**CSCI 241 Assignment 8, Part 1**

**Assignment**

For this part of the assignment you will write a pair of C++ template functions to read a series of items from an input file, and then print the items.

**Program**

Implement the following template functions in a header file called sorts.h. This header file should have header guards (as usual) and should contain both the prototypes and definitions for the functions.

* template <class T>  
  void buildList(vector<T>& set, const char\* fileName)

This function should read items from an input file and put them into a vector. The first argument to this function is a reference to a vector object that will be used to store the items. The second argument is a C-style string containing the full pathname of the input file.

The function should first open the file for input, then read items from the file using the >> operator one at a time until end of file, inserting them into the vector. Finally, it should close the input file. Here's some pseudocode for the logic:

T item;

ifstream inFile;

*Open* inFile *for input using* fileName *as the name of the file to open*

*Check that the file has been successfully opened*

*Read* item *from input file*

while (*not end of file*)

{

set.push\_back(item);

*Read* item *from input file*

}

*Close the input file*

* template <class T>  
  void printList(const vector<T>& set, int itemWidth, int numPerLine)

This function should print a list of items stored in a vector. The first argument to this function is a reference to a constant vector object that will contain the items to print. The second argument is an integer specifying the width an individual item should occupy when printed. The third argument is an integer specifying the maximum number of items that should be printed in a single line of output.

A driver program, assign8.cpp, is provided below to test your code for this part of the assignment. A copy of the driver program can also be found on turing at /home/turing/t90kjm1/CS241/Code/Fall2017/Assign8/Part1/assign8.cpp.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROGRAM: CSCI 241 Assignment 8, Part 1

PROGRAMMER: your name

LOGON ID: your z-ID

DUE DATE: due date of assignment

FUNCTION: This program builds and sorts lists using the quick

sort algorithm.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <iomanip>

#include <vector>

#include <string>

#include "sorts.h"

using std::cout;

using std::fixed;

using std::left;

using std::setprecision;

using std::string;

using std::vector;

// Data files

#define D1 "/home/turing/t90kjm1/CS241/Data/Fall2017/Assign8/data8a.txt"

#define D2 "/home/turing/t90kjm1/CS241/Data/Fall2017/Assign8/data8b.txt"

#define D3 "/home/turing/t90kjm1/CS241/Data/Fall2017/Assign8/data8c.txt"

// Output formatting constants

#define INT\_SZ 4 // width of integer

#define FLT\_SZ 7 // width of floating-pt number

#define STR\_SZ 12 // width of string

#define INT\_LN 15 // no of integers on single line

#define FLT\_LN 9 // no of floating-pt nums on single line

#define STR\_LN 5 // no of strings on single line

int main()

{

vector<int> v1; // vector of integers

vector<float> v2; // vector of floating-pt nums

vector<string> v3; // vector of strings

// Print header message

cout << "\*\*\* CSCI 241: Assignment 8 - Part 1 Output \*\*\*\n\n";

// Build and print first list

cout << "First list:\n\n";

buildList(v1, D1);

printList(v1, INT\_SZ, INT\_LN);

// Build and print second list

cout << fixed << setprecision(2);

cout << "\nSecond list:\n\n";

buildList(v2, D2);

printList(v2, FLT\_SZ, FLT\_LN);

// Build and print third list

cout << left;

cout << "\nThird list:\n\n";

buildList(v3, D3);

printList(v3, STR\_SZ, STR\_LN);

// Print termination message

cout << "\n\*\*\* End of program execution \*\*\*\n";

return 0;

}

**The STL Vector Class**

The vector class is part of the C++ Standard Template Library. It is a template class that provides an alternative to using an array.

The vector class is implemented as a dynamic array. Just like a regular array, a vector has its elements stored in contiguous storage locations. But unlike regular arrays, storage in a vector is handled automatically, allowing it to be expanded and contracted as needed.

Vectors are good at:

* Accessing individual elements by their position index.
* Iterating over the elements in any order.
* Adding to and removing elements from the tail end of the vector.

