CS 32 Worksheet 5

This worksheet is entirely **optional**, and meant for extra practice. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. All exams will be done on paper, so it is in your best interest to practice these problems by hand and not rely on a compiler.

If you have any questions or concerns, please go to any of the LA office hours.

Concepts

Recursion

Problems

1. What does the following code output and what does the function LA_power do? **(Easy)**

```
#include <iostream>
using namespace std;

int LA_power(int a, int b)
{
   if (b == 0)
       return 0;
   if (b % 2 == 0)
       return LA_power(a+a, b/2);

   return LA_power(a+a, b/2) + a;
}

int main()
{
  cout << LA_power(3, 4) << endl;
}
```

2. Given a singly-linked list class LL with a member variable *head* that points to the first *Node* struct in the list, write a function to recursively delete the whole list, void LL::deleteList(). Assume each Node object has a next pointer. **(Easy)**

```
struct Node {
      int data;
      Node* next;
```

```
};

class LL {
 public: // other functions such as insert not shown
      void deleteList(); // implement this function
private: // additional helper allowed
      Node* m_head;
};
```

3. Implement the function `getMax` recursively. The function returns the maximum value in `a`, an integer array of size `n`. You may assume that `n` will be at least 1. **(Easy)**

   ```
   int getMax(int a[], int n);
   ```

4. Given a string *str*, recursively compute a new string such that all the 'x' chars have been moved to the end. **(Medium)**

   ```
   string endX(string str);

   Example:
   endX("xrxe") → "rexx"
   ```

5. Implement the following function in a recursive fashion:

   ```
   bool isSolvable(int x, int y, int c);
   ```

   This function should return true if there exists nonnegative integers *a* and *b* such that the equation *ax + by = c* holds true. It should return false otherwise. **(Conceptually Hard)**

   Ex: `isSolvable(7, 5, 45) == true //a == 5 and b == 2`
   Ex: `isSolvable(1, 3, 40) == true //a == 40 and b == 0`
   Ex: `isSolvable(9, 23, 112) == false`

6. A robot you have programmed is attempting to climb a flight of stairs, for which each step has an associated number. This number represents the size of a leap that the robot is allowed to take backwards or forwards from that step (the robot, due to your engineering prowess, has the capability of leaping arbitrarily far). The robot must leap this exact number of steps.

   Unfortunately, some of the steps are traps, and are associated with the number 0; if the robot lands on these steps, it can no longer progress. Instead

of directly attempting to reach the end of the stairs, the robot has decided to first determine if the stairs are climbable. It wishes to achieve this with the following function:

```
bool isClimbable(int stairs[], int length);
```

This function takes as input an array of int that represents the stairs (the robot starts at position 0), as well as the length of the array. It should return true if a path exists for the robot to reach the end of the stairs, and false otherwise. (Note : the robot doesn't have to only end up at the first position past the end of the array) **(Hard)**

Ex: `isClimbable({2, 0, 3, 0, 0}, 5) == true`
`//stairs[0]->stairs[2]->End`
Ex: `isClimbable({1, 2, 4, 1, 0, 0}, 6) == true`
`//stairs[0]->stairs[1]->stairs[3]->stairs[2]->End`
Ex: `isClimbable({4, 0, 0, 1, 2, 1, 1, 0}, 8) == false`

7. Implement the function `sumOfDigits` recursively. The function returns the sum of all of the digits in the given *positive* integer `num`. **(Easy)**

```
int sumOfDigits(int num);

sumOfDigits(176); // return 14
sumOfDigits(111111); // return 6
```

8. Implement the function `isPalindrome` recursively. The function should return whether the given string is a palindrome. A palindrome is described as a word, phrase or sequence of characters that reads the same forward and backwards. **(Medium)**

```
bool isPalindrome(string foo);

isPalindrome("kayak"); // true
isPalindrome("stanley yelnats"); // true
isPalindrome("LAs rock"); // false (but the sentiment is true
:))
```

