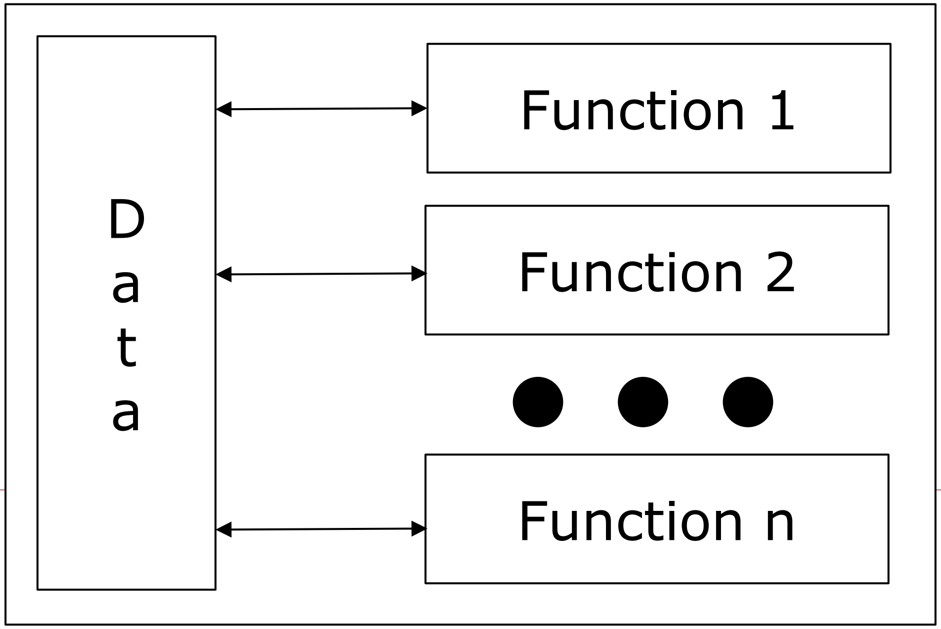
Classes and Objects

Jiun-Long Huang
National Chiao Tung University

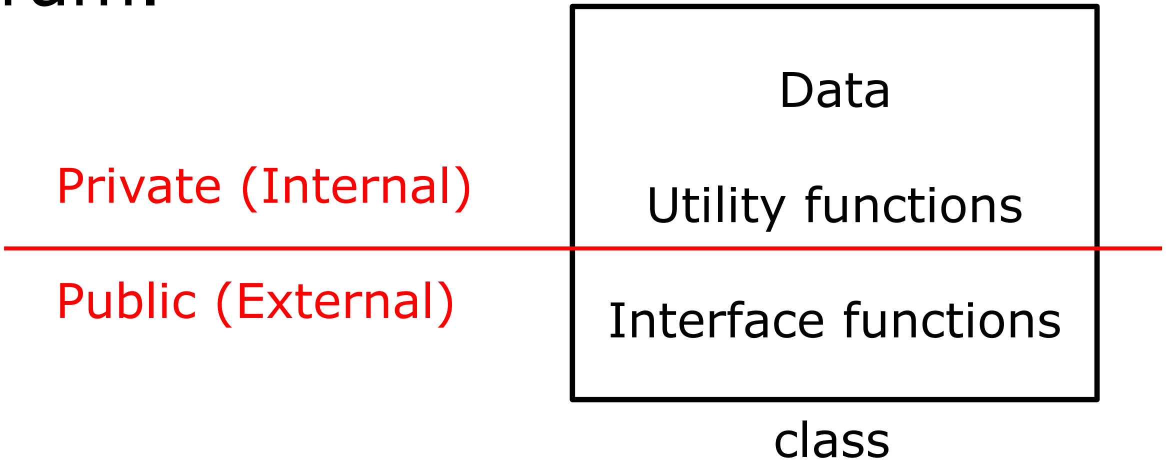
Object-Oriented Programming

- A struct (in C) is a single entity that groups related data
- An object (in C++) is a single entity that groups (1) related data and (2) functions that operate on that data



Object-Oriented Programming (contd.)

