PROBLEM SOLVING AND PROGRAMMING

User defined data types - class

CLASS

- Another data type ? Why ?
  - Primitive data types (int, float, char, ..) for storing any type of data.
  - Arrays for storing collection of homogeneous data items under a common identifier.
  - Structures for storing the collection of heterogeneous data items under a common identifier.
- All these data types are sufficient to model any type of real world data ?
  - Yes
- What is additionally needed – for storing data items.

Complex numbers

- How to represent a complex number?
- As two independent numbers – real, imag.
- A structure with two numbers as members
  - struct complex { float real; float imag;} a, b;
  - a.real = 4.5; a.imag = 2.0;
- In both the cases how to add, subtract or multiply two complex numbers.
- Write functions for each of these operations.
- There is no direct relationship between the definition of structure complex and the functions performing the operations.

Complex numbers

- How can the data definitions and the operations to manipulate the data be combined?
- Do we know the process of adding two integers or two floating point numbers?
- Integers or floating point numbers are called built-in data types. Their representation as well as the implementation of the operations are generally not known.
- A structure definition for complex number is called user defined data types.
- For these user defined data types – operations or methods have to be defined by the users.

An **abstract data type** is a **type** with associated operations, but whose representation is hidden.

Abstract Data Type(ADT) is a data type, where only behavior is defined but not implementation.

Process of defining ADTs

Built-in data types

Float, int, char, double etc.

Process of defining ADTs

Built-in data types

Float, int, char, double etc.

Constructs to define new ones:

User defined data types

Student, complex num etc using struct

Process of defining ADTs

Built-in data types

Float, int, char, double etc.

Constructs to define new ones:

User defined data types

Student, complex num etc using struct

Operations:

Defining Processes

Functions for Operations – member functions

Process of defining ADTs

Built-in data types

Float, int, char, double etc.

Constructs to define new ones:

User defined data types

Student, complex num etc using struct

Operations:

Defining Processes

Functions for Operations – member functions

Put things together:

Encapsulation

Bundling things together

Process of defining ADTs

Built-in data types

Float, int, char, double etc.

Constructs to define new ones:

User defined data types

Student, complex num etc using struct

Operations:

Defining Processes

Functions for Operations – member functions

Put things together:

Encapsulation

Bundling things together

Define ADTs:

