CS368 Lecture 3

Wednesday 17 September 2014

Reminders:

- Readings

Last class:

- Includes and Namespaces
- Basic Types, Enumerated Types
- Editing, compiling, running C++ programs
- Structures, Arrays

Today:

- Vectors
- Pointers and Reference Variables
- Parameter Passing

Arrays in C++

Similar to Java Arrays, with some critical differences.

Notation:

Bounds:

Search example (code on website)

Cards example (code on website)

Vectors

Standard Template Library

Including the vector header file

Declaring vectors

What's wrong with

```
vector<int> myVector();
```

?

Using vectors

A quick look at Functions

Default arguments

Pointers and Reference Variables

```
int x = 5;
int y = x + 1;
```

What are these instructions doing exactly?

In C++ you can access and manipulate memory addresses directly.

```
int *p = &x;
```

Pointer variables

Reference variables

Constant pointer, implicitly dereferenced (always)

Parameter Passing

Formal parameters:

```
// Function declaration/definition
int lesser(int a, int b) { … }
```

Here, `a` and `b` are “formal” parameters.

Actual parameters:

```
int x = lesser(10, 14);
int y = lesser(m, n); // m, n previously defined
```

Ways of passing parameters to functions:

In `main()`:

```
int a = 5;
foo(a);
print(“a”, a);
```

Pass by Value:

```
void foo(int x) {
    x = x + 1;
    print(“x”, x);
}
```

Pass by Reference:

```
void foo(int &x) {
    x = x + 1;
    print(“x”, x);
}
```

Discussion: By Value and By Reference

Pass by Constant Reference

Actual parameter cannot be changed

```
int findIndex(const vector<int> &v, int query) {
    v.push_back(500);
    for (int i = 0; i < v.size(); i++) {
        if (v[i] == query) return i;
    }
    return -1;
}
```

Calling code:

```
vector<int> nums;
nums.push_back(10);
nums.push_back(15);
nums.push_back(20);
nums.push_back(25);
cout << "nums.size() is " << nums.size() << endl;
cout << findIndex(nums, 20) << endl;
cout << "nums.size() is " << nums.size() << endl;
```

Practice with Pointers and References

```
int x = 9;
int* ptrToX = &x;
int& refToX = x;
```

```
int a = 15;
int *p = &a;
```

What do we see if we print out

- `a`
- `*p`
- `&a`
- `p`

```
int b = 17;
p = &b;
*p = 20;
```

What do we see if we print out

- `b`
- `*p`
- `&b`
- `p`

Passing Pointers as Arguments

```
void foo(int* px) {
    *px = *px + 1;
    print("px", px);
}
```

In `main()`:

```
int a = 5;
foo(&a);
print("a", a);
```

On Your Own

- What is the difference between a formal parameter and an actual parameter?

- Run the examples of different ways of passing parameters (by value, by reference, by constant reference)

- Examine and play with `pointerBasics.cpp`