- Some functions can be used as utility functions within the object while the others serve as interface functions to communicate with other objects within a program.



Private (Internal)

Public (External)

Data

Utility functions

Interface functions

class

OOP Concept

- Encapsulation is used to refer to one of two related but distinct notions, and sometimes to the combination thereof:
  - A language mechanism for restricting access to some of the object's components.
    - public/protected/private in C++
  - A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.
    - class/object in C++

http://en.wikipedia.org/wiki/Encapsulation_(computer_science)

OOP Concept (contd.)

- Abstraction is the process by which data and programs are defined with a representation similar to its meaning (semantics), while hiding away the implementation details.

http://en.wikipedia.org/wiki/Data_abstraction

OOP Concept (contd.)

- Information hiding is the principle of segregation of the design decisions in a computer program that are most likely to change, thus protecting other parts of the program from extensive modification if the design decision is changed.
  - The protection involves providing a stable interface which protects the remainder of the program from the implementation (the details that are most likely to change).

http://en.wikipedia.org/wiki/Data_hiding

OOP Concept (contd.)

- Polymorphism is a programming language feature that allows values of different data types to be handled using a uniform interface.
  - Ad-hoc polymorphism
    - The function denotes different and potentially heterogeneous implementations depending on a limited range of individually specified types and combination.
    - Function overloading in C++.

http://en.wikipedia.org/wiki/Polymorphism_(computer_science)

OOP Concept (contd.)

- Parametric polymorphism (generic programming)
  - All code is written without mention of any specific type and thus can be used transparently with any number of new types
  - Template in C++
- Subtype polymorphism is a concept wherein a name may denote instances of many different classes as long as they are related by some common super class.
  - Inheritance in C++

http://en.wikipedia.org/wiki/Polymorphism_(computer_science)

C++ Structures vs. C Structures

- Structures are used in C programming to group related variables together.

```
struct circuit {  // C structure declaration
  char description[10];
  int quantity;
  float impedance;
};
struct circuit amplifier, speaker;
```

Difference

- C: struct should be used to define the variable
- C++: no need struct.

```
//structure variable declarations in C
struct circuit amplifier, speaker;
//using typedef
typedef struct circuit circuit;
circuit * amplifier2;
----------------------------------------
//structure variable declarations in C++
circuit amplifier, speaker;
```

Separator

- A dot separator (.) is used to separate the structure variable name from its member variable
- An arrow operator (->) is used to when the variable is a pointer

```
speaker.impedance = 8;
amplifier.quantity = 1;
cout << amplifier.description;
amplifier2=&amplifier;
amplifier2->quantity = 2;
```

Example

- Compute for how long a battery can deliver a certain amount of current I to a device (load) at the rated voltage. The battery's voltage V_b and capacity (ampere-hour rating), as well as the impedance Z of the device are given.
- SOLUTION:
  - If given $V_b=12$ V, capacity=20 Ah, $Z=50\Omega$,
  - $I=V_b/Z=12/50=0.24A$
  - Time=capacity/I=20/0.24=83.33 h

```
#include <iostream>
#include <iomanip>
//using namespace std;
struct Battery {          //structure declaration
  float voltage;
  float capacity;
};
void setValues(Battery &); //reference to a structure as a
void getValues(Battery &); //function parameter
float getHours(Battery &, float);
int main()
{
  float imp=50; //device impedance
  Battery b;    //structure variable
  setValues(b); //passing structure variable by reference
  cout<<endl;
  getValues(b);
  cout<<"Device can be powered "<<getHours(b,imp)<<
        " hours.";
  return 0;
}
```

```
void setValues(Battery &rb)
//Gets battery's voltage and capacity from the user
{
  cout<<"Enter battery's voltage: ";
  cin>>rb.voltage;
  cout<<"Enter battery's capacity: ";
  cin>>rb.capacity;
}
void getValues(Battery &rb)
//Displays battery voltage and capacity
{
  cout<<setiosflags(ios::fixed)<<setprecision(1);
  cout<<"Voltage = "<<rb.voltage<<" [V]"<<endl;
  cout<<"Capacity = "<<rb.capacity<<" [Ah]"<<endl;
}
float getHours(Battery &rb, float imp)
//Computes and returns the time
{
  float current = rb.voltage/imp;
  return rb.capacity/current;
}
```