Abstract Data Types

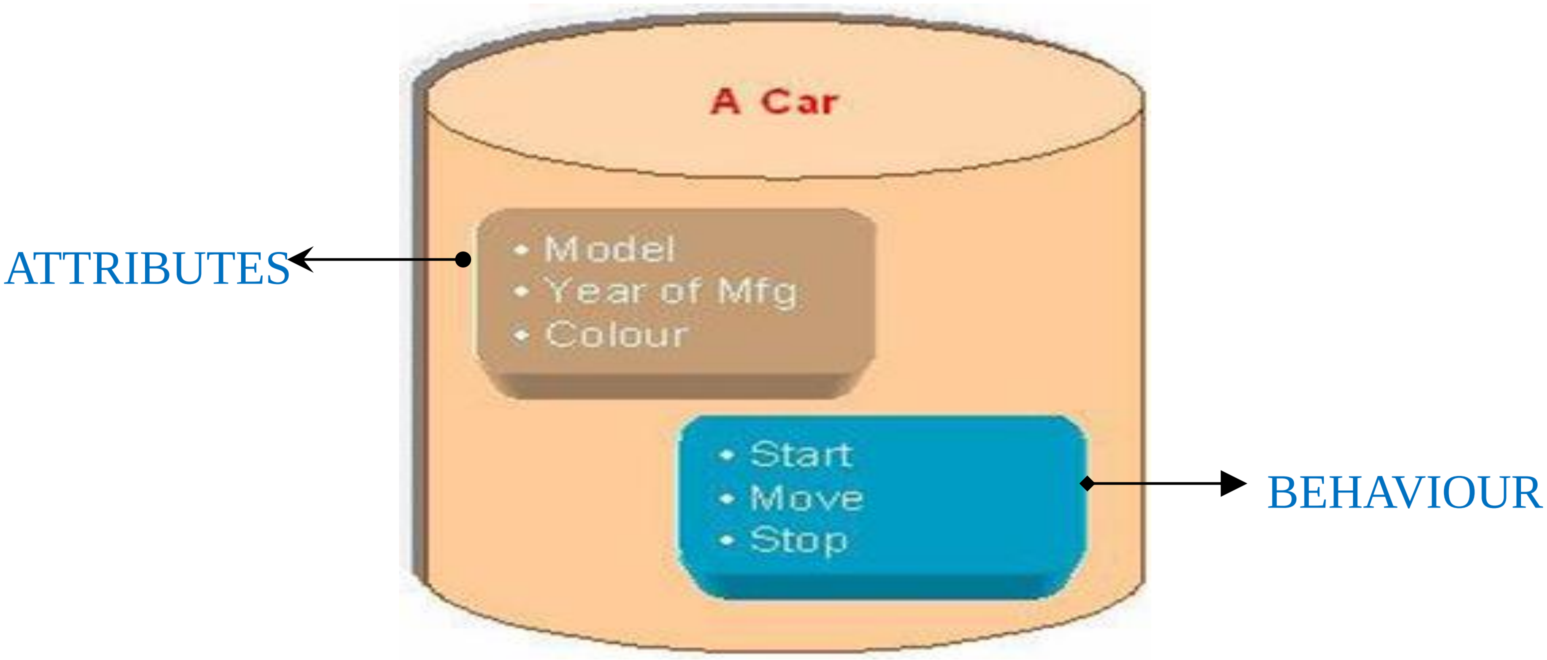
Class and access specifiers

CLASS

- Class in C++ programming language is a mechanism to define a user defined data type achieving:
  - Definition of data items – members
  - Definition of associated operations as functions – member functions
  - Putting the data definition and member functions together – encapsulation
  - Possible to hide data definitions and / or implementation details of member functions – information hiding.
- A user defined data type using class can be termed as Abstract Data Type (ADT).
- ADT is an abstraction of Data Type (built-in).
- A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

CLASS

- In The Real World Life Real world objects have two major things
  1. State/Attributes (what it is)
  2. Behavior/Properties/Actions (what it does)
- To simulate real world objects in software system, we can use ***Data Abstraction***
- Example – Consider a real world object car



CLASS

- A data abstraction is a simplified view of a real world object that
  - includes only features one is interested in (the operations of data object).
  - while hides away the unnecessary details (hides how these operations are implemented).

- Goals of new data type
  - Combining Attributes and Behaviors (Encapsulation).
  - Hiding unnecessary details (Information hiding).

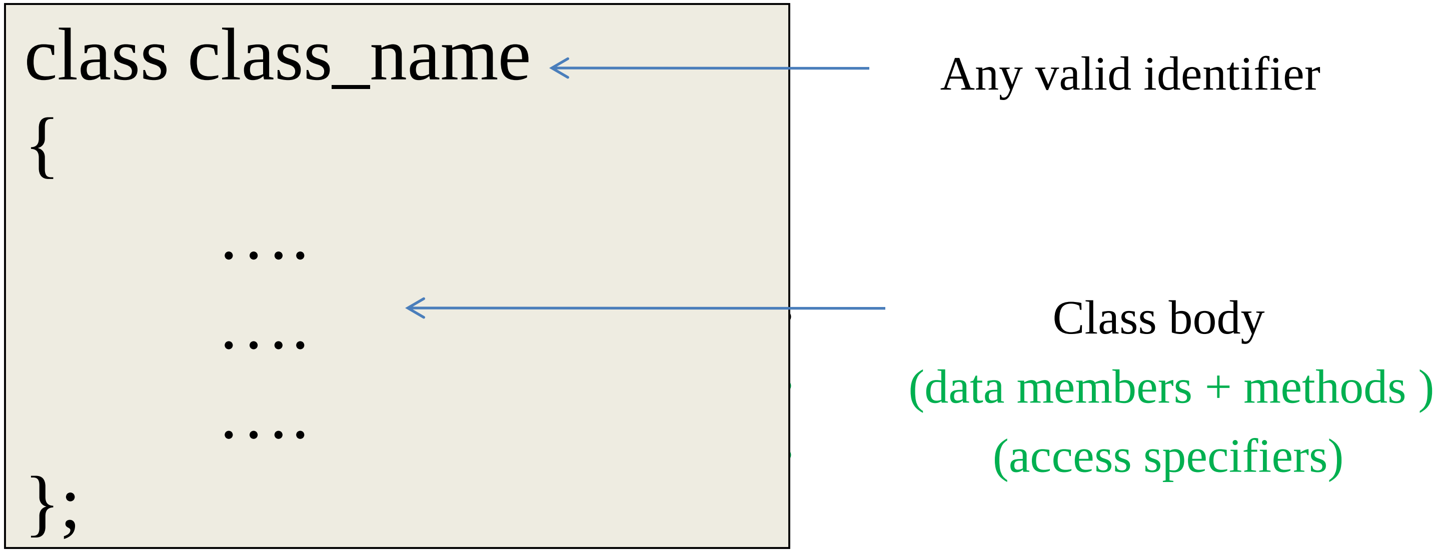
- Encapsulation - It is a mechanism that associates the code (Behavior) and the data (Attribute) it manipulates into a single unit to keeps them safe from external interference and misuse (C++ provides new data type class to achieve this).

