Contents

Lab # 01

How to install Dev C++ Compiler, Edit, compile and Run a C++ Program.

1.1 Objective:

1. Learn basic components of a C program.
2. Learn compilation process.
3. Learn how to use Dev C++.
4. Learn how to write, edit & run C-Program.

1.2 Scope:

The student should know the following at the end of this lab:

1. Problem solving
2. How to install run and compile.
3. Basis structure of program
4. Writing Complete Programs

1.3 Useful Concepts:

A C-program:

A typical structure of a C program consists of:

- Pre-processor Directives that always begin with # (e.g. #include <stdio.h>)
    - The two most frequent commands are #include and #define.
    - #include command will include header files that have the definition of a function used in the program like getche() function that is in stdio.h file.
    - Note: #include <*file*> tells the compiler to look for *file* where system include files are held
    - #include "*file*" tells the compiler to look for *file* in the current directory where the program was run from.
    - #define is used to replace a text with a value, e.g. #define PI 3.141593

- Functions

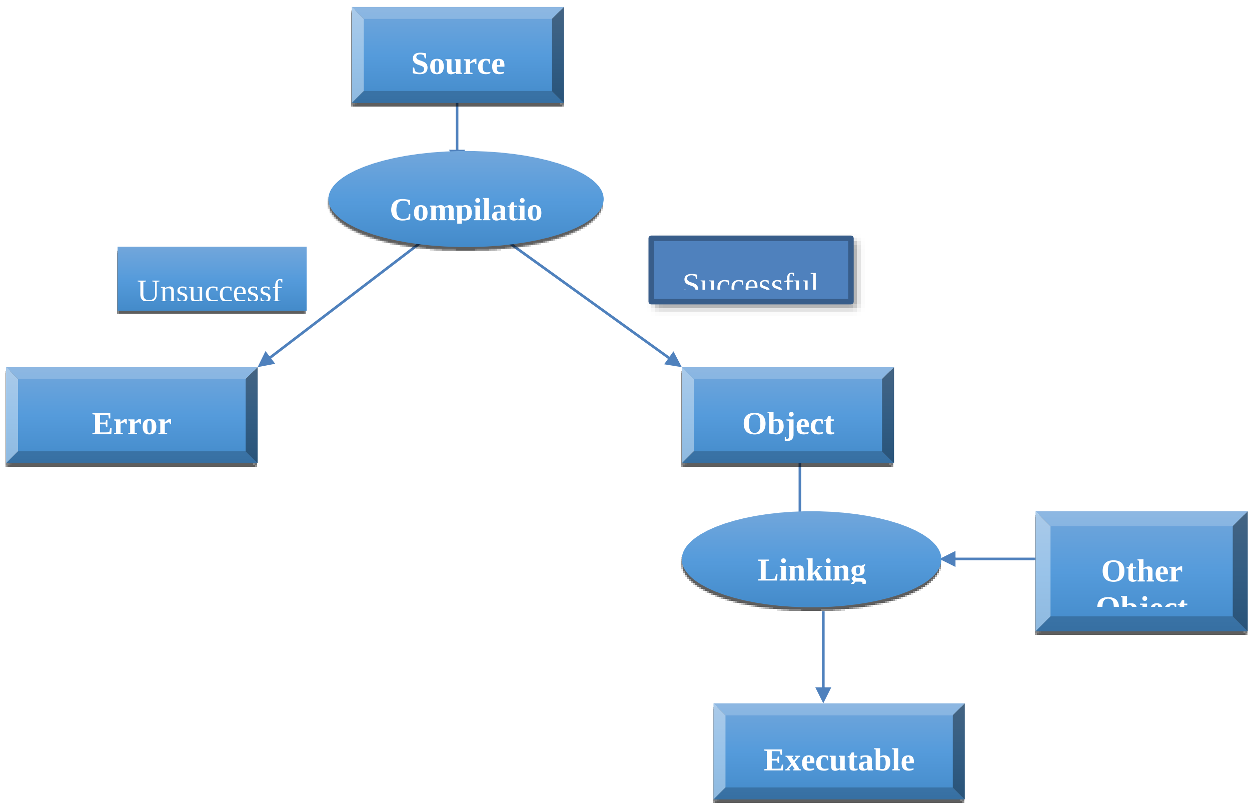
A function is a block of statements that make a specific kind of processing. Every C program has a main() function. The execution starts from main().

- Variables / identifiers

These are used to store data in a program. You must declare a variable before you use it.

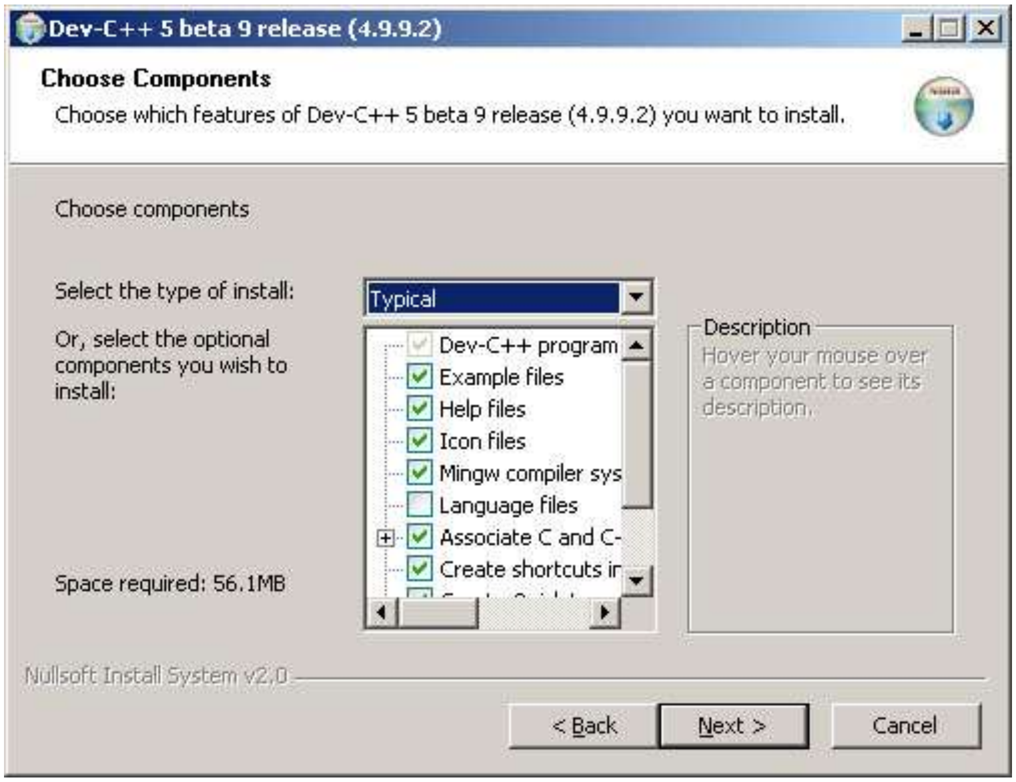
Here is a general structure of a C program.

pre-processor directives	`#include <iostream.h>`
main function heading	`#include <conio.h>`
{	`int main(void)`
declarations	`{`
executable statements	`cout<<"hello World";`
}	`getche();`
	`}`



Download and Install Dev C++

1. Download Dev C++ from given link. http://www.bloodshed.net/dev/devcpp.html
2. Save it to your computer
3. Double Click the setup and you will see a following dialog box.
4. Select typical and click next



Dev-C++ 5 beta 9 release (4.9.9.2)

Choose Components

Choose which features of Dev-C++ 5 beta 9 release (4.9.9.2) you want to install.

Choose components

Select the type of install: Typical

Or, select the optional components you wish to install:

- [✓] Dev-C++ program
- [✓] Example files
- [✓] Help files
- [✓] Icon files
- [✓] Mingw compiler sys
- [ ] Language files
- [✓] Associate C and C-
- [✓] Create shortcuts ir

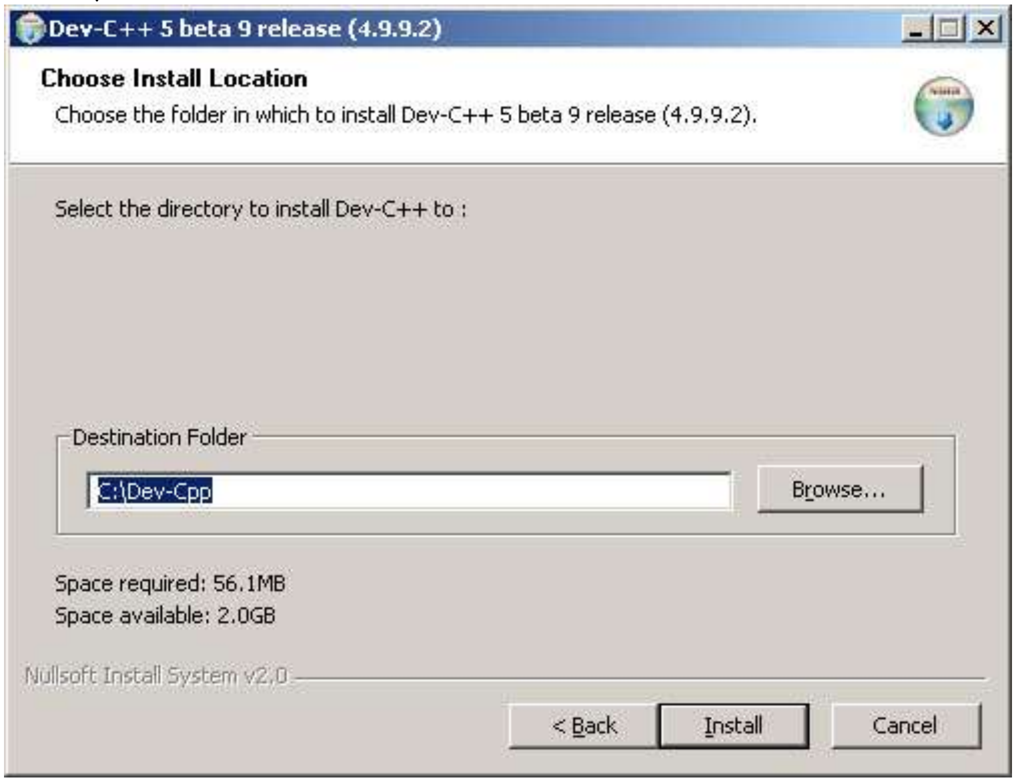
Description

Hover your mouse over a component to see its description.

Space required: 56.1MB

Nullsoft Install System v2.0

5. Select your destination folder and click install



Dev-C++ 5 beta 9 release (4.9.9.2)

Choose Install Location

Choose the folder in which to install Dev-C++ 5 beta 9 release (4.9.9.2).

Select the directory to install Dev-C++ to :

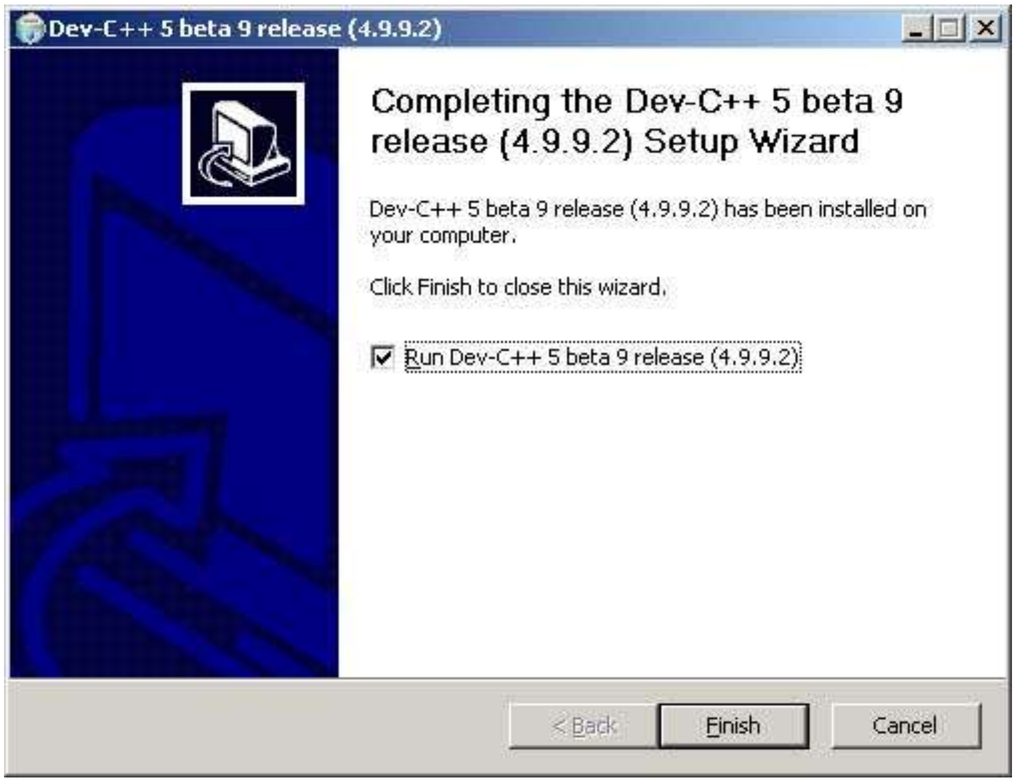
Destination Folder

C:\Dev-Cpp

Space required: 56.1MB
Space available: 2.0GB

Nullsoft Install System v2.0

6. Check Run Dev C++ and click finish.



Dev-C++ 5 beta 9 release (4.9.9.2)

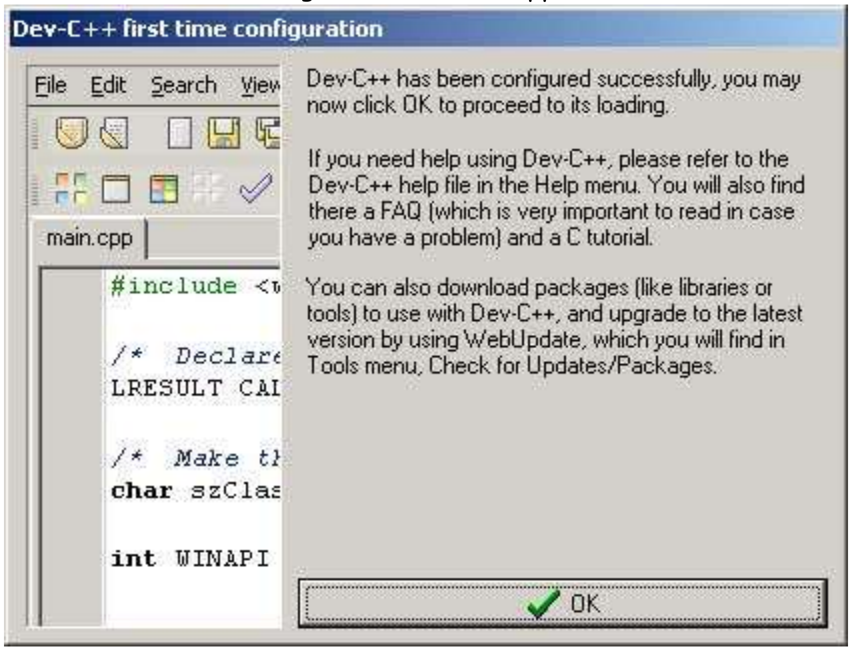
Completing the Dev-C++ 5 beta 9 release (4.9.9.2) Setup Wizard

Dev-C++ 5 beta 9 release (4.9.9.2) has been installed on your computer.

Click Finish to close this wizard.

☑ Run Dev-C++ 5 beta 9 release (4.9.9.2)

7. Dev C++ first time configuration box will appear, click next and then ok.



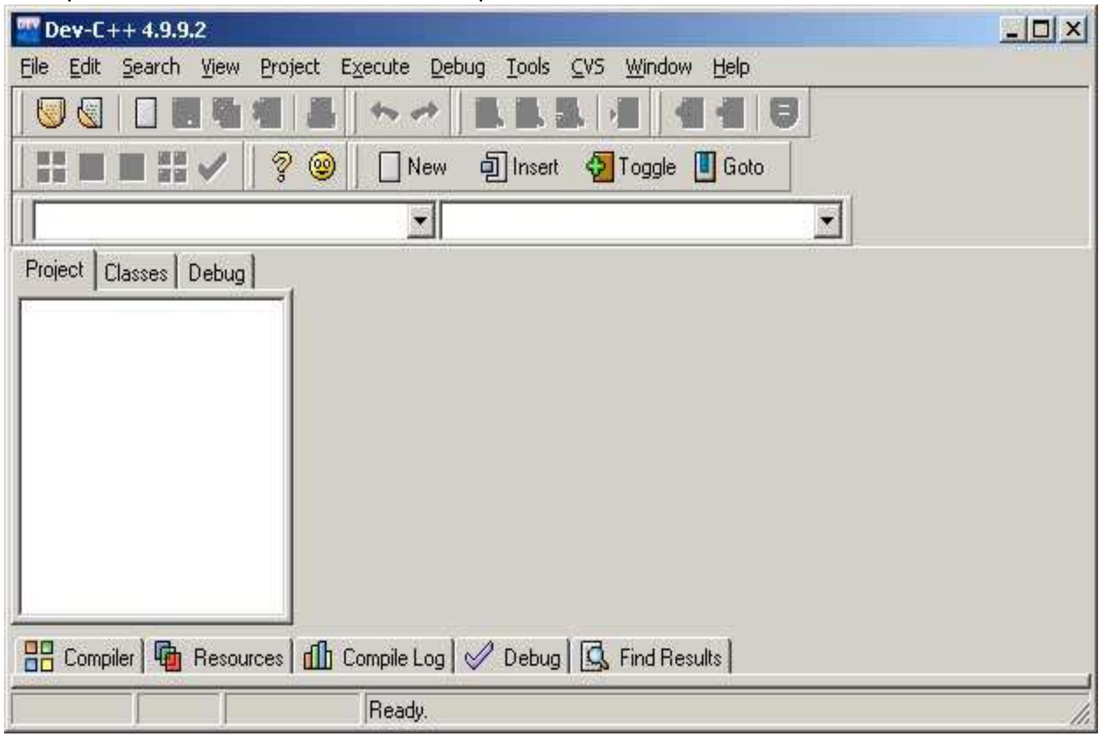
```
#include <w
/*  Declare
LRESULT CAL
/*  Make th
char szClas
int WINAPI
```

Dev-C++ has been configured successfully, you may now click OK to proceed to its loading.

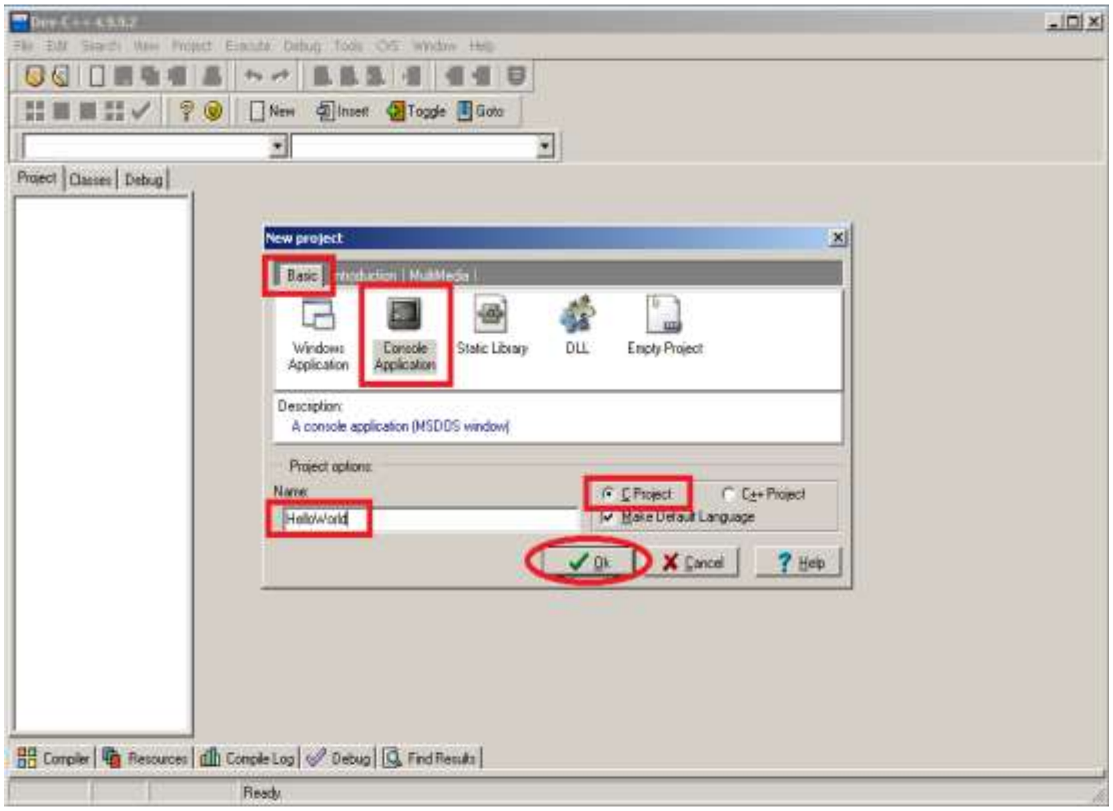
If you need help using Dev-C++, please refer to the Dev-C++ help file in the Help menu. You will also find there a FAQ (which is very important to read in case you have a problem) and a C tutorial.

You can also download packages (like libraries or tools) to use with Dev-C++, and upgrade to the latest version by using WebUpdate, which you will find in Tools menu, Check for Updates/Packages.

8. Now your Dev C++ environment is ready to code.


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Compiler Resources Compile Log Debug Find Results
Ready.

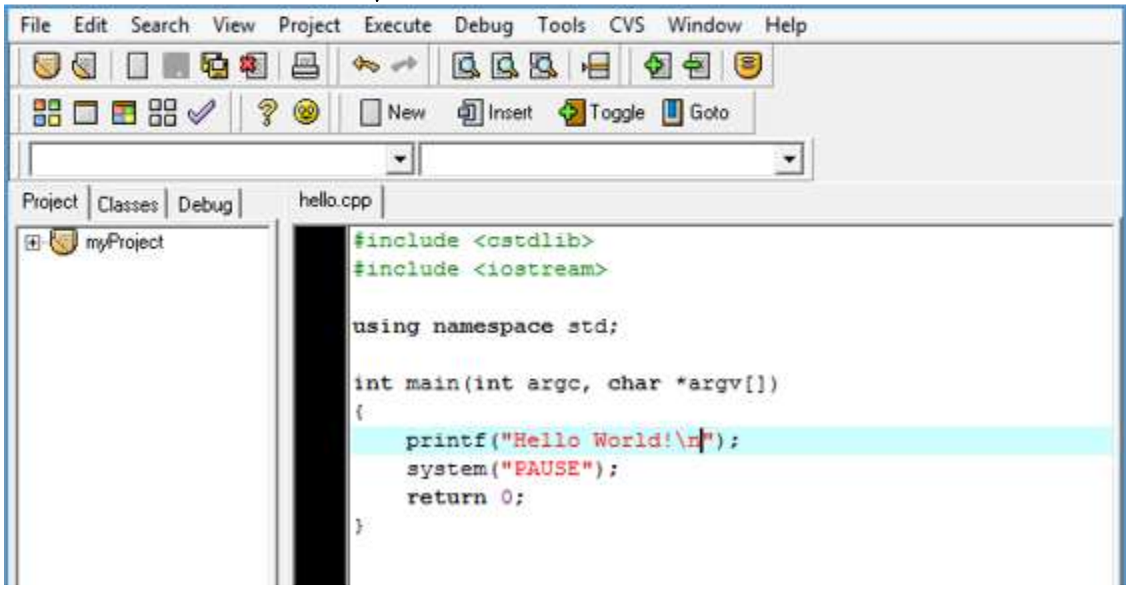
9. Go to File in menu and select new then click project.



Description:
A console application (MSDOS window)

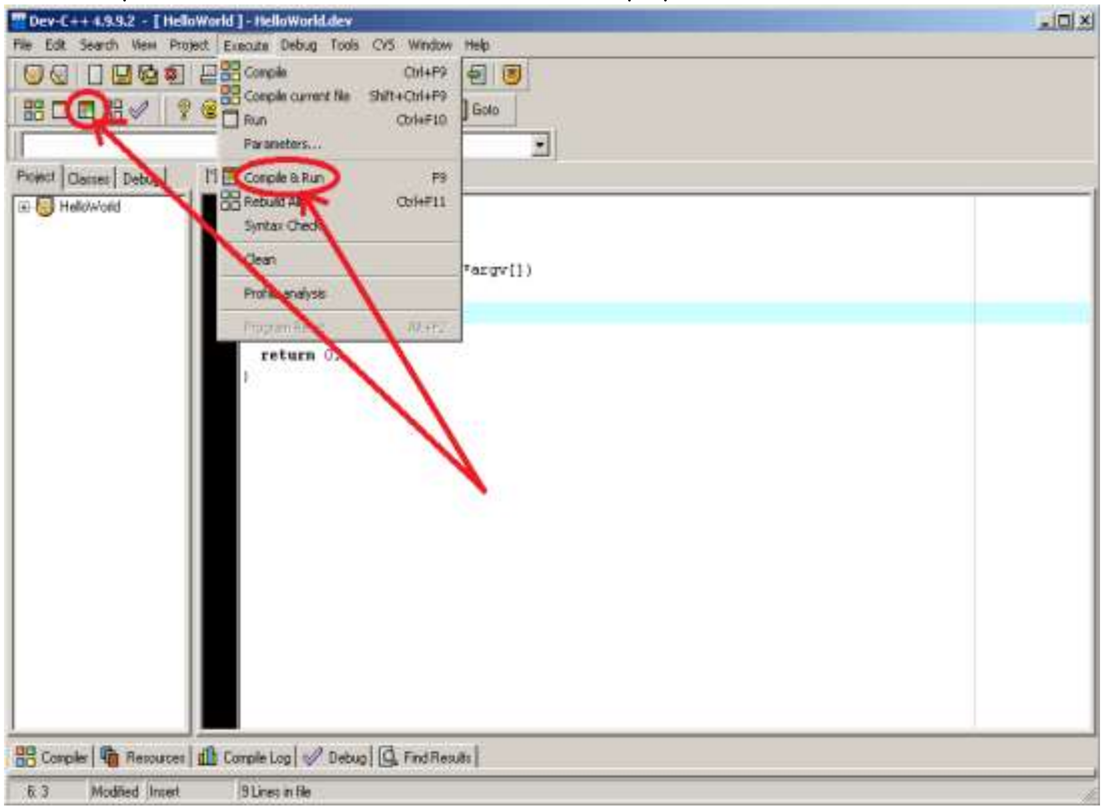
Project options:
Name: HelloWorld
[✓ C Project] C++ Project
✓ Make Default Language

10. Write these lines of codes in your IDE.

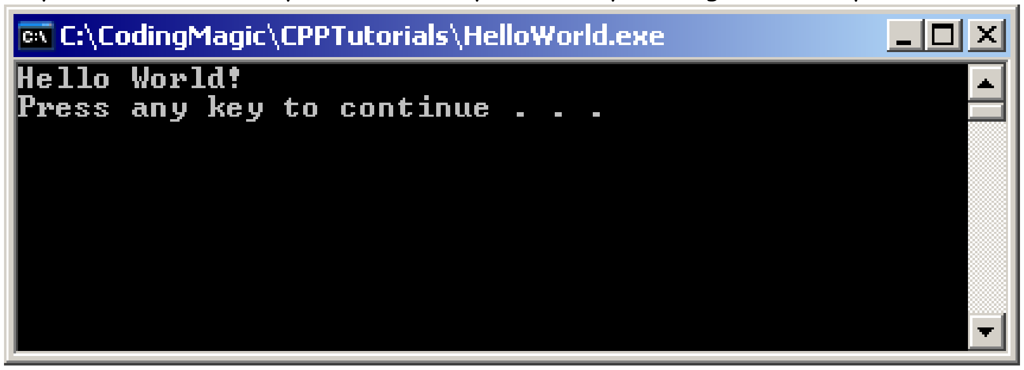


```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    printf("Hello World!\n");
    system("PAUSE");
    return 0;
}
```

11. Click Compile & Run from the menu item or the displayed icon.


Dev-C++ 4.9.9.2 - [HelloWorld] - HelloWorld.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
Compile Ctrl+F9
Compile current file Shift+Ctrl+F9
Run Ctrl+F10
Parameters...
Compile & Run F9
Rebuild All Ctrl+F11
Syntax Check
Clean
Profile analysis
Goto
Project Classes Debug
HelloWorld
Compiler Resources Compile Log Debug Find Results
6: 3 Modified Insert 9 Lines in file

12. If you did not have a syntax error on your code, you will get this output.



```
Hello World!
Press any key to continue . . .
```

1.4 Exercises for lab

Exercises 1: - Type and save the program with the name LAB0Q1.CPP

Exercises 2: - Compile, Run, and Test the program

```
/* Volume of a sphere */
#include <iostream.h>
#include <stdio.h>
#include<math.h>
#define  PI 3.14159
int main(void)
{
double radius, volume;
cout<<"enter the radius > ";
cin>>radius;
volume = 4.0 / 3.0 * PI * pow(radius,3);
cout<<"the radius of the sphere is :"<< radius;
cout<<"the volume of the sphere is :"<< volume;
getche();
return 0;
}
```

1.5 Home Work

1) Launch TurboC++ in your computer and run the above programs.

Lab # 02

Single Line Comment, Multi line Comment, printf and scanf Statements

2.1 Objective:

To familiarize the students with the use of single line comment, multi line comment cout and cin statements.

2.2 Scope:

The student should know the following at the end of this lab:

1. Problem Solving

2. How to use printf & scanf statements

3. Single & Multi line comments

4. Invalid nesting of multi line comments

5. Writing Complete Programs

2.3 Useful Concept

Functions printf() and scanf() are most commonly used to display output and take input respectively.

printf()

- Instructs computer to perform an action
  - Specifically, used to print the "character, string, float, integer, octal and hexadecimal values" onto the output screen.
- Escape character (\)
  - Indicates that printf should do something out of the ordinary
  - **\n** is the new line character
- Conversion Format String (%d,%s…)
  - If this conversion format string matches with remaining argument(variable/constant), value of that variable/constant is displayed.
  - We use printf() function with %d format specifier to display the value of an integer variable.
  - Similarly %c is used to display character, %f for float variable, %s for string variable, %lf for double and %x for hexadecimal variable.

Escape Sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

Scanf()

- Obtains a value from the user
  - **scanf** uses standard input (usually keyboard)
- When executing the program the user responds to the **scanf** statement by typing in a character from keyboard, then pressing the *enter* (return) key
- The format specifier %d is used in scanf() statement. So that, the value entered is received as an integer and %s for string.
- Ampersand is used before variable name in scanf() statement i.e. ch is written as &ch.

