CSCI 1730 Breakout Lab 07

More Pointer Activities

Learning Outcomes

- Demonstrate knowledge of pointers and arrays by tracing through the output of source code containing pointers to 1D arrays.
- Demonstrate knowledge of simple data structures by writing out their memory maps and tracing through the output of source code.
- Use memory maps to understand the memory model of a C process.

Problem / Exercise

Pointers are an integral part of C and C++. While they may be the cause of many headaches, they allow for a great deal of control and flexibility. Because they are low-level constructs, they require a solid understanding of the memory model behind it all. Not every language has pointers, but what you learn about memory management will apply to most programming languages. This lab is structured differently from previous code writing labs. You'll be asked to observe some code and answer questions about it. Also, you will fill out memory maps (as needed) and tables.

Write up your answers to this lab assignment in a text file named lab07Answers.txt. On the first line of this file, write your first and last name as they appear on eLC. On the second line of this file, write your 811 or 810 number. Your lab07Answers.txt file must be neat and organized (otherwise, points might be deducted). You must number and label all of your answers in the same way as questions/problems are numbered and labeled in this assignment.

Const Pointers Activity

Before we get started working with const pointers, we should be clear about the terminology (which sometimes varies from textbook to textbook). For this course, we will define the following terms.

- nonconstant pointer to nonconstant data: the pointer CAN be modified to point other data of the appropriate type, and the data to which it points CAN be modified
- nonconstant pointer to constant data: the pointer CAN be modified to point to other data of the appropriate type, but the data to which it points CANNOT be modified
- constant pointer to nonconstant data: the pointer CANNOT be modified to point to other data of the appropriate type, but the data to which it points CAN be modified
- constant pointer to constant data: the pointer CANNOT be modified to point to other data of the appropriate type, and the data to which it points CANNOT be modified

Declaration Syntax	**Meaning**
int * ptr1;	ptr1 is a nonconstant pointer to a nonconstant int
const int * ptr2;	ptr2 is a nonconstant pointer to a constant int
int const * ptr3;	ptr3 is a nonconstant pointer to a constant int (same as ptr2)
int * const ptr4;	ptr4 is a constant pointer to a nonconstant int
const int * const ptr5;	ptr5 is a constant pointer to a constant int
int const * const ptr6;	ptr6 is a constant pointer to a constant int (same as ptr5)

1. Consider the following C code:

```
int main(void) {

    int x = 44;
    int y = 22;

    // part (1a)
    int * p1 = &x;
    *p1 = 11;

    // part (1b)
    const int * p2 = &x;
    *p2 = 11;
    p2 = &y;

    // part (1c)
    const int * const p3 = &x;
    *p3 = 11;
    p3 = &y;

    return 0;
} // main
```

(1a) Which statements of the code marked for part (1a) are valid, and which statements are invalid? If a statement is invalid, please explain why.

(1b) Which statements of the code marked for part (1b) are valid, and which statements are invalid? If a statement is invalid, please explain why.

(1c) Which statements of the code marked for part (1c) are valid, and which statements are invalid? If a statement is invalid, please explain why.

2. Consider the following C code:

```
int main(void) {

    int a[2] = {1, 2};
    int b[2] = {3, 4};

    // part (2a)
    int * p1 = a;
    const int * p2 = a;
    const int * const p3 = a;

    // part (2b)
    p1++;
    p2++;
    p3++;

    // part (2c)
    a = b;
    a++;
    (*a)++;

    return 0;
} // main
```

(2a) Which statements of the code marked for part (2a) are valid, and which statements are invalid? If a statement is invalid, please explain why.

(2b) Which statements of the code marked for part (2b) are valid, and which statements are invalid? If a statement is invalid, please explain why.

(2c) Which statements of the code marked for part (2c) are valid, and which statements are invalid? If a statement is invalid, please explain why.

3. Consider the following C code:

```
int a[2] = {1, 2};

const int foo() {
    return 2;
} // foo

const int * bar() {
    return a;
} // bar

int * const baz() {
    return a;
} // baz

int main(void) {

    // part (3a)
    int x = foo();
    x++;
    foo()++;

    // part (3b)
    const int * p1 = bar();
    p1++;
    (*p1)++;
    bar()++;
    (*bar())++;

    // part (3c)
    const int * p2 = baz();
    p2++;
    (*p2)++;
    baz()++;
    (*baz())++;

    return 0;
} // main
```

(3a) Which statements of the code marked for part (3a) are valid, and which statements are invalid? If a statement is invalid, please explain why.

(3b) Which statements of the code marked for part (3b) are valid, and which statements are invalid? If a statement is invalid, please explain why.

(3c) Which statements of the code marked for part (3c) are valid, and which statements are invalid? If a statement is invalid, please explain why.

Functions, Pointers, and Tricky Declarations Activity

In this part, you will get to play and practice with pointers as function parameters. Also, you will become familiar with complicated mixed declarations of pointers, arrays, and functions.

(4a) What two values are printed out by the following C code? Why?

```
#include <stdio.h>

void divBy2(int n) {
    n = n / 2;
} // divBy2

void divBy2Again(int * np) {
    *np = *np / 2;
} // divBy2Again

int main(void) {
    int x = 44;

    divBy2(x);
    printf("%d\n", x);

    divBy2Again(&x);
    printf("%d\n", x);

    return 0;
} // main
```

(4b) In the code presented in the previous question, why is `&x` used for the function parameter for the call to divBy2Again?

(4c) Complete the following table by providing plain English meanings to the corresponding valid C declarations. Use the right-left rule as discussed during lecture class to complete this table. In your lab07Answers.txt file, create and correctly fill out the table below by using the same table headers and the same order of declarations as shown in the table below.

Declaration	**Meaning**
`int x;`	x is an int
`int * x;`	x is a pointer to an int
`char ** x;`	
`int * x [5];`	
`int (* x) [5];`	x is a pointer to an array of 5 ints
`int (* x [5]) [5];`	
`int * (* x [5]) [5];`	x is an array of 5 pointers to arrays of 5 pointers to ints
`int x();`	
`int x(int);`	
`int * x();`	
`int * x(int *);`	
`int (* x)();`	
`int ** (* x)(int **);`	

Submission

Submit your file, lab07Answers.txt, before the due date and due time stated on eLC. Submit only this file on eLC for this assignment. No README.txt file is required for this lab assignment.

Breakout Lab Attendance: 3 points

Breakout lab attendance is required for this assignment, and it is worth three points. Students must physically attend the entire period of the breakout lab section that they registered for during the week of this assignment to earn these three points for attendance. If you complete the assignment before the end of your breakout lab period during the week of this assignment, then you are still required to attend the entire breakout lab period to earn attendance points, and you may use this time for independent study for the next exam in this course.

Grading: 7 points

Students are expected to do all activities, follow all instructions, and answer all questions/problems. We will choose a subset of things to grade for this lab (you will not be told which parts we are grading since you must complete all parts). Furthermore, neatness and organization is expected (otherwise, we may deduct points). The following grading will be used.

Correct answers with correct explanations to a subset of things we are grading	7 points
Not submitting to the correct Assignment on eLC	-1 point
Late penalty for submitting 0 hours to 24 hours late	-2 points
Not submitting this assignment before its late period expires	-7 points
Penalty for not following instructions (disorganization, etc.)	Penalty decided by grader