CS2134 HOMEWORK 5*
Fall 2016
Due 11:00 p.m. Sat Oct 22, 2016

Be sure to include your name at the beginning of each file! Assignment 5 include a programming portion and a written part. The programming portion must compile and consist of a single file (hw05.cpp). The typed portion should consist of a single file (hw05written) in a .pdf format. Be sure to include your name at the beginning of each file! You must hand in the file via NYU Classes.

Programming Part:

1. Modify programming problem 1 in assignment 4B[1] to not store the duplicates, i.e. `101, 101N, 101S` are all the same stop.

2. Create a menu for the user to access the data created in programming problem 1 In the interactive phase, the program will repeatedly prompt the user to take any one of the following actions:

   (a) Print out the information about all the train stops on a specific route; if no route is found, you should print out a message informing the user.

   (b) Print out the information about a specific train stop; if no train stop is found, you should print out a message informing the user.

   (c) Print out all the train stops within a certain distance, where the user enters the geographical coordinates. If no stop is found within the distance, print out a message informing the user.

   (d) quit

   To perform these operations you will define one generic function template[2] and four functors.

   Write a generic function template called `perform_if` that takes four parameters: two iterators, `itrStart, itrEnd`, and two functors, `pred` and `op`. The two iterators should have have the capability of forward iterators. Both functors will have the overloaded `operator()` that takes one argument. The functor `pred`'s overloaded `operator()` returns a boolean value; the functor `op`'s overloaded `operator()` does not return any value. The function will apply `pred`'s overloaded `operator()` to each item in the range `[itrStart, itrEnd)`. If `pred`'s overloaded `operator()` returns `true`, the other functor `op`'s overloaded `operator()` is then applied to the item, e.g.

```
if ( pred( *itr ) )
{
    op(*itr);
    how_many++;
}
```

*5% extra credit will be given if you turn this assignment in on Fri Oct 21, 2016 by 5:00 p.m.

[1]You may use the published solution as long as you cite that you are using it.

[2]This generic function template should be written in the STL generic template function style and should *not* written to be used only with the train stop data.

The return value of the generic function template `perform_if` is an `int` which returns the number of items for which the functor `pred` returned `true`.

The four functors you will write are:

(a) Write a functor called `isStopOnRoute` that has a private member variable called `route` of type `char`. The constructor takes a single parameter of type `char` which it uses to initialize the variable `route`. It also contains an overloaded `operator()` that takes a single parameter of type `trainStopData`; the operator returns `true` if the train stop is on the route.[3]

(b) Write a functor called `isSubwayStop` that has a private member variable called `stopId` of type `string`. The constructor takes a single parameter of type `string` which it uses to initialize the variable `stopId`. It also contains an overloaded `operator()` that takes a single parameter of type `trainStopData`; the operator returns `true` if the train stop has an id which is the same as `stopid.`

(c) Write a functor called `isSubwayStopNearX` that has three private member variables called `longitude, latitude`, and `d` of type `double`. The constructor takes three parameters of type `double` which it uses to initialize the private member variables. It also contains an overloaded `operator()` that takes a single parameter of type `trainStopData`; the operator returns `true` if the distance between the train station location and the value of the functors private member variables, `longitude` and `latitude`, is at most `d`. The distance between two points on a sphere is computed using the `haversine` formula. The code to compute this formula is on NYU Classes in a file called `haversine.txt`.[4]

(d) Write a functor called `printTrainStopInfo`. This class contains a single method, an overloaded `operator()`, that takes a single parameter of type `trainStopData` that prints out the train stop information.