Single line comments

- Text after // till next line is ignored by computer
- Used to describe program

Multi line comments

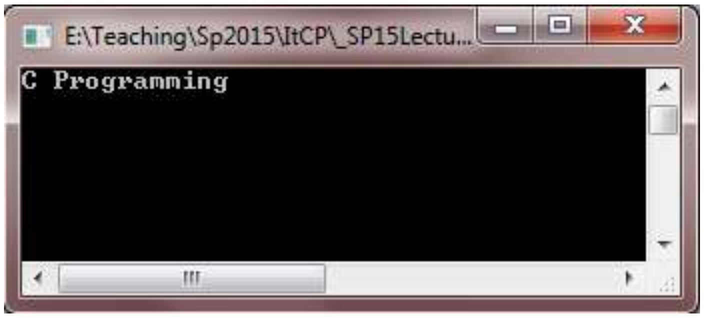
- Text surrounded by /* and */
- Used to describe program

2.4 Examples

Example-1:- This program demonstrates the use of printf() and the basic structure of a C program.

```
#include <stdio.h>      //This is needed to run printf() function.
void main()
{
   printf("C Programming");  //displays the content inside quotation
   getch();
}
```

Here's the program's output:

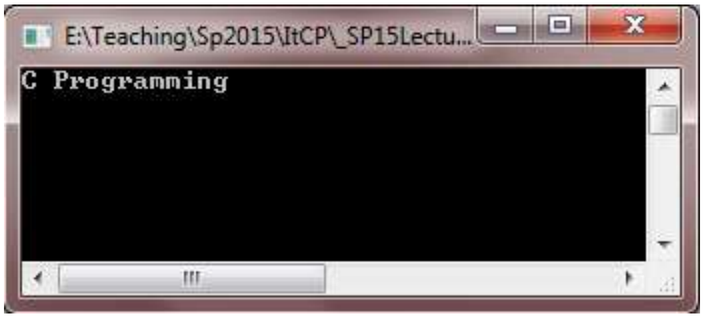


```
C Programming
```

Example -2:- This program illustrates the use of single line (//text) and multi line comments (* text *\).

```
#include <stdio.h>      //This is needed to run printf() function.
void main()
{
   printf("C Programming");  //displays the content inside quotation
/* this is multiline
  comments */
   getch();

}
```

Here's the program's output:



```
C Programming
```

Example-3:- This program illustrates the use of nested multi line comments.

```
# include <conio.h>

void main()
{
   printf("This is my first program in C language");
  /* this is not /* allowed nested comments */ */
  getch();

}
```

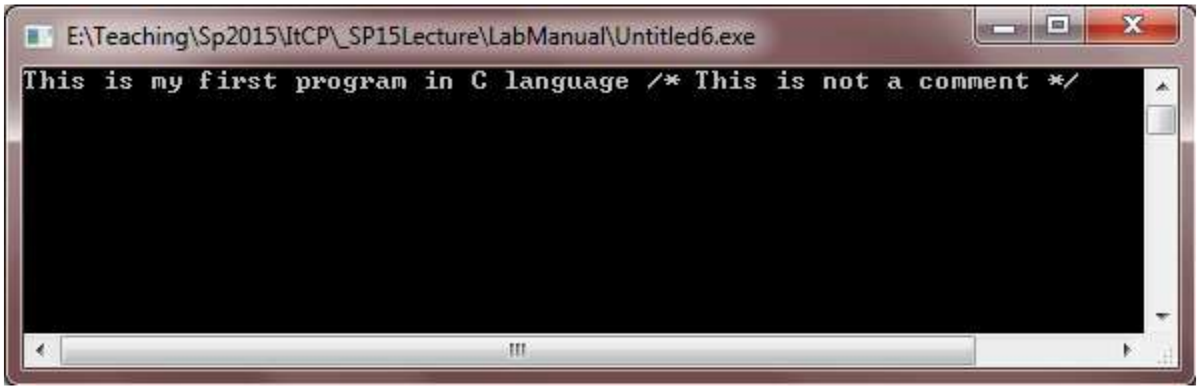
Here’s the program’s output:

Syntax error because of nested multi line comments

Example-4:- This program illustrates the use of nested multi line comments in printf statement.

```
# include <conio.h>

void main()
{
  printf( "This is my first program in C language /* This is not a comment */") ;
 getch();
}
```

Here’s the program’s output:

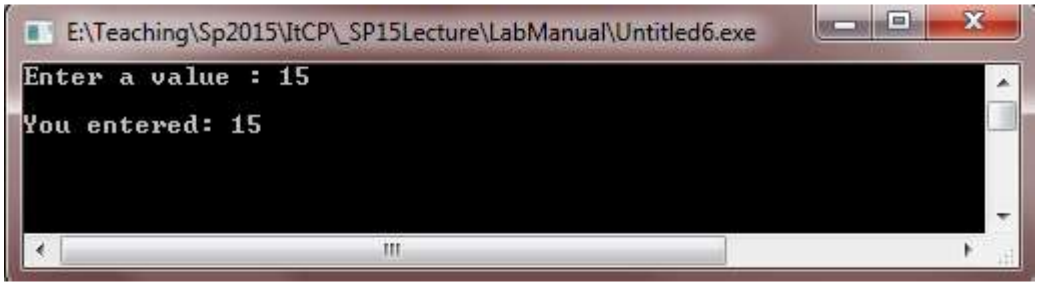


```
E:\Teaching\Sp2015\ItCP\_SP15Lecture\LabManual\Untitled6.exe
This is my first program in C language /* This is not a comment */
```

Example-5:- This program illustrates the use of scanf() for integer value.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i;
 printf("Enter a value");
 scanf("%d",&i);
 printf( "\nYou entered: %d",i);
 getch();
}
```

Here’s the program’s output:

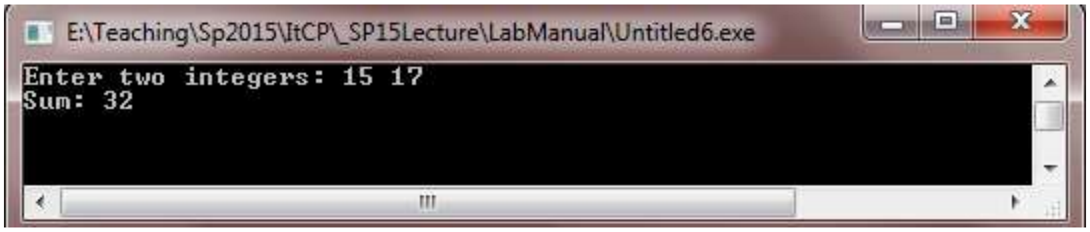

E:\Teaching\Sp2015\ItCP_SP15Lecture\LabManual\Untitled6.exe

```
Enter a value : 15

You entered: 15
```

Example-6:- This program illustrates integer addition. Also note the syntax necessary for the use of scanf().

```
#include <stdio.h>
void main( )
{
   int num1, num2, sum;
   printf("Enter two integers: ");
   scanf("%d %d",&num1,&num2); /* Stores the two integer entered by user in variable num1 and
num2 */

   sum=num1+num2;      /* Performs addition and stores it in variable sum */
   printf("Sum: %d",sum);  /* Displays sum */
   getch();

}
```

Here's the program's output:

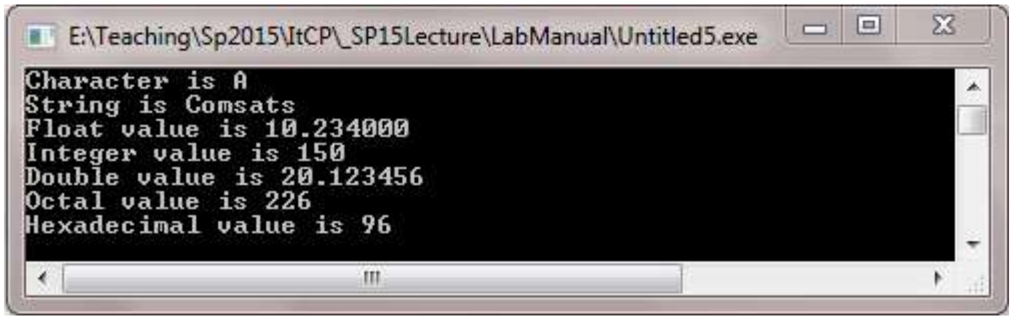

E:\Teaching\Sp2015\ItCP_SP15Lecture\LabManual\Untitled6.exe

```
Enter two integers: 15 17
Sum: 32
```

Example-7:- This program illustrates different format conversion in printf statement.

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

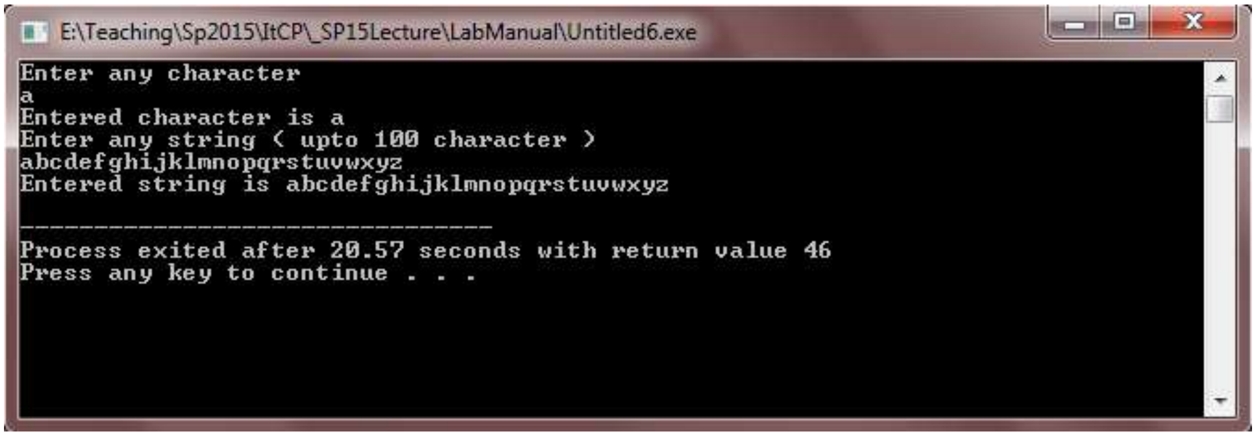
```
void main()
{
char ch = 'A';
char str[20] = "Comsats";
float flt = 10.234;
int no = 150;
double dbl = 20.123456;
printf("Character is %c \n", ch);
printf("String is %s \n" , str);
printf("Float value is %f \n", flt);
printf("Integer value is %d\n" , no);
printf("Double value is %lf \n", dbl);
printf("Octal value is %o \n", no);
printf("Hexadecimal value is %x \n", no);
getch();
}
```


E:\Teaching\Sp2015\ItCP_SP15Lecture\LabManual\Untitled5.exe

```
Character is A
String is Comsats
Float value is 10.234000
Integer value is 150
Double value is 20.123456
Octal value is 226
Hexadecimal value is 96
```

Example-8:- This program illustrates different format conversion in scanf statement.

```
#include <stdio.h>
int main()
{
char ch;
char str[100];
printf("Enter any character \n");
scanf("%c", &ch);
printf("Entered character is %c \n", ch);
printf("Enter any string ( upto 100 character ) \n");
scanf("%s", &str);
printf("Entered string is %s \n", str);
}
```


E:\Teaching\Sp2015\ItCP_SP15Lecture\LabManual\Untitled6.exe

```
Enter any character
a
Entered character is a
Enter any string ( upto 100 character )
abcdefghijklmnopqrstuvwxyz
Entered string is abcdefghijklmnopqrstuvwxyz

--------------------------------
Process exited after 20.57 seconds with return value 46
Press any key to continue . . .
```

<u>Exercises for lab</u>

Exercise 1:- Write a program to display:

Your name:

Your ID:

Your Course name:

Exercise 2:- Write a program that prints the following diagram using printf statement.

```
* * * * * * * * *
*               *
*               *
* * * * * * * * *
```

2.5 Home Work:

1) Write a program that take six subjects marks from user and display the sum of all marks.

Lab # 03

Data Types

3.1 Objective:

Learn the Problem Solving and Basics C Language

3.2 Scope:

The student should know the following:

- Problem Solving
- Different data types of C and their Use.
- Declaring Variables
- Standard Input and Output
- Writing Complete Programs

3.3 Useful Concept:

An useful List of basic data type of C , number of bytes used to store these data types in memory of computer system:

Data Type	***Bytes***
Signed char	1
unsigned char	1
sort signed int	2
short unsigned int	2
long signed int	4
long unsigned int	4
float	4
double	8
long double	10

Variables

- C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable.
- The value of the C variable may get change in the program.
- C variable might be belonging to any of the data type like int, float, char etc.

Rules for naming C variable:

1. Variable name must begin with letter or underscore.
2. Variables are case sensitive
3. They can be constructed with digits, letters.
4. No special symbols are allowed other than underscore.
5. sum, height, _value are some examples for variable name

Declaring & initializing C variable:

- Variables should be declared in the C program before to use.
- Memory space is not allocated for a variable while declaration. It happens only on variable definition.
- Variable initialization means assigning a value to the variable.

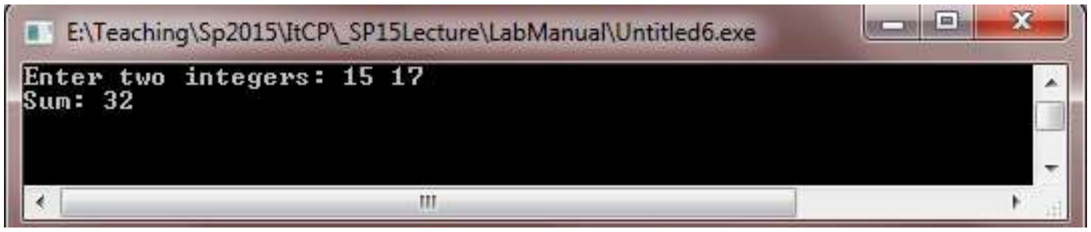
S.No	Type	Syntax	Example
1	Variable declaration	data_type variable_name;	int x, y, z; char flat, ch;
2	Variable initialization	data_type variable_name = value;	int x = 50, y = 30; char flag = 'x', ch='l';

3.4 Examples:

Example-1:- This program illustrates integer addition. Also note the syntax necessary for the use of scanf().

```
#include <stdio.h>
void main( )
{
   int num1, num2, sum;
   printf("Enter two integers: ");
   scanf("%d %d",&num1,&num2); /* Stores the two integer entered by user
in variable num1 and num2 */

   sum=num1+num2;      /* Performs addition and stores it in variable sum */
   printf("Sum: %d",sum);  /* Displays sum */
   getch();

}
```

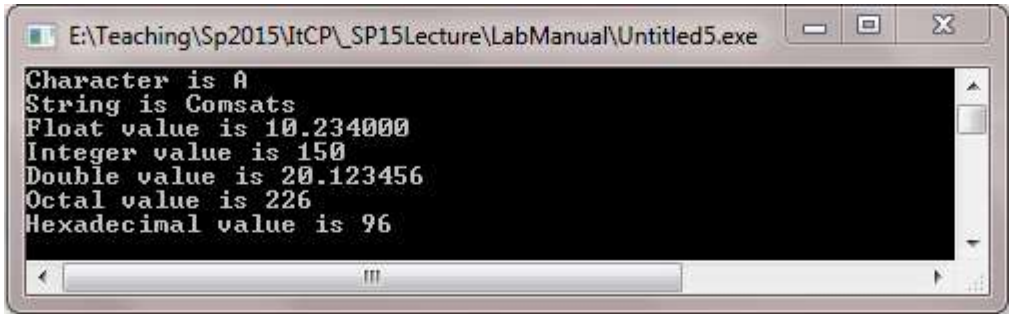
Here's the program's output:


E:\Teaching\Sp2015\ItCP_SP15Lecture\LabManual\Untitled6.exe

```
Enter two integers: 15 17
Sum: 32
```

Example-2:- This program illustrates different format conversion in printf statement.

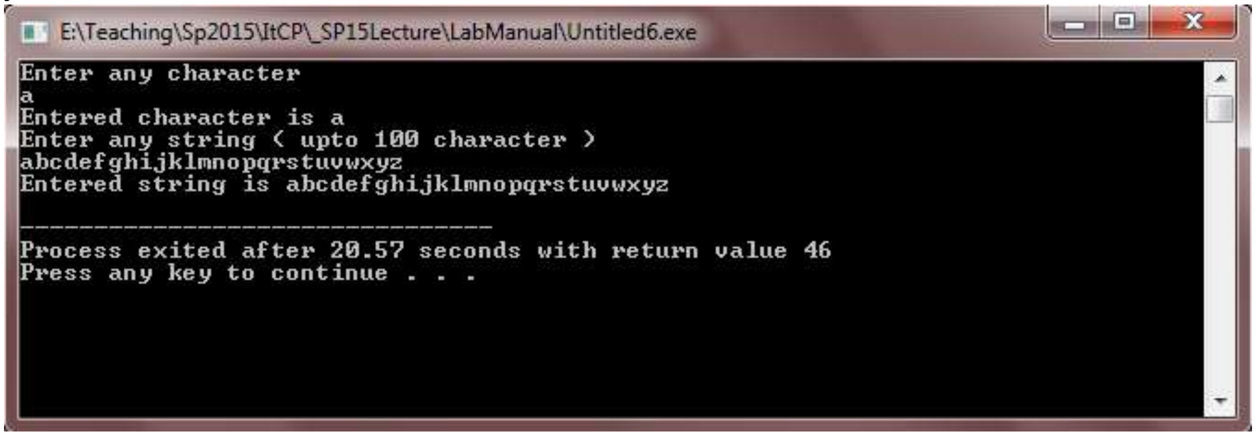
```
#include <stdio.h>
void main()
{
char ch = 'A';
char str[20] = "Comsats";
float flt = 10.234;
int no = 150;
double dbl = 20.123456;
printf("Character is %c \n", ch);
printf("String is %s \n" , str);
printf("Float value is %f \n", flt);
printf("Integer value is %d\n" , no);
printf("Double value is %lf \n", dbl);
printf("Octal value is %o \n", no);
printf("Hexadecimal value is %x \n", no);
getch();
}
```


E:\Teaching\Sp2015\ItCP_SP15Lecture\LabManual\Untitled5.exe

```
Character is A
String is Comsats
Float value is 10.234000
Integer value is 150
Double value is 20.123456
Octal value is 226
Hexadecimal value is 96
```

Example-3:- This program illustrates different format conversion in scanf statement.

```
#include <stdio.h>
int main()
{
char ch;
char str[100];
printf("Enter any character \n");
scanf("%c", &ch);
printf("Entered character is %c \n", ch);
printf("Enter any string ( upto 100 character ) \n");
scanf("%s", &str);
printf("Entered string is %s \n", str);
}
```



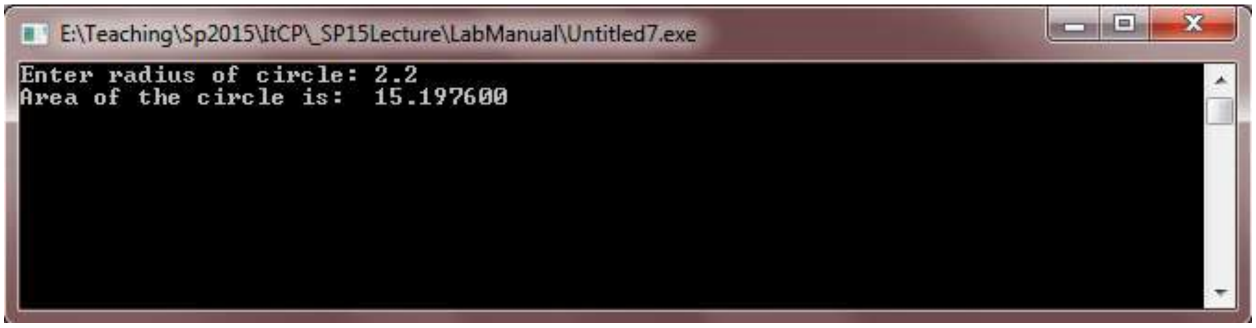
E:\Teaching\Sp2015\ItCP_SP15Lecture\LabManual\Untitled6.exe

```
Enter any character
a
Entered character is a
Enter any string ( upto 100 character )
abcdefghijklmnopqrstuvwxyz
Entered string is abcdefghijklmnopqrstuvwxyz

--------------------------------
Process exited after 20.57 seconds with return value 46
Press any key to continue . . .
```

Example - 4:- This program calculates the area of the circle. The area of the circle is $\prod r^2$. $\prod$ value is constant that is 3.14 but radius can change so this program gets the value of radius variable form user and calculate the area on that value.

```
# include <conio.h>
void main()
{
  float radius,area;
  printf( "Enter radius of circle: ");
```

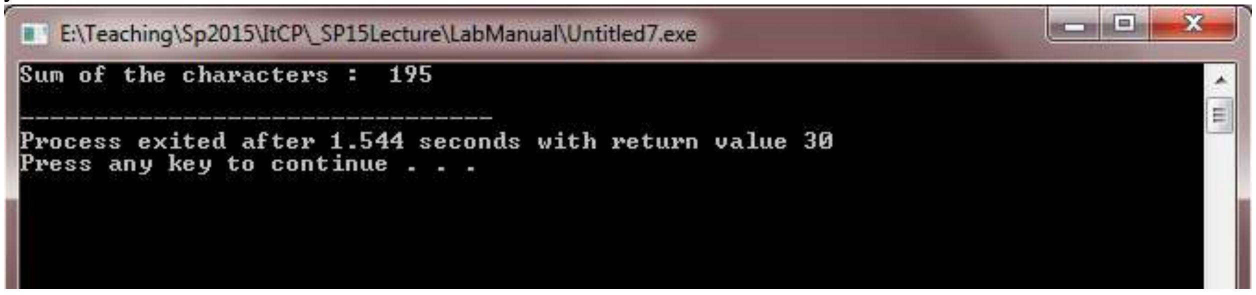
```
    scanf( "%f", &radius) ;
    area = 3.14*radius*radius;
    printf( "Area of the circle is:  %f \n", area);
    getch();
  }
```


E:\Teaching\Sp2015\ItCP_SP15Lecture\LabManual\Untitled7.exe

```
Enter radius of circle: 2.2
Area of the circle is:  15.197600
```

Example - 5:- This program illustrates the addition on charter values.

```
# include <conio.h>
void main()
{
char x, y ;
int z ;
x = 'a' ;
y = 'b' ;
z = x + y ;        //Add the assci value of 'a' with assci value of 'b' and store in z.
printf("Sum of the characters :  %d \n", z);
}
```

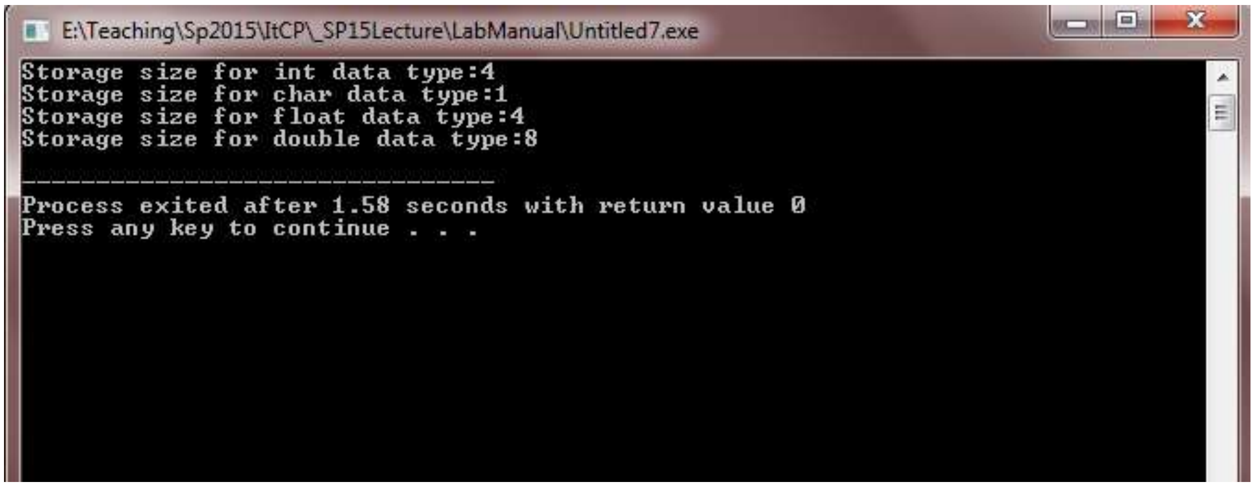

E:\Teaching\Sp2015\ItCP_SP15Lecture\LabManual\Untitled7.exe

```
Sum of the characters :  195

--------------------------------
Process exited after 1.544 seconds with return value 30
Press any key to continue . . .
```

Example - 6:- This program illustrates the use of sizeof() function which is used to find the memory space allocated for each C data types.

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

```
#include <limits.h>
int main()
{
int a;
char b;
float c;
double d;
printf("Storage size for int data type:%d \n",sizeof(a));
printf("Storage size for char data type:%d \n",sizeof(b));
printf("Storage size for float data type:%d \n",sizeof(c));
printf("Storage size for double data type:%d\n",sizeof(d));
return 0;
}
```


E:\Teaching\Sp2015\ItCP_SP15Lecture\LabManual\Untitled7.exe

```
Storage size for int data type:4
Storage size for char data type:1
Storage size for float data type:4
Storage size for double data type:8

--------------------------------
Process exited after 1.58 seconds with return value 0
Press any key to continue . . .
```

3.5 Exercises for lab

Exercise 1:- Write a program to compute circumference of a circle.

Exercise 2:-Write a program that takes any ASCII value from user and display next five char after that ASCII value.

Hints: - if user enters 95, your program should display the char against the ASSCII value 96,97,98,99 and 100.

3.6 Home Work

1. Write a program converts a temperature from Celsius to Fahrenheit. Use the following formula: F = 1.8 x C + 32 .

2. Write a program that reads three integers representing hours, minutes, and seconds of a time. Then it calculates the equivalent time in seconds.

Lab# 04

Arithmetic Operators, Arithmetic Expressions and math functions

4.1 Objective

Learn using operators, expressions and math functions.

4.2 Scope

The student should know the following at the end of this lab:

1. Arithmetic Operators
2. Arithmetic Expressions
3. Math Functions
4. Writing Complete Programs

4.3 Useful Concepts

Arithmetic Operators:

To solve most programming problems, you will need to write arithmetic expressions that manipulate type int and double data. In C language, these are the basic arithmetic operators: addition (+), subtraction (-), multiplication (*), division (/), and remainder (%)

If the operands are of different types, the one with weaker type will be raised to the level of the other type, then the operation will be performed. The order is char, int, long, float, double.

The division between two integer numbers results in an integer result (as you can see 5/2 gives 2 not 2.5).

The division by zero is undefined. Either you get a compilation warning or a run time error.

The remainder operator % is related to the division one. This operator will give the remainder produced by dividing the two numbers (5 % 2 = 1). Therefore, this operator is not used with double values. If one of the operand or both are double, you will get an error message from the compiler saying:

Illegal use of floating point

Arithmetic Expressions:

Arithmetic expressions contain a combination of the arithmetic operations including brackets:

x = z – (a + b / 2) + w * -y

To evaluate the expression, we need to follow the precedence rule which is as follows:

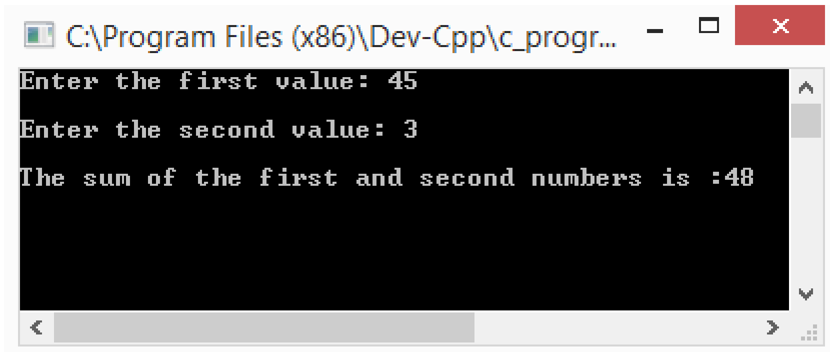
1. ( )	expression within parentheses
2. +, -	unary operators (plus and minus sign)
3. *, /, %	multiplication, division and remainder are evaluated from left to right
4. +, -	addition and subtraction are evaluated from left to right

4.4 Examples:

Example1: Program illustrates how to input two variables from the user and display their sum. Both variables are integers.

```
#include<stdio.h>
#include<conio.h>
int main ()
{
        int val1,val2, result=0;
        printf("Enter the first value: ");
        scanf("%d", &val1);
        printf("\nEnter the second value: ");
        scanf("%d", &val2);
        result=val1+val2;
        printf("\nThe sum of the first and second numbers is :%d", result);
        getch();
}
```

Here is the output of the above result

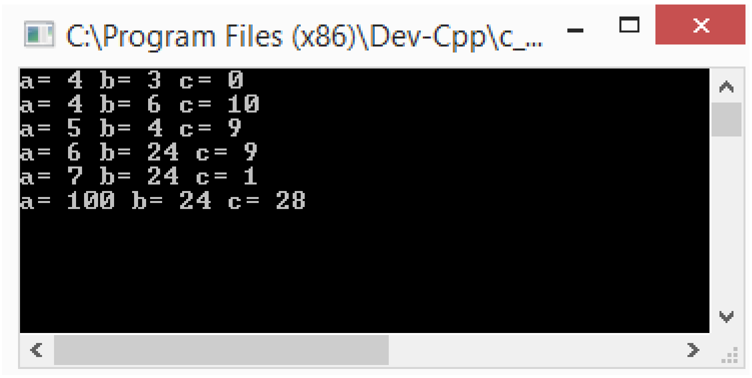


```
Enter the first value: 45

Enter the second value: 3

The sum of the first and second numbers is :48
```

Example2: This program illustrates some arithmetic properties of integer (int) variables. Trace through the program to make sure you understand the values of each variable in all stages of the program.

```
#include<stdio.h>
#include<conio.h>
int main ()
{
        int a=4, b, c;
        printf("a= %d b= %d c= %d",a,b,c);
        b=6;
        c=a+b;
        printf("\na= %d b= %d c= %d",a,b,c);
        a++;                                    //this statement is similar to a=a+1;
        b -= 2;                                 //this statement is similar to b= b-2;
        --c;                                    //this statement is similar to c=c-1;
        printf("\na= %d b= %d c= %d",a,b,c);
        b *= ++a;                               //a=a+1;, b=b*a;
        printf("\na= %d b= %d c= %d",a,b,c);
        c /= a++;                               //c=c/a;, a=a+1;
        printf("\na= %d b= %d c= %d",a,b,c);
        a = (b+c)*4;                            //First (b+c) then multiply the result with 4.
        c = b+c*4;                              //First c*4 then result is added with b.
         printf("\na= %d b= %d c= %d",a,b,c);
        getch();
}
```

The output for the above program is

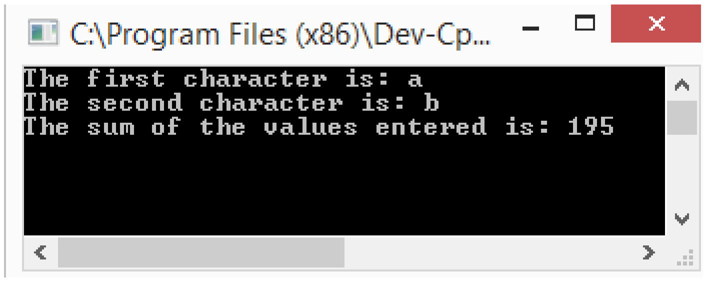

C:\Program Files (x86)\Dev-Cpp\c_...

```
a= 4 b= 3 c= 0
a= 4 b= 6 c= 10
a= 5 b= 4 c= 9
a= 6 b= 24 c= 9
a= 7 b= 24 c= 1
a= 100 b= 24 c= 28
```

Example3: Program illustrates how to initialize two characters and displays the effect of addition.

```
#include<conio.h>
#include<stdio.h>
int main ()
{
        char val1,val2;
        int sum=0;
        val1='a';
        val2='b';
        printf("The first character is: %c",val1);
        printf("\nThe second character is: %c",val2);
        sum=val1+val2;
        printf("\nThe sum of the values entered is: %d",sum);
        getch();
}
```

The output for the above given program is

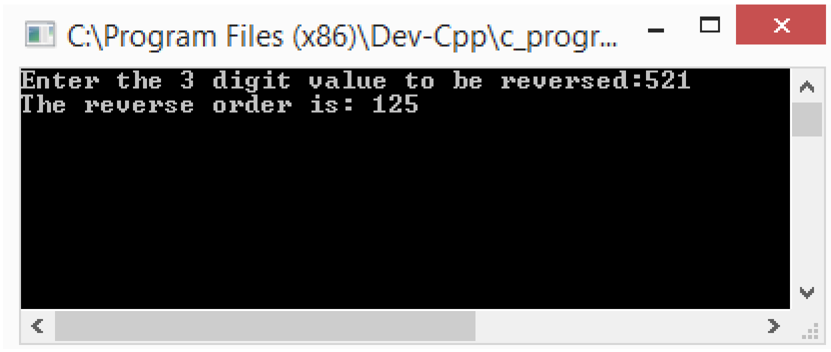

C:\Program Files (x86)\Dev-Cp...

```
The first character is: a
The second character is: b
The sum of the values entered is: 195
```

Example 4: Input a 3 digit value from the user (forexample 521) and display it in reverse order (i.e. 125).

