Fall 2014 ECE 198KL Final Exam

December 18 2014

- **This is a closed book exam except for 3 sheets of notes.**
- **You may not use a calculator.**
- **Absolutely no interaction between students is allowed.**
- **Don’t panic!**

Problem 1 [40 points]: Linked List

Part A [20 points] Detect Loop

Given a pointer to the head of a linked list, complete the function `DetectLoop` that returns 1 if the linked list has a loop and 0 otherwise. Do not modify the linked list or create a copy of the linked list. You may assume that the input to your function is a valid linked list that might possibly have a loop.

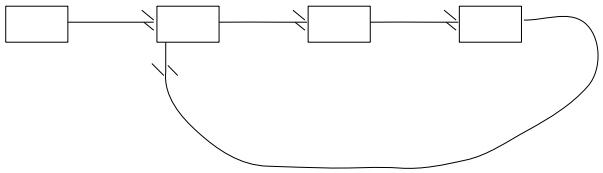
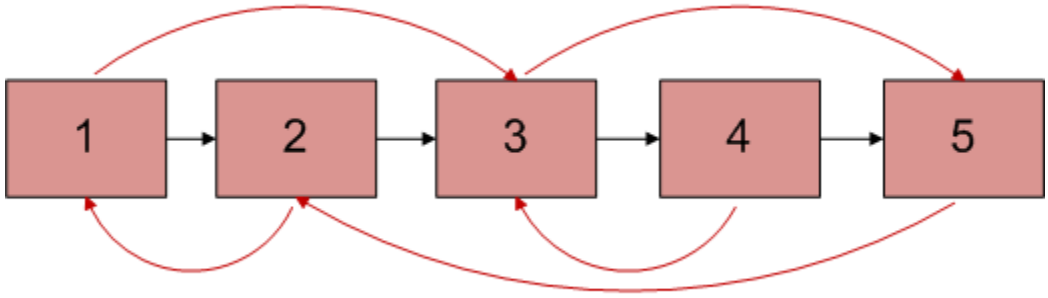


Figure 1: A linked list with a loop.

Part B [20 points] Copy List

Write a function `CopyList` that accepts a pointer to a linked list and returns a pointer to the copied list. The linked list you will be working with is a linked list with two pointers in each node. One is the regular next pointer, the other is an arbitrary pointer that points to some node in the linked list or is NULL. When you copy the linked list the values of the next pointer and arbitrary pointer in each node should be accurate too. You may assume that the input to your function is a valid linked list as



described.

Figure 2: Next pointers in black and arbit pointers in red.

Instructions:

- Write your code in linkedList.c

- Code to test your code and setup the linked lists has been given in main.c.
- Read the comments and code in main.c and linkedList.h to understand the implementation.
- To compile your code:- gcc main.c linkedList.c -o linkedList

Problem 2 [30 points]: Arrays

For this problem, you will develop an algorithm that can find a subarray within a 1D array of integers with the largest positive sum. For example, consider the following array with 10 elements:

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	x[9]
5	-10	11	5	6	-21	1	15	120	-32

In this example, the subarray starting at x[2] thru x[8] has the largest positive sum, which is 137.

The most straightforward way is to compare the sums of each possible subarray contained in the array, but this is a very inefficient algorithm for this problem. **If you use this algorithm, you will automatically lose 10 points on this problem**. Instead, a more efficient algorithm for this problem is Kadane's Algorithm, which is as such:

```
max_subarray = 0;
current_subarray = 0;

Loop for each element of the array, a[i] {
   current_subarray = current_subarray + a[i];

   if (current_subarray < 0)
         current_subarray = 0;

  if (max_subarray < current_subarray)
         max_subarray = current_subarray;
}
return max_subarray;
```

The idea of this algorithm is to search all positive-sum subarrays of the array (current_subarray is used for this), while keeping track of maximum sum subarray among all positive ones (max_subarray is used for this). The algorithm above only finds the maximum subarray sum, and doesn't identify the beginning or the end of

the subarray with the maximum positive sum. Your task will be to implement the function find_maxSum, which finds the starting point, ending point, and the sum of the subarray with the maximum positive sum. The main function is provided and you do not need to change it.

Note: you can assume that the array provided contains one positive value, and has a unique positive maximum sum.

To compile:

```
gcc -Wall -g pro_1.c -o pro_1.o
```

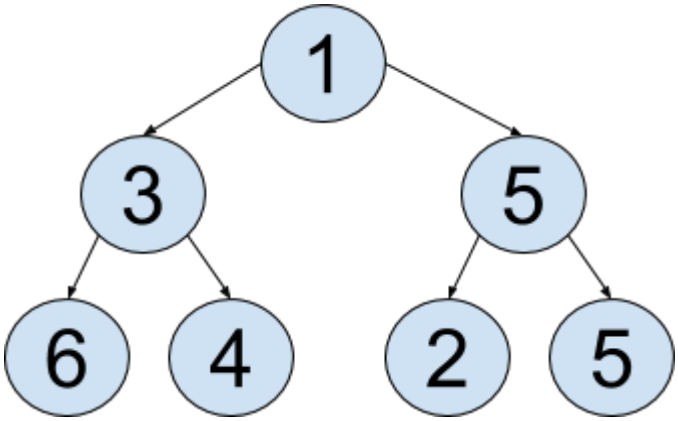
To run the code with input file:

```
./pro_1.o input_array
```

The input_array is provided. The first number in the file is the size of the array.

Problem 3 [30 Points]: Binary Trees

For this question you must print out all paths within a binary tree that add up to a given sum. Here, the tree will be a binary tree data structure, but the nodes will not be organized in any particular manner (for example, the left sub-tree need not contain nodes that are less than the root). A path is a sequence of nodes, starting at the root node and ending at a leaf node. For example given the following binary tree:



If the sum is 8, your code should print the two paths that add to 8
1 3 4
1 5 2

Your code must be written in the sums.c file. We will only test your code with valid trees and valid inputs. And a valid tree will have no more than 100 levels.

To compile your code use the following commands:

make clean
make

To run your code, where filename is the tree input file (test1.txt, test2.txt or one you wrote yourself):

./sums filename

To test your code we provide two test trees in the tests/test1.txt and tests/test2.txt files. Your code must be functional for all valid trees not just

these test cases. tests/out1.txt contains the solution output for test1 and sum = 8, tests/out2.txt contains the solution output for test2 and sum = 8.

To write your own tests you can create your own file with the list of numbers you want in the tree (white space doesn't matter). The tree is filled a level at a time, from left to right. Therefore only complete trees are valid (every level, except possibly the last, is completely filled, and all nodes are as far left as possible). The above example tree would be expressed as follows in an input file: 1 3 5 6 4 2 5

Hint: Think about traversing the tree and somehow keeping track of the current path and current sum. The challenge to this problem is printing out a valid path. For this, we have provided the functions in vector.c and vector.h, for you to use. A vector is like a stack, in that you can push and pop elements from the "back" of the vector ("back" is the top of the stack), but also like an array you can access any element in the vector using an index.