- Information Hiding - Data hiding means to secure data or sometimes specific behaviors from direct access (C++ provides Access Specifies for this purpose).

CLASS

- Abstract Data Type (ADT) is the key to Object-Oriented programming. An ADT is a set of data together with the operations on the data .

- A class is often used to describe an ADT in C++. A class is also called as User-Defined Data Type (structures + functions).



```
class class_name
{
        ....
        ....
        ....
};
```

Any valid identifier

Class body

(data members + methods)

(access specifiers)

Access Specifiers

- The members / member functions of a class belongs to any one of the three predefined access specifiers.

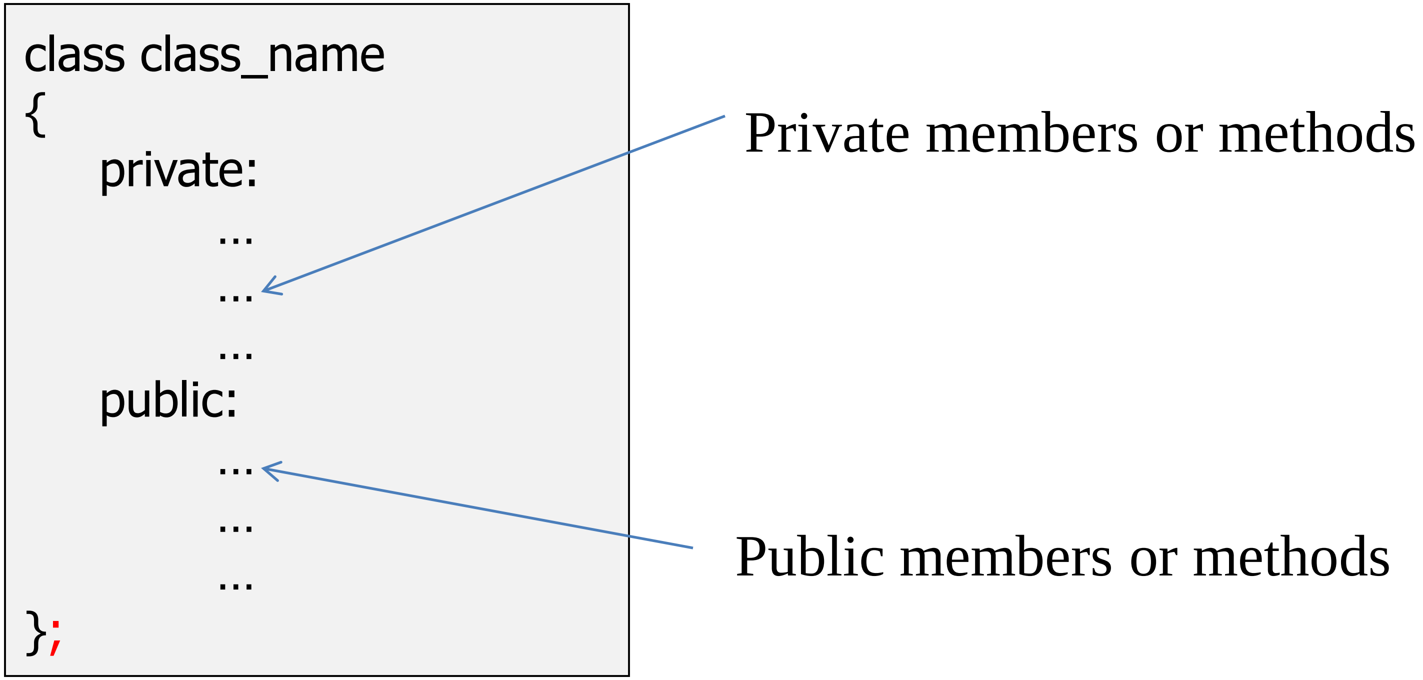
1. Private - A ***private* member** within a class denotes that only members of the same class have accessibility. The *private* member is inaccessible from outside the class (this is default case). Private members and methods are for internal use only.

