C++ at Velocity, Part 3

CS 002 - WI 2016
January 8, 2016

Last time...

Git introduction
Pointers
Pointer arithmetic
Array-pointer equivalence
`gdb`

Classes

Like Python classes, can have member variables and functions

```
class Polygon {
private:
    double width, height;
public:
    Polygon() { ... }
    ~Polygon() { ... }
    void SetValues(double w, double h)
     { ... }
};
```

Classes

Classes have **member visibility**.

- **Private**: accessible only to class code (default)
- **Public**: accessible to all code
- **Protected**: accessible to class code and subclasses

Encapsulation - protect internal state from unwanted changes; control state changes.

(C++ 1.17, 1.23)

Classes

We usually place a class's declaration in a header file (e.g. `Vector2.hpp`).

```
class Vector2 {
private:
    double x, y;
public:
    Vector2();                          // Constructor
    Vector2(double a, double b);        // Also a constructor!
    ~Vector2();                         // Destructor
    double GetX();                      // Accessor
    double GetY();
    double GetLength();
    void SetX(double val);              // Mutator
    void SetY(double val);
};
```

Classes

We place a class's implementation in a source file (e.g `Vector2.cpp`)

```
// constructor
Vector2::Vector2() {
    x = 0;
    y = 0;
}

double Vector2::Length() {
    return sqrt(x * x + y * y);
}
```

Classes

- `Vector2::Vector2()` is a **constructor**
  - Run whenever an object is instantiated
  - Used to initialize member variables, acquire resources, etc.
  - Can receive arguments
- `Vector2::~Vector2()` is a **destructor**
  - Run whenever an object is deleted or goes out of scope
  - Used to clean up dynamic resources
  - Never receives arguments

Classes

Classes are blueprints for **objects**

also called **instances**

We can create as many independent instances of a given class as we want

```
Vector2 x;
Vector2 y(3.0, 6.0);
Vector2 z = Vector2(5.0, 8.0);

// dynamic memory allocation
Vector2 * p = new Vector2;
Vector2 * q = new Vector2(7.5, 6.3);
Vector2 * arr = new Vector2[80];
```

Classes

Calling destructors

```
void foo() {
    Vector2 x;
    Vector2 y(3.0, 6.0);

    Vector2 * p = new Vector2;
    Vector2 * q = new Vector2(7.5, 6.3);

    …

    delete p;  // call destructor for p
    delete q;  // call destructor for q
}
// Destructors for x and y are called
// when x and y go out of scope
```

Classes

We can now use the **Vector2** class as a new type

```
#include "Vector2.hpp"
   …
   Vector2 v1;

   v1.SetX(314.159);
   v1.SetY(265.358);
   cout << "len(v1) = " <<  v1.GetLength() << endl;
```

But we have no direct access to the internals

```
   v1.x = 3.78; // NOT ALLOWED (compile error)
                // because x is a private member
```

this

In class context, **this** is a pointer to the calling object.

```
void Vector2::SetX(double x) {
    this->x = x;
    // equivalently:
    (*this).x = x;
}
```

Arithmetic on objects?

Recall that we can perform arithmetic on primitive types.

```
int a = 3, b = 4, c;
c = a + b;
```

What if we want to perform arithmetic on things that are not primitive types?

```
Vector2 a(3, 4), b(5, 12), c;
c = a + b; // ???
```

Operator overloading

Operator overloading is the mechanism that lets us do this.

Whenever we use an operator `+`, C++ calls some function `operator+(...)`.

Most operators can be overloaded.

Inheritance

Suppose you have a class for polygons:

```
class Polygon {
protected:
     double w, h;
public:
     double set_dim(double a, double b) {…}
};
```

Class hierarchy

A rectangle is a type of polygon:

```
class Rectangle : public Polygon {
public:
    double area() {
        return w * h;
    }
};
```

Class hierarchy

So is a triangle:

```
class Triangle : public Polygon {
public:
    double area() {
        return w * h / 2.0;
    }
};
```

Class hierarchy

Rectangles and triangles inherit `Polygon::set_dim()`

```
Rectangle a;
Triangle b;

a.set_dim(5.0, 5.0);
b.set_dim(6.0, 4.0);

cout << a.area() << endl; // prints 25.0
cout << b.area() << endl; // prints 12.0
```

Virtual functions

Notice that base-class pointers can point to subclass instances as well!

```
Polygon * p = new Triangle(...);
```

Suppose we defined **`Polygon::area()`**.

We want to be able to do the following...

```
Polygon * p = new Triangle(...);
cout << p->area() << endl;
```

and have it just work.

Virtual functions let us do this.

Virtual functions

```
class Polygon {
public:
   virtual double area();
};

class Triangle : public Polygon {
public:
   double area() { return w * h / 2; }
};
```

(C++ 7.20-23)

Virtual functions

```
Polygon * p1 = new Rectangle(6.0, 4.0);
Polygon * p2 = new Triangle(6.0, 4.0);

cout << p1->area() << endl; // prints 24
cout << p2->area() << endl; // prints 12
```

(C++ 7.20-23)

Abstract base classes

It's possible to define a virtual function with no given implementation:

```
virtual double foo() = 0;
```

This is a **pure virtual** function.

Any class with at least one pure virtual function is an **abstract base class**.

- Cannot be instantiated!
- Can only be used as a base class.

The C++ Standard Template Library

Provides a large basket of built-in algorithms and data structures

Use for your own projects

Template library

- can be used with arbitrary data types, including your own
- Example: list<int> v;

You'll encounter the STL in more depth from HW2 onwards

The C++ Standard Template Library

- Data structures
  - sequence types (list, vector, deque)
  - collection types (set, multiset)
  - mapping types (map, unordered_map)
  - common operations for each
  - iterator objects
- Strings
  - rewritable, resizable
- Other features (including C++11 features)
  - algorithms, random numbers, regexes, ...

Namespaces

Many of the C++ STL constructs are defined in the **`std`** namespace.

Namespaces are used to separate functions and classes with similar names but different origins.

To use a member of a namespace:

```
std::cout << "fish";
```

or

```
using namespace std;
cout << "fish";
```

Further reading

The CS11 C and C++ lecture slides:

- C: http://courses.cms.caltech.edu/cs11/material/c/mike/
- C++: http://courses.cms.caltech.edu/cs11/material/cpp/donnie/

External resources:

- cplusplus.com Tutorial:
  http://www.cplusplus.com/doc/tutorial/
- cplusplus.com Library Reference:
  http://www.cplusplus.com/reference/

Coding style guidelines

We don't really enforce a particular style in this coures

Only requirements:

- Clean
- Readable
- Consistent

```
int a=b+c;
int a = b + c;
```

Coding style guidelines

- Block indentation
  - Use spaces, not tabs
  - Keep indentation consistent (as if Python)

- Variable and function names
  - Try to pick descriptive names
    - But variables can just be `i, j, k`

Coding style guidelines

Commenting: explain, don't repeat
  Don't tell me i++ increments i; tell me why it's there

Write function headers (even small ones)
  Invalid arguments? Return value?