31251 – Data Structures and Algorithms

Week 3

Luke Mathieson

This Week's Fun!

- Simple I/O
- Classes and the importance of destructors
- Exceptions
- Compilation with multiple files
- Data Structures:
  - Queues
  - Stacks

`cin` and `cout`

- As with everything else, C++ has a number of ways to handle basic text input and output.
- We'll cover the "standard" `cin` and `cout`.
  - These are the same idea as Java's `System.in` and `System.out`.
  - But they come with special operators: >> and <<.

cin and cout

- They are in the `iostream` library in the C++ Standard Library.
- Write to standard output with `cout << [stuff to write] ;`
- Read with `cin >> [destination];`
  - To read a whole line: `getline(cin, [variable to read in to])`
- `cerr` also exists for errors (equivalent to `System.err` in Java).
- And now a demo!

File I/O

- Uses the same abstractions as `cin` and `cout`, but with a little more fiddling.
  - Use the library `fstream`.
  - Create an `ofstream` for writing.
  - Create an `ifstream` for reading.
  - For details:
    `http://www.cplusplus.com/doc/tutorial/files/`

Classes and Destructors

- You may have noticed a weirdly named function last week: `~intLinkedList()`.
- This is a *destructor*.
- This is a special method that's run when an object has the special `delete` operator called on it.
  - Syntax: `delete [pointer to thing to delete]`.
  - For arrays: `delete[] [array variable]`.

Why do we `delete` things?

- `delete` is needed when we've created something with `new`.
- Otherwise the heap memory is not deallocated, and we have a memory leak.
- The programmer has to choose when to do this (so again, don't use `new` unless you mean it!).

Exceptions

- C++ can `throw` exceptions, just like Java.
- It has `try ... catch([Exception Type] [parameter name]) ..` like Java.
- So why am I telling you this?
- Because C++ can throw *anything*. Demo!
- C++ does define a set of exceptions, defined in `<exception>`: `http://www.cplusplus.com/doc/tutorial/exceptions/`

Compiling with multiple files

Demonstrated in class (but a simple search will give lots of examples and even different, confusing ways to do it).

Queues

- The (basic) Queue is the basic FIFO (first-in-first-out) data structure.
- It keeps things in order (like a list), but...
- Things can only be added to the back, and
- Things can only be taken off the front.
- Normally has an unbounded capacity.

A Pure-ish Virtual Class for a Queue of ints

```
class intQueue {

public:

  virtual ~intQueue() {};
  virtual void enqueue(int n) = 0;
  virtual int dequeue() = 0;
  virtual int peek() = 0;

};
```

Other types of Queue

- A Deque is a double-ended queue - you can add and remove at both ends!
  - This is really useful for implement other data structures.
- A Priority Queue is a queue, but elements are inserted with a priority, and come out in priority order.

Stacks

- A Stack is like a queue, but it's a last-in-first-out (LIFO) data structure.
  - It's like a ... stack of things.
- You can add to the "top", and
- remove from the "top".

A Pure Virtual Class for a Stack of ints

```
class intStack {

public:

  virtual ~intStack() {};
  virtual void push() = 0;
  virtual int pop() = 0;
  virtual int peek() = 0;

};
```

Stacks and Queues

- Stacks and Queues are two of the most used data structures that "do" something.
- Buffers of all kinds are Queues (things go in, and get processed in order eventually).
- Stacks are built into the programming languages you're using - they control how the program functions.
- You'll see more applications of them as the course progress too.

Now let's code something!

In class programming. The results will be uploaded to UTSOnline afterwards.