Programming in C++

Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

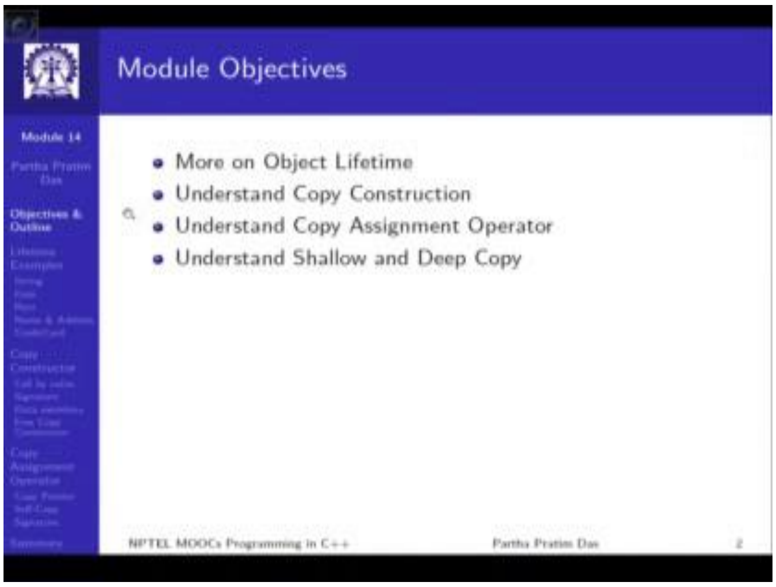
Lecture – 26
Copy Constructor and Copy Assignment Operator

Welcome to module 14 of Programming in C++. In the last module, we have discussed about the construction and destruction of objects and combined with that, we have taken a look in to the lifetime of objects.

We have seen after the construction the object can be used and the construction process actually ends and the object is considered fully constructed, when the initialization list of the object has been completely executed, and the constructor body is about to get executed. Then the object can be used for as long as is required and when the destruction or the destructor call has happened, then the destruction process starts, but the actual destruction happens only when the body of the destructor has completed.

We have seen this with couple of examples and we have also seen the construction and destruction based object lifetime for automatic, static and dynamically allocated objects. Before we move on to more of object construction in terms of copy construction and copy assignment operation. Will take a little bit more look in to the object lifetime.

(Refer Slide Time: 01:42)



Module Objectives

Module 14

Partha Pratim Das

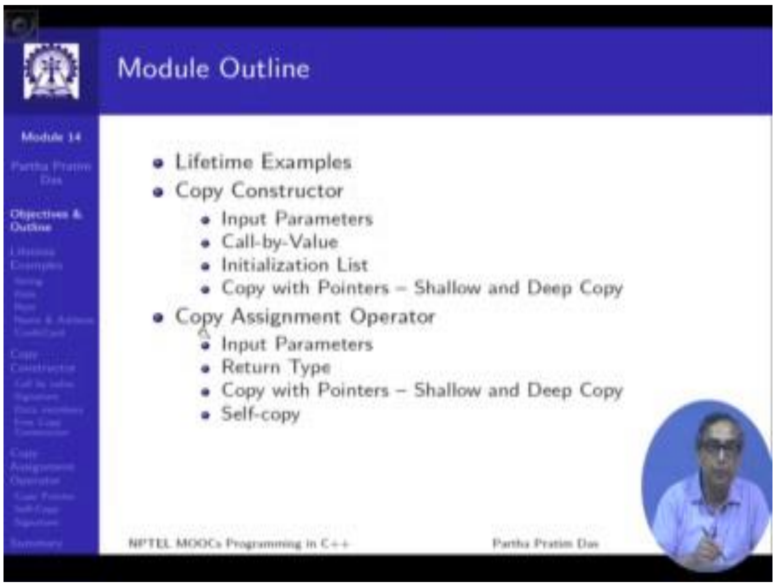
Objectives & Outline

- More on Object Lifetime
- Understand Copy Construction
- Understand Copy Assignment Operator
- Understand Shallow and Deep Copy

NPTEL MOOCs Programming in C++ Partha Pratim Das 2

And then we will primarily focus in this module to discuss on how objects can be copied. The process of copy, which in simple terms is, if x is a variable and y is another variable then assigning y to x is making a copy of y into x. Such copies have a lot more of meaning in terms of C++. So, in this module we will primarily take a look in to that and specifically discuss notions of shallow and deep copy.

(Refer Slide Time: 02:29)



Module Outline

Module 14

Partha Pratim Das

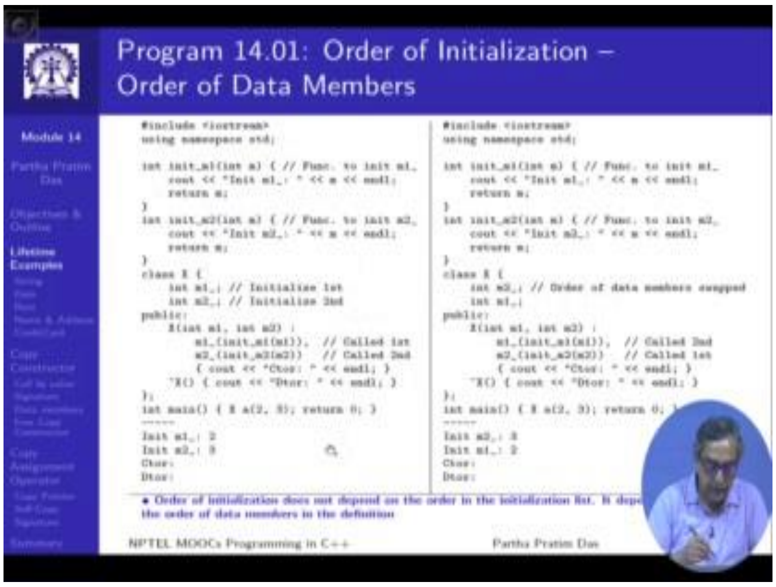
Objectives & Outline

- Lifetime Examples
- Copy Constructor
  - Input Parameters
  - Call-by-Value
  - Initialization List
  - Copy with Pointers – Shallow and Deep Copy
- Copy Assignment Operator
  - Input Parameters
  - Return Type
  - Copy with Pointers – Shallow and Deep Copy
  - Self-copy

NPTEL MOOCs Programming in C++ Partha Pratim Das

Consequently, this is what will be our outline, which is as you know can be visible on the left hand side of the border.