```
#include<conio.h>
#include<stdio.h>
int main ()
{
        int n,a,b;
        printf("Enter the 3 digit value to be reversed:");
        scanf("%d", &n);
        a=n/100;
        n=n%100;
        b=n/10;
        n=n%10;
        printf("The reverse order is: %d%d%d",n,b,a);
        getch();
}
```

The output of this program is shown below



```
Enter the 3 digit value to be reversed:521
The reverse order is: 125
```

Math function:

To do some advanced mathematical functions in C, there is a header file called <math.h> that has different trigonometric and algebraic functions. Here are some frequently used functions:

pow(x,y)	x^y	pow(5,3) = 5^3 = 125
sqrt(x)	$\sqrt{x}$ (x > 0)	sqrt(4) = $\sqrt{4}$ = 2

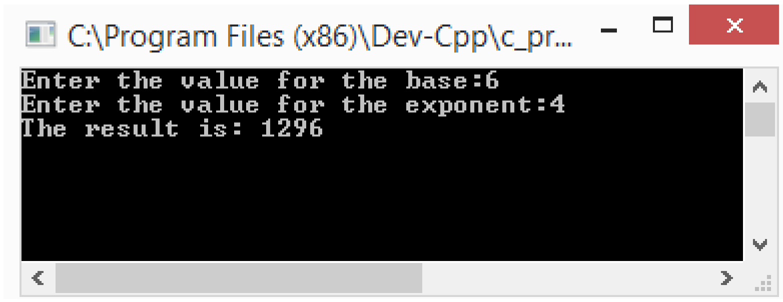
log(x)	ln x (x > 0)	log(5) = 1.6094
log10(x)	$\log_{10} x$ (x > 0)	log10(5) = 0.698970
exp(x)	e^x	exp(2) = 7.3891
sin(x)	sin x (x in radian)	sin(90) = 0.893997
cos(x)	cos x (x in radian)	cos(90) = -0.448074
tan(x)	tan x (x in radian)	tan(90) = -1.9952
asin(x)	$\sin^{-1}$ x (x in [-1,1])	asin (0) = 0
acos(x)	$\cos^{-1}$ x (x in [-1,1])	acos(0) = 1.570796
atan(x)	$\tan^{-1}$ x	atan(0) = 0

Note that you have to include the header file math.h before you use any of these functions. Also, the return type of these functions is double.

Example1: Input the values for base and exponent and calculate its power using the pow(x,y) built in function.

```
#include<conio.h>
#include<stdio.h>
#include<math.h>
int main ()
{
        int base,expo,result=0;
        printf("enter the value for the base:");
        scanf("%d",&base);
        printf("enter the value for the exponent:");
        scanf("%d",&expo);
        result= pow(base,expo);
        printf("the result is: %d", result);
        getch();
}
```

The output for the given program is



```
Enter the value for the base:6
Enter the value for the exponent:4
The result is: 1296
```

4.5 Exercise for Lab

Exercise 1: Write a program that finds the area of a triangle given the length of its sides: a, b, c.

$$area = \sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)}$$

$$s = \frac{a+b+c}{2}$$

Exercise 2: Write a program taking two values as inputs from the user and display the results for all the basic arithmetic operations performed on them

Addition

Subtraction

Multiplication

Division

Modulus

Exercise 3: Write a program that inputs a 4 digit value from the user (for example 6382) and displays a result with an increment of 1 in each digit (i.e. 7493)

4.6 Home Task

1. Write a program that takes any ASCII value from user and display next five char after that ASCII value.
2. Write a program that reads a four digit number from user, then the program separates digits of the number e.g. 4567 to be displayed as:

   4

   5

   6

   7

Lab# 05

Decision Control Structure

5.1 Objective

The student should practice the following statements:

1. if statemets
2. if-else statements
3. Nested if/if-else
5. Switch statement¨

5.2 Scope

By the end of this lab a student should know:

1. Syntax of if and nested if statements.
2. Syntax of switch and nested switch statements.

5.3 Useful concept

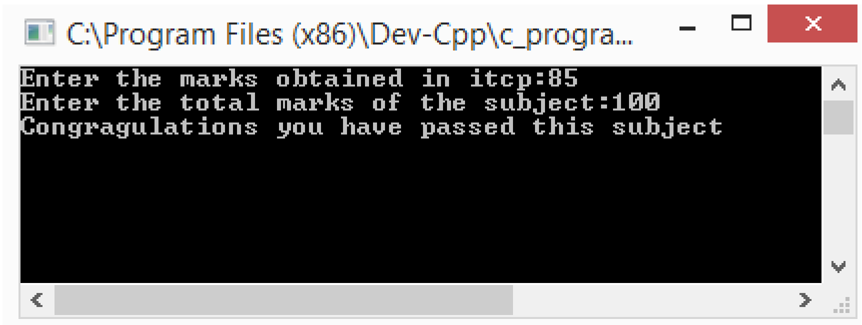
If Statement:

Sometimes in your program you need to do something if a given a condition is satisfied. Suppose for example, you want to display a congratulatory message in case a student has passed a course, the marks must be greater than 50%.

5.4 Examples

```
#include<conio.h>
#include<stdio.h>
int main ()
{
        int marks,total,percent=0;
        printf("Enter the marks obtained in itcp:");
        scanf("%d", &marks);
        printf("Enter the total marks of the subject:");
        scanf("%d",&total);
        percent= (marks*100)/total;
        if (percent>=50)
        printf("Congragulations you have passed this subject");
        getch();
}
```

Output for the above program is



```
Enter the marks obtained in itcp:85
Enter the total marks of the subject:100
Congragulations you have passed this subject
```

If you want to execute more than one statement based on that condition, then your statements should be enclosed with brackets. For example

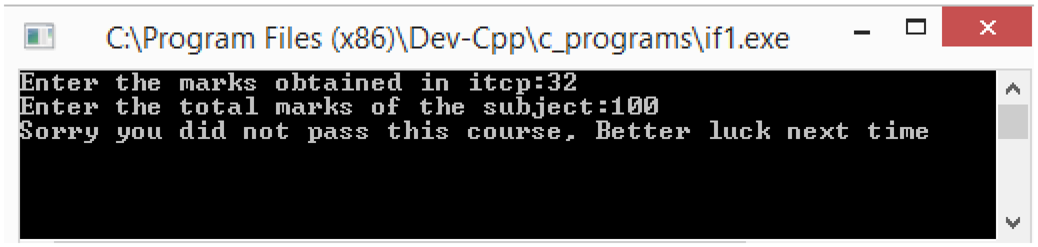
```
#include<conio.h>

#include<stdio.h>

int main ()

{

        int marks,total,percent=0;

        printf("Enter the marks obtained in itcp:");

        scanf("%d", &marks);

        printf("Enter the total marks of the subject:");

        scanf("%d",&total);

        percent= (marks*100)/total;

        if (percent>=50)

        {

        printf("Congragulations you have passed this subject");

        printf("Well done");

        }

         getch();

}
```

If-else statements

In case you want to have an alternative action to be taken if the condition is not satisfied, then use an if-else statement. For example:

```
#include<conio.h>

#include<stdio.h>

int main ()

{

        int marks,total,percent=0;

        printf("Enter the marks obtained in itcp:");

        scanf("%d", &marks);

        printf("Enter the total marks of the subject:");

        scanf("%d",&total);
        percent= (marks*100)/total;
        if (percent>=50)
        {
        printf("Congratulations you have passed this subject");
        }
        else
        printf(" Sorry you did not pass this course, Better luck next time");
         getch();
}
```

The output for the above program in case the condition is not satisfied (else part is executed) is shown below



```
Enter the marks obtained in itcp:32
Enter the total marks of the subject:100
Sorry you did not pass this course, Better luck next time
```

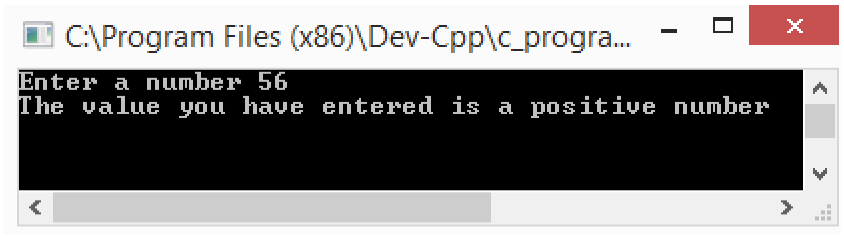
Here, if the condition is true, the if-part will be executed and the else-part is ignored. But if the condition is false, the if-part is ignored and the else-part is executed.

Be careful with compound statements, forgetting the brackets will produce an error because only the first statement is executed if the condition is true, the rest are considered to be unrelated statements and the compiler will complain about the keyword else.

Example: Input a number from user and display whether it is a positive or negative number

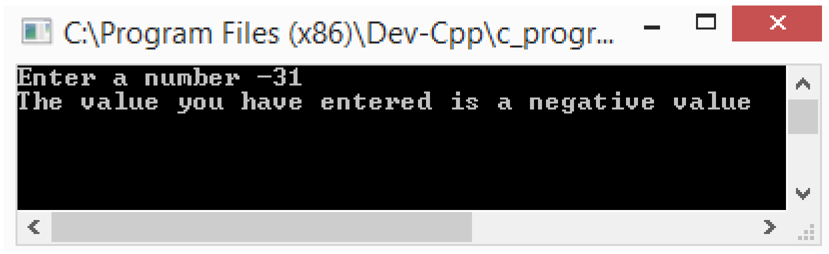
```
#include<conio.h>
#include<stdio.h>
int main ()
{
        int value;
        printf("Enter a number ");
        scanf("%d", &value);
        if (value>0)
        printf("The value you have entered is a positive number");
        else
        printf("The value you have entered is a negative value");
        getch();
}
```

Output for the program with if condition being true



```
Enter a number 56
The value you have entered is a positive number
```

Output for the same program with if condition false



```
Enter a number -31
The value you have entered is a negative value
```

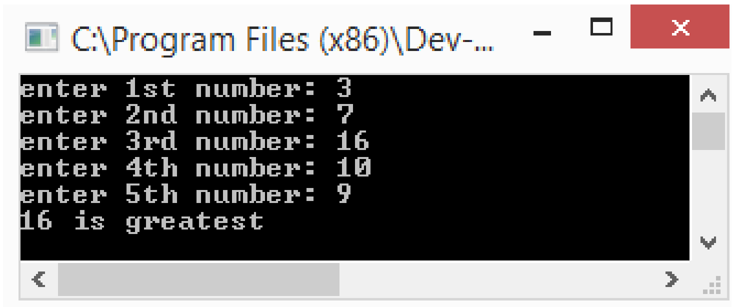
Multiple if-else-if statements

If-else-if statement can be used to choose one block of statements from many blocks of statements. It is used when there are many options and only one block of statements should be executed on the basis of a condition. Consider the following example

Example: Input 5 values from the user and display the maximum number from the list.

```
#include<stdio.h>

#include<conio.h>

int main ()

{

        int a,b,c,d,e;

        printf("enter 1st number: ");

        scanf("%d",&a);

        printf("enter 2nd number: ");

        scanf("%d",&b);

        printf("enter 3rd number: ");

        scanf("%d",&c);

        printf("enter 4th number: ");
        scanf("%d",&d);
        printf("enter 5th number: ");
        scanf("%d",&e);
        if(a>b&&a>c&&a>d&&a>e)
        {
        printf("%d is greatest",a);
        }
        else
        if(b>a&&b>c&&b>d&&b>e)
        {
        printf("%d is greatest",b);
        }
        else
        if(c>a&&c>b&&c>d&&c>e)
        {
        printf("%d is greatest",c);
        }
        else
        if(d>a&&d>b&&d>c&&d>e)
        {
        printf("%d is greatest",d);
        }
        else
        if(e>a&&e>b&&e>c&&e>d)
        {
        printf("%d is greatest",e);
         }
        getch();
}
```

The output for this program is

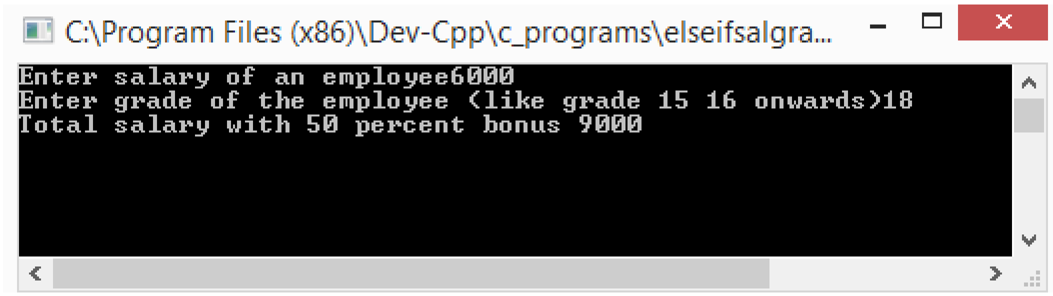


```
enter 1st number: 3
enter 2nd number: 7
enter 3rd number: 16
enter 4th number: 10
enter 5th number: 9
16 is greatest
```

Example: Prompt the user to enter the salary and grade of an employee. If the employee has a grade greater than 15 then add 50% bonus to the employee's salary. Otherwise if the employee's grade is less than 15 then add 25% bonus to the employee's salary.

```
#include<stdio.h>

#include<conio.h>

int main()

{

        int grad,sal,bonus;

        printf("enter salary");

        scanf("%d",&sal);

        printf("enter grade");

        scanf("%d",&grad);

        if (grad>15)

        {

        bonus=sal*(50.0/100.0);

        printf("total salary with 50 percent bonus %d",bonus+sal);

        }

        else if (grad<=15)

        {

        bonus= sal* (25.0/100.0);

        printf("total salary with 25 percent bonus %d",bonus+sal);

         }

        getch();

}
```

The output of this program is



```
Enter salary of an employee6000
Enter grade of the employee (like grade 15 16 onwards)18
Total salary with 50 percent bonus 9000
```

Here each condition is tested one by one starting from the first one. If one of them is true, then the statement associated with that condition is executed and the rest are ignored.

Nested if structure:

An if statement within an if statement is called nested if statement. In nested structure, the control enters into the inner if only when the outer condition is true. Only one block of statements are executed and the remaining blocks are skipped automatically.

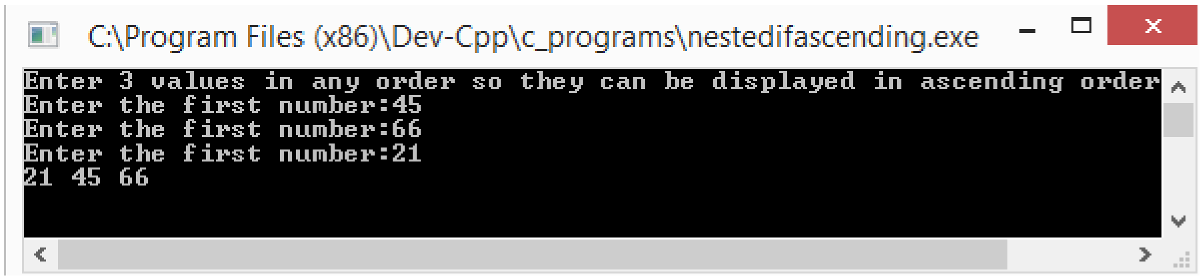
The user can use as many if statements inside another is statements as required. The increase in the level of nesting increases the complexity of nested if statement.

Example: Prompt the user to input three values so as to display them in ascending order.

```
#include<stdio.h>
#include<conio.h>
int main ()
{
        int a,b,c;
        printf("Enter 3 values in any order so they can be displayed in ascending order:");
        printf("\nEnter the first number:");
        scanf("%d",&a);
        printf("Enter the first number:");
        scanf("%d",&b);
        printf("Enter the first number:");
        scanf("%d",&c);
        if(a<b && a<c)
        {
                if (b<c)
                printf("%d %d %d", a,b,c);
                else
```

```
                        printf("%d %d %d", a,c,b);
            }
            if(b<a && b<c)
             {
                        if (a<c)
                printf("%d %d %d", b,a,c);
                         else
                        printf("%d %d %d", b,c,a);
             }
             if (c<a && c<b)
              {
                         if (b<a)
                         printf("%d %d %d", c,b,a);
                         else
                printf("%d %d %d", c,a,b);
              }

           getch();
}
```

Output for this program is shown below



C:\Program Files (x86)\Dev-Cpp\c_programs\nestedifascending.exe

```
Enter 3 values in any order so they can be displayed in ascending order
Enter the first number:45
Enter the first number:66
Enter the first number:21
21 45 66
```

Switch statement:

Another useful statement in C is the switch statement. This statement is somehow similar to if statement in giving you multiple options and do actions accordingly. But its behaviour is different. This statement tests whether an expression matches one of a number of constant integer values labelled as cases. If one matches the expression, execution starts at that case. This is the general structure of the switch statement:

```
switch (expression)
{
        case constant:  statements
         break;
        case constant:  statements
        case constant:  statements;
        break;
        default: statements
}
```

The default clause is optional. If it is not there and none of the cases matches, no action is taken. The break keyword is used to skip the switch statement. For example if a case matches the expression and no break key words are used, the execution will go for all the statements in all cases.

Consider this example:

```
#include<stdio.h>
#include<conio.h>
int main ()
{
        char grade;
        printf("Enter a grade in capital letters");
        scanf("%c",&grade);
        switch(grade)
        {
                case 'A':
                printf("You have scored 90% marks");
                break;
        case 'B':
                printf("You have scored 80% marks");
                break;
        case 'C':
            printf("You have scored 70% marks");
            break;
        case 'D':
            printf("You have scored 60% marks");
            break;
        case 'F':
            printf("You have scored less than 50%");
            break;
        default:
            printf("You have entered an invalid grade");
            break;
        }
        getch();
}
```

5.5 Exercise for Lab

Exercise1: Prompt the user to input 5 values and display the minimum number amongst them.

Exercise2: Input 5 values from the user and display the number of positives, the number of negatives and the number of zeros amongst the 5 values.

Exercise3: Prompt the user to input a character and display whether it is a vowel or consonant using switch statement.

5.6 Home Task

1. Ask the user to enter marks obtained in a course and the total marks of the course. Then display a menu

   Press 1 to calculate percentage.

   Press 2 to display grade.

   If the user presses 1 then percentage should be displayed and if the user presses 2 the grade against the marks should be displayed. (Hint: use switch statement for menu selection and else if to display the grade).
2. Prompt the user to enter 3 values. For any equal values, the program should display the numbers that are equal. (For example user input 34,6,34 the program should display the message that the 1st and 3rd values are equal).

Lab # 06

LOOPS

6.1 Objective:

Learn about the primary mechanism for telling a computer that a sequence of tasks needs to be repeated a specific number of times.

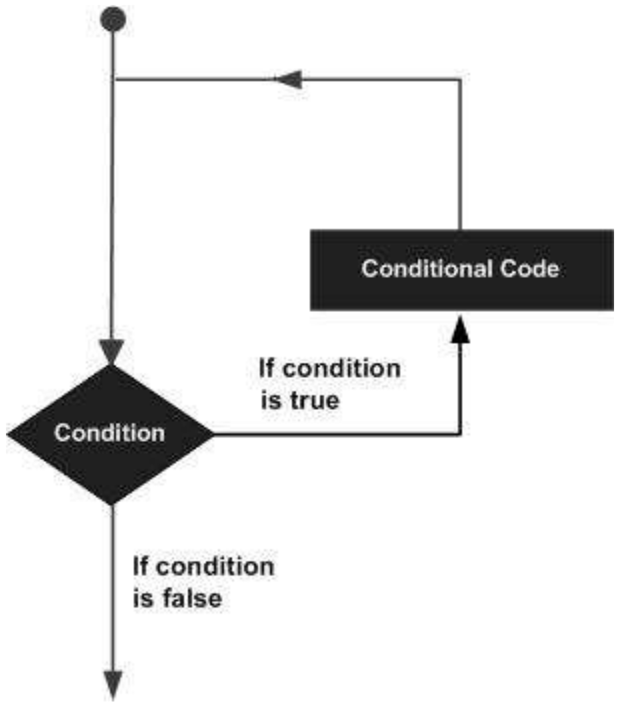
6.2 Scope:

The student should know the following:

1. Requirement to add a number to itself ten times.
2. To execute a statement or group of statements multiple times.

6.3 Useful Concepts:

Loop statements are the primary mechanism for telling a computer that a sequence of tasks needs to be repeated a specific number of times. Suppose, for example, that you have a requirement to add a number to itself ten times. A loop statement allows us to execute a statement or group of statements multiple times.



Why LOOPS are important?

Loops are relatively easy to implement, yet they can save you a lot of coding.

LOOPS can be of different types:

LOOPS can be divided as:

1. **While Loop:**

   Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

2. **Do while Loop:**

   Like a while statement, except that it tests the condition at the end of the loop body.

3. **For Loop:**

   Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

4. **Nested Loop:**

   You can use one or more loops inside any another while, for or do. While loop.

   Loops can further be distinguished as:

- Continue Statement: Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.Function returning no value but accepting one or more arguments.
- Break Statement: Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.

For loop:

- The for() loop is basically three statements in one line. Within the parentheses following the "for", you have the initial expression, condition, and the loop expression. These are separated by semicolons.
- **Syntax:**
- for (''initializer''; ''conditional expression''; ''loop expression'')
- {
- // statements to be executed
- }
- The initializer typically initializes a counter variable. Traditionally the variable name i is used for this purpose, though any valid variable name will do.
- **Example:**
- i = 0;
- The conditional expression specifies the test to perform to verify whether the loop has been performed the required number of iterations. For example, if we want to loop 10 times:
- i < 10;
- Finally the loop expression specifies the action to perform on the counter variable. For example to increment by 1:
- i++;

- The body of statements to be executed on each iteration of the loop is contained within the code block defined by the opening ({) and closing (}) braces. If only one statement is to be executed the braces are optional, though still recommended for code readability and so that you don't forget the add them if you later increase the number of statements to be performed in the loop.

While Loop:

The while() loop is better suited for cases where the program does not know how many times it is to be run, and instead checks for a different condition.

- Syntax:

```
while (input >= 0) { // "Greater-than or equal to zero" is the same as "not negative"
    sum = sum + input;
    printf("%d", &input);
  }
```

○

DO While Loop:

In a regular for() or while() loop, the condition is checked first, then the loop is executed. This means that if the initial condition was false to begin with, the body of the loop would never ever get executed; the code would skip directly over the loop. The do-while loop is guaranteed to perform one iteration of the loop first, before checking the condition. From that point on, as long as the condition remains true, the loop will continue to execute.

- Syntax:

```
 int n = 1;
do {
  sum = sum + n;
   n = n + 1;
} while (n <= 5)
```

Break and Continue Statement:

There are two statements (break; and continue ;) built in C programming to alter the normal flow of program. Loops are used to perform repetitive task until test expression is false but sometimes it is desirable to skip some statement/s inside loop or terminate the loop immediately with checking test condition. On these type of scenarios, continue; statement and break; statement is used respectively. The break statement is also used to terminate switch statement.

- Syntax:

```
break;
```

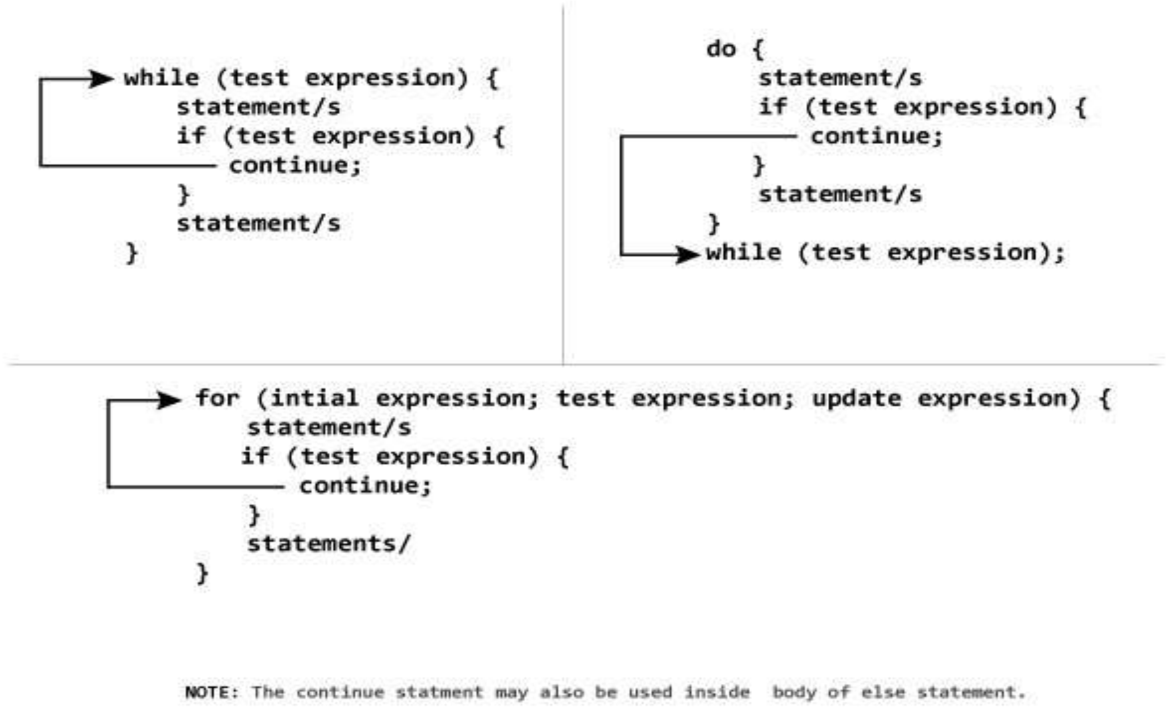
It is sometimes necessary to skip some statement/s inside the loop. In that case, continue; statement is used in C++ programming.

Syntax:

continue;

Working Body:

Syntax:



```
while (test expression) {
    statement/s
    if (test expression) {
        continue;
    }
    statement/s
}
```

```
do {
    statement/s
    if (test expression) {
        continue;
    }
    statement/s
}
while (test expression);
```

```
for (intial expression; test expression; update expression) {
    statement/s
    if (test expression) {
        continue;
    }
    statements/
}
```

NOTE: The continue statment may also be used inside body of else statement.

6.4 Examples

Example - 1:- Write a program Show all odd numbers using a for loop.

```
#include <iostream>
using namespace std;
int main()
{
        for (int count = 1; count <= 41; count += 2)
        {
                cout << count << ", ";
        }
        cout << endl;
        return 0;
}
```

Output:

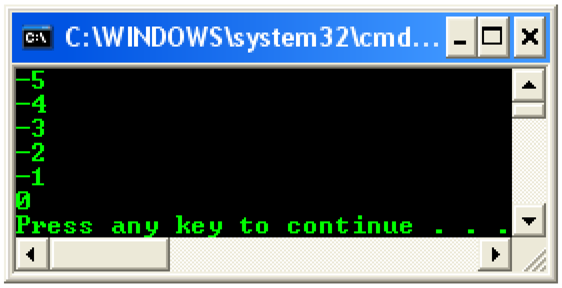
```
1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41,
```

Example - 2:- Write a program using do-while loop to display following output.

```
-5
-4
-3
-2
-1
0
```

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

int main()
{
    int j = -5; // initialization
    do
    {
        printf("%d\n", j);
        j = j + 1;
    }
    while(j <= 0);  // condition
    return 0;
}
```

Output:

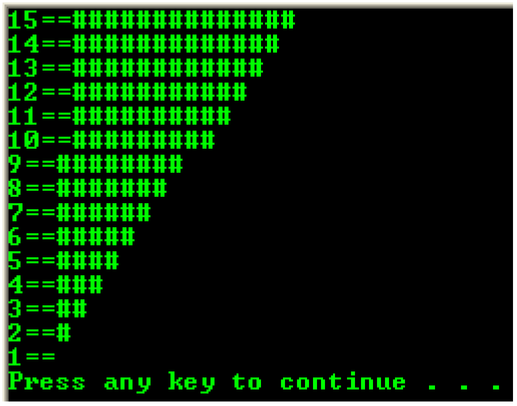


```
C:\WINDOWS\system32\cmd...
-5
-4
-3
-2
-1
0
Press any key to continue . . .
```

Example - 3:- Write a program that contains nested loops to draw a pyramid showing in output.

```
#include <stdio.h>
 int main()
{
    // variables for counter...
    int i = 15, j;
    // outer loop, execute this first...
    // for every i iteration, execute the inner loop
    while(i>0)
    {
        // display i==
        printf("%d==", i);
        // then, execute inner loop with loop index j,
        // the initial value of j is i - 1
        j=i-1;
        while(j>0)
        {
            // display #
            printf("#");
            // decrement j by 1 until j>10, i.e j = 9
            j = j - 1;
        }
        // go to new line, new row
        printf("\n");
        // decrement i by 1, repeat until i > 0 that is i = 1
        i = i - 1;
    }
    return 0;
}
```

Output:

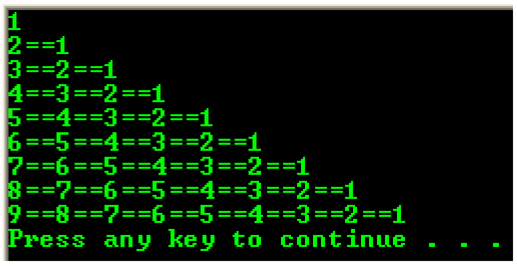


```
15==##############
14==#############
13==############
12==###########
11==##########
10==#########
9==########
8==#######
7==######
6==#####
5==####
4==###
3==##
2==#
1==
Press any key to continue . . .
```

Example - 4:- Write a program that contains nested loops to display pyramid as shown in output.

```
# include <stdio.h>

# include<conio.h>

int main()

{

      // variables for counter…

      int i, j;

      // outer loop, execute this first...for every i iteration,

      // execute the inner loop

      for(i = 1; i <= 9;)

      {

            // display i
            printf("%d", i);

            // then, execute inner loop with loop index j,

            // the initial value of j is i - 1

            for(j = i-1; j>0; )

            {

                  // display ==j

                  printf("==%d", j);

                  // decrement j by 1 until j>0, i.e j = 1

                  j = j - 1;

            }

              // go to new line

              printf("\n");

              // increment i by 1, repeat until i<=9, i.e i = 9

              i = i + 1;

      }

      return 0;

}
```

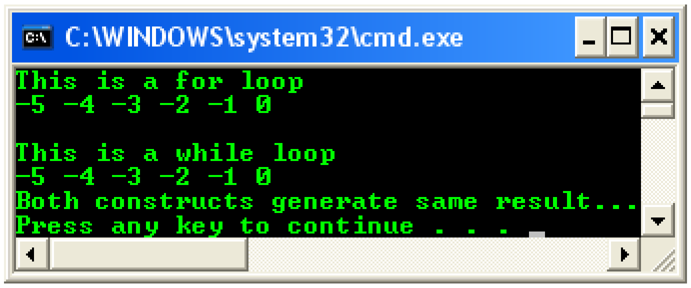
Output:



```
1
2==1
3==2==1
4==3==2==1
5==4==3==2==1
6==5==4==3==2==1
7==6==5==4==3==2==1
8==7==6==5==4==3==2==1
9==8==7==6==5==4==3==2==1
Press any key to continue . . .
```

Example - 5:- Write a program that display negative numbers from 0 to 5.

```
# include <stdio.h>
# include<conio.h>
int main(void)
{
    int i, j;
    // for loop
    printf("This is a for loop\n");
    for(i = -5; i <= 0; i = i +1) // initial, terminal condition and iteration
        printf("%d ", i);
    printf("\n");
    printf("\nThis is a while loop\n");
    j = -5; // initial condition
    // while loop
    while(j <= 0) // terminal condition
    {
        printf("%d ", j);
        j = j + 1;  // iteration
    }
    printf("\nBoth constructs generate same result...\n");
    return 0;
}
```



```
This is a for loop
-5 -4 -3 -2 -1 0

This is a while loop
-5 -4 -3 -2 -1 0
Both constructs generate same result...
Press any key to continue . . .
```

.

6.5 Exercises for lab

Exercises-1) Write a loop to do the following:

1. Read in 10 integers.

2. Find the maximum value.

3. Find the minimum value.

Exercises -2) Write a three-letter acronym-generating program. The program's output lists all three-letter combinations from AAA through ZZZ, spewed out each on a line by itself.

6.6 Home Work

1) Write a program that calculate the factorial of a number using while loop.

2) Write a program that inputs a number from a user and display its multiples using for loop.

3) Write a program that inputs a number and length from a user and display its table.

Lab # 7

Arrays

7.1 Objective:

Learn how to declare, initialize and use one-dimensional arrays.

