Chapter 8

Classes and Objects

Chapter 8 Topics

❖ `struct` Type Declaration

- Declaring `struct` Type
- Accessing `struct` Members
- Aggregate `struct` Operations

❖ `class` Type Declaration

- Declaring `class` Type
- Accessing `class` Members
- Aggregate `class` Operations

❖ Encapsulation （封装）

- Internal and External Views
- Separate Files Used for `class` Type

`struct` Type Declaration

SYNTAX

```
struct  TypeName
{
  // MemberList
   DataType   MemberName ;
   DataType   MemberName ;
        .
        .
        .

 };
```

More about `struct` Type Declarations

If the struct type declaration **precedes(先于)** all functions it will be visible throughout the rest of the file. If it is placed within a function, only that function can use it.

It is common to place struct type declarations with TypeNames in a (.h) header file and # include that file.

struct AnimalType

```
struct  AnimalType                  // declares a  struct data type
{                                   //  does not allocate memory
    long          id ;
    string        name ;
    string        genus ;
    string        species ;
    string        country ;
    int           age ;
    float         weight ;
} ;

AnimalType   thisAnimal ;       // declare  variables of AnimalType
AnimalType   anotherAnimal ;
```

Accessing `struct` Members

After the struct type declaration, the various members can be used in your program when they are preceded by a struct variable name and a dot.

EXAMPLES

thisAnimal.weight

anotherAnimal.country

Aggregate `struct` Operations

- I/O, arithmetic, and comparisons of entire struct variables are **NOT ALLOWED!**

- operations valid on an entire struct type variable:

  - ✓ **assignment** to another struct variable of same type,
  - ✓ **pass** to a function as argument
    (by value or by reference),
  - ✓ **return** as value of a function

Chapter 8 Topics

- `struct` Type Declaration
  - Declaring `struct` Type
  - Accessing `struct` Members
  - Aggregate `struct` Operations
- `class` Type Declaration
  - Declaring `class` Type
  - Accessing `class` Members
  - Aggregate `class` Operations
- Encapsulation
  - Internal and External Views
  - Separate Files Used for `class` Type

class Type Declaration

SYNTAX

```
class ClassName {
public:
        //public data members and member functions
        DataType   MemberName ;
            .
            .
            .
[private:]
        //private data members and member functions
        DataType   MemberName ;
            .
            .
            .
};
```

Visibility Modifiers

The terms *public* and *private* are used to differentiate the internal and external aspects of a class.

- public features can be seen and manipulated (操纵) by anybody -- they are the external (interface or service) view.
- private features can be manipulated only within a class. They are the internal (implementation) view.

class PlayingCard

```
class PlayingCard {
public:
        enum Suits {Spade, Diamond, Club, Heart};
        Suits suit ( ) { return suitValue; }
        int rank ( ) { return rankValue; }
private:
        Suits suitValue;
        int rankValue;
};
```

Object Creation

- declared with class definition:

```
class PlayingCard {
        ...
}aCard;
```

- using the class name as ordinary type define:

```
PlayingCard aCard;
```

- using the new operator to create an object pointer:

```
PlayingCard * pCard = new PlayingCard;
```

Accessing `class` Members

After the class type declaration, the various members can be used in your program when they are preceded by

- an object name and a dot sign
- an object pointer name and an arrow sign

EXAMPLE

```
aCard.rank( )
pCard->rank( )
```

Aggregate class Operations

- built-in operations valid on class objects are:
  - ✓ member selection using dot (.) operator ,
  - ✓ assignment to another class variable using (=),
  - ✓ pass to a function as argument (by value or by reference),
  - ✓ return as value of a function
- other operations can be defined as class member functions

Chapter 8 Topics

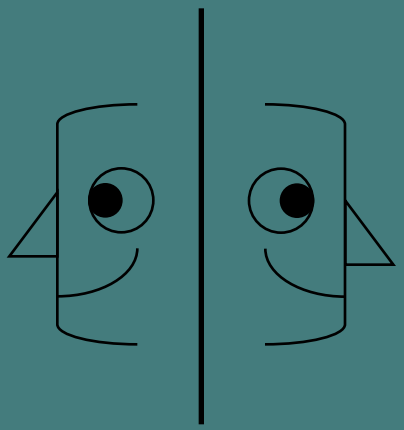
- `struct` Type Declaration
  - Declaring `struct` Type
  - Accessing `struct` Members
  - Aggregate `struct` Operations
- `class` Type Declaration
  - Declaring `class` Type
  - Accessing `class` Members
  - Aggregate `class` Operations
- Encapsulation
  - Internal and External Views
  - Separate Files Used for `class` Type

Encapsulation

- *Encapsulation* - The purposeful hiding of information, thereby reducing the amount of details that need to be remembered or communicated among programmers.
- *A Service View* - The ability to characterize an object by the service it provides, without knowing how it performs its task.

Internal and External Views

The outside, or service view, describes what an object does.



The inside, or implementation view, describes how it does it.

2 separate files Generally Used for `class` Type

```
//   SPECIFICATION FILE                ( timetype .h )
//   Specifies the data and function members.
class TimeType
 {
   public:
        . . .

   private:
        . . .
 };
```

```
//   IMPLEMENTATION FILE               ( timetype.cpp )
//   Implements the TimeType member functions.

        . . .
```

Scope Resolution Operator (::)

In the implementation file, the scope resolution operator is used in the heading before the function member's name to specify its class.

```
void  TimeType :: Write ( )
{       . . .
}
```

Separate Compilation and Linking of Files