CSCI 2270
Data Structures and Algorithms
Lecture 5

Elizabeth White
elizabeth.white@colorado.edu
Office hours: ECCS 128
Wed 1-2pm
Thurs 2-3pm

Dynamic memory arrays

Read http://www.cplusplus.com/doc/tutorial/dynamic/ for next week.

Read Chapter 1 of CLRS (Intro to Algorithms) for next week. We'll begin talking about the speed of simple algorithms on arrays as we move into next week.

Getting the most out of the CS department and the school

- Undergraduate groups in computer science
  - www.facebook.com/InnovateCU Undergraduate events
  - http://innovatecu.com/ Future site for undergraduate events
  - http://wic.cs.colorado.edu/ Women in Computing
  - CU ACM student chapter (link TBD)
  - Game Development club (link TBD)
- Departmental talks (colloquia)
  - http://www.colorado.edu/cs/colloquia/colloquium-schedule
  - A few of these involve our own faculty
- Other departmental talks
  - IQBio: http://biofrontiers.colorado.edu/events/upcoming-events
  - Linguistics: http://www.colorado.edu/linguistics/talks/
  - Electrical Eng: http://ecee.colorado.edu/news/seminars.html

Input variables, by value

```
int triple_a_number_1(int starting_number)
{
    // starting_number gets passed in as a copy
    // and then we triple this copy
    starting_number *= 3;
    // return another copy of our copy of the
    // new starting number
    return starting_number;
    // destroy our passed-in copy of the starting
    // number
}
```

Input variables, by reference

```
int triple_a_number_2(int& starting_number)
{
    // starting_number gets passed in by address
    // temporary copy of starting_number gets made
    starting_number *= 3;
    // we send back a copy of our copy of the
    // starting_number
    return starting_number;
    // now we destroy our temporary copy of the
    // starting number
}
```

Input variables, by reference

```
int main()
{
     int lebowski = 9;
     // copy of lebowski is passed in
     cout << triple_a_number_1(lebowski) << endl;
     cout << lebowski;
     int bob = 9;
     cout << triple_a_number_2(bob) << endl;
     cout << bob;
 }
```

Input variables, by const reference

```
int triple_a_number_3(const int& starting_number)
{
   starting_number *= 3;   // NO, compile error
   return starting_number;
}
```

Protects inputs from changing, while passing them by reference

Why do this? It’s not a great idea for passing an int. But if you have a big variable that takes up a lot of room, this lets you avoid copying it like a value parameter, while still protecting the original …

Normal return by value

```
int get_a_number_1()
{
  int answer;
  cout << "Tell me a number: " << endl;
  cin >> answer;
  return answer;
}
int main()
{
  int number = get_a_number_1();
  cout << "You entered  " << number << endl;
}
```

Dangerous return by reference

```
    int& get_a_number_2()
    {
      int answer;
      cout << "Tell me a number: " << endl;
      cin >> answer;
      return answer;
    }
    int main()
    {
      int other_number = get_a_number_2();
      cout << "You entered " << other_number <<
endl;
    }
```

Dynamic memory arrays

```
int* int_array_maker(unsigned int size)
 {
      int* heap_array;
      heap_array = new int[size];
      return heap_array;
 }
```

Using the new command causes memory to be allocated from a different memory pool, called the heap. Heap variables don’t get destroyed when they hit a closing bracket }.

Lots more space in the heap than in local memory (the stack).

Dynamic memory arrays

```
void int_array_maker_and_destroyer(unsigned int
size)
 {
      int* heap_array;
      heap_array = new int[size];
      delete [] heap_array;
 }
```

Heap variables have to be destroyed using the delete command. When the heap variable is an array, we add empty square brackets [] to the delete command.

Dynamic memory arrays

```
struct int_array {
      static const unsigned int
DEFAULT_CAPACITY = 20;
      int* data;
      unsigned int count;
      unsigned int capacity;
};
```

Array starts out as 20-slot array.

When you run out of room, you

make a new array with double the capacity,

copy your integers from the old array to the new array, delete the old array,

and update the capacity variable.

Dynamic memory array variables

```
struct int_array {
      static const unsigned int 
DEFAULT_CAPACITY = 20;
      int* data;
      unsigned int count;
      unsigned int capacity;
};
```

DEFAULT_CAPACITY: all arrays begin with capacity 20

data: array of integers we’re storing

count: how many integers we’ve added to the array

capacity: current maximum number of integers we could store