OBJECT ORIENTED PROGRAMMING LAB

Lab Manual # 07

Classes and Objects in C++

Instructor: Hurmat Hidayat

Semester Spring, 2022

Course Code: CL1004

Fast National University of Computer and Emerging Sciences Peshawar

Department of Computer Science

OBJECT ORIENTED PROGRAMMING LANGUAGE

Table of Contents

Constructor in C++

- ❖ Special method that is implicitly invoked.
- ❖ Used to create an object (an instance of the class) and initialize it.
- ❖ Every time an object is created, at least one constructor is called.
- ❖ It is special member function having same name as class name and is used to initialize object.
- ❖ It is invoked/called at the time of object creation.
- ❖ It constructs value i.e. provide data for the object that is why it called constructor.
- ❖ Can have parameter list or argument list. Can never return any value (no even void).
- ❖ Normally declared as public.
- ❖ At the time of calling constructor, memory for the object is allocated in the memory.
- ❖ It calls a default constructor if there is no constructor available in the class.

Note: It is called constructor because it constructs the values at the time of object creation.

There can be two types of constructors in C++.

1. Default constructor (no-argument constructor)
2. Parameterized constructor.

1. Default constructor

- ❖ A constructor is called "Default Constructor" when it doesn't have any parameter.
- ❖ It is also called non-parameterized constructor.
- ❖ A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.
- ❖ Note: If we have not defined a constructor in our class, then the C++ compiler will automatically create a default constructor with an empty code and no parameters.
- ❖ Let's see the simple example of C++ default Constructor.

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
        Employee()
        {
            cout<<"Default Constructor Invoked/Called"<<endl;
        }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2;
    return 0;
}
/*
Output
Default Constructor Invoked/Called
Default Constructor Invoked/Called
*/
```

```
#include<iostream>
#include<string.h>
using namespace std;
class Student
    {
        int Roll;
        char Name[25];
        float Marks;
        public:
        Student()           //Default Constructor
            {
                Roll = 1;
                strcpy(Name,"Kumar");
                Marks = 78.42;
            }
void Display()
       {
            cout<<"\n\tRoll : "<<Roll;
            cout<<"\n\tName : "<<Name;
            cout<<"\n\tMarks : "<<Marks;
       }
    };  // end of class
int main()
    {
        Student S;          //Creating Object

        S.Display();        //Displaying Student Details
        return 0;
    }

/*
Output
Roll : 1
Name : Kumar
Marks : 78.42
*/
```

```
#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();
};
int main ()
{
	ClassRoom CR0;
	CR0.display();
	system("PAUSE");
	return 0;
}
//--------------------------------------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 . . .
*/
```

2. Parameterized Constructor

❖ A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

- The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.
- Used to initialize objects with different values.
- This is the preferred method to initialize member data.

❖ Let's see the simple example of C++ Parameterized Constructor.

```
#include <iostream>
using namespace std;
class Employee {
   public:
       int id; //data member (also instance variable)
       string name; //data member(also instance variable)
       float salary;

      Employee(int i, string n, float s)
        {
            id = i;
            name = n;
            salary = s;
        }
void display()
        {
            cout<<id<<"  "<<name<<"  "<<salary<<endl;
        }
};
int main(void) {
  Employee e1 =Employee(101, "Ali", 890000); //creating an object of E
mployee
    Employee e2=Employee(102, "Saad", 59000);
    e1.display();
    e2.display();
    return 0;
}

/*
Output
101  Ali  890000
102  Saad  59000
*/
```

```
#include<iostream>
#include<string.h>
using namespace std;

class Student
    {
        int Roll;
        char Name[25];
        float Marks;
        public:
        Student(int r,char nm[],float m)    //Parameterize Constructor
        {
            Roll = r;
            strcpy(Name,nm);
            Marks = m;
        }
void Display()
        {
            cout<<"\n\tRoll : "<<Roll;
            cout<<"\n\tName : "<<Name;
            cout<<"\n\tMarks : "<<Marks;
        }
    };

int main()
    {
        Student S(2,"Sumit",89.63);
        //Creating Object and passing values to Constructor

