Object-Oriented Programming and Data Structures

COMP2012: Introduction

Prof. Brian Mak
Prof. C. K. Tang

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology
Hong Kong SAR, China

Why Take This Course?

You have taken COMP1021/1022P/1022Q and COMP2011. So you can already program, right?

- Think about this: You learned English for many years, but can you write a novel?
- You basically have learned the C part of C++ in COMP2011 with a brief introduction to C++ classes, and you can write small C++ programs.
- But what if you are to write a large program, probably with a team of programmers?

In this course, you will learn the essence of OOP with a few new constructs in C++ with an aim to write large softwares.

Spaghetti Code vs. Structured Programming

- Programming code with a complex and tangled control structure, especially one using
  - many GOTOs
  - other 'unstructured' branching constructs
- Hard to understand ⇒ structured programming.

Example: Spaghetti Code

```
10 k = 1
20 gosub 100
30 if y > 120 goto 60
40 k = k + 1
50 goto 20
60 print k, y
70 stop
100 y = 3*k*k + 7*k - 3
110 return
```

- Example Languages: FORTRAN, BASIC
- Loop constructs: goto, gosub

Procedural Programming (PP)

```
int func(int j) { return (3*j*j + 7*j - 3); }

int main()
{
    int k = 1;
    while (func(k) <= 120)
        k++;
    printf("%d\t%d\n", k, func(k));
    return 0;
}
```

- Example Languages: Algol, Pascal, C
- Loop constructs: for, while, repeat, do-while
- Program = a sequence of procedures/functions
- The focus is on the code — *how* to get things done.

Problems of Procedural Programming

```
const int MAX_ALTITUDE = 11000;
const int MAX_SPEED = 960;

struct Airplane
{
    int altitude;                  // in feet
    int speed;                     // in km/h
};

void takeoff(Airplane B747);
void descend(Airplane B747, int feet);
```

Data and codes are separated

Data is passive; code is active.

- Usually there are some constraints on the variables.
- e.g., the altitude of an airplane must be +ve, but less than some value (o.w. you'll be in space!).
- Notice also that not all speeds are possible in all altitudes.

Example: Problems of Procedural Programming

Question

With the loose relationship between data and codes in PP, how can the constraints be enforced?

```
const int MAX_ALTITUDE = 11000; const int MAX_SPEED = 960;
const int MAX_RUNWAY_SPEED = 400; const int MIN_FLY_SPEED = 350;

struct Airplane { int altitude; int speed; };

void takeoff(Airplane B747)
{
    // initial state: speed == 0, altitude == 0
    B747.speed = (MAX_RUNWAY_SPEED + MIN_FLY_SPEED)/2;
    // accelerate and climb to 1000 ft
    B747.altitude += 1000;
    B747.speed += 200;
    // cruising speed and altitude
    B747.altitude = MAX_ALTITUDE;
    B747.speed = MAX_SPEED;
}

void descend(Airplane B747, int feet);
```

How to Maintain Data Consistency?

- Data/State Consistency:
  Each time we change the value of a member of an Airplane structure, make sure that the new value is valid with respect to the values of other members.
- A snapshot of the values of all data members of an object represents the state of the object.
- Ensuring data consistency is one of the major challenges in (large) software projects.
- The problem becomes even more difficult when the program is modified and new constraints are added.

Example: Add a Data Member

Let's add a data member to the struct Airplane.

```
struct Airplane
{
    int altitude;
    int speed;
    bool flaps_out;
        // Flaps must be extended below a certain speed to
        // gain lift, but they must be retracted before the
        // speed gets to high, otherwise they will be damaged.
};
```

Solution to Maintain Data Consistency

One solution: Define a restricted set of functions that access the data members (of Airplane) which ensure data consistency (here speed and altitude).

```
struct Airplane
{
    int altitude;
    int speed;
};

int set_speed(Airplane A, int new_speed)
{
    // Make sure that we don't violate any constraints
    // when changing the speed of the Airplane
}
```

Solution to Maintain Data Consistency ..

- For this to work, the rest of the program must use only these functions to change an Airplane state, rather than changing the members directly.
- If we now modify the Airplane structure, then we only have to modify the restricted set of functions that directly access the Airplane members, and make sure they don't violate the new constraints.
- Since we don't access the Airplane members directly in the rest of the program, we don't have to worry about keeping the data consistent.
- But how can we make sure that the Airplane members are only accessed by the restricted set of functions?

In procedural programming, we can't . . . .

Object-Oriented Programming

Wouldn't it be great if we could simply make it impossible to access the Airplane members directly, and to enforce the use of the restricted set of functions?

$\Rightarrow$ OOP

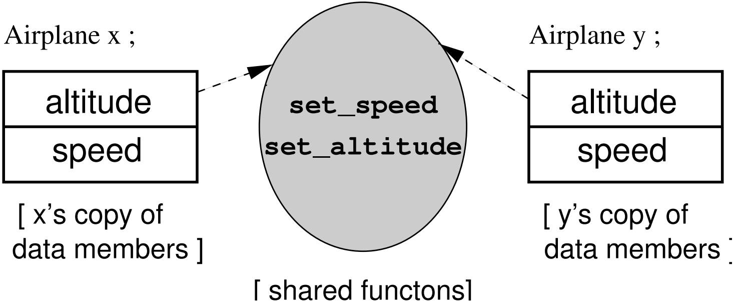
- In OOP, the fundamental entity is the class.
- In contrast with PP, objects are "alive" in OOP: they take care of their own internal state, and "talk" to other objects.
- In OOP, there is little code outside classes.
- Instead of focusing on how to do things (implementation), classes tell the world outside what they can do (interface).

