Lesson 1 - Outline

- **Object-Oriented Programming (OOP)**
- **History of C and C++**
- **Typical C++ Environment**
- **Class Definition**
- **Class Declarations**
- **Member Functions**
- **Class Scope and Accessing Class Members**

Object-Oriented Programming (OOP)

- **OOP** allows to encapsulate **data** (***attributes***) and **functions** (***behavior***) into packages called **classes**
- **Class** (standard unit of programming):
  - Component modeling real world items → modularity
  - Is like (or serves as) a "*blueprint*" → reusability/libraries
  - Contains **functions** and **data**
- **Object (**instance of a class):
  - Is like a "*magazine*" printed from a "*blueprint*"
  - E.g.: date objects, time objects, paycheck objects, invoice objects, audio objects, video objects, file objects, etc.

Object-Oriented Programming (OOP)

- Abstraction - think in terms of houses, not bricks
  - Natural way to think about the world and write computer programs
  - See a photograph rather than a group of colored dots
- Attributes - properties of objects
  - Size, shape, color, weight, etc.
- Behaviors - actions
  - A ball rolls, bounces, inflates and deflates
- Inheritance
  - New classes of objects absorb characteristics from existing classes
- Information hiding
  - Objects usually do not know how other objects are implemented
  - Implementation details are hidden within the classes themselves