7.2 Scope:

The student should know the following:

3. Syntax of array declaration
4. assigning and processing and elements

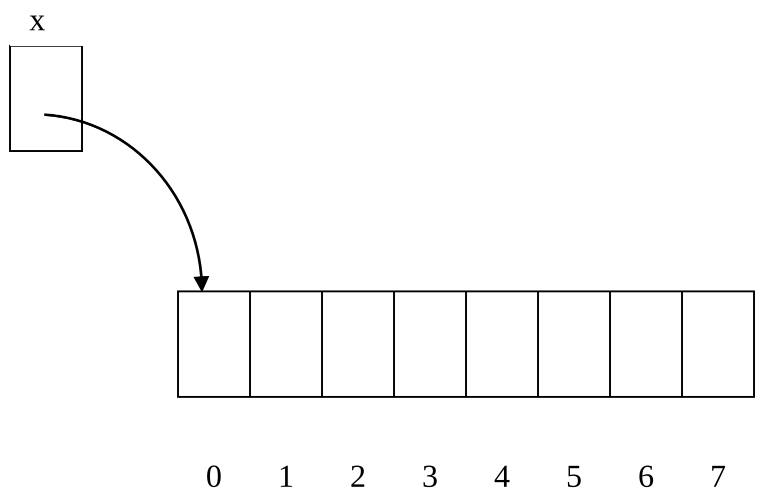
7.3 Useful Concepts:

An array is a collection of two or more adjacent memory cells, called array elements that are associated with a particular symbolic name. Arrays are very useful construct to store related values together instead of declaring several variables for each value.

To set up an array in memory, we must declare both the name of the array and the number of cells associated with it.

The following declaration will instruct the compiler to allocate eight memory cells with the name x; these memory cells will be adjacent to each other. Each element may contain a value of type double.

double x[8];



As you can see; the name of the array x points to the first element of the array.

We can define a constant for array size and use it whenever we declare an array.

```
#define NUM_STUDENTS  8
```

```
int  id[NUM_STUDENTS];
double gpa[NUM_STUDENTS];
```

To access the data stored in an array, we reference each individual element by specifying the array name and identifying the element desired.

x[0]	the value of the first element
x[3]	the value of the fourth element
x[7]	the value of the eighth element

NOTE: *The indices of the elements start from 0 not from one.*

Array initialization:

We can initialize an array directly by specifying each cell value individually as follows:

```
double values [] = {12.5,17.9,23.5,-2.5,115.75,-55.3};
```

Hence here we do not need to specify the size of the array or how many elements this array should have. This number can be deduced from the initialization list.

We can also use for loop which is the common way to deal with arrays in general. The following code initializes array square to squares of the indices:

```
int square [8], i;
for (i=0; i < 8; ++i)
  square[i] = i * i;
```

The array square will be like this after the loop:

0	1	4	9	16	25	36	49
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

Array Processing:

Elements of the array are dealt as normal variables, the only difference here is to specify the index desired. These statements are examples of using array elements:

16.0	12.0	28.0	26.0	2.5	12.0	14.0	-54.5
x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]

cout<< x[0];	display the value of x[0], which is 16.0
x[3] = 25.0;	stores the value 25.0 in x[3]
sum = x[0] + x[1];	stores the sum of x[0] and x[1], which is 28.0 in the variable sum
sum += x[2];	adds x[2] to sum. The new sum is 34.0
x[3] += 1.0;	increments x[3] by 1
x[2] = x[0] + x[1];	stores the sum of x[0] and x[1] in x[2] the new x[2] is 28.0

We can apply any expression to specify the index as well. Consider these statements:

i = 5;

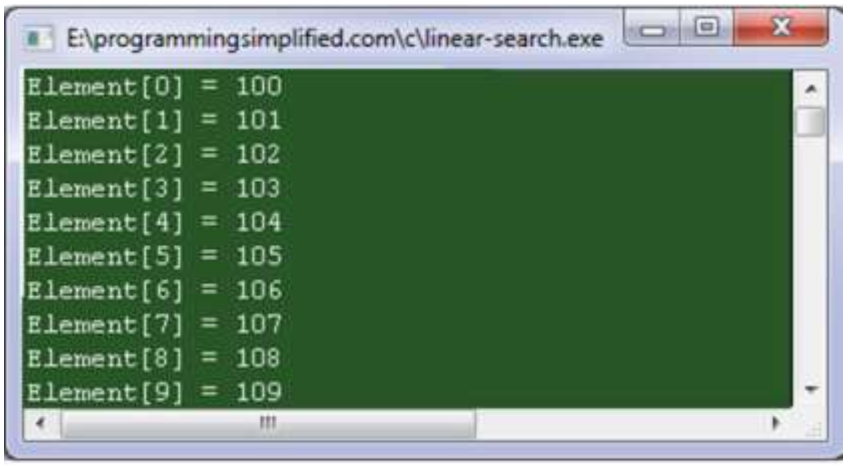
cout<< x[i+1];	display 14.0 (value of x[6])
x[i-1] = x[i];	assigns 12.0 (value of x[5]) to x[4]
cout<<x[i++];	display 12.0 (value of x[5])
cout<<x[2*i];	Invalid index. Attempt to display x[10]

7.4 Examples

Example – 1:- Write a program that contains an array of ten elements, take ten values from user, assign the value to each element of array and display the all array elements.

```
#include <stdio.h>
 int main ()
{
   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;

   n[10]={100, 101, 102, 103, 104, 105, 106, 107, 108,109 };

   /* output each array element's value */
   for (j = 0; j < 10; j++ )
   {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

Output:-



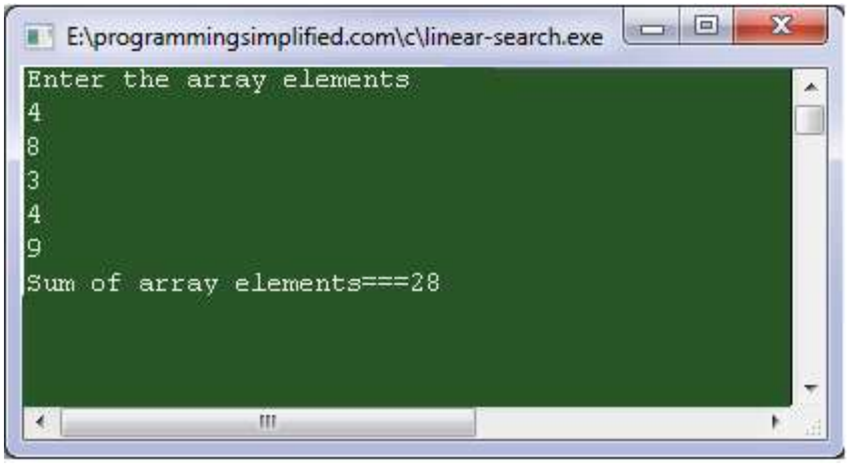
E:\programmingsimplified.com\c\linear-search.exe

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

Example – 2:- Write a program that adds all elements of array.

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

void main()
{
int a[5];
int i,sum=0;
clrscr();
printf("Enter the array elements\n");
for (i=0; i<5; i++)
scanf("%d",&a[i]);

// sum of array elements
for (i=0; i<5; i++)
{
sum=sum+a[i];
}
printf("Sum of array elements===%d",sum);
} // end of main()
```

Output:-

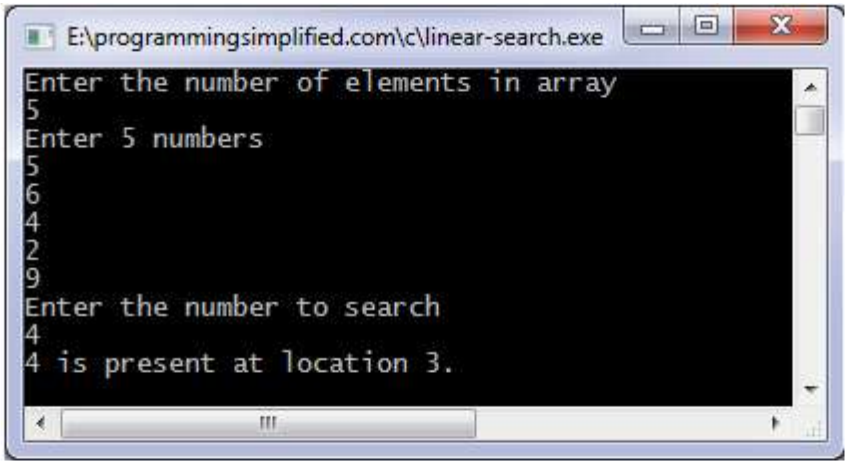


```
Enter the array elements
4
8
3
4
9
Sum of array elements===28
```

Example – 3:- Write a program that takes an integer value from user and search it in array, If found then display message 'Number Found' else 'Number Not Found.

```
#include <stdio.h>

int main()
{
  int array[100], search, c, n;

  printf("Enter the number of elements in array\n");
  scanf("%d",&n);

  printf("Enter %d integer(s)\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  printf("Enter the number to search\n");
  scanf("%d", &search);

  for (c = 0; c < n; c++)
  {
    if (array[c] == search)     /* if required element found */
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d is not present in array.\n", search);

  return 0;
}
```

Output:-



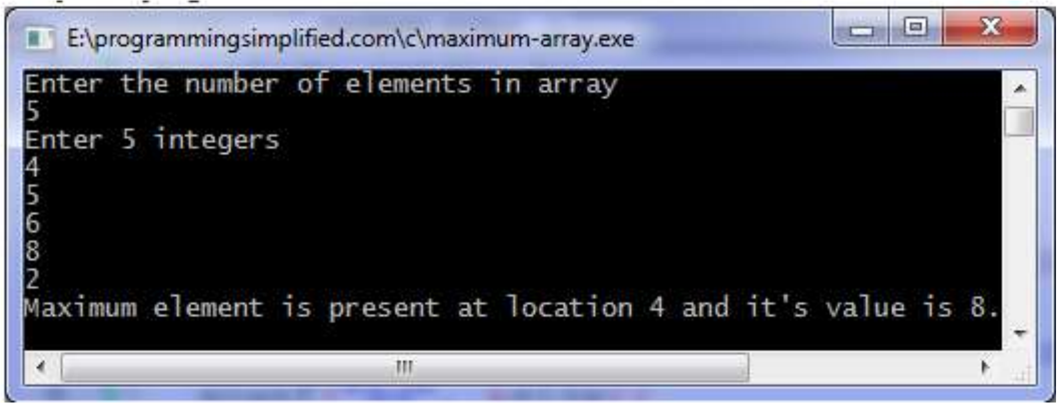
E:\programmingsimplified.com\c\linear-search.exe

```
Enter the number of elements in array
5
Enter 5 numbers
5
6
4
2
9
Enter the number to search
4
4 is present at location 3.
```

Example – 4:- Write a Program that finds maximum number in an array

```
#include <stdio.h>

int main()
{
  int array[100], maximum, size, c, location = 1;

  printf("Enter the number of elements in array\n");
  scanf("%d", &size);

  printf("Enter %d integers\n", size);

  for (c = 0; c < size; c++)
    scanf("%d", &array[c]);
  maximum = array[0];

  for (c = 1; c < size; c++)
  {
    if (array[c] > maximum)
    {
       maximum  = array[c];
       location = c+1;
    }
  }

  printf("Maximum element is present at location %d and it's value is %d.\n", location, maximum);
  return 0;
}
```

Output:-



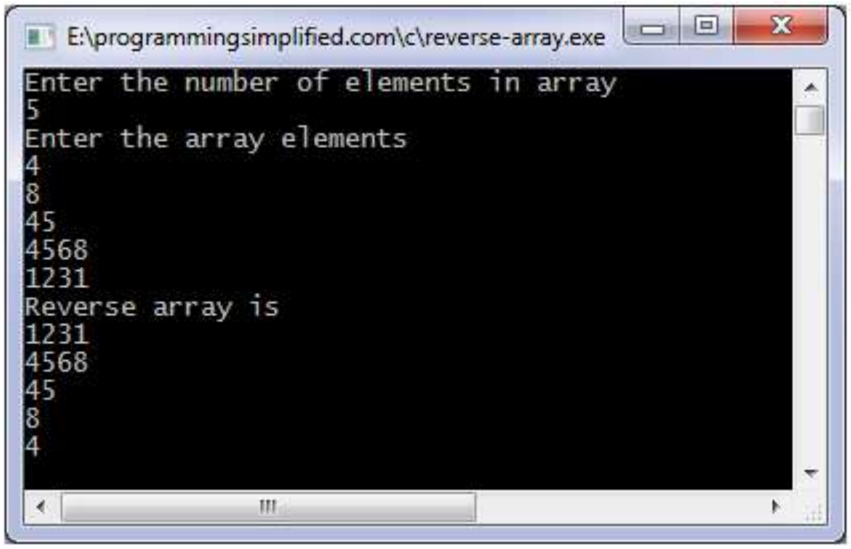
```
Enter the number of elements in array
5
Enter 5 integers
4
5
6
8
2
Maximum element is present at location 4 and it's value is 8.
```

Example – 5:- Write a program that shows reverse of an array.

```
#include <stdio.h>
 int main()
{
   int n, c, d, a[100], b[100];

   printf("Enter the number of elements in array\n");
   scanf("%d", &n);

   printf("Enter the array elements\n");

   for (c = 0; c < n ; c++)
      scanf("%d", &a[c]);

   for (c = n - 1, d = 0; c >= 0; c--, d++)
      b[d] = a[c];
```

```
    //Copying reversed array into original.
     //Here we are modifying original array, this is optional.


   for (c = 0; c < n; c++)
      a[c] = b[c];

   printf("Reverse array is\n");

   for (c = 0; c < n; c++)
      printf("%d\n", a[c]);

   return 0;
}
```

Output:-



E:\programmingsimplified.com\c\reverse-array.exe

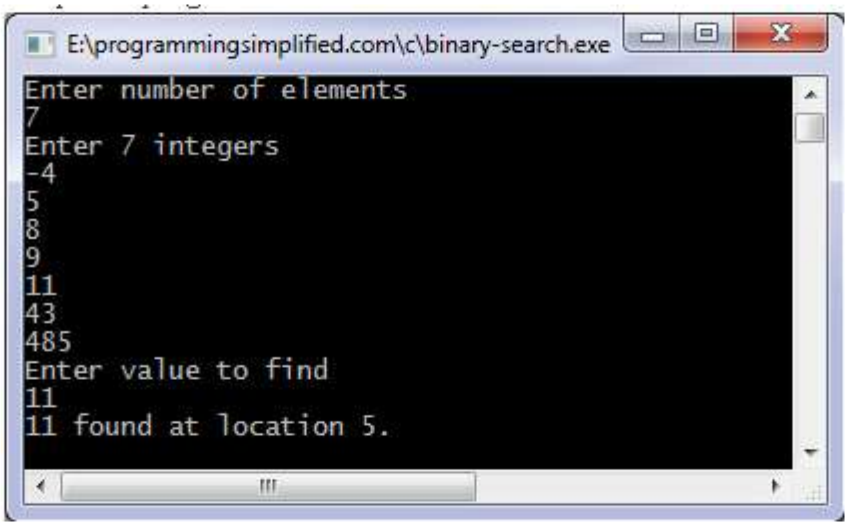
```
Enter the number of elements in array
5
Enter the array elements
4
8
45
4568
1231
Reverse array is
1231
4568
45
8
4
```

Example – 6:-Write a Program for binary search.

```
#include <stdio.h>

int main()
{
   int c, first, last, middle, n, search, array[100];
    printf("Enter number of elements\n");
   scanf("%d",&n);
    printf("Enter %d integers\n", n);
    for (c = 0; c < n; c++)
      scanf("%d",&array[c]);
    printf("Enter value to find\n");
   scanf("%d", &search);
    first = 0;
```

```
  last = n - 1;
  middle = (first+last)/2;

  while (first <= last) {
    if (array[middle] < search)
      first = middle + 1;
    else if (array[middle] == search) {
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;
     middle = (first + last)/2;
  }
  if (first > last)
    printf("Not found! %d is not present in the list.\n", search);
   return 0;
}
```

Output:-

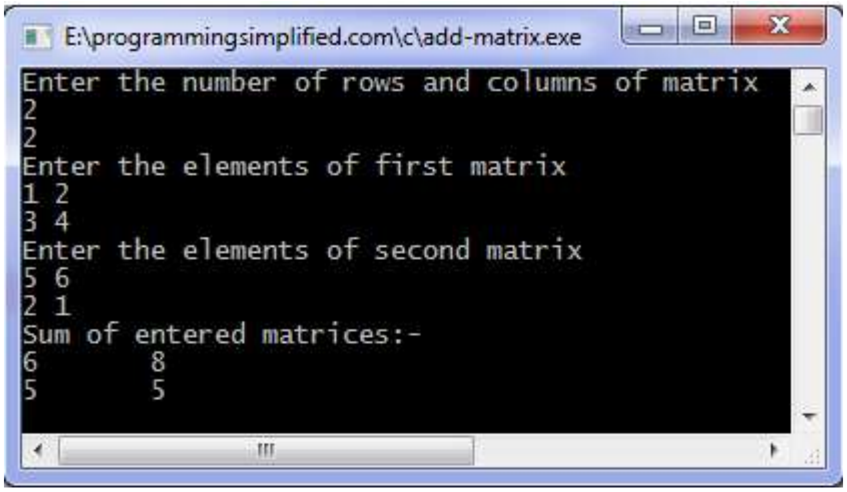


```
Enter number of elements
7
Enter 7 integers
-4
5
8
9
11
43
485
Enter value to find
11
11 found at location 5.
```

Example -7:- Write a Program to find sum of two matrices.

```
#include <stdio.h>

int main()
{
  int m, n, c, d, first[10][10], second[10][10], sum[10][10];

  printf("Enter the number of rows and columns of matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter the elements of first matrix\n");
  for (c = 0; c < m; c++)
    for (d = 0; d < n; d++)
      scanf("%d", &first[c][d]);

  printf("Enter the elements of second matrix\n");

  for (c = 0; c < m; c++)
    for (d = 0 ; d < n; d++)
        scanf("%d", &second[c][d]);

  printf("Sum of entered matrices:-\n");

  for (c = 0; c < m; c++) {
    for (d = 0 ; d < n; d++) {
      sum[c][d] = first[c][d] + second[c][d];
      printf("%d\t", sum[c][d]);
    }
    printf("\n");
  }

  return 0;
}
```

Output:-

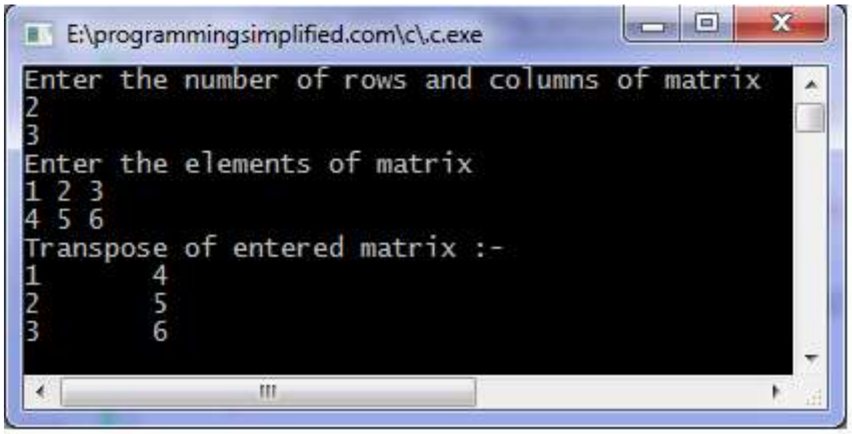


```
E:\programmingsimplified.com\c\add-matrix.exe
Enter the number of rows and columns of matrix
2
2
Enter the elements of first matrix
1 2
3 4
Enter the elements of second matrix
5 6
2 1
Sum of entered matrices:-
6       8
5       5
```

Example – 8:- Write a program to find transpose of matrix.

```
#include <stdio.h>

int main()
{
   int m, n, c, d, matrix[10][10], transpose[10][10];

   printf("Enter the number of rows and columns of matrix\n");
   scanf("%d%d", &m, &n);

   printf("Enter the elements of matrix\n");

   for (c = 0; c < m; c++)
      for(d = 0; d < n; d++)
         scanf("%d",&matrix[c][d]);

   for (c = 0; c < m; c++)
      for( d = 0 ; d < n ; d++ )
         transpose[d][c] = matrix[c][d];

   printf("Transpose of entered matrix :-\n");

   for (c = 0; c < n; c++) {
      for (d = 0; d < m; d++)
         printf("%d\t",transpose[c][d]);
      printf("\n");
   }

   return 0;
}
```

Output:-



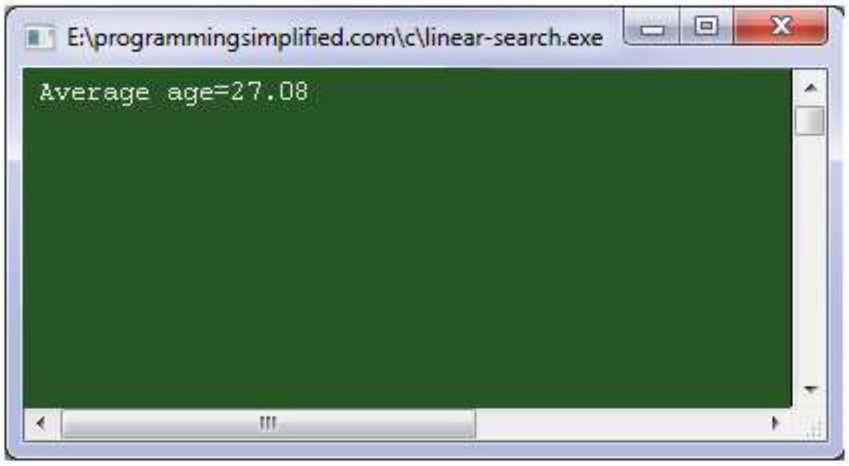
```
Enter the number of rows and columns of matrix
2
3
Enter the elements of matrix
1 2 3
4 5 6
Transpose of entered matrix :-
1       4
2       5
3       6
```

Example – 9:- Write a program that passes an array containing age of person to a function. This function should find average age and display the average age in main function

```
#include <stdio.h>
float average(float a[]);
```

```
int main(){
     float avg, c[]={23.4, 55, 22.6, 3, 40.5, 18};
     avg=average(c);   /* Only name of array is passed as argument. */
     printf("Average age=%.2f",avg);
     return 0;
   }
float average(float a[]){
     int i;
     float avg, sum=0.0;
     for(i=0;i<6;++i){
       sum+=a[i];
     }
     avg =(sum/6);
     return avg;
}
```

Output:-



E:\programmingsimplified.com\c\linear-search.exe

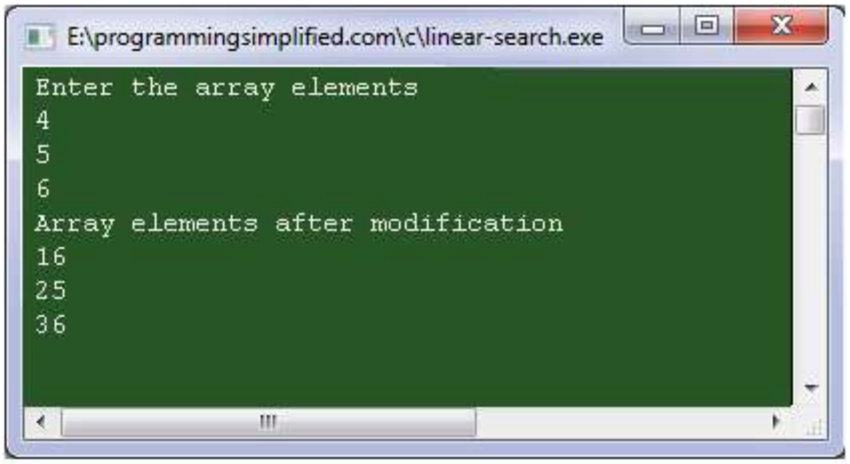
```
Average age=27.08
```

Example – 10:- Write a Program to show Array elements modified by a function.

```
#include<stdio.h>
#include<conio.h>
void modify(int b[3]);
void main()
{
int arr[3] = {1,2,3};
modify(arr);
for(i=0;i<3;i++)
  printf("%d",arr[i]);
getch();
}
```

```
void modify(int a[3])
{
int i;
for(i=0;i<3;i++)
  a[i] = a[i]*a[i];
}
```

Output:-



E:\programmingsimplified.com\c\linear-search.exe

```
Enter the array elements
4
5
6
Array elements after modification
16
25
36
```

7.5 Exercises for lab

Exercises-1) Write a program that declares an array of 5 double numbers. Use a loop to read 5 real numbers from user and fill the array. Then print the following on screen:

a. All the elements of array
b. The average of all the numbers in the array
c. The numbers below average
d. The numbers above average.

Exercises -2) Apply Bubble sort algorithms to sort elements of given array

7.6 Home Work

1) Consider the following list of student's grade

64	36	56	47	40	54	61	60	58	64	54
48	59	45	63	54	50	49	51	60	58	59

Initialize an array with above grades and find the following things about the above data.

a) The minimum grade
b) The maximum grade
c) Average

2) Example three only tells that number found in an array or not. Your program should also notify:

a) Index of the array where value found
b) Number of times element found in array

3) Find multiplication of two matrices using arrays

Lab # 8

2-D Arrays

8.1 Objective:

Learn how to declare, assign and manipulate two dimensions array.

8.2 Scope:

The student should know the following:

1. What is 2-D array?
2. Declaring 2-D arrays and bound checking.
3. Storing data in 2-D arrays and printing.
4. Passing 2-D arrays to functions.

8.3 Useful Concepts:

What is 2-D Array?

Just like single dimensional arrays we can have multidimensional arrays which are arrays of arrays. The multidimensional arrays can be visualized as a matrix.

	[0]	[1]	[2]
[0]	1	2	3
[1]	3	4	5
[2]	6	7	8

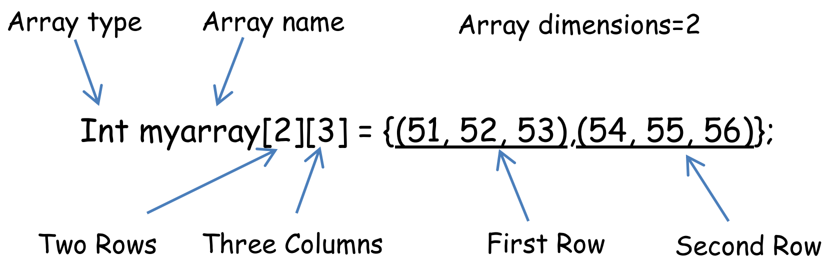
General representation of 2-D Array

How to declare and initialize two multidimensional Array?

```
int myarray[2][3];
```

The name of the array to be" myarray", type of the array elements to be "int", dimension to be 2 (two pairs of brackets [])

The number of elements or size to be 2*3 = 6



```
Int myarray[2][3] = {(51, 52, 53),(54, 55, 56)};
```

int b[2][3] = {51, 52, 53, 54, 55, 56};

- b[0][0] = 51   b[0][1] = 52   b[0][2] = 53

- Use braces to separate rows in 2-D arrays.

  - int c[4][3] = {{1, 2, 3},

```
{4, 5, 6},
{7, 8, 9},
{10, 11, 12}};
```

  - int c[ ][3] = {{1, 2, 3},

```
{4, 5, 6},
{7, 8, 9},
{10, 11, 12}};
```

Implicitly declares the number of rows to be 4.

Storing data in 2-D arrays and printing ?

- A nested for loop is used to input elemts in a two dimensional array.
- In this way by increasing the index value of the array the elements can be entered in a 2d array

```
for(i=0 ; i<3 ; i++)
  {
        for(j=0 ; j<3 ; j++)
        {
            cin<a[i][j];
        }
   }
```

- The output of two-dimensional arrays should be in the form of rows and columns for readability. Nested *for* loops are used to print the rows and columns in row and column order.
- By increasing the index value of the array the elements stored at that index value are printed on the output screen.

```
for(i=0 ; i<3 ; i++)
  {
        for(j=0 ; j<3 ; j++)
        {
            cout<<a[i][j];
        } cout<<endl;
   }
```

Passing 2-D arrays to function:

There are three ways to pass a 2D array to a function:

1. The parameter is a 2D array

```
int array[10][10];
void passFunc(int a[][10])
{
    // ...
}
passFunc(array);
```

2. The parameter is an array containing pointers

```
int *array[10];
for(int i = 0; i < 10; i++)
    array[i] = new int[10];
void passFunc(int *a[10]) //Array containing pointers
{
    // ...
}
```

```
passFunc(array);
```

3. The parameter is a pointer to a pointer

```
int **array;
array = new int *[10];
for(int i = 0; i <10; i++)
    array[i] = new int[10];
void passFunc(int **a)
{
    // ...
}
```

```
passFunc(array);
```

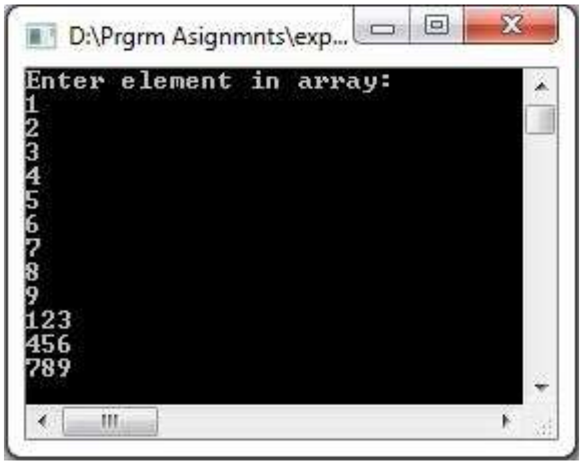
8.4 Examples

Example -1:- A program to input elements in a two dimensional array and print it.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[3][3];
   int i,j;
```

```
    clrscr();
     printf("enter the elements in the array:");
for(i=0 ; i<3 ; i++)
   {
            for(j=0 ; j<3 ; j++)
            {
                  scanf("%d",&a[i][j]);
            }
      }
for(i=0 ; i<3 ; i++)
   {
            for(j=0 ; j<3 ; j++)
            {
                  printf("%d",a[i][j]);
            } printf("\n");
      }
 getch();
}
```

Output:-

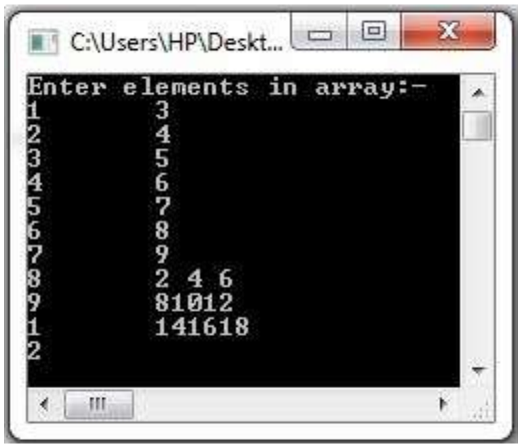


```
Enter element in array:
1
2
3
4
5
6
7
8
9
123
456
789
```

Example -2:- Write a program that will add two matrixes entered by the user and print it.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[3][3],b[3][3],c[3][3];
   int i,j;
   clrscr();
   printf("enter the elements in both the array:");
for(i=0 ; i<3 ; i++)
   {
         for(j=0 ; j<3 ; j++)
         {
              scanf("%d",&a[i][j]);
         }
    }
for(i=0 ; i<3 ; i++)
   {
         for(j=0 ; j<3 ; j++)
         {
              scanf("%d",&b[i][j]);
         }
   }
for(i=0 ; i<3 ; i++)
   {
         for(j=0 ; j<3 ; j++)
         {
              c[i][j]=a[i][j]+b[i][j];
              printf("%d",c[i][j]);
         }
         printf("\n");
     }
 getch();
}
```

Output:-



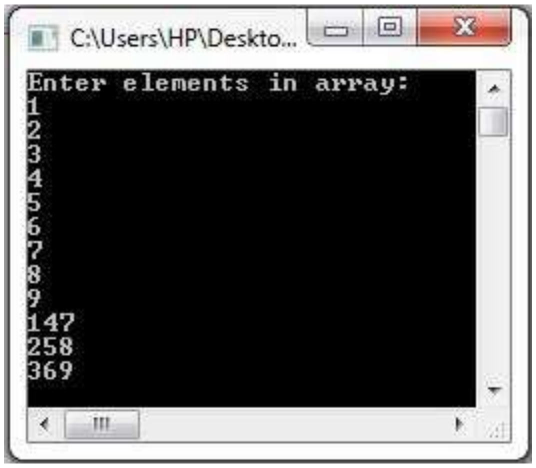
```
Enter elements in array:-
1           3
2           4
3           5
4           6
5           7
6           8
7           9
8           2 4 6
9           81012
1           141618
2
```

Example -3:- A program to input a matrix and print its transpose.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[3][3],b[3][3];
   int i,j;
```

```
    clrscr();
    printf("enter the elements in the array");
for(i=0 ; i<3 ; i++)
    {
            for(j=0 ; j<3 ; j++)
            {
                  scanf("%d",&a[i][j]);
            }
      }
for(j=0 ; i<3 ; i++)
    {
            for(i=0 ; j<3 ; j++)
            {
                  printf("%2d",&b[j][i]);
            }
    }
getch();
}
```

Output:-



```
Enter elements in array:
1
2
3
4
5
6
7
8
9
147
258
369
```

Example -4:- passing multi-dimensional array to function

```
#include<stdio.h>
#include<conio.h>
void Function(int c[2][2]);
int main(){
  int c[2][2],i,j;
  printf("Enter 4 numbers:\n");
  for(i=0;i<2;++i)
    for(j=0;j<2;++j){
       scanf("%d",&c[i][j]);
    }
  Function(c);   /* passing multi-dimensional array to function */
  return 0;
}
void Function(int c[2][2]){
/* Instead to above line, void Function(int c[][2]){ is also valid */
  int i,j;
  printf("Displaying:\n");
  for(i=0;i<2;++i)
    for(j=0;j<2;++j)
       printf("%d\n",c[i][j]);
}
```

8.5 Exercises for lab

- Exercises-1) Declare and initialize a 2D arrays with different methods.
- Exercises-2) multiply two matrices and store result in third matric.
- Exercises-3) Write a program that will sort 2d array in ascending order and descending order.
- Exercise-4) A program to input a matrix and pass to function(using all methods).
- Exercises-5) A program to input a matrix and find its determent(by using function).

8.6 Home Work

- A program to input a matrix and print its transpose.

- A program to input a matrix and find its inverse.
- A program to input a matrix and find its singular or non-singular(by using function).
- A program to input a matrix and find its mean ,median ,standard deviation(write separate function).

Lab # 9

Structures in C

9.1 Objective:

To familiarize the students with the use of structure and nested structure

9.2 Scope:

The student should know the following at the end of this lab:

1. How to define and declare structures
2. How to define nested Structures
3. Pointers to the structures
4. Array of structures
5. Passing structures to the function

9.3 Useful Concept

How to declare and initialize structures:

The elements can be of any type including enumerated types, arrays and even other structures.

