

# Object Oriented Programming

## Basic object-oriented features of C++

July 3, 2009 Dr. Mordo Shalom

- ## Classes
- A **class** in C++ is a type like struct, enum, etc...
  - A variable of this type, is called
    - an **instance** of this class, or
    - an **object** of this class
  - It is a collection of related data and behavior (functions and more...).
    - In this respect it is just like the ADT modules we already saw.
  - They have:
    - a **public interface**, and
    - a **private implementation**

- ## The interface
- It is written in an .h file.
  - By convention the name of the file is the name of the class.
  - The functions having a (typically first) parameter of the type of the ADT constitute the functions of the class.
  - struct+functions → class with **member functions**
  - Member functions also called **methods**

## The interface

```

#ifndef ELEVATOR_H
#define ELEVATOR_H
typedef enum {
    ELEVATOR_STOPPED,
    ELEVATOR_DOOR_OPEN,
    ELEVATOR_MOVING
} ElevatorState;
void elevatorInitModule(int lowestFloor, int highestFloor, int secondsToAccelerate, int secondsToSlowDown, int secondsBetweenFloors);
Elevator *elevatorCreate();
void elevatorOpenDoor(Elevator *);
void elevatorCloseDoor(Elevator *);
void elevatorMoveTo(Elevator *, int);
void elevatorTimeElapsed(Elevator *, int);
ElevatorState elevatorGetState(Elevator *);
int elevatorGetDirection(Elevator *);
int elevatorGetCurrentFloor(Elevator *e);
char *elevatorGetStatusString(Elevator *);
#endif //ELEVATOR_H_

class Elevator {
public:
    void openDoor();
    void closeDoor();
    void moveTo(int);
    void timeElapsed(int);
    ElevatorState getState();
    int getDirection();
    int getCurrentFloor();
    char *getStatusString();
};
    
```

Member functions

- ## Member Functions – the this pointer
- All the member functions receive an **implicit** first parameter called **this**:
  - this** is a pointer to the class under consideration.
  - We will get back to it soon.

- ## Constructor
- Special member functions having the name of the class.
  - It has an implicit return type which is the class under consideration
  - It is called automatically each time an object is instantiated immediately after memory allocation:
    - Static – in the static area
    - Automatic – in the stack
    - Dynamic – in the heap (using new keyword)
  - It receives implicitly the this pointer which points to the allocated memory.

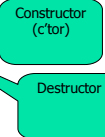
## The destructor

- A special member function having the name of the class prepended a ~
- It has no return type.
- It is called automatically each time an object is destroyed, immediately before memory de-allocation:
  - Automatic – in the stack – function returns,
  - Dynamic – in the heap (using delete keyword)
- It receives implicitly the this pointer.

## c'tor

```
#ifndef ELEVATOR_H_
#define ELEVATOR_H_
typedef enum {
    ELEVATOR_STOPPED,
    ELEVATOR_DOOR_OPEN,
    ELEVATOR_MOVING
} ElevatorState;
typedef struct elevator Elevator;
void elevatorInitModule (int lowestFloor, int highestFloor,
                        int secondsToAccelerate,
                        int secondsToSlowDown,
                        int secondsBetweenFloors);
Elevator *elevatorCreate();
void elevatorOpenDoor (Elevator *);
void elevatorCloseDoor (Elevator *);
void elevatorMoveTo (Elevator *, int);
void elevatorTimeElapsed (Elevator *, int);
ElevatorState elevatorGetState (Elevator *);
int elevatorGetDirection (Elevator *);
int elevatorGetCurrentFloor (Elevator *e);
char *elevatorGetStatusString (Elevator *);
#endif //ELEVATOR_H_

#ifdef ELEVATOR_H_
void elevatorInitModule (int lowestFloor, int highestFloor,
                        int secondsToAccelerate,
                        int secondsToSlowDown,
                        int secondsBetweenFloors);
class Elevator {
public:
    Elevator();
    ~Elevator();
    void openDoor();
    void closeDoor();
    void moveTo(int);
    void timeElapsed(int);
    ElevatorState getState();
    int getDirection();
    int getCurrentFloor();
    char *getStatusString();
};
#endif //ELEVATOR_H_
```



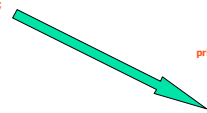
## Data Members – Encapsulation

- They are written in the **private** section.
- This means that they are visible only within the class code.
- A “client programmer” can not use (read/modify) them, but...
- He/she can see them by inspecting the .h file – (a slight deviation from the ADT concept)

## Data Members

```
elevator.cpp
...
struct elevator {
    ElevatorState state;
    int lastFloorStopped;
    int secondsToDestination;
    int secondsInFullSpeed;
    int movingTo;
};
...

elevator.h
class Elevator {
public:
    Elevator();
    void closeDoor();
    void openDoor();
    void moveTo(int);
    void timeElapsed(int);
    ElevatorState getState();
    int getDirection();
    int getCurrentFloor();
    char *getStatusString();
private:
    ElevatorState state;
    int lastFloorStopped;
    int secondsToDestination;
    int secondsInFullSpeed;
    int movingTo;
};
```