2. Public - ***Public* members** are accessible from outside the class. (only through class object). ‘

3. Protected - A ***protected access specifier*** is a stage between *private* and *public* access. If member functions defined in a class are *protected*, they cannot be accessed from outside the class but can be accessed from the derived class.

CLASS

***Note* -** Usually, the data members of a class are declared in the *private* section of the class and the member functions are in *public* section of the class.



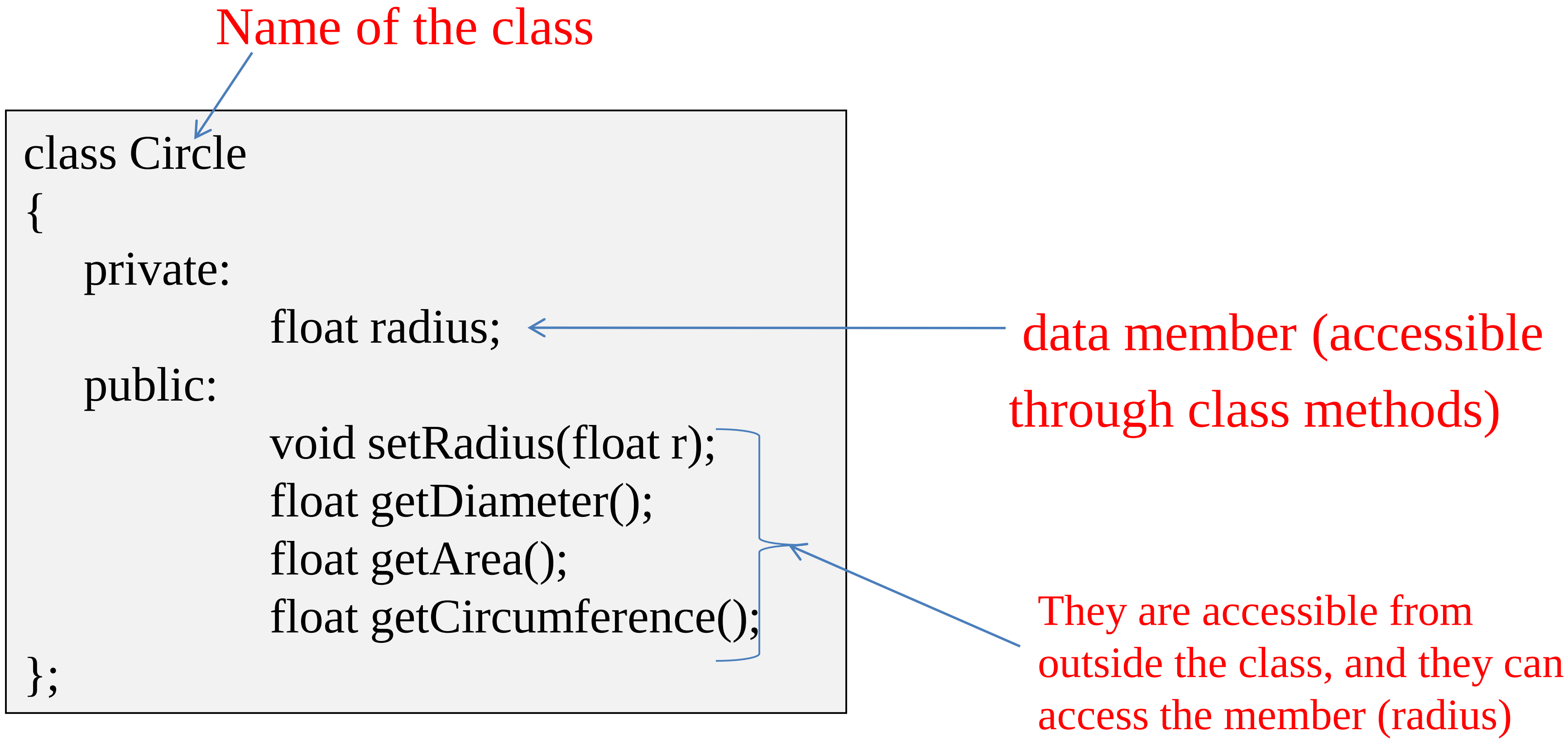
```
class class_name
{
    private:
            …
            …
            …
    public:
            …
            …
            …
};
```

