Chapter-1 Constructor and Destructor

(Week 1 and 2)

Dr.S.Manimurugan

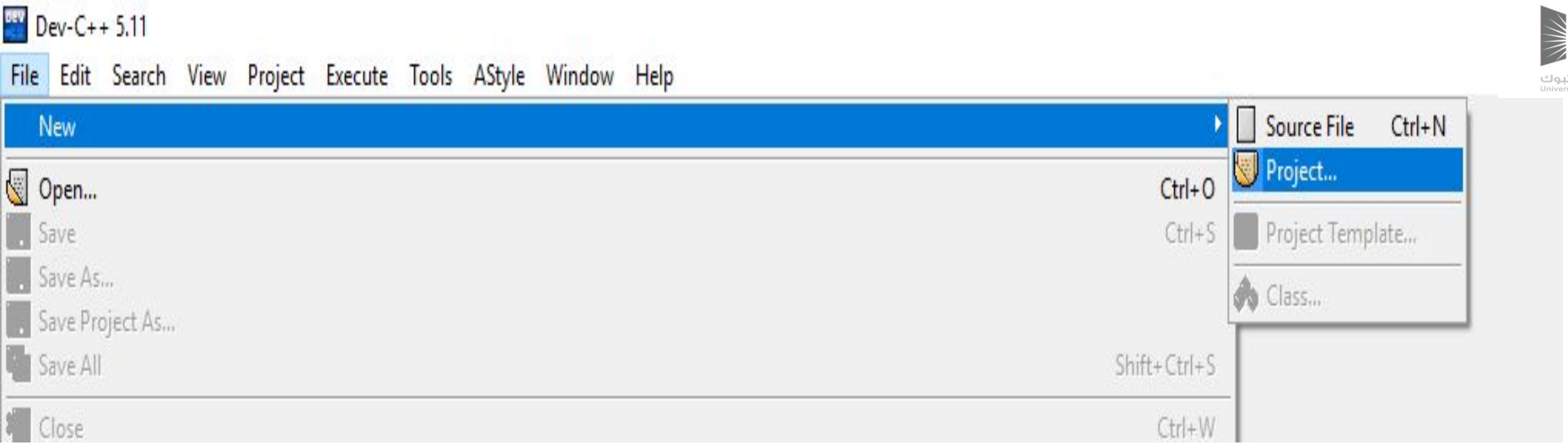
OUTLINE…

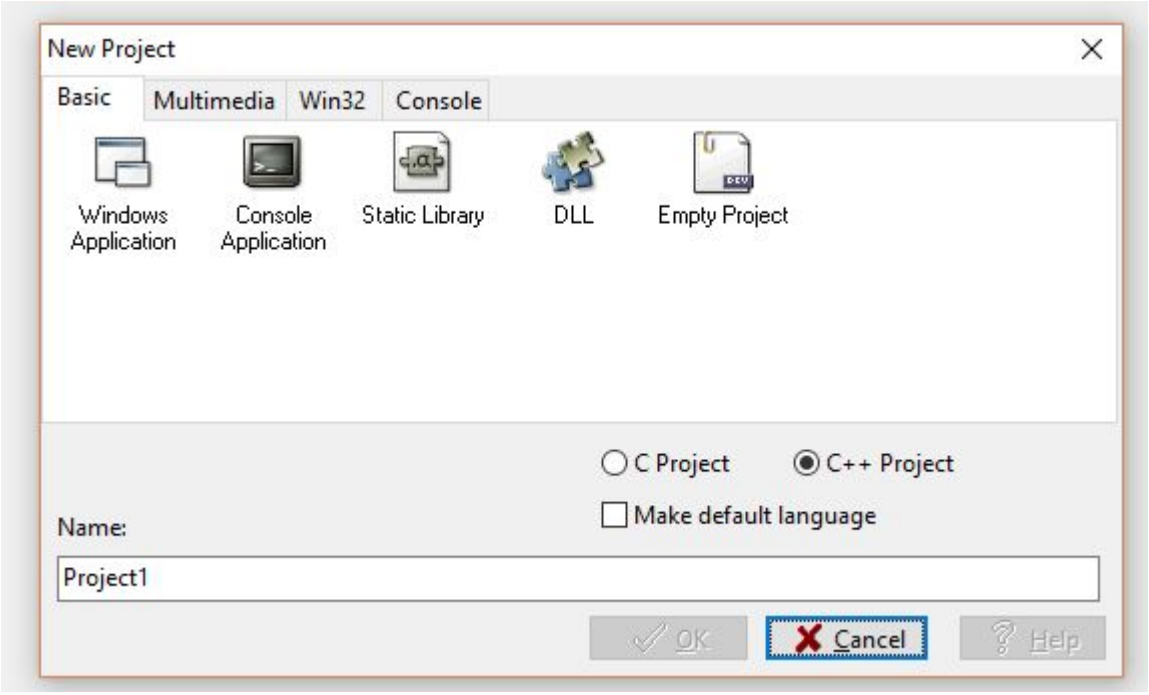
- PREPROCESSOR WRAPPERS
- CONSTRUCTORS AND DESTRUCTORS IN C++
- TYPES OF CONSTRUCTORS IN C++
  - Default Constructor
  - Parameterized Constructor
  - Copy Constructor
- DESTRUCTORS IN C++
- C++ CONSTRUCTORS OVERLOADING
- EXAMPLES