        S.Display();
        //Displaying Student Details
        return 0;
    }

/*
Output
Roll : 2
Name : Ali
Marks : 89.63
*/
```

```
#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();
};
int main ()
{
	ClassRoom CR(11,25,'M',"Repairing..","Remarks added from Main");
	CR.display();
	system("PAUSE");
	return 0;
}
//--------------------------------------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)
----------------------------------
*/
```

The Default Copy Constructor

- We've seen two ways to initialize objects. A no-argument constructor can initialize data members to constant values, and a multi-argument constructor can initialize data members to values passed as arguments.
- Let's mention another way to initialize an object: you can initialize it with another object of the same type.

- Surprisingly, you don't need to create a special constructor for this; one is already built into all classes. It's called the default copy constructor. It's a one argument constructor whose argument is an object of the same class as the constructor.
- Initialization of an object through another object is called **copy constructor**.
- In other words, copying the values of one object into another object is called **copy constructor.**

Example of The Default Copy Constructor

```
#include<iostream>
#include<string.h>
using namespace std;

class Student
    {
        int Roll;
        string Name;
        float Marks;

        public:
        Student(int r,string nm,float m)   //Parameterized Constructor
        {
            Roll = r;
            Name=nm;
            Marks = m;
        }

        // Student(Student &S)
        // {
        //    Roll=S.Roll;
        //    Name=S.Name;
        //    Marks =S.Marks;
        // }

        void Display()
        {
            cout<<"\n\tRoll : "<<Roll;
            cout<<"\n\tName : "<<Name;
            cout<<"\n\tMarks : "<<Marks;
        }
    };  // end of class
```

```
int main()
    {
        Student S1(2,"Ali",89.63);

        Student S2(S1);    //Copy S1 to S2
        Student S3=S1;    //copy S1 to S3

        cout<<"\n\tValues in object S1";
        S1.Display();

        cout<<"\n\tValues in object S2";
        S2.Display();

        cout<<"\n\tValues in object S3";
        S3.Display();
    }  // end of main() function


/*
Output
 Values in object S1
 Roll : 2
 Name : Ali
 Marks : 89.63
 Values in object S2
 Roll : 2
 Name : Ali
 Marks : 89.63
 Values in object S3
 Roll : 2
 Name : Ali
 Marks : 89.63
*/
```

- ❖ We initialize S1 to the value of “**2,"Ali",89.63**” using the three-argument constructor. Then we define two more objects of type Student Class, S2 and S3, initializing both to the value of S1.
- ❖ You might think this would require us to define a one-argument constructor, but initializing an object with another object of the same type is a special case. These definitions both use the default copy constructor.
- ❖ The object S2 is initialized in the statement

    Student S2(S1);

- This causes the default copy constructor for the Student class to perform a member-by-member copy of S1 into S2.
- Surprisingly, a different format has exactly the same effect, causing S1 to be copied member-by-member into S3:

  Student S3 = S1;

Constructor Overloading

- More than one constructor functions can be defined in one class. When more than one constructor functions are defined, each constructor is defined with a different set of parameters.
- Defining more than one constructor with different set of parameters is called constructor overloading.
- Constructor overloading is used to initialize different values to class objects.
- When a program that uses the constructor overloading is compiled, C++ compiler checks the number of parameters, their order and data types and marks them differently.
- When an object of the class is created, the corresponding constructor that matches the number of parameters of the object function is executed.
- In the following example two constructor functions are defined in the class "**Sum**".

```
using namespace std;
#include<iostream>
class Sum
{
 public:
   Sum(int l, int m, int n)
   {
       cout<<"Sum of three integer is= "<<(l+m+n)<<endl;
   }
Sum(int l, int m)
   {
       cout<<"Sum of two integer is= "<<(l+m)<<endl;
   }
}; // end of class body

int main () {

    Sum s1=Sum(3,4,5);
    Sum s2=Sum(2,4);
    //Sum s1(3,4,5), s2(2,4);
   return 0;
}

/*
Output
Sum of three integer is= 12
Sum of two integer is= 6
*/
```

- When the above program is executed, the object s1 is created first and then the Sum constructor function that has only three integer type parameters is executed.
- Then the s2 object is created. It has two parameters of integer type, So the constructor function that has two arguments of integer type is executed.

Write a program to define two constructors to find out the maximum values.