Private members or methods

Public members or methods

Example

Example – Consider the real world entity ***circle.***



Name of the class

```
class Circle
{
    private:
                float radius;
    public:
                void setRadius(float r);
                float getDiameter();
                float getArea();
                float getCircumference();
};
```

data member (accessible through class methods)

They are accessible from outside the class, and they can access the member (radius)

This class example shows how we can encapsulate (gather) a circle information into one package (class)

Example

Example – Consider the real world entity ***Person.***

Name of the class

```
class Person
{
    private:
                char name[15];
                 Date DOB;
    public:
                void setDOB(int mm,
                                  int dd,
                                  int yy);
                int getAge();
};
```

data members (accessible through class methods)

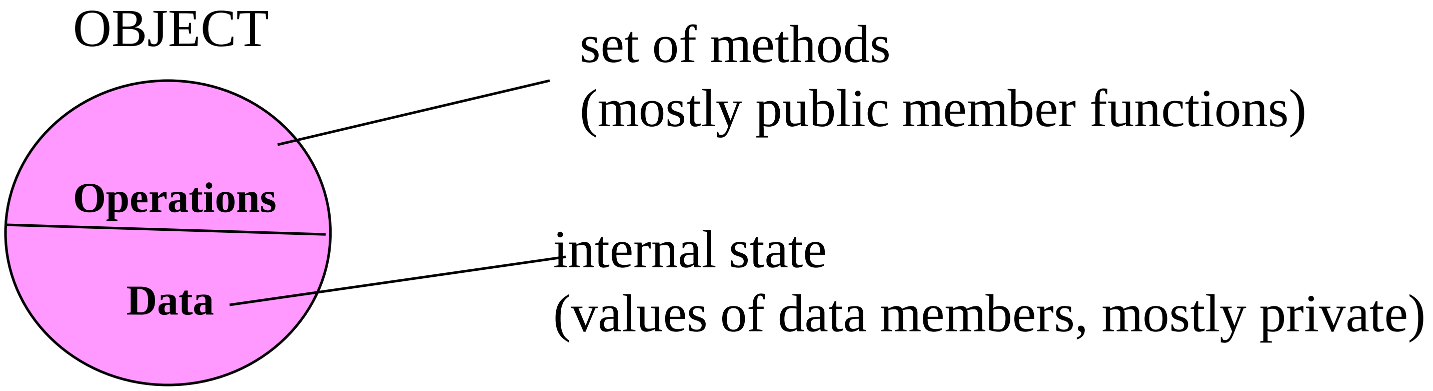
They are accessible from outside the class, and they can access the members (name, DOB)

Objects

- Instance of a class or variable of a class is an object.

- The existence of class is logical (no memory is allotted), but the existence of object is physical (memory will be allotted to object of a certain class is instantiated).

- You can instantiate many objects from a class type.
- Ex - Circle c; Circle *c;



OBJECT

Operations

Data

set of methods
(mostly public member functions)

internal state
(values of data members, mostly private)

Accessing Class Members

- Operators to access class members
  - Identical to those for **struct**s
  - Dot member selection operator (**.**)
    - Object
    - Reference to object
  - Arrow member selection operator (**->**)
    - Pointers

Special Member Functions

- Constructor: Are the methods of class used to initialize the data members of the class and have the following properties -
  - Same name as class
  - Public function member
  - called when a new object is created (instantiated).
  - Initialize data members.
  - No return type
  - Several constructors
  - Function overloading (same function name with different arguments)

Special Member Functions

- Destructor:
  - Same name as class but preceded with tilde (~) character
  - No arguments
  - No return value
  - Cannot be overloaded
  - Before system reclaims object’s memory
    - Reuse memory for new objects
    - Mainly used to de-allocate dynamic memory locations

A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.

