LECTURE-5

Basics of C++

C ++ is an object oriented programming language, C ++ was developed by Jarney
Stroustrup at AT & T Bell lab, USA in early eighties. C ++ was developed from c and simula 67 language. C ++ was early called 'C with classes'.

C++ Comments:

C++ introduces a new comment symbol //(double slash). Comments start with a
double slash symbol and terminate at the end of line. A comment may start any where in the line and what ever follows till the end of line is ignored. Note that there is no closing symbol.
The double slash comment is basically a single line comment. Multi line comments can be written as follows:

```
// this is an example of
// c++ program
// thank you
```

The c comment symbols /* ....*/ are still valid and more suitable for multi line comments.

```
/* this is an example of c++ program */
```

Output Operator:

The statement cout <<"Hello, world" displayed the string with in quotes on the screen. The identifier cout can be used to display individual characters, strings and even numbers. It is a predefined object that corresponds to the standard output stream. Stream just refers to a flow of data and the standard Output stream normally flows to the screen display. The cout object, whose properties are defined in iostream.h represents that stream. The insertion operator << also called the 'put to' operator directs the information on its right to the object on its left.

Return Statement:

In C++ main ( ) returns an integer type value to the operating system. Therefore every main ( ) in C++ should end with a return (0) statement, otherwise a warning or an error might occur.

Input Operator:

The statement

```
cin>> number 1;
```

is an input statement and causes. The program to wait for the user to type in a number. The number keyed in is placed in the variable number1. The identifier **cin** is a predefined object in C++ that corresponds to the standard input stream. Here this stream represents the key board.

The operator >> is known as get from operator. It extracts value from the keyboard and assigns it to the variable on its right.

Cascading Of I/O Operator:

```
cout<<"sum="<<sum<<"\n";
cout<<"sum="<<sum<<"\n"<<"average="<<average<<"\n";
cin>>number1>>number2;
```

Structure Of A Program :

Probably the best way to start learning a programming language is by writing a program.

Therefore,
here is our first program:

```
// my first program in C++

#include <iostream>
using namespace std;
int main ()
{
cout << "Hello World!";
return 0;
}
```

Output:-Hello World!

The first panel shows the source code for our first program. The second one shows the result of the
program once compiled and executed. The way to edit and compile a program depends on the
compiler you are using. Depending on whether it has a Development Interface or not and on its
version. Consult the compilers section and the manual or help included with your compiler if you
have doubts on how to compile a C++ console program.
The previous program is the typical program that programmer apprentices write for the first time,
and its result is the printing on screen of the "Hello World!" sentence. It is one of the simplest
programs that can be written in C++, but it already contains the fundamental components that every
C++ program has. We are going to look line by line at the code we have just written:

// my first program in C++

This is a comment line. All lines beginning with two slash signs (//) are considered comments and do
not have any effect on the behavior of the program. The programmer can use them to include short
explanations or observations within the source code itself. In this case, the line is a brief description
of what our program is.

#include <iostream>

Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code
lines with expressions but indications for the compiler's preprocessor. In this case the directive
#include<iostream> tells the preprocessor to include the iostream standard file. This specific file
(iostream) includes the declarations of the basic standard input-output library in C++, and it is
included because its functionality is going to be used later in the program.

using namespace std;

All the elements of the standard C++ library are declared within what is called a namespace, the
namespace with the name *std*. So in order to access its functionality we declare with this

executed in any C++ program. For that same reason, it is essential that all C++ programs have a main function.
The word main is followed in the code by a pair of parentheses (()). That is because it is a function declaration: In C++, what differentiates a function declaration from other types of expressions are these parentheses that follow its name. Optionally, these parentheses may enclose a list of parameters
within them.
Right after these parentheses we can find the body of the main function enclosed in braces ({}).
What is contained within these braces is what the function does when it is executed.
cout << "Hello World!";
This line is a C++ statement. A statement is a simple or compound expression that can actually
produce some effect. In fact, this statement performs the only action that generates a visible effect in
our first program.
cout represents the standard output stream in C++, and the meaning of the entire statement is to
insert a sequence of characters (in this case the Hello World sequence of characters) into the standard
output stream (which usually is the screen).
cout is declared in the iostream standard file within the std namespace, so that's why we needed to
include that specific file and to declare that we were going to use this specific namespace earlier in
our code.
Notice that the statement ends with a semicolon character (;). This character is used to mark the end
of the statement and in fact it must be included at the end of all expression statements in all C++
programs (one of the most common syntax errors is indeed to forget to include some semicolon after
a statement).
return 0;
The return statement causes the main function to finish. return may be followed by a return code (in
our example is followed by the return code 0). A return code of 0 for the main function is generally
interpreted as the program worked as expected without any errors during its execution. This is the
most usual way to end a C++ console program.
You may have noticed that not all the lines of this program perform actions when the code is executed. There were lines containing only comments (those beginning by //). There were lines with
directives for the compiler's preprocessor (those beginning by #). Then there were lines that began
the declaration of a function (in this case, the main function) and, finally lines with statements (like
the insertion into cout), which were all included within the block delimited by the braces ({}) of the
main function.
The program has been structured in different lines in order to be more readable, but in C++, we do
not have strict rules on how to separate instructions in different lines. For example, instead of

```
int main ()
{
cout << " Hello World!";
return 0;
}
```