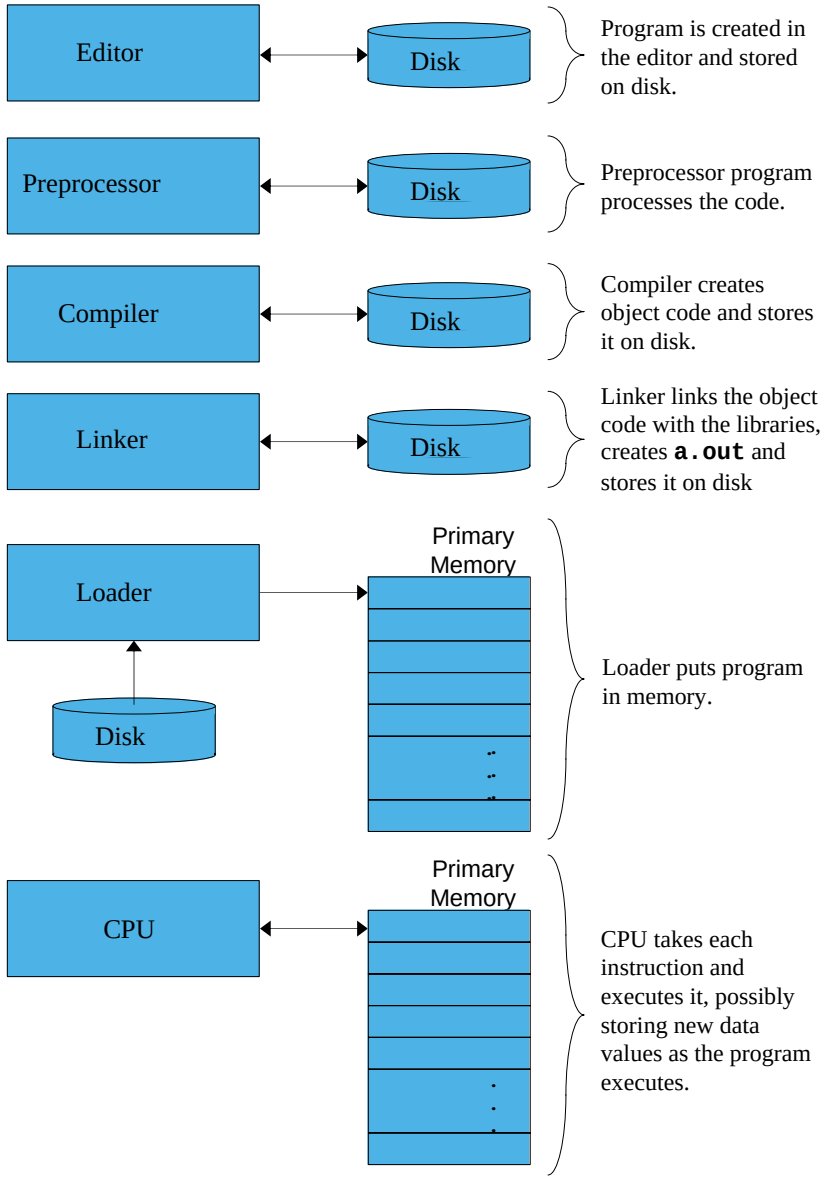
History of C and C++

- C++ evolved from C
  - C evolved from BCPL and B
- ANSI C
  - Established worldwide standards for C programming
- C++ “spruces up” C
  - Provides capabilities for object-oriented programming
  - Object-oriented programs are easy to
    - Understand
    - Correct
    - Modify

Typical C++ Environment

Phases of C++ Programs:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



Class Definition

- Starts with the keyword **`class`**
- Has a body delineated with braces **`{`** … and … **`}`**
- Terminates with a semicolon
- Contains
  - attributes (or *data members*)
  - behaviors (or *member functions*)
- Example:

```
1  class Time {
2  public:
3     Time();
4     void setTime( int, int, int );
5     void printMilitary();
6     void printStandard();
7  private:
8     int hour;     // 0 - 23
9     int minute;   // 0 - 59
10    int second;   // 0 - 59
11 };
```

`public:` and **`private:`** are member-access specifiers.

`setTime`, **`printMilitary`**, and **`printStandard`** are **member functions**.
`Time` is the **constructor.**

`hour`, **`minute`**, and **`second`** are **data members**.

Class Definition

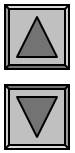
- Member access specifiers
  - Classes can limit the access to their member functions and data
  - The three types of access a class can grant are:
    - **`public`** — Accessible wherever the program has access to an object of the class
    - **`private`** — Accessible only to member functions of the class
    - **`protected`** — Similar to private and discussed later
- Constructor
  - Special *member function* that initializes the *data members* of a class object
  - Cannot return values
  - Have the same name as the class

Class Declarations

- Once a class has been defined, it can be used as a type in object, array and pointer declarations
- Example:

```
Time sunset;                    // object of type Time
Time arrayOfTimes[ 5 ];         // array of Time objects
Time *pointerToTime;            // pointer to a Time object
Time &dinnerTime = sunset;      // reference to a Time object
```


Note: The class name becomes the new type specifier.



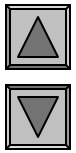
Outline

1. Define a `Time` class

1.1 Define default values for the time

```
1  // Fig. 6.3: fig06_03.cpp
2  // Time class.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // Time abstract data type (ADT) definition
9  class Time {
10 public:
11    Time();                          // constructor
12    void setTime( int, int, int ); // set hour, minute, second
13    void printMilitary();          // print military time format
14    void printStandard();          // print standard time format
15 private:
16    int hour;     // 0 – 23
17    int minute;   // 0 – 59
18    int second;   // 0 – 59
19 };
20
21 // Time constructor initializes each data member to zero.
22 // Ensures all Time objects start in a consistent state.
23 Time::Time() { hour = minute = second = 0; }
24
25 // Set a new Time value using military time. Perform validity
26 // checks on the data values. Set invalid values to zero.
27 void Time::setTime( int h, int m, int s )
28 {
29    hour = ( h >= 0 && h < 24 ) ? h : 0;
30    minute = ( m >= 0 && m < 60 ) ? m : 0;
31    second = ( s >= 0 && s < 60 ) ? s : 0;
32 }
```

Note the `::` preceding the function names.



