Lecture 04

Lecture contents:

1- Constructor function
2- Function overloading
3- Copy constructor function
4- Destructor function
5- Demo code

المحاضرة دي هنخلص فيها اللي عايزين نتكلم فيه عن classes .. وهنبدأ نتكلم عن حاجة جديدة المحاضرة الجاية ان شاء الله ..

Constructor function:

"A class constructor is a member function that is automatically called when an object of the class is instantiated (created)."

Notes:
1- Constructor name has the same name as the class
2- Constructor is used to initialize objects of a class to a well defined state
3- There can be more than one constructor per class

يعني ايه بقى الكلام اللي فوق دة؟؟
فيه حاجة في ال class اسمها constructor .. دي member function عادية جداً بس بيتعمل لها automatic call لما نيجي نعرف object جديد، عشان كدة لازم اسم ال constructor يبقى هو نفس اسم ال class بالظبط .. طيب ايه فايدة ال constructor دة بقى؟؟ .. فايدته انه مش بيخلي فيه garbage متخزنة في ال class اول ما يتعمل ((يعني تخيل مثلا زمان لما كنا بنكتب int x; .. حد كان يقدر يقول ال x دي بكام دلوقتي؟؟ الاجابة لأ لأن انا مش مديها initial value فانا كدة سيبت ال x يتحط في الميموري بتاعتها اي value وخلاص .. فكان الحل اننا نقول مثلا int x = 0; .. فكدة انا عارف ان x دلوقتي فيها 0 فأقدر اشتغل من غير ما ابقى خايف من مشكلة غريبة تحصل)) .. فال constructor بيعمل كدة بالظبط مع ال class objects، يعني من الاخر مادام ال constructor موجود في ال class بتاعي يبقى كل ال member variables اللي هتبقى عندي معمول لها initialization ..
ال note رقم 3 بقى بتقول اني ممكن اعمل كذا constructor في ال class الواحد .. واحنا قلنا ان ال constructor دة مش اكتر من function عادية زي اي function وقلنا ان ال constructor لازم يبقى اسمه هو نفس اسم ال class .. يعني معنى كدة اني عشان اعمل كذا constructor فانا محتاج اعمل كذا function بنفس الاسم بالظبط ... بس هو دة ينفع؟؟! .. اه ينفع اعمل كذا function بنفس الاسم ودة اللي اسمه function overloading ..

Function overloading:

عشان اعرف اعمل function overloading بنفس الاسم لازم يكون فيه اختلاف ال compiler يكون فاهمه عشان يعرف هو هيعمل call لأنهي function منهم بالظبط .. الاختلاف دة ممكن يبقى في عدد ال parameters اللي ال function بتاخدها او ال parameter types .. لو مش فاهم كمل عادي والدنيا هتظبط لما نكتب الكود دلوقتي

نرجع بقى تاني لل constructor :

"Both constructor & destructor must be **Public Member Functions**"

ودة بديهي لأني لو عملت غير كدة يبقى انا مش فاهم ال constructor & destructor

“Constructor & destructor functions must **Return Nothing**”

دايماً بتبقى void functions ..
وكمان وانا بكتب الكود مش بكتب في
ال declaration بتاعها انها void ..
هنشوف دلوقتي

نكتب الكود بقى ..

Complex.h

```
class Complex {
public:
        Complex ();    //default constructor
        Complex (double re, double im);      //Another constructor
        Complex (int r, int m); //another constructor, we won't implement nor use it as it's just for
                                illustration
        Complex (double x);  //another constructor .. we won't use it as it's just for illustration

        Complex (const Complex &c); //copy constructor

        ~Complex();   //destructor
        void Add (Complex c);
        Complex operator + (Complex c); //هنتكلم عن البتاع دة في اخر المحاضرة

private:
        double real;
        double imag;
};
```

لو لاحظت مفيش return type اتحط قبل ال constructor او ال destructor .. ولا حتى void ودة لأنهم special functions
مش بيرجّعوا حاجة
تعالى بقى نـ implement ال member functions دي :

