CS 32 Worksheet Week 6

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. 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 contact your LA or go to any of the LA office hours.

Concepts

Recursion, Templates, STL

Recursion Problems (some problems may require a helper function)

1) 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.

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

}
```

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

   This function sums the elements of an array of nonnegative numbers 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;
```

```
}
```

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

```
string endX(string str);
```

Hint: https://www.cplusplus.com/reference/string/string/substr/

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

4) 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.

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`

Template/STL Problems

1) What is the output of this program?

```
template <class T>
void foo(T input) {
      cout << "Inside the main template foo(): " <<input<< endl;
}

template<>
void foo(int input) {
      cout << "Specialized template for int: " << input << endl;
}

int main() {
      foo<char>('A');
      foo<int>(19);
      foo<double>(19.97);
}
```

2) The following code has 3 errors that cause either runtime or compile time errors. Find and fix all of the errors.

```
class Potato {
public:
  Potato(int in_size) : size(in_size) { }
  int getSize() const {
    return size;
  }
private:
  int size;
};

int main() {
  vector<Potato> potatoes;
  Potato p1(3);
  potatoes.push_back(p1);
  potatoes.push_back(Potato(4));
  potatoes.push_back(Potato(5));

  vector<int>::iterator it = potatoes.begin();
  while (it != potatoes.end()) {
    potatoes.erase(it);
    it++;
```

```
  }

  for (it = potatoes.begin(); it != potatoes.end(); it++) {
    cout << it.getSize() << endl;
  }
}
```

3) Implement a stack class *Stack* that can be used with any data type using templates. Use a linked list (not an STL `list`) to store the stack and implement the functions *push(), pop(), top(), isEmpty(),* a default constructor, and a destructor that deletes the linked list nodes.

4) Implement a vector class *Vector* that can be used with any data type using templates. Use a dynamically allocated array to store the data. Implement only the *push_back()* function, default constructor, and destructor.

Extra Practice

Recursion:

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

```
int sumOfDigits(int num);

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

2) Write the following linked list functions recursively.

```
// 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);
```

3) 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)

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`

Template/STL:

1) Will this code compile? If so, what is the output? If not, what is preventing it from compiling?

   Note: We did not use `namespace std` because `std` has its own implementation of `max` and `namespace std` will thus confuse the compiler.

```
#include <iostream>

template <typename T>
T max(T x, T y)
{
      return (x > y) ? x : y;
}

int main()
{
      std::cout << max(3, 7) << std::endl;
      std::cout << max(3.0, 7.0) << std::endl;
      std::cout << max(3, 7.0) << std::endl;
}
```

2) The following code has 3 errors that cause either runtime or compile time errors. Find and fix all of the errors.

```
class Potato {
public:
  Potato(int in_size) : size(in_size) { }
  int getSize() const {
    return size;
  }
private:
  int size;
};

int main() {
  set<Potato> potatoes;
  Potato p1(3);
  Potato p2(4);
  Potato p3(5);
  potatoes.insert(p1);
  potatoes.insert(p2);
  potatoes.insert(p3);

  set<Potato>::iterator it = potatoes.begin();
  while (it != potatoes.end()) {
    potatoes.erase(it);
    it++;
  }

  for (it = potatoes.begin(); it != potatoes.end(); it++) {
    cout << it.getSize() << endl;
  }
}
```

3) You are given an STL `set<list<int>*>`. In other words, you have a set of pointers, and each pointer points to a list of ints. Consider the sum of a list to be the result of adding up all elements in the list. If a list is empty, treat its sum as zero.
Write a function that removes the lists with odd sums from the set. The lists with odd sums should be deleted from memory and their pointers should be removed from the set. This function should return the number of lists that are removed. You may assume that none of the pointers is null.

```
int deleteOddSumLists(set<list<int>*>& s);
```