```
e.g: struct user_record {
                    int id_no;
                    STRING name;
                    STRING dept;
                    int no_of_books;
                    USER_STATUS status;
                      };
```

This declares a structure with user_record as tag. Again, memory is not allocated until a variable is declared.

A variable with the above structure can be declared as:

```
struct user_record user;
```

We can simplify the variable declaration above, if we use typedef in the declaration.

The above can then be defined as:

```
typedef struct {
     int id_no;
    STRING name;
    STRING dept;
    int no_of_books;
    USER_STATUS status;
  } USER_RECORD;
```

The variable declaration then becomes:

```
USER_RECORD user;
```

Once a variable is declared, we can access the individual elements (called fields) of the Record using the member operator, also called the dot (.) operator.

We can combine declarations and declare structure variables at the same time that the structure type is declared. For example:

```
 struct date
{
int day, month, year;
          } arrival_date, departure_date;
```

or without a structure tag

```
 structure { int day, month, year;} arrival_date, departure_date;
```

Structure Initialization:

```
struct employee agent1={“Omer Ali”,96,3524};
```

For example, the following program reads and prints values for the structure variable user:

```
 #include <stdio.h>
#include <string.h>
typedef char STRING[81];
typedef enum {STUDENT, STAFF, FACULTY} USER_STATUS;
typedef struct {int id_no;
```

```
                    STRING name;
                    STRING dept;
                    int no_of_books;
                    USER_STATUS status;
                       } USER_RECORD;

main()
{
  USER_RECORD user;
  user.no_of_books=0;
  printf("Enter user details:\n");
  printf ("ID_NO: ");
  scanf("%d", user.id_no);
              printf("NAME: ");
              fflush(stdin);
              gets(user.name);
              printf("DEPARTMENT: ");
              gets(user.dept);
              printf("STATUS [0 for student, 1 for staff or 2 for faculty: ");
  scanf("%s",user.status);
  printf("\nThe details you entered are:\n");
  printf("\nID_NO: %d    , user.id_no);
  printf("\nNAME:  %s", user.name);
  printf("\nDEPARTMENT:%s ", user.dept);
  printf("\nSTATUS:   %s  ", user.status);
  printf("\nNO OF BOOKS  %d ",user.no_of_books);
        return 0;
}
```

Nested or Compound Structure:

A structure can have as its field another structure.

e.g.

```
    typedef struct {
                                        int     day;
                                        int     month;
                                        int     year;
                                        }  DATE;

     typedef struct {
                                        int     bk_no;
                                         STRING author;
                                         STRING title;
                                         STRING  publisher;
                                         DATE pub_date;
                                         BOOK_STATUS status;
                                         int edition           } BOOK_RECORD;
                            BOOK_RECORD book;
```

Notice that we need two dot operators to access the publication date in the variable book as shown below:

```
book.pub_date.day=2;
book.pub_date.month=5;
book.pub_date.year=1980;
```

Arrays of structure:

Just as we use an array of standard data types, we can define an *array of structures*. For example,

```
                              struct  date  dates [10] ;
```

defines an *array* called dates, which consists of 10 elements. Each element inside the array is defined to be of type struct date. Referencing a particular structure element inside the array is done, by using the *subscripts* as done for normal variables. For example, to set the day of the 4th structure element of dates array to 25, we write

```
                        dates[3].day = 25 ;
```

Assignment and comparison among structures:

Like simple variables, and unlike arrays, it is allowed to assign an entire structure to another structure of the same type.

```
        e.g.  BOOK_RECORD book1, book2;
              . .  . . . . . . .           /* read values for book1 */
              book2 = book1.
```

However, comparison is not allowed on entire structures. It has to be done on field by field basis.

```
              Thus        if (book1 = = book2)   /* is not allowed */
                   . . .  . . . . .
```

The following function checks if two books are the same.

```
int samebook (BOOK_RECORD book1,  BOOK_RECORD book2)
{   int  samebook;
    samebook = book1.bk_no == book2.bk_no;
    return samebook;
}
```

Structures and pointers:

We have seen how a pointer can be defined to point to a basic data type in C such as int or char. But pointers can also be defined to point to structures as well. Consider

```
                        struct  date
                        {
                              int    month ;
                             int      day ;
                             int       year ;
                         } ;
```

Just as we defined variables to be of type struct date, as in

```
struct  date   today ;
```

So can we define a variable to be a pointer to a struct date variable:

```
struct   date   *date_pointer ;
```

The variable date_pointer can be set to point to today with the assignment statement

```
date_pointer = &today ;
```

Once such an assignment has been made, we can then indirectly access any of the members of the date structure pointed to by date_pointer in the following way:

```
(*date_pointer).day = 21 ;
            OR
date_pointer->day = 21 ;
```

The symbol -> is known as structure pointer operator, which is *a minus sign immediately followed by a greater than sign.*

A pointer to a structure identifies its address in the memory, and is created in the same way that a pointer to a simple data type such as int or char is created.

9.4 Examples

Example -1:- Write a program that updates the record of a user and a book when the user borrows the book from our small library.

```
void  update_records( BOOK_RECORD *book,  USER_RECORD *user)
                  {   (*book).status = BORROWED;
                      (*user).no_of_books++;
                  }
```

In the above example, the * is necessary since book in the function is a pointer variable, so we need to dereference it to access the object it is pointing to.

The bracket in (*book).status is also necessary since the dot operator has high priority than *.

Another way to represent above statement is by using the arrow operator (->).

e.g.

```
book -> status;  is equivalent to   (*book).status;
book->pub_date.day; is equivalent to (*book).pub_date.day;
```

Different Ways of Passing structure to Function:

This is an interactive program. It shows different ways of passing struct to functions

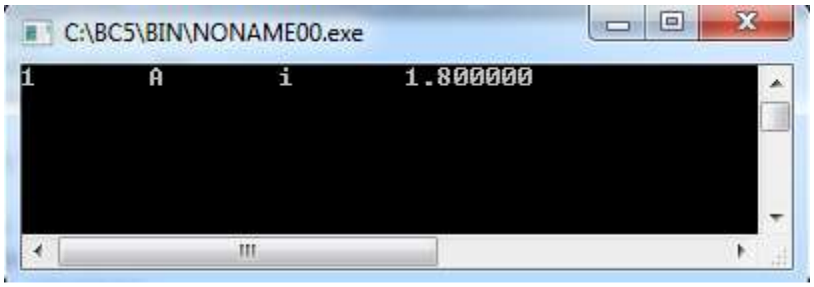
```
#include<stdio.h>
void  input  ( struct  car  * ) ;
void  output  ( char  *, char  *, int  * ) ;

 struct   car  {   char  maker [40] , model [40] ;
                              int   year ;
                  } ;
int  main  (  )
{
       /* declaring a structure variable */
struct   car   FirstCar  ;
       /* function calls */
       /* passing the structure address */
input ( &FirstCar ) ;

/* passing the addresses of the structure member variables */
output  (  FirstCar.maker, FirstCar.model, &FirstCar.year ) ;

puts ( "End of my act! :-) \n" ) ;
return 0 ;
}

/* input:          This function expects a pointer to a structure of type struct car. It reads
                   data items from the keyboard and saves them in the structure member
               variables.
*/
```

```
void  input  ( struct car  * sp  )
{          puts ( "*** Car Information System ***" ) ;     /* title message */
                    printf("What is the maker of your car? " ) ;     /* prompt */
                    gets ( sp -> maker ) ;                           /* read */
                    printf("What is the model of your car? " ) ;     /* prompt */
                    gets ( sp -> model ) ;                           /* read */
                    printf("What year is  your car? "  );                /* prompt */
                    scanf("%d",sp -> year)  ;                            /* read */
return ;
}
/*
output: This function expects pointers to two strings and a pointer to an int type variable. It
displays the specified two strings and the integer number.*/
void  output ( char  * s1, char  * s2,  int  * ip )
{
                    Printf("Your Car:%c %c  %d", s1, s2, * ip  ;
                    puts ("Nice Car!" ) ; return ;


 }
```

Example -2:- Write a program that implements the concept of nested structure.

```
void main()
{
        struct world
        {
        int a;
        char b;
        struct country
        {
        char c;
        float d;
        }p;
        };
        struct world st ={1,'A','i',1.8};
        clrscr();
        printf("%d\t%c\t%c\t%f",st.a,st.b,st.p.c,st.p.d);
        getch();
}
```

Output:



```
1         A         i         1.800000
```

9.5 Exercises for lab

Exercise -1) Find output of the following:

```
#include<stdio.h>
typedef struct { int hour, minutes, seconds;} time;
void ampm(int h, int m, int s)
{
if(h>0 && h<12)
    cout<<"The time now is "<< h<<":" m<<":" <<s<<"am";
else
    cout<<"The time now is "<<h <<":"<< m <<":" s<<"pm";
}
 int main ()
{
time t;
cout<<"Enter the hour";
cin>>t.hour;
cout<<"Enter the minutes";
cin>>t.minutes;
cout<<"Enter the second";
cin>>t.second;
ampm(t.hour, t.minutes, t.second);
return 0;
}
```

Exercise -2) Write a program that demonstrates the passing of individual structure member to the function.

Exercise -3) What will be output of following c code?

```
void main()
{
        struct employee
        {
        unsigned id: 8;
        unsigned sex:1;
        unsigned age:7;
        };
        struct employee emp1={203,1,23};
        clrscr();
        printf("%d\t%d\t%d",emp1.id,emp1.sex,emp1.age);
        getch();
}
```

Exercise -4) what will be output of following c code?

```
void main()
{
        struct field
        {
        int a;
```

```
            char b;
            }bit;
            struct field bit1={5,'A'};
            char *p=&bit1;
            *p=45;
            clrscr();
            printf("\n%d",bit1.a);
            getch();
}
```

Exercise -5) what will be output of following c code?

```
void main()
{
            Struct country
            {
            char c;
            float d;
            };
            struct world
            {
            int a[3];
            char b;
            struct country abc;
            };
            struct world st ={{1,2,3},'P','q',1.4};
            clrscr();
            printf("%d\t%c\t%c\t%f",st.a[1],st.b,st.abc.c,st.abc.d);
            getch();
}
```

9.6 Home Work

1) Write a program that demonstrates the passing of entire structure as parameters to the called function.

2) Write a program that demonstrates the passing address of the structure variable to the called function using pointers.
3) Write a structure that demonstrates the returning of array from a function.

Lab # 10

Function and Parameters

10.1 Objective:

Learn how to write user defined functions which do not return value (void) **OR** can return single value.

10.2 Scope:

The student should know the following:

5. Functions which neither accept any argument nor return any value(void)
6. Functions which accept arguments but do not return any value(void)
7. Functions which accept arguments and return a single result.

10.3 Useful Concepts:

Basic Concepts about Function :

A function is a block of code that performs a task. The statements written in a function are executed when it is called by its name. The functions provide a **structured programming** approach. It is a modular way of writing programs. The whole program logic is divided into a number of smaller **modules** or functions. The main function calls these function when they are needed to execute.

NOTE: The program should always have a main function whether it has other function or not. Without the main function the program will not do anything as the program always starts from the main function.

Why functions are important?

The program may need to repeat the same piece of code at various places or it may require repeating same task again and again. The program may become very large and messy if we do not divide it into sub programs (functions). The piece of code that is executed repeatedly is stored in a separate function, and is called when and where it is required.

Functions can be of different types:

Functions can be divided as:

1. **User- defined Functions:**

   Functions written by programmers are called user-defined functions.

2. **Built- in Functions:**

   A type of function that is available as a part of language is known as built-in function or library function. These functions are stored in different header files e.g. clrscr ( ) is a built-in function to clear the screen and it is part of header file called "conio.h".

   Function can further be distinguished as:

- Functions returning no value and accepting no arguments.
- Function returning no value but accepting one or more arguments.

- Function returning one value and accepting one or more arguments.

User Defined Functions:

A user-defined function consists of the following:

- Function declaration or function prototype
- Function definition

Functions Declaration or Function Prototype:

Function declaration provides information to compiler about the structure of the function to be used in program. It ends with a semi colon. Function prototypes are usually placed at the beginning of the source file before the main function. Function prototype consists of the following:

- Function return type
- Function name
- Number and types of parameters (Optional)

Syntax:

Return-type Function-name (Parameter list **(optional)**);

Examples:

1. **The function show neither accepts any argument nor returns any value.**

   Function name | *No parameters*

   void *indicates no return type* → ***void show ( );***

2. **The function alpha accepts a character type argument and returns no value.**

   Function name | ***char*** *indicates character parameter*

   void *indicates no return type* → ***void alpha (char);***

3. **The function alpha1 accepts a character type argument and returns a char value.**

   Function name | *char indicates character parameter*

   char *indicates character return type* → ***char alpha1 (char);***

4. **The function multiple accepts an integer type argument and returns an integer value.**

   Function name | *int indicates integer parameter*

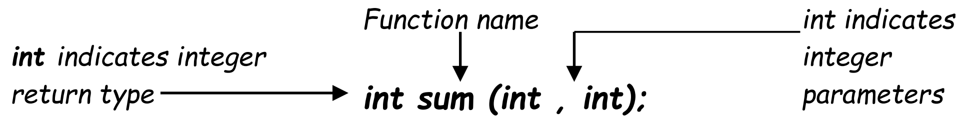
   int *indicates integer return type* → ***int multiple (int);***

5. **The function fact accepts an integer type argument and returns a long value.**

   Function name | *int indicates integer parameter*

   long *indicates long return type* → ***int fact (int);***

6. **<u>The function sum accepts two integer type arguments and returns an integer value.</u>**


Function name
int indicates integer return type
int indicates integer parameters

int sum (int , int);

Functions Definition:

A set of statements that explains what a function does is called function definition. The function definition can be written at the following places.

- Before main function
- After main function

Function declaration is not required if the function definition is written before main function. Function declaration is compulsory if function definition is written after main function. The function definition consists of two parts:

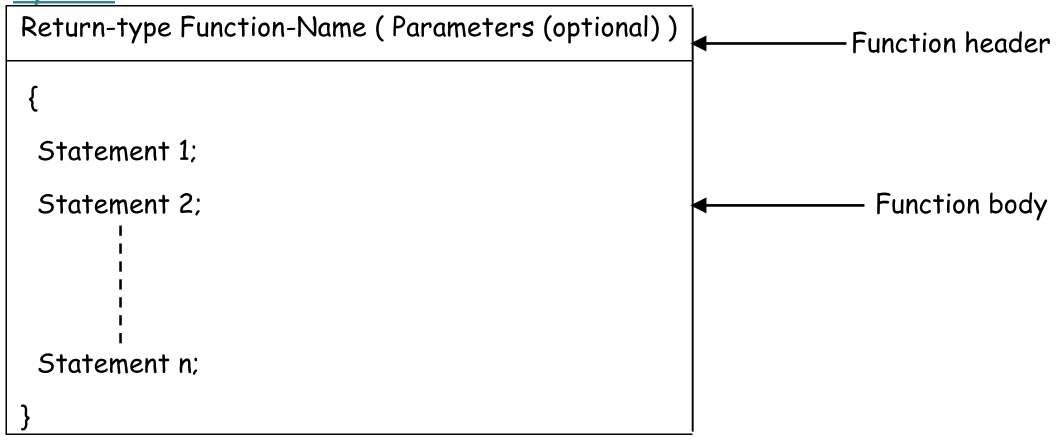
Functions Header:

The first line of function definition is called function header. It is similar to function prototype except that it is not terminated with semicolon. The number and sequence of parameter in function header must be same as that of function prototype.

Functions Body:

The set of statements written inside the function are called function body. The body of the function appears after function header and the statements are enclosed in curly braces { }.

Syntax:



```
Return-type Function-Name ( Parameters (optional) )

{
  Statement 1;
  Statement 2;
      |
      |
  Statement n;
}
```

Functions Call:

The statement that activates a function is called function call. A function is called with its name followed by required parameters in parenthesis "( ) ". If there are many parameters, these are separated by comas. If there are no parameter, empty parenthesis are used. Following steps show the process of function call.

- The control moves to the function that is called.

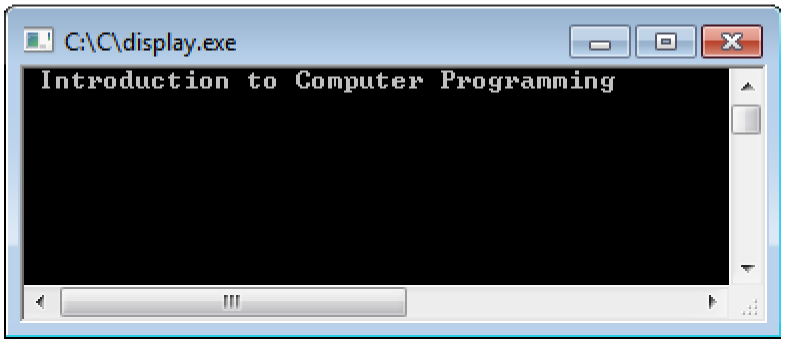
- All the statements in that function body are executed.
- The control returns back to the calling function.
- The rest of the body of calling function is executed.

10.4 Examples

Example - 1:- Write a program that contains a function which displays a message “Introduction to Computer Programming”.

```
# include <stdio.h>

# include<conio.h>

void display( );     //function declaration or prototype

void main( )

{

  clrscr( );

  display( );   //function call

  getch();

}

void display( )       //function definition

{

   printf(“ Introduction to Computer Programming”);

}
```

Output:



```
Introduction to Computer Programming
```

Execution Steps:

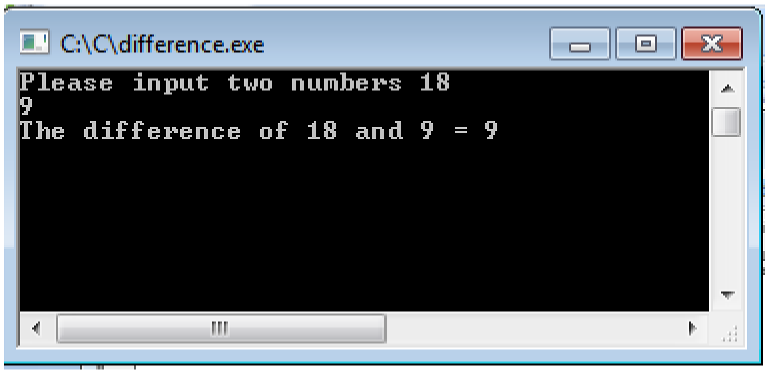
When user executes the program:

1. The control enters the main ( ) function's body.
2. clrscr( ) clears the previous data from the output screen.
3. Function call statement throws the control to "display( )".
4. "printf" command prints the message on the screen (as shown in image above).
5. Control is thrown back to the main ( ) function. Since there are no more statements in main ( ) function the output screen stays until user presses any key from keyboard.

Example - 2:- Write a program that contains a function which takes two numbers from user and displays their difference.

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

void diff ( );
void main( )
{
  clrscr ( );
  diff();
  getch ( );
}

void diff ( )
{
    int x;
    int y;
    printf( "Please input two numbers " );
    scanf( "%d", &x );
    scanf( "%d", &y );
    printf( "The difference of %d and %d = %d", x,y, x-y);
}
```

Output:



```
Please input two numbers 18
9
The difference of 18 and 9 = 9
```

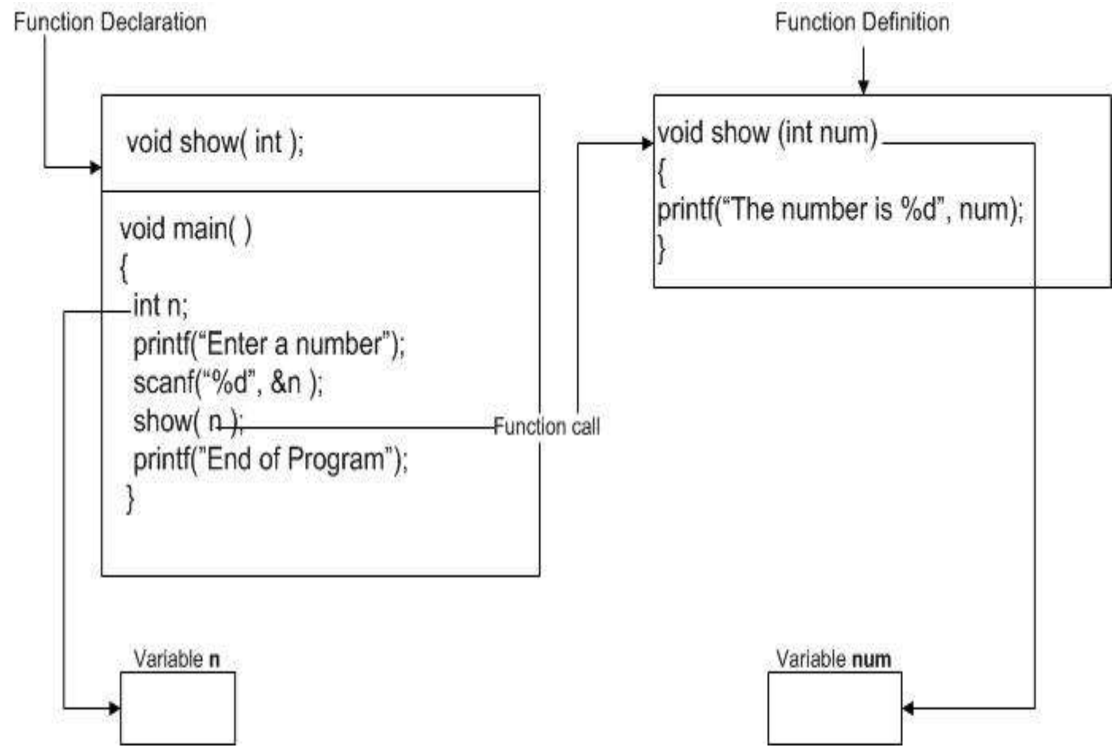
Passing Parameters to Functions:

The values passed to the function at the time of function call are called parameters. Parameters are written in the parenthesis. If there are more than one parameters, these are separated by commas. Both variables and constants can be passed to function as parameters. The sequence and type of parameters in function call must be similar to the sequence and type of parameters in function declaration.

- Parameters in function declaration or function header are called **formal parameters**.
- Parameters in function call are called **actual parameters**.

Passing Parameters by Value:

In pass by value the value of actual parameter is copied to formal parameters. If the function makes any changes in formal parameters, it does not affect the values of actual parameter.



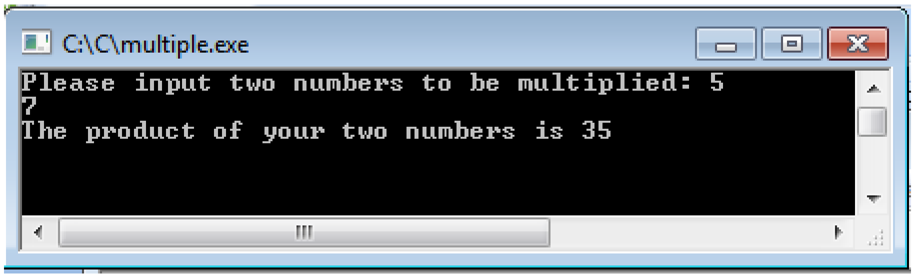
```
void show( int );

void main( )
{
 int n;
 printf("Enter a number");
 scanf("%d", &n );
 show( n );
 printf("End of Program");
}
```

```
void show (int num)
{
printf("The number is %d", num);
}
```

Example – 3:- Write a program that contains a function which takes two integer types of arguments and displays the multiple of both integers.

```
#include <stdio.h>
int mult ( int x, int y );
int main()
{
  int x;
  int y;
  printf( "Please input two numbers to be multiplied: " );
  scanf( "%d", &x );
  scanf( "%d", &y );
  mult(x,y);
  getch ();
}

void mult (int x, int y)
{
    printf( "The product of your two numbers is %d\n", x*y);

}
```

Output:



C:\C\multiple.exe

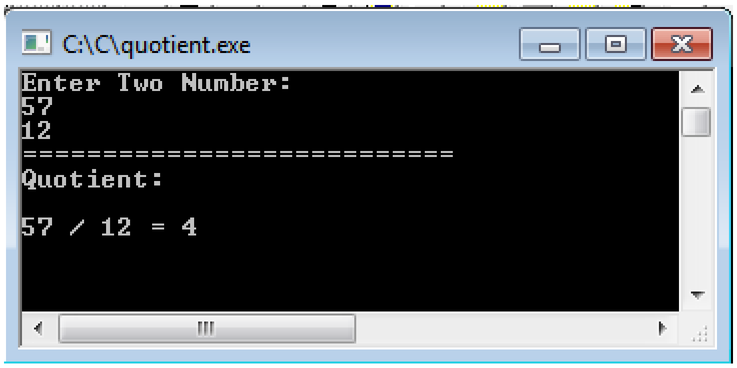
```
Please input two numbers to be multiplied: 5
7
The product of your two numbers is 35
```

Example - 4:- Write a program that contains a function which takes two integer types of arguments and display their quotient.

```
# include <stdio.h>

# include<conio.h>

void add(int , int); //function declaration or prototype

void main()

  {

  int num1,num2,result;

  printf("Enter Two Number:\n");

  scanf("%d %d",&num1,&num2);

  add(num1,num2); //function call

  getch();

  }

void add(int x,int y) //function definition

{
```

```
printf("==========================\n");

printf("Quotient: \n\n");

printf("%d / %d = %d\n", x, y, x/y);

}
```

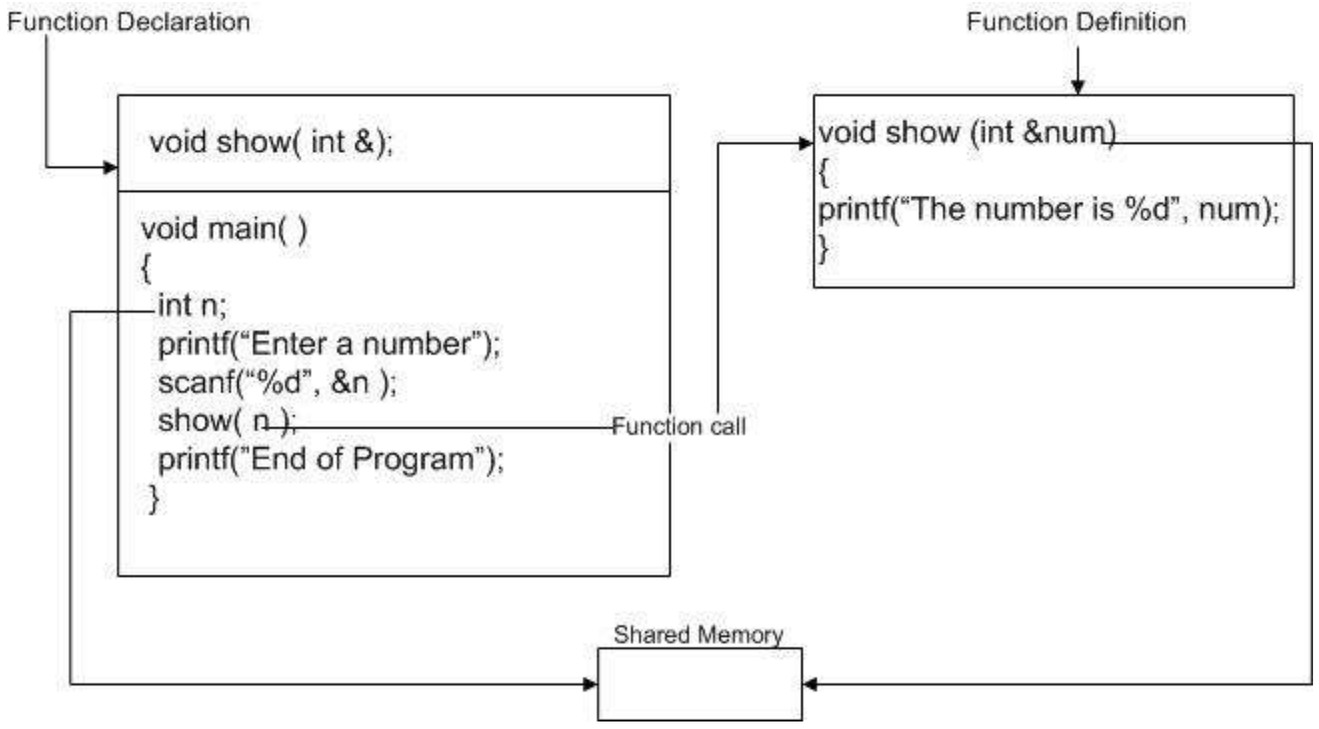
Output:



```
Enter Two Number:
57
12
==========================
Quotient:

57 / 12 = 4
```

Passing Parameters by Reference:

In pass by reference method, the memory address of actual parameter is passed to the function. The formal parameter becomes second name for actual parameter. Formal parameter does not occupy separate memory, it means both actual and formal parameters refer to the same memory location. If the function makes any changes in formal parameters, the change is also reflected in actual parameter.



```
void show( int &);
```

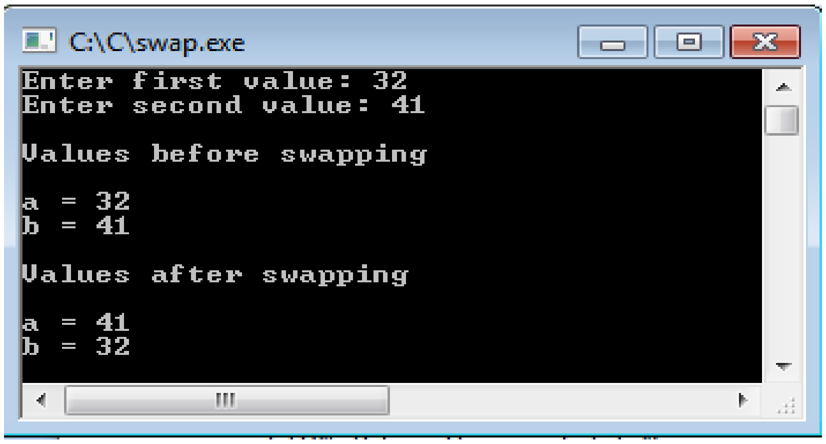
```
void main( )
{
 int n;
 printf("Enter a number");
 scanf("%d", &n );
 show( n );
 printf("End of Program");
}
```

```
void show (int &num)
{
printf("The number is %d", num);
}
```

Example - 5:- Write a program that inputs two integers in main ( ) and passes the integers to swap( ) by reference. The swap( ) swaps the values. The main( ) should display the values before and after swapping.

```
# include <stdio.h>
# include<conio.h>
void swap ( int &x, int &y );
void main()
{
  clrscr();
  int a,b;
  printf("Enter first value: ");
  scanf("%d", &a);
  printf("Enter second value: ");
  scanf("%d", &b);
  printf("\nValues before swapping\n\n");
  printf("a = %d\n",a);
```

```
 printf("b = %d\n",b);
swap(a,b);
 printf("\nValues after swapping\n\n");
 printf("a = %d\n",a);
 printf("b = %d\n",b);
 getch ();
}
void swap( int &x, int &y)
{
   int t;
   t=x;
   x=y;
   y=t;
}
```



C:\C\swap.exe

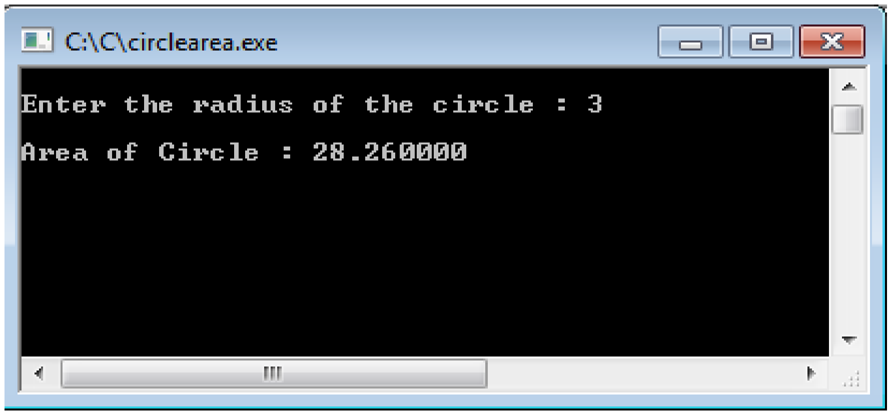
```
Enter first value: 32
Enter second value: 41

Values before swapping

a = 32
b = 41

Values after swapping

a = 41
b = 32
```

Returning Value from Functions:

A function can return a single value. The return type in function declaration indicates the type of value returned by a function. The keyword return is used to return the value back to the calling function. When the return statement is executed in a function, the control moves back to the calling function along with the returned value.

