CS368 Lecture 5

Wednesday 1 October 2014

Reminders:

- HW1 due 11 PM Tuesday (extended)

Last class:

- Pointers (cont'd)
    - Practice
    - Passing Pointers as Arguments
- Dynamic Allocation
- Pointers and Arrays
- Pointer caveats

Today:

- `typedef`
- `void`
- Static variables
- More pointers
    - Pointers as return types
    - Pointers to functions
    - Pointers to pointers
    - Arrays of pointers

The `typedef` keyword

Simplifying and abstracting:

```
typedef vector<int> List;
List l;
l.push_back(99);
```

The `void*` pointer

Static Memory

Pointers Wrapup

Pointers to pointers:

```
int p = new int(10);
int** q = &p;

cout << q << *q << **q;
cout << &p << p << *p;
```

Arrays of pointers:

```
int* a[3];
a[0] = new int(5);
a[1] = new int(2);
a[2] = new int[6];
a[2][0] = 7;

cout << **a;
cout << **(a+2);
```

Pointers as Return Types

```
int* stackfoo() {
    int x = 50;
    return &x;
}


int* heapfoo() {
    int *y = new int(60);
    return y;
}


int* sf = stackfoo();
cout << *sf;


int* sf = stackfoo(); int* hf = heapfoo();
cout << *sf << *hf;
```

Pointers to Functions

```
int foo() { … }

int (*fp) ();
fp = foo; // NOT fp = foo();

fp(5);
(*fp)(5);
```

Why would this ever be useful?

Using `typedef` with function pointers

```
typedef bool (*compareFunction) (int, int);

void sortFunction(int* list, compareFunction cmp) { … }
```