**CS233 Data Structures and Systems Programming**

Homework 1 - Spring 2017 - 50 Points

DUE @ TAPS on Lesson 10, M-day: Tue, 31 Jan, T-day: Wed 1 Feb.

***Documentation Policy:***

* You must document all help received from any source other than your instructor.
* The documentation statement must explicitly describe WHAT assistance was provided. WHERE on the homework the assistance was provided, and WHO provided the assistance.
* If no help was received on this assignment, the documentation statement must state “NONE.”
* If you checked answers with anyone, you must document with whom on which problems. You must document whether or not you made any changes, and if you did make changes you must document the problems you changed and the reasons why.
* Vague documentation statements must be corrected before the assignment will be graded, and will result in a 5% deduction on the assignment.

***Help policy for Homework #1:***

**AUTHORIZED RESOURCES: Any, except another cadet’s work or programs.**

**NOTE:**

* Never copy another person’s work and submit it as your own.
* Do not jointly complete this assignment.
* You must document all help received from sources other than your instructor.
* **DFCS will recommend a course grade of F for any cadet who egregiously violates this Help Policy or contributes to a violation by others.**

**Instructions:** Type your answers into this document and add the document to the Homeworks folder of your *source code repository*. Make sure you *commit* and *push* your repository before the due date.

**Documentation Statement:** None

1. (10 pts, 2 pts each) Study the "**Data Structures Summary**" document that is at the top of the course web site. For each of the basic types of data structures, give at least two problems or situations where the data structure would be the appropriate data organization for that problem. (You can't use any of the examples that are already given in the document.)
   1. Set
      1. Dating websites, finding commonalities between people
      2. Creating an email mailing list, grouping and separating
   2. Map
      1. Data encryption, linking key/signature
      2. Team roster, linking number to player, position
   3. List
      1. Representing the chain of command that my Form 34 must pass through to be processed
      2. Recording race results, can sort by age and place, etc.
   4. Tree
      1. Showing how rumors spread from person to person in a high school, or USAFA
      2. Recall roster for Exercise Swift Falcon, multiple element leaders linked to flight commander
   5. Graph
      1. Create a mind map to determine a friendly or enemy CoG for MSS, CoG is item with most connections
      2. Representing how pieces for Ikea furniture are connected
2. (10 pts, 2 pts each) Read the introductory section of these two Wikipedia web pages, [compiler](https://en.wikipedia.org/wiki/Compiler) and [linker](https://en.wikipedia.org/wiki/Linker_(computing)), and then answer the following questions.
   1. What is a *source code* *file*?  
      A source code file is the file written by a programmer, in the source language.
   2. What does a compiler do?  
      A compiler transforms the source file into the target language.
   3. What is an *object file*?  
      An object file is a file created by the compiler containing object code, a form of machine code.
   4. What does a linker do?  
       A linker takes object files from the compiler and combines them to make an executable file.
   5. What is an executable file?  
      An executable file is a file, usually in machine code, that gives instructions to the computer to execute the program.
3. (8 pts, ½ pt each) Implement a C program on your computer that prints out the number of bytes used to store each of the following C data types. Then fill in the chart below. You answers must be in bytes.

|  |  |
| --- | --- |
| **Data type** | **# of bytes** |
| Int | 4 |
| unsigned int | 4 |
| float | 4 |
| double | 8 |
| short | 2 |
| Char | 1 |
| unsigned char | 1 |
| Long | 8 |
| long long | 8 |
| char alpha[20] | 20 |
| int beta[30] | 120 |
| struct gamma {  int field1;  float field2;  char field3[10];  }; | 20 |
| int \* | 8 |
| float \* | 8 |
| double \* | 8 |
| char \* | 8 |
| long \* | 8 |
| struct gamma \* | 8 |

1. (6 pts, 1 pt each) Given the following C code, in the comments to the right of each assignment statement, give the value of the variable after the assignment. For arrays, show the entire array.