9. Write the following linked list functions recursively. **(Hard)**

```
// Node definition for singly linked list
struct Node {
    int data;
```

```
        Node* next;
    };

    // inserts a value in a sorted linked list of integers
    //         returns list head
    // before: 1 → 3 → 5 → 7 → 15
    // insertInOrder(head, 8);
    // after: 1 → 3 → 5 → 7 → 8 → 15
    Node* insertInOrder(Node* head, int value);

    // deletes all nodes whose keys/data == value, returns list
head
    // use the delete keyword
    Node* deleteAll(Node* head, int value);

    // prints the values of a linked list backwards
    // e.g. 0 → 2 → 1 → 4 → 1 → 7
    // reversePrint(head) will output 714120
    void reversePrint(Node* head);
```

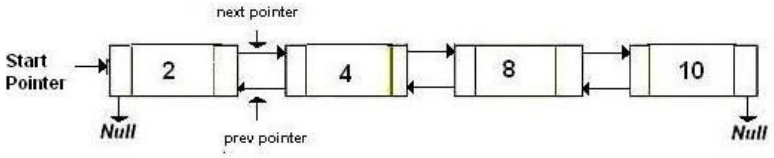
10. Write a recursive function `isPrime` to determine whether a given integer input is a prime number or not. You may add an auxiliary helper function if necessary. **(Medium)**

    Example:
    isPrime(11) → true
    isPrime(4) → false

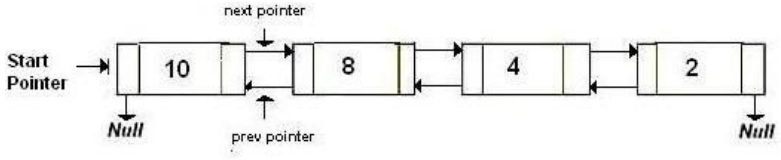
```
bool isPrime(int num) {
    // Fill in code here
}
```

11. Implement `reverse`, a recursive function to reverse a doubly linked list. It returns a pointer to the new head of the list. The integer value in each node must not be changed (but of course the pointers can be). **(Medium)**

    Example:
    Original:



    After:



```
// Node definition for doubly linked list
struct Node {
      int val;
      Node* next;
      Node* prev;
};

Node* reverse(Node* head);
```

12. Implement the following recursive function: **(Hard)**

    ```
    string longestCommonSubsequence(string s1, string s2);
    ```

    The function should return the longest common subsequence of characters between the two strings `s1` and `s2`. Basically, it should return a maximum length string of characters that are common to both strings and are in the same order in both strings.

    Example:
    ```
    string res = longestCommonSubsequence("smallberg",
          "nachenberg");
    //res should contain "aberg" as seen in the green chars
    res = longestCommonSubsequence("los angeles", "computers");
    //res should contain the string "oes"
    ```

13. Implement the recursive function `merge` that merges two sorted linked lists `l1` and `l2` into a single sorted linked list. The lists are singly linked; the last node in a list has a null next pointer. The function should return the head of the merged linked list. No new Nodes should be allocated while merging. **(Medium)**

    Example:
    ```
          l1:   1 -> 4 -> 6 -> 8
          l2:   3 -> 9 -> 10
    ```
    After merge: `1 -> 3 -> 4 -> 6 -> 8 -> 9 -> 10`

    ```
    // Node definition for singly linked list
    ```

```
struct Node {
      int val;
      Node* next;
};

Node* merge(Node* l1, Node* l2);
```

14. Rewrite the following function recursively. You can add new parameters and completely change the function implementation, but you can't use loops. (**Medium**)

    This function sums the numbers of an array from left to right until the sum exceeds some threshold. At that point, the function returns the running sum. Returns -1 if the threshold is not exceeded before the end of the array is reached.

```
int sumOverThreshold(int x[], int length, int threshold) {
  int sum = 0;
  for(int i = 0; i < length; i++) {
    sum += x[i];
    if (sum > threshold) {
      return sum;
    }
  }

  return -1;
}
```

15. Given the following program, give the output of each function call for parts a, b, and, c. **(Easy)**

```
void fizzbuzz(int x){
  if (x == 0) {
    cout << "fizzbuzz" << endl;
    return;
  }

  cout << "fizz" << endl;
  fizzbuzz(x-1);
  fizzbuzz(x-1);
}
```

    a. fizzbuzz(1);

b. fizzbuzz(2);
c. fizzbuzz(3);