Example: Object-Oriented Programming

```
class Airplane
{
    public:
      int set_speed(int new_speed);
      int set_altitude(int new_altitude);
    private:
      int altitude;
      int speed;
};

void some_function()
{
    Airplane B747;
    B747.set_speed(340);
    B747.set_altitude(1500);
    B747.speed = 3441873923;                // Error: speed is private!
}
```

- A class is a user-defined type representing a set of objects with the same structure and behavior.
- Objects are variables of a class type.
- Instantiation: The process of creating an object of a class is called instantiating an object.
- Each object of a class has its own copies and values of its data members.
- All objects of a class share a common set of member functions.
- To call a procedure in PP, we say "call function X" or simply "call X".
- In OOP, we have to say "invoke method/operation/function X on object Y of class Z".

Private Copy of Members, Shared Functions



OOP: Need Good Design

- There is no magic to OOP or C++: you don't get well-designed and correct programs just because you use classes instead of structs.
- For instance, there is no actual difference between the following struct and class.

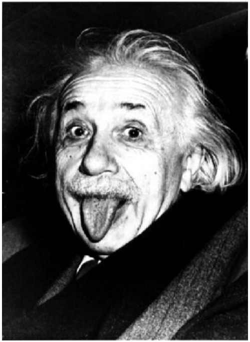
```
struct Airplane
{
    int speed;
    int altitude;
};

class Airplane
{
  public:
    int get_speed() { return speed; }
    int get_altitude() { return altitude; }
    void set_speed(int x) { speed = x; }
    void set_altitude(int x) { altitude = x; }
  private:
    int speed;
    int altitude;
};
```

Supports Needed in OOP Languages

1. data abstraction (abstract data type)
2. data encapsulation (information hiding)
3. inheritance (hierarchy)

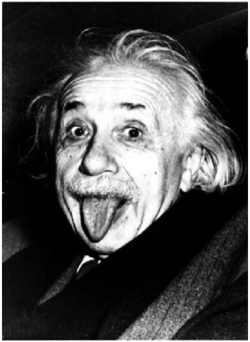
- One can do some OOP in C or other procedural languages, or even in assembly languages!
- What makes an OOP language is its support and enforcement for the above 3 concepts built in its language constructs.
- Example languages: C++, Java, Smalltalk, Eiffel, Object Pascal

Example: Data Abstraction



$\Longrightarrow$

Example: Bad Data Abstraction



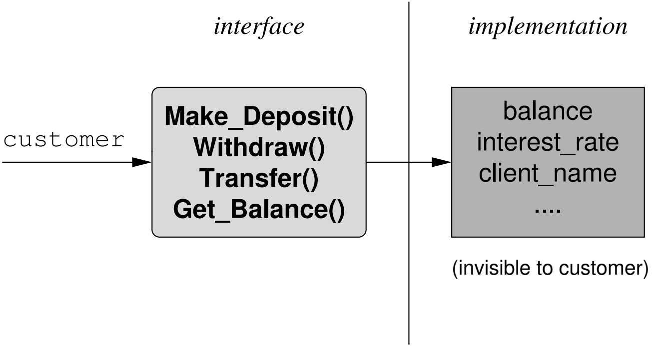
$\Longrightarrow$

Example: C++ Implementation of Abstract Data Type

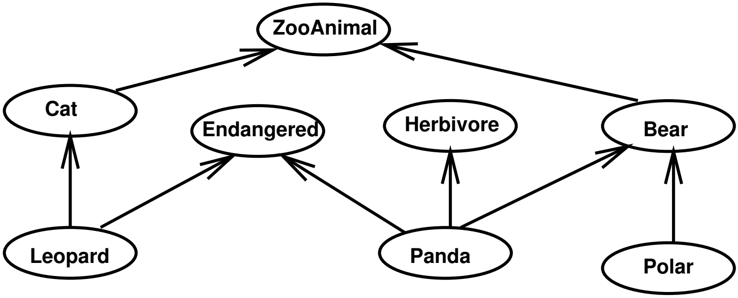
⇒

```cpp
#include <iostream>
#include <cstdlib>
using namespace std;
const int BUFFER_SIZE = 5;

class int_stack
{
  private:
    int data[BUFFER_SIZE];
    int top_index;

  public:
    int_stack(void);
    bool empty(void) const;
    bool full(void) const;
    int size(void) const;
    int top(void) const;
    void push(int);
    void pop(void);
};
```

Example: Data Encapsulation



Example:Inheritance



Generic Programming and Reusable Code

Generic Programming

Programming with types as parameters so that a function may apply for many different types of data.

- e.g. One single sorting function for int, float, Airplanes, etc.

Reusable Code

It is a dream that a piece of software code is like a Lego block, and one builds a large program like building toys with Lego blocks.

- Codes are reusable if:
  - it is easy to find and understand
  - it can assumed to be correct
  - its interface is clear and generic enough
  - it requires no change to be used in a new program

When properly applied, OOP and generic programming can help a lot in writing reusable codes.