code in different lines serves only to make it more legible and schematic for the humans that may read it.

Let us add an additional instruction to our first program:

```
// my second program in C++
#include <iostream>
using namespace std;
int main ()
{
cout << "Hello World! ";
cout << "I'm a C++ program";
return 0;
}
```

Output:-Hello World! I'm a C++ program

In this case, we performed two insertions into cout in two different statements. Once again, the separation in different lines of code has been done just to give greater readability to the program, since main could have been perfectly valid defined this way:

```
int main ()
{
cout << " Hello World! ";
cout << " I'm a C++ program ";
return 0;
}
```

We were also free to divide the code into more lines if we considered it more convenient:

```
int main ()
{
cout << "Hello World!";
cout << "I'm a C++ program";
return 0;
}
```

And the result would again have been exactly the same as in the previous examples.
Preprocessor directives (those that begin by #) are out of this general rule since they are not statements. They are lines read and processed by the preprocessor and do not produce any code by themselves. Preprocessor directives must be specified in their own line and do not have to end with a semicolon (;).

STRUCTURE OF C++ PROGRAM

~~Include files~~
Class declaration
Class functions, definition
Main function program

Example :-

# include<iostream.h>

class person

```
{
char name[30];
int age;
public:
          void getdata(void);
           void display(void);
};


void person :: getdata ( void )
{
cout<<"enter name";
cin>>name;
            cout<<"enter age";
cin>>age;
}
void display()
{
        cout<<"\n name:"<<name;
cout<<"\n age:"<<age;
}
int main( )
{
person p;
p.getdata();
p.display();
return(0);
}
```

LECTURE-6

TOKENS:

The smallest individual units in program are known as **tokens.** C++ has the following tokens.

i. Keywords
ii. Identifiers
iii. Constants
iv. Strings
v. Operators

KEYWORDS:

The keywords implement specific C++ language feature. They are explicitly reserved identifiers and can't be used as names for the program variables or other user defined program elements. The keywords not found in ANSI C are shown in red letter.

C++ KEYWORDS:

Asm double new switch

Auto else operator template

Break enum private this

Case extern protected throw

Catch float public try

Char for register typedef

Class friend return union

Const goto short unsigned

Continue if signed virtual

Default inline sizeof void

Delete long struet while

IDENTIFIERS:

Identifiers refers to the name of variable , functions, array, class etc. created by programmer. Each language has its own rule for naming the identifiers.
The following rules are common for both C and C++.