(Refer Slide Time: 02:43)



Program 14.01: Order of Initialization – Order of Data Members

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

int init_m1(int m) { // Func. to init m1_
    cout << "Init m1_: " << m << endl;
    return m;
}
int init_m2(int m) { // Func. to init m2_
    cout << "Init m2_: " << m << endl;
    return m;
}
class X {
    int m1_; // Initialize 1st
    int m2_; // Initialize 2nd
public:
    X(int m1, int m2) :
        m1_(init_m1(m1)),  // Called 1st
        m2_(init_m2(m2))   // Called 2nd
        { cout << "Ctor: " << endl; }
    ~X() { cout << "Dtor: " << endl; }
};
int main() { X a(2, 3); return 0; }
------
Init m1_: 2
Init m2_: 3
Ctor:
Dtor:
```

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

int init_m1(int m) { // Func. to init m1_
    cout << "Init m1_: " << m << endl;
    return m;
}
int init_m2(int m) { // Func. to init m2_
    cout << "Init m2_: " << m << endl;
    return m;
}
class X {
    int m2_; // Order of data members swapped
    int m1_;
public:
    X(int m1, int m2) :
        m1_(init_m1(m1)),  // Called 2nd
        m2_(init_m2(m2))   // Called 1st
        { cout << "Ctor: " << endl; }
    ~X() { cout << "Dtor: " << endl; }
};
int main() { X a(2, 3); return 0; }
------
Init m2_: 3
Init m1_: 2
Ctor:
Dtor:
```

- Order of initialization does not depend on the order in the initialization list. It depe[nds on] the order of data members in the definition

NPTEL MOOCs Programming in C++ Partha Pratim Das

So, first let us take re-look into the object lifetime. Now, what we want to highlight in terms of object lifetime here are few things, one we want to see that when an object has a number of data members, what is the order in which these data members are constructed or initialized and what is the order in which they are destructed or deinitialized. And when these data members particularly are of user defined types, that they are objects of some other class then what happens to the lifetime of the object, which has the data members and the lifetime of the data member themselves.

So, with that we will start with an example program, which you can see here, it is not a very meaningful class. It is just a class x and it has two data members they are integer data members and they are being initialize certain values in the initialization list.

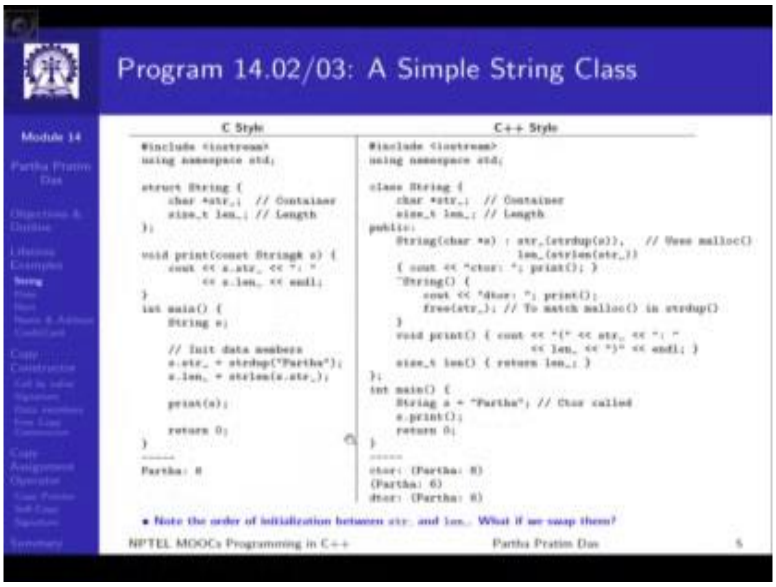
Now, we are interested to find out, what is the order in which they will get initialized. So, we have done a small trick, while initializing the value instead of just directly putting the value, that is what you actually want to do is take m 1 and assign it to this data member instead of doing that we have written a wrapper function, which takes m and returns m.

So, it basically it is input and output are the same, but because this function will get called when this initialization will happen, we will get to know the order in which this functions are getting called or the order in which the initialization happens. So, if you look into this, then you can get to see that the order is first m 1 is initialized and then m 2 is initialized which is what we would expect to happen.

Now, let us look into the right hand side, we have the same set of rapper functions we have the same initialization. The only difference is in the class, we have the same data members, but we have just swap the order of the data members and now you can see between the order of initialization earlier and the order of initialization this time the order of initialization has changed, that is when m 1 was the first data member followed by m 2 they were initialized in that order. Now, when we have changed this order of m 1 and m 2, the initialization order also has changed.

From this, we take a very important lesson in terms of the order of initialization of data members in a class; the lesson is the order depends on the order of data members as they are listed in the class. The order in which they are listed and not the order in which they are actually invoked in the initialization list and we will soon show an example to see what consequence this particular property can have.

(Refer Slide Time: 06:09)



Program 14.02/03: A Simple String Class

Module 14

Partha Pratim Das

C Style

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

struct String {
    char *str_;  // Container
    size_t len_; // Length
};

void print(const String& s) {
    cout << s.str_ << ": "
         << s.len_ << endl;
}
int main() {
    String s;

    // Init data members
    s.str_ = strdup("Partha");
    s.len_ = strlen(s.str_);

    print(s);

    return 0;
}
-----
Partha: 6
```

C++ Style

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

class String {
    char *str_;  // Container
    size_t len_; // Length
public:
    String(char *s) : str_(strdup(s)),   // Uses malloc()
                      len_(strlen(str_))
    { cout << "ctor: "; print(); }
    ~String() {
        cout << "dtor: "; print();
        free(str_); // To match malloc() in strdup()
    }
    void print() { cout << "(" << str_ << ": "
                        << len_ << ")" << endl; }
    size_t len() { return len_; }
};
int main() {
    String s = "Partha"; // Ctor called
    s.print();
    return 0;
}
-----
ctor: (Partha: 6)
(Partha: 6)
dtor: (Partha: 6)
```

