 6-3:

```
#include<stdio.h>
/* use function prototypes */
sayhello(int count);
main()
{
 sayhello(4);
}

sayhello(int count)
{
 int c;

 for(c=0;c<count;c++)
  printf("\nHello");
}
```

In above example we have called sayhello() function with the parameter “4”.

Function receives an input value and assigns it to “count” variable before starting execution of function body.

sayhello() function will print a message for ‘count’ times on the screen.

Pay attention that we must use a function prototype before we can call a function inside main() or another function in our program. We usually insert these prototypes after precompiler commands in our program. This is usually before main() function at the top of our program.

You can copy function header exactly for prototype section.

If you do not use prototypes of your functions you will get an error in most of new C compilers. This error mentions that you need a prototype for each of your functions.

---

## 6-3 Function return values

In mathematics we generally expect a function to return a value. It may or may not accept arguments but it always returns a value.

```
y=f(x)
y=f(x)=x+1
y=f(x)=1 (arguments are not received or not important)
```

In C programming language we expect a function to return a value too. This return value has a type as other values in C. It can be integer, float, char or anything else. Type of this return value determines type of your function.

Default type of function is int or integer. If you do not indicate type of function that you use, it will be of type
int.

As we told later every function must return a value. We do this with return command.

```
Sum()
{
 int a,b,c;
 a=1;
 b=4;
 c=a+b;
 reurn c;
}
```

Above function returns the value of variable C as the return value of function.

We can also use expressions in return command. For example we can replace two last lines of function with ‘return a+b;’

If you forget to return a value in a function you will get a warning message in most of C compilers. This will warn you that your function must return a value. Warnings do not stop program execution but errors stop it.

In our previous examples we did not return any value in our functions. For example you must return a value in main() function.

```
main()
{
.
.
.
return 0;
}
```

Default return value for an int type function is 0. If you do not insert ‘return 0’ or any other value in your main()
function a 0 value will be returned automatically.

---

## 6-4 “void” return value

There is another type of function yet. It is “void” type. Void type function is a function that does not return a
value.

For example you can define a function that does not need a return value as “void”.

```
void test ()
{
 /* fuction code comes here but no return value */
}
```

void functions cannot be assigned to a variable because it does not return value. So you cannot write:

```
a=test();
```

Using above command will generate an error.

In next lesson we will continue our study on functions. This will include function arguments and more.

---

## Exercises:

Course support:
Paid students must send exercises to their tutor. Tutor will return corrected exercise to the student. Others can ask their questions in Support forums in our web site.

http://www.learnem.com/forums

---

1- Write a complete program that accepts a decimal value and prints its binary value. Program must contain a function that accepts a decimal number and prints the binary value string.

2- Write a complete program that asks for radius of a circle and calculates its area using a function with return value of float type.

==================================================================

Back Next

# Programming in C

## Lesson 7

### 7-1 Function Arguments

In previous lesson you saw how we could send a single value to a function using a function argument.

In fact you can use more than one argument in a function. Example 7-1 will show you how you can do this.

Example 7-1:

```
#include<stdio.h>

int min(int a,int b);
main()
{
 int m;
 m=min(3,6);
 printf("Minimum is %d",m);

 return 0;
}

int min(int a,int b)
{
 if(a<b)
  return a;
 else
```

```
  return b;
}
```

As you see you can add your variables to arguments list easily. Function prototype is a copy of function header. The only difference is that prototype ends with semicolon ';' but function header does not.

---

## 7-2 Call by value

C programming language function calls use call by value method. Let's see an example to understand this subject
better.

Example 7-2:

```
#include<stdio.h>

void test(int a);

main()
{
 int m;
 m=2;

 printf("\nM is %d",m);
 test(m);
 printf("\nM is %d",m);

 return 0;
}

void test(int a)
{
 a=5;
}
```

In main() function, we have declared a variable m. We have assigned value '2' to m. We want to see if function call is able to change value of m as you may expect because we have modified its value inside test() function. What do you think?