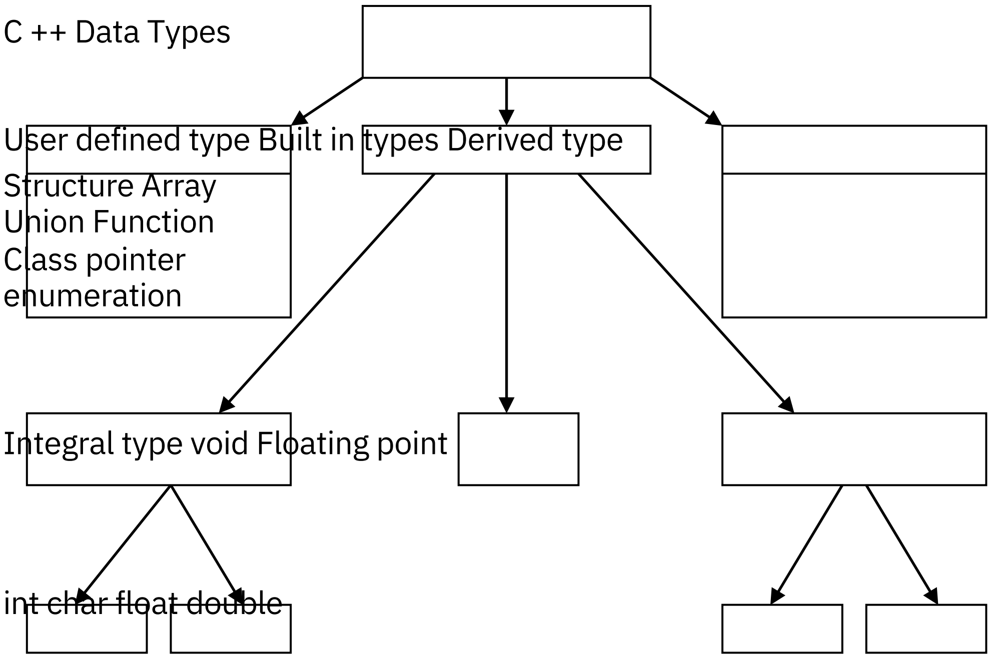
1. Only alphabetic chars, digits and under score are permitted.
2. The name can’t start with a digit.
3. Upper case and lower case letters are distinct.
4. A declared keyword can’t be used as a variable name.

In ANSI C the maximum length of a variable is 32 chars but in c++ there is no bar.

P.T.O

Lecture-7

BASIC DATA TYPES IN C++



Both C and C++ compilers support all the built in types. With the exception of void the basic datatypes may have several modifiers preceding them to serve the needs of various situations. The modifiers signed, unsigned, long and short may applied to character and integer basic data types. However the modifier long may also be applied to double.

Data types in C++ can be classified under various categories.

TYPE	**BYTES**	**RANGE**
char	1	-128 to – 127
usigned	1	0 to 265
sgned char	1	-128 to 127
int	2	-32768 to 32768
unsigned int	2	0 to 65535
singed int	2	-32768 to 32768
short int	2	-32768 to 32768

long int 4 -2147483648 to 2147483648

signed long int 4 -2147483648 to 2147483648

unsigned long int 4 0 to 4294967295

float 4 3.4E-38 to 3.4E+38

double 8 1.7E -308 to 1.7E +308

long double 10 3.4E-4932 to 1.1E+ 4932

The type void normally used for:

1) To specify the return type of function when it is not returning any value.
2) To indicate an empty argument list to a function.
Example:

Void function(void);

Another interesting use of void is in the declaration of genetic pointer

Example:

Void *gp;

Assigning any pointer type to a void pointer without using a cast is allowed in both C and ANSI C. In ANSI C we can also assign a void pointer to a non-void pointer without using a cast to non void pointer type. This is not allowed in C ++.

Example:

v**oid *ptr1;**

void *ptr2;

Are valid statement in ANSI C but not in C++. We need to use a cast operator.

p**tr2=(char *) ptr1;**

USER DEFINED DATA TYPES:

STRUCTERS AND CLASSES

We have used user defined data types such as struct,and union in C. While these more features have been added to make them suitable for object oriented programming. C++ also permits us to define another user defined data type known as class which can be used just like any other basic data type to declare a variable. The class variables are known as objects, which are the central focus of oops.

ENUMERATED DATA TYPE:

An enumerated data type is another user defined type which provides a way for attaching names to number, these by increasing comprehensibility of the code. The enum keyword automatically enumerates a list of words by assigning them values 0,1,2 and soon. This facility provides an alternative means for creating symbolic.

Example:

enum shape { circle,square,triangle}

enum colour{red,blue,green,yellow}

enum position {off,on}

The enumerated data types differ slightly in C++ when compared with ANSI C. In C++, the tag names shape, colour, and position become new type names. That means we can declare new variables using the tag names.

Example:

Shape ellipse;//ellipse is of type shape

colour background ; // back ground is of type colour

ANSI C defines the types of enums to be ints. In C++,each enumerated data type retains its own separate type. This means that C++ does not allow an int value to be automatically converted to an enum.

Example:

colour background =blue; //vaid

colour background =7; //error in c++

colour background =(colour) 7;//ok

How ever an enumerated value can be used in place of an int value.

Example:

i**nt c=red ;//valid, colour type promoted to int**

By default, the enumerators are assigned integer values starting with 0 for the first enumerator, 1 for the second and so on. We can also write

e**num color {red, blue=4,green=8};**

enum color {red=5,blue,green};

C++ also permits the creation of anonymous enums (i.e, enums without tag names)

Example:

e**num{off,on};**

Here off is 0 and on is 1.these constants may be referenced in the same manner as regular constants.