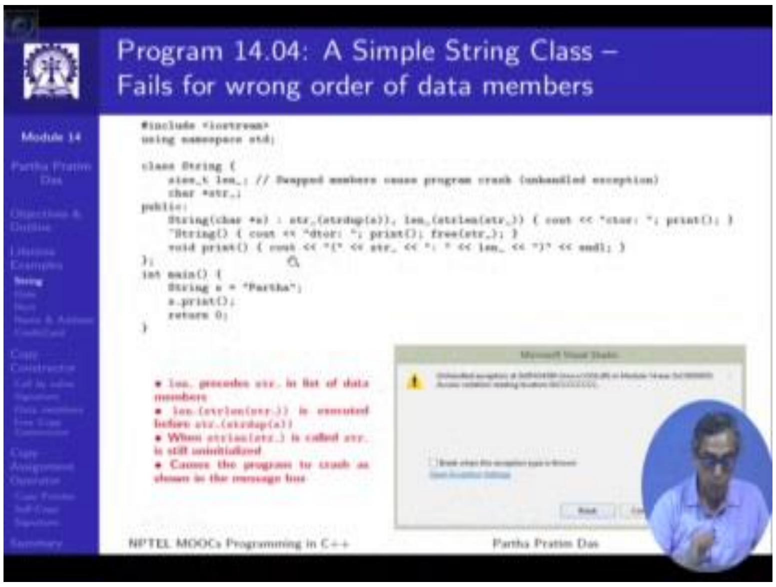
- Note the order of initialization between str_ and len_. What if we swap them?

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

So, let us go to the next slide. Here, we have a simple string class as you have seen earlier also there is a container which basically is a pointer to character which is suppose to keep as history and another data member gives the length. So, if I initialize it in the C style, then we will initialize it with certain initial value of; here we have used a name Partha. So, we will string duplicate Partha into the str field and keep its length in the len field and then we can print. So, it will print whatever is the string and its length.

When we write this in C++ in terms of the class, then certainly we will have the same data members, but we write a constructor and in the constructor initialization, the initialization list, we string dup the given string s first and then we take the str field that has been created, that is the string that has been duplicated, we go and compute the length of that to set the length field and certainly we get the similar set of results when we execute them. So, this is the example we are talking of. Now, the question is, what if between these two data members if you swap them, will this program scale what or will this program have some difficulties. So, let us just try to swap and this is the example that you can see.

(Refer Slide Time: 07:56)



Program 14.04: A Simple String Class – Fails for wrong order of data members

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

class String {
    size_t len_; // Swapped members cause program crash (unhandled exception)
    char *str_;
public:
    String(char *s) : str_(strdup(s)), len_(strlen(str_)) { cout << "ctor: "; print(); }
    ~String() { cout << "dtor: "; print(); free(str_); }
    void print() { cout << "(" << str_ << ": " << len_ << ")" << endl; }
};
int main() {
    String s = "Partha";
    s.print();
    return 0;
}
```

- len_ precedes str_ in list of data members
- len_(strlen(str_)) is executed before str_(strdup(s))
- When strlen(str_) is called str_ is still uninitialized
- Causes the program to crash as shown in the message box

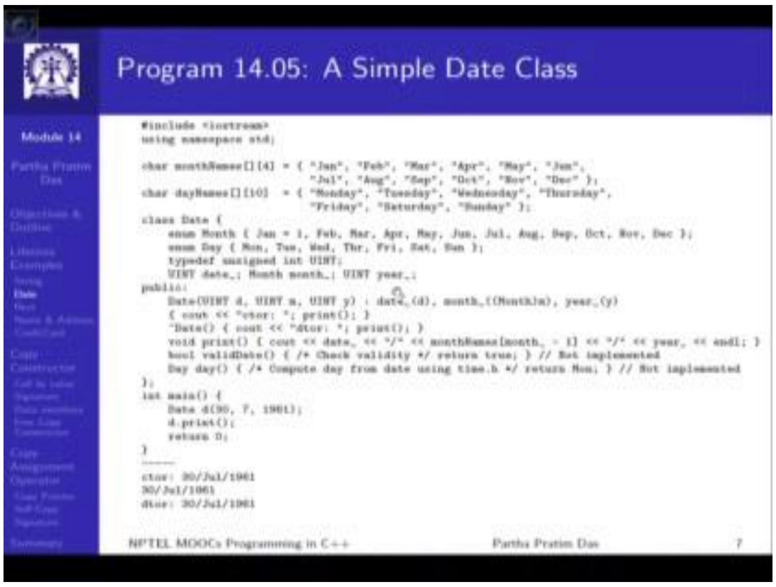
NPTEL MOOCs Programming in C++ Partha Pratim Das

The only difference from the earlier one is, we have swapped the order of these two data members. The constructor and the initialization list has not been changed, they are exactly the same, but just the order of the data members has been swapped and as you can see here, in this I have executed this in Microsoft Visual Studio, we can see that the program crashes.

The reason is understandable because as I said earlier that the order of initialization depends on the order of the data members on the list. So, since len here is given earlier, this particular initialization happens first and when this happens, the str field. This str field has not yet been initialized no memory has been allocated in this pointer and no string has been copied. So, the call to str len actually get what we commonly say garbage values and therefore, from that we get this kind of an error.

So, len will produce this is the points are highlighted here for your reference and because of this call to str len with uninitialized values, we will get this kind of an error. So, this is just to highlight through a simple example that the order of the data member can be critical and while writing the initialization list, you should be aware of the order in which the data members are actually created.

(Refer Slide Time: 09:42)



Program 14.05: A Simple Date Class

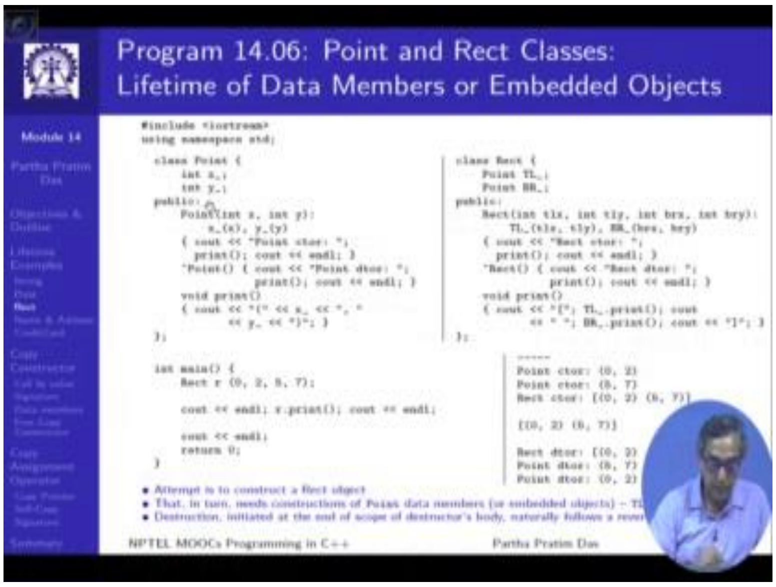
```
#include <iostream>
using namespace std;