Complex.cpp

```
Complex :: Complex (){ //if this constructor is called, both real and imaginary values are initialized to 0
        real = 0.0;
        imag = 0.0;
}

Complex :: Complex (double re, double im){ //if this constructor is called, then the real part is assigned
                                            to value = re, and imaginary part is assigned to value = im
        real = re;
        imag = im;
}
```

```
Complex :: Complex (const Complex &c) { //copy constructor, assigns the real value of 'c' to the real value of this object, and imaginary value of 'c' to imaginary value of this object
        real = c.GetReal();
        imag = c.GetImag();
}
```

يبقى احنا كدة يا معلم فهمنا الكلام اللي كان مكتوب فوق وخلاص كدة كتبنا الدرايفر بتاعنا .. ال user بقى يقدر يكتب ال program بتاعه دلوقتي :

Client program (main.cpp)

```
#include "complex.h"
int main (){
Complex x; /* create a class object named x, and since we have a constructor, it will be automatically
              called .. no arguments called then the default constructor is called */

Complex y(10.5, -3.6); /* a class object named y is created, a constructor is called, since the function
                          parameters are 2 double values, then the second constructor is called */

double r = 6.5;
double p = -3.2;
Complex z(r, p); // 2nd constructor is called
Complex a(r, p, 5); // syntax error, we haven't implemented a constructor which takes 3 parameters

x.Add(y); // copy constructor is called


Complex j = x;
Complex z(x);

}
```

الكود اللي فوق دة فيه 1statement بيعمل syntax error فمكتوب بلون احمر .. وفيه 3statements عن حاجة اسمها copy constructor هنتكلم عليها دلوقتي وال 3statements دول هم اللي معمول لهم highlighting بلون اصفر .. اشطة كدة؟؟

ايه بقى ال copy constructor دة؟؟

Copy constructor:

This constructor is called in 3 cases :

1- Initialization of an object from another of the same type

2- Upon function call using pass by value

3-Upon returning from a function

When an object is passed by value to a function, a bitwise copy of the object is made in the function parameter, This is Ok for simple data types, however if an object contains pointers, this can lead to disasters.

نفسر بقى كل case من دول ..
اول case دي اللي هي complex z(x); اللي موجودة في ال main ... عملت object جديد اسمه z وعايز اعمل له initialization ويبقى فيه نفس ال real & imag بتوع ال object اللي اسمه x
فدة كدة بيعمل call ل copy constructor .. لأنه بيعمل ال init بتاعه عن طريق انه يعمل copy من object ل object

وبرضه لو بصينا على Complex j= x; هنلاقي اننا برضه بنعمل declaration ل object جديد اسمه j وعايزينه ياخد نفس قيمة ال object اللي اسمه x ... هل دة بقى هيعمل call ل copy constructor برضه ولا لأ؟؟
الإجابة هي على حسب .. لو انا عامل copy constructor يبقى هيعمل call .. لو انا مش عامل يبقى كدة واللي هيحصل ان ال compiler هيعمل default copy constructor من نفسه و ال default copy constructor دة هيعمل bitwise copy، يعني الداتا اللي في ال x هيتعمل لها copy لل j ب bit by bit

تاني case بقى اللي هي دي x.Add(y); .. انا كدة عملت call ل function اسمها Add وبتاخد argument passed by value اسمه y .. والكلام دة كله جوة ال object اللي اسمه x .. لو بصيت على ال implementation بتاع ال Add في المحاضرة اللي فاتت هتلاقي ان فيها copy constructor، بس الزيادة ان في ال add ال copy constructor بيزود قيمة معينة بعد ما يـ copy

تالت case بقى .. مش مكتوبة في الكود فوق فتعالى نتخيلها
هنتخيل اننا كتبنا function اسمها myFunc() .. ال function دي بتعمل object اسمه c وبعدها تعمل حاجات تانية وفي الاخر بتـ return c :

```
Complex :: myFunc(){
        Complex c;
        .
        .
        .
        .
        return c;
}
```