## public Data Members

- It is also possible to declare public data members. In this case:
  - A “client programmer” can use (read/modify) them. However
  - This is against the Encapsulation principle of OOP.
  - **DON'T use this feature.**

## Access to members

- Public member data and functions are accessed by client programmers using the normal struct fields' access syntax, i.e.
  - s.field // the usual way
  - p->field // the usual way
  - s.function(params) // function(&s,params)
  - p->function(params) // function(p,params)

## The OO Design process

- We have been describing the transition from structured design to Object Oriented design.
- This was only for the sake of learning and understanding the analogy between them.
- This is **not** a description of the design process.

## The OO Design process (cont'd)

- The correct design process:
  - Identify objects (classes) in your application.
    - Each class should implement a well defined and unique functionality, something that can be described in a short sentence. (Coherence)
  - Design the interfaces of the classes (Decoupling)
  - Design the implementation of the classes. Use functional decomposition (structured design) in this stage.

## private Member Functions

- They are written in the private section.
- This means that they are visible only within the class code.
- A "client programmer" can not use (invoke) them, but...
- He/she can see them by inspecting the .h file – (a slight deviation from the ADT concept)

## The this pointer – cont'd

- The member functions can use it among others to:
  - Pass a pointer to the object to non-member functions.
  - Resolve name conflicts, e.g. when a parameter has the same name as a data member.
  - Return it as a return value.

## Classes – the implementation

- It is written in a .cpp file.
- By convention the name of the file is the name of the class.
- It contains the code of the member functions

## Member Functions

```
.....
int elevatorGetDirection(Elevator *e) {
    switch (e->state) {
    case ELEVATOR_STOPPED:
    case ELEVATOR_DOOR_OPEN:
        return 0;
    case ELEVATOR_MOVING:
        return e->lastFloorStopped < e->movingTo
        ? 1 : -1;
    }
    UNREACHABLE;
    exit(1);
}
.....

int Elevator::getDirection() {
    switch (this->state) {
    case ELEVATOR_STOPPED:
    case ELEVATOR_DOOR_OPEN:
        return 0;
    case ELEVATOR_MOVING:
        return this->lastFloorStopped <
        this->movingTo ? 1 : -1;
    }
    UNREACHABLE;
    exit(1);
}
```

Sample code in project

ElevatorCPP-2-Classes

### Member Functions

```

int Elevator::getDirection() {
    switch (this->state) {
    case ELEVATOR_STOPPED:
    case ELEVATOR_DOOR_OPEN:
        return 0;
    case ELEVATOR_MOVING:
        return this->lastFloorStopped <
        this->movingTo ? 1 : -1;
    }
    UNREACHABLE;
    exit(1);
}

```

```

int Elevator::getDirection() {
    switch (state) {
    case ELEVATOR_STOPPED:
    case ELEVATOR_DOOR_OPEN:
        return 0;
    case ELEVATOR_MOVING:
        return lastFloorStopped <
        movingTo ? 1 : -1;
    }
    UNREACHABLE;
    exit(1);
}

```

### Static members

- Some functions do not operate on a specific object.
- Some data is not related to a specific object, but the entire class.
- In this case we prepend the keyword **static** to their declaration.
- Member functions:
  - do not receive a this pointer.
  - can not access non-static members (data and function)

### Static Member Functions

```

#ifndef ELEVATOR_H
#define ELEVATOR_H
typedef enum {
    ELEVATOR_STOPPED,
    ELEVATOR_DOOR_OPEN,
    ELEVATOR_MOVING
} ElevatorState;
typedef struct elevator Elevator;
void elevatorInitModule(int lowestFloor, int highestFloor,
    int secondsToAccelerate,
    int secondsToSlowDown,
    int secondsBetweenFloors);
Elevator *elevatorCreate();
void elevatorOpenDoor(Elevator *);
void elevatorCloseDoor(Elevator *);
void elevatorMoveTo(Elevator *, int);
void elevatorTimeElapsed(Elevator *, int);
ElevatorState elevatorGetState(Elevator *);
int elevatorGetDirection(Elevator *);
int elevatorGetCurrentFloor(Elevator *e);
char *elevatorGetStatusString(Elevator *);
#endif //ELEVATOR_H_

class Elevator {
public:
    static void initModule(int lowestFloor,
        int highestFloor,
        int secondsToAccelerate,
        int secondsToSlowDown,
        int secondsBetweenFloors);
    Elevator();
    ~Elevator();
    void openDoor();
    void closeDoor();
    void moveTo(int);
    void timeElapsed(int);
    ElevatorState getState();
    int getDirection();
    int getCurrentFloor();
    char *getStatusString();
};

