

C++ Tutorial

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.
 - But I'll give it my best shot, okay?

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;
}
```

← Includes function definitions for console input and output.

← Function declaration.

← Function definition.

← Program starts here.

← Local variable declaration.

← Console input.

← Console output.

← Exit main function.

Program Structure and Compilation

Multiple Files

```
// This is main.cc
#include <iostream>
#include "mymath.h"
using namespace std;

int main() {
    // ...stuff...
}
```

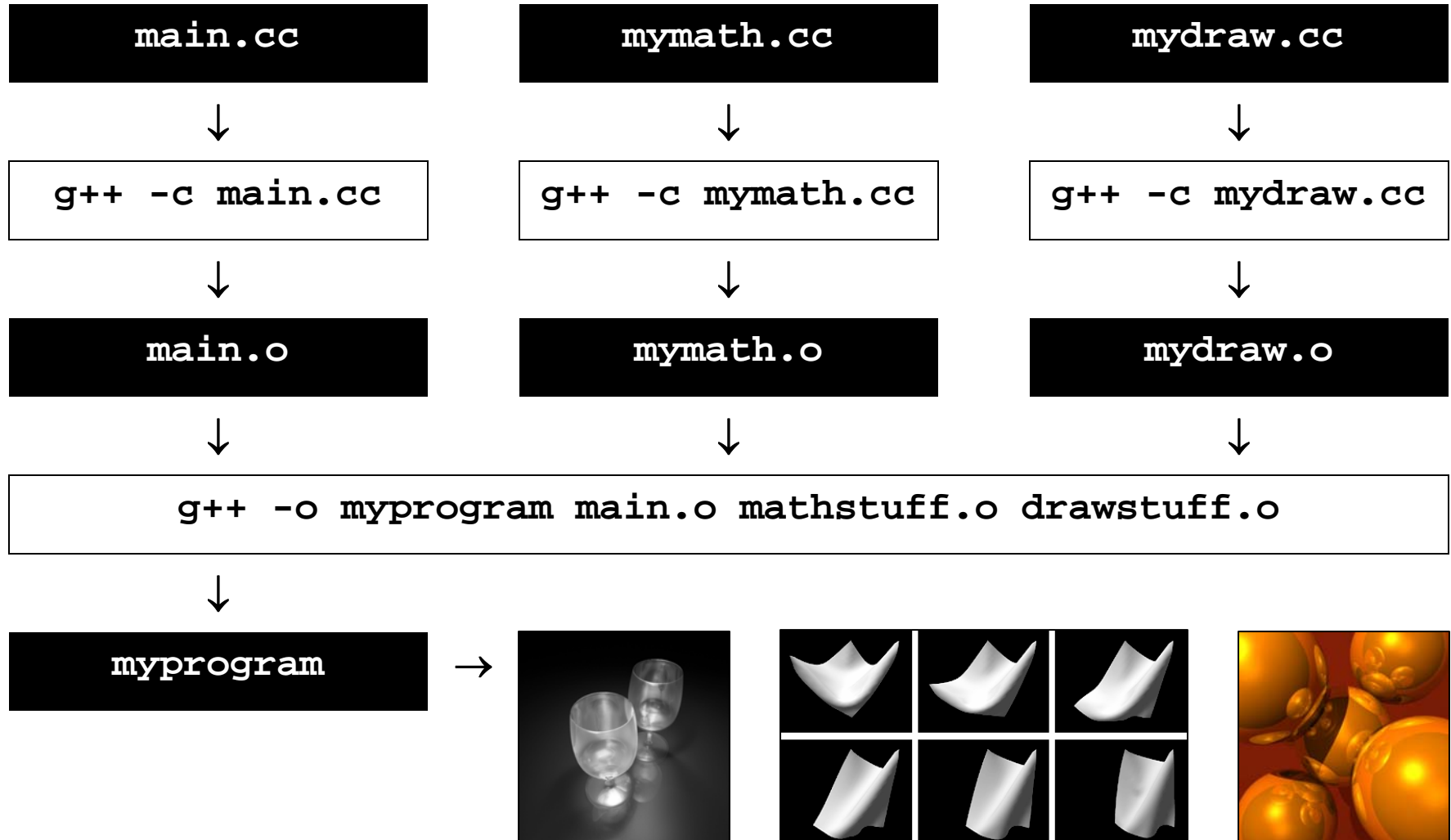
```
// This is mymath.h
#ifndef MYMATH
#define MYMATH

float c(float x);
float d(float x);

#endif
```

Functions are declared in `mymath.h`, but not defined.
They are implemented separately in `mymath.cc`.

Multiple Files



Libraries

```
// 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.

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.

Makefiles

```
INCFLAGS = \  
    -I/afs/csail/group/graphics/courses/6.837/public/include  
LINKFLAGS = \  
    -L/afs/csail/group/graphics/courses/6.837/public/lib \  
    -lglut -lvl  
CFLAGS = -g -Wall -ansi  
CC = g++  
SRCS = main.cc parse.cc curve.cc surf.cc camera.cc  
OBJS = $(SRCS:.cc=.o)  
PROG = a1  
  
all: $(SRCS) $(PROG)  
  
$(PROG): $(OBJS)  
    $(CC) $(CFLAGS) $(OBJS) -o $@ $(LINKFLAGS)  
  
.cc.o:  
    $(CC) $(CFLAGS) $< -c -o $@ $(INCFLAGS)  
  
depend:  
    makedepend $(INCFLAGS) -Y $(SRCS)  
  
clean:  
    rm $(OBJS) $(PROG)  
  
main.o: parse.h curve.h tuple.h  
  
# ... LOTS MORE ...
```

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;
}
```

Arrays must have known sizes at compile time.

This doesn't compile.

Dynamic Memory

```
#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;
}
```

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.

Standard Template Library

```
#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;
}
```

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.

Standard Template Library

```
#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;
}
```

An alternative method that does the same thing.

Methods are called with the dot operator (same as Java).

vector is poorly named, it's actually just an array.

Argument Passing

```
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;  
}
```

← This works as expected.

← This does nothing.

← The variable is unchanged.

Argument Passing

```
vector<float>
twice(vector<float> x) {
    int n = x.size();
    for (int i=0; i<n; i++)
        x[i] = 2*x[i];
    return x;
}

int main() {
    vector<float> y(9000000);
    y = twice(y);
    return 0;
}
```

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.
- Let's implement a simple image object.
 - Show stuff we've seen, like dynamic memory.
 - Introduce constructors, destructors, **const**, and operator overloading.
 - I'll probably make mistakes, so some debugging too.

Classes

Live Demo!

Resources

- *The C++ Programming Language*
 - A book by Bjarne Stroustrup, inventor of C++.
 - My favorite C++ book.
- *The STL Programmer's Guide*
 - Contains documentation for the standard template library.
 - <http://www.sgi.com/tech/stl/>
- *Java to C++ Transition Tutorial*
 - Probably the most helpful, since you've all taken 6.170.
 - <http://www.cs.brown.edu/courses/cs123/javatoc.shtml>