CS 368 Announcements
Wednesday, February 6, 2013

Record your attendance on the sign-in sheet.

Programming Assignment 1
- due Friday, February 15 by 10:00 pm

Last Time
- booleans
- constants
- enumerations
- structures
- *cardExample.cpp*
- arrays

Today
- arrays
- vectors
- start Ch. 3
- variables, references, pointers
- parameter passing
- pointer basics

Next Time
- continue Ch. 3
- pointers to structs/classes
- dynamic allocation
- pointers and arrays
- pointers and const
- pointer caveats

Arrays

Declaration (and creation)

Accessed as in Java

Warnings:

- array elements are not automatically initialized
- no `length` built in – arrays do not know how big they are
- no runtime check for array index out-of-bounds
- array size must be known at compile-time
- can't change size of an array once it's been created

Vectors

In standard library – need to `#include <vector>`

See STL Reference link under Quick Links for more info about vectors

Declaration (and creation):

```
vector<type> identifier;
```

Card example from last week:

```
    vector<Card> deck2;

    // Fill it
    for (int s = 0; s < 4; s++) {
        for (int val = 1; val <= 13; val++) {
            deck2.push_back(deck[s][val-1]);
        }
    }

    // Print it out
    cout << endl << "Printing vector deck: " << endl;
    for (int i = 0; i < 52; i++) {
         printCard(deck2[i]);
         cout << ", ";
    }
    cout << endl;

    return 0;
}
```

Variables, References, Pointers Overview

Variables

- are identifiers associated with memory
- must be declared before used
- can be initialized with a value
- can be uninitialized
- can be assigned a value

Variable (regular)

Reference variable

Pointer variable

Parameter Passing

actual parameter = parameter value (argument)

```
printcard(c1);
```

formal parameter = parameter variable (parameter)

```
void printCard(Card c) …
```

Java - all parameters are **pass-by-value**

C++ - has three ways to do parameter passing**:**

Which way is specified in function's header (prototype):

Parameter Passing (cont.)

Pass-by-value

- formal parameter gets a copy of the actual parameter's value

Pass-by-reference

- formal parameter gets a reference to the actual parameter.

Parameter Passing (cont.)

Pass-by-const-reference

- formal parameter gets a reference to the actual parameter
  but the actual parameter cannot be changed by the function.

Passing pointers

Pointer Basics

Given

```
int i = 11, j = 22;
```

Declaring

```
int *k, *m = &j;
int *n = new int();
int *o;
```

working with the pointER

```
k = &i;
o = new int;
k = new int(7);
m = n;
delete n;
n = NULL;
```

working with the pointEE

```
*k = i;
*m = 11;
cout << *o << endl;
*o = *k + 5;
cout << *o << endl;
```