



CS 11 C++ track: lecture 1

- Administrivia

- Need a CS cluster account
 - http://www.cs.caltech.edu/cgi-bin/sysadmin/account_request.cgi
- Need to know UNIX (Linux)
www.its.caltech.edu/its/facilities/labsclusters/unix/unixtutorial.shtml
- Need to know C at level of CS 11 C track
- Track home page:
 - <http://www.cs.caltech.edu/courses/cs11/material/cpp/mike/index.html>



Assignments

- 1st assignment is posted now
- due one week after class, midnight
- late penalty: 1 mark/day
- redos



Textbook

- suggested:

Essential C++ by Stanley Lippman

- covers more material than this track
- Stroustrup NOT recommended!
 - except as reference



Why use C++?

- speed and low-level control of C
- higher level of abstraction than C
- object-oriented programming
- generic programming
- can define own data types that work like built-in data types



What's not to like about C++?

- incredibly complex!
- tons of features that don't always interact well
- takes a long time to master
- suffers from many of C's problems
 - memory leaks, crashes
- much less safe/portable than java
- "language for experts"



Some features of C++

- everything that's in C
- object-oriented programming
 - very powerful (and complex!)
- generic programming (templates)
 - `list<int>`, `multiset<Window *>`
- exception handling
 - better way to deal with errors



Getting started

- The “hello, world!” program:

```
#include <iostream>
// Access values in namespace "std":
using namespace std;
int main()
{
    cout << "hello, world!" << endl;
    return 0;
}
```



Compiling

- Save as "hello.cc" and do:

```
% g++ -Wall hello.cc -o hello
```

```
% hello
```

```
hello, world!
```

- woo hoo!



Alternatively...

- The “hello, world!” program:

```
#include <iostream>
int main()
{
    std::cout << "hello, world!"
               << std::endl;
    return 0;
}
```

- `std::` says use name in “std” namespace



cin and cout (1)

- Don't use `printf()` or `scanf()` for i/o
- Use `cout` for output, `cin` for input

```
int i = 10;
double d = 3.14159;
string s = "I am a string!";
cout << "i = " << i << " d = " << d
      << " s = " << s << endl;
```



cin and cout (2)

```
int i; double d; string s;
cout << "enter i: ";
cin >> i;
cout << "enter d: ";
cin >> d;
cout << "enter s: ";
cin >> s;
cout << "i = " << i << " d = " << d
      << " s = " << s << endl;
```



Objects

- An *object* consists of:
 - data
 - functions (methods) that act on data
- Idea:
 - rest of program only interacts with an object by calling its methods
 - data in object stays private to that object
- C++ supports objects directly



Classes

- A *class* is a template for building an object
 - most C++ code consists of class descriptions
- Classes include:
 - *constructors* (functions that create *instances* of the class *i.e.* new objects of that class)
 - *data members* or *fields* (data associated with each instance)
 - *member functions* or *methods* (functions that can act directly on the data members)
 - *destructors* (functions that destroy instances)



Example: 2d point

```
class Point {  
    private:  
        int x_coord, y_coord;  
    public:  
        Point(); // constructor  
        Point(int x, int y); // constructor  
        void setX(int val); // mutator  
        int getX(); // accessor  
        ~Point(); // destructor  
};
```



Example: 2d point

```
class Point {
```

```
    private:
```

```
        int x_coord, y_coord;
```

```
    public:
```

```
        Point(); // constructor
```

```
        Point(int x, int y); // constructor
```

```
        void setX(int val); // mutator
```

```
        int getX(); // accessor
```

```
        ~Point(); // destructor
```

```
};
```

class



Example: 2d point

```
class Point {
```

```
private:
```

data members (fields)

```
int x_coord, y_coord;
```

```
public:
```

```
Point(); // constructor
```

```
Point(int x, int y); // constructor
```

```
void setX(int val); // mutator
```

```
int getX(); // accessor
```

```
~Point(); // destructor
```

```
};
```



Example: 2d point

```
class Point {  
    private:  
        int x_coord, y_coord;  
    public:                                     constructors  
        Point(); // constructor  
        Point(int x, int y); // constructor  
        void setX(int val); // mutator  
        int getX(); // accessor  
        ~Point(); // destructor  
};
```



Example: 2d point

```
class Point {  
    private:  
        int x_coord, y_coord;  
    public:  
        Point(); // constructor  
        Point(int x, int y); // constructor  
        void setX(int val); // mutator  
        int getX(); // accessor  
        ~Point(); // destructor  
};
```

member functions
(methods)



Example: 2d point

```
class Point {  
    private:  
        int x_coord, y_coord;  
    public:  
        Point(); // constructor  
        Point(int x, int y); // constructor  
        void setX(int val); // mutator  
        int getX(); // accessor  
        ~Point(); // destructor  
};
```

destructor



Example: 2d point

- Put previous code in "Point.hh" file
- Implementation goes in "Point.cc" file:

```
#include "Point.hh"
```

```
Point::Point(int x, int y) {  
    x_coord = x; y_coord = y;  
}
```

```
void Point::setX(int val) {  
    x_coord = val;  
}  
// etc.
```



Using 2d points

- Using 2d points in other code:

```
// ...  
Point p();           // calls no-arg ctor  
Point p2(10, 10);   // calls 2-arg ctor  
cout << p2.getX() << endl; // call method  
p.setX(20);  
// etc.
```



Destructors

- What about destructors?
- destructor is automatically called at end of block where Point objects created
- useful when allocating memory dynamically
- not mandatory
- (not really needed here)



References (1)

- contrast:

```
void setXto10(Point p) { // copies p
    p.setX(10);
}
```

- with:

```
void setXto10(Point *p) { // doesn't copy
    p->setX(10); // (*p).setX(10);
}
```



References (2)

- C++ shortcut:

```
void setXto10(Point &p) { // note the &
    p.setX(10);
}
```

- no copy is made
- like pointers, but with nicer syntax
- use this instead of pointers when possible



Odds and ends

- C header files are included differently
`#include <cmath> // not <math.h>`
- meaning is the same
- don't mix `<cstdio>` and `<iostream>` in the same code!
 - results are "unpredictable"



That's all for now!

- First lab will walk you through these steps.
 - Should be pretty easy.
- Future labs will be much more complicated
 - matrix classes (regular, sparse)
 - operator overloading
 - templates
 - inheritance
- So stay tuned!