A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Example - Special Member Functions

```
class Circle
{
    private:
                float radius;
    public:
                Circle();
                Circle(int r);
                void setRadius(float r);
                float getDiameter();
                float getArea();
                float getCircumference();
};
```

Constructor with no arguments

Constructor with one argument

Implementing class member functions(methods)

- ➢ Where do we define class methods ?
- ➢ There are two ways:

1. Member functions defined outside class
   - Using Binary scope resolution operator (::) "Ties" member name to class name
     - – Uniquely identifies functions of particular class
     - – Different classes can have member functions with same name
   - Format for defining member functions

```
Return Type Class Name::MemberFunctionName( )
{
        …
}
```

2. Member functions defined inside class
   - Do not need scope resolution operator or class name

Example - Definition of class member functions outside the class

```
class Circle
{
    private:
            float radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(float r){radius = r;}
            float getDiameter(){ return radius *2;}
            float getArea();
            float getCircumference();
};
Circle::Circle(int r)  {   radius = r;  }

float Circle::getArea() {return radius * radius * (22.0/7); }

float Circle:: getCircumference() { return 2 * radius  * (22.0/7); }
```

Example - Definition of class member functions inside the class

```
class Circle
{
    private:
                        float radius;
    public:
                        Circle() { radius = 0.0;}
                        Circle(int r);
                        void setRadius(float r){radius = r;}
                        float getDiameter(){ return radius *2;}
                        float getArea();
                        float getCircumference();
};
```

```
class Box {
  public:
    double length;   // Length of a box
    double breadth;  // Breadth of a box
    double height;   // Height of a box
};
int main() {
  Box Box1;        // Declare Box1 of type Box
  Box Box2;        // Declare Box2 of type Box
  double volume = 0.0;     // Store the volume of a box here
  Box1.height = 5.0;
  Box1.length = 6.0;
  Box1.breadth = 7.0;
  Box2.height = 10.0;
  Box2.length = 12.0;
  Box2.breadth = 13.0;
 volume = Box1.height * Box1.length * Box1.breadth;
  cout << "Volume of Box1 : " << volume <<endl;
  // volume of box 2
  volume = Box2.height * Box2.length * Box2.breadth;
  cout << "Volume of Box2 : " << volume <<endl;
  return 0;
}
```

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Example

```
class Circle
{
    private:
            float radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(float r){radius = r;}
            float getDiameter(){ return radius *2;}
            float getArea();
            float getCircumference();
};

Circle::Circle(int r)
{  radius = r; }

float Circle::getArea()
{ return radius * radius * (22.0/7);}

float Circle:: getCircumference()
{ return 2 * radius  * (22.0/7); }
```

```
int main()
{
   Circle c1,c2(7);

   cout<<“The area of c1:” << c1.getArea()<<“\n”;

   //c1.raduis = 5;//syntax error
   c1.setRadius(5);

   cout<<“The circumference of c1:”
           << c1.getCircumference()<<“\n”;

   cout<<“The Diameter of c2:”
           <<c2.getDiameter()<<“\n”;
   return 0;
}
```

The second constructor is called

Since radius is a private class data member

```
class Circle
{
    private:
            float radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(float r){radius = r;}
            float getDiameter(){ return radius *2;}
            float getArea();
            float getCircumference();
};

Circle::Circle(int r)
{  radius = r; }

float Circle::getArea()
{ return radius * radius * (22.0/7);}

float Circle:: getCircumference()
{ return 2 * radius  * (22.0/7); }
```

```
the area of c1
0
31.4286
14
```

E
x
a
m
p
l
e

The second constructor is called

Since radius is a private class data member

```
int main()
{
   Circle c1,c2(7);

   cout<<"The area of c1:"<< c1.getArea()<<"\n";

   //c1.raduis = 5;//syntax error
   c1.setRadius(5);

   cout<<"The circumference of c1:"
            << c1.getCircumference()<<"\n";

   cout<<"The Diameter of c2:"
            <<c2.getDiameter()<<"\n";
   return 0;
}
```