```

### Static Data Members

```

elevator.h
class Elevator {
public:
    Elevator();
    ~Elevator();
    void openDoor();
    void closeDoor();
    void moveTo(int);
    void timeElapsed(int);
    ElevatorState getState();
    int getDirection();
    int getCurrentFloor();
    char *getStatusString();
private:
    static int lowestFloor;
    static int highestFloor;
    static int secondsToAccelerate;
    static int secondsToSlowDown;
    static int secondsBetweenFloors;
    ElevatorState state;
    int lastFloorStopped;
    int secondsToDestination;
    int secondsInFullSpeed;
};

elevator.cpp
static struct {
    int lowestFloor;
    int highestFloor;
    int secondsToAccelerate;
    int secondsToSlowDown;
    int secondsBetweenFloors;
} globals;

```

Don't be confused !!  
static has another meaning here

### Static Data Members

```

elevator.cpp
int Elevator::lowestFloor = 0;
int Elevator::highestFloor = 0;
int Elevator::secondsToAccelerate = 1;
int Elevator::secondsToSlowDown = 1;
int Elevator::secondsBetweenFloors = 1;

```

Sample code in project

ElevatorCPP-3-StaticMembers

## Mutators vs. Accessors

- A non-static member function is called
  - A **mutator**, if it modifies the contents of the class
  - An **accessor**, otherwise.
- An accessor is tagged as **const**.

```
void openDoor();
void closeDoor();
void moveTo(int);
void timeElapsed(int);
ElevatorState getState() const;
int getDirection() const;
int getCurrentFloor() const;
string getStatusString() const;
```

## sizeof (class)

- A class contains data and functions.
- Consider:

```
class A {
private:
    int x;
public:
    int f(int);
}
```
- What is sizeof(A) ?

## A word about containment

- Consider:

```
class B {
private:
    int f(void);
...
}
class A {
private:
    int i;
    B *b;
}
```
- Can a method of the class A invoke b->f() ?

## A word on global functions

- Any C program is a legitimate C++ program. Consequently:
- A C++ program can contain functions which do not make part of any class.
- These are called **global functions**.
- They constitute a deviation from the OO design principles, **use them sparingly**.
- e.g. the functions in simulator.c
- OO design asks for a definition of a simulator class.

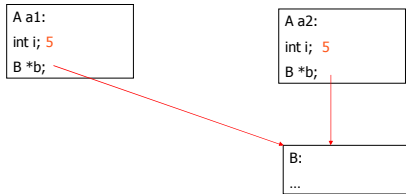
## Assignments involving objects

- Consider:

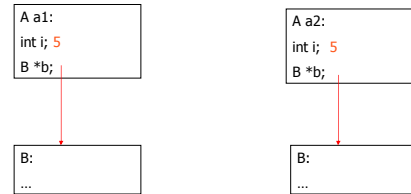
```
class B {
...
}
class A {
private:
    int i;
    B *b;
}
A a1, a2;
...
a2 = a1; // *

```
- What happens in (\*) ?

## Answer: a shallow copy



## As opposed to a deep copy



We will learn soon how to change this default behavior.

## Const data members

- First way:
  - static const double PI = 3.1415936
  - static const int STOPPED = 1;
- Second way:
  - private:
    - enum {STOPPED = 1}
    - // only for integral values

## The (built-in) Class String

```
class String
{
public:
    void Assign(const char *p);
    int Length() const;
    void Print() const;
private:
    char *s;
};
```

## The Class String – cont'd

```
void String::Assign(const char *p)
{
    s=new char[ strlen(p)+1 ];
    strcpy(s,p);
}

int String::Length() const
{
    return strlen(s);
}

void String::Print() const
{
    std::cout << s << std::endl;
}
```

Q: What is missing ?

A: De-allocation of previously occupied space (if any)

## The string Class – cont'd

client code

```
main()
{
    String st1, st2;
    char my_string[20];
    cout<<"Enter a word of length less than 20: ";
    cin>>my_string;
    st1.Assign("my first string");
    st2.Assign(my_string);
    st1.Print();
    st2.Print();
    return 0;
}
```

Objects allocated on stack

## The string Class – cont'd

```
class String
{
public:
    void Assign(const String&);
    void Assign(const char *p);
    int Length() const;
    void Print() const;
private:
    char *s;
};
```

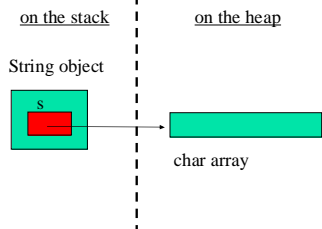
Passing const by ref

## The string Class – cont'd

client code:

```
int main()
{
    String st1, st2;
    char temp_string[20];
    cout<<"Enter a word of length less than 20: ";
    cin>>temp_string;
    st1.Assign(temp_string); // 2nd Assign
    st2.Assign(st1); // 1st Assign
    st1.Print();
    st2.Print();
    return 0;
}
```

## The string Class – cont'd



Sample code in project  
ElevatorCPP-4-Strings

(pay attention to the usage of  
ostringstream instead of  
sprintf)