ECE 220 Computer Systems & Programming

Lecture 23 – Intro to C++ and Inheritance

ECE ILLINOIS

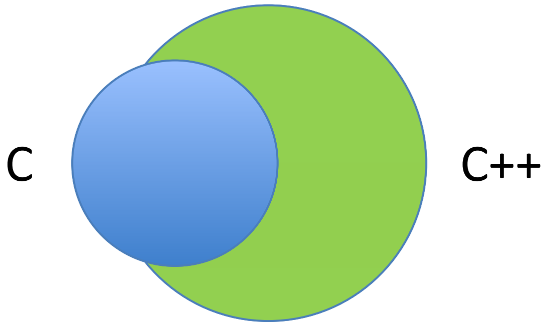
ILLINOIS

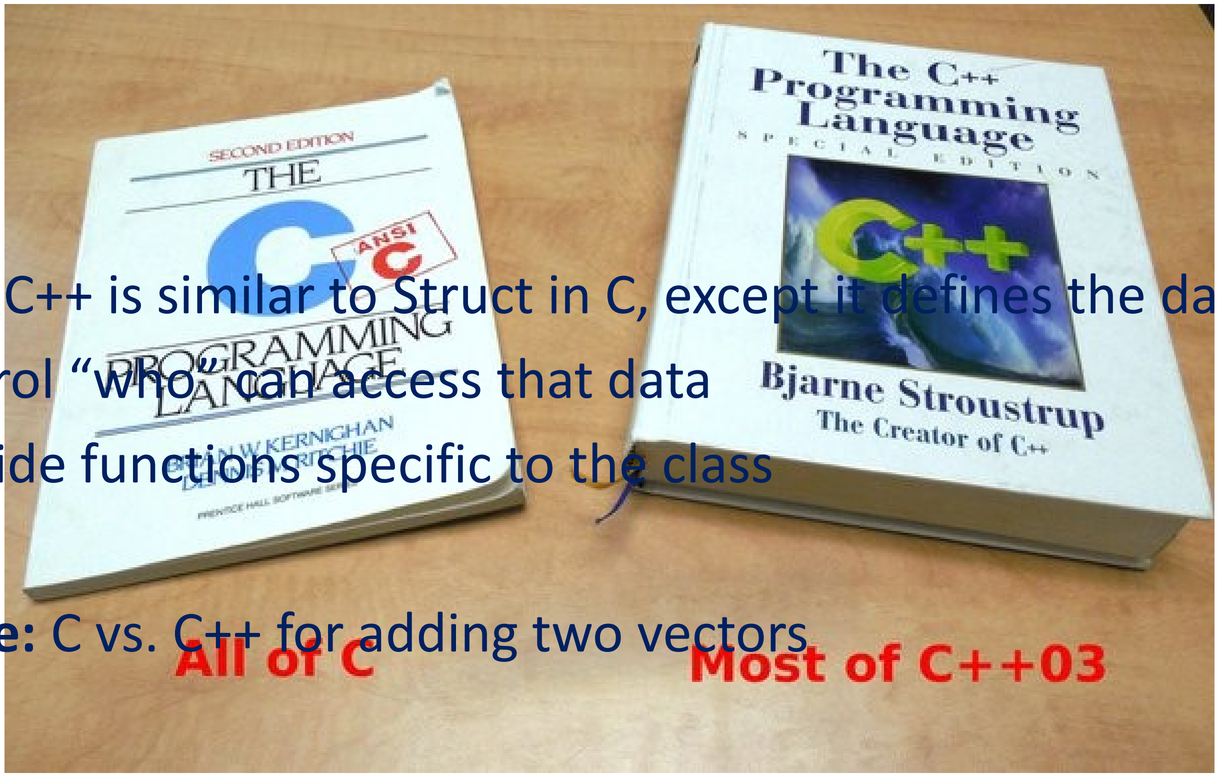
C++ - Class & Encapsulation

- Created in 1979 by Bjarne Stroustrup at Bell Labs, as an extension to C
- It's an **object oriented** language

