Object Oriented & Programming Language

Lab#03

Table of Contents

Classes Constructor

A **class constructor** is a special member function of a **class** that is executed whenever we create new objects of that **class**. A **constructor** will have exact same name as the **class** and it does not have any return type at all, not even void.

Default Constructor

Given example demonstrates the concept of default constructor.

```
#include <string>
#include <iostream>
using namespace std;
class ClassRoom {
private:
int roomID;
int numberOfChairs;
char boardType; // C for chalk and M for marker
string multimedia;
string remarks;
public:
	ClassRoom();
	void setroomID(int);
	void setnumberOfChairs(int);
	void setboardType(char);
	void setmultimedia(string);
	void setremarks(string);
	int getroomID();
	int getnumberOfChairs();
	char getboardType();
	string getmultimedia();
	string getremarks();
	void display();
};
void main ()
{
	ClassRoom CR0;
	CR0.display();
	system("PAUSE");
}
//--------------------------------------Constructors
ClassRoom::ClassRoom()
{
	this->roomID=0;
	this->numberOfChairs=0;
	this->boardType='M';
```

```
	this->multimedia="New";
	this->remarks="Default Constructor";
}
//-------------------------------------Getter Functions
int ClassRoom::getroomID()
{
	return this->roomID;
}
int ClassRoom::getnumberOfChairs()
{
	return this->numberOfChairs;
}
char ClassRoom::getboardType()
{
	return (this->boardType);
}
string ClassRoom::getmultimedia()
{
	return this->multimedia;
}

string ClassRoom::getremarks()
{
	return this->remarks;
}
//-------------------------------------Printing Functions
void ClassRoom::display()
{
	cout<<"Room ID \t: "<<this->getroomID()<<endl;
	cout<<"N.O. Chairs \t: "<<this->getnumberOfChairs()<<endl;
	cout<<"Board Type \t: "<<this->getboardType()<<endl;
	cout<<"Multimedia \t: "<<this->getmultimedia()<<endl;
	cout<<"Remarks \t: \n"<<this->getremarks()<<endl;
	cout<<"---------------------------------"<<endl;
}

/*
Room ID         : 0
N.O. Chairs     : 0
Board Type      : M
Multimedia      : New
Remarks         :
Default Constructor
---------------------------------
Press any key to continue . . .
*/
```

Parameterized Constructor

Given example demonstrates the concept of parameterized constructor.

```
#include <string>
#include <iostream>
using namespace std;
class ClassRoom {
private:
int roomID;
int numberOfChairs;
char boardType; // C for chalk and M for marker
string multimedia;
string remarks;
public:
	ClassRoom(int id,int NOC,char,string mul, string rmks);
	void setroomID(int);
	void setnumberOfChairs(int);
	void setboardType(char);
	void setmultimedia(string);
	void setremarks(string);
	int getroomID();
	int getnumberOfChairs();
	char getboardType();
	string getmultimedia();
	string getremarks();
	void display();
};
void main ()
{
	ClassRoom CR(11,25,'M',"Repairing..","Remarks added from Main");
	CR.display();
	system("PAUSE");
}
//--------------------------------------Constructors
ClassRoom::ClassRoom(int id,int NOC,char c,string mul, string remks)
{
	this->roomID=id;
	this->numberOfChairs=NOC;
	this->boardType=c;
	this->setmultimedia(mul);
	this->setremarks(remks+"\nConstructor with All(5)Parameters
(ID,NOC,BoardType,Multimedia,Remarks)");
}
//--------------------------------------Setter Functions
void ClassRoom::setroomID(int id)
{
	this->roomID=id;
}
```

```
void ClassRoom::setnumberOfChairs(int NOC)
{
	this->numberOfChairs=NOC;
}
void ClassRoom::setboardType(char c)
{
	this->boardType=c;
}
void ClassRoom::setmultimedia(string str)
{
	this->multimedia=str;
}
void ClassRoom::setremarks(string str)
{
	this->remarks=str;
}
//-------------------------------------Getter Functions
int ClassRoom::getroomID()
{
	return this->roomID;
}
int ClassRoom::getnumberOfChairs()
{
	return this->numberOfChairs;
}
char ClassRoom::getboardType()
{
	return (this->boardType);
}
string ClassRoom::getmultimedia()
{
	return this->multimedia;
}

string ClassRoom::getremarks()
{
	return this->remarks;
}
//-----------------------------------Printing Functions
void ClassRoom::display()
{
	cout<<"Room ID \t: "<<this->getroomID()<<endl;
	cout<<"N.O. Chairs \t: "<<this->getnumberOfChairs()<<endl;
	cout<<"Board Type \t: "<<this->getboardType()<<endl;
	cout<<"Multimedia \t: "<<this->getmultimedia()<<endl;
	cout<<"Remarks \t: \n"<<this->getremarks()<<endl;
	cout<<"---------------------------------"<<endl;
}
/*
Room ID         : 11
```

```
N.O. Chairs     : 25
Board Type      : M
Multimedia      : Repairing..
Remarks         :
Remarks added from Main
Constructor with All(5)Parameters (ID,NOC,BoardType,Multimedia,Remarks)
--------------------------------
Press any key to continue . . .


*/
```

