Pointer Lab: A review of C and pointers

Introduction

This assignment serves as a review of C and pointers. You will be implementing all the functions in pointer.c. The comments provide details on what each function should do. Any relevant structure definitions, constants, and inline functions are provided for you in pointer.h. **You should NOT make any changes in any file besides pointer.c.**

The assignment consists of 4 parts that build on top of each other:

1. Object functions (20 pts)

   These functions serve to introduce the concept of polymorphism and object-oriented programming. You will be programming two types of objects that represent inventory in a store. The StaticPriceObject represents an item with a fixed price. The DynamicPriceObject represents an item that varies the price based on the remaining quantity of items left. Both objects are based on a base Object that corresponds to the base class in an object-oriented language.

   Polymorphism is the concept that a pointer to a base object can refer to objects of different sub-types. For example, an Object pointer can refer to a StaticPriceObject or a DynamicPriceObject. Object-oriented languages handle this automatically, but in a non-object-oriented language like C, we must emulate this behavior ourselves. This is done by having the Object struct located at the beginning of the StaticPriceObject and DynamicPriceObject structs (as shown in pointer.h). Thus, casting a StaticPriceObject pointer to an Object pointer is valid since the beginning of the StaticPriceObject is an Object.

   To support virtual functions (i.e., Object functions that can either refer to StaticPriceObject functions or DynamicPriceObject functions), we use what's known as a virtual function pointer table that contains function pointers that refer to the correct functions. For example, the StaticPriceObject will have function pointers that point to StaticPriceObject functions, and the DynamicPriceObject will have function pointers that point to DynamicPriceObject functions. Thus, an Object can call the appropriate function by using the virtual function pointer table. For example, the provided object_price function uses the table to call the appropriate price function for the Object.

   Your role in this part of the assignment is to 1) understand how the objects are manipulated, and 2) implement a set of functions that emulate object-oriented programming in C. The first two functions illustrate how to compare generic Objects, which may either be StaticPriceObjects, DynamicPriceObjects, or some other Object. The next two functions illustrate the initialization of each object. The following two functions implement the pricing function for each object, and the last two functions implement the bulk pricing function for each object.
2. Iterator functions (30 pts)

   An iterator is a structure that refers to the current position in some data structure. It is used to iterate through a data structure without needing to know the underlying details of the data structure. In other words, it encapsulates the process of iterating through a data structure.

   In this part of the assignment, you will be implementing a linked list iterator for a singly-linked list of Objects. The linked list structure and iterator structure can be found in pointer.h. Each linked list node will have a pointer to an Object as well as a pointer to the next node in the linked list. The head of the list is simply a piece of data that points to the first node in the list. Functions will typically use a pointer to the head of the list (i.e., a pointer to that piece of data). The iterator will contain a pointer to the current node as well as a pointer to the previous node's

next pointer to support removals and insertions. The first four functions allow you to iterate through the list and get the objects in the list. The following function provides the ability to remove nodes from the list, and the last two functions provide the ability to insert nodes into the list.

3. List functions (20 pts)

   The list functions illustrate how to utilize the iterator to go through the list. You should be calling the functions you implemented in the previous part rather than reimplementing list iteration. The memory for the iterator structure can be allocated as a local variable in your functions. The first function is a simple list iteration, the second function illustrates how to use function pointers, and the last function returns the length of the list. You are welcome to create a helper function and use the foreach function in implementing the length function. The foreach function uses a union datatype, which can be copied/assigned/passed by value without issue.

4. Mergesort functions (30 pts)

   These last three functions implement the mergesort algorithm. They provide more experience with using iterators, function pointers, working with pointers, and recursion. Mergesort works by splitting the list into two halves, sorting each half recursively, and then merging the sorted halves.

Programming rules

You are not allowed to use any libraries besides the math.h library already included in pointer.h. Do not include any other libraries/files in pointer.c. If you really think you need some library function, please contact the course instructor to determine eligibility. You are allowed to use any of the inline functions, constants, etc. within the pointer.h file.

Evaluation and testing your code

You will receive zero points if:

- You violate the academic integrity policy (sanctions can be greater than just a 0 for the assignment)
- You don't show your partial work by periodically adding, committing, and pushing your code to GitHub
- You break any of the programming rules
- Your code does not compile/build
- Your code crashes the grading script

Your code will be evaluated for correctness. We have provided some tests, though we reserve the right to add additional tests during the final grading, so you are responsible for ensuring your code is correct. In terms of a grade point breakdown, we will assign:

- (20 pts) Object functions
- (30 pts) Iterator functions
- (20 pts) List functions
- (30 pts) Mergesort functions

To run the grading script, simply run the following command in the assignment folder:

```
make test
```

Handin

We will be using GitHub for managing submissions, and **you must show your partial work by periodically adding, committing, and pushing your code to GitHub.** This helps us see your code if you ask any questions on Canvas (please include your GitHub username) and also helps deter academic integrity violations.

Additionally, please input the desired commit number that you would like us to grade in Canvas. You can get the commit number from github.com. In your repository, click on the commits link to the right above your files. Find the commit from the appropriate day/time that you want graded. Click on the clipboard icon to copy the commit number. Note that this is a much longer number than the displayed number. Paste your very long commit number and only your commit number in this assignment submission textbox.

Hints

To compile your code in debug mode (to make it easier to debug with gdb), you can simply run:

```
make debug
```

You can launch gdb on your program by running:

```
gdb pointer
```

Within gdb, you can type:

```
r
```

to run all of the tests, or you can type:

```
r test_name
```

with test_name replaced by whatever test you want to specifically run. You can find the tests in test.c. When debugging test failures, you should read the relevant test code in test.c to understand the failure.