char monthNames[][4] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
char dayNames[][10]  = { "Monday", "Tuesday", "Wednesday", "Thursday",
                         "Friday", "Saturday", "Sunday" };
class Date {
    enum Month { Jan = 1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };
    enum Day { Mon, Tue, Wed, Thr, Fri, Sat, Sun };
    typedef unsigned int UINT;
    UINT date_; Month month_; UINT year_;
public:
    Date(UINT d, UINT m, UINT y) : date_(d), month_((Month)m), year_(y)
    { cout << "ctor: "; print(); }
    ~Date() { cout << "dtor: "; print(); }
    void print() { cout << date_ << "/" << monthNames[month_ - 1] << "/" << year_ << endl; }
    bool validDate() { /* Check validity */ return true; } // Not implemented
    Day day() { /* Compute day from date using time.h */ return Mon; } // Not implemented
};
int main() {
    Date d(30, 7, 1961);
    d.print();
    return 0;
}
-----
ctor: 30/Jul/1961
30/Jul/1961
dtor: 30/Jul/1961
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 7

Now, let us look at few more simple examples. These are just initially for illustration and then we will put this together into building, a little bit of a bigger class so that we can see the order of lifetime or the order of construction destruction better. So, here we just show a simple date class. So, this date class has 3 data members; date, month and year. So, which are basically either numbers or they are enumerated types like Jan, Feb and so on.

So, which are basically sub types of integer we use them to initialize a date in the constructor and therefore, if you execute this program you will get this kind of an output where first the date will be constructed as in here, then it will be printed using this print and finally, it will get destructed. There is nothing more in this example will just use more of these examples subsequently.

(Refer Slide Time: 10:49)



Program 14.06: Point and Rect Classes: Lifetime of Data Members or Embedded Objects

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

class Point {
    int x_;
    int y_;
public:
    Point(int x, int y):
        x_(x), y_(y)
    { cout << "Point ctor: ";
      print(); cout << endl; }
    ~Point() { cout << "Point dtor: ";
               print(); cout << endl; }
    void print()
    { cout << "(" << x_ << ", "
           << y_ << ")"; }
};
```

```
class Rect {
    Point TL_;
    Point BR_;
public:
    Rect(int tlx, int tly, int brx, int bry):
        TL_(tlx, tly), BR_(brx, bry)
    { cout << "Rect ctor: ";
      print(); cout << endl; }
    ~Rect() { cout << "Rect dtor: ";
              print(); cout << endl; }
    void print()
    { cout << "["; TL_.print(); cout
           << " "; BR_.print(); cout << "]"; }
};
```

```
int main() {
    Rect r (0, 2, 5, 7);

    cout << endl; r.print(); cout << endl;

    cout << endl;
    return 0;
}
```

```
-----
Point ctor: (0, 2)
Point ctor: (5, 7)
Rect ctor: [(0, 2) (5, 7)]

[(0, 2) (5, 7)]

