6.s096

Lecture 5

Today

- C++ (!)
    - Compiling
    - Memory management
- Classes
- Templates

C++

C++

- Bjarne Stroustrup
- 1983
- Object-oriented (but later than Simula, Smalltalk)
- Like C, but introduces real objects and classes
- (plus loads of other features (kitchen sink?))

g++ (C++ compiler)

- Very similar to gcc.

```
g++ -o test test.cpp
./test
=> hi from C++
```

```
#include <stdio.h>

int main(){
    printf("hi from C++\n");
}
```

Wait... was that really C++?

- Yes.
- C++ is pretty close to being a superset of C.
- We know C, thus we'll build from that knowledge to learn C++.

new memory management syntax

- The new operator allocates space on the heap.
- new and delete take the place of malloc and free.

```
int * numArray = new int[100];
delete numArray;

struct foo * bar = new struct foo; // delete later
```

Classes

Why classes?

- Modularity
- Objects (data + behavior)
- Lets programmers (you) define behavior for your own data

Basic Class Example

```
#include <stdio.h>

class Rectangle {
    int * width;
    int * height;

public:
    Rectangle(int, int); // constructor
    ~Rectangle(); // destructor
    void printMe(){ // 'method' / member function
        printf("Dimensions: %d by %d.\n", *width, *height);
    }
};
```

```
Rectangle::Rectangle(int w, int h){
    // constructor definition
    width = new int;
    height = new int;
    *width = w;
    *height = h;
}

int main(){
    Rectangle box(5, 7);
    box.printMe();
}
```

Constructors and Destructors

- This destructor should have fit on the last slide...:
- Since we explicitly allocated something with new, we must also explicitly de-allocate it.
- Rectangle itself is automatically deallocated when it goes out of scope.

```
Rectangle::~Rectangle(){
    delete width;
    delete height;
}
```

Default constructors

```
Rectangle::Rectangle(){
    width = new int;
    height = new int;
    *width = 5;
    *height = 5;
}
```

```
// no arguments needed!
Rectangle box;
```

Templates

- Syntax for making code more flexible.
- Similar in spirit to Java’s generics.
- Applied at compile-time, like C macros (the preprocessor).
- Can be applied to classes, functions.
- Trivia: language of templates is Turing complete.

Function Template Example

```
template <class typeParam>
typeParam max(typeParam a, typeParam b){
    return (a > b ? a : b);
}

int main(){
    int a = 3, b = 7;
    double c = 5.5, d = 1.5;
    printf("%d\n", max(a, b)); // 7
    printf("%f\n", max(c, d)); // 5.5
}
```

Class Template Example

```
template <class T>
class mypair {
    T a, b;
public:
    mypair(T first, T second){
        a = first;
        b = second;
    }
    T getmax();
};
```

```
template <class T>
T mypair<T>::getmax(){
    return a > b ? a : b;
}

int main(){
    mypair<int> myints(100, 75);
    printf("%d\n",
 myints.getmax()); // 100
```