```
using namespace std;
#include<iostream>
class Find
{
   private:
   int max;
public:
   Find(int x, int y, int z)
   {
       if (x>y)
        {
            if(x>z)
            {
                max=x;
            }
            else
            {
                max=z;
            }
        }
        else if(y>z)
            max=y;
        else
            max=z;
       cout<<"Maximum between three numbers is= "<<max<<endl;
   }
 Find(int x, int y)
   {
       if (x>y)
       {
           max=x;
       }
       else
       {
           max=y;
       }
       cout<<"Maximum between two numbers is= "<<max<<endl;
   }
}; // end of class body


int main () {

    int a=9, b=56, c=67;
    Find f1=Find(a,b,c);
    Find f2=Find(a,b);
    //Find f1(a,b,c), f2(a,b);

   return 0;
}

/*
Output
Maximum between three numbers is = 67
Maximum between two numbers is = 56
*/
```

```
//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 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();

};
int 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();
return 0;
}
//--------------------------------------Constructors
ClassRoom::ClassRoom()
{
    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->setboardType('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,BoardTy
pe,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;
}
```

```
/*
Output:
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 . . .
*/
```

Defining Constructor Outside Class

```
class class_name {

public:
   //Constructor declaration
   class_name();
   //... other Variables & Functions
}; // Class body ends

// Constructor definition outside Class
class_name::class_name()
{
   // Constructor code
}
```

Defining Constructor Outside Class Example

```
using namespace std;
#include<iostream>
class Sum
{
   //Constructor declaration
   public:
   Sum(int l, int m, int n);
   Sum(int l, int m);

}; // end of class body
int main () {
    Sum s1=Sum(3,4,5);
    Sum s2=Sum(2,4);
    //Sum s1(3,4,5), s2(2,4);

   return 0;
}  //end of main() function
// Constructor definition outside Class
Sum::Sum(int l, int m, int n)
   {
       cout<<"Sum of three integer is= "<<(l+m+n)<<endl;
   }
Sum::Sum(int l, int m)
   {
       cout<<"Sum of two integer is= "<<(l+m)<<endl;
   }
```

C++ this pointer

In C++ programming, **this** is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C++.

- It can be used **to pass current object as a parameter to another method.**
- It can be used **to refer current class instance variable.**
- It can be used **to declare indexers.**

```
#include <iostream>
using namespace std;
class Employee {
   public:
       int id; //data member (also instance variable)
       string name; //data member(also instance variable)
       float salary;
       Employee(int id, string name, float salary)
        {
            this->id = id;
            this->name = name;
            this->salary = salary;
        }
void display()
        {
            cout<<id<<"  "<<name<<"  "<<salary<<endl;
        }
};   // class body ends

int main(void) {
Employee e1 =Employee(101, “Ali", 890000); //creating an object of Emp
loyee
Employee e2=Employee(102, “Sania", 59000); //creating an object of Emp
loyee

e1.display();
e2.display();
    return 0;
}

/*
Output
101  Ali  890000
102  Sania  59000
*/
```

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 r
emks="Defaut Value")
    {
        //this->roomID=id;
    this->setroomID(id);
    this->numberOfChairs=NOC;
    this->boardType=c;
    this->setmultimedia(mul);
    this->setremarks(remks);
    }

    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();
};
int main ()
{
    ClassRoom CR0,CR1(1),CR2(2,30),CR3(3,35,'M'),
    CR4(4,40,'M',"Repairing.."),CR5(5,40,'M',"Repairing..","Remarks ad
ded from Main");
    CR0.display();
    CR1.display();
    CR2.display();
    CR3.display();
    CR4.display();
    CR5.display();
    return 0;
}

//----------------------------------------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         : 0

N.O. Chairs     : 25

Board Type      : C

Multimedia      : New

Remarks         :

Defaut Value

----------------------------------

Room ID         : 1

N.O. Chairs     : 25

Board Type      : C

Multimedia      : New

Remarks         :

Defaut Value

----------------------------------

Room ID         : 2

N.O. Chairs     : 30

Board Type      : C

Multimedia      : New

Remarks         :

Defaut Value
```

```
----------------------------------

Room ID         : 3

N.O. Chairs     : 35

Board Type      : M

Multimedia      : New

Remarks         :

Defaut Value

----------------------------------

Room ID         : 4

N.O. Chairs     : 40

Board Type      : M

Multimedia      : Repairing..

Remarks         :

Defaut Value

----------------------------------

Room ID         : 5

N.O. Chairs     : 40

Board Type      : M

Multimedia      : Repairing..

Remarks         :

Remarks added from Main

----------------------------------*/
```

The new Operator

The new operator is an **operator** which denotes a request for memory allocation on the Heap. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable. When you create an object of class using new keyword (normal new).