Compared to arrays, they provide almost the same performance for these tasks, plus they have the ability to be easily resized. They usually consume more memory than arrays when their capacity is handled automatically (this is in order to accommodate extra storage space for future growth).

*Example Vector Operations*

To create an empty vector, you can use the default constructor:

vector<int> v1; // Create an empty vector of integers

You can also create a vector of a specific initial size:

vector<int> v1(20); // Create a vector of integers with size 20

To add a new element to the tail end of a vector, you can use the push\_back() method:

v1.push\_back(12); // Add a new element with value 12 to the tail end of the vector

You can use a for loop and the subscript operator to loop through the elements of a vector, similar to the way you would loop through an array:

for (int i = 0; i < (int) v1.size(); i++)

cout << v1[i] << ' ';

cout << endl;

(The size() method typically returns an unsigned int, so we need to perform a type cast to avoid a warning from the compiler.)

Vectors also have an at() method that will throw an out\_of\_range exception if you try to access an element outside the bounds of the dynamic array.

More information about the vector class can be found here:

[Vector class reference](http://www.cplusplus.com/reference/stl/vector/)

**CSCI 241 Assignment 8, Part 2**

**Assignment**

For this part of the assignment you will write a number of C++ template functions to sort a list of items using the recursive **quick sort** algorithm.

**Program**

Add the following template functions to your sorts.h file. The header file should contain both the prototypes and definitions for these functions.

* template <class T>  
  bool lessThan(const T& item1, const T& item2)

This function should return true if item1 is less than item2 and false if not. You may assume that the data type T can be compared using the standard relational operators.

* template <class T>  
  bool greaterThan(const T& item1, const T& item2)

This function should return true if item1 is greater than item2 and false if not. You may assume that the data type T can be compared using the standard relational operators.

Implement the following template functions in a header file called quicksort.h. This header file should have header guards (as usual) and should contain both the prototypes and definitions for the functions.

* template <class T>  
  void quickSort(vector<T>& set, bool (\*compare)(const T&, const T&))

This function should sort the items in the vector set using the quick sort algorithm. The first argument to this function is a reference to a vector object containing the list of items to sort. The second argument is a pointer to a comparison function that can be used to compare two items of the template type.

This function should call the recursive quick sort function, passing it the vector, the subscript of the first vector element (which is 0), the subscript of the last vector element (which is set.size() - 1), and the pointer to the comparison function (compare), e.g.:

quickSort(set, 0, set.size()-1, compare);

* template <class T>  
  void quickSort(vector<T>& set, int start, int end, bool (\*compare)(const T&, const T&))

This function performs the recursive calls to implement the quick sort algorithm. The logic is:

int pivotPoint;

if (start < end)

{

pivotPoint = partition(set, start, end, compare); // Get the pivot point

quickSort(set, start, pivotPoint - 1, compare); // Sort first sublist

quickSort(set, pivotPoint + 1, end, compare); // Sort second sublist

}

* template <class T>  
  int partition(vector<T>& set, int start, int end, bool (\*compare)(const T&, const T&))

This function selects a pivot element and then partitions the vector around the pivot. The logic is:

int pivotIndex, mid;

T pivotValue;

mid = (start + end) / 2;

*Swap elements* start *and* mid *of the vector*

pivotIndex = start;

pivotValue = set[start];

for (int scan = start + 1; scan <= end; scan++)

{

if (compare(set[scan], pivotValue))

{

pivotIndex++;

*Swap elements* pivotIndex *and* scan *of the vector*

}

}

*Swap elements* start *and* pivotIndex *of the vector*

return pivotIndex;

A driver program, assign8.cpp, is provided below to test your code for this part of the assignment. A copy of the driver program can also be found on turing at /home/turing/t90kjm1/CS241/Code/Fall2017/Assign8/Part2/assign8.cpp.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROGRAM: CSCI 241 Assignment 8, Part 2

PROGRAMMER: your name

LOGON ID: your z-ID

DUE DATE: due date of assignment

FUNCTION: This program builds and sorts lists using the quick

sort algorithm.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <iomanip>

#include <vector>

#include <string>

#include "sorts.h"