Program output results is:

```
M is 2
M is 2
```

So you see function call has not changed the value of argument. This s is because function-calling method only sends value of variable m to function and it does not send variable itself. Actually it places value of variable m in a memory called stack and then function retrieves the value without having access to variable itself.

This is why we call this method of calling "call by value".

---

## 7-3 Call by reference

There is another method of sending variables that we can call it “Call by reference”. This second method enables
function to modify value of argument variables used in function call.

We will first see an example and then we will describe it.

Example 7-3:

```
#include<stdio.h>
void test(int *ptr);

main()
{
 int m;
 m=2;

 printf("\nM is %d",m);
 test(&m);
 printf("\nM is %d",m);

 return 0;
}

void test(int *ptr)
{
 *ptr=5;
}
```

To be able to modify the value of a variable that is used as an argument in a function, function needs memory address of the variable.

In C programming language ‘&’ operator gives the address at which the variable is stored. For example if ‘m’ is a
variable of type ‘int’ then ‘&m will give us the start memory address of our variable.

We call this resulting address ‘a pointer’.

```
ptr=&m;
```

In above command ptr variable will contain memory address of variable m. This method is used in some of standard functions in C. For example scanf function uses this method to be able to receive values from console keyboard and put it in a variable. In fact it places received value in memory location of the variable used in  function.

```
scanf("%d",&a);
```

Now that we have memory address of variable we must use an operator that enables us to assign a value or access to value stored in that address.

As we told we can find address of variable ‘a’ using ‘&’ operator.

```
ptr=&a;
```

Now we can find value stored in variable ‘a’ using ‘*’ operator:

```
val=*ptr; /* finding the value ptr points to */
```

We can even modify the value inside the address:

```
*ptr=5;
```

Let’s look at our example again. We have passed pointer (memory address) to function. Now function is able to modify value stored inside variable.

If you run the program, you will get these results:

```
M is 2
M is 5
```

So you see this time we have changed value of an argument variable inside called function.

---

## 7-4 A useful example, Bubble Sort

In this section we will write a program that sorts an array using famous bubble sort algorithm.

We have written a function “swap” that swaps values stored in two memory locations. Then we use this function to perform sort.

Bubble sort compares each two cell of array and if they are out of sort, swaps them. If we perform these compares and swaps for n-1 times on all array cells then our array will be completely sorted.

Example 7- Bubble sort:

```
#include<stdio.h>

void swap(int *a,int *b);

main()
{
 int ar[5],i,j,n;

 ar[0]=7;ar[1]=3;ar[2]=9;ar[3]=2;ar[4]=11;

 printf("Array before sort:\n\n");
 for(i=0;i<5;i++)
 printf("ar[%d]=%d\n",i,ar[i]);

 n=5; /*numberof items in sort array*/

 for(i=0;i<n-1;i++)
 for(j=0;j<n-1;j++)
  {
  if(ar[j]>ar[j+1])
  swap(&ar[j],&ar[j+1]);
  }

 printf("Array after sort:\n\n");
 for(i=0;i<5;i++)
   printf("ar[%d]=%d\n",i,ar[i]);
  return 0;
}


void swap(int *a,int *b)
{
 int temp;

 temp=*a;
 *a=*b;
 *b=temp;
}
```

---

7-5 End

We have learned fundamentals of programming in C in these 7 first lessons but there are a lot of things we must learn.

In next lessons we will learn more about arrays, pointers, strings, files, structures, memory management etc.

But it’s enough for now!

---

## Exercises:

Course support:
Paid students must send exercises to their tutor. Tutor will return corrected exercise to the student. Others can ask their questions in Support forums in our web site.

http://www.learnem.com/forums

---

1- Write a function with two arguments. First argument is “ch” from character type and second variable is “repeat” from int type. When you pass a character and a repeat number (int) it prints character for “repeat” times on console.

2- Write a program that reverses the order of members of an array. Use addresses and pointers to displace values in array.

==================================================================

Back Next

## Learnem, Learn Everything by Mail

Free and Fee Based Email Courses, Learning Resources, Tutorials, Ezines

## OnlineProgrammer (.org) a Society for Web and Network Programmers

Free Resources for Programmers, Source Code, Ezines, Tutorials, Job Links