Example

```
class Circle
{
    private:

            float radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(float r){radius = r;}
            float getDiameter(){ return radius *2;}
            float getArea();
            float getCircumference();
}; Circle::Circle(int r)
{   radius = r; }
float Circle::getArea()
{  return radius * radius * (22.0/7); }
float Circle:: getCircumference()
{ return 2 * radius  * (22.0/7); }
```

```
int main()
{Circle c(7); // object c is created
  Circle *cp1 = &c; // pointer cp1 points to object c
  Circle *cp2 = new Circle(8);
  // new object created (un named) and assigned to a pointer
  cout <<“The area of c:”<< c.getArea()<<endl;
  cout <<“The area of cp1:”<<cp1->getArea();
  cout<<“The area of cp2:”<<cp2->getArea();
}
```

```
154
154
201.143
```

```
 class Line
 {
 public:
 void setLength( double len );
 double getLength( void );
 Line(); // This is the constructor declaration
 ~Line(); // This is the destructor: declaration
 private:
 double length;
 };
```

```
 // Main function for the program
int main()
{
Line line;
// set line length
line.setLength(6.0);
cout << "Length of line : " << line.getLength()
<<endl;
return 0;
}
```

```
// Member functions definitions including constructor
Line::Line(void)
 {
   cout << "Object is being created" << endl;
}
Line::~Line(void)
{
cout << "Object is being deleted" << endl; }
void Line::setLength( double len )
{
 length = len; }
double Line::getLength( void )
 { return length; }
```

```
Object is being created
 Length of line : 6
Object is being deleted
```

```
class A
{
public:
A()
{
cout << "Constructor" << endl;
}
~A()
{
cout << "Destructor" << endl;
}
};
int main()
{
 A* a = new A[4];
delete [] a; // Delete array
return 0;
}
```

```
Constructor
Constructor
Constructor
Constructor
Destructor
Destructor
Destructor
Destructor
```

Example

```
class Time
{
   private:
               int *hour,*minute,*second;
   public:
               Time();
               Time(int h, int m, int s);
               void printTime();
               void setTime(int h, int m, int s);
               int getHour(){return *hour;}
               int getMinute(){return *minute;}
               int getSecond(){return *second;}
               void setHour(int h){*hour = h;}
               void setMinute(int m){*minute = m;}
               void setSecond(int s){*second = s;}
               ~Time();
};
```

Destructor

Example

```
Time :: Time()
{ hour = new int;
  minute = new int;
  second = new int;
 *hour  = *minute = *second = 0;
}

Time :: Time(int h, int m, int s)
{
    hour = new int;
   minute = new int;
   second = new int;
   *hour = h;
   *minute = m;
  *second = s;
}
void Time :: setTime(int h, int m, int s)
{*hour = h;
 *minute = m;
 *second = s;
}
```

Dynamic locations should be allocated to pointers first

Example

```
void Time :: printTime()
{cout<<"The time is : ("<<*hour<<":"<<*minute<<":"<<*second<<")">>endl; }

Time::~Time()
{  delete hour;      delete minute;       delete second;  }
```

Destructor: used here to de-allocate memory locations

```
int main()
{           Time *t;
            t= new Time(3,15,15);
            t->printTime();

            t->setHour(19);
            t->setMinute(17);
            t->setSecond(43);

            t->printTime();
            delete t;
             return 0;
}
```

When executed, the destructor is called

```
Output:
The time is : (3:15:15)
The time is : (19:17:43)
Press any key to continue
```

Example-Write the program for performing operations on complex number

```
class Complex
{private:
              float real;
              float imaginary;
  public:
             Complex(){imaginary = 0.0; real = 0.0;};
             Complex( float x, float y) {real=x; imaginary = y;};
             void setComplex(float x, float y) {real=x; imaginary = y;}
             void addComplex(Complex C1, Complex C2);
             void mulComplex(Complex C1, Complex C2);
             void displayComplex();
};
                              // Write the Complete the program
```

Reasons for CLASS (ADT / OOP)

1. Simplify programming

2. Interfaces
   - Information hiding:
   - Implementation details hidden within classes themselves

3. Software reuse
   - Class objects included as members of other classes