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/Fall2016/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 prints lists.
*****************************************************************/

#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/Fall2016/Assign8/data8a.txt"
#define D2 "/home/turing/t90kjm1/CS241/Data/Fall2016/Assign8/data8b.txt"
#define D3 "/home/turing/t90kjm1/CS241/Data/Fall2016/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