C++ Tutorial

Emily Whiting
(Stolen from Eugene Hsu)
Introduction

• This tutorial offers several things.
  – You’ll see some neat features of the language.
  – You’ll learn the right things to google.
  – You’ll find a list of useful books and web pages.

• But don’t expect too much!
  – It’s complicated, and you’ll learn by doing.
Overview

- Basic syntax
- Compiling your program
- Argument passing
- Dynamic memory
- Object-oriented programming
Basic C++ Program

#include <iostream>
using namespace std;

float c(float x) {
    return x*x*x;
}

int main() {
    float x;
    cin >> x;
    cout << c(x) << endl;
    return 0;
}
Program Structure and Compilation
Functions are declared in `mymath.h`, but not defined. They are implemented separately in `mymath.cc`.
Why?

- **Software engineering reasons.**
  - Separate interface from implementation.
  - Promote modularity.
  - The headers are a contract.

- **Technical reasons.**
  - Only rebuild object files for modified source files.
  - This is much more efficient for huge programs.
// This is main.cc
#include <GL/glut.h>
#include <iostream>
using namespace std;

int main() {
    cout << "Hello!" << endl;
    glVertex3d(1,2,3);
    return 0;
}

% g++ -c main.cc
% g++ -o myprogram -lglut main.o
% ./myprogram

← Include OpenGL functions.
← Include standard IO functions.
← Long and tedious explanation.
← Calls function from standard IO.
← Calls function from OpenGL.
← Make object file.
← Make executable, link GLUT.
← Execute program.
Multiple Files

- main.cc
- mymath.cc
- mydraw.cc

- `g++ -c main.cc`
- `g++ -c mymath.cc`
- `g++ -c mydraw.cc`

- `main.o`
- `mymath.o`
- `mydraw.o`

- `g++ -o myprogram main.o mymath.o drawstuff.o`

→ myprogram
Most assignments include makefiles, which describe the files, dependencies, and steps for compilation.

You can just type make.

So you don’t have to know the stuff from the past few slides.

But it’s nice to know.
Memory and Functions
Dynamic Memory

#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    float f[n];

    for (int i=0; i<n; i++)
        f[i] = i;

    return 0;
}
Dynamic Memory

Allocate the array during runtime using `new`.

No garbage collection, so you have to `delete`.

Dynamic memory is useful when you don’t know how much space you need.

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    float *f = new float[n];

    for (int i=0; i<n; i++)
        f[i] = i;

    delete [] f;
    return 0;
}
```
Standard Template Library

STL `vector` is a resizable array with all dynamic memory handled for you.

STL has other cool stuff, such as strings and sets.

If you can, use the STL and avoid dynamic memory.

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<float> f(n);

    for (int i=0; i<n; i++)
        f[i] = i;

    return 0;
}
```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<float> f;

    for (int i=0; i<n; i++)
        f.push_back(i);

    return 0;
}
float twice1(float x) {
    return 2*x;
}

void twice2(float x) {
    x = 2*x;
}

int main() {
    float x = 3;
    twice2(x);
    cout << x << endl;
    return 0;
}
Argument Passing

There is an incredible amount of overhead here.

This copies a huge array two times. It’s stupid.

Maybe the compiler’s smart. Maybe not. Why risk it?
Argument Passing

void twice3(float *x) {
    (*x) = 2*(*x);
}

void twice4(float &x) {
    x = 2*x;
}

int main() {
    float x = 3;
    twice3(&x);
    twice4(x);
    return 0;
}

← Pass pointer by value and access data using asterisk.

← Pass by reference.

← Address of variable.

← The answer is 12.
Argument Passing

- You’ll often see objects passed by reference.
  - Functions can modify objects without copying.
  - To avoid copying objects (often `const` references).

