ECE 220: Computer Systems & Programming

Spring 2020 – Final Exam

May 13, 2020

1. This is an open book exam; you may use any resources available to you.
2. You cannot ask another person for help, provide help to a fellow student, or post/share your answers online.
3. Your programs will be graded on functionalities only. If your code does not compile, you will receive a zero for that question.

Exam Workflow:

1. Fetch exam files from course GitHub (same as if fetching a new MP)
2. Complete the exam and **commit changes to GitHub** (you should fully test your code for each question)
3. Use the following command in the 'MT2' folder to create a zip file: **zip -r final.zip P1 P2 P3 P4**
4. **Upload final.zip** to **Gradescope**

Disclaimer: you can modify the provided test case in each question to fully test your code. Additional test cases will be used during grading.

Problem 1 (30 points): C to LC-3 Conversion

This question tests your understanding of the run-time stack in LC-3. Translate the leaf_count() function from C to LC-3 at label **LEAFCOUNT** in **leafcount.asm**. The LC-3 code for main() and callee set-up of LEAFCOUNT has been given. You should implement the function logic and callee tear-down of LEAFCOUNT. Do not change anything else in leafcount.asm, or you may receive a score of zero. **Your program will be tested for functionality and the correct use of run-time stack.**

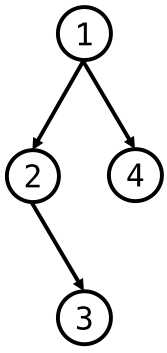
C code:

```
typedef struct treeNode node;
struct treeNode{
    int data;
    node *left;
    node *right;
};

int leaf_count(node *root){
    int left_count, right_count;
    if(root == NULL)
        return 0;
    if(root->left == NULL && root->right == NULL)
        return 1;
    else{
        left_count = leaf_count(root->left);
        right_count = leaf_count(root->right);
        return left_count + right_count;
    }
}
```

Input:
The root of the binary tree
Output:
The total number of the leaf nodes in the tree

Example:



Return value of the above example is **2** (node3 and node4 are the leaf nodes).

Testing:
Input files provided: test1.asm, test2.asm, test3.asm
Assemble with: lc3as leafcount.asm
Assemble the test code as: lc3as test1.asm
Test with:
lc3sim
>file test1.obj
>file leafcount.obj
>finish

Problem 2 (25 points): Linked List

An organization stores its customer data using `customer` structs. It uses a linked list to maintain pointers to the customer structs: each node in the linked list has a pointer to a customer struct and a next pointer, as shown in Figure 1. The linked list nodes are **sorted in ascending order by customerID** to which that node points, as shown in Figure 2.

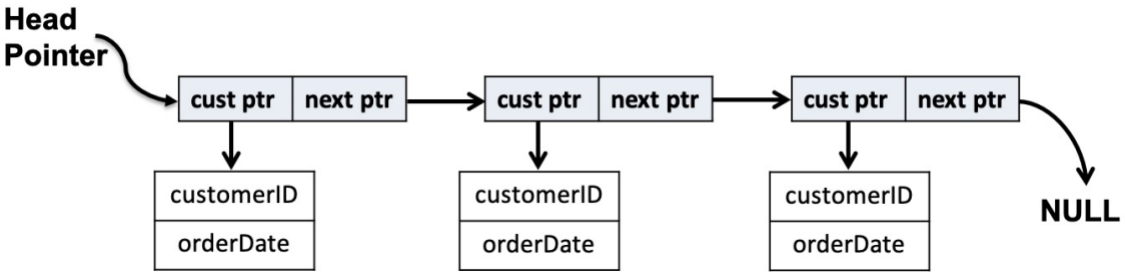


Figure 1. Each node in the linked list has a pointer to a customer struct and a next pointer.

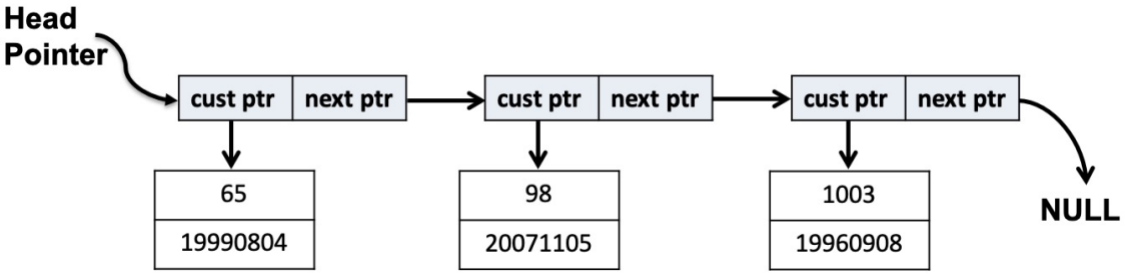


Figure 2. The linked list is sorted by customerID in ascending order.

