COP 3502 Section 2 Exam #1 - Part B (Linked Lists) Solutions

Date: 6/8/2020
Start Time: 4:25 pm EST
End Time: 4:55 pm EST

1) (25 pts) In the first part of this exam, you wrote a function that took in a row of Pascal's Triangle stored in an integer array, used it to dynamically allocate a new array and calculate the following row of Pascal's Triangle, freed the memory for the current row and returned a pointer to the newly created array storing the next row. For this question, you'll do the **exact same task**, but a row of Pascal's Triangle will be stored in a linked list. Thus, your function will take in a pointer to the front of the linked list storing the current row, will create a linked list storing the next row, free the memory associated with the linked list for the current row, and return a pointer to the front of the list storing the next row. To accomplish this task, you'll use the following struct (note the different names of the components):

```
typedef struct node {
    int coeff;
    struct node* next;
} node;
```

and write a function with the following prototype:

```
node* getNextRow(node* curRowPtr);
```

Since solving this problem requires the help of two utility functions we wrote in class, you may call both of these functions in your solution. These functions (with their internal code) are given below:

```
node* makeNewNode(int value) {
    node* res = malloc(sizeof(node));
    res->coeff = value;
    res->next = NULL;
    return res;
}
```

```
void freeList(node* listPtr) {
    if (listPtr != NULL) {
        freeList(listPtr->next);
        free(listPtr);
    }
}
```

This question is reasonably challenging, so I'll provide some hints for coding it:

1. Create separate new nodes storing 1 for the front and the back of the new list.

2. Save a pointer to the front of the original list.

3. Create a current pointer, which will "stitch" the list together, starting at the front 1.

4. Loop through the input list, accessing consecutive values in the list, and create a new node storing their sum, and incorporating each new node into the list, updating the current pointer of the new list as well as the pointer into the current list.

5. After the loop stitch together the last node you created to the back of the list.

6. free the old list.

7. Return a pointer to the front of the newly created list.

Solution

```
node* getNextRow(node* curRowPtr) {

    node* front = makeNewNode(1);           // 2 pts
    node* back = makeNewNode(1);            // 2 pts
    node* saveOrig = curRowPtr;             // 1 pt

    node* cur = front;                      // 1 pt
    while (curRowPtr->next != NULL) {       // 3 pts
        node* tmp=makeNewNode(curRowPtr->coeff+curRowPtr->next->coeff);
        cur->next = tmp;                    // 2 pts
        cur = tmp;                          // 2 pts
        curRowPtr = curRowPtr->next;        // 2 pts
    }

    cur->next = back;                       // 2 pts

    freeList(saveOrig);                     // 3 pts

    return front;                           // 1 pt
}
```

Note: the long line of code is worth 4 pts, 1 for the makeNewNode call, 1 pt for adding, 1 pt for accessing the correct successive elements, 1 pt for assigning to a node*.