- Pointers are kind of old school, but still useful.
  - For super-efficient low-level code.
  - Within objects to handle dynamic memory.
  - You shouldn’t need pointers for this class.
  - Use the STL instead, if at all possible.
Object Oriented Programming
Classes

• Classes implement objects.
  – You’ve probably seen these in 6.170.
  – C++ does things a little differently.
C++ objects
or
data with knobs
class Vector3f
{
    public:

        Vector3f(); // default constructor

    private:

        // data members
        float x;
        float y;
        float z;

};

// this semicolon is very important
// without it, you will get 8 pages of error messages
Implementing a class: Vector3f.cpp

#include "Vector3f.h"

Vector3f::Vector3f()
{
    x = 0;
    y = 0;
    z = 0;
}

Default constructor: what happens when you do nothing.

int main()
{
    // default constructor gets called
    // initialized to (0,0,0)
    Vector3f myvec;
}
class Vector3f
{
    public:

        /* default constructor */
        Vector3f();

        /* constructor with parameters */
        Vector3f( float fx, float fy, float fz );

    private:

        /* data members */
        float x;
        float y;
        float z;

};
More constructors

... 

```cpp
Vector3f::Vector3f( float fx, float fy, float fz )
{
    x = fx;
    y = fy;
    z = fz;
}
```

```cpp
int main()
{
    // initialize vector with values
    Vector3f myvec( -5.3f, 1.8f, 42.0f );
}
```
Member functions: operating on data

class Vector3f
{
    public:
        Vector3f();
        Vector3f(float fx, float fy, float fz);
        float absSquared();

    private:
        // data members
        float x;
        float y;
        float z;

};
Implementing absSquared()

```cpp
float Vector3f::absSquared()
{
    return x * x + y * y + z * z;
}
```

```cpp
int main()
{
    Vector3f myvec( 3, 4, 5 );
    cout << myvec.absSquared() << endl;
    // should print "50"
}
```


```cpp
class Vector3f
{
    public:

        Vector3f();
        Vector3f( float fx, float fy, float fz );

        float absSquared();
        float x();
        void setX( float fx );

    private:

        // data members
        float x;
        float y;
        float z;

};
```
Accessors

```cpp
float Vector3f::x()
{
    return x;
}

void Vector3f::setX( float fx )
{
    x = fx;
}

int main()
{
    Vector3f myvec( 3, 4, 5 );
    cout << myvec.x(); // prints “3”
    myvec.setX( 8 );
    cout << myvec.x(); // now prints “8”
}
```
class Vector3f
{
public:
    Vector3f();
    Vector3f( float fx, float fy, float fz );

    float absSquared();
    float x();
    void setX( float fx );

    static float dot( Vector3f u, Vector3f v );
    static Vector3f cross( Vector3f u, Vector3f v );

private:

    // data members
    float x;
    float y;
    float z;
};
Implementing static methods

```cpp
float Vector3f::dot( Vector3f u, Vector3f v )
{
    return u.x * v.x + u.y * v.y + u.z * v.z;
}

Vector3f Vector3f::cross( Vector3f u, Vector3f v )
{
    // left as exercise to see who’s still awake ...
}

int main()
{
    Vector3f v0( 1, 2, 3 );
    Vector3f v1( 4, 5, 6 );

    float dp = Vector3f::dot( v0, v1 );
    cout << dp << endl;
}
```
Plenty of features left

• copy constructors / assignment operator
• destructors (for dynamic memory)
• operator overloading
  • so you can + and - vectors
• templates
• inheritance
• ...
• See http://www.cplusplus.com/doc/tutorial/
Resources

• The C++ Programming Language
  – A book by Bjarne Stroustrup, inventor of C++.
  – Eugene’s favorite C++ book.

• The STL Programmer’s Guide
  – Contains documentation for the standard template library.

• Java to C++ Transition Tutorial
  – Probably the most helpful, since you’ve all taken 6.170.
  – http://www.cs.brown.edu/courses/cs123/javatoc.htm

-- http://www.cplusplus.com/doc/tutorial/