**Experiment#09**

**OBJECTIVES**

**a) To become familiar with the function that does not return the value.**

**b) To become familiar function overloading and inline functions.**

**c) To understand the usage of Macros.**

**THEORY**

:

**Functions that does not return a value**

The keyword void can use as a type spacifeir when defining the function that does not return any thing or when the function definition does not include any arguments. The presence of this key word is not mandatory, but it is good programming practice to make of this future.

**Example**

#include<iostream.h>  
#include<conio.h>

void add(int a,int b);

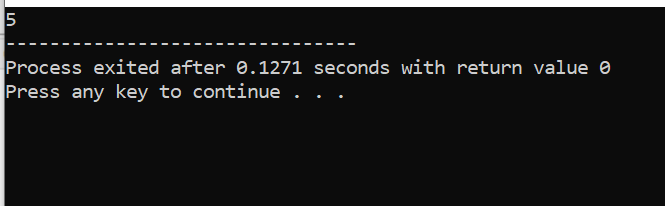
main( )  
{  
add(3,2);

}

void add(int a ,int b)

{  
cout<<a+b;

}



**Overloaded Function :**

Different functions with same name perform different operation, depends upon the number of arguments & type of the arguments is known as Function Overloading.

**Example**

#include<iostream.h>  
#include<conio.h>  
int add(int a,int b);

float add(float a,int b);

float add(int a,float b);

int add(int a,int b,int c);

main()  
{  
clrscr();

cout<<add(2,2);

cout<<add(2.1,2);

cout<<add(2,2.1);

cout<<add(2,3,4);

getch();

}

int add(int a,int b)

{  
return a+b;

}

float add(float a,int b)

{  
return a+b;

}

float add(int a,float b)

{  
return a+b;

}

int add(int a,int b,int c)

{  
return a+b+c;

}  
**Result:**

4 4.1 4.1 9

**Inline Function:**

You can create short functions that are not actually called; rather their code is expanded inline at the point of each invocation. This process is similar to using function like macro. To cause the function to be extended inline rather then called, precede its definition with the inline keyword.

**Example**

#include<iostream.h>  
#include<conio.h>  
inline int add(int a,int b)

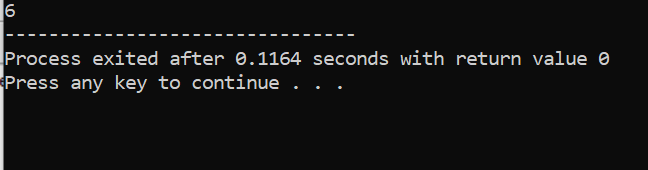
{  
return a+b;

}

main()  
{  
clrscr();

cout<<add(3,3);

}



**Default Arguments:**

When passing the parameters, you do not need to specify all the arguments to a function, it may assume some of them on its own.

**Example**

#include<iostream.h>  
#include<conio.h>  
int add(int a,int b,int c=3);

main()  
{  
cout<<add(2,2);

}  
int add( int a, int b, int c)

{  
return a+b+c;

}

**Result: 7**

**Macros**

The #define statement can be used for more, however, then simply defining symbolic constant. In particular, it can be used to define macros, i.e. single identifier that is equivalent to expressions, complete statement or group of statements. Macros resemble function in this sense. They are defined in an altogether different manner than functions, however, and they are treated differently during the compilation process.

**Example**

#include<iostream.h>  
#include<conio.h>  
#define area( r ) 3.14 \* r \* r

main()  
{  
cout<<area(2);

}  
**Result:**

12.56

**Lab Tasks:**

**Q1:** Develop a program in C++ that contains macro, that evaluate the volume of the sphere

Where v= 4/3(( \* r \* r \* r)

**Q2** : Develop a code that overloads the Sub (subtraction) function?

**QUESTION NUMBER : 01:**

Develop a program in C++ that contains macro, that evaluate the volume of the sphere

Where v= 4/3(( \* r \* r \* r)

**PROGRAM:**

#include<iostream>

using namespace std;

int main()

{

int rad,vol;

cout<<" Enter Radius of Sphere : ";

cin>>rad;