- The memory for the object is allocated using **operator new** from heap.
- The constructor of the class is invoked to properly initialize this memory.

```
// CPP program to illustrate use of new keyword

#include<iostream>
using namespace std;
class car
{
	string name;
	int num;
	public:
		car(string a, int n)
		{
			cout << "Constructor called" << endl;
			this ->name = a;
			this ->num = n;
		}
		void enter()
		{
			cin>>name;
			cin>>num;
		}
		void display()
		{
			cout << "Name: " << name << endl;
			cout << "Num: " << num << endl;
		}
};
int main()
{
	// Using new keyword
	car *p = new car("Honda", 2017);
	p->display();
}

/*
Output
Constructor called
Name: Honda
Num: 2017
*/
```

Destructors

- When an object is destroyed, a special member function of that class is executed automatically. This member function is called **destructor function** or **destructor**.

- The destructor function has the same name as the name of a class but a tilde sign (**~**) is written before its name. It is executed automatically when an object comes to end of its life.
- Like constructors, destructor do not return any value. They also do not take any arguments.
- **For example**, a local object is destroyed when all the statements of the function in which it is declared are executed. So, at the end of the function, the destructor function is executed.
- Similarly, global objects (objects that are declared before main function) or static objects are destroyed at the end of main function. The life time of these objects end when the program execution ends.
- So, at the end of program the destructor function is executed. The destructor functions are commonly used to free the memory that was allocated for objects.
- Constructor is invoked automatically when the object created.
- Destructor is invoked when the object goes out of scope. In other words, Destructor is invoked, when compiler comes out from the function where an object is created.
- The following example explains the concept of constructors and destructors.

```
using namespace std;
#include<iostream>
class Prog
{
    public:
    Prog()
    {
        cout<<"This is constructor function "<<endl;
    }

    ~Prog()
    {
        cout<<"This is destructor function "<<endl;
    }

}; // end of class body

int main () {

    Prog x;
    int a, b;
    a=10;
    b=20;
    cout<<"Sum of two numbers is = "<<(a+b)<<endl;
```

```
   return 0;
}

/*
Output
This is constructor function
Sum of two numbers is = 30
This is destructor function
*/
```

C++ Objects and Functions

- ❖ In this tutorial, we will learn to pass objects to a function and return an object from a function in C++ programming.
- ❖ In C++ programming, we can pass objects to a function in a similar manner as passing regular arguments.

Passing Objects as Arguments to Function

Objects can also be passed as arguments to member functions. When an object is passed as an argument to a member function:

- only the name of the object is written in the argument.
- The number of parameters and their types must be defined in the member function which the object is to be passed. The objects that are passed are treated local for the member functions and are destroyed when the control returns to the calling function.

Example 1: C++ Pass Objects to Function

```
using namespace std;
#include<iostream>
class Test
{
   private:
   char name[20];

   public:
   void get()
   {
      cout<<"Enter your name: ";
      cin.get(name, 20);
   }
  void print(Test s)
   {
      cout<<"Name is: "<<s.name<<endl;
   }

}; // end of class body

int main () {

   Test test1,test2;
   test1.get(); // calling get() function for object initialization

   test2.print(test1);  //Passing object as argument to function

  return 0;
}

/*
Output
Enter your name: Nouman Yousaf
Name is: Nouman Yousaf
*/
```

- In the above program, the class “Test” has one data member “name” of string type and two member functions “get()” and “print()”. The print() function has parameter of class Test type.

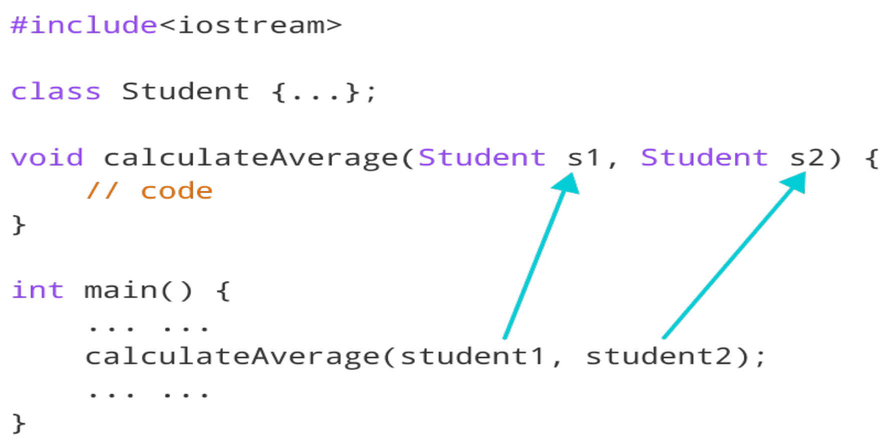
- The objects test1 and test2 are declared of class Test. The member function gets the name in object test1, and store it into the data member “name”. The member function “print()” for objects “test2” is called by passing argument of object “test1”. When the control shifts to member function “print()”, a copy of “test1” is created as a local object in the print function with name “s”.