Overloaded Constructor

Given example demonstrates the concept of overloaded constructor

```
#include <string>
#include <iostream>
using namespace std;
class ClassRoom {
private:
int roomID;
int numberOfChairs;
char boardType; // C for chalk and M for marker
string multimedia;
string remarks;
public:
ClassRoom();
ClassRoom(int id);
ClassRoom(int id,int NOC);
ClassRoom(int id,int NOC,char c);
ClassRoom(int id,int NOC,char,string mul);
ClassRoom(int id,int NOC,char,string mul, string rmks);
void ClassRoom::display()
{
cout<<"Room ID \t: "<<this->getroomID()<<endl;
cout<<"N.O. Chairs \t: "<<this->getnumberOfChairs()<<endl;
cout<<"Board Type \t: "<<this->getboardType()<<endl;
cout<<"Multimedia \t: "<<this->getmultimedia()<<endl;
cout<<"Remarks \t: \n"<<this->getremarks()<<endl;
cout<<"--------------------------------"<<endl;
}
};
void main ()
{
ClassRoom CR0,CR1(1),CR2(2,30),CR3(3,35,'M'),
CR4(4,40,'M',"Repairing.."),CR5(5,40,'M',"Repairing..","Remarks added from Main");
CR0.display();
CR1.display();
CR2.display();
CR3.display();
CR4.display();
CR5.display();
system("PAUSE");
```

```
}
//-------------------------------------Constructors
ClassRoom::ClassRoom()
{
	this->setmultimedia("New");
this->remarks="Constructor with 1-Parameter (ID) ";
}
ClassRoom::ClassRoom(int id,int NOC)
{
this->roomID=id;
this->numberOfChairs=NOC;
this->setboardType('M');
this->setmultimedia("New");
this->remarks="Constructor with 2-Parameters (ID,NOC) ";
}
ClassRoom::ClassRoom(int id,int NOC,char c)
{
this->roomID=id;
this->numberOfChairs=NOC;
this->boardType=c;
this->setmultimedia("New");
this->remarks="Constructor with 3-Parameters (ID,NOC,BoardType) ";
}
ClassRoom::ClassRoom(int id,int NOC,char c,string mul)
{
this->setroomID(id); //this->roomID=id;
this->setnumberOfChairs(NOC); //this->numberOfChairs=NOC;
this->setboardType(c); //this->boardType=c;
this->setmultimedia(mul); //this->multimedia=str;
this->setremarks("Constructor with 4-Parameters (ID,NOC,BoardType,Multimedia)");
}
ClassRoom::ClassRoom(int id,int NOC,char c,string mul, string remks)
{
this->setroomID(id); //this->roomID=id;
this->setnumberOfChairs(NOC); //this->numberOfChairs=NOC;
this->setboardType(c); //this->boardType=c;
this->setmultimedia(mul); //this->multimedia=str;
this->setremarks(remks+"\nConstructor with All(5)Parameters
(ID,NOC,BoardType,Multimedia,Remarks)");
}
}
this->roomID=0;
this->numberOfChairs=0;
this->boardType='M';
this->multimedia="New";
this->remarks="Default Constructor";
}
ClassRoom::ClassRoom(int id)
{
this->roomID=id;
this->setnumberOfChairs(0);
this->setbo          rdType('M');
this->setmultimedia("New");
this->remarks="Constructor with 1-Parameter (ID) ";
}
ClassRoom::ClassRoom(int id,int NOC)
{
this->roomID=id;
```

```
this->numberOfChairs=NOC;
this->setboardType('M');
this->setmultimedia("New");
this->remarks="Constructor with 2-Parameters (ID,NOC) ";
}
ClassRoom::ClassRoom(int id,int NOC,char c)
{
this->roomID=id;
this->numberOfChairs=NOC;
this->boardType=c;
this->setmultimedia("New");
this->remarks="Constructor with 3-Parameters (ID,NOC,BoardType) ";
}
ClassRoom::ClassRoom(int id,int NOC,char c,string mul)
{
this->setroomID(id); //this->roomID=id;
this->setnumberOfChairs(NOC); //this->numberOfChairs=NOC;
this->setboardType(c); //this->boardType=c;
this->setmultimedia(mul); //this->multimedia=str;
this->setremarks("Constructor with 4-Parameters (ID,NOC,BoardType,Multimedia)");
}
ClassRoom::ClassRoom(int id,int NOC,char c,string mul, string remks)
{
this->setroomID(id); //this->roomID=id;
this->setnumberOfChairs(NOC); //this->numberOfChairs=NOC;
this->setboardType(c); //this->boardType=c;
this->setmultimedia(mul); //this->multimedia=str;
this->setremarks(remks+"\nConstructor with All(5)Parameters
(ID,NOC,BoardType,Multimedia,Remarks)");
}
/*
Room ID : 0
N.O. Chairs : 0
Board Type : M
Multimedia : New
Remarks :
Default Constructor
---------------------------------
Room ID : 1
N.O. Chairs : 0
Board Type : M
Multimedia : New
Remarks :
Constructor with 1-Parameter (ID)
---------------------------------
Room ID : 2
N.O. Chairs : 30
Board Type : M
Multimedia : New
Remarks :
Constructor with 2-Parameters (ID,NOC)
---------------------------------
Room ID : 3
N.O. Chairs : 35
Board Type : M
Multimedia : New
Remarks :
Constructor with 3-Parameters (ID,NOC,BoardType)
```

```
---------------------------------
Room ID : 4
N.O. Chairs : 40
Board Type : M
Multimedia : Repairing..
Remarks :
Constructor with 4-Parameters (ID,NOC,BoardType,Multimedia)
---------------------------------
Room ID : 5
N.O. Chairs : 40
Board Type : M
Multimedia : Repairing..
Remarks :
Remarks added from Main
Constructor with All(5)Parameters (ID,NOC,BoardType,Multimedia,Remarks)
---------------------------------
Press any key to continue . . .
*/
```

