Classes and Methods

Chapter 2

A simple C++ program comparing with C

```
/*  C program that prints
    Hello BEE on screen */
#include <stdio.h>
 main ( )
{
    printf("Hello BEE \n");
    return 0;
}
```

```
/*  C++ program that prints
    Hello BEE on screen */
#include <iostream>
using namespace std;
 int main ( )
{
    cout<<"Hello BEE \n";
    return 0;
}
```

Introduction to C++

- Object oriented programming language
- Developed by Bjarne Stroustrup in 1979 at Bell Laboratories, New Jersey
- In 1998 ANSI/ISO standards committee provide the final draft of C++ • Object oriented extension of C i.e, addition of classes to C
- Initially known as C with classes
- later the name changed to C++ i.e, incremented version of C
- Almost all features of C is also applicable to C++

Basic Elements of C++

Basic vocabulary/elements used in C++ programs:

- Tokens
- Statements
- Variables
- Data Types

Tokens

Smallest individual units in a program which may be:

- keywords (reserved words that can’t use for anything else)
- identifiers (name given to variable, function etc. e.g. variable names, function names like “sum”, “main” ...)
- constants (which does not change e.g. the number 5)
- strings (sequence of chracters e.g. “Hello\n”)
- operators (symbol to indicate task e.g. +, -, =)
- punctuators (symbols e.g. ; ,{})
- whitespace (Spaces of various types; ignored by the compiler e.g. space, newline etc)

Statements

–unit of code that that perform task

–a basic building block of a program ends with semicolon

–e.g. cout << "Hello, world!\n";

Variables – Quantity which may vary (change)

– should declare at the beginning of the program before they are used

– Syntax for declaring variable is:

• Type-name variable name,…, variable name;

• Type- name refers data type

• Eg : int x, y, area;

```
/*Program to demonstrate structures*/
#include <stdio.h>

struct book {
        char name;
        float price;
        int pages;
        };

void main() {
        struct book b1={'A',23.5,5};
        struct book b2;
        b2.name='C';
         b2.price=45.0;
        b2.pages=23;

        printf("\n%c %.2f %d", b1.name, b1.price, b1.
pages);
        printf("\n%c %.2f %d", b2.name, b2.price, b2.
pages);

}
```

Limitations of Structures

- Does not allow the struct type to be treated like built-in (basic) data type
- Structures members are public To resolve these limitations, structures with OO approach called class was introduced in C++

Class

Class is user defined data type that is used to specify data representation and methods for manipulating the data in one package.

The data and functions within a class are called members of the class.

When you define a class, you define blueprint for a data type.

Class defines what an object of class will contain and what operations can be performed in it.

The data members and member functions can be grouped in private, public or protected section.

Review of Structure

- Collection of one or more than one variable, possibly of different data type, grouped together under a single name for convenient handling
- individual structure elements are referred to as members
- For accessing structures element/member dot (.) operator is used as: v_name.m_name

Syntax:

```
struct name {
        member 1;
        member 2;
        member 3;
        ……;
        member n;
        };

struct name v1,v2,…vn;
```

Class Declaration

- Similar to structure declaration
- Syntax:

class c_name {

access specifier/visibility labels:

variable declaration(data member);

function declaration(member function);

};

- Function need to be defined
- Access specifier: A keyword(public,private,protected) that controls the access to members of the class

```
class book {

        private:
        int pages;
        float price;

        public:
        void getdata(int b,float c);
        void display();

};
```

Class:

- Syntax for defining a class;

```
class class_name
{
        access specifier:
                variable_declaration;
                function_declaration;
         access specifier:
                variable_declaration;
                function_declaration;
};
```

- Example:

```
class Student
{
        private:
                char name[20];
                int roll;
                int marks;
        public:
                void getDetails();
                void display();
};
```

Data hiding in C++

Data hiding is technique used in object oriented programming to hide object details i.e. data members to limit access to data and prevent them from unwanted manipulation

- There are three access specifiers:

i. private

ii. public

iii. protected

i. Private access specifier:

- If the class members are declared private, then they can be accesses by member functions of that class only.
- Data members are made private to prevent direct access from outside the class.

ii. Public access specifier:

- If the class members are declared public then they can be accessed from anywhere in the program.
- Member functions are usually public which is used to manipulate the data present in the class.

iii. Protected access specifier:

It is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any derived class of that class.