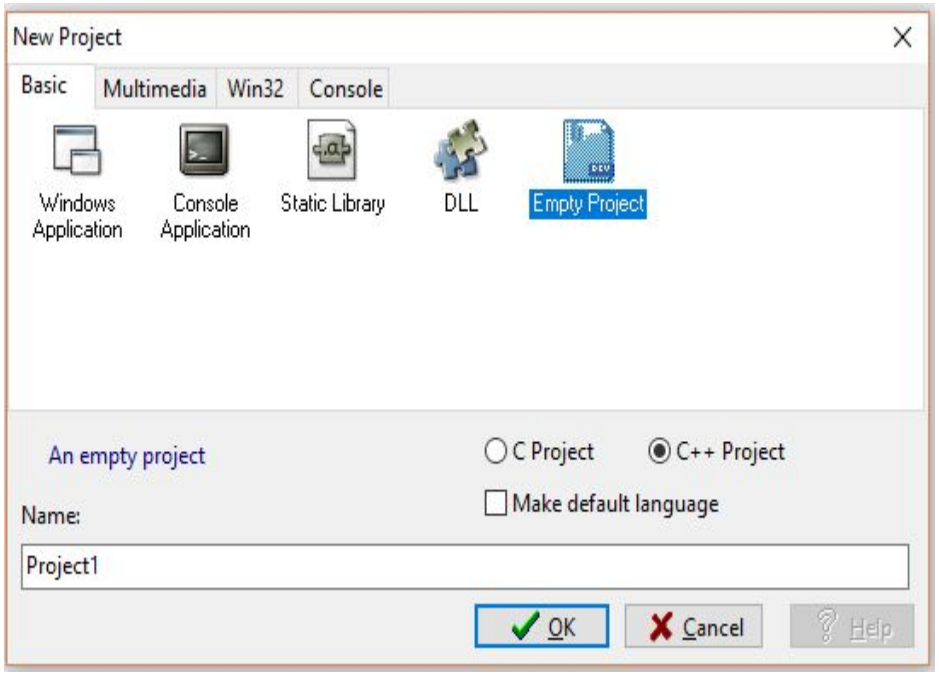
PREPROCESSOR WRAPPERS

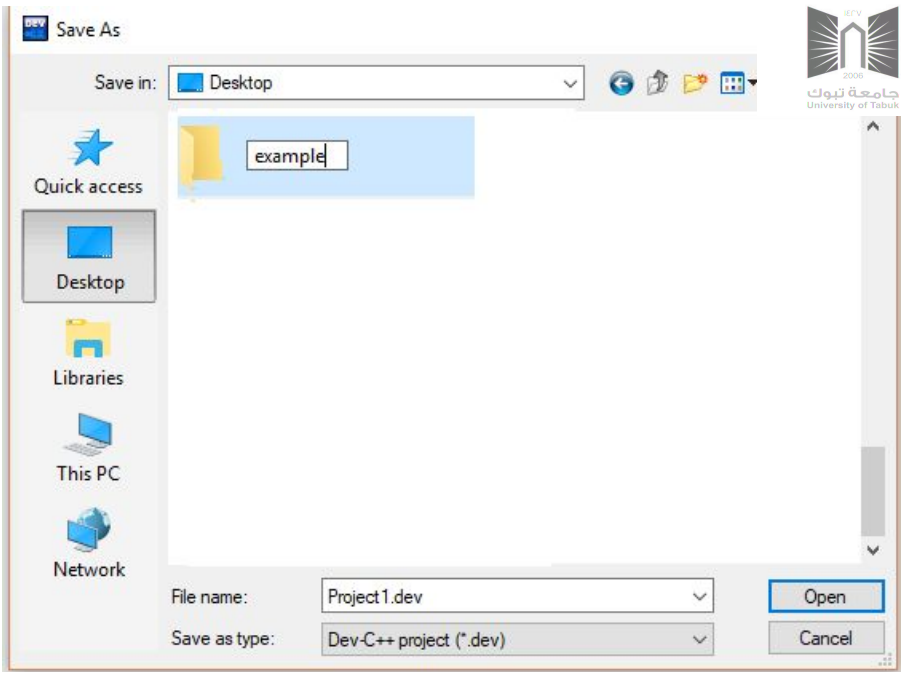
- Prevents code from being included more than once
  - **#ifndef** – “if not defined”
    - Skip this code if it has been included already
  - **#define**
    - Define a name so this code will not be included again
  - **#endif**
- If the header has been included previously
