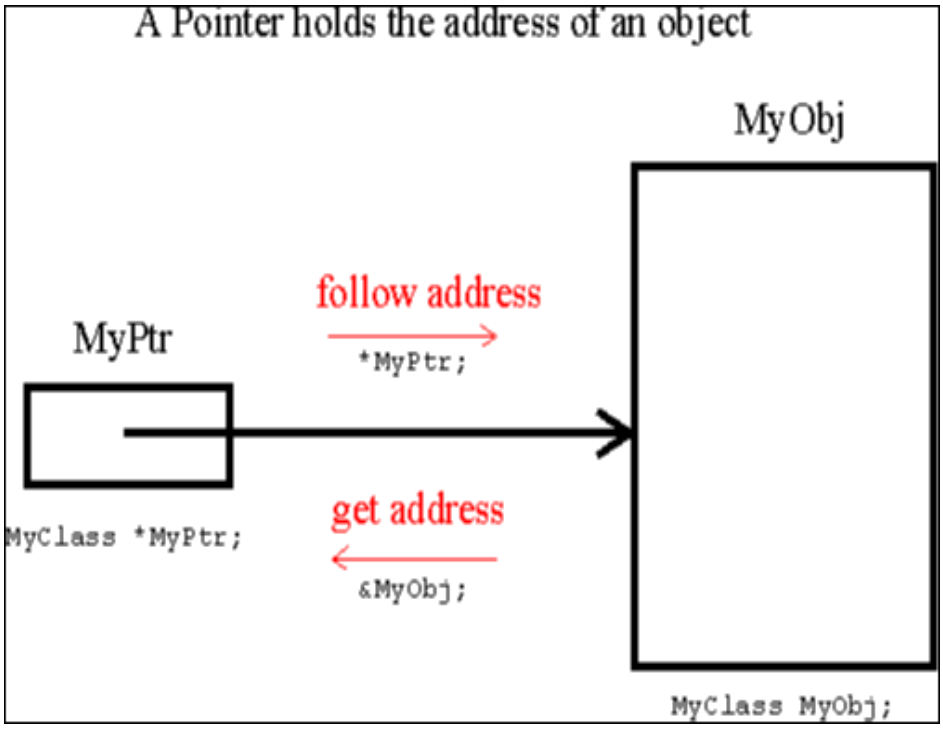
Object Oriented & Programming Language
Lab#07

Contents

Pointers



Accessing Pointers to Classes and Structs

To assign a value to a component of object using a pointer type, use the following syntax:

```
(*ptrVariableName).classMemberName=value;
```

The dot operator (.) has a higher precedence than the dereferencing operator, requiring the use of parentheses. C++ provides another operator, called the **member access operator arrow** (->).To assign a value to a component of object using the -> operator, use the following syntax:

```
ptrVariableName->classMemberName=value;
```

If you want to initialize a pointer variable, the only number that can be directly assigned to a pointer variable is 0 or NULL.

Structure & Pointer

```
struct studentType
{
	char name[26];
	double gpa;
	int sID;
	char grade;
};
studentType student;
studentType *studentPtr;
```

```
studentPtr = &student;
```

To store 3.9 in gpa

```
(*studentPtr).gpa = 3.9;
```

```
pointerVariableName->classMemberName
```

Following statement is equal to above statement.

```
studentPtr->gpa = 3.9;
```

Pointers to Structures

Like any other type, structures can be pointed by its own type of pointers:

```
struct movies_t {
       string title;
       int year;
};
movies_t amovie;
```

```
movies_t * pmovie;
```

Here amovie is an object of structure type movies_t, and pmovie is a pointer to point to objects of structure type movies_t. So, the following code would also be valid:

```
pmovie = &amovie;
```

The value of the pointer pmovie would be assigned to a reference to the object amovie (its memory address).

We will now go with another example that includes pointers, which will serve to introduce a new operator: the arrow operator (->):

```
// pointers to structures
#include <iostream> #include
<string> #include <sstream>
using namespace std; struct
movies_t {

       string title;
       int year;
};
int main ()
{
       string mystr;
       movies_t amovie;
       movies_t * pmovie;
       pmovie = &amovie;
       cout << "Enter title: "; getline (cin,
       pmovie->title); cout << "Enter year:
       "; getline (cin, mystr);
       (stringstream) mystr >> pmovie->year;
       cout << "\nYou have entered:\n"; cout
       << pmovie->title;
```

```
        cout << " (" << pmovie->year << ")\n";
        return 0;
}
/* Sample Output
Enter title: Invasion of the body snatchers
Enter year: 1978
You have entered:
Invasion of the body snatchers (1978)
*/
```

The previous code includes an important introduction: the arrow operator (->). This is a dereference operator that is used exclusively with pointers to objects with members. This operator serves to access a member of an object to which we have a reference. In the example we used:

```
pmovie->title
```