Constructor with Default Parameters

Given example demonstrates the concept of constructor with default values

```
#include <string>
#include <iostream>
using namespace std;
class ClassRoom {
private:
	int roomID;
	int numberOfChairs;
	char boardType; // C for chalk and M for marker
	string multimedia;
	string remarks;
public:
	ClassRoom(int id=0,int NOC=25,char c='C',string mul="New",string
rmks="Defaut Value");
	void setroomID(int);
	void setnumberOfChairs(int);
	void setboardType(char);
	void setmultimedia(string);
	void setremarks(string);
	int getroomID();
	int getnumberOfChairs();
	char getboardType();
	string getmultimedia();
	string getremarks();
	void display();
};
void main ()
{
	ClassRoom CR0,CR1(1),CR2(2,30),CR3(3,35,'M'),
```

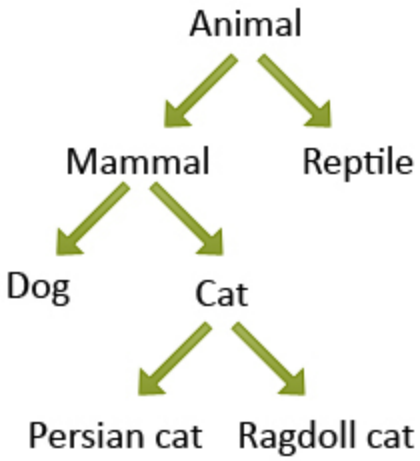
```
      CR4(4,40,'M',"Repairing.."),CR5(5,40,'M',"Repairing..","Remarks added
from Main");
      CR0.display();
      CR1.display();
      CR2.display();
      CR3.display();
      CR4.display();
      CR5.display();
      system("PAUSE");
}

//--------------------------------------Constructor
ClassRoom::ClassRoom(int id,int NOC,char c,string mul, string remks)
{
this->setroomID(id); //this->roomID=id;
this->setnumberOfChairs(NOC); //this->numberOfChairs=NOC;
this->setboardType(c); //this->boardType=c;
this->setmultimedia(mul); //this->multimedia=str;
this->setremarks(remks);
}
// All Setter & Getter Functions Deffinitions <Code here all the setter and
getter functions>
//Printing Functions <code here all printing functions>
/*
Room ID : 0
N.O. Chairs : 0
Board Type : M
Multimedia : New
Remarks :
Default Constructor
---------------------------------
Room ID : 1
N.O. Chairs : 0
Board Type : M
Multimedia : New
Remarks :
Constructor with 1-Parameter (ID)
---------------------------------
Room ID : 2
N.O. Chairs : 30
Board Type : M
Multimedia : New
Remarks :
Constructor with 2-Parameters (ID,NOC)
---------------------------------
Room ID : 3
N.O. Chairs : 35
Board Type : M
Multimedia : New
Remarks :
```

```
Constructor with 3-Parameters (ID,NOC,BoardType)
---------------------------------
Room ID : 4
N.O. Chairs : 40
Board Type : M
Multimedia : Repairing..
Remarks :
Constructor with 4-Parameters (ID,NOC,BoardType,Multimedia)
---------------------------------
Room ID : 5