  - Name is defined already and the header file is not included again
- Prevents multiple-definition errors
- Example

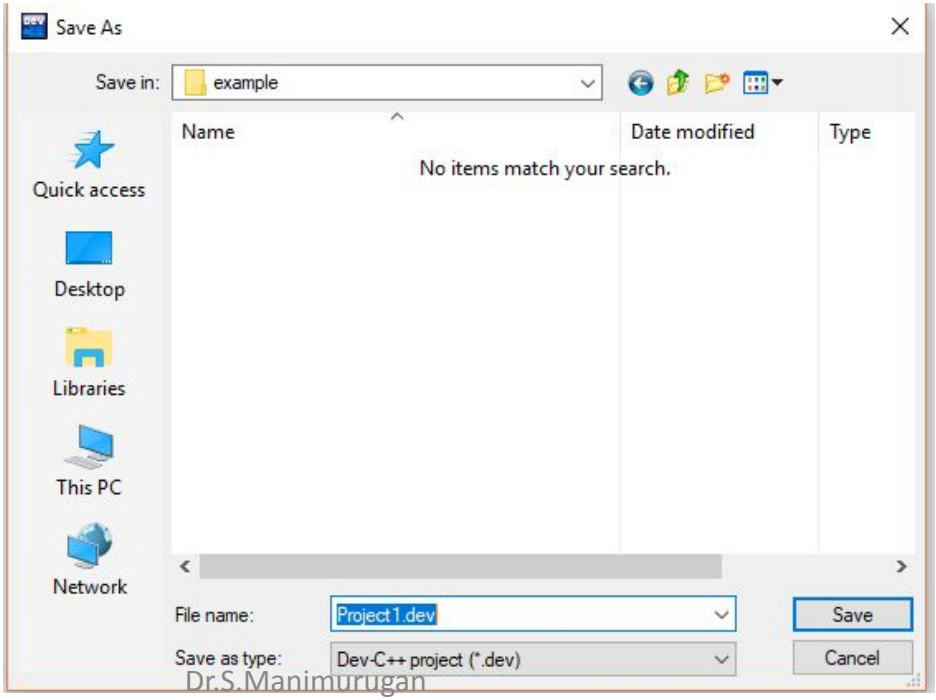
```
#ifndef TIME_H
#define TIME_H
… // code
#endif
```


Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
New
Open... Ctrl+O
Save Ctrl+S
Save As...
Save Project As...
Save All Shift+Ctrl+S
Close Ctrl+W
Source File Ctrl+N
Project...
Project Template...
Class...