  OOP Concepts:

  Encapsulation, Inheritance, Polymorphism, Abstraction





Class in C++ is similar to Struct in C, except it defines the data structure **AND**

- control "who" can access that data
- provide functions specific to the class

Example: C vs. C++ for adding two vectors

Concepts Related to Class

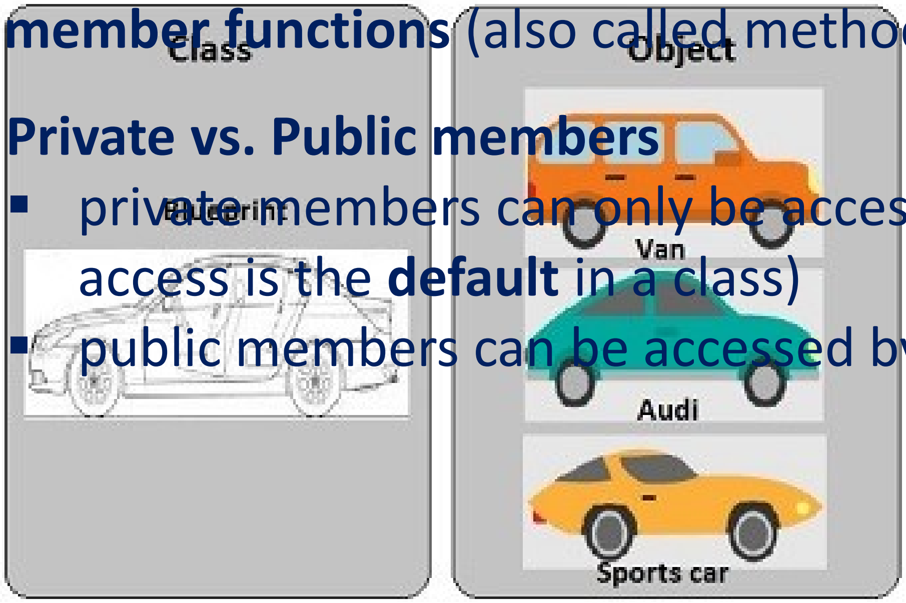
An **object** is an instance of the class

- shares the same functions with other objects of the same class
- but each object has its own copy of the data

member functions (also called methods) - functions that are part of a class

Private vs. Public members

- private members can only be accessed by member functions (private access is the **default** in a class)
- public members can be accessed by anyone



ction that *creates* (initiates) a new object

- Destructor – a special member function that *deletes* an object.

Basic Input / Output

cin – standard input stream
cout – standard output stream

namespace –
“using namespace” directive tells compiler the subsequent code is using names in a specific namespace

Example:
```
#include <iostream>
using namespace std;
int main(){
      char name[20];
      cout << “Enter your name: ”;
      cin >> name; //cin.getline(name, sizeof(name));
      cout << “Your name is: ” << name << endl;
}
```

```
// Fig. 16.4: time1.h
// Declaration of the Time class.
// Member functions are defined in time1.cpp

// prevent multiple inclusions of header file
#ifndef TIME1_H
#define TIME1_H

// Time abstract data type definition
class Time {
public:
   Time();                           // constructor
   void setTime( int, int, int ); // set hour, minute, second
   void printMilitary();             // print military time format
   void printStandard();             // print standard time format
private:
   int hour;      // 0 - 23
   int minute;    // 0 - 59
   int second;    // 0 - 59
};

#endif
```

Outline

Header file (function prototypes, class definitions)

1. Class definition

```cpp
// Fig. 16.4: time1.cpp
// Member function definitions for Time class.
#include <iostream>

using namespace std;

#include "time1.h"

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time() { hour = minute = second = 0; }

// Set a new Time value using military time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
{
   hour   = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

// Print Time in military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
}
```

Outline

Source code file (function definitions)