Creating the object:

Syntax:

class_name object_name;

Student s1; //this statement creates a variable s1 of type Student //the variable s1 is object of class Student

Student s1,s2,s3,s4; // defining multiple objects of class Student

Creating the object:

```
class Student {

private:

char name[20];

int roll;

int marks; public:

void getDetails(); void display();

}s1,s2;
```

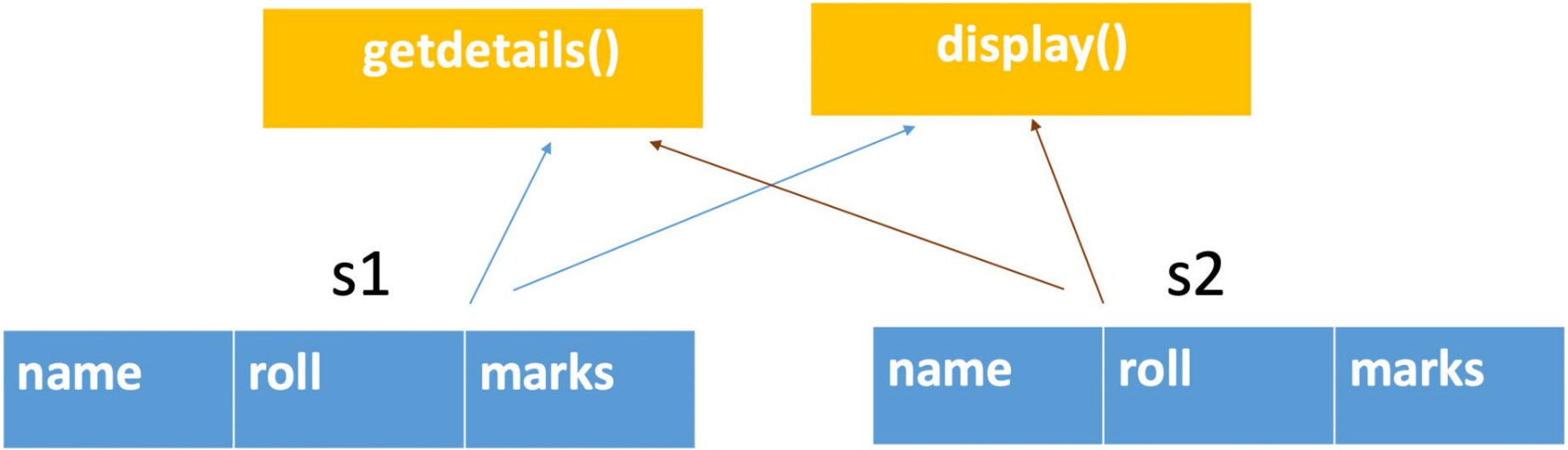
Creating the object:

```
class Student
{
    private:
        char name[20];
        int roll;
        int marks;
    public:
        void getDetails();
        void display();
};
```

```
void main()
    {
        Student s1,s2;
    ...
    }
```

Creating the object:

Student s1,s2;



Accessing Class Members

- The private members of a class can be accessed only through members of same class
- The public members of a class can be accessed from outside the class using object name and dot operator

Syntax for accessing public data member:

```
object_name.data_member;
```

Syntax for accessing public member functions:

```
object_name.function_name(arguments);
```

Accessing class members: Example

```
class Student
{
    private:
        char name[20];
        int roll;
        int marks;
    public:
        void getDetails();
        void display();
};
```

```
void main()
{
        Student s1;
        s1.getDetails();
        s1.display();

        s1.roll=14578; //invalid as
roll is private member
```

Accessing class members: Example

```
class Student
{
    private:
        char name[20];
        int marks;
    public:
        int roll;
        void getDetails();
        void display();
};
```

```
void main()
{
    Student s1;
    s1.getDetails();
    s1.display();
```

s1.roll=14578; valid as roll is public member

Defining member function:

1. Inside the class body
2. Outside the class body

1. Defining member function inside class body

If we define member function inside class definition, it is inline function

Defining member function: inside class body

```
class Student
{
    private:
            char name[20];
            int roll;
            int marks;
    public:
            void getDetails()
            {
            cout<<"Enter details";
            cin>>name>>roll>>marks;
            }
            void display()
            {
            cout<<"The name is "<<name<<endl;
            cout<<"Roll is "<<roll<<endl;
            cout<<"Marks is "<<marks<<endl;
            }
};
```

Defining member function: outside class body

- The function prototype is defined within the class body and its detail definition is written outside class body
- If we define function outside class body, we use scope resolution **::** operator

Syntax:

```
return_type class_name :: function_name(arguments)
{
        function body
}
```

Defining member function: outside class body

```
class Student
{
    private:
            char name[20];
            int roll;
            int marks;
    public:
            void getDetails();
            void display();
    };
```

```
void Student :: getDetails()
 {
  cout<<“Enter details”;
  cin>>name>>roll>>marks;
  }
 void Student :: display()
 {
cout<<“The name is “<<name<<endl;
cout<<“Roll is “<<roll<<endl;
cout<<“Marks is “<<marks<<endl;
}
```

Example Program:

// class and object example program
//member function defined inside class body

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

class Student
{
        private:
                char name[20];
                int roll;
                int marks;
        public:
        void getDetails()
        {
                cout<<"Enter name of student : ";
                cin>>name;
                cout<<"Enter roll number : ";
                 cin>>roll;
                 cout<<"Enter marks obtained :";
                  cin>>marks;
        }
void display()
                {
                cout<<endl<<endl<<"The  Details are :"<<endl;
                cout<<"The  name is "<<name<<endl;
                cout<<"Roll is "<<roll<<endl;
                cout<<"Marks  is "<<marks<<endl;
                }
};
```

```
void main() {
Student S1,S2;
S1.getDetails();
cout<<endl;
S2.getDetails();
cout<<endl;
S1.display();
cout<<endl;
S2.display();
 getch();
}
```

```
class Student {
private:
char name[20]; int roll;
int marks;
public:
void getDetails();
 void display();
};
```

```
void Student:: getDetails() //member function definition outside class {

cout<<endl<<"Enter name of student : ";

 cin>>name;

cout<<"Enter roll number : ";

cin>>roll;

cout<<"Enter marks obtained :";

cin>>marks ;                }

void Student :: display() //member function definition outside class {

cout<<endl<<endl<<"The Details are :"<<endl; cout<<"The name is
"<<name<<endl; cout<<"Roll is "<<roll<<endl;

cout<<"Marks is "<<marks<<endl;

}
```

Practice:

Q1. Design a class called Person that contains appropriate members for storing name, age, gender, telephone number. Write member functions that can read and display these data.

Q2. Write a program to represent a Circle that has member functions to perform following tasks.

- Calculate area of circle
- Calculate perimeter of the circle

Q3. Create a class Point that represents a three dimensional coordinate system. Each object of Point should have coordinates x,y,z and methods to assign coordinates to the objects. Add a method to calculate the distance from origin and to the point (x,y,z). Define member functions outside the class body.

Nesting of member functions

A member function of a class can be called only by an object of that class using dot operator

- However, a member function can be called by using its name inside another member function of same class
- This is known as nesting of member function

Nesting of member function : Example

```
class Student
{
        private:
                char name[20];
                int roll;
                int marks;
        public:
                void studentDetails();
                void display();
};
```