#include "quicksort.h"

using std::cout;

using std::fixed;

using std::left;

using std::setprecision;

using std::string;

using std::vector;

// Data files

#define D1 "/home/turing/t90kjm1/CS241/Data/Fall2017/Assign8/data8a.txt"

#define D2 "/home/turing/t90kjm1/CS241/Data/Fall2017/Assign8/data8b.txt"

#define D3 "/home/turing/t90kjm1/CS241/Data/Fall2017/Assign8/data8c.txt"

// Output formatting constants

#define INT\_SZ 4 // width of integer

#define FLT\_SZ 7 // width of floating-pt number

#define STR\_SZ 12 // width of string

#define INT\_LN 15 // no of integers on single line

#define FLT\_LN 9 // no of floating-pt nums on single line

#define STR\_LN 5 // no of strings on single line

int main()

{

vector<int> v1; // vector of integers

vector<float> v2; // vector of floating-pt nums

vector<string> v3; // vector of strings

// Print header message

cout << "\*\*\* CSCI 241: Assignment 8 - Part 2 Output \*\*\*\n\n";

// sort and print first list

cout << "First list - ascending order:\n\n";

buildList(v1, D1);

quickSort(v1, &lessThan);

printList(v1, INT\_SZ, INT\_LN);

// Sort and print second list

cout << fixed << setprecision(2);

cout << "\nSecond list - descending order:\n\n";

buildList(v2, D2);

quickSort(v2, &greaterThan);

printList(v2, FLT\_SZ, FLT\_LN);

// Sort and print third list

cout << left;

cout << "\nThird list - ascending order:\n\n";

buildList(v3, D3);

quickSort(v3, &lessThan);

printList(v3, STR\_SZ, STR\_LN);

// print termination message

cout << "\n\*\*\* End of program execution \*\*\*\n";

return 0;

}

**CSCI 241 Assignment 8, Part 3  
100 points**

**Assignment**

For this final part of the assignment you will write several C++ template functions to sort a list of items using the recursive **merge sort** algorithm.

**Program**

Implement the following template functions in a header file called mergesort.h. This header file should have header guards (as usual) and should contain both the prototypes and definitions for the functions.

* template <class T>  
  void mergeSort(vector<T>& set, bool (\*compare)(const T&, const T&))

This function should sort the items in the vector set using the merge sort algorithm. The first argument to this function is a reference to a vector object containing the list of items to sort. The second argument is a pointer to a comparison function that can be used to compare two items of the template type.

This function should call the recursive merge sort function, passing it the vector, the subscript of the first vector element (which is 0), the subscript of the last vector element (which is set.size() - 1), and the pointer to the comparison function (compare), e.g.:

mergeSort(set, 0, set.size()-1, compare);

* template <class T>  
  void mergeSort(vector<T>& set, int low, int high, bool (\*compare)(const T&, const T&))

This recursive function divides a vector into two subvectors, sorts them, and then merges the two sorted subvectors.

int mid;

if (low < high)

{

mid = (low + high) / 2;

// Divide and conquer

mergeSort(set, low, mid, compare);

mergeSort(set, mid+1, high, compare);

// Combine

merge(set, low, mid, high, compare);

}

}

* template <class T>  
  void merge(vector<T>& set, int low, int mid, int high, bool (\*compare)(const T&, const T&))

This function merges two sorted subvectors.