Which is for all purposes equivalent to:

```
(*pmovie).title
```

Both expressions pmovie->title and (*pmovie).title are valid and both mean that we are evaluating the member title of the data structure pointed by a pointer called pmovie. It must be clearly differentiated from:

```
*pmovie.title
```

which is equivalent to:

```
*(pmovie.title)
```

And that would access the value pointed by a hypothetical pointer member called title of the structure object pmovie (which in this case would not be a pointer). The following panel summarizes possible combinations of pointers and structure members:

Expression	What is evaluated	Equivalent
a.b	Member b of object a	
a->b	Member b of object pointed by a	(*a).b
*a.b	Value pointed by member b of object a	*(a.b)

Nesting Structures with Pointers

Structures can also be nested so that a valid element of a structure can also be in its turn another structure.

```
struct movies_t {
        string title;
        int year;
};
struct friends_t {
        string name;
        string email;
        movies_t favorite_movie;
} charlie, maria;
friends_t * pfriends = &charlie;
```

After the previous declaration we could use any of the following expressions:

```
charlie.name
maria.favorite_movie.title
charlie.favorite_movie.year
pfriends->favorite_movie.year
```

(where, by the way, the last two expressions refer to the same member).

Class & Pointer

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
class classExample
{
public:
	void setX(int a);
	//Function to set the value of x
	//Postcondition: x = a;
	void print() const;
	//Function to output the value of x
private:
	int x;
};
void classExample::setX(int a)
{
	x = a;
}
void classExample::print() const
{
	cout << "x = " << x << endl;
}
int main()
{
	classExample *cExpPtr; //Line 1
	classExample cExpObject; //Line 2
	cExpPtr = &cExpObject; //Line 3
	cExpPtr->setX(5); //Line 4
	cExpPtr->print(); //Line 5
	return 0;
}
/*
Sample Run:
x = 5*/
```

Object Creation with Class Pointer

```
#include <string>
#include <iostream>
using namespace std;
class ClassRoom {
public:
	int roomID;
	string remarks;
private:
	int numberOfChairs;
```

```
public:
	ClassRoom()
		{
			this->roomID=0;
			this->numberOfChairs=0;
/*
Room ID
N.O. Chairs
Remarks
---------------------------------
Room ID
N.O. Chairs
Remarks
---------------------------------
Room ID
N.O. Chairs
Remarks
---------------------------------
*/
```

Member Variable as Pointer inside Class

```
#include <string>
#include <iostream>
using namespace std;
class ClassRoom {
public:
	int *roomID;
	string remarks;
private:
	int *numberOfChairs;
public:
	ClassRoom(int id,int NOC, string remks)
		{
			roomID= new int;
			numberOfChairs= new int;
			*roomID=id;
			*numberOfChairs=NOC;
			this->remarks=remks;
		}

	//-----------------------------------Printing Functions
	void display()
		{
			cout<<"\nRoom ID \t: "<<*this->roomID<<endl;
			cout<<"N.O. Chairs \t: "<<*this->numberOfChairs<<endl;
			cout<<"Remarks \t: "<<this->remarks<<endl;
			cout<<"---------------------------------"<<endl;
		}
};
void main ()
{
	ClassRoom cr1(1,3,"Object is created without pointer");
	*cr1.roomID=5;
	cr1.display();
```

```
      *cr1.roomID=5;
      cr1.remarks="value of roomID chnaged in main funcion using pointer
...";
      cr1.display();
      ClassRoom *cr2;
      cr2 = new ClassRoom(5,7,"Object is created pointer");
      cr2->display();
      cr2=&cr1;// interesting
      cr2->remarks="The value of CR1's Remarks are changed by cr2";
      cr2->display();
      system("PAUSE");
}
/*
Room ID        : 5
N.O. Chairs    : 3
Remarks        : Object is created without pointer
---------------------------------

Room ID        : 5
N.O. Chairs    : 3
               : value of roomID chnaged in main funcion using
Remarks        pointer ...
---------------------------------

Room ID        : 5
N.O. Chairs    : 7
Remarks        : Object is created pointer
---------------------------------

Room ID        : 5
N.O. Chairs    : 3
                 The value of CR1's Remarks are changed by
Remarks        : cr2
---------------------------------
*/
```

Pointers in Destructor

Usually used to delete pointers initialized with new-operator.

```
      ~ClassRoom()
      {
            delete this->roomID;
            delete this->numberOfChairs;
      }
```

Shallow versus Deep Copy and Pointers using Arrays

In a shallow copy, two or more pointers of the same type point to the same memory; that is, they point to the same data. If the memory is deallocated by deleting the pointer that points to that location, any other pointer that points to that location subsequently becomes invalid.

In a deep copy, two or more pointers have their own data.

