Introduction to Programming with C++

Chieh-Sen (Jason) Huang

Department of Applied Mathematics

National Sun Yat-sen University

INTRODUCTION TO
PROGRAMMING
WITH
C++

Third Edition

Contents are based on book by Y. Daniel Liang

C-string v.s. The strings class

- In C++ there are two ways to process strings. One way is to treat them as arrays of characters ending with the null terminator ('\0'). These are known as C-strings.
  ```
  char s1[] = "Good morning"; \\ the length of s1 is 13
  ```
- The other way is to process strings using the string class. The string class hides the low-level storage from the programmer. The programmer is freed from implementation details.
  ```
  string s("Welcome to C++");
  ```

C-string v.s. The strings class

- Learn to use string class from class definition, i.e. (.h) file.

```
+append(s: string): string    |Appends string s into this string object.
+assign(s: string): string    |Assigns string s to this string.
+at(index: int): char         |Returns the character at the position index from this string.
+length(): int                |Returns the number of characters in this string.

string s1("Welcome");
s1.append(" to C++"); // Appends " to C++" to s1
cout << s1 << endl; // s1 now becomes Welcome to C++

string s1("Welcome");
s1.assign("Dallas"); // Assigns "Dallas" to s1
cout << s1 << endl; // s1 now becomes Dallas

string s1("Welcome");
cout << s1.at(3) << endl; // s1.at(3) returns c

string s1("Welcome");
cout << s1.length() << endl; // Length is 7
```

Passing Objects to Functions

- Objects can be passed to a function by value or by reference, but it is more efficient to pass objects by reference.

```
#include <iostream> // pass by value List10_3.cpp Proj0.
// CircleWithPrivateDataFields.h, Listing 9.9
#include "CircleWithPrivateDataFields.h"
using namespace std;

void printCircle(Circle c)
{
  cout << "The area of the circle of "
       << c.getRadius() << " is "
       << c.getArea() << endl;
}

int main()
{
  Circle myCircle(5.0);
  printCircle(myCircle);

  return 0;
}
------------------------------------------------
The area of the circle of 5 is 78.5397
```

- To pass an object argument by value is to copy the object to the function parameter. So the object **c** in the printCircle function is independent of the object **myCircle** in the main function.

Passing Objects to Functions

```
#include <iostream> // pass by reference List10_4.cpp Proj1.
// CircleWithPrivateDataFields.h, Listing 9.9
#include "CircleWithPrivateDataFields.h"
using namespace std;

void printCircle(Circle &c)
{
  cout << "The area of the circle of "
       << c.getRadius() << " is "
       << c.getArea() << endl;
}
------------------------------------------------
The area of the circle of 5 is 78.5397
```

- The object **c** in the printCircle function is essentially an alias of the object **myCircle** in the main function.

- Though you can pass an object to a function by value or by reference, passing by reference is preferred, because it takes time and additional memory space to pass by value.

  In-Class Exercise: Write a code to pass (by reference) a Square object to a function.

Array of Objects

- You can create an array of any objects just like an array of primitive values or strings.

- The name of the array is **circleArray**, and the **no-arg** constructor is called to initialize each element in the array.
  ```
  Circle circleArray[10];
  Declare an array of ten Circle objects
  ```

- You can also use the array initializer to declare and initialize an array using a constructor with arguments.
  ```
  Circle circleArray[3] = {Circle(3), Circle(4), Circle(5)};
  ```

Passing array of Objects

```
double sum(Circle circleArray[], int size)
{
  // Initialize sum, List10_5.cpp Proj2.
  double sum = 0;

  // Add areas to sum
  for (int i = 0; i < size; i++)
  sum += circleArray[i].getArea();

  return sum;
}
int main()
{
  Circle circleArray[10];
  cout << "The sum of areas is "
       << sum(circleArray, 10) << end;
}
```

Static variables and class variables

- An instance variable is tied to a specific instance of the class; it is not shared among objects of the same class.
- If you want all the instances of a class to share data, use **static variables**, also known as **class variables**. Static variables store values for the variables in a common memory location.
- A static function cannot access instance members of the class.

Static variables and class variables

```
class Vector
{
 private:  //List10_2a.cpp
   double x, y;  // coordinate (x, y)
   static double OriX, OriY; // origin

 public:
   static void SetOri(double orix = 0, double oriy = 0)
   {
     OriX = orix;
     OriY = oriy;
   }
   static void ShowOri()
   {
     cout << "[" << OriX << ","
                 << OriY << "]" << endl;
   }
};
double Vector::OriX = 1;
double Vector::OriY = 1;
int main()
{
  // You do not need an instance to use static functions
   Vector::ShowOri();
   Vector V(5,4);
   V.SetOri(2,2);
   // V.ShowOri();
   Vector::ShowOri(); //This improves readability,
     //because the reader can easily recognize the static function.
   V.Show();
}
----------------------------------------------------------
[1,1]
[7,6]
```

Static variables and class variables

- The static data fields OriX and OriY are initialized first.
- Instance functions (e.g., Show()) and instance data fields (e.g., x, y) belong to instances and can be used only after the instances are created. They are accessed from a specific instance.
- Static functions (e.g., SetOri()) and static data fields (e.g., OriX, OriY) can be accessed from any instance of the class, as well as from their class name.
- `Vector::ShowOri();` is better than `V.ShowOri();`, since it improves readability, because the reader can easily recognize the static function.

