Data Structures and Algorithms

Lab 3 (5% of final grade)

(100 points total)

All code must be your own and follow the guidelines in lecture notes and textbook. Use proper coding styles including comments and indentation. No code from Internet sources is allowed in this assignment. All code must be in C++ programming language.

1. Implement a simple text editor, which stores a string of characters using the List ADT, together with a cursor object that highlights the position of some character in the string (or possibly the position before the first character).
   Your editor should support the following operations and redisplay the current text (that is, the list) after performing any one of them.

   • left: Move cursor left one character (or nothing if at the beginning)

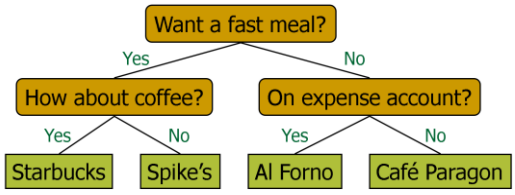
   • right: Move cursor right one character (or do nothing if at the end)

   • delete: Delete the character to the right of the cursor (or do nothing if at the end)

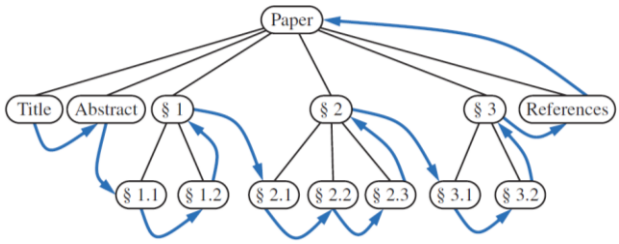
   • insert *c*: Insert the character *c* just after the cursor

   Test your implementation with a realistic input. Submit a writeup of your design, screenshots of the results, and code files in a zip file.

2. An important class of binary trees, called decision trees, arises in contexts where we wish to represent a number of different outcomes that can result from answering a series of yes-or-no questions. Implement the Dining Decision algorithm using representations of Binary Trees as listed below. Load the binary tree from an input file. Test using the example below. Submit screenshots of the results, code files, and the input file corresponding to the example below, in a zip file.
   a. linked-based representation of binary tree
   b. array-based representation of binary tree



3. Implement the Postorder traversal using linked-based representation of General Trees. Load the general tree from an input file. Test using the example below. Submit screenshots of the results and code files, and the input file corresponding to the example below, in a zip file.



4. One of the main applications of priority queues is in operating systems—for *scheduling jobs* on a CPU. Implement a scheduler for simulated CPU jobs. Your program should run in a loop, each iteration of which corresponds to a *time slice* for the CPU. Each job is assigned a priority, which is an integer between -20 (highest priority) and 19 (lowest priority), inclusive. From among all jobs waiting to be processed in a time slice, the CPU must work on the job with highest priority. In this simulation, each job will also come with a *length* value, which is an integer between 1 and 100, inclusive, indicating the number of time slices that are needed to process this job. For simplicity, you may assume jobs cannot be interrupted—once it is scheduled on the CPU, a job runs for a number of time slices equal to its length. Your simulator must output the name of the job running on the CPU in each time slice and must process a sequence of commands, one per time slice, each of which is of the form "add job *name* with length *n* and priority *p*" or "no new job this slice."

   Test your implementation with a random input that adheres to the criteria above. Submit a writeup of your design, screenshots of the results, and code files in a zip file.

5. Implement the array-based HeapSort in C++ and compare its results with the sorting algorithms from Project. Submit a writeup of your findings (including at least one comparison graph) and code files in a zip file.