Rect dtor: [(0, 2)
Point dtor: (5, 7)
Point dtor: (0, 2)
```

- Attempt is to construct a Rect object
- That, in turn, needs constructions of Point data members (or embedded objects) – TL
- Destruction, initiated at the end of scope of destructor's body, naturally follows a rever

NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, I show another example for the lifetime, which is point and rectangle classes, so a point class has two data members the x and y coordinates of a point. It has constructors which initialize these points and a rectangle is basically a pair of points, where tl stands for top left and br stands for bottom right, that is a two corner points of a rectangle, if I specify that, then the rectangle is fully specified.

Now, if I want to initialize if you look carefully then I need actually initialize these two. So, for rectangle we have a constructor, which specifies four integers, of that the first two are meant for the x and y coordinate of the top left point the next two are x and y coordinates of the bottom right point. So, we take the first two and construct the tl component take the next two to construct the br component and the rectangle gets constructed fully.

So, if we look at if you try to see the order in which the constructors get called, we will see that the first this constructor gets called. Why, because these are the two integers which is tlx and tly, tl is the first data member of rect class. So, naturally this is the first initialization to happen in the initialization list. So, this initialization means that a point has to get constructed, which means that for this initialization this particular constructor will have to get called, this constructor gets called. Accordingly, this particular body of the constructor prints this output, it says that a point with 0, 2 has been constructed.

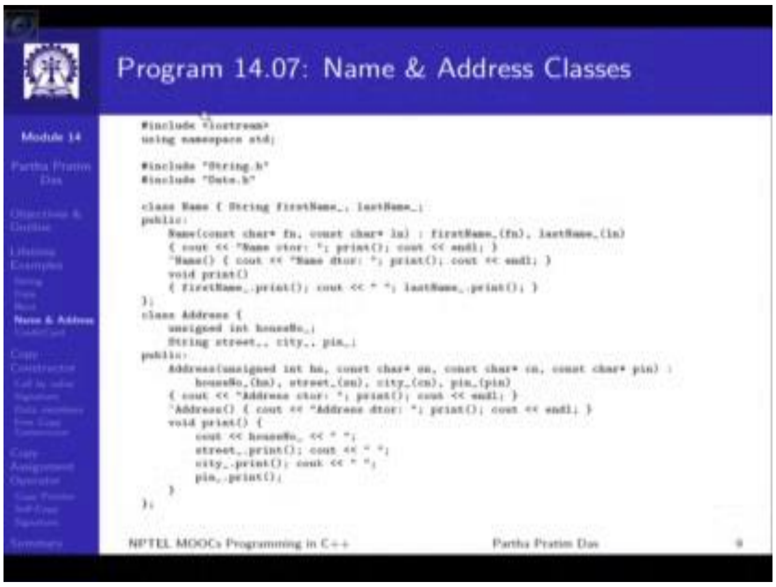
Next, the next two integers are taken; the second element in the initialization list is to be constructed. Again, another point is constructed to the same constructor and you get the output for that; then there is a print which shows that the rectangle, I am sorry. Then once that has been done, then the body of the rectangle constructed gets executed. So, you get this output which says that a rectangle with tl 0, 2 and br 5, 7 has been constructed, and then you print. So, the print comes in here and finally, the destruction process starts when the destruction process starts.

At this point, we have already explained that this rect is an automatic object. So, the destruction will start, when the scope of that automatic object ends, which is the closing bracket of the block of main. So, what will happen at this? This was the construction order. So, the destruction order will be exactly the reverse that is a LIFO.

So, the first call will happen to the destructor of the rectangle. So, first this gets executed and when it comes to the end of the destructor body of rectangle, at that time the rectangle is ready to be destroyed, which means that now the data members of the rectangle object has to be destroyed. So, the next one which was constructed just before this that is the br, this is basically for br that will get destructed. So, for that the destructor of the br, the destructor of br will get called and you get this output and finally, the first object that was constructed which is the object tl, the destructor for tl will get called and that results in the results in the messages that we have.

Now, it is kind of a one function destructor function calling the next and once that completes then it calls the next destructor function and so on and when all these are over then the destruction process is over. So, this is just too clearly highlighted to you, as to what happens when you have data members which are of user define types that is data members, which are not just built-in the data members, which actually have their constructors and destructors. So, you will have to remember that they are constructed in the order in which they are listed in the class and they are destructed in the reverse order of their construction.

(Refer Slide Time: 15:39)



Program 14.07: Name & Address Classes

Module 14

Partha Pratim Das

Objectives & Outline

Lifetime Examples

String

Date

Rect

Name & Address

CreditCard

Copy Constructor

Call by value

Signature

Data members

Free Copy Constructor

Copy Assignment Operator

Copy Pointer

Self-Copy

Signature

Summary

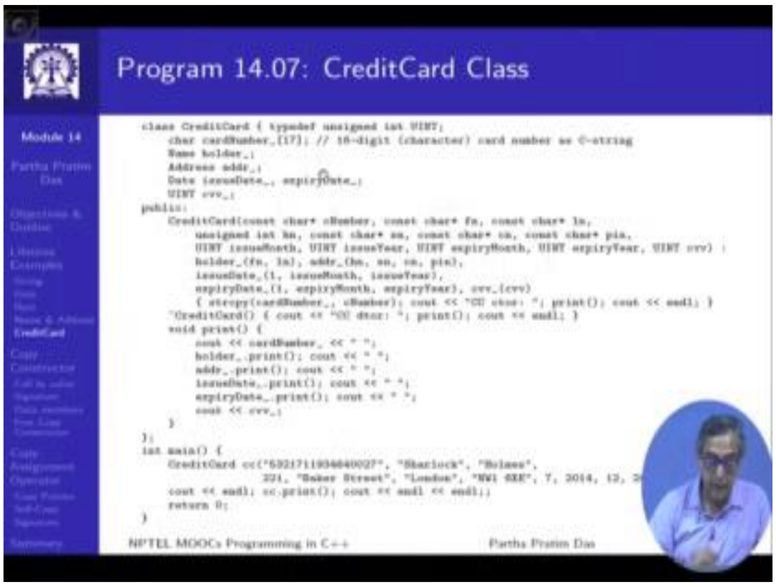
```
#include <iostream>
using namespace std;

#include "String.h"
#include "Date.h"

class Name { String firstName_, lastName_;
public:
    Name(const char* fn, const char* ln) : firstName_(fn), lastName_(ln)
    { cout << "Name ctor: "; print(); cout << endl; }
    ~Name() { cout << "Name dtor: "; print(); cout << endl; }
    void print()
    { firstName_.print(); cout << " "; lastName_.print(); }
};
class Address {
    unsigned int houseNo_;
    String street_, city_, pin_;
public:
    Address(unsigned int hn, const char* sn, const char* cn, const char* pin) :
        houseNo_(hn), street_(sn), city_(cn), pin_(pin)
    { cout << "Address ctor: "; print(); cout << endl; }
    ~Address() { cout << "Address dtor: "; print(); cout << endl; }
    void print() {
        cout << houseNo_ << " ";
        street_.print(); cout << " ";
        city_.print(); cout << " ";
        pin_.print();
    }
};
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 9

Let us move on, in the next couple of slides we will just build up to show you a bigger example. So, in this slide we show two more classes, one is a name class designed to keep or maintain names of a person. So, it has two different data members; first name and last name. They are of the type string that we have already discussed. So, we have included the string class as in the header. So, for these two data members we provide the parameters to the constructor and they get constructed in the initialization list.

We have another class address which is to keep a house number and three strings; street, city and pin, which are strings describing the respective data members. So, they will be constructed in the constructor of address and will be printed subsequently. So, this is what using the string, now we have a name class which can maintain names of persons and address class which can maintain address of persons and so on. We are just building this up to for a bigger example.