N.O. Chairs : 40
Board Type : M
Multimedia : Repairing..
Remarks :
Remarks added from Main
Constructor with All(5)Parameters (ID,NOC,BoardType,Multimedia,Remarks)
---------------------------------
Press any key to continue . . .
*/
```

Inheritance

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.
When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.
Inheritance is an "**is-a**" relation. As showed in f



Inheritance is an "**is-a**" relation. As showed in figure,

- Every mammal **is an** animal.
- Every cat **is a** mammal.
- Every dog **is a** mammal.

To have a quick overview of **basics of OOP**, visit

- http://teknadesigns.com/what-is-object-oriented-programming/
- http://www.tutorialspoint.com/cplusplus/cpp_pdf_version.htm

Types of Inheritance

C++ supports three types of inheritance.

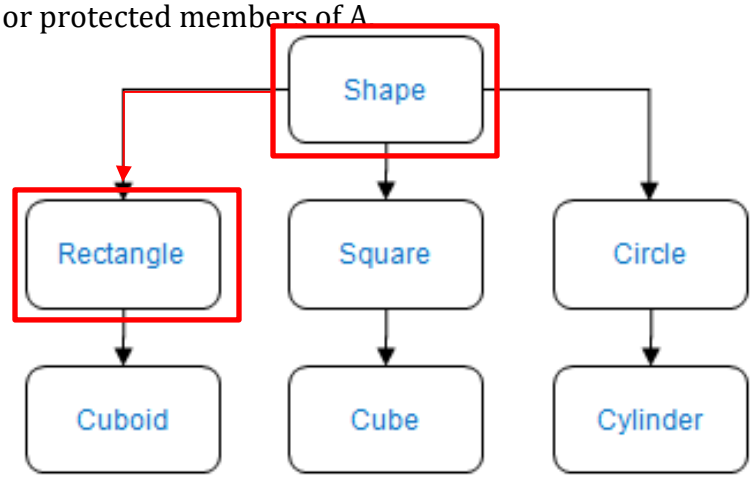
I. Public Inheritance
II. Private Inheritance
III. Protected Inheritance

Public Inheritance

In this diagram, shape is the base class. The class rectangle is derived from shape. Every rectangle is a shape. Suppose class B is derived from class A. Then, B cannot directly access the private members of A. That is, the private members of A are hidden in B.

If *memberAccessSpecifier* is public—that is, the inheritance is public—then:

a) The public members of A are public members of B. They can be directly accessed in class B.
b) The protected members of A are protected members of B. They can be directly accessed by the member functions (and friend functions) of B.
c) The private members of A are hidden in B. They cannot be directly accessed in B. They can be accessed by the member functions (and friend functions) of B through the public or protected members of A.




```
#include <iostream>
using namespace std;
// Base class
class Shape
{
public:
	void setWidth(int w)
	{
		width = w;
	}
	void setHeight(int h)
	{
		height = h;
	}
```

```
	protected:
	int width;
	int height;
};
// Derived class
class Rectangle: public Shape
{
public:
	int getArea()
	{
		return (width * height);
	}
};
int main(void)
{
Rectangle Rect;
Rect.setWidth(5);
Rect.setHeight(7);
// Print the area of the object.
cout << "Total area: " << Rect.getArea() << endl;
system("pause");
return 0;

}
/*
Total area: 35
Press any key to continue . . .

*/
```