Example:

i**nt switch-1=off;**

int switch-2=on;

ANSI C permits an enum defined with in a structure or a class, but the enum is
globally visible. In C++ an enum defined with in a class is local to that class.

LECTURE-8

SYMBOLIC CONSTANT:

There are two ways of creating symbolic constants in c++.

1. using the qualifier const.
2. defining a set of integer constants using enum keywords.

In both C and C++, any value declared as const can't be modified by the program in any way. In C++, we can use const in a constant expression. Such as

const int size = 10 ;
char name (size) ;

This would be illegal in C. const allows us to create typed constants instead of having to use #defme to create constants that have no type information.
const size=10;

Means

const int size =10;

C++ requires a const to be initialized. ANSI C does not require an initializer, if none is given, it initializes the const to 0.

In C++ const values are local and in ANSI C const values are global .However they can be made local made local by declaring them as static .In C++ if we want to make const value as global then declare as storage class.

Ex: external const total=100; Another method of naming integer constants is as follows:-
enum {x,y,z};

DECLARATION OF VARIABLES:

In ANSIC C all the variable which is to be used in programs must be declared at the beginning of the program .But in C++ we can declare the variables any whose in the program where it requires .This mak
program much easier to write and reduces the errors that may be caused by having to scan back and for
also makes the program easier to understand because the variables are declared in the context of their

Example:

main( )
{
float x,average;
float sum=0;

```
for(int i=1;i<5;i++)
{
cin>>x;
sum=sum+x
}
float average;
average=sum/x;
cout<<average;
}
```

REFERENCE VARIABLES:

C++interfaces a new kind of variable known as the reference variable. A references variable provides an alias.(alternative name) for a previously defined variable. For example ,if we make the variable sum a reference to the variable total, then sum and total can be used interchangeably to repres the variuble.

A reference variable is created as follows:

Synatx: Datatype & reference –name=variable name;

Example:

float total=1500;
float &sum=total;

Here sum is the alternative name for variables total, both the variables refer to the same data object in t memory .

A reference variable must be initialized at the time of declaration .

Note that C++ assigns additional meaning to the symbol *&* here *&* is not an address operator .The notation float *&* means reference to float.

Example:

```
                                int n[10];
int &x=n[10];
char &a=’\n’;
```

P.T.O

LECTURE-9

OPERATORS IN C++ :

C++ has a rich set of operators. All C operators are valid in C++ also. In addition. C++ introduces some new operators.

<< insertion operator

>> extraction operator

: : scope resolution operator

: :* pointer to member declarator

* pointer to member operator

.* pointer to member operator

Delete memory release operator

Endl line feed operator

New memory allocation operator

Setw field width operator

SCOPE RESOLUTION OPERATOR:

Like C,C++ is also a block-structured language. Block -structured language. Blocks and scopes can be used in constructing programs. We know same variables can be declared in different blocks because the variables declared in blocks are local to that function.
Blocks in C++ are often nested.
Example:

```
----------------
----------------
{
Int x =10;
---------------
---------------
{
Int x=1; Block2
                                   --------------- Block1
---------------
}
----------------
----------------
}
```

Block2 contained in block l .Note that declaration in an inner block hides a declaration of the same variable in an outer block and therefore each declaration of x causes it to refer to a different data
With in the inner block the variable x will refer to the data object declared there in.

In C,the global version of a variable can't be accessed from with in the inner block. C++ resolves this problem by introducing a new operator :: called the scope resolution operator .This can be used to uncover a hidden variable.

Syntax: : : variable –name;

Example:

```
#include <iostrcam.h>
                                                   int m=10;
main()
{
int m=20;
{
int k=m;
int m=30;
cout<<"we are in inner block";
cout<<"k="<<k<<endl;
cout<<"m="<<m<<endl;
cout<<":: m="<<:: m<<endl;
}
cout<<"\n we are in outer block \n";
cout<<"m="<<m<<endl;
cout<<":: m="<<:: m<<endl;
}
```

Memory Management Operator

C uses malloc and calloc functions to allocate memory dynamically at run time . Similarly it uses the fun
Free( ) to free dynamically allocated memory. We use dynamic allocation techniques when it is not know
advance how much of memory space as needed .

C++ also support those functions it also defines two unary operators new and delete that

perform the task of allocating and freeing the memory in a better and easier way.