// Create temporary vector to hold merged subvectors

vector<T> temp(high - low + 1);

int i = low; // Subscript for start of left sorted subvector

int j = mid+1; // Subscript for start of right sorted subvector

int k = 0; // Subscript for start of merged vector

// While not at the end of either subvector

while (i <= mid && j <= high)

{

if (compare(set[j], set[i]))

{

*Copy element* j *of* set *into element* k *of* temp

*Add one to* j

*Add one to* k

}

else

{

*Copy element* i *of* set *into element* k *of* temp

*Add one to* i

*Add one to* k

}

}

// Copy over any remaining elements of left subvector

while (i <= mid)

{

*Copy element* i *of* set *into element* k *of* temp

*Add one to* i

*Add one to* k

}

// Copy over any remaining elements of right subvector

while (j <= high)

{

*Copy element* j *of* set *into element* k *of* temp

*Add one to* j

*Add one to* k

}

// Copy merged elements back into original vector

for (i = 0, j = low; j <= high; i++, j++)

*Copy element* i *of* temp *into element* j *of* set

A driver program, assign8.cpp, is provided below to test your code for this part of the assignment. A copy of the driver program can also be found on turing at /home/turing/t90kjm1/CS241/Code/Fall2017/Assign8/Part3/assign8.cpp.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROGRAM: CSCI 241 Assignment 8

PROGRAMMER: your name

LOGON ID: your z-ID

DUE DATE: due date of assignment

FUNCTION: This program builds and sorts lists using the quick

sort algorithm.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <iomanip>

#include <vector>

#include <string>

#include "sorts.h"

#include "quicksort.h"

#include "mergesort.h"

using std::cout;

using std::fixed;

using std::left;

using std::setprecision;

using std::string;

using std::vector;

// Data files

#define D1 "/home/turing/t90kjm1/CS241/Data/Fall2017/Assign8/data8a.txt"

#define D2 "/home/turing/t90kjm1/CS241/Data/Fall2017/Assign8/data8b.txt"

#define D3 "/home/turing/t90kjm1/CS241/Data/Fall2017/Assign8/data8c.txt"

// Output formatting constants

#define INT\_SZ 4 // width of integer

#define FLT\_SZ 7 // width of floating-pt number

#define STR\_SZ 12 // width of string

#define INT\_LN 15 // no of integers on single line

#define FLT\_LN 9 // no of floating-pt nums on single line

#define STR\_LN 5 // no of strings on single line

int main()

{

vector<int> v1; // vector of integers

vector<float> v2; // vector of floating-pt nums

vector<string> v3; // vector of strings

// Print header message

cout << "\*\*\* CSCI 241: Assignment 8 - Output \*\*\*\n\n";

// sort and print first list

cout << "First list - ascending order:\n\n";

buildList(v1, D1);

quickSort(v1, &lessThan);

printList(v1, INT\_SZ, INT\_LN);

v1.clear();

cout << "\nFirst list - descending order:\n\n";

buildList(v1, D1);

mergeSort(v1, &greaterThan);

printList(v1, INT\_SZ, INT\_LN);

// Sort and print second list

cout << fixed << setprecision(2);

cout << "\nSecond list - descending order:\n\n";

buildList(v2, D2);

quickSort(v2, &greaterThan);

printList(v2, FLT\_SZ, FLT\_LN);

v2.clear();

cout << "\nSecond list - ascending order:\n\n";

buildList(v2, D2);

mergeSort(v2, &lessThan);

printList(v2, FLT\_SZ, FLT\_LN);

// Sort and print third list

cout << left;

cout << "\nThird list - ascending order:\n\n";

buildList(v3, D3);

quickSort(v3, &lessThan);

printList(v3, STR\_SZ, STR\_LN);

v3.clear();

cout << "\nThird list - descending order:\n\n";

buildList(v3, D3);

mergeSort(v3, &greaterThan);

printList(v3, STR\_SZ, STR\_LN);

// print termination message

cout << "\n\*\*\* End of program execution \*\*\*\n";

return 0;

}

**Other Points**

* Assignment 8 as a whole is worth 100 points, and only this final part of the assignment needs to be submitted for grading. However, if you are unable to complete all three parts of the assignment, you may submit one of the earlier parts for partial credit.
* As usual, you must have a makefile. The name of your final executable should be assign8.
* Don't forget to get rid of all compiler warnings when the -Wall compilation option is used.
* As always, programs that do not compile on turing/hopper automatically receive 0 points.
* Submit your program using the electronic submission guidelines posted on the course web site and discussed in class.