COMP 2401 -- Tutorial #5
Managing Memory

Learning Objectives

After this tutorial, you will be able to:

- use dynamic memory when writing C programs, specifically:
    - allocate memory
    - pass parameters by reference
    - free memory no longer needed
- use valgrind to debug memory leaks

Tutorial

In this tutorial, we'll be implementing a dynamic array, which grows in capacity as needed.

1. Download the `T05.tgz` file from the tutorial page in *cuLearn*. Untar and read through the tutorial files.

   Rather than tar a file (to make a `.tar` file) and then gzip it (to make a `.tar.gz` file), you can create one file (a `.tgz` file) that does both. You can extract the files in a `.tgz` file with an appropriate option using `tar`.

2. Compile the program with the `-g` flag to include debugging information (this is helpful when using `valgrind`):

   ```
   gcc –Wall –std=c99 -o t05 t05.c t05util.c -g
   ```

   Note that we are compiling two source files. What happens if we compile only `t05.c`?

3. Run `valgrind` on `t05` to check for memory leaks:

   ```
   valgrind ./t05
   ```

4. Fix `t05.c` to remove the memory leaks.

5. Implement the functions `growArray()` and `addStudent()` as prototyped. If `addStudent()` is called and the array is full, use `growArray()` to increase the array's capacity before adding the student.

   Hint: The “array” in StudentArray is a pointer to a dynamically allocated block of memory. Is there any reason we can't point it to a larger, newly allocated block?

6. Use `valgrind` to ensure that the changed program has no memory leaks.

Exercises

1. Write a function `removeStudent(StudentArray *stuArray, int index)` which removes the student at the given index. What impact does the removal have on `stuArray->count`? Make the necessary changes to `stuArray` so that `printArray()` functions correctly after a removal.

2. Alter the function `addStudent()` so that `stuArray` will always be sorted alphabetically by last name.