ECE/CS 250 – Recitation #3 – Prof. Sorin
C with Pointers & Memory Management

Objective: In this recitation, you will learn how to write C programs that use pointers and memory management.

Complete as much of this as you can during recitation. If you run out of time, please complete the rest at home.

1. Task 1: Pointers Are Fun

Write a program that, in main(), declares an array of 100 ints and sets them to the values 0 through 99. (That is myArray[x]=x.) Have main() pass a pointer to this array to a function called sumArray(int* ptr) that returns the sum of all the entries in the array.

2. Task 2: The Joy of Seg Faults

Write a program called bad.c that mis-uses pointers such that it causes a segmentation fault. (Enjoy being asked to do something wrong!) Be sure to use the –g flag when you compile to include debugging info in your binary – without this, you won't be able to determine which line of code caused the fault. E.g., g++ -g -o bad bad.c

Identify the exact line causing the fault using both **gdb** (a command-line debugger that lets you step through programs line by line and can report the cause of a segfault) and **valgrind** (a memory access checking tool used to detect memory leaks and pointer errors). Below is an example where typed commands are blue and evidence of the fault is red. In C programming, your debugging tools are your eyes – you'll be blind if you don't learn to use them!

```
$ g++ -g -o bad bad.c
$ ./bad
Segmentation fault (core dumped)
$ gdb bad
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
                                             ...
Reading symbols from /x/tkbletsc/bad...done.
(gdb) run
Starting program: /x/tkbletsc/bad

Program received signal SIGSEGV, Segmentation fault.
0x00000000004004bc in main () at bad.c:3
3         *p = 55;
(gdb) quit
A debugging session is active.
```

```
        Inferior 1 [process 22024] will be killed.

Quit anyway? (y or n) y
$ valgrind ./bad
==22035== Memcheck, a memory error detector
==22035== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==22035== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==22035== Command: ./bad
==22035==
==22035== Use of uninitialised value of size 8
==22035==    at 0x4004BC: main (bad.c:3)
==22035==
==22035== Invalid write of size 1
==22035==    at 0x4004BC: main (bad.c:3)
==22035==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==22035==
==22035==
==22035== Process terminating with default action of signal 11 (SIGSEGV)
==22035==  Access not within mapped region at address 0x0
==22035==    at 0x4004BC: main (bad.c:3)
                                             ...
==22035== For counts of detected and suppressed errors, rerun with: -v
==22035== Use --track-origins=yes to see where uninitialised values come from
==22035== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 2 from 2)
Segmentation fault (core dumped)
```

3. Task 3: Malloc/Free and Linked Lists

We're going to re-do task 3 from the previous recitation, but with malloc/free. Write a program that uses a two-element struct called HoopsPlayer. This struct has two elements: an int for the player's number, and a float for his average points per game. Write a program that in main() loops and, in each loop, asks the user to input the info for one player (ask for an int then ask for a float). If the user inputs "-1" for the player number, then the loop ends (without asking for a float). You may NOT assume any limit on the number of players entered and thus you MUST use malloc to allocate new entries in the list. After the loop ends, main() must call a function that you write called sortList(<args>) that prints the list of players sorted in ascending order of points per game. IMPORTANT: main() must pass a pointer to sortList(). You may not use global variables for this purpose.

During Task 3, I highly encourage you to run your program through the debugger gdb. Even if your program works fine, you should get used to using gdb. Remember that you have to compile with the –g flag to create a binary for use with gdb.