```
void Student:: studentDetails()
{
..
 display();
}


 void main()
{
        Student S;
        S.studentDetails();
        ...
}
```

Array of Objects

Syntax for declaring array of objects:

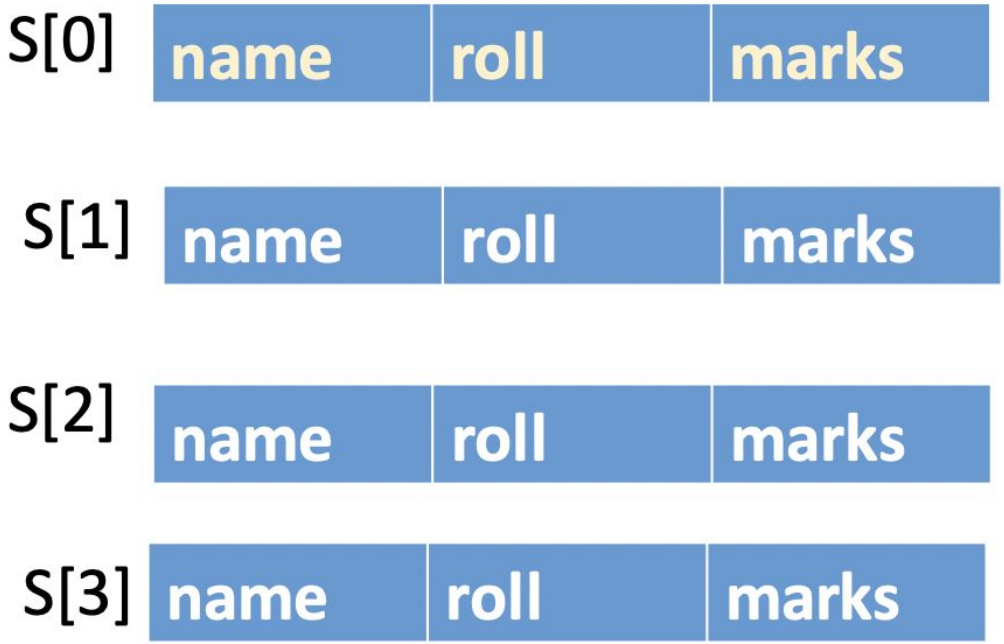
class_name object_name[size];

Example:

Student S[50];

Array of Objects : Example Program

Student S[4];



Example

Write the same above code for students and create the objects using array.

```
Students s[50];

cout<<”Enter no. of Students:”;

cin>>n;
```

```
void Student:: getDetails(int x) {

cout<<endl<<"Enter details of student "<<x<<" : "<<endl;

cout<<"Name : “;

cin>>name;

cout<<"Roll number : ";

cin>>roll;

cout<<"Marks obtained :";

cin>>marks;

cout<<endl<<endl;            }
```

```
void main() {
Student S[50];
int i,n;
cout<<"Enter number of Students : ";
cin>>n;
cout<<endl<<endl<<"Enter details of students"<<endl;
 for(i=0;i<n;i++){
S[i].getDetails(i+1);
}
cout<<endl<<endl<<"Details of student are :"<<endl;
for(i=0;i<n;i++)
{
S[i].display();

}
}
```

Enumeration

C++ allows programmers to create their own data type called enumerated type

Syntax:

enum enumerated_datatype { Enumerator1, Enumerator2,…. };

Example:

```
enum days {
    sunday, monday, tuesday, wednesday, thursday, friday
    };
```

```
#include<iostream>
using namespace std;
int main()
{
enum days
{
sun,mon,tues,wed,thur,fri,sat
};
days day1,day2;
day1=sun;
day2=thur;
cout<<"day 1= "<< day1 <<endl<<"day 2= "<< day2;
}
```

Practice:

Q1. Write a program to define a enumerated data type Month with name of 12 months. Assign first month as 1 and display the integer value assigned to the months.

```
#include <iostream>

using namespace std;
enum month {January,February, March, April, May,June, July,
August,September, October, November, December};
int main()
{
    int i;
    for(i=January;i<December;i++)
    {
        cout<<i<<endl;
    }

    return 0;
}
```

Inline function

Inline function is a function that is expanded in line when it is called

When the inline function is called whole code of the inline function gets inserted at the point of inline function call

Syntax:

```
inline returnType functionName( parameters) {
        function body;
        }
```

Program to calculate area of rectangle using inline function.

```
#include<iostream>
using namespace std;
inline int area(int l,int b){
return (l*b);
}
int main()
{
int len,bre,result;
cout<<“Enter values of length and breadth”;
cin>>len>>bre;
result=area(len,bre);
cout<<endl<<“The area of rectangle is “<<result;
}
```

Inline function

Compiler can ignore the request for inlining in following cases:

- If function contains loop
- If function contains static variables
- If function is recursive
- If the function is large

Practice:

Q1. Write a program using inline function to calculate the square of a number.

Q2. Write a program to calculate volume of a cube. (vol=side*side*side)

Q3. Write a program using inline functions to calculate the multiplication and division of two user input numbers.

Default Argument

In C++, we can call function without specifying all its arguments by assigning default values for some arguments

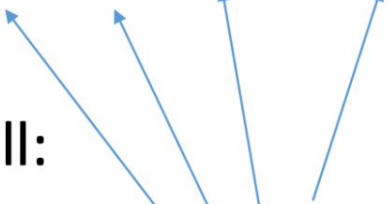
Syntax for declaring function with default argument:

returnType functionName(argument1, argument2=default_value, argument3=default_value);

Default argument

Function prototype:

```
void test(int a,int b,int c=0,int d=10);
```



Function call:

```
test(w,x,y,z);
```

Function prototype:

```
void test(int a,int b,int c=0,int d=10);
```

Function call:

```
test(w,x,y);
```

Default Argument:

Function Prototype:

returntype functionName (argument1, argument2=defaultValue, argument3=defaultValue);

default values must be defined from right to left

- int testFunction(int a=5, int b=10, int c=20) ; //valid
- int testFunction(int a, int b=10, int c=20) ; //valid

- int testFunction(int a=5, int b, int c=20) ; //invalid
- int testFunction(int a, int b=10, int c) ; //invalid
- int testFunction(int a=5, int b, int c) ; //invalid

Example:

```
void add(int a, int b, int c=0, int d=0);
void main()
        {
                int x,y,z,w;
                /*ask user and input values
              for x, y and z*/
                add(x,y);
                add(x,y,z);
                add(x,y,z,w);

        getch();
        }
void add( int a, int b, int c, int d)
        {
                int sum=a+b+c+d;
                cout<<endl<<"The sum
                is "<<sum;
        }
```

Practice:

Q1. Write a program to calculate simple interest using default value of r=1.5%. Ask the user for principal amount and time [SI=PTR/100]

Function overloading

Function overloading is a feature in C++ where two or more functions can have the same name but can do different operations

• Two or more functions can have same name but different in their argument list

• We can overload function by either making

i. type of arguments different

ii. making number of argument different

• The correct function is invoked whose argument list matches the arguments in the function call

Function overloading example

Example:

```
int calculate(int ); //function1

float calculate(float ); //function2

void calculate( int, int ); //function3

int calculate(int, int, int); //function4

float calculate(int ,float); //function5

float calculate(float, int); //function6
```

Function Overloading:

- Example:
- Declaration:

```
int add(int a,int b); //function1
int add(int a, int b, int c); //function2
double add(double x , double y); //function3
double add(int p ,double q); //function4
double add(double a, int b); //function5
```

- Function call:

```
cout<<add(5,10);
cout<<add(100,100.5);
cout<<add(5.5,10);
cout<<add(12,23,56);
cout<<add(10.5,5.6);
```