The new operator can be used to create objects of any type. **Syntax: pointer-**

variable =new datatype;

Example:

p=new int; q=new int;

Where p is a pointer of type int and q is a pointer of type float.

int *p=new int;

float *p=new float;

Subsequently, the statements

*p=25;

*q=7.5;
Assign 25 to the newly created int object and 7.5 to the float object.We can also initialize the memory using the new operator.

Syntax:

int *p=ne\v int(25);
float *q =new float(7.5);
new can be used to create a memory space for any data type including user defined such as arrays,structures,and classes .The general form for a one-dimensional array is:

pointer-variable =new data types [size];

creates a memory space for an array of 10 integers.

If a data object is no longer needed, it is destroyed to release the memory space for reuse.

Syntax: delete pointer-variable;

Example:
delete p;
delete q;

form of deleteIf we want to free a dynamically allocated array ,we must use the following

delete [size] pointer-variable;
or

delete [ ] pointer variable;

MANIPULATERS:

Manipulators are operator that are used to format the data display. The most commonly manipulators ar
endl and setw.
The endl manipulator, when used in an output statement, causes a line feed to be insert.(just like \n)

Example:
cout<<"m="<<m<<endl;
cout<<"n="<<n<<endl;
cout<<"p="<<p<<endl;
If we assume the values of the variables as 2597,14 and 175 respectively
m=2597; n=14;
p=175
It was want to print all nos in right justified way use setw which specify a common field width for all the nos.

Example: cout<<setw(5)<<sum<<endl;

```
cout<<setw(10)<<"basic"<<setw(10<<basic<<endl;
              Cout<<setw(10)<<"allowance"<<setw(10<<allowance<<endl;
cout<<setw(10)<<"total="<<setw(10)<<total;
```

LECTURE-10

CONTROL STRUCTURES:

Like c,c++, supports all the basic control structures and implements them various control statements.

The if statement:

The if statement is impklemented in two forms:

1. simple if statement

2. if... else statement

Simple if statement:

if (condition)

{

Action;

}

If.. else statement

If (~~condition)~~

Statment1

Else

Statement2

The switch statement

This is a multiple-branching statement where, based on a condition, the control is transferred to one of t
possible points;

```
Switch(expr)

{

case 1:

action1;

break;

case 2:

action2;

break;

..

..

default:

message

}
```

The while statement:

Syn:

```
While(condition)

{

Stements

}
```

The do-while statement:

Syn:

```
do

{

Stements

} while(condition);
```

The for loop:

```
for(expression1;expression2;expression3)

{

Statements;

Statements;

}
```

P.T.O

LECTURE-11

FUNCTION IN C++ :

The main( ) Functon ;

~~ANSI does not spe~~cify any return type for the main ( ) function which is the starting point for the executic
of a program . The definition of main( ) is :-

```
main()
{
//main program statements
}
```

This is property valid because the main () in ANSI C does not return any value. In C++, the main () return
type int to the operating system. The functions that have a return value should use the return statement
The main () function in C++ is therefore defined as follows.

```
int main( )
{
--------------
--------------
return(0)
}
```

Since the return type of functions is int by default, the key word int in the main( ) header is optional.

INLINE FUNCTION:

~~To eliminate the cost~~ of calls to small functions C++ proposes a new feature called inline function.
An inline function is a function that is expanded inline when it is invoked .That is the compiler
replaces the function call with the corresponding function code.
The inline functions are defined as follows:-

```
inline function-header
{
function body;
}
```

Example: inline double cube (double a)

```
{
return(a*a*a);
}
```

The above inline function can be invoked by statements like

```
c=cube(3.0);
d=cube(2.5+1.5);
```

remember that the inline keyword merely sends a request, not a command to the compliler. The
compiler may ignore this request if the function definition is too long or too complicated and compile
the function as a normal function.
Some of the situations where inline expansion may not work are:
1. For functions returning values if a loop, a switch or a go to exists.

2. for function s not returning values, if a return statement exists. 3. if functions contain static variables.
4. if inline functions are recursive,.