New Project
Basic
Multimedia
Win32
Console
Windows Application
Console Application
Static Library
DLL
Empty Project
C Project
C++ Project
Make default language
Name:
Project1
OK
Cancel
Help



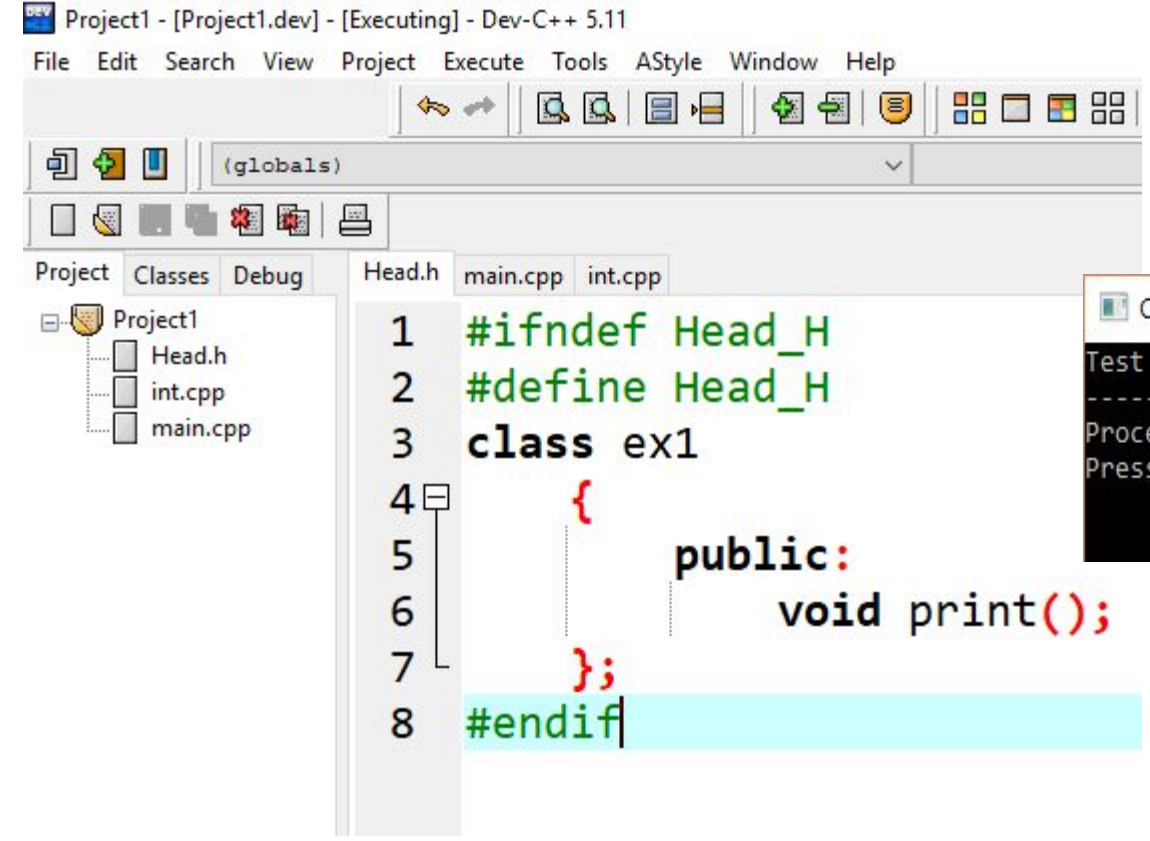


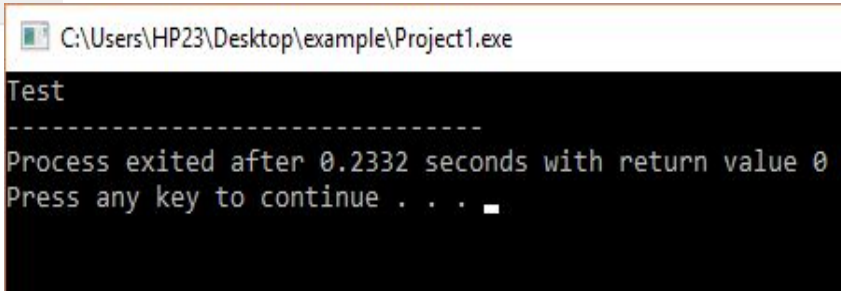


Simple Class and objects

```
1  #include<iostream>
2  using namespace std;
3  class ex1
4      {
5          public:
6              void print()
7                  {
8                      cout<<"Test";
9                  }
10     };
11 int main()
12     {
13         ex1 obj;
14         obj.print();
15     }
```

```
C:\Users\HP23\Desktop\example\Project1.exe
Test
--------------------------------
Process exited after 0.2332 seconds with return value 0
Press any key to continue . . .
```





Head.h main.cpp int.cpp

```
1 #include<iostream>
2 #include "Head.h"
3 using namespace std;
4 void ex1::print()
5 {
6 cout<<"Test";
7 }
8
```

Head.h main.cpp int.cpp

```
1 #include<iostream>
2 #include "Head.h"
3 using namespace std;
4 int main()
5 {
6 ex1 obj;
7 obj.print();
8 }
```

Constructors and Destructors in C++

- Constructors are special class functions which performs **initialization of every object**.
- The Compiler calls the Constructor whenever **an object** is created.
- Constructors initialize values to object members after storage is allocated to the object.
- Wh ther ha **destroy** the

```
class A
{
   int x;
   public:
   // constructor
   A()
   {
      // object initialization
   }
};
```

```
class A
{
   public:
   A()
   {
      // statement
   }
   // defining destructor for class
   ~A()
   {
      // statement
   }
};
```

- While defining a constructor you must remember the **name of constructor** will be same as the **name of the class**, and constructors will never have a return type.
- Constructors can be defined either inside the class definition or outside class definition using class name and scope resolution :: operator

```
class A
{
   int i;
   public:
   A(); // constructor declared
};

// constructor definition
A::A()