2.1 Load the header

2.2. Define the member functions

```cpp

// Print time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
}
```

2.2. Define the member functions

```cpp
// Fig. 16.4: fig16_04.cpp
// Driver for Time1 class
// NOTE: Compile with time1.cpp
#include <iostream>

using namespace std;


#include "time1.h"

// Driver to test simple class Time
int main()
{
   Time t;  // instantiate object t of class time

   cout << "The initial military time is ";
   t.printMilitary();
   cout << "\nThe initial standard time is ";
   t.printStandard();

   t.setTime( 13, 27, 6 );
   cout << "\n\nMilitary time after setTime is ";
   t.printMilitary();
   cout << "\nStandard time after setTime is ";
   t.printStandard();

```

Outline

1. Load header

1.1 Initialize object

2. Function calls

3. Print

Outline

```
   t.setTime( 99, 99, 99 );  // attempt invalid settings
   cout << "\n\nAfter attempting invalid settings:\n"
        << "Military time: ";
   t.printMilitary();
   cout << "\nStandard time: ";
   t.printStandard();
   cout << endl;
   return 0;
}
```

Program Output

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

```
// Fig. 16.5: fig16 05.cpp
// Demonstrate errors resulting from attempts
// to access private class members.
#include <iostream>

using namespace std;

#include "time1.h"

int main()
{
   Time t;

   // Error: 'Time::hour' is not accessible
   t.hour = 7;

   // Error: 'Time::minute' is not accessible
   cout << "minute = " << t.minute;

   return 0;
}
```

Outline

1. Load header file for `Time` class.

2. Create an object of class `Time`.

2.1 Attempt to set a `private` variable

2.2 Attempt to access a `private` variable.

Program Output

```
Compiling...
Fig06_06.cpp
D:\Fig06_06.cpp(15) : error C2248: 'hour' : cannot access private
member declared in class 'Time'
D:\Fig6_06\time1.h(18) : see declaration of 'hour'
D:\Fig06_06.cpp(18) : error C2248: 'minute' : cannot access private
member declared in class 'Time'
D:\time1.h(19) : see declaration of 'minute'
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```

Dynamic Memory Allocation

new – operator to *allocate* memory (similar to *malloc* in C)

delete – operator to *deallocate* memory (similar to *free* in C)

Example:

```
int *ptr;
ptr = new int;
delete ptr;

int *ptr;
ptr = new int[10];
delete [] ptr;
```

Exercise – Write Constructors

```
class Rectangle(
      int width, height;
   public:
      Rectangle();
      Rectangle(int, int);
      int area() {return width*height;}
};
Rectangle::Rectangle(){
//set both width and height to 1

}
Rectangle::Rectangle(int a, int b){
//set width to a and height to b


}
```

Exercise – Access Member in a Class

```
int main(){
      Rectangle rect1(3,4);
      Rectangle rect2;

      //print rect1's area


      //print rect2's area


      return 0;
}
```

What is the area of rect1? How about rect2?

Exercise – Pointer to a Class

```
int main(){
      Rectangle rect1(3,4);
      Rectangle *r_ptr1 = &rect1;
      //print rect1's area through r_ptr1

      Rectangle *r_ptr2, *r_ptr3;
      r_ptr2 = new Rectangle(5,6);
      //print area of rectangle pointed to by r_ptr2

      r_ptr3 = new Rectangle[2]{Rectangle(),Rectangle(2,4)};
      //print area of the 2 rectangles in the array


      //deallocate memory


      return 0;
}
```

Inheritance & Abstraction

C++ allows us to define a class based on an existing class, and the new class will inherit members of the existing class.

- the **existing** class –
- the **new** class –

A derived class inherits all base class member functions with the following exceptions:

- Constructors, destructors and copy constructors of the base class.
- Overloaded operators of the base class.
- The friend functions of the base class.

(**ref**: **C How to Program, 6/e, Deitel & Deitel)**

Chapter#20-Inheritance.pdf and related codes are posted on the Github