Test public inheritance

```
// tests publicly
#include <iostream>
#include <conio.h>
using namespace std;
class A //base class
{
private:
	int privdataA;
protected:
	int protdataA;
public:
	int pubdataA;
};
class B : public A //publicly-derived class
{
public:
	void funct()
	{
		int a;
		a = privdataA; //error: not accessible
		a = protdataA; //OK
```

```
            a = pubdataA; //OK
      }
};
void main()
{
      int a;
      B objB;
      a = objB.privdataA; //error: not accessible
      a = objB.protdataA; //error: not accessible
      a = objB.pubdataA; //OK (A public to B)
      system("pause");
}
```

Private Inheritance

If access specifier is private—that is, the inheritance is **private**—then:

a) The public members of A are private members of B. They can be accessed by the member functions (and friend functions) of B.
b) The protected members of A are private members of B. They can be accessed by the member functions (and friend functions) of B.
c) The private members of A are hidden in B. They cannot be directly accessed in B. They can be accessed by the member functions (and friend functions) of B through the public or protected members of A.

```
// tests publicly- and privately-derived classes
#include <iostream>
#include <conio.h>
using namespace std;
class A //base class
{
private:
      int privdataA;
protected:
      int protdataA;
public:
      int pubdataA;
};
class B : public A //publicly-derived class
{
public:
      void funct()
            {
                  int a;
                  a = privdataA; //error: not accessible
                  a = protdataA; //OK
                  a = pubdataA; //OK
            }
};
class C : private A //privately-derived class
{
public:
```

```
        void funct()
                {
                        int a;
                        a = privdataA; //error: not accessible
                        a = protdataA; //OK
                        a = pubdataA; //OK
                }
};
void main()
{
        int a;
        B objB;
        a = objB.privdataA; //error: not accessible
        a = objB.protdataA; //error: not accessible
        a = objB.pubdataA; //OK (A public to B)
        C objC;
        a = objC.privdataA; //error: not accessible
        a = objC.protdataA; //error: not accessible
        a = objC.pubdataA; //error: not accessible (A private to C)
        system("pause");
}
```

Note

If you don't supply any access specifier when creating a class, `private` is assumed.

Protected Inheritance

If Access Specifier is protected—that is, the inheritance is protected—then:

a) The public members of A are protected members of B. They can be accessed by the member functions (and friend functions) of B.
b) The protected members of A are protected members of B. They can be accessed by the member functions (and friend functions) of B.
c) The private members of A are hidden in B. They cannot be directly accessed in B. They can be accessed by the member functions (and friend functions) of B through the public or protected members of A.

```
#include <iostream>
using namespace std;
class bClass
{
public:
      void setData(double);
      void setData(char, double);
      void print() const;
      bClass(char ch = '*', double u = 0.0);
protected:
      char bCh;
private:
      double bX;
};
void bClass::setData(double u)
{
```

```
	bX = u;
}
void bClass::setData(char ch, double u)
{
	bCh = ch;
	bX = u;
}
void bClass::print() const
{
	cout << "Base class: bCh = " << bCh << ", bX = " << bX<< endl;
}
bClass::bClass(char ch, double u)
{
	bCh = ch;
	bX = u;
}
class dClass: public bClass
{
public:
	void setData(char, double, int);
	void print() const;
	dClass(char ch = '*', double u = 0.0, int x = 0);
private:
	int dA;

};
void dClass::setData(char ch, double v, int a)
{
	bClass::setData(v);
	bCh = ch; //initialize bCh using the assignment statement
	dA = a;
}
void dClass::print() const
{
	bClass::print();
	cout << "Derived class dA = " << dA << endl;
}
dClass::dClass(char ch, double u, int x)
	: bClass(ch, u)
{
	dA = x;
}
int main()
{
	bClass bObject; //Line 1
	dClass dObject; //Line 2
	bObject.print(); //Line 3
	cout << endl; //Line 4
	cout << "*** Derived class object ***" << endl; //Line 5
	dObject.setData('&', 2.5, 7); //Line 6
```

```
      dObject.print(); //Line 7
      system("pause");
      return 0;

}
/*
    Base class: bCh = *, bX = 0

*** Derived class object ***
Base class: bCh = &, bX = 2.5
Derived class dA = 7
Press any key to continue . . .



*/
```