For functionality

For fast execution

C++ Structure

- The structure in C++ is like the class where all members are by default public
  - The structure type in C++ can also include functions as structure members along with the data they process.
  - By default, structure members are public
- It is a good practice to use C-style structure in C++

Classes

- The class is similar to the expanded structures in C that group related data and functions together.
  - By default, the class members are private
  - Public structure members can be used/accessed outside the structure, while private members cannot

```
class class_name {
  public:
    //public data and functions
  private:
    //private data and functions
};
```

Classes (contd.)

- Class member
  - Member variables, also called data members
    - Data members can be viewed as the object's attributes or properties (Noun)
  - Member functions (Verb)
    - Member functions describe its behavior or methods
- Class is a concept while object is an instance of class.

Accessing Class Members

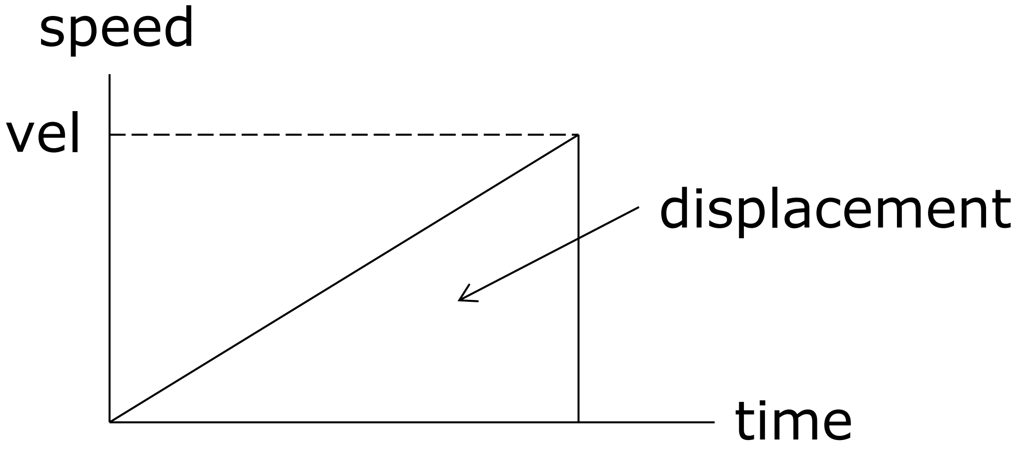
- C++ provides three ways of accessing class members.
  - Private: can only be accessed by other members of the same class.
  - Public: can be accessed by members of its class as well as members of any other class and non-member functions, including main()
  - Protected: when dealing with inheritance

Interface functions

```
class Battery {
  public:  //public structure members
    void setValues() {...};
    void getValues() {...};
    float getHours(float imp) {...};
  private: //private data members
    float voltage; float capacity;
};
Battery b1;
```

```
#include <iostream>
#include <iomanip>
using namespace std;
class Battery {
  // public structure members
public:
  void setValues() {
    cout<<"Enter battery voltage: ";
    cin>>voltage;
    cout<<"Enter battery capacity: ";
    cin>>capacity;
  }
  void getValues() {
    cout<<setiosflags(ios::fixed)<<setprecision(1);
    cout<<"Voltage = "<<voltage<<" [V]"<<endl;
    cout<<"Capacity = "<<capacity<<" [Ah]"<<endl;
  }
  float getHours(float imp) {
    float current=voltage/imp;
    return capacity/current;
  }
private:
    float voltage; //private data members
    float capacity;
};
```

```
int main()
{
  float imp=50;
  Battery b;
  //calling a structure member function
  b.setValues(); cout<<endl;
  b.getValues();
  cout<<"Device can be powered "<<b.getHours(imp)<<
        " hours.";
    return 0;
}
```



```
#include <iostream>
using namespace std;
class Jet{
private:
  float acc, vel; //acceleration, velocity
  float getTime() {//Computes the time during which
    return vel/acc;     //the jet is being accelerated
  }
public:
void setValues(float x, float y){
    acc=x;
    vel=y; //Sets the acceleration and velocity
  }
  float getDisplacement() {
    return (vel*getTime())    //Returns the displacement
  }         //of the jet
};
```

```
int main()
{
  Jet plane;    //Instantiates an object
  plane.setValues(40, 65);//Calls a member function
  cout <<"The time during which the plane ";
  cout <<"is being accelarated = ";
  cout << plane.getTime(); //ERROR!!! (private)
  cout <<"\n The plane's displacement = ";
  cout <<plane.getDisplacement();
  return 0;
}
```

