Attention

Online tutorial 1 is available now. It is due at 10 pm 04 Sept 2022. You have three attempts. The result will be based on your final attempts.

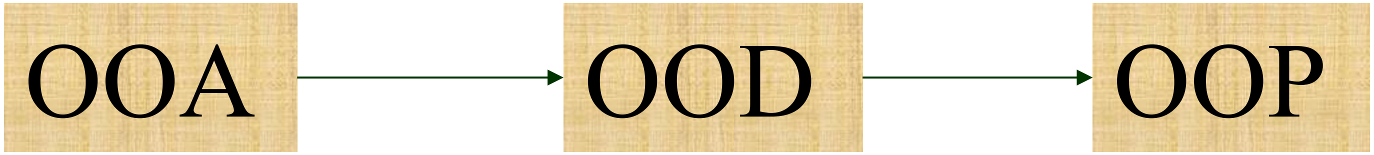
Comp2014 Object Oriented Programming

Lecture 5

Class Design & Implementation

Topics covered by last lecture

- Objects
- Data abstraction
- Classes and objects
- Class definition
- Member functions
- Applications



OOA → OOD → OOP

Topics covered by this lecture

- Class declaration, definition and application
- Constructor
- Destructor
- Object composition

Class creation and application

Three steps of OOP

```
class Date {
private:
   int day;
   int month;
   int year;
public:
   void setDate();
   void showdate();
};
```

class declaration

date.h

```
Void Date::setDate(
   int d, int m, int y){
      day = d;
      month = m;
      year = y;
}
void Date::showdate() {
      cout<< day <<" ”
            << month <<" ”
            << year << endl;
}
```

class definition

date.cpp

```
int main() {
   Date d;
   d.setDate(27,8,2021);
   d.showdate();
   return 0;
}
```

class driver

dateApp.cpp

You may put all of them in one file, two or three files but you must have a .cpp file that contains the *main* function.

Difference between
C++ and Java

Define and use data items/member functions

Data items

- Declaration:
  - Declare data types for each item
- Definition:
  - No need
- Uses:
  - For *internal* use, directly call their names
  - For *external* use, use . or public interface, *e.g*, objectVar.dataItem

Member functions

- Declaration:
  - Declare the prototype of each member function
- Definition:
  - Implement each member function
- Uses:
  - For *internal* use, directly call their names
  - For *external* use, use . or public interface, *e.g*, objectVar.memberFunction(…)

In-class Function Definitions

In C++, a member function can also be defined within the class declaration, named inline member function, as normally done in Java.

```
class Date {
    public:
        int getDay() { return day; }
  };
```

Alternatively

```
class Date {
    public:
        int day();
  };

int Date::getDay() { return day; }
```

Date4.h

dateApp4.cpp

Class declaration

- You cannot declare the same class more than one time.
- To avoid declaring a class more than once, using

```
#ifndef DATE4
#define DATE4
    //DATE4 is the id of the file
    //Class declared here
#endif //DATE4
```

Your IDE can help you to generate the id of the file if you create a class by using new Header file

Topics covered by the lecture

- Class declaration, definition and application
- Constructor
- Destructor
- Object composition

Constructor

A constructor is a specific member function of each class **that is called whenever an object of the class is created**. A constructor is similar to any other member function, with three exceptions:

1) *Constructor must have the same name as the class.*

```
    Date();//declare a constructor of Date
```

2) *Constructor has no return value.*

```
    ~~void~~Date();
```

3) *Constructor is automatically called when object is declared:*

```
   Date myBirthday;//declare an object of Date
    ~~myBirthday.Date()~~;//incorrect call
```

Anything that can be done in a normal member function can be done in a constructor. However, constructors are mostly used for initialisation because *in many situations, initialisation cannot take place elsewhere in the class.*

Initialisation using constructors

```
class Date {
private:
    int day;
    int month;
    int year;
public:
    Date();
    void setdate(int,int,int);
    void showdate();
};
```

constructor

```
Date::Date() {
  day = 0;
  month = 0;
  year = 0;
}
```

Initialisation using constructors

```
class Date {
private:
    int day;
    int month;
    int year;
public:
    Date() {
        day = 0;
        month = 0;
        year = 0;
    }
    void setdate(int,int,int);
    void showdate();
};
```

```
int main() {
    Date d;
    d.showdate();
}
```