```
#include <iostream>
#include <conio.h>
using namespace std;
const int size=5;
void initilize_array(int a[])
```

```
{
	for(int i=0; i<size;i++)
	{
		a[i]=i;
	}
}
void print_array(int a[])
{
	for(int i=0; i<size;i++)
	{
		cout<<a[i]<<", ";
	}
	cout<<endl<<endl;
}
void copy_array(int a[],int b[])
{
	for(int i=0; i<size;i++)
	{
		a[i]=b[i];
	}
}
void main()
{
 int *p,*q;
 p=new int[size];
 ::initilize_array(p);
 cout<<"Array P is : "<<endl;
 ::print_array(p);
  q=p;
 cout<<"Array Q is : "<<endl;
 ::print_array(q);
 delete q;
 p = new int[size];
 cout<<"After deletion of pointer Q, Array P is : "<<endl;
 ::print_array(p);
 ::initilize_array(p);
 cout<<"Array P again initialized is : "<<endl;
 ::print_array(p);
 q=new int[size];
 ::copy_array(q,p);
 cout<<"Array Q After deep copy is : "<<endl;
 ::print_array(q);
 delete q;
 cout<<"After deletion of pointer Q, Array P is : "<<endl;
 ::print_array(p);
 system("PAUSE");
}


/*
Array P is :
0,  1, 2,  3,  4,

Array Q is :
0,  1, 2,  3,  4,
After deletion of pointer Q, Array P is :
-842150451, -842150451, -842150451, -842150451, -842150451,

Array P again initialized is :
0, 1, 2, 3, 4,

Array Q After deep copy is :
```

```
0, 1, 2, 3, 4,

After deletion of pointer Q, Array P is :
0, 1, 2, 3, 4,

*/
```

Shallow & Deep Copy in Classes' Objects

```
#include <string>
#include <iostream>
using namespace std;
class ClassRoom {
public:
	int roomID;
	string remarks;
private:
	int numberOfChairs;

public:
	ClassRoom()
		{
			this->roomID=0;
			this->numberOfChairs=0;
			this->remarks="Default Constructor";
		}
	ClassRoom(int id,int NOC, string remks)
		{
			this->roomID=id;
			this->numberOfChairs=NOC;
			this->remarks=remks;
		}

	void display()
		{
			cout<<"\nRoom ID \t: "<<this->roomID<<endl;
			cout<<"N.O. Chairs \t: "<<this->numberOfChairs<<endl;
			cout<<"Remarks \t: "<<this->remarks<<endl;
			cout<<"---------------------------------"<<endl;
		}
};
void main ()
{
	ClassRoom *cr1;
	cr1= new ClassRoom(1,3,"Object is created with pointer");
	cr1->display();
	ClassRoom *cr2;
	cr2=cr1;
	cout<<"\nAddress of CR1 : "<<cr1;
	cout<<"\nAddress of CR2 : "<<cr2<<endl<<endl; cr2->remarks="Shallo Copy:
	One object's address is opied into second"; cr2->display();

	delete cr2;
	//cr1->display(); this statement will cause a break as cr1's actula addres has
been deleted
	ClassRoom R1(1,3,"Object is created without pointer");
	ClassRoom R2;
	R1.display();
	R2=R1;
	R2.remarks="R1 is copied into R2 : "+R2.remarks;
	R2.display();
	system("PAUSE");
```

```
}
/*
Room ID         : 1
N.O. Chairs     : 3
Remarks         : Object is created with pointer
---------------------------------

Address of CR1 : 00395EE0
Address of CR2 : 00395EE0


Room ID          : 1
N.O. Chairs     : 3
Remarks         : Shallo Copy: One object's address is opied into second
---------------------------------

Room ID         : 1
N.O. Chairs     : 3
Remarks         : Object is created without pointer
---------------------------------

Room ID         : 1
N.O. Chairs     : 3
Remarks         : R1 is copied into R2 : Object is created without pointer
---------------------------------
Press any key to continue . . .
*/
```

Copy Constructor & Deep Copy

A copy constructor is a special constructor for a class/struct that is used to make a copy of an existing instance. According to the C++ standard, the copy constructor for MyClass must have one of the following signatures:

```
//DEEP and SHALLOW copy concept example, for classes with primitive
members //(for classes with only dynamic member data, a copy
constructor would be needed for a deep copy):
#include <string>
#include <iostream>
using namespace std;
class ClassRoom {
public:
      int roomID;
      string remarks;
private:
      int numberOfChairs;

public:
      ClassRoom()
            {
                  this->roomID=0;
                  this->numberOfChairs=0;
                  this->remarks="Default Constructor";
            }
      ClassRoom(int id,int NOC, string remks)
            {
                  this->roomID=id;
                  this->numberOfChairs=NOC;
```