{
   i=1;
}
```

Types of Constructors in C++

Constructors are of three types:

- Default Constructor
- Parameterized Constructor
- Copy Constructor

Default Constructors

- Default constructor is the constructor which doesn't take any argument. It has no parameter.
- In this case, as soon as the object is created the constructor is called which initializes its **data members**.
- A default constructor is so important for initialization of object members, that even if **we do not define a constructor** explicitly, the compiler will provide a **default constructor implicitly**.

Syntax:

```
class_name(parameter1, parameter2, ...)
{
   // constructor Definition
}
```

For example:

```
class Cube
{
   int side;
   public:
   Cube()
          {
                    side=10;
          }
};

int main()
          {
                    Cube c;
                    cout << c.side;
          }
```

Parameterized Constructors

- These are the constructors with parameter.
- Using this Constructor you can provide different values to data members of different objects, by **passing the appropriate values as argument**.
- By using parameterized constructor in above case, we have initialized **3 objects with user defined values**. We can have **any number of**

For example:

```
class Cube
{
   public:
          int side;
          Cube(int x)
                    {          side=x;    }
};
int main()
{
   Cube c1(10);
   Cube c2(20);
   Cube c3(30);
   cout << c1.side;
   cout << c2.side;
   cout << c3.side;
}
```

```
10
20
30
```

COPY CONSTRUCTOR

- A copy constructor is a member function which initializes an object using another object of the same class.
- Copy Constructor is a type of constructor which is used to create a copy of an already existing object of a class type. It is usually of the form **X (X&)**, where X is the class name. The compiler provides a default Copy Constructor to all the classes.

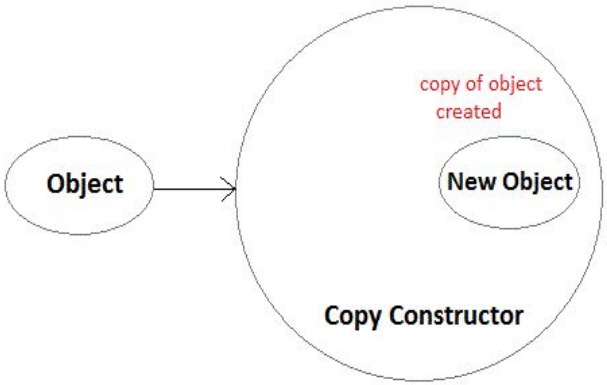
Syntax of Copy Constructor

```
Classname(const classname & objectname)
{
   ....
}
```