**int** alpha[6] = {10, 20, 30, 40};  
**char** courseName[] = **"CS223 Data Structures and System Programming"**;  
**char** aCharacter;  
  
alpha[0] = alpha[1] + 2; *// alpha = {22, 20, 30, 40, 0, 0, 0}*aCharacter = courseName[3]; *// aCharacter = 2*courseName[5] = **'\_'**; *// courseName =* **"CS223\_Data Structures and System Programming"**strcpy(courseName + 6, **"DataStructures"**); *// courseName = “CS223\_DataStructures”*alpha[10] = 2; *// alpha = {22, 20, 30, 40, 0, 0, 0, 0, 838860800, 0, 10}*\*(alpha + 3) = -3; *// alpha = {22, 20, 30, -3, 0, 0, 0, 0, 838860800, 9, 11}*

1. Use *algorithm analysis* to create an equation that describes the worst-case run-time behavior expected from the following code. The equation must be a function of the number of elements stored in the data structure, which we refer to using the variable "n".
   1. (4 pts) Assume that your data structure is an *array list* that is defined using the following C declarations:

**typedef int** Element\_type;  
  
**typedef struct** ArrayList {  
 Element\_type \*array;  
 **int** arraySize;  
 **int** numberElements;  
} ArrayList;

What is the big-O run-time complexity of the following function that searches a list for a particular value and returns its position in the list:

**int** arrayListFindElement(ArrayList \*list, ElementType value) {  
 **for** (**int** j=0; j < list->numberElements; j++) {  
 **if** (list->array[j] == value) {  
 **return** j;  
 }  
 }  
 *// The value was not found* **return** -1;  
}

Answer: O(n)

* 1. (4 pts) Assume that your data structure is the same as question (a), i.e., an ArrayList.

What is the big-O run-time complexity of the following function which returns a specified element from the list:

ElementType arrayListGetElement(ArrayList \*list, **int** index) {  
 **if** (index >= 0 && index < list->numberElements) {  
 **return** list->array[index];  
 } **else** {  
 printf(**"arrayListGetElement failed to get element %d."**, index);  
 printf(**"The list only contains %d elements."**, list->numberElements);

exit(1);  
 }  
}

Answer: O(n)

* 1. (4 pts) Assume that your data structure is a *linked list* that is defined using the following C declarations:

*// Define the data type for the elements that will be stored in this linked list.  
// This definition simply makes it easier to change the type of data the list stores.***typedef int** ElementType;  
  
  
  
*// Define on node of the linked list***typedef struct** node {  
 ElementType data;  
 **struct** node \*next;  
} Node;  
  
**typedef** Node \* NodePtr;  
  
*// Define the meta-data that stores the linked list.***struct** linkedList {  
 NodePtr first;  
 NodePtr last;  
 **int** numberElements;  
};  
  
*// A "linked list" is a pointer to a linkedList structure***typedef struct** linkedList \*LinkedList;

What is the big-O run-time complexity of the following function which returns a specified element from the list:

ElementType linkedListGetElement(LinkedList list, **int** index) {  
 *// Start at the beginning and get to the correct node* **int** counter = 0;  
 NodePtr nodePointer = list->first;  
 **while** (nodePointer != NULL && counter < index) {  
 nodePointer = nodePointer->next;  
 counter++;  
 }  
  
 **if** (nodePointer != NULL) {  
 **return** nodePointer->data;  
 } **else** {  
 printf(**"linkedListGetElement failed to get element %d."**, index);  
 printf(**"The list only contains %d elements."**, list->numberElements);  
 exit(1);  
 }  
}

Answer: O(n)

* 1. (4 pts) Assume that your data structure is a *linked list* as defined for problem (c).

What is the big-O run-time complexity of this function that appends an element to the list?

**void** linkedListAppend(LinkedList list, ElementType \*element) {  
 *// Create a new node for this new element* NodePtr newNode = (NodePtr) malloc(**sizeof**(Node));  
 newNode->data = \*element;  
 newNode->next = NULL;  
  
 *// Link the new node into the linked list. There are two cases:*

*// adding to an empty list, or to a list that has one or more elements.* **if** (list->last == NULL) {  
 *// This is the first node in the linked list* list->first = newNode;  
 list->last = newNode;  
 } **else** {  
 *// The last node, which is the current node, points to this new node* list->last->next = newNode;  
 list->last = newNode;  
 }  
 list->numberElements++;  
}

Answer: O(1)