CS 368 Announcements
Wednesday, February 20, 2013

Record your attendance on the sign-in sheet.

Homework
- will be assign later this week
- can use late days

Last Time
- continue Ch. 3
- dereferencing pointers
- pointers to structs/classes
  - use `->` to dereference: `(*s).name` equivalent to `s->name`
- dynamic allocation
- arrays
  - array variables are pointers
  - key relationship: `p == &p[0]` (where `p` is either `int *` or `int []`)
- pointers and `const`
- pointer caveats

Today
- wrap up Ch. 3
- `typedef`
- C++ memory model
- reference variables
- pointers and parameter passing
- pointers and return values

Next Time
- start Ch. 4
- `.h` and `.cpp` files
- intro to defining classes
- multi-file compilation
- makefiles

Defining your Own Types using `typedef`

C++ Abstract Memory Model

Static Memory

- stores literals, global variables, function and class `static` variables
- memory allocation remains during entire execution (i.e., is static)

Call Stack Memory

- stores local (non-static) variables, parameters
- memory allocations automatically managed by function calls and returns

Heap Memory

- stores dynamically allocated values (allocated using `new`)
- programmer must manage memory allocations (freed using `delete`)

Using Reference Variables

Recall `Student` and `Address` structures:

```
struct Address {
    string city;
    int zip;
};

struct Student {
    int id;
    Address addr;
};
```

* and & Summary

Pass-by-reference vs. Passing a Pointer

```
// Pass-by-reference (simple swap)
void swap1(int &x, int &y) {
    int temp = x;
    x = y;
    y = temp;
}
```

```
// Simulated pass-by-reference (simple swap)
void swap2(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

In main:

```
int a = 1, b = 2, c = 3, d = 4;
swap1(a, b);
swap2(&c, &d);
```

Pass-by-value

```
// Pass-by-value with: primitive type, pointer, array
void passByValue(int x, int *y, int z[]) {
    x = 7;
    y = new int;
    *y = 2;
    *z = 3;
    z = new int[3];
    y = new int[2];
    *z = 4;
    *y = 5;
}
```

In main:

```
int a = 1, b = 2;
int e[4];
for (int i = 0; i < 4; i++) {
    e[i] = 3*i;
}

passByValue(a, &b, e);


int f[4];
f = e;
```

Pass-by-reference

```
// Pass-by-reference with: primitive type, pointer
// Note that body of function is same as body of passByValue
void passByRef(int &x, int * &y, int *z) {
    x = 7;
    y = new int;
    *y = 2;
    *z = 3;
    z = new int[3];
    y = new int[2];
    *z = 4;
    *y = 5;
}
```

In main:

```
int a = 1, b = 2;
int e[4];
int f[] = {4, 3, 2, 1};

passByRef(a, &b, f);
passByRef(a, e, f);

int *g = new int[4];
for (int i = 0; i < 4; i++){
    g[i] = i;
}

passByRef(a, g, f);
```

Pass-by-reference (cont.)

```
// Pass-by-reference with pointers: good use vs. bad use
void passByRef2(int * &x, int * &y) {
    int z[4];
    z[0] = 3;
    x = new int[2];
    y = z;
}
```

In main:

```
int *p = new int;
int *q = new int;
*p = *q = 8;
passByRef2(p, q);
```

Return Values

```
// Return value: primitive value
int returnVal1() {
    int x = 4;
    return x;
}

// Return value: pointer (bad use)
int *returnVal2() {
    int x = 5;
    return &x;  // note: generates warning, but compiles
}

// Return value: pointer (good use)
int *returnVal3() {
    int *x = new int;
    *x = 6;
    return x;
}
```

In main:

```
int aa = 1;
int *bb, *cc;
aa = returnVal1();
*bb = returnVal1();
bb = returnVal2();
cc = returnVal3();
```