Example:

```
#include<iostream.h>
#include<stdio.h>
inline float mul(float x, float y)
{
return(x*y);
}
inline double div(double p.double q)
{
return(p/q);
}



main( )
{
float a=12.345;
float b=9.82;
cout<<mul(a,b)<<endl;
cout<<div (a,b)<<endl;
}
```

output:-

```
121.227898
1.257128
```

DEFAULT ARGUMENT:-

C++ allows us to call a function with out specifying all its arguments.In such cases, the function assigns a default value to the parameter which does not have a matching aguments in the function call.Default values are specified when the function is declared .The compiler looks at the prototype to see how many arguments a function uses and alerts the program for possible default values.
Example: float amount (float principle, int period ,float rate=0.15);
The default value is specified in a manner syntactically similar to a variable initialization .The above prototype declares a default value of 0.15 to the argument rate. A subsequent function call like
value=amount(5000,7); //one argument missing
passes the value of 5000 to principle and 7 to period and then lets the function, use default value of 0.15 for rate.
The call:- value=amount(5000,5,0.12);
//no missing argument passes an explicite value of 0.12 rate.
One important point to note is that only the trailing arguments can have default values. That is, we must add default from right to left .We cannot provide a default to a particular argument in the middle of an argument list.
Example:- int mul(int i, int j=5,int k=10);//illegal
int mul(int i=0,int j,int k=10);//illegal
int mul(int i=5,int j);//illegal
int mul(int i=2,int j=5,int k=10);//illegal
Default arguments are useful in situation whose some arguments always have the some value.
For example,bank interest may retain the same for all customers for a particular period of deposit.

Example:

```
#include<iostream.h>
#include<stdio.h>
mainQ
{
float amount;
float value(float p,int n,float r=0.15);
void printline(char ch=’*’,int len=40);
printline( );
amount=value(5000.00,5);
cout<<”\n final value=”<<amount<<endl;
printline(‘=’);
//function definitions
float value (float p,int n, float r)
{
float si;
si=(p*n*r)/100;
return(si);
}
void printline (char ch,int len)
{
for(inti=l;i<=len;i++)
cout<<ch<<endl;
}
output:-
* * * * * * * * * * * * * * * *
final value=10056.71613
= = = = = = = = = = = = = = = =
```

Advantage of providing the default arguments are:
1. We can use default arguments to add new parameters to the existing functions.
2. Default argument s can be used to combine similar functions into one.

CONST ARGUMENT:-

In C++, an argument to a function can be declared as unit as const as shown below.
int strlen(const char *p);
int length(const string &s);
The qualifier const tells the compiler that the function should not modify the argument .the compiler will generate an error when this condition is violated .This type of declaration is significant only when we pass arguments by reference or pointers.

LECTURE-12

FUNCTION OVERLOADING:

Overloading refers to the use of the same thing for different purposes . C++ also permits overloading functions .This means that we can use the same function name to creates functions that perform a variety of different tasks. This is known as function polymorphism in oops.

Using the concepts of function overloading , a family of functions with one function name but with different argument lists in the functions call .The correct function to be invoked is determined by checking the number and type of the arguments but not on the function type.

For example an overloaded add() function handles different types of data as shown below.

//Declaration
int add(int a, int b); //prototype 1
int add (int a, int b, int c); //prototype 2
double add(double x, double y); //prototype 3
double add(double p , double q); //prototype 4

//function call
cout<<add(5,10); //uses prototype 1
cout<<add(15,10.0); //uses prototype 4
cout<<add(12.5,7.5); //uses prototype 3
cout<<add(5,10,15); //uses prototype 2
cout<<add(0.75,5); //uses prototype 5

A function call first matches the prototype having the same no and type of arguments and then calls the appropriate function for execution.
The function selection invokes the following steps:-

a) The compiler first tries to find an exact match in which the types of actual arguments are the same and use that function .
b) If an exact match is not found the compiler uses the integral promotions to the actual arguments such as :
char to int
float to double
to find a match
c)When either of them tails ,the compiler tries to use the built in conversions to the actual arguments and them uses the function whose match is unique . If the conversion is possible to have multiple matches, then the compiler will give error message.
Example:
long square (long n);
double square(double x);
A function call such as :- square(lO)
Will cause an error because int argument can be converted to either long or double .There by creating an ambiguous situation as to which version of square( )should be used.

PROGRAM

```
#include<iostream.h>
int volume(double,int);
double volume( double , int );
double volume(longint ,int ,int);
main( )
{
cout<<volume(10)<<endl;
                              cout<<volume(10)<<endl; cout<<volume(10)<<endl;
}
int volume( ini s)
{
return (s*s*s); //cube
}
double volume( double r, int h)
{
return(3.1416*r*r*h); //cylinder
}
long volume (longint 1, int b, int h)
{
return(1*b*h); //cylinder
}
```

```
output:- 1000
157.2595
                                            112500
```