3. Rewrite the recursive part of the merge sort algorithm presented in class to work with any container which has random access iterators, and has an overloaded `operator<` for comparison of items in the container. You will not write your own merge method. Instead you will use the STL merge algorithm. Here is the driver for the mergesort algorithm you will write.

```
template <class RandItr>
void mergeSort( RandItr start, RandItr end )
{
     int  sz = end - start;  // or use auto sz = end-start;
     typedef typename iterator_traits< RandItr >::value_type Object; //Xcode
     // typedef  iterator_traits< RandItr >::value_type Object; //Other compilers
     // Don't worry about this line of code
      vector<Object> tmp( sz );
      mergeSort( tmp.begin(), start, end );
}
```

The STL algorithm[5] merge takes five arguments, `first1, last1, first2, last2, result`. The merge algorithm combines "the elements in the sorted ranges `[first1,last1)` and `[first2,last2)`, into a new range beginning at result with all its elements sorted."[6]

[3]The train stop route is the first letter of the train stop id. For example, train stop id 202 is on route 2, and train stop id $B20$ is on route B.

[4]You might need to add `#define _USE_MATH_DEFINES`.

[5]You might need to add `#include<algorithm>`

[6]The quote is from http://www.cplusplus.com/reference/algorithm/merge/. Don't forget to copy the elements back into the original container after calling the merge function. The STL merge function does NOT have the same signature as the merge function we discussed in class.

4. Create a functor called `meFirst` which has a private member variable, `me`, of type `string`. Its constructor will take one argument of type `string` that it uses to initialize its private member variable, `me`. The functor's overloaded `operator( )` takes two arguments of type student and returns a boolean value. It returns `true` if the first students' name is less than the second students' name unless either of the students' name equals the variable me then that name is always less than the other name. Test your functor using the STL 3 parameter algorithm sort, the student class we discussed in class and some data you create yourself.[7]

   The class student is from the lecture notes on C++

```
class student
{
  private:
    string name;
    double gpa;
  public:
     student ( const string & name, double gpa ): name( name ), gpa( gpa ) { }
     string get_name() const { return name; }
     double get_gpa() const { return gpa; }
};
```

5. Write an algorithm to do the following: given a vector of boolean values `(true/false)`, order the container such that the `false` values come before the `true` values. Your algorithm must run in $O(n)$ and use $O(1)$ space. You algorithm may <u>not</u> simply count the number of `true` or `false` values and then assign the correct number of `false` and `true` values in the vector. You should think of your algorithm as the first step in creating an algorithm that sorts based on `true/false` values, where the items are not simply `true/false` values, but large objects that evaluate to true/false by using a functor. Hint: Be inspired by one of the sorting algorithms we discussed in class.

6. (Extra Credit) Rather than sorting the numbers in either ascending or descending order, suppose we wanted to sort the an array in a new way. The criteria for this sort is given as: if the index of the i'th element is odd, then that element should be less than it's neighboring elements. If the index of the i'th element is even, it should be greater than it's neighboring elements. Your algorithm must run in $O(n \log n)$ time and can use at most $O(n)$ extra space Ex: Consider the array `[5, 9, 8, 2, 3, 4]`. After the sort the array should have: `[5, 2, 4, 3, 9, 8]`
   Explanation: 5 is at index 0, so it must be greater than it's neighbors (2)
   2 is at index 1, so it must be less than it's neighbors `(5 & 4)`
   4 is at index 2, so it must be greater than it's neighbors `(2 & 3)`.... and so on
   Note: The solution may not be unique.

   This function should not have any loops.

[7] This is a biased sort. One item is always first regardless of the other items.

Written Part:

1. In programming part 2 of this assignment, you are asked to write a generic function template called `perform_if`.

   - Write the pseudo code for the three to six main steps need to implement `perform_if`.
   - Write the `preconditions` and `postconditions` of the function.
   - Using Big-Oh notation, what is the running time of this generic function template?

2. (a) Draw the recursion tree for `myRecFunc(4)`.
   (b) What is printed by the following function call: `myRecFunc(4)`.
   (c) Look at your recursion tree [8]. What is the running time of `myRecFunc(n)`.

```
void myRecFunc(int n)
{
   cout << n << ": ";
   if (n < 1) return;
   myRecFunc(n/2);
   myRecFunc(n/2);
   for (int i = 1; i < n; ++i)
      cout << "*";
   cout << endl;
}
```

3. (a) Draw the recursion tree for `myRecFunc(4)`.
   (b) What is printed by the following function call: `myRecFunc(4)`.
   (c) Look at your recursion tree[9]. Determine what is the big-oh running time for `myRecFunc(n)`.

```
void myRecFunc(int n)
{
   cout << n << ": ";
   if (n < 1) return;
   myRecFunc(n/2);
}
```