As it is used to create an object, hence it is called a constructor. And, it creates a new object, which is exact copy of the existing copy, hence it is called **copy constructor**.

```
#include<iostream>
using namespace std;
class SC
{
   private:
   int x, y;   //data members
   public:
         SC(int x1, int y1)
         {
            x = x1;
            y = y1;
         }
   /* Copy constructor */
   SC(const SC &obj2)
         {
            x = obj2.x;
            y = obj2.y;
         }
   void display()
         {
            cout<<x<<" "<<y<<endl;
         }
};
```

```
/* main function */
int main()
{
   SC obj1(10, 15);     // Normal constructor
   SC obj2 = obj1;      // Copy constructor
   cout<<"Normal constructor : ";
   obj1.display();
   cout<<"Copy constructor : ";
   obj2.display();
   return 0;
}
```



DESTRUCTORS IN C++

Destructor is a special member function that is executed automatically when an object is destroyed that has been created by the constructor. C++ destructors are used to de-allocate the memory that has been allocated for the object by the constructor.

Its syntax is same as constructor except the fact that it is preceded by the tilde sign.

```
~class_name() { }; //syntax of destructor
```

CONSTRUCTORS AND DESTRUCTORS IN C++

- Constructors are special class functions which performs **initialization of every object**.
- The Compiler calls the Constructor whenever **an object** is created.
- Constructors initialize values to object members after storage is allocated to the object.
- Wh ther ha **destroy** the

```
class A
{
   int x;
   public:
   // constructor
   A()
   {
      // object initialization
   }
};
```

```
class A
{
   public:
   A()
   {
      // statement
   }
   // defining destructor for class
   ~A()
   {
      // statement
   }
};
```

STRUCTURE OF C++ DESTRUCTORS

```
/*...syntax of destructor....*/
class class_name
    {
        public:
        class_name();    //constructor.
        ~class_name();       //destructor.
    }
```

Unlike constructor a destructor neither takes any arguments nor does it returns value. And destructor can’t be overloaded.

```
#include <iostream>
using namespace std;
class ABC
{
    public:
       ABC () //constructor defined
            {
                cout << "Hey look I am in constructor" << endl;
            }
      ~ABC() //destructor defined
            {
                 cout << "Hey look I am in destructor" << endl;
            }
};
int main()
{
    ABC cc1; //constructor is called
    cout << "function main is terminating...." << endl;
    return 0;
}  //end of program
```

```
Hey look I am in constructor
function main is terminating....
Hey look I am in destructor
```

C++ CONSTRUCTORS OVERLOADING

- Every constructor has same name as class name but they differ in terms of either number of arguments or the datatypes of the arguments or the both.
- As there is more than one constructor in class it is also called **multiple constructor**.

```
/*.....A program to highlight the concept of constructor
overloading..... */
#include <iostream>
using namespace std;
class ABC
{
    private:
       int x,y;
    public:
       ABC ()       //constructor 1 with no arguments
      {
          x = y = 0;
       }
       ABC(int a)    //constructor 2 with one argument
      {
           x = y = a;
       }
```

```
ABC(int a,int b)    //constructor 3 with two argument
        {
             x = a;
             y = b;
        }
  void display()
        {
             cout << "x = " << x << " and " << "y = " << y << endl;
        }
};
int main()
{
     ABC cc1; //constructor 1
     ABC cc2(10); //constructor 2
     ABC cc3(10,20); //constructor 3
     cc1.display();
     cc2.display();     cc3.display();      return 0;
 }  //end of program
```

```
x = 0 and y = 0
x = 10 and y = 10
x = 10 and y = 20
```

1. What is called a class constructor? What is the purpose of class constructor?

A class can contain special functions: constructors and **destructors**. A class **constructor** is a special method (function) of a class. The constructor is called when a class object is created. Typically, the constructor is used for:

- allocating memory for a class object;
- initial initialization of the internal data of the class.