وبعد كدة جينا عملنا call لل function دي في ال main :

```
y = x.myFunc();
```

هنا بقى فيه مشكلة .. c دة المفروض ان ال scope بتاعه جوة ال function اللي اسمها myFunc بس .. يعني اول ما ال function دي تخلص، ال c دة هيبقى destroyed ومش هيبقى موجود .. بس انا محتاج ال c دة عشان هو ال return بتاع ال function اللي هاخده في ال main واشوف بقى بعد كدة هعمل بيه ايه
فلو كان ال c دة int مثلا او اي data type عادية ساعتها ال compiler بيحل المشكلة دي عن طريق انه يعمل temp variable ياخد ال return بتاع myFunc ويحطه في ال y (وبرضه لو كان global variable ال compiler بيعمل كدة، برغم ان المشكلة دي خلاص مش موجودة)
بس بالنسبة بقى ل object من class .. هنعمل ايه؟؟
ساعتها ال copy constructor هيبقى هو الحل .. بس لو ال copy constructor مش موجود؟؟ .. يبقى هيحصل bitwise copy

Destructor function:

It’s a member function that is automatically called when the lifetime of an object belonging to class ends (when the object is destroyed).
Notes:
1- Destructor must have the same as the class, preceded by a ‘~’ symbol : **~class_name**
2- There can be only one destructor
3- It’s used to clean up any explicitly allocated resources by the objects

نشرح بقى الحوار دة .. ال destructor بيتعمل له automatic call برضه بس لما ال scope بتاع ال object يخلص .. لو اخدنا مثال ال return اللي فات يبقى ال destructor هيشتغل اول ما يشوف كلمة return عشان يـ destroy ال c ..
عشان كدة محتاجين نعرف ال scope او ال lifetime بتاع كل object عشان نعرف امتى هيتعمل له destroy :
1- Global object:
ال object بيتعرّف برة اي function ومش بيتعمل له destroy غير لما البرنامج كله يخلص

2- Local object:
ال object بيتعرّف جوة function معينة (زي ال main مثلا او اي function تانية) .. وبيتعمل له destroy لما البرنامج يخرج من ال function

تعالى بقى نكتب كود ال destructor … بس احنا قولنا ان في العادي اللي بيحصل في ال code ان ال RAM بيتحط فيها ال variables لحد ما ال scope بتاعها يخلص وبعد كدة بيحصل لها destruction .. ففي الحقيقة في ال case بتاعتنا ال destructor اللي احنا عايزين نعمله دلوقتي دة مالوش اي لازمة لأن كدة كدة لما ال scope بتاع ال object او ال variable يخلص هو هيتعمل له automatic destruction من غير ما انا اكتب destructor يعمل كدة .. بس هو مهم طبعاً في cases تانية وقدام هنحتاجه فعلاً ونعرف فايدته
بس دلوقتي خلينا نكتب ال destructor اللي مش مهم دلوقتي :

```
Complex :: ~Complex() {
        cout << “ Program terminated, bye \n” ;
}
```

كدة احنا خلصنا المحاضرة .. اللي جي هيبقى demo على الاكواد بتاعت المحاضرة دي والمحاضرة اللي فاتت وشوية notes .. اي كود الدكتور بعتهولنا هتلاقيه في فولدر اسمه codes على الدرايف

في ال .h هنلاقي الكلام دة :

```
Complex(const Complex &c);
/* Note it has to be defined as above using pass by reference.
 * Complex (Complex c);
 * will not work. Why?
 */
```

ليه عملنا كدة و ايه الكومنت دة؟؟!
الاجابة ان احنا ماينفعش نعمل copy constructor ب argument passed by value .. ودة لأن ال pass by value عشان يتعمل محتاج يعمل copy لل constructor وعشان يعمل copy محتاج يـ call ال copy constructor .. فال copy constructor دة كمان له argument passed by value فهنفضل في infinite recursions مش هنخرج منها .. فعشان كدة اي copy constructor لازم يبقى pass by reference .. (ماتتلخبطش عشان الدكتور في المحاضرة كان بيشرح ال copy constructor على اساس انه بياخد pass by value .. بس المحاضرة اللي بعدها قال ان الكلام اللي اتقال في المحاضرة دة غلط وان دة الصح)

بالمرة بقى مادام الكلام جه في ال pass by value & pass by reference :

```
//void Add(Complex c);
void Add(const Complex &c);
```