Example 2: C++ Pass Objects to Function

```
// C++ program to calculate the average marks of two students
#include <iostream>
using namespace std;

class Student {

   public:
    double marks;

    // constructor to initialize marks
    Student(double m) {
        marks = m;
    }
};  // class body ends
// function that has objects as parameters
```

```
void calculateAverage(Student s1, Student s2) {

    // calculate the average of marks of s1 and s2
    double average = (s1.marks + s2.marks) / 2;

   cout << "Average Marks = " << average << endl;

}
int main() {
    Student student1(88.0), student2(56.0);

  // pass the objects as arguments
   calculateAverage(student1, student2);

    return 0;
}

/*
Output
Average Marks = 72
*/
```

Here, we have passedtwo Student objects student1 and student2 as arguments to the calculateAverage() function.



```
#include<iostream>

class Student {...};

void calculateAverage(Student s1, Student s2) {
    // code
}

int main() {
    ... ...
    calculateAverage(student1, student2);
    ... ...
}
```

Example 3: C++ Return Object from a Function

```
#include <iostream>
using namespace std;
class Student {
   public:
    double marks1, marks2;
};  //class body ends
// function that returns object of Student
Student createStudent() {
    Student student;

    // Initialize member variables of Student
    student.marks1 = 96.5;
    student.marks2 = 75.0;
    // print member variables of Student
    cout << "Marks 1 = " << student.marks1 << endl;
    cout << "Marks 2 = " << student.marks2 << endl;

    return student;
}
int main() {
    Student student1;
    // Call function
    student1 = createStudent();
    return 0;
}
/*
Output
Marks 1 = 96.5
Marks 2 = 75
*/
```

Arrays as Class Members in C++

- Arrays can be declared as the members of a class. The arrays can be declared as private, public or protected members of the class.
- To understand the concept of arrays as members of a class, consider this example.

Example: A program to demonstrate the concept of arrays as class members.

```
#include<iostream>
using namespace std;
const int size=5;
class Student
{
  int roll_no;
  int marks[size];

  public:
    void getdata ();
    void tot_marks ();
};  // end of clas body
```

```
int main()
{
   Student s1;
   s1.getdata() ;
   s1.tot_marks() ;
   return 0;
} //ends of main() function
//function definitions
void Student :: getdata ()
{
      cout<<"\nEnter roll no: ";
      cin>>roll_no;
      for(int i=0; i<size; i++)
      {
        cout<<"Enter marks in subject"<<(i+1)<<": ";
        cin>>marks[i] ;
      }
}
//calculating total marks
void Student :: tot_marks ()
{
      int total=0;
      for(int i=0; i<size; i++)
      {
          total=total+marks[i];
      }
      cout<<"\nTotal marks "<<total;
}

/*
Output
Enter roll no: 333
Enter marks in subject1: 56
Enter marks in subject2: 88
Enter marks in subject3: 77
Enter marks in subject4: 66
Enter marks in subject5: 88
Total marks 375
*/
```

In this example, an array marks is declared as a private member of the class student for storing a student's marks in five subjects. The member function tot_marks () calculates the total marks of all the subjects and displays the value.

Similar to other data members of a class, the memory space for an array is allocated when an object of the class is declared. In addition, different objects of the class have their own copy of the array. Note that the elements of the array occupy contiguous memory locations along with other data members of the object. For instance, when an object s1 of the class student is declared, the memory space is allocated for both rollno and marks.

Ways to initialize object

There are 3 ways to initialize object in C++.

- By directly accessing data members of class using object
- By member functions of the class
- By constructors of class

References

https://beginnersbook.com/2017/08/cpp-data-types/

http://www.cplusplus.com/doc/tutorial/basic_io/

https://www.w3schools.com/cpp/default.asp

https://www.javatpoint.com/cpp-tutorial

https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/?ref=lbp

https://www.programiz.com/

https://ecomputernotes.com/cpp/