4. For this recursive Fibonacci function, `fib`, how many function calls are made if n=3; how about if n=4? Using the the number of function calls `fib` made when when n=3 and when n= 4, compute how many function calls are made when n=5. **Show your work.**

```
int fib( int n )
{
    if( n <= 1 )
        return 1;
    else
        return fib( n - 1 ) + fib( n - 2 );
}
```

[8] Hint: draw the recursion tree for `myRecFunc(5)`, `myRecFunc(6)`, etc. You should be able to notice a pattern
[9] Hint: draw the recursion tree for `myRecFunc(5)`, `myRecFunc(6)`, etc. You should be able to notice a pattern

5. What would be printed out by the following code, if the vector `a` contained $\{9, 8, -11, 2, 0, 3\}$, and the function `printVec` printed out contents of `a`? (Don't worry about the exact format of the output, the point is to show how the contents of the vector is or is not changing)

```
// Simple insertion sort.
template<class Comparable>
 void insertionSort( vector<Comparable> & a )
 {
      int j;
      for( int p = 1; p < a.size( ); p++ )
      {
           Comparable tmp = a[ p ];
           for( j = p; j > 0 && tmp < a[ j - 1 ]; j-- )
                    a[ j ] = a[ j - 1 ];
           a[ j ] = tmp;
           printVec(a);  // prints the contents of the vector in order
      }
  }
```

6. What would be printed out by the following code, if the vector `a` contained $\{9, 8, -11, 2, 0, 3\}$, and the function `printVec` printed out contents of `a`? (Don't worry about the exact format of the output, the point is to show how the contents of the vector is or is not changing)

```
template <class Comparable>
void mergeSort( vector<Comparable> & a,
           vector<Comparable> & tmpArray, int left, int right )
{
    if( left < right )
    {
        int center = ( left + right ) / 2;
        mergeSort( a, tmpArray, left, center );
        mergeSort( a, tmpArray, center + 1, right );
        mymerge( a, tmpArray, left, center + 1, right );
        printVec(a); // prints the contents of the vector in order

    }
}
```

7. What would be printed out by the following code, if the vector `a` contained $\{9, 8, -11, 2, 0, 3\}$, and the function `printVec` printed out contents of `a`? (Don't worry about the exact format of the output, the point is to show how the contents of the vector is or is not changing)

```
void quickSort( vector<int> & a, int low, int high )
{
     if (low < high)
     {
         int mid = ( low + high )/2; // select pivot to be element in middle position
         int pivot = a[ mid ];
         swap( a[high], a[mid] ); // put pivot in a[high]

         // Begin partitioning
         int i, j;
         for( i = low, j = high - 1; ; )
         {
             while (  a[i ] < pivot  )  ++i;
             while(  j > i && pivot < a[j ] ) --j;
             if( i < j )
                   swap( a[ i++ ], a[ j-- ]  );
             else
                   break;
         }
         swap( a[ i ], a[ high  ] );  // Restore pivot
         printVec(a); // prints the contents of the vector in order
         quickSort( a, low, i - 1 );     // Sort small elements
         quickSort( a, i + 1, high );    // Sort large elements
     }
}
```

8. Show all the function calls organized as a recursion tree where you include the contents of the container `a` for:

   (a) mergeSort on input `a = {28, 10, 2, 27, 5, 1}`

   (b) quickSort on input `a =  {28, 10, 2, 27, 5, 1}`

9. When all the items in the vector are in "almost" sorted order (e.g. `a` contains $\{2, 1, 4, 3, 6, 5, \ldots, n/2, n/2-1, \ldots, n, n-1\}$, what is the *average* running time in Big-Oh notation for:

   (a) insertionSort

   (b) mergeSort

   (c) quickSort

10. What is the average running time of the quickSelect algorithm we discussed in class?

11. For the quickSelect algorithm we discussed in class, if after the partition it turns out that $i + 1 = k$ why does the function not recursively call itself?