CISC220H Homework 1 - LAB

Lab portion due September 21st at midnight on Sakai. The paper copy is due Tuesday, September 22nd in lab.

Each part is marked as **[PAIR]** or **[INDIVIDUAL]** and must be completed accordingly. Any part that has a * in front of it is worth 0 points and is optional.

Do not use any classes from any template library; the point of these exercises is for you to code from scratch. All work must be typed, including individual short answer questions **and diagrams**.

Academic Honesty Reminder

For parts that are marked as **[PAIR]** you may work with one other student. Both students must submit separate copies of the assignment and include the names of both members of the group.

For parts that are marked as **[INDIVIDUAL]** you may discuss general algorithms with other people, but you may not share any of your code in **any** form. You may not help other people debug their programs, except in the very limited way described on the class website. You may not use any code written by others, whether they are students or not. The TA and the instructor are available for office hours, and by appointment if you can't make hours because of a class conflict.

Assignment

1. * **[PAIR] 0 pts.** Consider the following code (available on our course website as pointerTest.cpp)

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

void swap(int *x, int *y) {
  int *temp = x;
  x = y;
  y = temp;
  cout << "--- In swap function ---\n";
  cout << "values= " << *x << ' ' << *y << '\n';
  cout << "addresses= " <<  x << ' ' << y << '\n';
}

int main(void) {
  int *x = new int(5);
  int *y = new int(6);

  cout << "--- Before swap function ---\n";
  cout << "values= " << *x << ' ' << *y << '\n';
  cout << "addresses= " <<  x << ' ' << y << '\n';

  swap(x, y);
  cout << "--- After swap function ---\n";
  cout << "values= " << *x << ' ' << *y << '\n';
  cout << "addresses= " <<  x << ' ' << y << '\n';
```

```
  delete x;
  delete y;
}
```

First, execute the code to see what happens. Does this code successfully swap x and y? Write a short answer describing the main reasons why or why not.

*** [INDIVIDUAL] 0 pts.** Next, draw three diagrams showing how the variables are stored in memory during execution of this code (one diagram for each state corresponding to the output lines with Before, In, and After). Similar to examples in class, use boxes to represent memory blocks, label variable names outside of the boxes, and use arrows to show pointers. Indicate whether each memory block is allocated on the stack or on the heap. This can be done on the Unix machines using xfig (or in a program of your choosing, but **do not submit a hand drawn diagram**).

2. **[PAIR] 30 pts.** You covered simple structures in CISC181. Consider the following structure that stores a Fraction as a numerator and a denominator:

```
struct Fraction {
  int numerator;
  int denominator;
};
```

Your task is to create a program that stores a collection of Fractions in an array and calculates the decimal valued sum of all of the Fractions. Users of your program must be able to input the values for numerator and denominator, and your program must input these Fractions one at a time and ask the user if they would like to add additional Fractions after each entry. Here is a sketch of the program:

```
int main(void) {
  // create a pointer to reference our collection of fractions
  Fraction **fractions;

  // while user has more fractions to input
  //   create new Fraction
  //   assign numerator and denominator to new Fraction
  //   add new Fraction to **fractions

  // create a double to store the sum of our fractions and init it to 0
  double sum = 0;

  // for each Fraction *fraction in **fractions
  //   add value of *fraction to sum

  // output the sum of the fractions to the console
  cout << "Sum of fractions = " << sum << '\n';
}
```

The difficulty of this problem lies in the adding of the new Fraction to **fractions. For this assignment, you need to grow the **fractions array:

(a) Create a new array of Fraction pointers with size one larger than the previous array

(b) Copy the old pointers to the new array

(c) Add the newest Fraction to the array

(d) Delete the memory used for the previous Fraction pointer array

Your solution must clean up ALL memory allocated from the heap before exiting the main function. You must also document every single line of your code (as done in the program sketch).

3. **[PAIR] 10 pts.** The previous part created a growable array structure to store a collection of fractions. In a short answer, describe why this is inefficient. Suggest some possible ways that would improve this inefficiency.

   [PAIR] 10 pts. Choose one efficiency improvement and implement it in a new version of the program.

4. **[PAIR] 50 pts.** Copy your code and Makefile from Part 2 to a new directory (you can call it Part4). Modify your code to use a single linked list instead of a growable array to store the Fractions. Use the following redefinition for the struct Fraction:

```
struct Fraction {
  int numerator;
  int denominator;
  Fraction *next;
};
```

   Your solution should add new Fractions created by the user to the END of the linked list. To do this you will need to keep track of a pointer to the head of your Fraction linked list and to the tail of your Fraction linked list. You must also implement a link traversal to calculate the sum (in place of the for loop from Part 2).

   Your solution must clean up ALL memory allocated from the heap before exiting the main function. You must also document every single line of your code (as done in the program sketch).

   *** If you already implemented a linked list as your efficiency improvement in Part 3, implement a growable array that doubles in size every time it needs to grow (instead of one element at a time). You will need to keep track of the count of actual array elements you are using to properly implement the sum function.

Submission of completed Homework 1 is due September 21st at midnight. You should submit the following to Sakai and printed copies to your TA:

1. *optional* Short answer and diagram for Part 1.
2. Code for Part 2, including Makefile.
3. Script output showing execution of Code for Part 2.
4. Short answer for Part 3.

5. Code for Part 3, including Makefile.
6. Script output showing execution of Code for Part 3.
7. Code for Part 4, including Makefile.
8. Script output showing execution of Code for Part 4.