ليه هنا الدكتور خلي ال Add تاخد by reference مش by value ؟؟
الاجابة عشان ال argument اللي Add بتاخده دة مش simple data type (حاجة مكلكعة يعني بتاخد ميموري كبيرة) .. فلو عملتها pass by value مش هيبقى غلط بس هيحصل copy في الميموري وبالتالي هنستنزف ميموري اكتر .. في حين ان احنا لو عملناها pass by reference مش هيحصل copy لأني هشتغل على نفس ال object
بس عندي مشكلة .. ال pass by reference بيسمحلي اني اغير في ال argument حتى لو بالغلط .. وانا مش عايز اغير اي حاجة فيه .. فهنحل المشكلة دي عن طريق اننا نخلي ال argument دة const فكدة خلاص مفيش مجال اني ألعب في ال argument دة وبكدة انا عملت اللي ال pass by value بتعمله بس بميزة زيادة اني مش بستنزف الميموري على الفاضي

تعالى بقى نـ run كود ال main دة :

```
#include <cstdlib>

using namespace std;

#include "complex.h"

/*
template <class T>
T GetMax (T a, T b) {
  return (a>b ? a:b);
}
*/


int main(int argc, char** argv) {

    Complex x;
    Complex y(5,7); // implicit casting to double will work!

    Complex z=x;
    //int i, j;
    //Complex u(y);
    //y.PrintComplex();

    x.ReadComplex();
    //y.real = 5.9809;
    y = z; //This is assignment operator. No copy constructor called

    y.SetReal(5.6);
    y.SetImag(12.4);

    x.Add(y);
    x.PrintComplex();

    return 0;
}
```

بس قبل ما نـ run تعالى نتوقع ايه اللي هيحصل
عشان نـ initialize x يبقى هنعمل call ل Complex::Complex()

وعشان نـ initialize y يبقى هنعمل call ل Complex::Complex(double r, double im) .. بس ثواني دة 5 و 7 دول integer مش double يعني مش نفس ال data type فالمفروض يحصل compilation error .. هقولك لأ ال compiler هيعمل implicit casting to double يعني هيشوف ال 5 وال 7 دول على انهم double مش int

بعد كدة عايزين نـ initialize z بقيمة x فهيحصل call لل copy constructor دة
Complex :: Complex (const Complex &c)

و x هي اللي هتكون ال argument بتاع ال copy constructor
بس ثواني برضه .. هو ليه حصل call لل copy constructor ؟؟ ليه ال compiler مقالش انها assignment operator فهيدور على assignment operator overloading ولو مالقاش يبقى هيعمل bitwise copy وخلاص؟؟
الإجابة ان دة ال standard بتاع اللغة .. لو عملت declaration ل object وفي نفس الوقت عملت initialization له في نفس ال statement يبقى ال copy constructor هو اللي هيشتغل .. ولو مش موجود يبقى bitwise copy

بس لو كنا عملنا كدة :

```
Complex z;
z = x;
```

ساعتها دة كان هيبقى assignment operator وال compiler هيدور على assignment operator overloading (مش هيدور على copy constructor) .. ولو مالقاش يبقى هيعمل bitwise copy (زي ال case بتاعت y=z في الكود اللي فوق) .. وبرضه ماتنساش ان ال bitwise copy دة مش مشكلة خالص مادام ماعندناش pointers

بس بقى .. بعد كدة هنعمل الحاجات البسيطة بتاعت المحاضرة اللي فاتت ال add وال set وكدة
لما نعمل بقى run للكود دة المفروض يطلع اللي احنا توقعناه :

```
Constructor 1 called  on object:0x7ffd63c537a0
Constructor 2 called  on object:0x7ffd63c537b0
Constructor 3 (copy constructor) called on object: 0x7ffd63c537c0
Enter Real Part:  2
Enter Imag Part:  3
end of add
7.6+j15.4
Destructor called  on object: 0x7ffd63c537c0
Destructor called  on object: 0x7ffd63c537b0
```