Member Functions

- Member functions are usually used to manipulate class data members, and in most cases provide the only way to access the private class data.
- A member function can be either an inline or non-inline function.
    - To create an inline member function, it is only necessary to place the function's definition inside the class.

Non-inline Function

- Non-inline member functions have their prototypes inside the class and definitions outside the class.

```
class class_name
{
  //Prototype
  return_type function_name(parameters);
}

return_type class_name::function_name(parameters)
{
  //Body of the function
}
```

```
class Jet {
private:
  float acc, vel; //acceleration, velocity
  float getTime();
public:
void setValues(float x, float y){
    acc=x;
    vel=y; //Sets the acceleration and velocity
  }
  float getDisplacement() {
    //Returns the displacement of the jet
    return (vel*getTime())
  }
};

float Jet::getTime() {
    return vel/acc;
}
```

Allocating Objects at Run-Time

- A class object or an array of objects can be dynamically allocated at run-time in the same way as ordinary variables of built-in types.
  - A pointer of the class type and the new operator are needed to perform this operation.
  - The delete operator is used to free memory dynamically allocated to store class object(s).

```
int main()
{
  Jet plane1;       //Instantiates plane1
  Jet *plane2;
  plane2=new Jet(); //Instantiates plane2
  plane1.setValues(40, 65);
  plane2->setValues(30,20);
  // plane2 is manually destroyed
  delete plane2;
  return 0;
  // plane1 is automatically destroyed
}
```

Constructors

- Functions with the same name as the class
    - Default constructor
- One constructor will be invoked when an object is initialized
    - It is a common programming method to use constructors to initialize class data members.

```
class Jet {
  public:
    Jet() {...};
};
```

Constructors (contd.)

- Characteristics of constructors:
  - It has the same name as the class for which it is designed.
  - It has no return type, not even void.
  - It can have arguments, including default arguments.
  - A constructor function is automatically called whenever an object is declared.

Constructors (contd.)

- Constructors should be public, so they can be called outside the class
- A class can have as many constructors as necessary
  - They can be overloaded.
- Constructors cannot be inherited (inheritance will be discussed later).
- Each class should have its own constructors

Constructors (contd.)

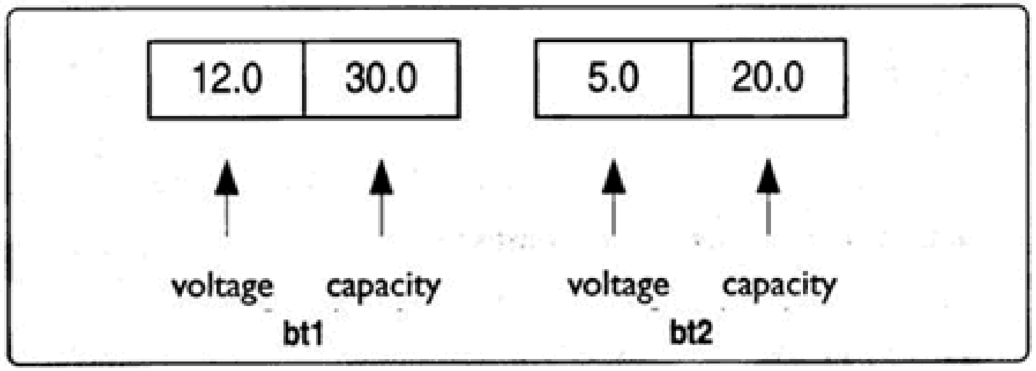
- When an object of the class is declared, the constructor is automatically called.
- With parameters
  - When calling this constructor, two arguments should be passed to the function

```
Battery(float v, float c) {
   voltage = v;
   capacity = c;
}
...
Battery bt; //Calling default constructor
Battery bt2 (1.5, 2.2);
```

