DigiPen
INSTITUTE OF TECHNOLOGY

CS170#04.1

Classes And Objects

Vadim Surov

Outline

- Procedural & Object-Oriented Programming
- Classes & Objects
- Access Specifiers
- Constructor
- Destructor
- Pointer To Object And Self Reference
- Const Member Functions
- Accessors & Mutators
- Mutable Data Member
- Static Members
- Friends

Procedural & Object-Oriented programming

- Procedural programming
  - break down a programming task into a collection of variables, data structures, and functions
  - data and functions are separated

- Object-oriented programming
  - break down a programming task into objects with each object encapsulating its own data and methods (functions)
  - methods are more focused on object's data

Classes & Objects

- A class is an **expanded concept of a data structure** declared using the keyword **`class`**
- It can hold both data methods as members with access specifiers
- An object is an **instance** of the class

```
class Shape {
  private:
    int area;
  public:
    int get_area();
} triangle;
```

Access Specifiers

- By **default**, all members of a class have **private** access for all its members
- **Access specifiers** modify the access rights that the members following them acquire:
  - **private** members of a class are accessible only from within other members of the same class or from their friends
  - **protected** members are accessible from members of their same class and from their friends, but also from members of their derived classes
  - **public** members are accessible from anywhere where the object is visible

Constructor

- Object constructor is like a member function but
  - must have the same name as the class
  - cannot have any return type; not even void
  - must be public (but can be private)
  - cannot be called explicitly
- Automatically called whenever a new object of the class is created
- Use to initialize data members or assign dynamic memory
- **Default constructor** (without arguments) is created automatically (synthesized) if and only if there is **no any** overloaded constructor

Overloaded Constructor

- Constructor can be overloaded as any other function that have the same name but **different types or number of parameters**

- As soon as you declare **your own** constructor for a class, the compiler **no** longer **provides** an implicit **default constructor**

Example With Overloaded Constructor

```
class CRectangle {
  int x, y;
public:
  // Overloaded constructor
  CRectangle(int _x, int _y) {
    x = _x;
    y = _y;
  }
};

void main() {
  CRectangle rect(3, 6); //Using overloaded ctor
  CRectangle rect2; //Error: there is no default ctor
}
```

Copy Constructor

- Copy constructor makes copy all the data contained in another object to the data members of the current object
- Ex: `rect2` is created with copy constructor:

```
CRectangle rect(3, 6);
CRectangle rect2(rect);
```

- Created automatically (synthesized) if not specified
- Synthesized copy constructor makes "shallow" copy
- You can overload the copy constructor:

```
CRectangle(const CRectangle& rect) {
    x = rect.x;
    y = rect.y;
}
```

Copy Assignment Operator

- As well as copy constructor makes "shallow" copy, all the data contained in another object assigned to the data members of the current object using assignment operator
- Created automatically (synthesized) if not specified

```
CRectangle& operator=(const CRectangle&
rect)
{
  x = rect.x;
  y = rect.y;
};
CRectangle rect(3, 6);
CRectangle rect2 = rect;
```

Destructor

- Has opposite functionality to constructor
- Created automatically (synthesized) if not specified
- It is automatically called when an object is destroyed, either
  - because its scope of existence has finished
    - for example, if it was defined as a local object within a function and the function ends
  - because it is an object dynamically assigned and it is released using the operator **delete**

Destructor (contd)

- The destructor must have the same name as the class, but preceded with a tilde sign (~)
- It must also return no value
- The use of destructors is especially suitable when an object assigns dynamic memory during its lifetime and at the moment of being destroyed we want to release the memory that the object was allocated

Example With Destructor

```
class CRectangle {
  int x, y;
  char* tag;
public:
  CRectangle() {
    x = 0;
    y = 0;
    tag = (char*)malloc(10); // allocate memory
  }
  ~CRectangle() {
     free(tag); // free memory
  }
};
void main() {
  CRectangle rect; // Constructor here
} // Destructor here is called automatically
```

Pointer To Object And Self Reference

- Class name can be used as the type for a pointer to an object:
- Use `->` to get access to public member of the object via pointer

- Use **this** to points to the object whose member function is being executed (self reference):

```
CRectangle *rect;
rect->set_values(5, 6);
```

```
class CRectangle {
 int x, y;
public:
 // Overloaded constructor
 CRectangle(int x, int
y) {
    this->x = x;
    this->y = y;
 }
};
```

this-> here removes umbiguity