CSE-165 Lab 04

Points: 100

Write a separate file for each of the following tasks.

Zip all your files together and submit the zip file to CatCourses.

---

1. **Stash (10 points):** Consider the file Stash.h. This file contains the Stash data structure from Chapter 4 of the textbook. Your task is to write a program that declares an instance of Stash, fills it up with numbers (doubles), and then prints out the numbers. You must use the Stash member functions to complete this exercise.

   The first line of input contains an integer N. This is followed by N lines, each containing a double value. Read in these values and store them in your Stash object. Afterwards, print out the values.

   After the double values have been printed out, use the appropriate member function of Stash to free up the memory your object occupied.

---

2. **Stack (10 points):** Consider the file Stack.h. This file contains the Stack data structure from Chapter 4 of the textbook. Your task is to write a program that declares an instance of Stack, fills it up with numbers (doubles), and then prints out the numbers. You must use the Stack member functions to complete this exercise.

   The first line of input contains an integer N. This is followed by N lines, each containing a double value. Read in these values and store them in your Stack object. Afterwards, print out the values. (Note, due to the nature of the stack, you will essentially have to print them in reverse order.)

   When the stack is finally empty, call the cleanup() function.

---

3. **LinkedList (20 points):** You are now going to create a LinkedList class that will work very similarly to the Stack class used in the previous exercise.

   Write new methods as follows:

   - add(LinkedList::Link* l, int n): will insert in the linked list, after link l, a chain of n new links. Each link will store an integer that will be set to receive, in order, a value starting from 0 until n-1. In the last link of the list, always set the next pointer to be NULL in order to mark the end of the list.

   - print(): this method moves through the entire list printing out each integer value stored in the links.

- cleanup(): updated cleanup function that will automatically delete all links in the linked list, including the data stored on each link. (Ignore the problems related to deleting void* for now.)

Your linked list struct should be named LinkedList and should be saved in a header file named LinkedList.h. The file linkedLists.cpp will be used to evaluate your struct.

Note that you will have to provide a constructor for the creation of the LinkedList in linkedLists.cpp to work. The given integer can be used to construct an initial node (otherwise your add() will have to handle the case when it receives a null pointer as its first parameter).

Expected Output: 47 0 1 0 1 2 3 4 2 3 4

---

4. **Pointers to Functions (20 points):** One of the problems of our Stack and LinkedList classes is that if we make them handle generic types with void* pointers, they will not know how to delete the elements we insert in them. We will see later how to solve that in a better way, but for now let's practice using pointers to functions.

   Extend the book's Stack struct with one more member variable that will hold a pointer to the following function prototype:

   void deletecb (void* pt);

   You will then add a member function to set this pointer:

   void Stack::setDeleteCallback (void (*delcb) (void* pt));

   After you do this, then add the corresponding code in the cleanup method to traverse all elements of the stack, deleting the links and calling the delete callback once for each stored void pointer. The user will be responsible for providing the delete callback and implementing it with the correct delete call after converting the void pointer argument to its correct type.

   You may download the Stack.h file, make the modifications described above and submit it. Your code will be tested using voidPointers.cpp.

---

5. **Address Entry (20 points):** Create a struct named Entry that stores an entry of an address book. It should have fields for first name, last name, and email address. Provide the appropriate getter and setter functions for each one of the fields. In addition, provide a function called print. This function should print out the three fields of information in the struct.

Your code should go into a file named Entry.h with all the member functions implemented inline (in the header file). Your code will be tested using the file address_book_entry.cpp.

---

6. **Address Book (20 points):** Create a struct named AddressBook that stores multiple Entry structs (that you created in the previous exercise). Your code must be saved in a file named AddressBook.h and should reuse your code from Entry.h.

   Member functions for AddressBook should be able to add an entry and print all the entries in the address book. Your code will be tested with the file addressBook.cpp, which reads in several entries from standard input, stores them in an AddressBook instance and prints them all out.