You need to implement the following functions in **list.c**:

1. **int** `changeDate(node* head, int custID, int newDate)`

This function takes a pointer to the head node of the list, a target customerID, and an integer representing a date. It finds the node associated with the target customerID, and changes the order date associated with that customer with the new date and returns 1. If a node with the target customerID is not found, return 0.

2. `destroyList(node* head)`

This function takes the head of the linked list and deallocates all nodes in the linked list and the customers structs associated with them.

You may assume:

1. Each customerID is unique (will not appear more than once).
2. All orderDates are in the form of YYYYMMDD and only valid dates (greater than zero) will be used.
3. The linked list is sorted by customerID in ascending order.
4. Grading of destroyList() will include memory leak checks using valgrind.

Type definitions (in list.h):

```
typedef struct{
    int customerID;
    int orderDate;
}customer;


typedef struct node{
    customer *cust;
    struct node *next;
}node;
```

You are provided the following helper function(s) in helpers.{c, h}:

```
void printList(node* head); /*prints out the linked list
and the associated customer data*/
```

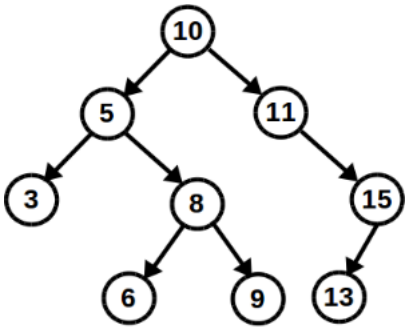
There are other helper functions used by main() or by these two helper functions. You may freely edit all of them according to your preferences and debugging needs.

Details:

1. You ONLY need to implement functions in list.c
2. Do NOT change the function definitions
3. You can modify the main function to test your code, we will only grade your list.c file
4. To compile, run the makefile: **make**
5. To run your code: **./list**

Problem 3 (25 points): Binary Search Tree

Given a binary search tree and two nodes (integers) in this tree, implement a function to check whether the two nodes are at the same level or not. The level of a node is the number of passing edges from the root to the node.



A binary search tree

Node	level
10	0
5	1
11	1
3	2
8	2
15	2
6	3
9	3
13	3

The function to implement:
int same_level(NODE* root, int node1, int node2) in binarytree.c, where node1 and node2 are the two nodes to be compared. Return 1 if both nodes are at the same level; otherwise, return 0.

Details:

1. Assume the two nodes will be in the binary search tree.
2. All nodes in the binary search tree are unique.
3. Do NOT change the function definition.
4. Do NOT use helper functions.
5. You can change main.c for testing. We will only grade your binarytree.c file.

Hint: you can find the distance from root to each node and compare whether the two distances are the same.

To compile the program:

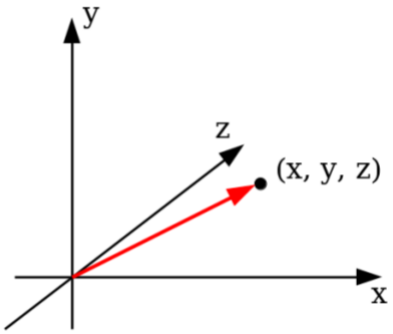
```
gcc -g -Wall main.c binarytree.c -o tree
```

To run the program:

```
./tree
```

Problem 4 (20 points): C++

We define a class ***Vec3*** with (x, y, z) as a 3d vector in math on a 3-d coordinate system.



You should complete the member functions (methods) in the class. The following are the list of member variables and member functions.

Class ***Vec3***:
Private member variables: _x, _y, _z
Public member functions:

- Constructor with different arguments:
  1) default constructor initializes variables (_x, _y, _z) = (0, 0, 0)
  2) initialize variables with given arguments
- Copy constructor.
- set_x(), set_y(), set_z(): Functions to modify _x, _y, _z with a given argument.
- x(), y(), z(): Functions to access _x, _y, _z.
- Operator overloading: + to do addition on two Vec3:
  a **+** ***b*** is defined as
  a+b = (a.x + b.x, a.y + b.y, a.z + b.z)
- Operator overloading: - to do subtraction on two Vec3:
  a **-** ***b*** is defined as
  a-b = (a.x - b.x, a.y - b.y, a.z - b.z)
- Operator overloading: ^ to do cross product on two Vec3:
  a **^** ***b*** is defined as
  a^b = (a.y * b.z - a.z * b.y , a.z * b.x - a.x * b.z , a.x * b.y - a.y * b.x)

Details:

1. Do NOT change the function definition.
2. Your code should be implemented in graphic.cpp. Do NOT change prog.cpp and graphic.hpp.
3. You have to handle corner cases.
4. To compile, run the makefile: **make**
5. To run your code: **./prog**

End of ECE 220 Final Exam