Example - 6:- Write a program that contains calculate_area( ) which takes radius of circle as argument and returns area of circle to main ( );

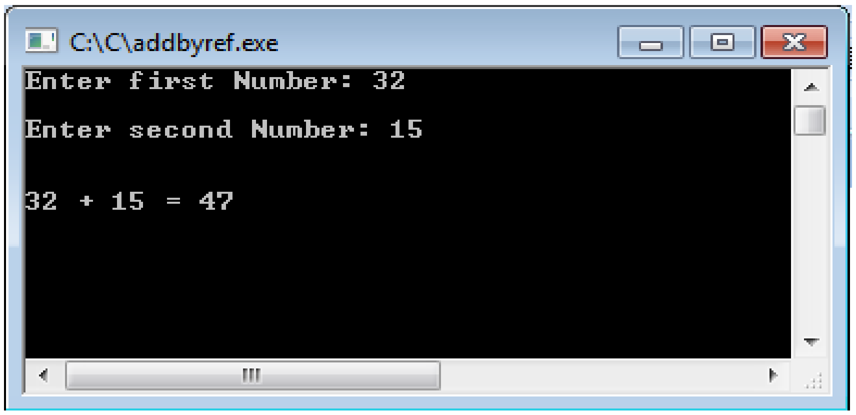
```
# include <stdio.h>

# include<conio.h>

float calculate_area(int);

void main()

{  clrscr();

   int radius;

   float area;

   printf("\nEnter the radius of the circle : ");

   scanf("%d",&radius);

   area = calculate_area(radius);

   printf("\nArea of Circle : %f ",area);

   getch();  }

float calculate_area(int r)

{

   float areaOfCircle;

   areaOfCircle = 3.14 * r * r;

   return areaOfCircle;

}
```



```
Enter the radius of the circle : 3
Area of Circle : 28.260000
```

Example - 7:- Write a program that contains a function which takes two integer types of arguments and display sum in main function without returning any value.

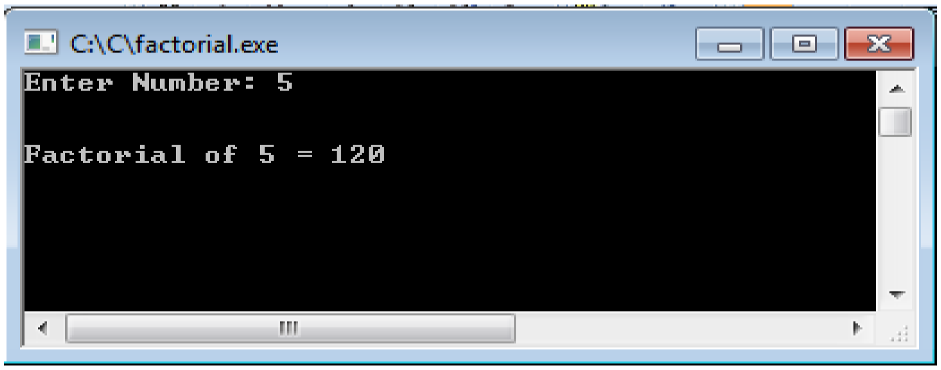
```
# include <stdio.h>
# include<conio.h>
void add(int , int, int &);

void main()
  {
  clrscr();
  int number1,number2,sum;
  printf("Enter first Number: ");
  scanf("%d", &number1);
  printf("\nEnter second Number: ");
  scanf("%d", &number2);
  add(number1,number2,sum);
  printf("\n\n%d + %d = %d", number1, number2, sum);
  getche();
  }
void add(int x,int y,int &r)
{
r=x+y;
}
```



```
Enter first Number: 32

Enter second Number: 15

32 + 15 = 47
```

Example - 8:- Write a program that contains a function which takes one integer type of argument and calculate the factorial of the number.

```
# include <stdio.h>
# include<conio.h>
long factorial(int);

void main()
  {
  clrscr();
  int num; long fact;
  printf("Enter Number: ");
  scanf("%d",&num);
  fact=factorial(num);
  printf("\n\nFactorial of %d = %d",num,fact);
  getch();

  }
long factorial(int x)
{
  long temp=1;
        for(int i=x; i>0; i--)

        temp=temp*i;
        return temp;

}
```



```
Enter Number: 5

Factorial of 5 = 120
```

10.5 Exercises for lab

Exercises-1) Write a program that contains a function which takes one integer type of argument and displays whether it is even or odd

Exercises -2) Write a program that contains a function which takes one integer type of argument, calculate the factorial of the number and display result in main function without returning any value from function.

10.6 Home Work

1) Write a program that contains a function which takes one integer type of argument and confirm that number is prime or not.

2) Solve question one but confirmation about prime number should be in main function. Your function prototype should be look like this:

int prime(int);   return 1 if number is prime else return 2.

3) Solve question one but confirmation about prime number should be in main function and your main function should not return any value.

void prime(int, int&);

Lab # 11

Recursion in C

11.1 Objectives:

This lab reviews even more basic C programming with an emphasis on recursive algorithms. You will:

1. Learn basic idea of recursion in C program.
2. Learn how to make a recursive function
3. Use programs to aid your understanding of recursion.
4. To know about importance, working & use of recursive functions.

11.2 Scope:

The student should know the following at the end of this lab:

1. Understand the idea of recursion
2. Trace a recursive function
3. The student should know how to write a recursive function
4. Should be able to convert any program written using loops into a recursive one

11.3 Useful Concepts:

What is Recursion or recursive function?

A recursive function is one that solves its task

by calling itself on smaller pieces of data.

• Similar to recurrence function in mathematics.

• Like iteration -- can be used interchangeably;

sometimes recursion results in a simpler solution.

Mathematical Definition:

RunningSum(1) = 1

RunningSum(n) =

n + RunningSum(n-1)

Recursive Function:

```
int RunningSum(int n) {
if (n == 1)
return 1;
else
return n + RunningSum(n-1);
}
```

A recursive function has two parts:

1. Base case (stopping condition)
2. General case (recursive step: which must always get closer to base case from one invocation to another.)

A recursive solution always need a stopping condition to prevent an infinite loop and we achieve it by using base case.

e.g

```
 power(int x, int y)
    {
         if(y==0) return 1;  /* base case */
         else
         return (x*power(x,y-1));  /*general case */
    }
```

Any problem which we can solve using recursion, we can also solve that problem using iteration.

Generally, a recursive solution is slightly less efficient, in terms of computer time, than an iterative one because of the overhead for the extra function calls. In many instances, however, recursion enables us to specify a natural, simple solution to a problem that otherwise would be difficult to solve. For this reason, recursion is an important and powerful tool in problem solving and programming. Recursion is used widely in solving problems. That is not numeric, such as proving mathematical theorems, writing compilers, and in searching and sorting algorithms.

In iterative method we use for, while, do-while for achieving iteration for problem solving. In recursive method of problem solving we replace for, while, do-while statement by if statement that selects between the recursive case and the base case (i.e terminating condition).

Detailed Example: Fibonacci Numbers

Mathematical Definition:

$$f(n) = f(n-1) + f(n-2)$$
$$f(1) = 1$$
$$f(0) = 1$$

In other words, the n-th Fibonacci number is the sum of the previous two Fibonacci numbers.

Fibonacci: C Code:

```
int Fibonacci(int n)
{
if ((n == 0) || (n == 1))
return 1;
else
return Fibonacci(n-1) + Fibonacci(n-2);
}
```

Tracing the Function Calls:

If we are debugging this program, we might want to trace all the calls of Fibonacci.

• Note: A trace will also contain the arguments passed into the function.

For Fibonacci(3), a trace looks like:

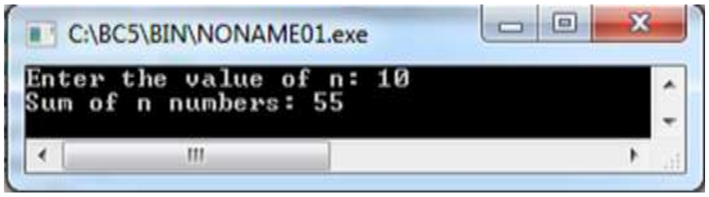
```
Fibonacci(3)
Fibonacci(2)
Fibonacci(1)
Fibonacci(0)
Fibonacci(1)
```

11.4 Examples

Example – 1:- Write a program that takes value from user and Sum all number using recursion in c

```
#include<stdio.h>
using namespace std;
int main(){
int n,sum;
printf("Enter the value of n: ");
scanf("%d",&n);
sum = getSum(n);
printf("Sum of n numbers: %d",sum);
return 0;
}
int getSum(n){
static int sum=0;
if(n>0){
sum = sum + n;
getSum(n-1);
}
return sum;
}
```

Output:



```
Enter the value of n: 10
Sum of n numbers: 55
```

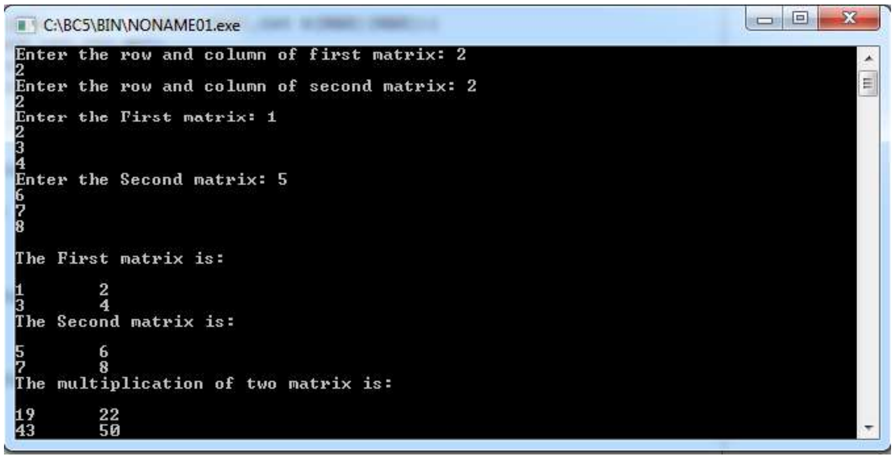
Example – 2:- Write a program that multiple two Matrixes using recursion in c

```
#include<stdio.h>
#define MAX 10
using namespace std;
void multiplyMatrix(int [MAX][MAX],int [MAX][MAX]);
int m,n,o,p;
int c[MAX][MAX];
int main(){
int a[MAX][MAX],b[MAX][MAX],i,j,k;
printf("Enter the row and column of first matrix: ");
scanf("%d %d",&m,&n);
printf("Enter the row and column of second matrix: ");
scanf("%d %d",&o,&p);
if(n!=o){
```

```
printf("Matrix multiplication is not possible");
printf("\nColumn of first matrix must be same as row of second matrix");
}
else{
printf("Enter the First matrix: ");
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);

printf("Enter the Second matrix: ");
for(i=0;i<o;i++)
for(j=0;j<p;j++)
scanf("%d",&b[i][j]);
printf("\nThe First matrix is: \n");
for(i=0;i<m;i++){
printf("\n");
for(j=0;j<n;j++){
printf("%d\t",a[i][j]);
}
}
printf("\nThe Second matrix is: \n");
for(i=0;i<o;i++){
printf("\n");
for(j=0;j<p;j++){
printf("%d\t",b[i][j]);
}
}
multiplyMatrix(a,b);
}
printf("\nThe multiplication of two matrix is: \n");
for(i=0;i<m;i++){
printf("\n");
```

```
for(j=0;j<p;j++){
printf("%d\t",c[i][j]);
}
}
return 0;
}
void multiplyMatrix(int a[MAX][MAX],int b[MAX][MAX]){
static int sum,i=0,j=0,k=0;
if(i<m){ //row of first matrix
if(j<p){ //column of second matrix
if(k<n){
sum=sum+a[i][k]*b[k][j];
k++;
multiplyMatrix(a,b);
}
c[i][j]=sum;
sum=0;
k=0;
j++;
multiplyMatrix(a,b);
}
j=0;
i++;
multiplyMatrix(a,b);
}
}
```

Output:

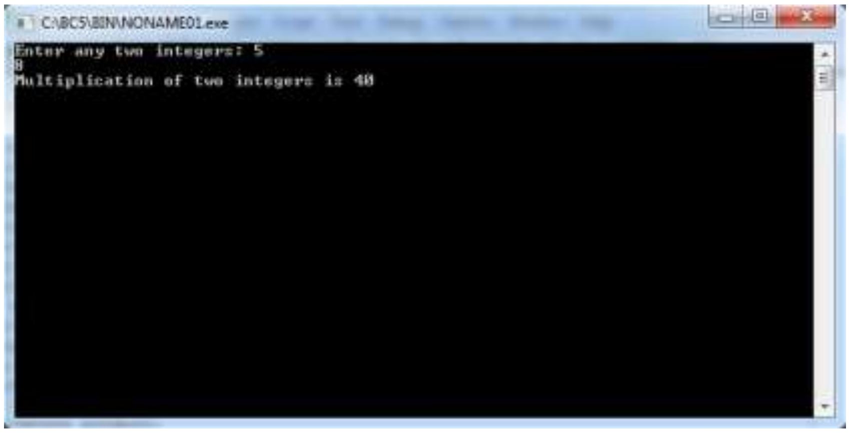


```
Enter the row and column of first matrix: 2
2
Enter the row and column of second matrix: 2
2
Enter the First matrix: 1
2
3
4

Enter the Second matrix: 5
6
7
8

The First matrix is:

1       2
3       4
The Second matrix is:

5       6
7       8
The multiplication of two matrix is:

19      22
43      50
```

Example – 3:- Write a program that Multiply two numbers in C

```
#include<stdio.h>
using namespace std;
int multiply(int,int);
int main(){
int a,b,product;
printf("Enter any two integers: ");
scanf("%d%d",&a,&b);
product = multiply(a,b);
printf("Multiplication of two integers is %d",product);
return 0;
}
int multiply(int a,int b){
static int product=0,i=0;
if(i < a){
product = product + b;
i++;
multiply(a,b);
}
return product;
}
```

Output:

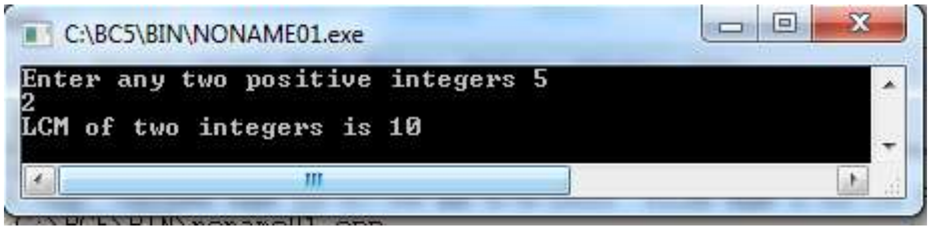


```
Enter any two integers: 5
8
Multiplication of two integers is 40
```

Example – 4:- Write a program that find LCM using recursion in c.

```
#include<stdio.h>
using namespace std;
int lcm(int,int);
int main(){
int a,b,l;
printf("Enter any two positive integers ");
scanf("%d%d",&a,&b);
if(a>b)
l = lcm(a,b);
else
l = lcm(b,a);
printf("LCM of two integers is %d",l);
return 0;
}
int lcm(int a,int b){
static int temp = 1;
if(temp % b == 0 && temp % a == 0)
return temp;
temp++;
lcm(a,b);
return temp;
}
```

Output:

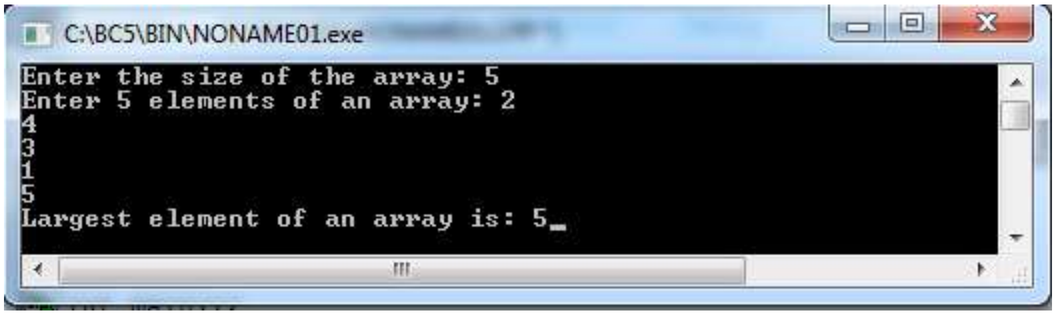


```
Enter any two positive integers 5
2
LCM of two integers is 10
```

Example – 5:- Write a program that find largest element in an array using recursion in c.

```
#include<stdio.h>
#define MAX 100
using namespace std;
int getMaxElement(int []);
int size;
int main(){
int arr[MAX],max,i;
printf("Enter the size of the array: ");
scanf("%d",&size);
printf("Enter %d elements of an array: ", size);
for(i=0;i<size;i++)
scanf("%d",&arr[i]);
max=getMaxElement(arr);
printf("Largest element of an array is: %d",max);
return 0;
}
int getMaxElement(int arr[]){
static int i=0,max =-9999;
if(i < size){
if(max<arr[i])
max=arr[i];
i++;
getMaxElement(arr);
}
return max;
}
```

Output:



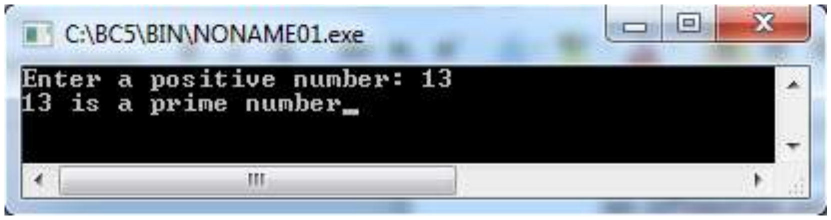
```
Enter the size of the array: 5
Enter 5 elements of an array: 2
4
3
1
5
Largest element of an array is: 5_
```

Example – 6:-Write a program that identifies the number is prime or not using recursion in c.

```
#include<stdio.h>
using namespace std;
int isPrime(int,int);
int main(){
int num,prime;
printf("Enter a positive number: ");
scanf("%d",&num);
prime = isPrime(num,num/2);
if(prime==1)
printf("%d is a prime number",num);
else
printf("%d is not a prime number",num);
return 0;
}
int isPrime(int num,int i){
if(i==1){
return                                                                    1;
}else{
```

```
if(num%i==0)
return 0;
else
isPrime(num,i-1);
}
}
```

Output:

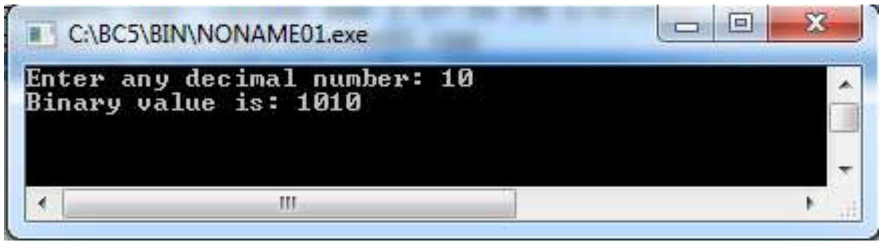


```
Enter a positive number: 13
13 is a prime number_
```

Example – 7:- Write a program that convert decimal to binary using recursion in c.

```
#include<stdio.h>
using namespace std;
long toBinary(int);
int main(){
long binaryNo;
int decimalNo;
printf("Enter any decimal number: ");
scanf("%d",&decimalNo);
binaryNo = toBinary(decimalNo);
printf("Binary value is: %ld",binaryNo);
return 0;
}
long toBinary(int decimalNo){
static long binaryNo,remainder,factor = 1;
if(decimalNo != 0){
remainder = decimalNo % 2;
binaryNo = binaryNo + remainder * factor;
factor = factor * 10;
toBinary(decimalNo / 2);
}
return binaryNo;
}
```

Output:



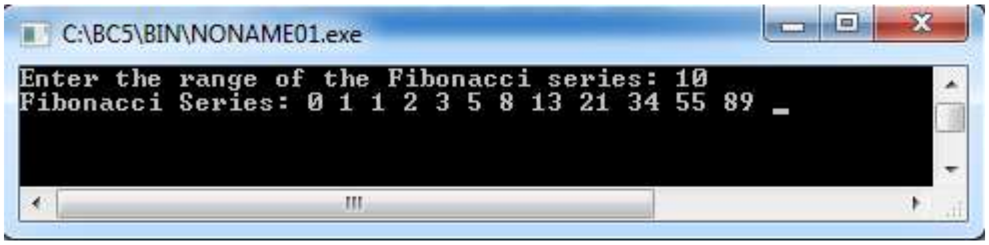
```
Enter any decimal number: 10
Binary value is: 1010
```

Example – 8:- Write a program for Fibonacci series using recursion in c.

C program for fibonacci series using recursion

```
#include<stdio.h>
using namespace std;
Void  printFibonacci(int);
int main(){
int k,n;
long int i=0,j=1,f;
printf("Enter the range of the Fibonacci series: ");
scanf("%d",&n);
printf("Fibonacci Series: ");
printf("%d %d ",0,1);
printFibonacci(n);
return 0;
}
void printFibonacci(int n){
static long int first=0,second=1,sum;
if(n>0){
sum = first + second;
first = second;
second = sum;
printf("%ld ",sum);
printFibonacci(n-1);
}
}
```

Output:

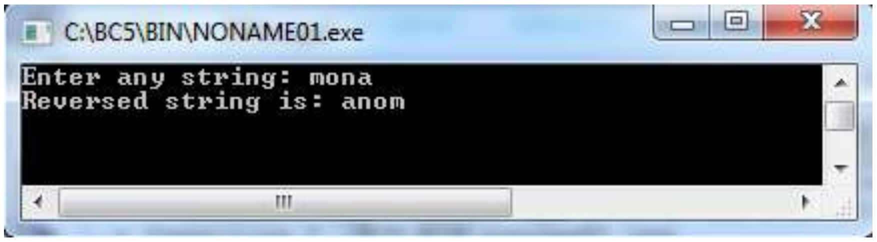


```
Enter the range of the Fibonacci series: 10
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34 55 89 _
```

Example – 9:- Write a program that takes string from user and print it in reverse using recursion in c.

```
#include<stdio.h>
#define MAX 100
using namespace std;
char* getReverse(char[]);
int main(){
char str[MAX],*rev;
printf("Enter any string: ");
scanf("%s",str);
rev = getReverse(str);
printf("Reversed string is: %s",rev);
return 0;
}
char* getReverse(char str[]){
static int i=0;
static char rev[MAX];
if(*str){
getReverse(str+1);
rev[i++] = *str;
}
return rev;
}
```

Output:

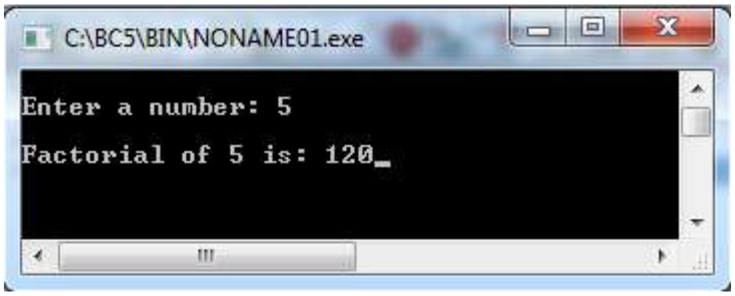


```
Enter any string: mona
Reversed string is: anom
```

Example – 10:- Write a program that finds a factorial using recursion in c.

```
#include<stdio.h>
using namespace std;
int fact(int);
int main(){
int num,f;
printf("\nEnter a number: ");
scanf("%d",&num);
f=fact(num);
printf("\nFactorial of %d is: %d",num,f);
return 0;
}
int fact(int n){
if(n==1)
return 1;
else
return(n*fact(n-1));
}
```

Output:



```
Enter a number: 5
Factorial of 5 is: 120_
```

11.5 Exercises for lab

Exercise -1) Find gcd of a number using recursion in c program

Exercise -2) Find sum of digits of a number using recursion using cprogram

11.6 Home Work

1. Find power of a number using recursion using c program
2. Binary search through recurssion using c program
3. Write a program for palindrome using recursion
4. Find power of a number using recursion using c program
5. Reverse a number using recursion in c program
6. Write a recursive C++ function to calculate and return the sum of the following series then call your function from the main.
7. Sum=1+2+3+….+N
8. Write a recursive C++ function to calculate and return:
9. Sum=1! +2! + 3! + … + n!

Lab # 12

Pointers

12.1 Objective:

a. Declaring pointers.

b. Storing addresses of different data type variables.

12.2 Scope:

The student should know the following at the end of this lab:

1. How to declare pointer variables.
2. How to assign address of memory location to the pointer variable.
3. How to access values pointed by pointers.

12.3 Useful Concept

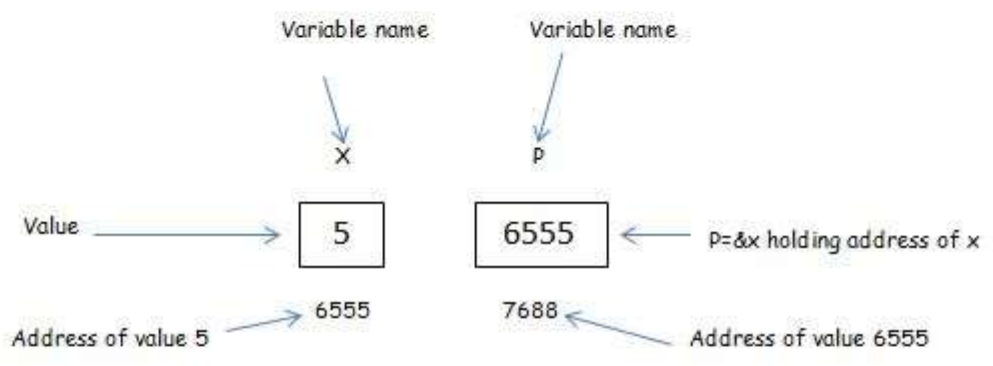
- Pointers are symbolic representation of addresses.
- and can be used to access and manipulate data stored at the address to which it is pointing.

Declaring pointers:

- Just like other variables, pointers have to be declared before their usage.

Syntax:

```
int x=5;
int *p;
p=&x;
```



- Data type of variable and pointer must be same like in this case int x and int *p.

- The * is called indirection operator.
- The & or address operator, is a unary operator that returns the address of its operand.

How many levels of pointers can you have?

```
int i = 0;
int *ip01 = & i;
int **ip02 = & ip01;
int ***ip03 = & ip02;
int ****ip04 = & ip03;
int *****ip05 = & ip04;
int ******ip06 = & ip05;
int *******ip07 = & ip06;
int ********ip08 = & ip07;
int *********ip09 = & ip08;
```

there is no limit , but just use that that much levels which u can understand easily and programs programs does not become confusing.