  In-Class Exercise: Revised member function `double GetLengthTo0()` to accomodate the static data fields OriX and OriY.

Constant Member Functions

- C++ also enables you to specify a constant member function to tell the compiler that the function should not change the value of any data fields in the object.

- To do so, place the const keyword at the end of the function header.

```
class Vector
{
 private:  //List10_2a.cpp
   double x, y;  // coordinate (x, y)

 public:
   void Show() const
   {
    cout << "[" << x + OriX << ","
                << y + OriY << "]" << endl;
          //    << ++y + OriY << "]" << endl;// wrong
   }
};
```

- `void Show() const` indicates that we shall not change the data fileds (x,y).

Constant Member Functions

- Like constant parameters, constant functions are for defensive programming.
- If your function mistakenly changes the value of data fields in a function, a compile error will be reported.
- Note that you can define only instant functions constant, **not static functions**.
- An instance **get function** should always be defined as a constant member function, because it does not change the contents of the object.

Thinking in Objects

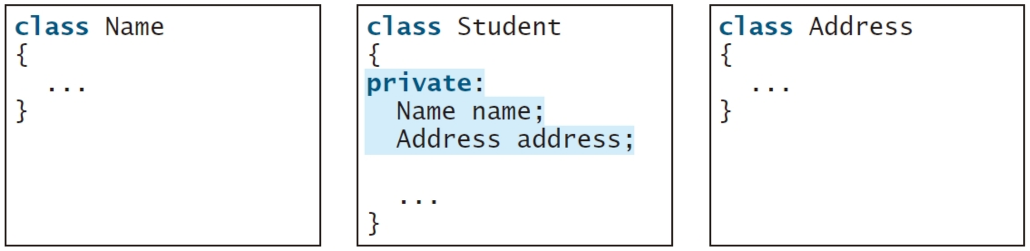
- The procedural paradigm focuses on designing functions.
- The object-oriented paradigm couples data and functions together into objects. Software design using the object-oriented paradigm focuses on objects and operations on objects.
- In procedural programming, data and operations on the data are separate, and this methodology requires sending data to functions.
- Object-oriented programming places data and the operations that pertain to them within a single entity called an object; this approach solves many of the problems inherent in procedural programming.
- The object-oriented programming approach organizes programs in a way that mirrors the real world, in which all objects are associated with both attributes and activities.

Thinking in Objects

- Using objects improves software reusability and makes programs easier to develop and easier to maintain.
- The object-oriented approach combines the power of the procedural paradigm with an added dimension that integrates data with operations into objects.

Object Composition

- An object may be owned by several other aggregating objects. If an object is exclusively owned by an aggregating object, the relationship between the object and its aggregating object is referred to as composition.



```
class Name
{
  ...
}
```

Aggregated class

```
class Student
{
private:
   Name name;
   Address address;

   ...
}
```

Aggregating class

```
class Address
{
  ...
}
```

Aggregated class

- A default constructor: When there are no constructors are explicitly defined in the class, a no-arg constructor with an empty body is implicitly defined in the class.

```
ClassName(parameterList)
  : datafield1(value1), datafield2(value2) // Initializer list
{
// Additional statements if needed
}
-------------------------------------------------------
  Quad()  //See List10_3a.cpp
    : Vec1(0,0), Vec2(0,0) {}
```

Object Composition

- Note that Quad can not access private members of Vector. We have to use public functions of Vector when accessing the private members of Vector.

```
class Quad // List10_3a.cpp
{
 private:
   Vector Vec1; // Object Composition
   Vector Vec2;
 public:
   Quad() // A default constructor
    : Vec1(0,0), Vec2(0,0) {}
   void Set(Vector v1, Vector v2){
      Vec1 = v1;
      Vec2 = v2;}
};
int main()
{
  Vector Vec1(1,2),Vec2(3,4);
  Quad Quad1;
  Quad2.Set(Vec3,Vec4);
}
```

Object Composition

- A friend class could be defined to circumvent this, see Chapter 14.

- In-Class Exercise: Write a member function `void Sub(Quad OtherQuad)` which substract `OtherQuad` from the instance.
  Homework 10.1: Write a code to simulate complex numbers with various properities such as $|z|, \bar{z}, z^2$ of complex numbers.
  Homework 10.2: Programming exercise: 10.10.

- Go back to Chapter 11.