(Refer Slide Time: 16:59)



Module 14

Partha Pratim Das

Objectives & Outline

Lifetime Examples

String

Date

Rect

Name & Address

CreditCard

Copy Constructor

Call by value

Signature

Data members

Free Copy Constructor

Copy Assignment Operator

Copy Pointer

Self-Copy

Signature

Summary

Program 14.07: CreditCard Class

```
class CreditCard { typedef unsigned int UINT;
    char cardNumber_[17]; // 16-digit (character) card number as C-string
    Name holder_;
    Address addr_;
    Date issueDate_, expiryDate_;
    UINT cvv_;
public:
    CreditCard(const char* cNumber, const char* fn, const char* ln,
        unsigned int hn, const char* sn, const char* cn, const char* pin,
        UINT issueMonth, UINT issueYear, UINT expiryMonth, UINT expiryYear, UINT cvv) :
        holder_(fn, ln), addr_(hn, sn, cn, pin),
        issueDate_(1, issueMonth, issueYear),
        expiryDate_(1, expiryMonth, expiryYear), cvv_(cvv)
        { strcpy(cardNumber_, cNumber); cout << "CC ctor: "; print(); cout << endl; }
    ~CreditCard() { cout << "CC dtor: "; print(); cout << endl; }
    void print() {
        cout << cardNumber_ << " ";
        holder_.print(); cout << " ";
        addr_.print(); cout << " ";
        issueDate_.print(); cout << " ";
        expiryDate_.print(); cout << " ";
        cout << cvv_;
    }
};
int main() {
    CreditCard cc("5321711934640027", "Sherlock", "Holmes",
                  221, "Baker Street", "London", "NW1 6XE", 7, 2014, 12, 2
    cout << endl; cc.print(); cout << endl << endl;;
    return 0;
}
```

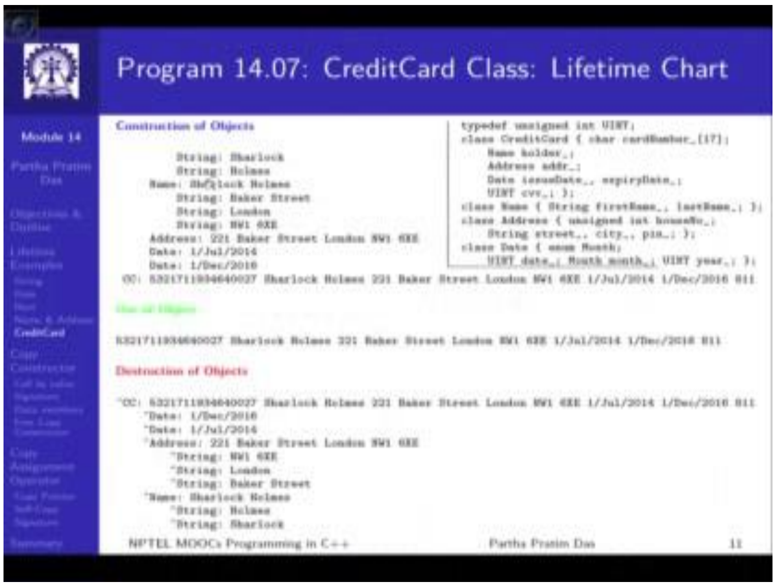
NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, this is the example which we were interested in; let consider a class to represent a credit card. I am sure all of you have or at least have known, how a credit card can be described. A credit card has a card number which is typically a 16 character string. It certainly has embossed in it, the name of the holder, the address of the holder is given though is not written on the card. A card has an issue date, it has an expiry date and on the reverse of the card, we have a verification number call the cvv number. So, if you want to describe a credit card object, then this typically; this is a very minimal description that we can have for a credit card object.

So, for at the constructor which is here a big list, this is actually the prototype of the constructor. This is a three lines which give us the string giving the credit card number then it gives the names; first and second name of the holder. Then it gives the house number, street name, city, and pin for the address of the holder. Then it gives the issue date, that is issue and expiry dates. These use the date class that we have defined and finally, the cvv and we use the initialization list to initialize.

Therefore, if we use this class to create a credit card object for say, Mister Sherlock Holmes, staying in 221 Baker street, London, then the invocation of the constructor will look something like this and with that let us take a look into how will the construction-destruction, how will the lifetime look like .

(Refer Slide Time: 19:01)



Program 14.07: CreditCard Class: Lifetime Chart

Construction of Objects

```
        String: Sharlock
        String: Holmes
    Name: Sharlock Holmes
        String: Baker Street
        String: London
        String: NW1 6XE
    Address: 221 Baker Street London NW1 6XE
    Date: 1/Jul/2014
    Date: 1/Dec/2016
 CC: 5321711934640027 Sharlock Holmes 221 Baker Street London NW1 6XE 1/Jul/2014 1/Dec/2016 811
```

```
typedef unsigned int UINT;
class CreditCard { char cardNumber_[17];
    Name holder_;
    Address addr_;
    Date issueDate_, expiryDate_;
    UINT cvv_; };
class Name { String firstName_, lastName_; };
class Address { unsigned int houseNo_;
    String street_, city_, pin_; };
class Date { enum Month;
    UINT date_; Month month_; UINT year_; };
```

Use of Object

```
5321711934640027 Sharlock Holmes 221 Baker Street London NW1 6XE 1/Jul/2014 1/Dec/2016 811
```