The constructor is intended to form an instance of a class object. The name of the class constructor is the same as the class name.

2. At what point does the program call the class constructor?

The constructor is called when a class object is created. The class

3. Can the constructor have parameters? Examples of constructors with different number of parameters

The constructor can have any number of parameters. Also, the constructor can be without parameters (the default constructor).

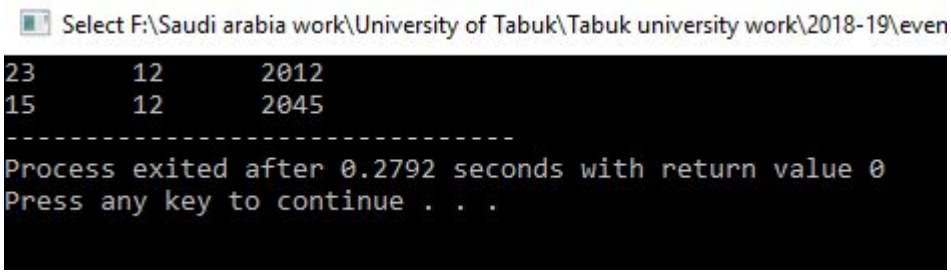
```
#include<iostream>
using namespace std;
class CMyDate
{
    int day;
    int month;
    int year;
    public:
    // class constructors
    CMyDate(); // constructor without parameters
    CMyDate(int d, int m, int y); // constructor with 3 parameters
```

```
 // class methods
    void SetDate(int d, int m, int y); // set a new date
    int GetDay(void); // returns day
    int GetMonth(void); // returns month
    int GetYear(void); // returns year
};
// implementation of class constructors and methods
// constructor without parameters (default constructor)
CMyDate::CMyDate()
{
    // set the date 01.01.2001
    day = 1;
    month = 1;
    year = 2001;
}
// constructor with 3 parameters
CMyDate::CMyDate(int d, int m, int y)
{
    day = d;
    month = m;
    year = y;
}
```

```
// set a new date
void CMyDate::SetDate(int d, int
m, int y)
{
    day = d;
    month = m;
    year = y;
}
// return day
int CMyDate::GetDay(void)
{
    return day;
}
// return month
int CMyDate::GetMonth(void)
{
    return month;
}
```

```
// retutn year
int CMyDate::GetYear(void)
{
    return year;
}

int main()
      {
      CMyDate     obj;
      CMyDate obj1(15,12,2045);
      obj.SetDate(23, 12,2012); // set a new date
            cout<<obj.GetDay()<<"\t"<<obj.GetMonth()<<"\t"<<obj.GetYear()
            cout<<obj1.GetDay()<<"\t"<<obj1.GetMonth()<<"\t"<<obj1.GetYe
      }
```



4. Is it necessary to declare a constructor in a class?

Not. When you create a class object that does not contain any constructors, the implicit default constructor will be called. This constructor allocates memory for the class object. However, in the class, you can declare your own default constructor. This constructor is called: an explicitly defined default constructor.

5. What is the default constructor? Examples

The default constructor is the constructor of a class that is declared without parameters. If the class does not explicitly contain a specific constructor, then when the object is created, the default constructor is automatically called. When declaring a class object, the class constructor simply allocates memory for it.

```
// A class that defines a point on the
coordinate plane
class CMyPoint
{
    int x;
    int y;

    public:
    // class methods
    void SetPoint(int nx, int ny)
    {
        x = nx;
        y = ny;
    }

    int GetX(void) { return x; }
    int GetY(void) { return y; }
};
```

```
CMyPoint MP; // the default constructor is automaticall
called

MP.SetXY(4, -10); // call of class methods
int t;
t = MP.GetY(); // t = -10
```

```
// A class that defines a point on the
coordinate plane
class CMyPoint
{
    int x;
    int y;

    public:
    // class methods
    void SetPoint(int nx, int ny)
    {
        x = nx;
        y = ny;
    }

    int GetX(void) { return x; }
    int GetY(void) { return y; }
};
```

```
CMyPoint MP; // the default constructor is automaticall
called

MP.SetXY(4, -10); // call of class methods
int t;
t = MP.GetY(); // t = -10
```