Outline

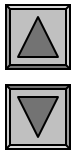
```
33
34 // Print Time in military format
35 void Time::printMilitary()
36 {
37    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
38         << ( minute < 10 ? "0" : "" ) << minute;
39 }
40
41 // Print Time in standard format
42 void Time::printStandard()
43 {
44    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
45         << ":" << ( minute < 10 ? "0" : "" ) << minute
46         << ":" << ( second < 10 ? "0" : "" ) << second
47         << ( hour < 12 ? " AM" : " PM" );
48 }
49
50 // Driver to test simple class Time
51 int main()
52 {
53    Time t;  // instantiate object t of class
54
55    cout << "The initial military time is ";
56    t.printMilitary();
57    cout << "\nThe initial standard time is ";
58    t.printStandard();
59
```

1.2 Define the two functions `printMilitary` and `printstandard`

2. In `main`, create an object of class `Time`

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

Notice how functions are called using the dot (`.`) operator.



Outline

```
60    t.setTime( 13, 27, 6 );
61    cout << "\n\nMilitary time after setTime is ";
62    t.printMilitary();
63    cout << "\nStandard time after setTime is ";
64    t.printStandard();
65
66    t.setTime( 99, 99, 99 );  // attempt invalid settings
67    cout << "\n\nAfter attempting invalid settings:"
68         << "\nMilitary time: ";
69    t.printMilitary();
70    cout << "\nStandard time: ";
71    t.printStandard();
72    cout << endl;
73    return 0;
74 }
```

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

2.2 Set and print the time

e to an

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

2.4 Print the time

```
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
```

Program Output

Member Functions

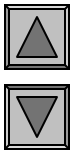
- Binary scope resolution operator (`::`)
  - Combines the class name with the member function name
  - Different classes can have member functions with the same name
- Format for defining member functions

```
returnType ClassName::memberFunctionName( )
{
  …
}
```

- If a member function is defined inside the class
  - Scope resolution operator and class name are not needed
  - Defining a function outside a class does not change it being **`public`** or **`private`**

Class Scope and Accessing Class Members

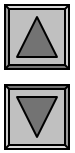
- Function scope
  - Variables only known to function they are defined in
  - Variables are destroyed after function completion
- Accessing class members
  - Same as structs
  - Dot (`.`) for objects and arrow (`->`) for pointers
  - Example:
    - **`t.hour`** is the **`hour`** element of **`t`**
    - **`TimePtr->hour`** is the **`hour`** element



Outline


```
1  // Fig. 6.4: fig06_04.cpp
2  // Demonstrating the class member access operators . and ->
3  //
4  // CAUTION: IN FUTURE EXAMPLES WE AVOID PUBLIC DATA!
5  #include <iostream>
6
7  using std::cout;
8  using std::endl;
9
10 // Simple class Count
11 class Count {
12 public:
13    int x;
14    void print() { cout << x << endl; }
15 };
16
17 int main()
18 {
19    Count counter,               // create counter object
20          *counterPtr = &counter, // pointer to counter
21          &counterRef = counter;  // reference to counter
22
23    cout << "Assign 7 to x and print using the object's name: ";
24    counter.x = 7;          // assign 7 to data member x
25    counter.print();        // call member function print
26
27    cout << "Assign 8 to x and print using a reference: ";
28    counterRef.x = 8;       // assign 8 to data member x
29    counterRef.print();     // call member function print
30
```

It is rare to have **`public`** member variables. Usually only member functions are **`public`**; this keeps as much information hidden as possible.



Outline

2.3 Set a new value and print it using a pointer

```
31    cout << "Assign 10 to x and print using a pointer: ";
32    counterPtr->x = 10;  // assign 10 to data member x
33    counterPtr->print(); // call member function print
34    return 0;
35 }
```

Program Output

```
Assign 7 to x and print using the object's name: 7
Assign 8 to x and print using a reference: 8
Assign 10 to x and print using a pointer: 10
```