Date4.h

dateApp4.cpp

Initialisation using constructors

```
class TicTacToe {
private:
  char board[3][3];
  int noOfMoves;
public:
  TicTacToe() { //Default constructor.
    for (int row = 0; row < 3; row++)
      for (int col = 0; col < 3; col++) {
        board[row][col] = ' ';
    }
    noOfMoves = 0;
  }
  //More code
};
```

Constructor with parameters

```
class Date {
private:
    int day;
    int month;
    int year;

Public:
    Date(int, int, int);
    void showdate();
};
```

```
Date::Date(int d, int m, int y) {
  day = d;
  month = m;
  year = y;
}
```

Like normal functions, constructors can take parameters.

Date4_1.h

dateApp4_1.cpp

Multiple Constructors

```
class Date {
    private:
        int day, month, year;
    public:
        Date ();                 // take no argument
        Date (int );             // take one argument
        Date ( int, int );       // take two arguments
        Date ( int, int, int );  // take three arguments
        void showdate();
};
```

Default constructor

Function overloading allows us to have more than one constructors.

A default constructor is the constructor which can be called with no arguments.

Call constructors

```
class Date {
  private:
    int day, month, year;
  public:
    Date ();
    Date (int );
    Date (int, int );
    Date (int, int, int );
   void showdate();
};
```

```
int main() {
   Date d1;
   Date d2(17);
   Date d3(17,8);
   Date d4(17, 8, 2020);
   Date d5 = Date(17,8,2020);
   d1.showdate();
   d2.showdate();
   d3.showdate();
   d4.showdate();
}
```

The same

Date4_2.h

dateApp4_2.cpp

Default constructors

If the user does not define any constructor, the compiler will generate a default constructor with empty body. However, if the user define a constructor (with or without arguments), there is no automatically generated constructor.

```
class A {
        private: int val;
        public: A() { val=0; }
};

class B {
        private: int val;
        public: B(int i) { val=i; }
};

class C {
        private: int val;
};
```

Which of definitions of objects gives compiling error?

```
int main() {
        A a;
        B b;
        C c;
        return 0;
}
```

constructor.cpp

Initializer list

Date4_3.h

dateApp4_3.cpp

There are four ways to initialise data members:

- Pre-set the values:

```
Date() { day=16; month=8; year=2021; }
```

- Take from the user (provide the values when create an object of the class):

```
Date(int d, int m, int y)
      { day=d; month=m; year=y; }
```

Date d(16,8,2021);

- Initializer list (very useful for inheritance):

```
Date(): day(26), month(8), year(2021){ }
Date(int d, int m, int y): day(d), month(m), year(y){}
```

- Copy constructor: using an existing object to initialise the current object

InitializerList.cpp

Initialisation

```
class Date {
private:
     int day = 0; // C++11
     int month = 0; // C++11
     int year = 0; // C++11
public:
      void setdate(int,int,int);
      void showdate();
};
```

Some IDEs do not support C++11 or require specific setting for using C++11

When you define a class, the compiler does not allocate memories to the class (except for static data members). Memories are allocated to objects of a class. Therefore, when an object is created, the object will gain memory for each data member and the "*constructors*" can then initialise these memories as specified in the constructors.

Constructor using default values

```
#include<iostream.h>
class Time {
private:
    int hrs , mins, secs;
public:
    Time(int = 0, int = 0, int = 0);
    void display() {cout << hrs << ":"
          << mins << ":" << secs << endl;}
};
Time::Time(int h, int m, int s) {
    hrs = h;
    mins = m;
    secs = s;
}
```