Destruction of Objects

```
~CC: 5321711934640027 Sharlock Holmes 221 Baker Street London NW1 6XE 1/Jul/2014 1/Dec/2016 811
    ~Date: 1/Dec/2016
    ~Date: 1/Jul/2014
    ~Address: 221 Baker Street London NW1 6XE
        ~String: NW1 6XE
        ~String: London
        ~String: Baker Street
    ~Name: Sharlock Holmes
        ~String: Holmes
        ~String: Sharlock
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 11

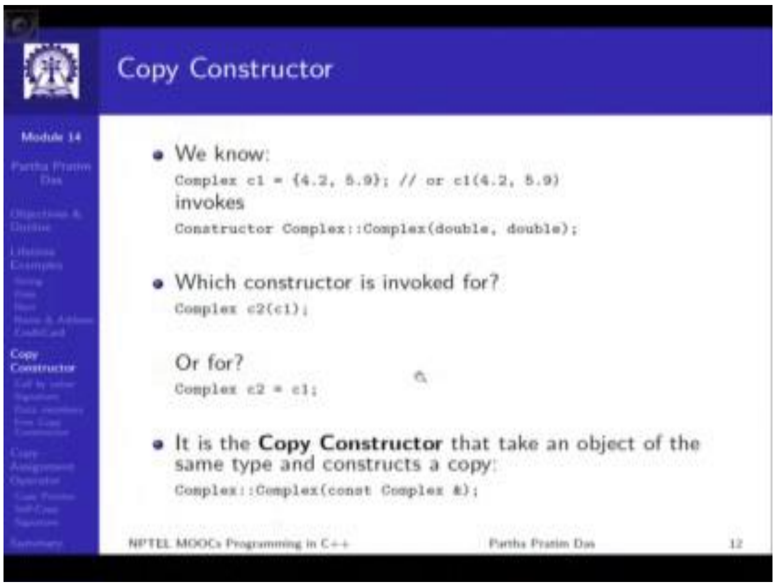
So, this is how the object is getting constructed and just for reference, I have put the definition of the credit class and other classes here, this is just for your reference, but this is the output this the whole thing is output. So, this is the construction phase, where naturally if you want to construct, the first field here, is a card number which is nothing, but a C string. So, there is no constructed to be explicitly called. So, therefore, there is no output, but the next certainly what will get constructed is the holder.

So, the constructor of name will get called. So, for that then constructor of the name has two fields, the first name and the second name which are strings. So, first thing that gets called is the first name string then the second name string and then the construction of the name object. Then address starts, again the house number is an integer. So, that has no constructor, but the other three strings will get constructed next, which give you the street name, the city name and the pin and then the address object gets constructed. Then subsequently the two data objects get constructed here and here and finally, the credit card object get constructed.

So, this is just shows and I have intentionally done this kind of indentation to show that the more indented is a statement is what has been constructed earlier. So, this is the order in which this will get constructed. So, these two will construct this, these two will finally give these. These are the separate objects and all of these together will finally construct the credit card object. Then the credit card can be used which we just show by print and then the destruction order if you look at and I will leave this as for your study to look at carefully the order of the destruction is exactly in the reverse order of.

So, if you just read this list, bottom to top that is the order in which the destruction will happen for this is the objects and that is just for your practice also, I would suggest it is naturally in my quick description you may not be able to see all of the details here, but the whole program is given in the presentation and you have or if go through the presentation and I would suggest that you also try to run it in your system and try to see that you are getting the same result and get convinced about what is the different lifetime of objects that you get when you have nested objects this data member like this.

(Refer Slide Time: 22:06)



Copy Constructor

Module 14

Partha Pratim Das

- We know:
  `Complex c1 = {4.2, 5.9}; // or c1(4.2, 5.9)`
  invokes
  `Constructor Complex::Complex(double, double);`
- Which constructor is invoked for?
  `Complex c2(c1);`

  Or for?
  `Complex c2 = c1;`
- It is the **Copy Constructor** that take an object of the same type and constructs a copy:
  `Complex::Complex(const Complex &);`

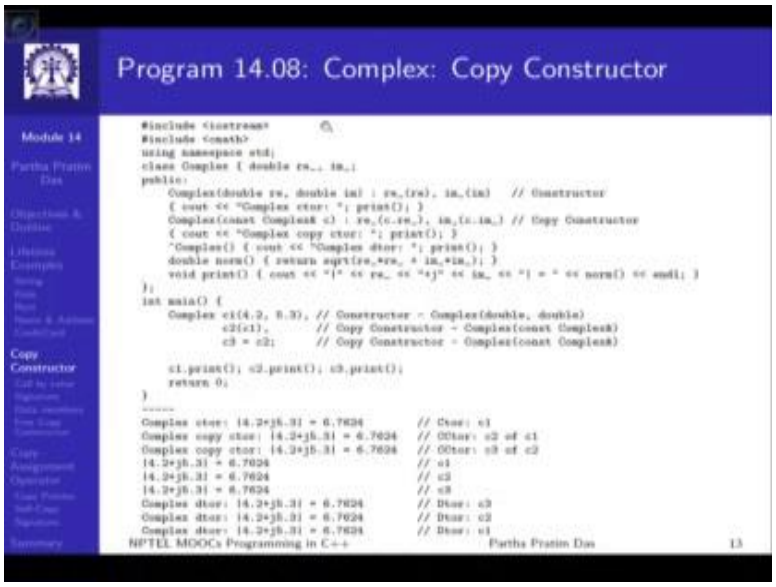
NPTEL MOOCs Programming in C++ Partha Pratim Das 12

Now, we will move into discussing a new kind of constructor and the reason I discussed object lifetime here, is we will see that with this new kind of constructor the lifetime of objects will get new dimensions to understand. So, we just start by showing a simple example say, we know complex can be constructed in this way and that will call a complex constructor like this we have seen the complex class quite often, but suppose I write it like this what does that mean.

So, just see the difference, here it is written like this, here it is or if I write it like this. There is a main difference is here, I have specified the parameters of the constructor the two double numbers one after the other by comma. Whereas, here I have used a complex object itself to initialize another complex object C 2.

When I try to do this, then I am doing a construction which is a special kind of construction which is known as the copy construction and a copy constructor looks like this, this is just a constructor. So, it is complex colon colon complex, only difference being that it takes another complex object as a parameter and it takes it as a reference parameter and we are using const in front of this, let us slowly understand why we do all these.

(Refer Slide Time: 22:53)



Program 14.08: Complex: Copy Constructor

Module 14

Partha Pratim Das

Objectives & Outline

Lifetime Examples

Copy Constructor

Copy Assignment Operator

Summary

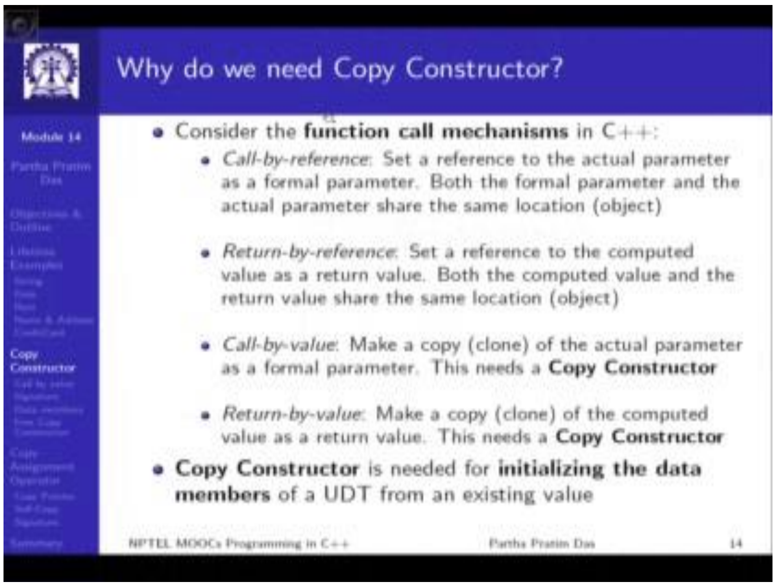
```
#include <iostream>
#include <cmath>
using namespace std;
class Complex { double re_, im_;
public:
    Complex(double re, double im) : re_(re), im_(im)   // Constructor
    { cout << "Complex ctor: "; print(); }
    Complex(const Complex& c) : re_(c.re_), im_(c.im_) // Copy Constructor
    { cout << "Complex copy ctor: "; print(); }
    ~Complex() { cout << "Complex dtor: "; print(); }
    double norm() { return sqrt(re_*re_ + im_*im_); }
    void print() { cout << "|" << re_ << "+j" << im_ << "| = " << norm() << endl; }
};
int main() {
    Complex c1(4.2, 5.3), // Constructor - Complex(double, double)
            c2(c1),       // Copy Constructor - Complex(const Complex&)
            c3 = c2;      // Copy Constructor - Complex(const Complex&)