```
#include<iostream>
using namespace std;
int add(int a,int b)
{
cout<<"Function adds two integer"<<endl;
return (a+b);
}
int add(int a,int b, int c)
{
cout<<endl<<"Function adds three integer"<<endl;
return (a+b+c);
}
double add(double a,double b)
{
cout<<endl<<"Function adds two double
values"<<endl;
return (a+b);
}
double add(double a,int b)
{
cout<<endl<<"Function adds double and
int"<<endl;
return (a+b);
}
double add(int a,double b)
{
cout<<endl<<"Function adds int and
double"<<endl;
return (a+b);
}
int main()
{
cout<<"Result is "<<add(2,10)<<endl<<endl;
cout<<"Result is "<<add(10,10,20)<<endl;
cout<<"Result is "<<add(10.5,20.52)<<endl;
cout<<"Result is "<<add(10.5,30)<<endl;
cout<<"Result is "<<add(35,70.5);
}
```

Practice:

Q1. Write a program to find maximum of 2 numbers and maximum of 3 numbers using same function name, maximum().

Q2. Write a program to find the volume of 3 objects :cube, cylinder and Rectangular box using same function name, volume().

Q3. Write a program to find the area of cube, cylinder and rectangle using concept of function overloading.

Reference variable

- Reference variable is another name for an already existing variable
- When a variable is declared as reference, it becomes an alternative name for an existing variable
- A variable can be declared as reference by putting ‘&’ in the declaration

Syntax:

datatype &referenceVariableName= VariableName;

int a;

int &ref=a;

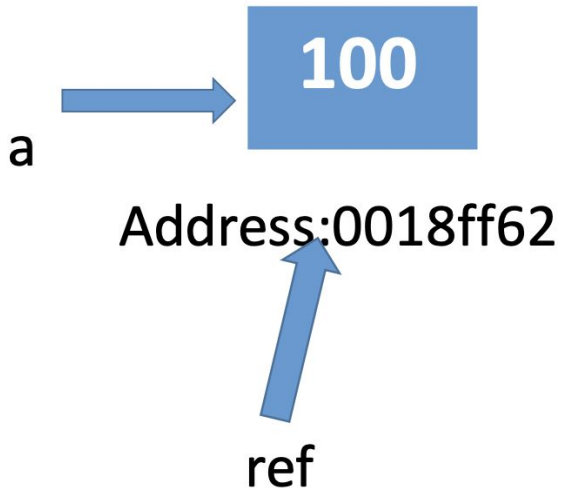
Reference Variable:

Syntax:

datatype &referenceVariableName= VaraibleName;

Example:

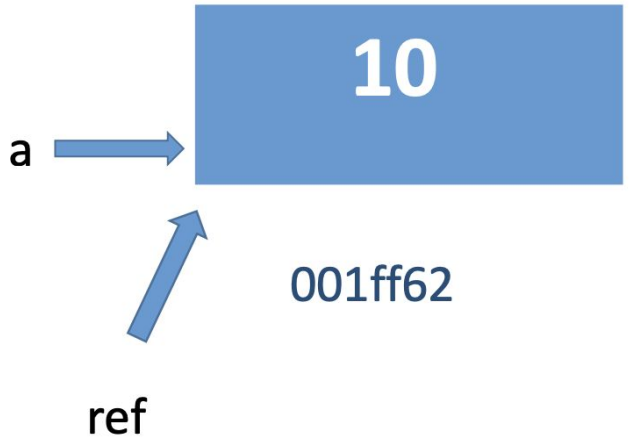
int a=100;



int &ref=a;

Example:

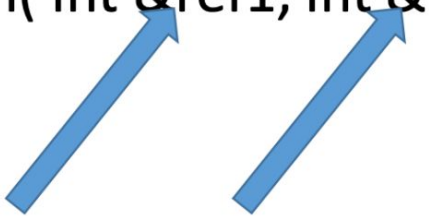
```
#include<iostream.h>
void main()
{
        int a=10;
        int &ref=a;
        cout<<"a="<<a<<endl;
         ref=55;
         cout<<"a="<<a<<endl;
        a=99;
         cout<<"ref="<<ref<<endl;
        cout<<"address of ref ="<<&ref<<endl;
         cout<<"address of a ="<<&a<<endl;

}
```



Call by Reference:

Function Prototype:

```
void testFunction( int &ref1, int &ref2);
```



Function Call:

```
testFunction (  a     ,    b  );
```

When you pass parameters by reference, a new storage location is not created for these parameters. The reference parameters represent the same memory location as the actual parameters that are supplied to the method.

Here, ref1 will be alternative name of a and ref2 will be alternative name of b.

Example Program:

```
#include<iostream.h>
#include<conio.h>
void swap(int &x, int &y);
void main ()
{
        int a = 10;
         int b = 20;
        cout << "Before swap, value of a :" <<a << endl;
        cout << "Before swap, value of b :" <<b << endl;
        swap(a, b);
        cout << "After swap, value of a :" << a << endl;
         cout << "After swap, value of b :" << b << endl;
        getch();
}
```

```
void swap(int &x, int &y)
{
        int temp;
         temp = x;
        x = y;
        y = temp;
}
```

Practice:

Q1. Write and test the following computeSphere() function that returns the volume “v” and surface area “s” of a sphere with the given radius. void computeSphere(float &v, float &s, float r)

```
#include <iostream>

using namespace std;
class Rational{
    private:
    int num;
    int den;
    public:
    void input(){
        cout<<"Enter numerator and denominator";
        cin>>num>>den;
    }
    void display(){
        cout<<num<<"/"<<den<<endl;
    }
    void sum(Rational r1, Rational r2){
        num = r1.num+r2.num;
        den= r1.den * r2.den;
    //    cout<<"The sum is "<<num<<"/"<<den<<endl;
    }
    void inverse(){
        int temp;
        temp=num;
```