CMPE-380: Applied Programming

Laboratory Exercise 04

Pointers

Table of Contents

Data Types, Strings

Pre-Lab – 0 pts

There is no formal prelab but pre-starting the exercises would be a good idea.

Interactive Exercises – 40 pts

The purpose of this exercise is to exercise pointer usage, data structure access and memory leak detection. All exercises should use the following compile and valgrind styles:

```
gcc -O1 -Wall -std=c99 -g  lab_a.c -o lab_a

valgrind --tool=memcheck --leak-check=yes --track-origins=yes ./lab_a
```

The **return codes** from malloc/calloc/etc. functions **MUST BE CHECKED** and **safe string functions** must be used.

Data & Function Pointers

The purpose of this exercise is to using data and function pointers and the value of using NULL pointer assignments. You are provided a frame work file called **lab_a.c**, make all your changes in this file. Use the following sample output to create your printf() statements:

```
Program     0x40074d
Static data 0x6010a0
Ram data    0x6010e0
Heap data   0xe56010
Stack data  0x7ffff2f43340

Passing the 2X function a 2, I got 4
Passing the 3X function a 2, I got 6

Stack data values before free: 1 'Stack'
Heap data values before free: 2 'Pointer'
Heap data values after free:  <could be anything>
```

1) Examine the lab_a.c file and understand the contents.
2) Allocate space for: heapData_p
3) Print out the memory addresses of the following items using “%8p” (in order):
   main, staticData, ramData, heapData_p, stackData.
   Note: **Don’t compile with – Pedantic**
   What general conclusion can you draw from the data.
4) Call function “fun2x” using a function pointer passing the value “2” and then using the function “fun3x”. Refer to the class pointer lecture notes on function pointer usage.
5) Initialize the stack data variable “stackData” with the values “1” and "Stack" then print out the values as shown above.

6) Initialize the heap data variable “heapData_p” with the values “2” and "Pointer" then print out the values as shown above.
7) Free the “heapData_p” BUT DO NOT SET THE POINTER TO NULL.
8) Re-print the “heapData_p” data as shown above.
9) Run the resulting application lab_a and save the output to **exercise.txt**.
10) Run the valgrind command and append the output to exercise.txt. Are there any memory leaks or memory problems? Document your conclusion in **exercise.txt**
11) Set the heapData_p to NULL after the free() and rerun the application and valgrind. Append the output to exercise.txt. The program will now crash, why? Document your conclusion in **exercise.txt.**

Creating and destroying objects using pointers

The purpose of this exercise is to exercise, creating, use and destroy pass by pointer and pass by value data structures. You are provided a frame work file called **lab_b.c**, make all your changes in this file. Use the following sample output to create your printf() statements.

```
Print poly by reference:  0  2  4  6  8 10 12 14 16 18
Print poly by value:  0  2  4  6  8 10 12 14 16 18
Print poly by reference:  0  2  4  6  8 10 12 14 16 18
Print poly by value:  0  2  4  6  8 10 12 14 16 18
```

1) Examine the lab_b.c file and understand the contents.
2) Code the following functions incrementing by 2 as demonstrated in the above output sample: createPoly, destroyPoly, createPoly_p, destroyPoly_p, printPolyRef and printPolyVal.
3) Create a 10-digit polynomial variable and print the results using “stackPoly” and: createPoly, printPolyRef, printPolyVal and destroyPoly.
4) Create a 10-digit polynomial variable and print the results using “poly_p” and: createPoly, printPolyRef, printPolyVal and destroyPoly_p.
5) Run you code, verify it conforms to the expected results and append the output to exercises.txt
6) Run the given valgrind command and conform there are no memory leaks or access errors, append your valgrind output to **exercises.txt**.

Simple linked lists using pointer

The purpose of this exercise is to use wrapper objects and data structures to simulate a very simple linked list. You are provided a frame work file called **lab_c.c**, make all your changes in this file. Use the following sample output to create your printf() statements.

```
Printing 1 node linked list
  node 1 contains the string 'one'

Printing 2 node linked list
  node 1 contains the string 'one'
  node 2 contains the string 'two'

Clean up
```

1) Examine the lab_c.c file and understand the contents.
2) Write the function "printList" which will walk the linked list and print the contents as shown above. It must be written to traverse a linked list of arbitrary length.
3) Add code to main to allocate a single node with an index of "1" and string of "one".
4) Use your printList to print out the linked list.
5) Add code to main to allocate a second node with an index of "2" and string of "two".
6) Use your printList to print out the linked list.
7) Add code to clean up memory.
8) Run you code, verify it conforms to the expected results and append the output to exercises.txt
9) Run the given valgrind command and conform there are no memory leaks or access errors, append your valgrind output to **exercises.txt**.

Show the "**exercises.txt**" file to your TA.

Assignment – 60 pts

Objective

Implement a basic linked list abstract data type as a C module. Practice the use of pointers and memory allocation.

Background: For a review on linked lists study G. Semeraro "Chapter 4: Data Structure" and also N. Parlante "Linked List Basics" (posted in MyCourses)

Program Specification

1. Upload the file **files.tar** (available in MyCourses) to your working directory. The tarball contains the:

   Header files **LinkedLists.h & ClassErrors.h**

   C framework file: **LinkedLists.c**

   A test harness: **simpleTest.c**

   Some scripts: **build test mem**

   The expected results: **solution.txt**

2. Implement a ***linked list module*** using the interface specification in the file **LinkedLists.h** and **LinkedLists.C**.

   Program Behavior:

3. Use the provided **simpleTest.c** to help test your code.
   1. Build your code: ***./build***
   2. You can manually debug your code using: **gdb ./simpleTest**
   3. You can automatically test your code: ***./test***
   4. Memory leak detection is required for this assignment: **./mem**

Note: You may want to comment out some of the line in **test** and **mem** during early development. Your final submission must include the complete test set.

Analysis:

Write an **analysis.txt** summarizing your implementation. Create a tarball **lastName_hw5.tar** (lastName is your last name) with all relevant files and submit it.

Grading Criteria

1. (45 points) Correct implementation of basic Linked List Module, including memory leaks, error messages, etc
2. (15 points) Analysis of results concise and clear.

Notes

1. To learn more about doubly linked lists read chapter 4 of G. Semeraro's book (posted onMyCourses). If you use other reference sources list those in your **analysis.txt** file.

Student Name: _____

Laboratory Grading Sheet

Lab04 - Pointers

Component	Point Value	Points Earned	Comments and Signatures
Pre-Lab	**0**		
Interactive Exercises : Data and function pointers	**13**		
Interactive Exercises: Creating and destroying objects	**14**		
Interactive Exercises: Simple linked lists	**13**		
Total	**40**		

You must turn this signed sheet in at the end of lab to receive credit!