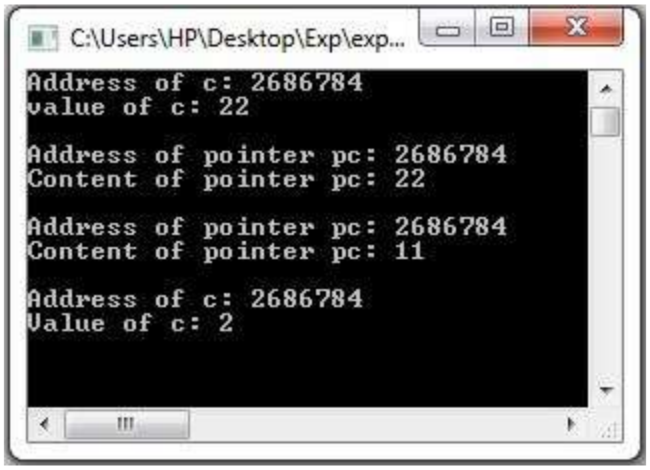
12.4 Examples

Example -1:- // showing pointer application

```
#include <stdio.h>
int main(){
  int* pc;
  int c;
  c=22;
  printf("Address of c:%d\n",&c);
  printf("Value of c:%d\n\n",c);
  pc=&c;
  printf("Address of pointer pc:%d\n",pc);
  printf("Content of pointer pc:%d\n\n",*pc);
  c=11;
  printf("Address of pointer pc:%d\n",pc);
  printf("Content of pointer pc:%d\n\n",*pc);
  *pc=2;
  printf("Address of c:%d\n",&c);
```

```
   printf("Value of c:%d\n\n",c);
   return 0;
}
```

Output:-

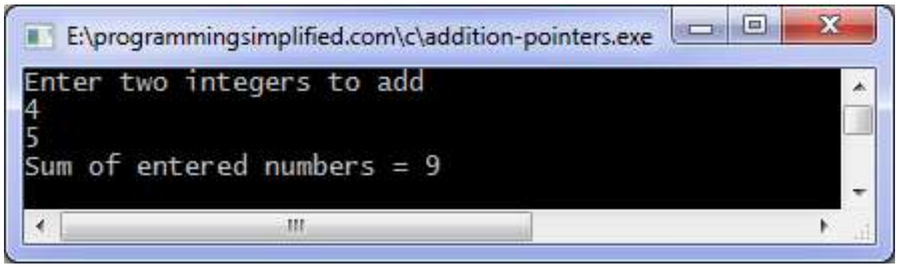


```
Address of c: 2686784
value of c: 22

Address of pointer pc: 2686784
Content of pointer pc: 22

Address of pointer pc: 2686784
Content of pointer pc: 11

Address of c: 2686784
Value of c: 2
```

Example -2:- Find a cube of a variable with a pointer argument

```
#include <stdio.h>
 int main()
{
   int first, second, *p, *q, sum;

   printf("Enter two integers to add\n");
   scanf("%d%d", &first, &second);

   p = &first;
   q = &second;

   sum = *p + *q;

   printf("Sum of entered numbers = %d\n",sum);

   return 0;
}
```

Output:-

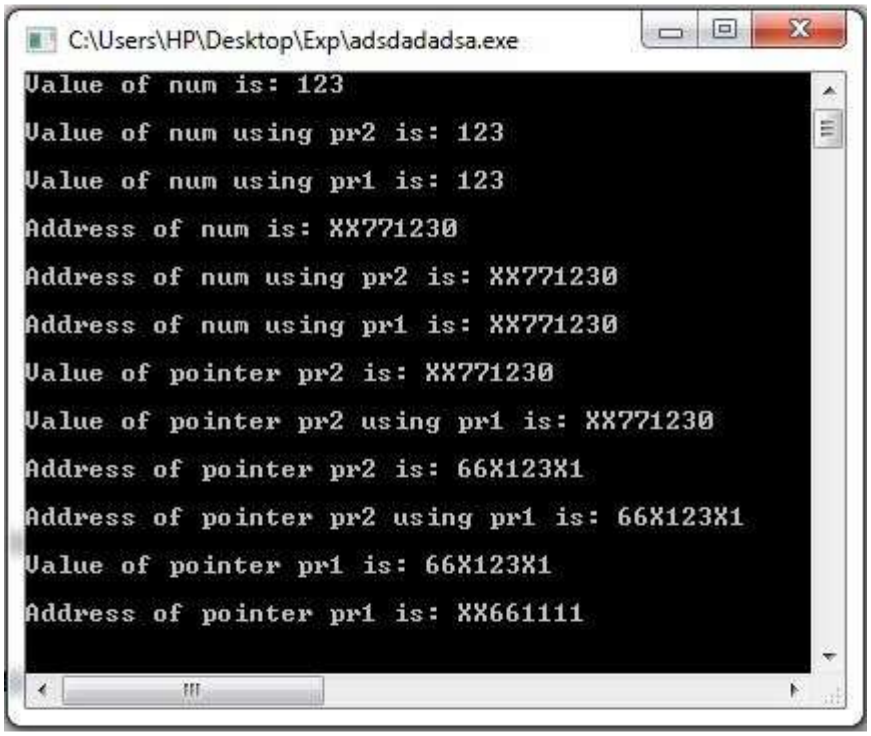


```
Enter two integers to add
4
5
Sum of entered numbers = 9
```

Example -3:-

```
 #include <stdio.h>
int main()
{
    int num=123;

    /*Pointer for num*/
    int *pr2;

    /*Double pointer for pr2*/
    int **pr1;

    /* I'm reading the address of variable num and
     * storing it in pointer pr2*/
    pr2 = &num;

    /* storing the address of pointer pr2 into another pointer pr1*/
    pr1 = &pr2;

    /* Possible ways to find value of variable num*/
    printf("\n Value of num is: %d", num);
    printf("\n Value of num using pr2 is: %d", *pr2);
    printf("\n Value of num using pr1 is: %d", **pr1);

    /*Possible ways to find address of num*/
    printf("\n Address of num is: %u", &num);
    printf("\n Address of num using pr2 is: %u", pr2);
    printf("\n Address of num using pr1 is: %u", *pr1);

    /*Find value of pointer*/
    printf("\n Value of Pointer pr2 is: %u", pr2);
    printf("\n Value of Pointer pr2 using pr1 is: %u", *pr1);

    /*Ways to find address of pointer*/
    printf("\n Address of Pointer pr2 is:%u",&pr2);
    printf("\n Address of Pointer pr2 using pr1 is:%u",*pr1);

    /*Double pointer value and address*/
    printf("\n Value of Pointer pr1 is:%u",pr1);
    printf("\n Address of Pointer pr1 is:%u",&pr1);

    return 0;
}
```

Output:-



C:\Users\HP\Desktop\Exp\adsdadadsa.exe

```
Value of num is: 123
Value of num using pr2 is: 123
Value of num using pr1 is: 123
Address of num is: XX771230
Address of num using pr2 is: XX771230
Address of num using pr1 is: XX771230
Value of pointer pr2 is: XX771230
Value of pointer pr2 using pr1 is: XX771230
Address of pointer pr2 is: 66X123X1
Address of pointer pr2 using pr1 is: 66X123X1
Value of pointer pr1 is: 66X123X1
Address of pointer pr1 is: XX661111
```

12.5 Exercises for lab

- Exercise 1) A program to declare 5 pointers and print its addresses.
- Exercise 2) A program to Add address of two pointers.
- Exercise 3) A program to point a pointer upto 5 levels and print them.
- Exercise 4) A program to implement basic mathematics (Add, Subtract, Multiply) by using functions and pointers.
- Exercise 5) A program to declare a pointer of 1D Array and print its addresses.

12.6 Home Work

- A program to take information of Student (name, age, department, cgpa) and print its addresses along with its values.
- A program to sort a 2D array (using pointers).
- A program to declare a pointer of 2D array and check its address is even or odd(using functions).

Lab # 13

Strings

13.1 Objective:

Learn how to declare, initialize and use one-dimensional character arrays (string).

13.2 Scope:

The student should know the following:

1. What is string?
2. How to Read and Print Strings.
3. Some important built-in string functions. How to use them?

13.3 Useful Concepts:

What is String?

A string is any sequence of characters *enclosed* in double quotes. There is no separate data type for strings as integer, float or double. The string data type can be considered as a char array. The difference is that a character variable can hold only *one character* but a string can have *more than one character* in a character array. For example, a string called name with 9 characters can be declared as:

```
char    name [9] = "I like C" ;
```

Check for the double quotes, the char array size 9. Here the name can be considered as one string or as an array of characters. That is if you refer name it is "I like C" and if you refer name[0] it is 'I', name[1] is ' ' (a *blank*), name[2] is 'l', name[3] is 'i', and so on. The last character name[8] is '\0' which indicates a NULL character which is not displayed but is stored as last character of the string to indicate its end.

<u>**Note**</u>: Strings are stored as arrays of characters and have a special '\0' termination character called NULL appended (attached) to them to signify the end of the string.

Note that if the number of characters including the **'\0'** is more than the size of the array the results will be unpredictable. However, if the size of the array is more than the number of characters the extra spaces after the last **'\0'** character are kept blank and not referred because the string ends at **'\0'**. So, always make sure that the size of the array is sufficient for the string. For example, the above declaration would be wrong if we write

```
char    name [8] = "I like C" ;
```

The size can be ignored also. In that case the size is considered as that of the number of characters specified in the declaration.

```
char    name [ ] = "I like C" ;
```

String Input/Output:

The easiest way to input strings is by using the C library function gets (means get string). The gets( ) function reads a string of characters entered at the keyboard until you strike the enter key (carriage return). The carriage return does not become part of the string; instead a null terminator '\0' is placed at the end.

For example, in the following program fragment

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

and if the user enters

```
Rafiqul Zaman Khan
```

and presses the enter key the string variable str has the characters "Rafiqul Zaman Khan" with '\0' appended at the end but is not displayed.

Similarly, there is an output function puts ( ) which prints or displays the value of the string variable. Unlike the cout, which stays on the same line after printing puts automatically advances the output to the next line after displaying it.

```
puts (str) ;
```

Example:

```
char    name [81] = {"Rafiqul Zaman Khan"} ;
gets (name) ;    /* reads the name */
puts (name) ;    /* prints the name */
```

Built-in String Functions:

Large collections of string processing functions are provided in C through **string.h** file. So, include the **string.h** file in the programs to use these functions. To use the string functions make sure that the size of the array is sufficient so that the strings are terminated with the **'\0'** character or the functions will not work properly.

strcat (string1, string2) :

The **strcat** function concatenates or joins the **string1** and **string2**. A copy of **string2** is put at the end of the **string1**. Make sure that **string1** size is long enough to hold the resulting string (string1 + string2).

Example:

```
char    string1[ 81] = "abc",  string2 [ ] = "def" ;
strcat ( string1, string2) ;
puts ( string1 ) ;                    /* outputs "abcdef" which is stored in string1 */
```

strcpy (string1, string2) :

The **strcpy** function copies **string2** into **string1**. Again make sure that **string1** size is long enough to hold the resulting string.

Example: char string1 [81] , string2 [ ] = "memory" ;

strcpy (string1, string2) ;

puts (string1) ; /* outputs "memory" copied into string1 */

strcmp (string1, string2) :

The **strcmp** function compares the **string1** to **string2** and returns an integer value to show the status of the comparison. A value of **0** indicates that the two strings are identical. A value of **less than 0** shows that **string1** is lexicographically (according to alphabetic ordering) less than **string2**. A value of **greater than 0** shows that string1 is lexicographically (according to alphabetic ordering) greater than **string2**.

See example 6

strlen (string1) :

The **strlen** function returns an integer equal to the **length** of the stored string including blanks, not including the termination character.

strchr (string, ch) :

The **strchr** function searches **string** for the first occurrence of **ch**. This function only tells whether the string contains **ch** or not and it will not tell the position of the **ch** in the string if found.

13.4 Examples:

Example1:- Write a program that prints a string. String can be printed by using various functions such as printf, puts.

```
#include <stdio.h>

int main()
{
   char array[20] = "Hello World";

   printf("%s\n",array);

   return 0;
}
```

Example2:- Write a program that input a string we use scanf function.

```
#include <stdio.h>

int main()
{
    char array[100];

    printf("Enter a string\n");
    scanf("%s", array);

    printf("You entered the string %s\n",array);
    return 0;
}
```

Example3:- Write a Program thats Input string containing spaces.

```
#include <stdio.h>

int main()
{
   char a[80];

   gets(a);

   printf("%s\n", a);

   return 0;
}
```

//Note that scanf can only input single word strings, to receive strings containing spaces use gets function.

Example4:- Write a Program to Print string using loop.

We print string using for loop by printing individual characters of string.

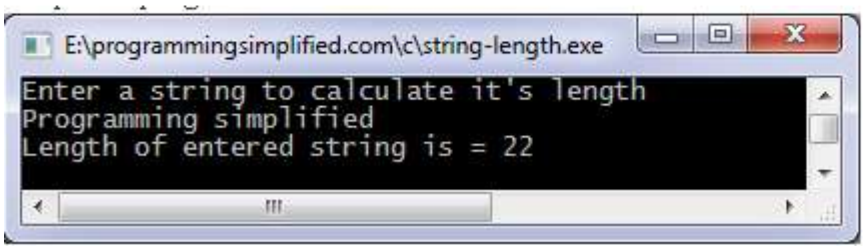
```
#include <stdio.h>
#include <string.h>

int main() {
   char s[100];
   int c, l;

   gets(s);

   l = strlen(s);
```

```
  for (c = 0; c < l; c++)
    printf("%c", s[c]);

  return 0;
}
```

Example5:- Write a program that finds a string length.

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

int main()
{
  char a[100];
  int length;

  printf("Enter a string to calculate it's length\n");
  gets(a);

  length = strlen(a);

  printf("Length of entered string is = %d\n",length);

  return 0;
}
```

Output:-



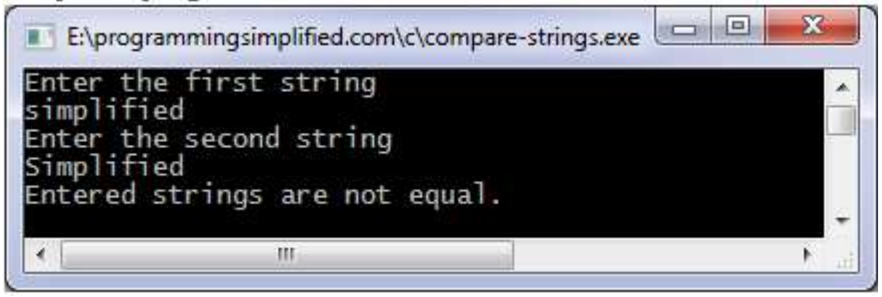
E:\programmingsimplified.com\c\string-length.exe

```
Enter a string to calculate it's length
Programming simplified
Length of entered string is = 22
```

Example6:- Write a program to compare two strings using strcmp.

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

int main()
{
  char a[100], b[100];

  printf("Enter the first string\n");
  gets(a);

  printf("Enter the second string\n");
  gets(b);

  if (strcmp(a,b) == 0)
    printf("Entered strings are equal.\n");
  else
    printf("Entered strings are not equal.\n");

  return 0;
}
```

Output:-

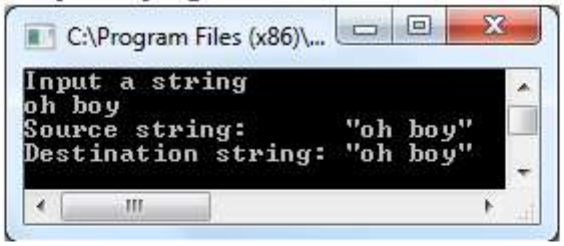


```
Enter the first string
simplified
Enter the second string
Simplified
Entered strings are not equal.
```

Example7:- Write a program to copy a string.

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

int main()
{
  char source[1000], destination[1000];

  printf("Input a string\n");
  gets(source);

  strcpy(destination, source);

  printf("Source string:      \"%s\"\n", source);
  printf("Destination string: \"%s\"\n", destination);

  return 0;
}
```

Output:-

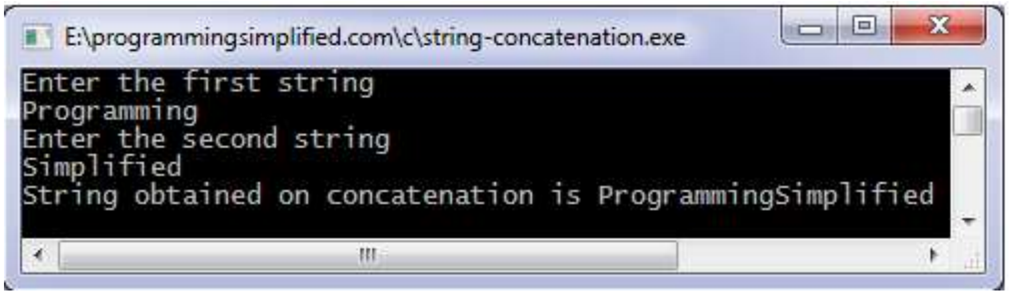


```
C:\Program Files (x86)\...
Input a string
oh boy
Source string:      "oh boy"
Destination string: "oh boy"
```

Example8:- Write a program that concatinate a string.

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

int main()
{
   char a[1000], b[1000];

   printf("Enter the first string\n");
   gets(a);

   printf("Enter the second string\n");
   gets(b);

   strcat(a,b);

   printf("String obtained on concatenation is %s\n",a);

   return 0;
}
```

Output:



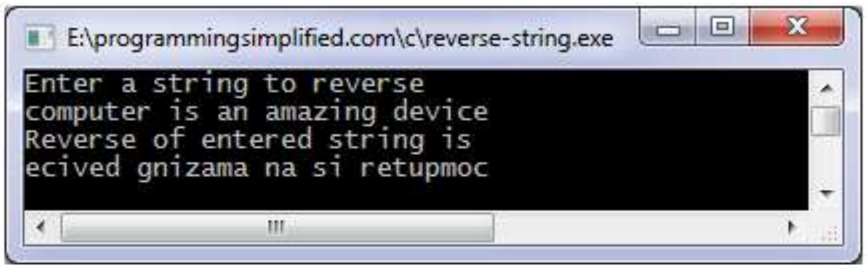
```
E:\programmingsimplified.com\c\string-concatenation.exe
Enter the first string
Programming
Enter the second string
Simplified
String obtained on concatenation is ProgrammingSimplified
```

Example9:- Write a program that prints string in reverse form.

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

int main()
{
   char arr[100];
```

```
  printf("Enter a string to reverse\n");
  gets(arr);

  strrev(arr);

  printf("Reverse of entered string is \n%s\n",arr);

  return 0;
}
```

Output:-



```
Enter a string to reverse
computer is an amazing device
Reverse of entered string is
ecived gnizama na si retupmoc
```

13.5 Exercises for lab

Exercise -1) Write a program that reads a string and print the number of vowels letters.

Exercise -2) Write a Program number that reads a string and print upper case and number of lower case letters.

Exercise -3) Write a program that reads a string and print the string with the first letter capitalized and the remaining in lower case.

13.6 Home Work

1) Write a program that reads a string and print the reverse of that string:
e.g. User Enter string----------This is a cat
Output: - tac a si siht

2) Write a program that reads a string and print the string's words in reverse order:
e.g. User Enter string----------This is a cat
Output: - cat a is This

Hints: - Use two arrays of same size. One is used to take value from user and other is used to store reverse string.

Lab # 14

File handling in C

14.1 Objective:

Often it is not enough to just display the data on the screen. This is because if the data is large, only a limited amount of it can be stored in memory and only a limited amount of it can be displayed on the screen. It would be inappropriate to store this data in memory for one more reason. Memory is volatile and its contents would be lost once the program is terminated. So if we need the same data again it would have to be either entered through the keyboard again or would have to be regenerated programmatically.

Obviously both these operations would be tedious. At such times it becomes necessary to store the data in a manner that can be later retrieved and displayed either in part or in whole. This medium is usually a 'file' on the disk

14.2 Scope:

The student should know the following:

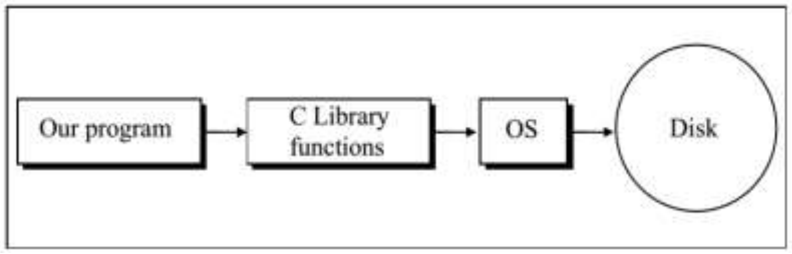
8. To create, read, write and update the files
9. Copy the file, writing into file from consoles etc

14.3 Useful Concepts:

How data is organized:

1. All data stored on the disk is in binary form.
2. How this binary data is stored on the disk varies from one OS to another.

3. However, this does not affect the C programmer since he has to use only the library functions written for the particular OS to be able to perform input/output.
4. It is the compiler vendor's responsibility to correctly implement these library functions by taking the help of OS.
5. Following figure illustrates the concept,



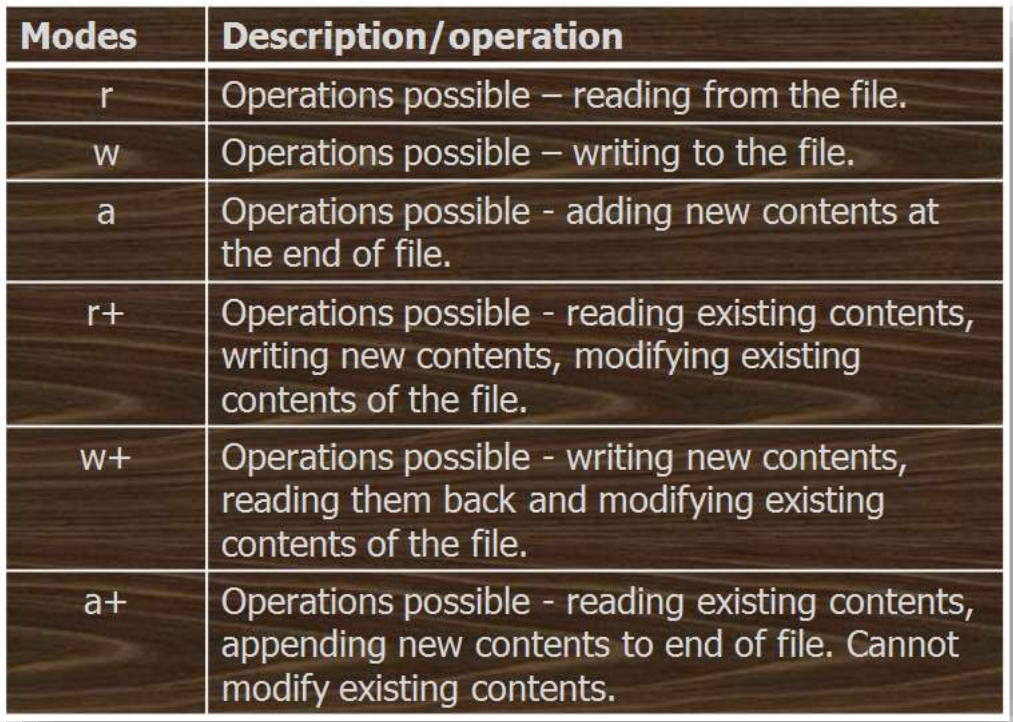
File Operations:

There are different operations that can be carried out on a file. These are:

1. Creation of a new file
2. Opening an existing file
3. Reading from a file
4. Writing to a file
5. Moving to a specific location in a file (seeking)
6. Closing a file

Let us now write a program to read a file and display its contents on the screen. We will first list the program and show what it does, and then dissect it line by line.

File Opening: Modes:

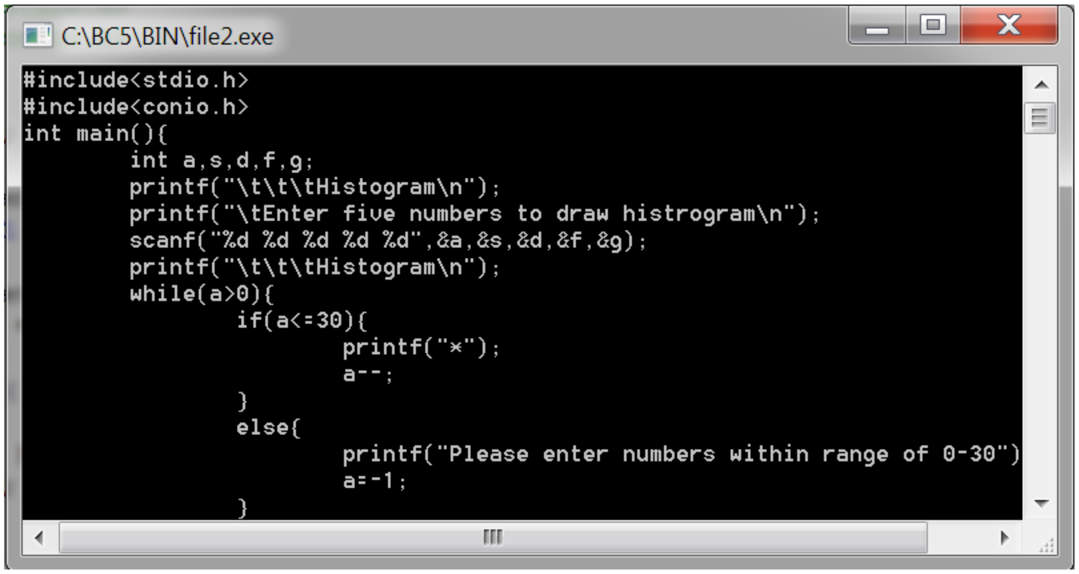


Modes	Description/operation
r	Operations possible – reading from the file.
w	Operations possible – writing to the file.
a	Operations possible - adding new contents at the end of file.
r+	Operations possible - reading existing contents, writing new contents, modifying existing contents of the file.
w+	Operations possible - writing new contents, reading them back and modifying existing contents of the file.
a+	Operations possible - reading existing contents, appending new contents to end of file. Cannot modify existing contents.

Example -1:- Write a program that reads the content of a file.

```
#include<stdio.h>
#include<conio.h>
main( )
{
 FILE *fp;
 char ch;
 fp = fopen ( "printingHistogram.c", "r" );
 while( 1 )
 {
  ch = fgetc ( fp );
  if ( ch = = EOF )
  break;
  printf ( "%c", ch );
 }
 fclose ( fp );
 getche();
}
```

Output:-



```
#include<stdio.h>
#include<conio.h>
int main(){
        int a,s,d,f,g;
        printf("\t\t\tHistogram\n");
        printf("\tEnter five numbers to draw histrogram\n");
        scanf("%d %d %d %d %d",&a,&s,&d,&f,&g);
        printf("\t\t\tHistogram\n");
        while(a>0){
                if(a<=30){
                        printf("*");
                        a--;
                }
                else{
                        printf("Please enter numbers within range of 0-30")
                        a=-1;
                }
```

Line by Line Program Illustration

1. To open the file we have called the function fopen ( ).
2. open( )performs three important tasks when you open the file in "r" mode:
    a. Firstly it searches on the disk the file to be opened.
    b. Then it loads the file from the disk into a place in memory called buffer.
    c. It sets up a character pointer that point to the first character of the buffer.
3. To be able to successfully read from a file information like mode of opening, size of file, place in the file from where the next read operation would be performed, etc. has to be maintained.

14.3.4 Reading from the File:

1. Since all this information is inter-related, all of it is gathered together by fopen() in a structure called FILE.
2. fopen( )returns the address of this structure, which we have collected in the structure pointer called fp. We have declared fp as FILE *fp ;
3. The FILE structure has been defined in the header file "stdio.h"
4. Once the file has been opened for reading using fopen( ), file's contents are brought into buffer (partly or wholly) and a pointer is set up that points to the first character in the buffer. This pointer is one of the elements of the structure to which fp is pointing.
5. To read the file's contents from memory there exists a function called fgetc( ). This has been used in our program as, ch = fgetc (fp) ;

fgetc() function

fgetc( )reads the character from the current pointer position, advances the pointer position so that it now points to the next character, and returns the character that is read, which we collected in the variable ch. Note that once the file has been opened, we no longer refer to the file by its name, but through the file pointer fp.

End of file:-

1. We have used the function fgetc( )within an indefinite while loop.
2. There has to be a way to break out of this while. When shall we break out... the moment we reach the end of file.
3. But what is end of file? A special character, whose ASCII value is 26, signifies end of file.
4. This character is inserted beyond the last character in the file, when it is created.
5. In our program we go on reading each character from the file till end of file is not met. As each character is read we display it on the screen. Once out of the loop, we close the file.

Trouble in opening File:

1. Crux of the matter is that it is important for any program that accesses disk files to check whether a file has been opened successfully before trying to read or write to the file.
2. If the file opening fails due to any of the several reasons mentioned above, the fopen( )function returns a value NULL.

Example -2:- Write a Program that open a file.

```
#include <stdio.h>
main( )
{
 FILE *fp;
 fp = fopen ( "PR1.C", "r" );
 if ( fp == NULL )
 {
  puts ( "cannot open file" );
  exit( );
 }
}
```

Output:

```
Cannot open the file
```

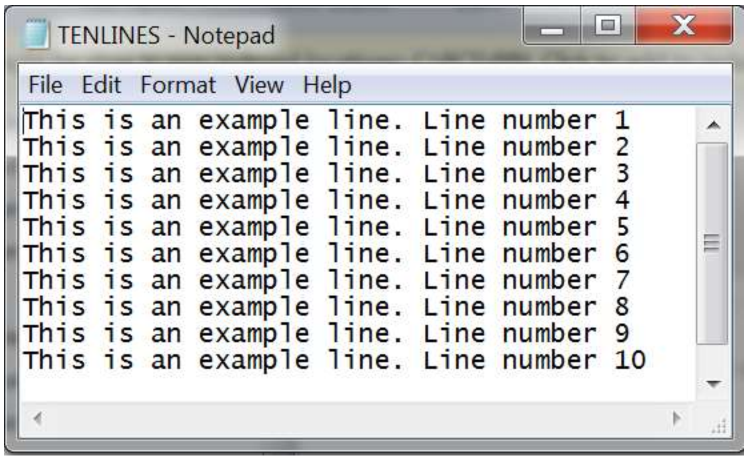
Closing File:

1. When we have finished reading from the file, we need to close it.
2. This is done using the function fclose( )through the statement,
    a. fclose (fp) ;
3. Once we close the file we can no longer read from it using getc( ) unless we reopen the file.
4. Note that to close the file we don't use the filename but the file pointer fp.
5. On closing the file the buffer associated with the file is removed from memory.
6. When we attempt to write characters into this file using fputc( ) the characters would get written to the buffer.
7. When we close this file using fclose( ) three operations would be performed:
8. The characters in the buffer would be written to the file on the disk.
9. At the end of file a character with ASCII value 26 would get written.
10. The buffer would be eliminated from memory

Example -3:- Write a Program to create a file and write some data the file.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
Void main( )
{
    FILE *fp;
    char stuff[25];
    int index;
    fp = fopen("TENLINES.TXT","w"); /* open for writing */
    strcpy(stuff,"This is an example line.");
    for (index = 1; index <= 10; index++)
          fprintf(fp,"%s Line number %d\n", stuff, index);
    fclose(fp); /* close the file before ending program */
}
```

Output:



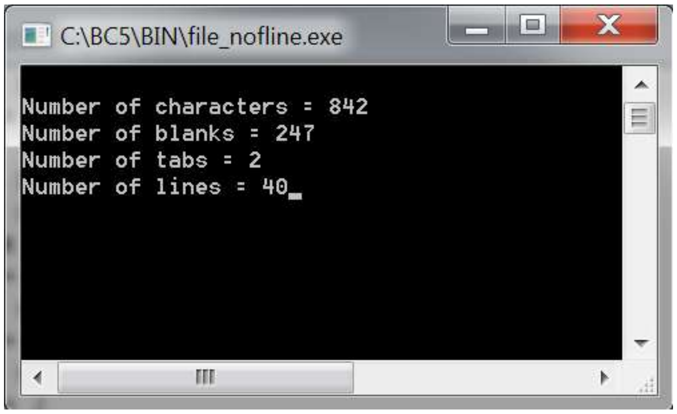
```
This is an example line. Line number 1
This is an example line. Line number 2
This is an example line. Line number 3
This is an example line. Line number 4
This is an example line. Line number 5
This is an example line. Line number 6
This is an example line. Line number 7
This is an example line. Line number 8
This is an example line. Line number 9
This is an example line. Line number 10
```

Example -4:- Write a Program that counts chars, spaces, tabs and new lines in a file.

```
/* Count chars, spaces, tabs and newlines in a file */
# include "stdio.h"
main( )
{
    FILE *fp ;
    char ch ;
    int nol = 0, not = 0, nob = 0, noc = 0 ;

    fp = fopen ( "PR1.C", "r" ) ;

    while ( 1 )
    {
        ch = fgetc ( fp ) ;

        if ( ch == EOF )
            break ;

        noc++ ;
```

```
            if ( ch == ' ' )
                nob++ ;

            if ( ch == '\n' )
                nol++ ;

            if ( ch == '\t' )
                not++ ;
        }

        fclose ( fp ) ;
        printf ( "\nNumber of characters = %d", noc ) ;
        printf ( "\nNumber of blanks = %d", nob ) ;
        printf ( "\nNumber of tabs = %d", not ) ;
        printf ( "\nNumber of lines = %d", nol ) ;
    }
```

Output:-



```
Number of characters = 842
Number of blanks = 247
Number of tabs = 2
Number of lines = 40_
```

fgetc() and fputc() functions ---- Copying Files:

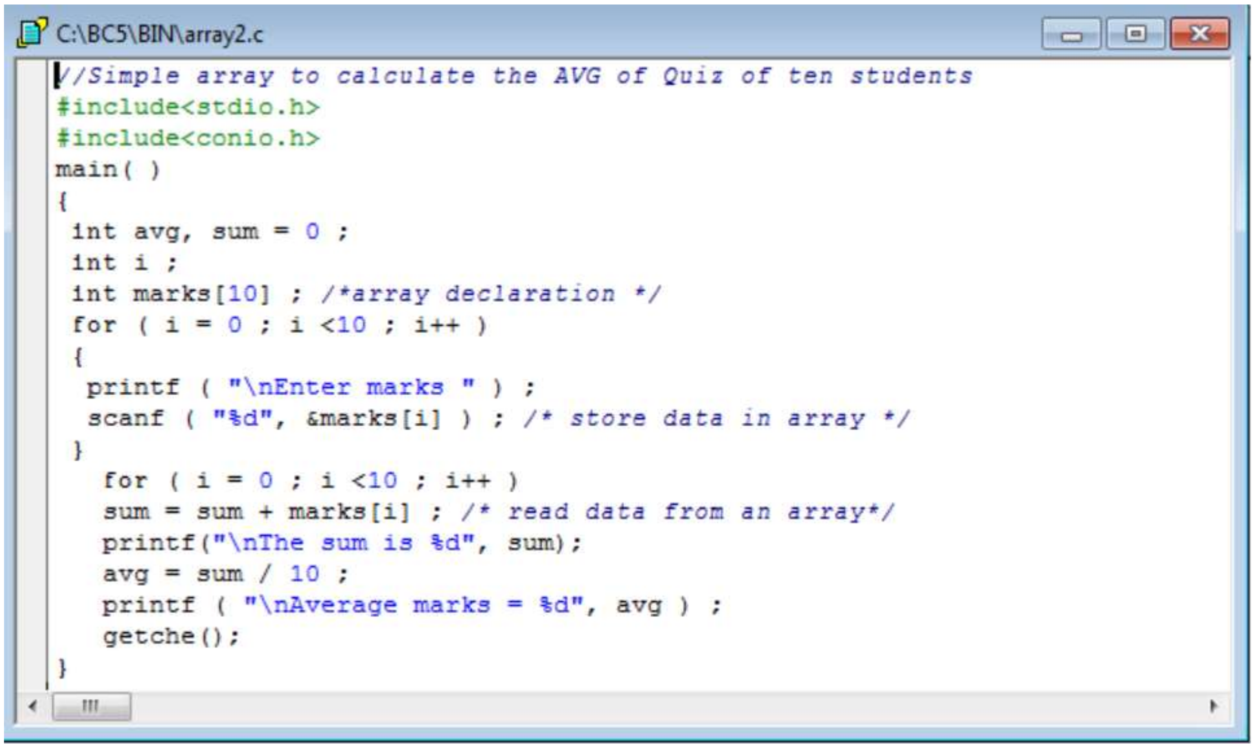
1. The function fgetc( ) which reads characters from a file.
2. Its counterpart is a function called fputc( )which writes characters to a file.
3. As a practical use of these character I/O functions we can copy the contents of one file into another, as demonstrated in the program on the next slide

Example -5:- Write a Program that Copy File.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
main( )
{
 FILE *fs, *ft ;
 char ch ;
 fs = fopen ( "Array1.c", "r" ) ;
 if ( fs == NULL )
 {
 puts ( "Cannot open source file" ) ;
 exit(0) ;
 }
 ft = fopen ( "array2.c", "w" ) ;
 if ( ft == NULL )
 {
 puts ( "Cannot open target file" ) ;
 fclose ( fs ) ;
 exit(0) ;
 }
 while ( 1 )
 {
 ch = fgetc ( fs ) ;
 if ( ch == EOF )
 break ;
 else
 fputc ( ch, ft ) ;
 }
 fclose ( fs ) ;
 fclose ( ft ) ;
 getche();
}
```

Output:

It copies the contents of array1.c to array2.c, the file is open below with the contents of array1.c



```
//Simple array to calculate the AVG of Quiz of ten students
#include<stdio.h>
#include<conio.h>
main( )
{
 int avg, sum = 0 ;
 int i ;
 int marks[10] ; /*array declaration */
 for ( i = 0 ; i <10 ; i++ )
 {
  printf ( "\nEnter marks " ) ;
  scanf ( "%d", &marks[i] ) ; /* store data in array */
 }
   for ( i = 0 ; i <10 ; i++ )
   sum = sum + marks[i] ; /* read data from an array*/
   printf("\nThe sum is %d", sum);
   avg = sum / 10 ;
   printf ( "\nAverage marks = %d", avg ) ;
   getche();
}
```

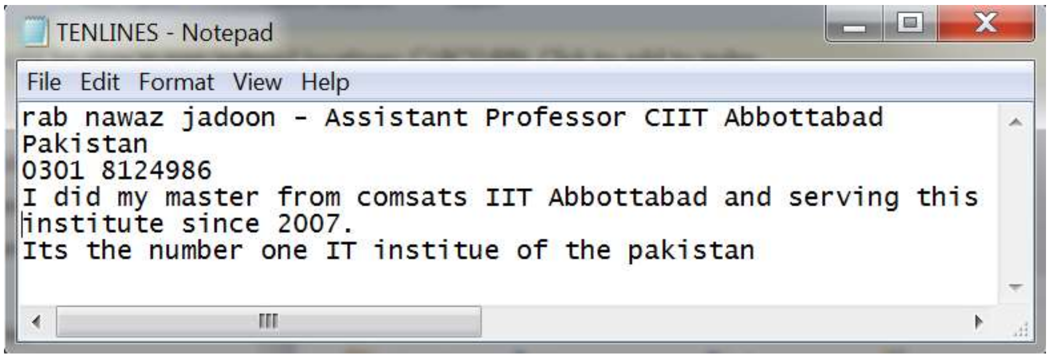
String (Line) I/O Files:

Reading or writing strings of characters from and to files is as easy as reading and writing individual characters. Program below, writes strings to a file using the function fputs( ).

Example -6:- Write a Program that receives strings from keyboard and writes them to file.

