Assg 11: Multiple Indirection of Pointers

CSci 515 Spring 2015

2015-04-14

Dates:

Due: Tuesday April 21, by Midnight

Objectives

- Learn about pointers to pointers to pointers (multiple dereferencing)
- Practice more with using structs
- Learn about dereferencing struct members that are pointed to by a pointer variable.
- Learn about chaining memory references, and multiple dereferences to manipulate data.

Description

In this assignment we will focus on some practice examples where we have multiple chains of pointer references, that we want to dereference simultaneously in order to get to some data to manipulate it. This assignment is broken up into two parts. In the first part we will simply create a chain of pointers, and practice dereferencing them. In the second part, we will create a structure that has a self-referential pointer in the definition, and use this self-referential pointer to create a circular chain of structure references. You don't have to write any functions for your assignment for this week, so all of the code and tasks you are asked to perform next should simply be done in your `main()` function in the program that you submit. Also, I encourage you to create diagrams of memory and the pointers/structures you are creating while you do the two parts of this exercise. When learning pointers

and dereferencing memory, it can be useful to diagram out the relationships you create visually, so you can better follow along with what you need to do with your code.

Part 1: Pointers to Pointers (to Pointers)

Perform the following tasks in the `main()` function of your program.

1. Create a regular floating point variable named `f`. Create a pointer to a float called `fPtr`. Create a pointer to a pointer to a float called `fPtrPtr` (two levels of indirection). And finally create a pointer to a pointer to a pointer to a float called `fPtrPtrPtr` (three levels of indirection).

2. Initialize the 4 variables you created in Step 1. Initialize your `f` variable with the value 42.42. Then initialize the `fPtr` pointer variable to point to `f`. Likewise initialize your `fPtrPtr` variable to point to `fPtr`, and finally for the third level of indirection, `fPtrPtrPtr` should be initialized to point to `fPtrPtr`.

3. Output the value you stored in f by directly reading the value from f using iostream output to cout. Then access the value in f and display it using 1, 2 and 3 levels of dereferencing using the `fPtr`, `fPtrPtr` and `fPtrPtrPtr` pointer variables respectively. You must use the dereference operator correctly for this step. See the example output below for what you should display on standard output.

Part 2: Self-referential Pointers in Structures

Here is a structure that contains a self-referential pointer inside of the structure definition. Put this definition of the `IntegerItem` user defined type at the top of your program before you `main()` function.

```
// A structure to hold an Integer
// and a pointer to some other item.
struct IntegerItem
{
  int value;
  IntegerItem* nextPtr;
};
```

Perform the following tasks for part 2 of the assignment in your `main()` function:

1. Create 4 instances/variables of type `IntegerItem`. Name your instances `a`, `b`, `c` and `d`.

2. Initialize the value member of your 4 instances to the values `5`, `7`, `10` and `15` for `a`, `b`, `c` and `d` respectively.

3. Initialize the `nextPtr` members of each of your instances. Notice that `nextPtr` is a pointer to an `IntegerItem` type. Assign the `nextPtr` member of your `a` item to point to `b`. Then assign `b`'s `nextPtr` to point to `c`. `c` should point to `d`. And finally, create a circular chain of references by causing `d`'s `nextPtr` member to point to `a`.

4. Create two pointer variables that will point to `IntegerItem`, and name these variables `aPtr` and `cPtr` respectively. Initialize `aPtr` so that it is pointing to (has the address) of item `a`. Likewise, initialize `cPtr` so that it is pointing to item `c`.

5. Using `aPtr` and the `->` member dereference operator, access and display the value of `a` using an iostream operation to cout. The value in `a` should be `5` if you performed the initializations in step 2 correctly.

6. Again using `aPtr` and the `->` member dereference operator, access the value in `c` by chaining together two dereferences through the `nextPtr` member pointer variables. Display the value you access to cout. You must do this access starting from `aPtr` and using only the `->` dereference operator.

7. Using `cPtr` and the `->` member dereference operator, follow the links for 3 hops and display the value you find. Again you must start at `cPtr` and only use `->` dereference operations.

8. Using `aPtr` and the `->` member dereference operator, follow the links in your chain for 4 hops. If you did this correctly, you should of course end up back at `a` where you started.

Here is an example output from running a correct implementation of this assignment for both parts 1 and 2:

```
f = 42.42
*fPtr = 42.42
**fPtrPtr = 42.42
***fPtrPtrPtr = 42.42
```

```
The value of a, dereferenced using aPtr and -> operator: 5
Two hops on chain away from a: 10
Three hops on chain away from c: 7
Four hops on chain away from a: 5
```

HINT: You may find it helpful to draw diagrams of the pointers and their references by hand, especially for the Part 2 exercise using self-referential structure pointers.

NOTE: Now that our programs have more functions than just the `main()` function, the use of the function headers becomes meaningful and required. Make sure that all of your functions have function headers preceding them that document the purpose of the functions, and the input parameters and return value of the function.

Assignment Submission

An eCollege dropbox has been created for this assignment. You should upload your version of the assignment to the eCollege dropbox named `Assg 11 Multiple Indirection` created for this submission. Work submitted by the due date will be considered for evaluation.

Requirements and Grading Rubrics

Program Execution, Output and Functional Requirements

1. Your program must compile, run and produce some sort of output to be graded. 0 if not satisfied.
2. 40+ pts. For the correct coding of the Part 1 tasks.
3. 30+ pts. For correctly implementing creation and initialization of the Part 2 structures.
4. 30+ pts. For correctly dereferencing the structure members as asked for.

Program Style

Your programs must conform to the style and formatting guidelines given for this course. The following is a list of the guidelines that are required for the assignment to be submitted this week.

1. The file header for the file with your name and program information and the function header for your main function must be present, and filled out correctly.

2. A function header must be present for all functions you define. You must document the purpose, input parameters and return values of all functions. Your function headers must be formatted exactly as shown in the style guidelines for the class.

3. You must indent your code correctly and have no embedded tabs in your source code. (Don't forget about the Visual Studio Format Selection command).

4. You must not have any statements that are hacks in order to keep your terminal from closing when your program exits (e.g. no calls to system()).

5. You must have a single space before and after each binary operator.

6. You must have a single blank line after the end of your declaration of variables at the top of a function, before the first code statement.

7. You must have a single blank space after , and ; operators used as a separator in lists of variables, parameters or other control structures.

8. You must have opening `{` and closing `}` for control statement blocks on their own line, indented correctly for the level of the control statement block.

9. All control statement blocks (if, for, while, etc.) must have `{ }` enclosing them, even when they are not strictly necessary (when there is only 1 statement in the block).

10. You should attempt to use meaningful variable and function names in your program, for program clarity. Of course, when required, you must name functions, parameters and variables as specified in the assignments. Variable and function names must conform to correct `camelCaseNameingConvention` .

11. Put the `*` for pointer variable declarations next to the type declaration, with no space between the type and the `*`. Also please follow the convention of using `Ptr` at the end of names for pointer variables.

Failure to conform to any of these formatting and programming practice guidelines for this assignment will result in at least 1/3 of the points (33) for the assignment being removed for each guideline that is not followed (up to 3 before getting a 0 for the assignment). Failure to follow other class/textbook programming guidelines may result in a loss of points, especially for those programming practices given in our Deitel textbook that have been in our required reading so far.