Constructors (contd.)

- A constructor function may use default arguments

```
Battery(float v = 12.0, float c = 30.0)
{
  voltage = v;
  capacity = c;
}
```



```
Battery bt1; //use default value
Battery bt2(5.0, 20.0); //Overrides default values
```

Destructors

- Functions with the same name as the class but preceded with a tilde character (~)
- Cannot take arguments and cannot be overloaded
- Performs "termination housekeeping"
- Will be invoked when an object is destroyed
  - Automatically destroy
  - Manually destroy

Destructors (contd.)

```
class ErrMessage {
  private:
    char *message;
  public:
    ErrMessage(char *x) {
      message=new char[strlen(x)+1];
      strcpy(message,x);
    }
    ~ErrMessage() {
      delete [] message; // Cleaning
    }
};
```

Memory leak occurs if ErrMessage does not clean itself

```
class Jet{
  private:
  ...
  char name[50];
  public:
  Jet(char x)
  {
    strcpy(name,x);
    cout <<"Jet "<<name<<“ has been initialized\n";
  };
  ~Jet()
  {
    cout <<"Jet "<<name<<“ has been destroyed\n";
  };
};
```

```
int main()
{
  Jet plane1("Plane1");
  Jet *plane2;
  plane2=new Jet("Plane2");
  delete plane2;
  return 0;
}
```

```
Jet Plane1 has been initialized
Jet Plane2 has been initialized
Jet Plane2 has been destroyed
Jet Plane1 has been destroyed
```

Separating Interface from Implementation

```
// SalesPerson.h
// SalesPerson class definition.
// Member functions defined in SalesPerson.cpp.
#ifndef SALESP_H
#define SALESP_H
class SalesPerson
{
  public:
    SalesPerson();
    void getSalesFromUser();
    ...
  private:
    double totalAnnualSales();
    double sales[ 12 ];
};
#endif
```

```
// SalesPerson.cpp
// Member functions for class SalesPerson.
#include <iostream>
#include <iomanip>
using namespace std;
// include SalesPerson class definition
#include "SalesPerson.h"
SalesPerson::SalesPerson()
{
  for ( int i = 0; i < 12; i++ )
    sales[ i ] = 0.0;
} // end SalesPerson constructor
...
```

```
//main.cpp
//Compile this program with SalesPerson.cpp
//include SalesPerson class definition from SalesPerson.h
#include "SalesPerson.h"
int main()
{
  SalesPerson s; // create SalesPerson object s
  s.getSalesFromUser(); // note simple sequential code;
  s.printAnnualSales(); // no control statements in main
  return 0;
} // end main
```

Should link SalesPerson.o

A Subtle Trap: Returning a Reference to a Private Data Member

- One dangerous way to use this capability
    - A public member function of a class returns a reference to a private data member of that class
        - Client code could alter private data
        - Same problem would occur if a pointer to private data were returned

```
#include <iostream>
using namespace std;
class test
{
  private:
    int value;
  public:
    test() { value=10; }
    int getValue1(void) { return value; }
    int& getValue2(void) { return value; }
    void showValue(void) {
      cout<<"Value: "<<value<<endl;
    }
};
```

```
int main()
{
  test t;
  int v1;
  t.showValue();              //10
  v1=t.getValue1();
  v1++;
  t.showValue();              //10
  int & v2=t.getValue2();
  v2++;
  t.showValue();              //11
  return 0;
}
```

The private member is modified outside the class