```
#include<conio.h>
#include<string.h>
main( )
{
 FILE *fp ;
 char s[80] ;
 fp = fopen ( "TENLINES.TXT", "w" ) ;
 if ( fp == NULL )
 {
 puts ( "Cannot open file" ) ;
 exit(0) ;
 }
 printf ( "\nEnter a few lines of text:\n" ) ;
 while ( strlen ( gets ( s ) ) > 0 )
 {
 fputs ( s, fp ) ;
 fputs ( "\n", fp ) ;
 }
 fclose ( fp ) ;
}
```

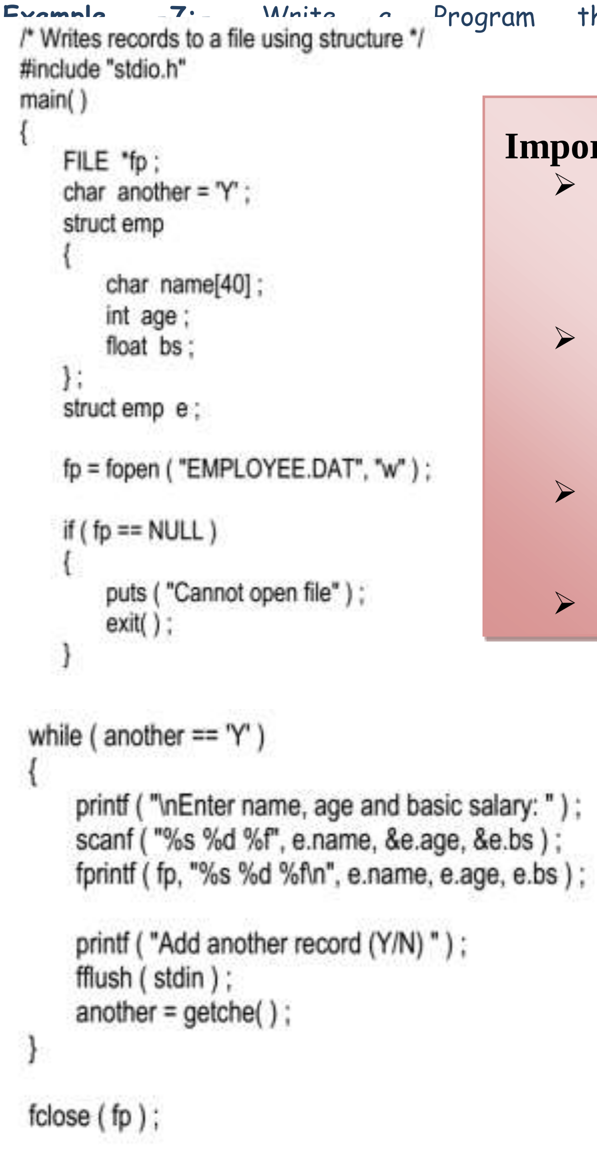
Output :



```
rab nawaz jadoon - Assistant Professor CIIT Abbottabad
Pakistan
0301 8124986
I did my master from comsats IIT Abbottabad and serving this
institute since 2007.
Its the number one IT institue of the pakistan
```

- The function fgets( )takes three arguments.
- The first is the address where the string is stored.
- The second is the maximum length of the string.
- The third argument, as usual, is the pointer to the structure FILE.

Example 7: Write a Program that record to a file using structures.



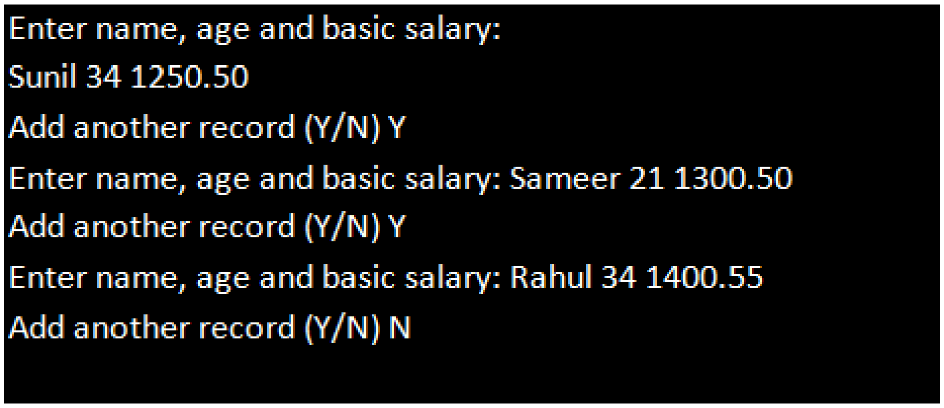
```
/* Writes records to a file using structure */
#include "stdio.h"
main( )
{
    FILE *fp ;
    char another = 'Y' ;
    struct emp
    {
        char name[40] ;
        int age ;
        float bs ;
    } ;
    struct emp e ;

    fp = fopen ( "EMPLOYEE.DAT", "w" ) ;

    if ( fp == NULL )
    {
        puts ( "Cannot open file" ) ;
        exit( ) ;
    }

  while ( another == 'Y' )
  {
      printf ( "\nEnter name, age and basic salary: " ) ;
      scanf ( "%s %d %f", e.name, &e.age, &e.bs ) ;
      fprintf ( fp, "%s %d %f\n", e.name, e.age, e.bs ) ;

      printf ( "Add another record (Y/N) " ) ;
      fflush ( stdin ) ;
      another = getche( ) ;
  }

  fclose ( fp ) ;
```

```
}        //end of main
```

Important Points

- The key to this program is the function fprintf(), which writes the values in the structure variable to the file.
- This function is similar to printf( ), except that a FILE pointer is included as the first argument.
- As in printf( ), we can format the data in a variety of ways, by using fprintf( ).
- In fact all the format conventions of

Output:



```
Enter name, age and basic salary:
Sunil 34 1250.50
Add another record (Y/N) Y
Enter name, age and basic salary: Sameer 21 1300.50
Add another record (Y/N) Y
Enter name, age and basic salary: Rahul 34 1400.55
Add another record (Y/N) N
```

Lab # 15

Graphics in C

15.1 Objective:

To start with graphics programming, Turbo C is a good choice. Even though DOS has its own limitations, it is having a large number of useful functions and is easy to program.

15.2 Scope:

To implement graphics algorithms,

To give graphical display of statistics,

To view signals from any source, we can use C graphics.

Draw basic geometric shapes.

15.3 Useful Concepts

1. Graphics.h Header File
2. Graphics.lib library file
3. Graphics driver (BGI file)
4. At least 640×480 VGA monitor
5. Header File : graphics.h
6. All Graphical Functions are Included in Graphics.h
7. After Including graphics.h Header File [You can get access graphical functions]
8. You must Know Following Things before Learning Turbo Graphics
9. InitGraph : Initializes the graphics system.
10. In C Program execution starts with main() similarly Graphics Environment Starts with this function.
11. initgraph() initializes the graphics system by loading a graphics driver from disk then putting the system into graphics mode
12. As this is Our first topic Under Graphics so it is better not to go in details of Parameters.

Simple Program using Graphic:

Example -1:- Write a program using graphics.

```
#include<graphics.h>

#include<conio.h>

void main()

{
 int gd = DETECT, gm;
 initgraph(&gd, &gm, "c:\\tc\\bgi");
///Your Code goes here
 getch();
 closegraph();
}
```

Output:

Firstly let me tell you what the output of this program is.

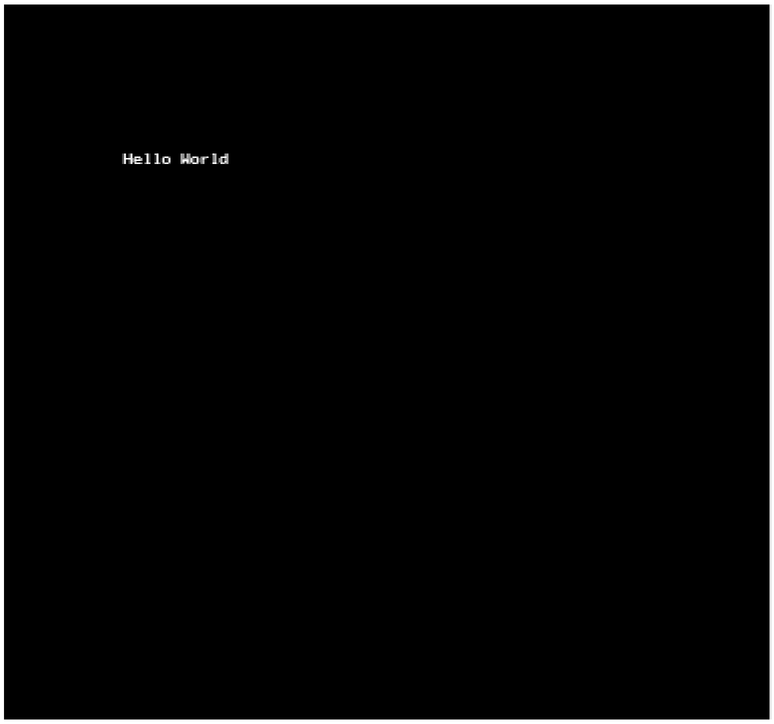
1. This program initializes graphics mode and then closes it after a key is pressed.
2. To begin with we have declared two variables of int type gd and gm for graphics driver and graphics mode respectively.
3. DETECT is a macro defined in "graphics.h" header file.
4. Then we have passed three arguments to initgraph function 1st is the address of gd, 2nd is the address of gm and 3rd is the path where your BGI files are present (you have to adjust this accordingly where you turbo compiler is installed).
5. getch helps us to wait until a key is pressed, closegraph function closes the graphics mode and finally return statement returns a value 0 to main indicating successful execution of your program
6. After you have understood initgraph function then you can use functions to draw shapes such as circle, line , rectangle etc, then you can learn how to change colors and fonts using suitable functions, then you can go for functions such as getimage, putimage etc for doing animation.

Program: Printing text in Graphics using Outtestxy Function

Example -2:- Write a program that prints text in Graphics using Outtestxy function.

```
#include<graphics.h>
#include<stdio.h>
int main(void)
{
  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  int x = 200, y = 200;
  outtextxy(x, y, "Hello World");
  closegraph();
}
```

Output:



```
Hello World
```

outtextxy(x,y,"Hello World");

This Function is Similar to Printf Statement.

Printf Prints Text on Screen in "Text Mode" while outtextxy() function Prints Text onto Screen in "Graphics Mode".

This Function Accepts 3 Parameters.

Syntax:

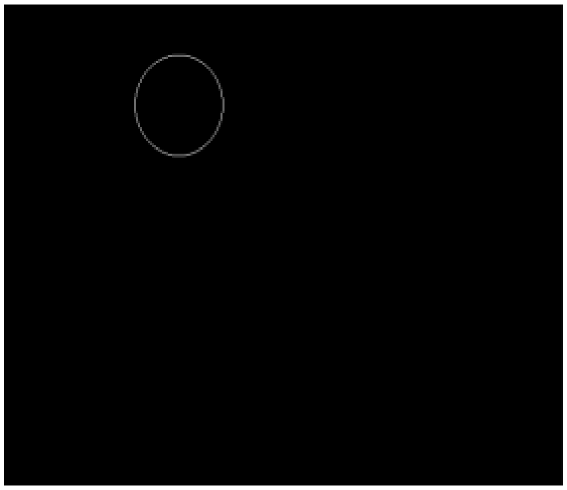
outtextxy(x,y,"Hello World");

Program: How to Draw a Circle:

Example -3:- Write a program that draws a circle.

```
#include<graphics.h>
#include<conio.h>
void main()
{
   int gd=DETECT, gm;
   initgraph(&gd, &gm, "c:\\turboc3\\bgi " );
   circle(200,100,150);
   getch();
   closegraph();
}
```

Output:



More detail on Syntax

1. InitGraph: Initializes the graphics system.
2. Declaration: void far initgraph(int far *graphdriver, int far *graphmode, char far *pathtodriver);
3. Remarks: To start the graphics system, you must first call initgraph.
4. initgraph initializes the graphics system by loading a graphics driver from disk (or validating a registered driver) then putting the system into graphics mode.
5. initgraph also resets all graphics settings (color, palette, current position, viewport, etc.) to their defaults, then resets graphresult to 0.

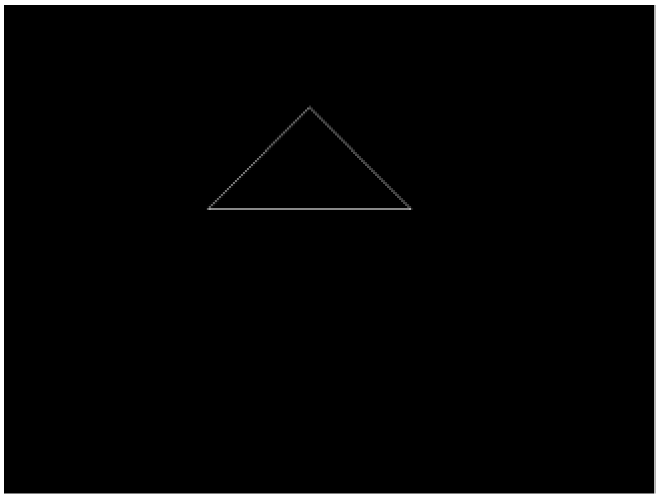
Program --- Draw a Rectangle:

Example -4:- Write a program that draws a rectangle.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
```

```
{
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "c:\\tc\\bgi");
   line(300, 100, 200, 200); // (from_x,from_y, to_x,to_y)
   line(300, 100, 400, 200);
   line(200, 200, 400, 200);
   getch();
   closegraph();
}
```

Output:

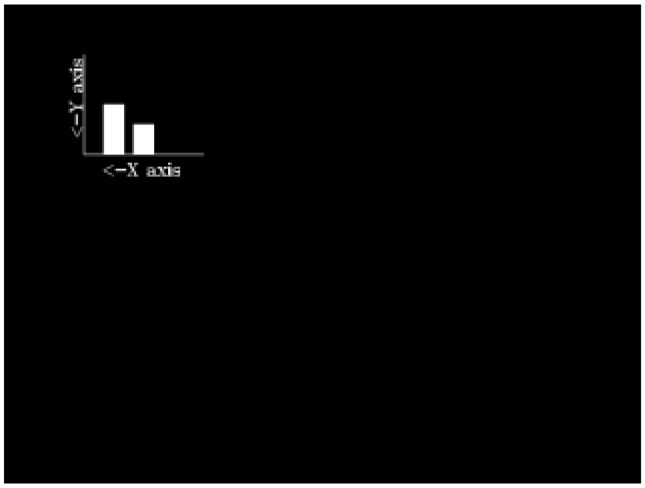


Program: Setting Styles and More:

Example -5:-write a program to illustrate how to use BARS which are used for visual statistics.

```
#include<graphics.h>
main() {
int gd=DETECT,gm,maxx,maxy,x,y,button;
initgraph(&gd,&gm,"");
line(80,150,200,150);
line(80,150,80,50);
settextstyle(1,HORIZ_DIR,1);
outtextxy(100,153,"<-X axis");
settextstyle(1,VERT_DIR,1);
outtextxy(60,50,"<-Y axis");
bar(100,100,120,150);
bar(130,120,150,150);
getch();
closegraph();
}
```

Output:

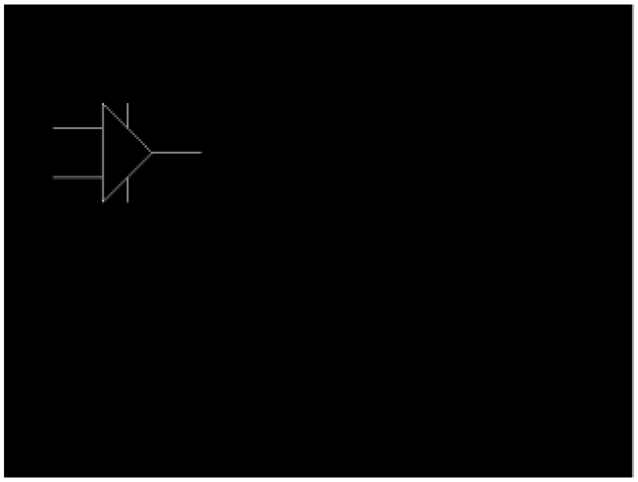


Program: Draw Op-amp using Graphics Function:

Example -6:- Write a program that draw Op-amp using graphics function.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main() {
int gd = DETECT, gm;
initgraph(&gd, &gm, "c:\\tc\\bgi");
line(50, 125, 100, 125);   //Horizontal Line -VE Terminal
line(50, 175, 100, 175);   //Horizontal Line +VE Terminal

    line(100, 100, 100, 200);  //Vertical Line
    line(100, 100, 150, 150);  //Top to Bottom Slant Line
    line(150, 150, 100, 200);  //Bottom to Top slant Line
    line(125, 100, 125, 125);  //Vertical Line +VCC
line(125, 175, 125, 200);  //Vertical Line -VCC
line(150, 150, 200, 150);  //Horizontal line
getch();
closegraph();

}
```

Output:



Important Functions:

Clearing the graphics window…

cleardevice();

Delay the program, so that users can see what is happening…sending in the number of msec

delay(milliseconds);

Wait for a keyboard hit:

getch();or,kbhit();

Important Function --- Draw Lines:

1) Set Drawing Color (for lines and edges)
2) (colors typically range from 0-15; 0 is usually black and 15 is white)
3) setcolor(color);
4) Set Background Color (usually for text)

5) setbkcolor(color);

6) Set Fill Style and Color (for interiors)

7) Pattern 0-12, 0 = empty, 1 = solid

8) setfillstyle(pattern, color)

9) Set Line Style and Thickness

10) Style: 0 = solid, 1 = dotted, 3 = dashed

11) Thickness is the width in terms of pixels

12) setlinestyle(style, pattern, thickness)

Important Function --- Draw Areas:

1. Drawing absolute (from one coordinate to another)
2. linerel(from_x, from_y,
3. Drawing a Circle
4. lGiven center and radius as whole numbers
5. circle (center_x, center_y, radius);
6. lDrawing a filled Rectangle
7. (given upper left and lower right corners)
8. bar(ul_x, ul_y, lr_x,lr_y);
9. lDrawing an unfilled Rectangle
10. (given upper left and lower right corners)
11. rectangle(ul_x, ul_y, lr_x, lr_y); to_x, to_y);

Important Function --- How Text looks:

1. Text Formatting
2. Set the justification
3. Horizontal: (0 = left, 1 = center, 2= right)
4. Vertical: (0 = bottom, 1 = center, 2 = top)

5. settextjustify(horizontal, vertical)
6. Set the text style
7. Font: (0-11)
8. Direction: 0 = left to right direction
9. Character Size: 0 = normal, 6 is really big!
10. settextstyle(font,direction, character size)

BLACK:	0
BLUE:	1
GREEN:	2
CYAN:	3
RED:	4
MAGENTA:	5
BROWN:	6
LIGHTGRAY:	7
DARKGRAY:	8
LIGHTBLUE:	9
LIGHTGREEN:	10
LIGHTCYAN:	11
LIGHTRED:	12
LIGHTMAGENTA:	13
YELLOW:	14
WHITE:	15

Important Function --- Messages:

1. Text Output
2. Set Text color (index ranges 0-15)
3. setcolor(index);
4. Output a message on the graphics window at the current position
5. outtext("messages on graphics window");
6. Output a message on the graphics window at the given x,y coordinate
7. outtextxy(x,y,"message");

Important Function --- Mouse Inputs:

1. Has there been a mouse click?
2. Right Click is 513
3. Left Click is 516Middle Click is 519 (the wheel…)
4. answer = ismouseclick(kind)
5. Clear the mouse click

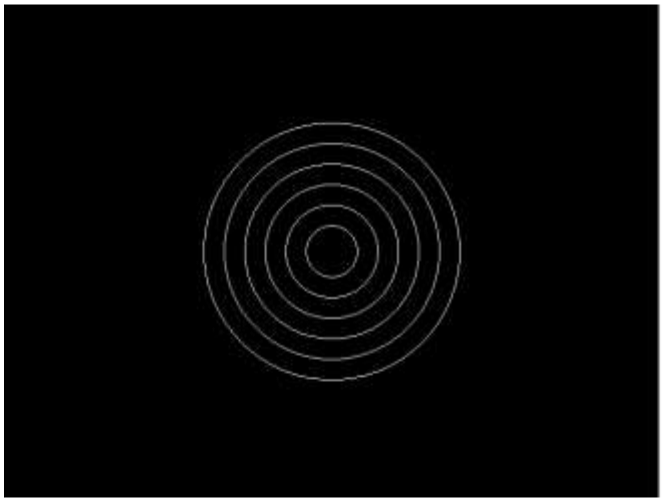
6. *(if you don't do this you can't get the next mouse click!)*
7. clearmouseclick(kind);
8. What was the coordinate when the mouse click happens…
9. x = mousex(); y = mousey();

Program: Moving on (concentric circles)

Example -7:-Write a program that moves on concentric circle.

```
#include <graphics.h>
int main()
{
    int gd = DETECT, gm;
    int x = 320, y = 240, radius;
    initgraph(&gd, &gm, "C:\\TC\\BGI");
    for ( radius = 25; radius <= 125 ; radius = radius + 20)
    circle(x, y, radius);
    getch();
    closegraph();
    return 0;
}
```

Output:



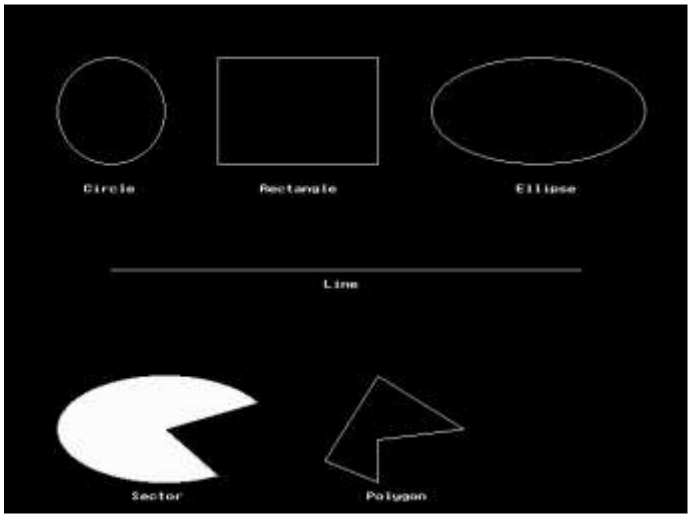
Basic Shapes and Colours:

Example -8:- Write a program that implements basic shapes and colours.

```
#include<graphics.h>
```

```
#include<conio.h>
void main()
{
int gd=DETECT, gm;
int poly[12]={350,450, 350,410, 430,400, 350,350, 300,430, 350,450 };
initgraph(&gd, &gm, "");
circle(100,100,50);
outtextxy(75,170, "Circle");    rectangle(200,50,350,150);
outtextxy(240, 170, "Rectangle");
ellipse(500, 100,0,360, 100,50);
outtextxy(480, 170, "Ellipse");
line(100,250,540,250);
outtextxy(300,260,"Line");
sector(150, 400, 30, 300, 100,50);
outtextxy(120, 460, "Sector");
drawpoly(6, poly);
outtextxy(340, 460, "Polygon");
getch();
closegraph();
}
```

Output:

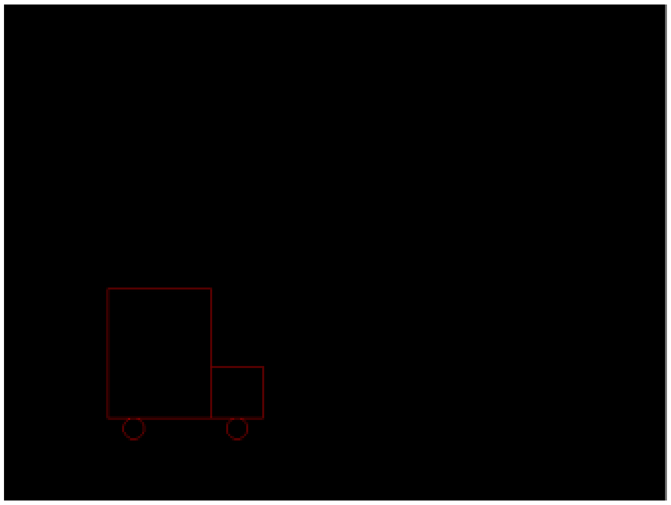


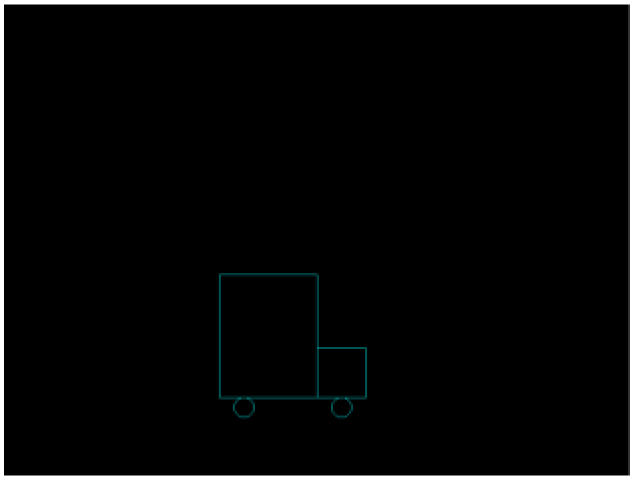
Program: Moving Car:

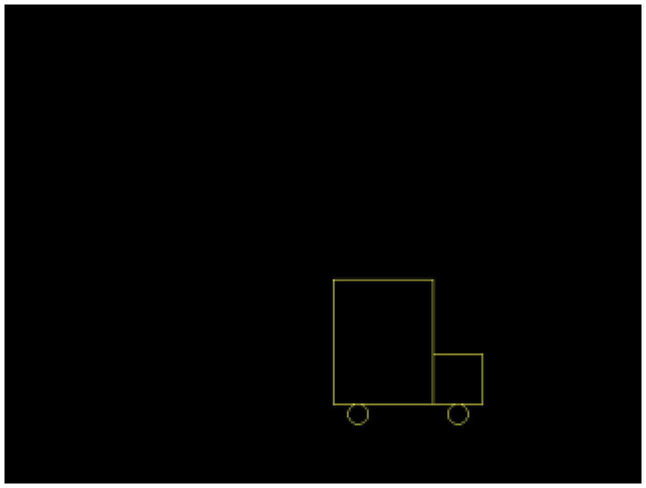
Example -9:- Write a program that implements the code of moving car.

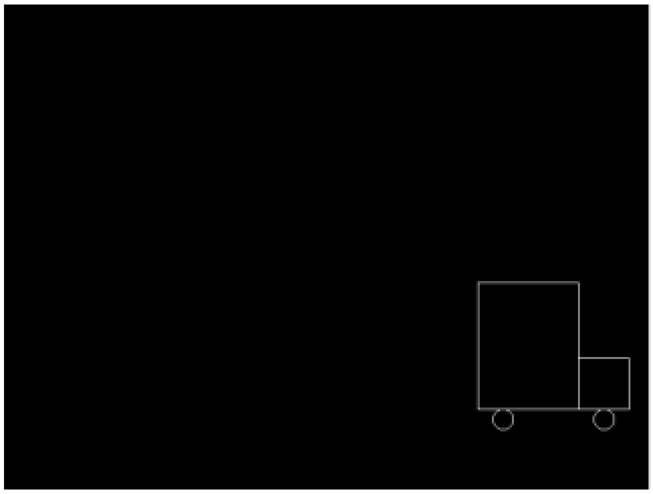
```
#include <graphics.h>
#include <dos.h>
int main()
{
int i, j = 0, gd = DETECT, gm;
initgraph(&gd,&gm,"C:\\TC\\BGI");
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
outtextxy(25,240,"Press any key to view the moving car");
getch();
for( i = 0 ; i <= 420 ; i = i + 10, j++ )
{
rectangle(50+i,275,150+i,400);
rectangle(150+i,350,200+i,400);
circle(75+i,410,10);
circle(175+i,410,10);
setcolor(j);
delay(100);
if( i == 420 )
break;
if ( j == 15 )
j = 2;
cleardevice(); // clear screen
}
getch();
closegraph();
return 0;
}
```

Output:









Some Graphic Effects using Random Numbers:

15.4 Examples

Example -10:-Write a program that calculates random numbers using some graphics.

```
#include "graphics.h"

#include "conio.h"

#include "stdlib.h"

void main()

{

int gd,gm;

gd=DETECT;

initgraph(&gd, &gm, "");

setcolor(3);

setfillstyle(SOLID_FILL,RED);

bar(50, 50, 590, 430);

setfillstyle(1, 14);

bar(100, 100, 540, 380);

while(!kbhit())

{

putpixel(random(439)+101,  random(279)+101,random(16));

setcolor(random(16));

circle(320,240,random(100));

}
getch();
closegraph();
}
```