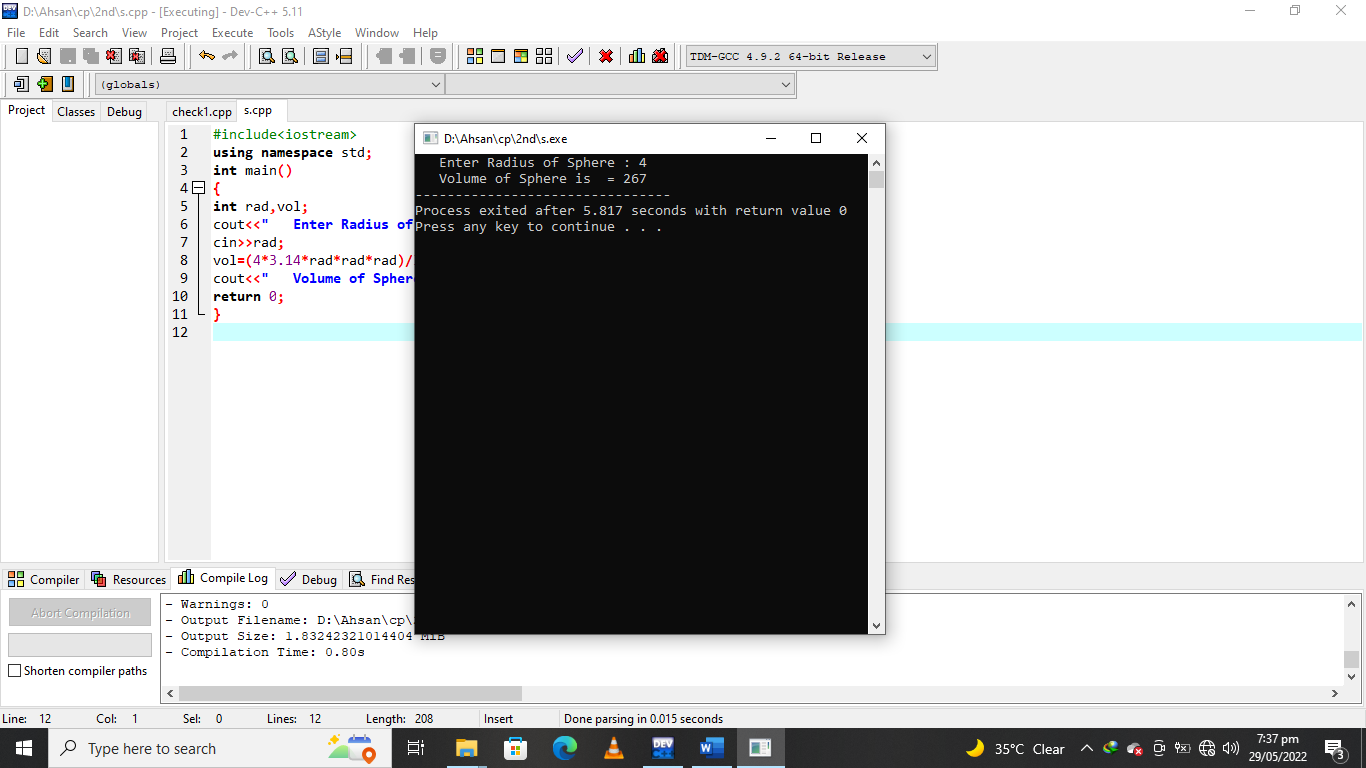
vol=(4\*3.14\*rad\*rad\*rad)/3;

cout<<" Volume of Sphere is = "<<vol;

return 0;

}

**OUTPUT:**



**QUESTION NUMBER : 02:**

Develop a code that overloads the Sub (subtraction) function?

**PROGRAM:**

#include <iostream>

using namespace std;

float absolute(float var){

if (var < 0.0)

var = -var;

return var;

}

int absolute(int var) {

if (var < 0)

var = -var;

return var;

}

int main() {

cout << "Absolute value of -5 = " << absolute(-5) << endl;

cout << "Absolute value of 5.5 = " << absolute(5.5f) << endl;

return 0;

}

**OUTPUT:**