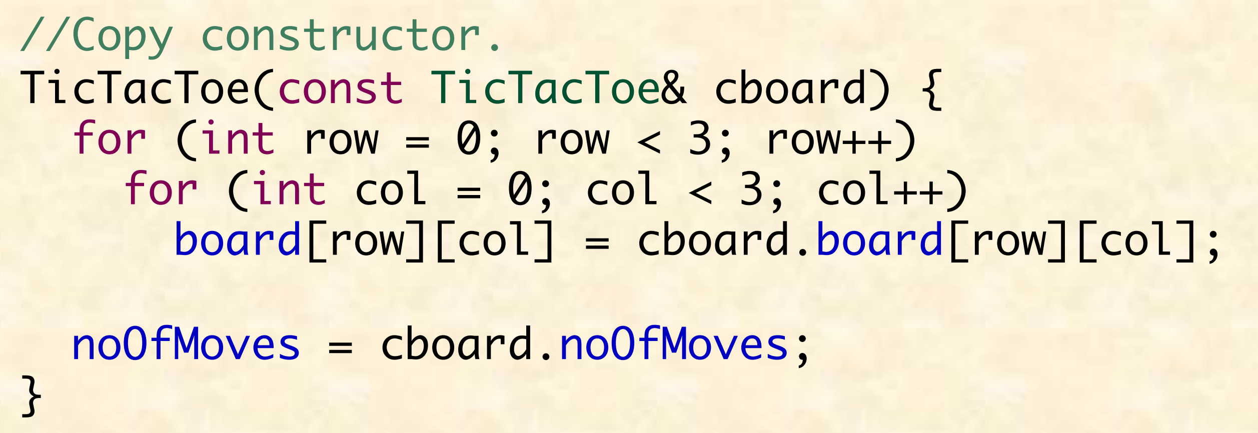
```
void main()
{
          Time t;
          Time t1(1);
          Time t2(2,20);
          Time t3(3,30,30);

         t.display();
         t1.display();
         t2.display();
         t3.display();
}
```

defaultArg.cpp

Copy constructor: a preview

A copy constructor of a class is a special constructor for creating a new object as a copy of an existing object. The copy constructor is called whenever an object is initialized from another object of the same class. Typical declaration of a copy constructor:

```
      ClassName(const ClassName&);
```



```
//Copy constructor.
TicTacToe(const TicTacToe& cboard) {
  for (int row = 0; row < 3; row++)
    for (int col = 0; col < 3; col++)
      board[row][col] = cboard.board[row][col];

  noOfMoves = cboard.noOfMoves;
}
```

Ways of using a copy constructor:

```
   TicTacToe board;
   //More code on board
   TicTacToe tempBoard(board); //tempBoard is a new object
   TicTacToe* tempBoard = new TicTacToe(board);
```

Topics covered by the lecture

- Class declaration, definition and application
- Constructor
- Destructor
- Object composition

Destructors

- A destructor is called **when an object is destroyed**. It is a function with the same name of the class only with a ~ (Tilda) in the beginning without parameters.
- One class has only one destructor.
- The destructor of a class is called automatically when an object of the class goes out of scope. It is typically used for clean-up and resource release (*no automated garbage collection in C++*).

Difference between C++ and Java

Destructors

```
class A {
  private:
      int num ;
  public:
      int getNumber() { return num; }
      A(int i=0) { num=i; cout << i << “ctor” << endl;} //constructor
      ~A() { cout << num << “dtor” << endl; } //destructor
};

int main() {
      A x(1) ;
      {
            { A y(2) ; }
             A z(3) ;
      }
}
```

Automatic objects

destructor.cpp

What is the output of the program?

Topics covered by the lecture

- Class declaration, definition and application
- Constructor
- Destructor
- Object composition

Composition

- Complex objects are often built from smaller, simpler objects.
- The process of building complex objects from simpler ones is called object composition.
- Object composition models a "*has-a*" relationship between two objects.

```
class Room {
  Desk console;
  Chair chairs[50];
  Doors doors[2];
};
```

Any object of room contains contains space to store one object of console, 50 objects of chairs and 2 objects of doors.

Class communication in composition

- Objects in a composited class can communicate each other via their interface (public functions).

```
class Board {
   char grid[BOARDSIZE][BOARDSIZE];
public:
  bool addMove(int x1,int y1, int x2,int y2);
  bool checkWin();
  bool validInput(int x, int y);
  void printBoard();
};
```

```
class Player {
protected:
  int playerType;
public:
  void getMove(Board b,int& x,int& y);
  int getType();
};
```

```
class Game {
  Board board;
  Player player[2];
public:
  void play();
};
```

```
void Game::play() {
   while(!board.checkWin()) {
      int x1, y1, x2, y2;
      player[0].getMove(board, x1, y1);
      player[1].getMove(board, x2, y2);
      board.addMove(x1,y1,x2,y2);
   }
}
```

A game consists of a game board and two players.

Homework

- Read textbook Chapter 6 & 7.
- Complete online tutorial 1 if you haven’t.
- Show your tutor the solution of practical 4 if you haven’t
- Start to work on assignment 1 if you haven’t. Feel free to ask us any questions related to the assignment.