زي الفل .. نلعب شوية بقى

لو حبينا نستعبط ونـ access ال member value اللي اسمه real من ال main ايه اللي هيحصل .. هنشيل بس الكومنت من line 33 :

```
22  int main(int argc, char** argv) {
23
24      Complex x;
25      Complex y(5,7); // implicit casting to double will work!
26
27      Complex z=x;
28      //int i, j;
29      //Complex u(y);
30      //y.PrintComplex();
31
32      x.ReadComplex();
33      y.real = 5.9809;
34      y = z; //This is assignment operator. No copy constructor called
35
36      y.SetReal(5.6);
37      y.SetImag(12.4);
38
39      x.Add(y);
40      x.PrintComplex();
41
42      return 0;
43  }
44
```

ساعتها دة اللي هيطلع لما نعمل run :

```
main.cpp: In function 'int main(int, char**)':
main.cpp:33:7: error: 'double Complex::real' is private within this context
     y.real = 5.9809;
       ^~~~
In file included from main.cpp:12:0:
complex.h:39:12: note: declared private here
     double real, imag;
            ^~~~
```

بيقولك بطل لعب بس بالأدب :'D

المهم يعني نكمل لعب .. تعالى نغير ال prototype بتاع Add .. نخليه pass by value بدل pass by reference
من الاخر في Complex.cpp هنحط كومنت في سطر 65 ونشيل الكومنت من سطر 66 .. زي كدة

```
65  //void Complex::Add(const Complex &c)
66  void Complex::Add(Complex c)
67  {
68      //c.real = 0.0;
69      real += c.real;
70      imag += c.imag;
71
72      cout << "end of add" << "\n";
73  }
```

وهنغير في Complex.h برضه

ونشوف بقى ايه اللي هيفرق عن اول run عملناه :

```
Constructor 1 called  on object:0x7ffd9f0e2f20
Constructor 2 called  on object:0x7ffd9f0e2f30
Constructor 3 (copy constructor) called on object: 0x7ffd9f0e2f40
Enter Real Part:  2
Enter Imag Part:  3
Constructor 3 (copy constructor) called on object: 0x7ffd9f0e2f50
end of add
Destructor called  on object: 0x7ffd9f0e2f50
7.6+j15.4
Destructor called  on object: 0x7ffd9f0e2f40
Destructor called  on object: 0x7ffd9f0e2f30
Destructor called  on object: 0x7ffd9f0e2f20
```

اللي حصل في ال code ان Add بقت pass by value .. يعني في الأول ال compiler هيحتاج ياخد الداتا اللي جوة ال argument ويعمل لها copy في مكان جديد في الميموري وبعد كدة يبدأ يشتغل .. وال argument اللي add بتاخده دة نوعه Complex .. فهيحصل copy ازاي؟؟ عن طريق ال copy constructor .. عشان كدة في ال terminal لقينا ان فيه copy constructor زيادة اتعمل ل this object جديد .. وبالتالي لما Add تخلص محتاجين نعمل لها destroy .. ففيه destructor زيادة اشتغل

من الاخر يعني :
لو عملنا function ليها arguments passed by value من class object .. يبقى كدة هيحصل call لل copy constructor وبعد كدة هيحصل call لل destructor
أما لو عملنا pass by reference فساعتها مش هيحصل copy constructor ولا destructor

فيه بقى معلومة زيادة الدكتور قالها في اخر المحاضرة .. لو عملت 3class objects زي كدة :

Complex x,y,z;

وعايز اعمل operations على ال class objects دي .. احنا عايزين نعمل ال operations العادية اللي نعرفها زي + & - & ~ وكدة :

x = y + z;
y = x * z;
y = ~z;
x = y + z – x;

احنا مانقدرش نعمل حاجة زي كدة من نفسنا لأن دي مش ارقام عادية بنجمعها .. بس نقدر نعمل حاجة اسمها operator overloading جوة ال class بتاعنا(هي نفس فكرة ال function overloading بس بشكل مختلف عشان دي operators مش functions) وساعتها نقدر نقول لل compiler لما يجيلك ال operator دة اعرف اني عايزك تعمل بيه حاجة معينة .. تعالى ناخد مثال

عايزين نعمل operator overloading لل +

هي دي اللي مكتوبة في ال .h فوق خالص في اول المحاضرة .. فهنكتبها كدة:

Complex operator+(Complex c);

وهنيجي في ال .cpp نكتب ال implementation بتاعها عادي وساعتها نقدر نعمل y +z عادي جداً