    c1.print(); c2.print(); c3.print();
    return 0;
}
-----
Complex ctor: |4.2+j5.3| = 6.7624        // Ctor: c1
Complex copy ctor: |4.2+j5.3| = 6.7624   // CCtor: c2 of c1
Complex copy ctor: |4.2+j5.3| = 6.7624   // CCtor: c3 of c2
|4.2+j5.3| = 6.7624                      // c1
|4.2+j5.3| = 6.7624                      // c2
|4.2+j5.3| = 6.7624                      // c3
Complex dtor: |4.2+j5.3| = 6.7624        // Dtor: c3
Complex dtor: |4.2+j5.3| = 6.7624        // Dtor: c2
Complex dtor: |4.2+j5.3| = 6.7624        // Dtor: c1
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 13

Let us look into an example here is the total complex class. So, this is the constructor that we have seen earlier. These are copy constructor that is will specifically have to look into this part of the court that it takes a complex object and then it initializes the data members from the data members of the complex objects. So, the object that you want to copy from is C - the re data member, re underscore data member is c dot re. So, we take that and put it to re. Similarly, we take c dot im and put it to im. So, basically what happens is the new object which is getting constructed by this constructor has identical values in re and im fields from the object it is copying from.

So, in this context if you look into the construction of a couple of different objects then naturally the first construction is simply using are parameterized constructor of two doubles. The second one will use the copy constructor because it takes the C 1 object which is a complex object and uses the copy constructor to construct C 2. Similarly, the next one will take C 2 and copy construct to find of to construct C 3 as copies of these. So, after this construction if you try to print them as we do it here then we finds that all of them have become identical. They have this identical construction values and in the print they show identical values and in the destruction also they show identical values. So, this is how copies can be made of objects very easily to this copy constructor mechanism.

(Refer Slide Time: 25:56)



Why do we need Copy Constructor?

Module 14

Partha Pratim Das

Objectives & Outline

Lifetime Examples
String
Date
Rect
Name & Address
CreditCard

Copy Constructor
Call by value
Signature
Data members
Free Copy Constructor

Copy Assignment Operator
Copy Pointer
Self-Copy
Signature

Summary

- Consider the **function call mechanisms** in C++:
  - *Call-by-reference*: Set a reference to the actual parameter as a formal parameter. Both the formal parameter and the actual parameter share the same location (object)
  - *Return-by-reference*: Set a reference to the computed value as a return value. Both the computed value and the return value share the same location (object)
  - *Call-by-value*: Make a copy (clone) of the actual parameter as a formal parameter. This needs a **Copy Constructor**
  - *Return-by-value*: Make a copy (clone) of the computed value as a return value. This needs a **Copy Constructor**
- **Copy Constructor** is needed for **initializing the data members** of a UDT from an existing value

NPTEL MOOCs Programming in C++ Partha Pratim Das 14

Then the question certainly is why we need copy constructors and there are primarily two reasons that copy constructors are provided or need to be provided in C++. To understand, first consider the function call mechanism. As we have already seen, when we talked about better C part of C++, we talked about references; we talked about call by reference or return by reference mechanism. So, if you just recap those then there are four things, we can do we can call by reference in which case the actual formal parameter actually takes the reference to an actual parameter, that is both the formal and the actual parameter share the same memory location, share the same object.

The same thing happens at the time of return, the value that you return by from the function and the value that you get from the calling function. Basically, these two objects are same if you would return by reference whereas, if you do call by value then you need to make a copy of the actual parameter as a formal parameter, you need to make a copy. Please, this is what most significant point is.

Now, as long as the objects that we pass or the values that we pass, they were of the built-in type making this copy was straight forward because it was just making bit copy of an int or a double or a character and so on. But, when we have user defined objects then we actually need, given actual parameters. So, this is an actual parameter which is say an object C and I need a formal parameter say f. So, this f will have to be an object of the same type and, but it must be different from C because I want to make a copy and the values of the data members of C, somehow needs to be copied to f. So, the purpose of the copy constructor is significantly to achieve this process of call by values.

So, if a user defined type does not have or is not provided with copy construction process if this is not provided with the copy construction then the consequences, the objects of that class, objects of that user define type cannot be passed as call by value mechanism to any function. The similar observations will happen if you want to return a value, return something from a function by value because we will again need copy constructor to copy the value. The other situation where copy constructor is needed is for initializing the data members.

You have already seen that we have been copying one value into another regularly in the previous examples of object lifetime, but the copied values were typically where whatever data members we had for construction they were typically built-in types, but if I have to copy a value of a user define type as a data member then I will again faced with the same situation as the call by value situation. So, initializing the data members of UDT will also need the existence of a copy constructor without that a data member of that corresponding type cannot be defined.

We have just seen; we have revisited the object lifetime and we have specifically taken a deep look into the different object lifetime scenarios, particularly with user define types and discussed the issue of ordering of data members and their consequence on the object lifetime and we have just got introduced to the copy constructor.