```
                this->remarks=remks;
        }
    ClassRoom(ClassRoom &temp)
        {
            this->roomID=temp.roomID; this-
            >numberOfChairs=temp.numberOfChairs;
            this->remarks=temp.remarks;
        }
    //--------------------------------
    ----                                Printing Functions
    void display()
        {
            cout<<"\nRoom ID \t: "<<this->roomID<<endl;
            cout<<"N.O. Chairs \t: "<<this->numberOfChairs<<endl;
            cout<<"Remarks \t: "<<this->remarks<<endl;
            cout<<"-----------------------------------"<<endl;
        }
};
void main ()
{
    ClassRoom *cr1;
    cr1= new ClassRoom(1,3,"Object is created with pointer");
    cr1->display();
    ClassRoom *cr2;
    cr2= new ClassRoom(*cr1);
    delete cr1;
    cr2->remarks="Copy Constructor : "+cr2->remarks;
    cr2->display();
    system("PAUSE");
}


/*
Room ID         : 1
N.O. Chairs     : 3
Remarks         : Object is created with pointer
---------------------------------

Room ID         : 1
N.O. Chairs     : 3
                : Copy Constructor : Object is created with
Remarks         pointer
---------------------------------
Press any key to continue . . .
*/
```

Diff between *a.b, *(a.b) and (*a).b

```
#include<iostream>
using namespace std;

class A{
public:
    int b;
    A(int b=0)
    {
```

```
                this->b = b;
        }
        void print()
        {
        cout<<"b : "<<b<<endl;
        }
};
void main()
{
        A x(5);
        x.print();

        A *a;
        a = &x;

        (*a).b = 7;
        (*a).print();
        a->print();
}


/*
b : 5
b : 7
b : 7
*/
```

```
#include<iostream>

using namespace std;

class B{
public:
        int *b;
        B(int i=0)
        {
                b= new int;
                *b = i;
        }
        void print()
        {
        cout<<"b : "<<*b<<endl;
        }
};
void main()
{
        B a(5);
        a.print();

        *(a.b)=7;
        a.print();

        *a.b = 9;
        a.print();

}

/*
b : 5
b : 7
```

```
b : 9
*/
```

This Pointer

```
#include <string>
#include <iostream>
using namespace std;
class ClassRoom {
public:
      int roomID;
      string remarks;
private:
      int numberOfChairs;

public:
      ClassRoom()
            {
                  this->roomID=0;
                  this->numberOfChairs=0;
                  this->remarks="Default Constructor";
            }
      ClassRoom(int id,int NOC, string remks)
            {
                  this->roomID=id;
                  this->numberOfChairs=NOC;
                  this->remarks=remks;
            }

      //-------------------------------------Printing Functions
      void display()
            {
                  cout<<"\nRoom ID \t: "<<this->roomID<<endl;
                  cout<<"N.O. Chairs \t: "<<this->numberOfChairs<<endl;
                  cout<<"Remarks \t: "<<this->remarks<<endl;
                  cout<<"---------------------------------"<<endl;
            }
};
void main ()
{

      ClassRoom cr1(1,3,"Object is created without
      pointer"); cr1.display();

      ClassRoom *cr2;
      cr2 = new ClassRoom(5,7,"Object is created pointer");
      cr2->display();
      cr2=&cr1;// interesting
      cr2->remarks="The value of CR1's Remarks are changed by cr2";
      cr2->display();
      system("PAUSE");
}


/*
Room ID        : 1
N.O. Chairs    : 3
Remarks        : Object is created without pointer
```

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

Room ID         : 5
N.O. Chairs     : 7
Remarks         : Object is created pointer
---------------------------------

Room ID         : 1
N.O. Chairs     : 3
                : The value of CR1's Remarks are changed by
Remarks         cr2
---------------------------------
*/
```

Understanding Peculiarities of Classes with Pointer Data Members

When an object goes out of scope, all data members of the object are destroyed.However, if a data member of that object is a pointer used to create a dynamic array, that dynamic memory must be deallocated using the operator delete. If the pointer p does not use the delete operator to deallocate dynamic memory, the memory space stays marked and cannot be accessed. A destructor can be used to ensure that all memory created by a pointer is deallocated when an object goes out of scope.

To avoid shallow copying of data for classes with a pointer data member, C++ allows the programmer to extend the definition of the assignment operator.This process is called overloading the assignment operator.

When declaring a class object, you can initialize it by using the value of an existing object of the same type. This initialization is called the default member-wise initialization.The default member-wise initialization is due to the constructor, called the copy constructor (provided by the compiler). This would result in shallow copying.To force each object to have its own copy of the data, you must override the definition of the copy constructor provided by the compiler; that is, you must provide your own definition of the copy constructor.