

Laboratory Exercise 7 (Makefiles and Templates)

Topics: dynamic arrays

makefiles

Template functions

Sorting

SLT vector class and iterators

Goals: Upon successful completion of this lab you should be able to:

1. Resize a dynamic array of objects
2. Create a simple makefile and use the make utility
3. Write a C++ program that uses an STL vector.

Related text sections:

Chapter 11

Chapter 19

Laboratory Exercise 7 Instructions

Part 1.

1. In your CS 215 directory, create a new subdirectory called Lab7.1.
2. Divide the header file you submitted for Lab 6 (Lab6.h) into 2 files: worddata.h and worddata.cpp, in directory Lab7.1.
 - a. your worddata.h file should contain your WordData class description and appropriate macroguards.
 - b. your worddata.cpp file should contain an “include” for your worddata.h file and should contain implementations for all of the friend and member functions you have created for the class.
3. Create a copy of your words.txt test file from lab 6 as Lab7.txt in directory Lab7.1.
4. Enter the following make commands into a new file called makefile in directory Lab7.1.

```
lab7 : application.o worddata.o
      g++ -o lab7 application.o worddata.o

application.o : application.cpp sorts.h worddata.h
      g++ -c application.cpp

worddata.o : worddata.cpp worddata.h
      g++ -c worddata.cpp

clean :
      rm -f core.* *.o lab7
```

5. The dependencies in this makefile indicate that 6 files are need to compile the executable lab7: application.o, worddata.o, application.cpp, sorts.h, worddata.h, amd worddata.cpp. The .o files will be created by the make utility. You have already created worddata.h and worddata.cpp. The missing files are application.cpp and sorts.h.
6. Enter the following template swap and sorts into a new file called sorts.h in directory Lab7.1.

```
#ifndef SORT_TMP
#define SORT_TMP

// Description : This file contains template implementations of 3
// simple sorts (exchange, insertion, and selection).

template <class T>
void Swap (T & a, T & b)
{
    T t = a;
    a = b;
    b = t;
}
```

```

template <class T>
void eSort (T * A, size_t N)
{ // implemetation of exchange (bubble) sort for array (A) of N entries
  // of type T
    bool swapped = true;
    for (int i = 0; swapped && i < N; i++)
    {
        swapped = false;
        for (int j = 1; j < N-i; j++)
            if (A[j-1] > A[j])
            {
                Swap (A[j], A[j-1]);
                swapped = true;
            }
    }
}

template <class T>
void iSort (T * A, size_t N)
{ // implemetation of insertion sort for array (A) of N entries
  // of type T
    for (int i = 1; i < N; i++)
        for (int j = i; j > 0 && A[j] > A[j-1]; j--)
            Swap (A[j], A[j-1]);
}

template <class T>
void sSort (T * A, size_t N)
{ // implemetation of selection sort for array (A) of N entries
  // of type T
    for (int i = 0; i < N; i++)
    {
        int p = i;
        for (int j = i+1; j < N; j++)
            if (A[p] > A[j])
                p = j;
        Swap (A[i], A[p]);
    }
}
#endif

```

7. Create an application program (called application.cpp) that
 - a. Uses a dynamic array of WordData entries. The initial size of the array should be 20 entries.
 - b. Reads words from the file Lab7.txt into the array of WordData entries using your overloaded input (>>) operator.
 - c. If the array size has been exceeded, increases the size of the array by 10 entries.
 - d. Sorts the entries in your array into ascending order using the eSort or sSort function found in sorts.h.
 - e. Prints a list of the sorted entries using your overloaded output (<<) function. The listing should be preceded by a heading line which identifies the columns and followed by a line of 2 blank lines.
 - f. Sorts the entries in your array into descending order using the iSort function found in sorts.h.

- g. Prints a list of the sorted entries using your overloaded output (<<) function. The listing should be preceded by a heading line which identifies the columns and followed by 2 blank lines.
 - h. Resorts the entries in your array into ascending order using the `ssort` function found in `sorts.h`.
 - i. Prints a list of the sorted entries using your overloaded output (<<) function. The listing should be preceded by a heading line which identifies the columns and followed by 2 blank lines.
8. Using the `make` utility, try to compile your program. You should simply enter “`make`” at the Linux prompt.
 9. Your program probably did not compile. If you carefully parse the error messages, you will see the statement “no match for operator `>`”; the class is missing a greater-than (`>`) operator. Add a greater-than operator prototype to the `WordData` class description in `worddata.h` and the implementation of the operator to your `worddata.cpp` file.
 - a. The prototype should be similar to the prototype for the `==` operator.
 - b. The implementation should use the `strcmp` function to compare the words in the implicitly and explicitly passed objects. If the word in `*this` strictly follows the other word, the operator should return `true`; otherwise it should return `false`.
 10. Use `make` to compile your files and create the executable “`lab7`”.
 11. Execute the program. You should see 2 lists of word data elements, the first sorted in ascending order, the second in descending order.
 12. Create a copy of your `manywords.txt` test file from lab 6 as `Lab7.txt` in directory `Lab7.1`.
 13. Execute the program. Again, you should see 2 lists of word data elements, the first sorted in ascending order, the second in descending order. These lists should be much longer than the lists from the previous execution.
 14. When it is executing correctly, drop a copy of your application program into the CS 215 drop box as **`yourlastnameL7.app`**.

Part 2.

1. In your CS 215 directory, create a new subdirectory called `Lab7.2`.
2. Copy `worddata.h`, `worddata.cpp`, and `makefile` from `Lab7.1` to `Lab7.2`.
3. Remove all references to `sorts.h` from your new `makefile`:

```
lab7 : application.o worddata.o
      g++ -o lab7 application.o worddata.o

application.o : application.cpp worddata.h
      g++ -c application.cpp

worddata.o : worddata.cpp worddata.h
      g++ -c worddata.cpp

clean :
      rm -f core.* *.o lab7
```

4. Enter the following program as application.cpp:

```
// Lab 7 Part 2 application program
// Author: Dr. Watts
// Data: Spring 2009
// Description: This program is designed to test the WordData class when
//              it is used in a vector. The vector will also be sorted
//              using the sort in the algorithm library.

#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include "worddata.h"

using namespace std;

int main (int argc, char * argv [])
{
    // Test for existence of file name argument
    if (argc < 2)
    {
        cerr << "Usage: " << argv[0] << " <file-name>\n";
        exit (1);
    }

    // Test to see if file was opened
    ifstream ins (argv[1]);
    if (ins.fail ())
    {
        cerr << "No file " << argv[1] << " found\n";
        exit (2);
    }
    vector<WordData> words;
    WordData one;

    // Read words from file into WordData objects and push them onto the end
    // of the vector
    while (ins >> one)
        words.push_back (one);
    ins.close();

    // Sort the WordData object vector into ascending order
    sort (words.begin(), words.end());

    // Print the vector of Word Data objects using an index
    for (int i = 0; i < words.size(); i++)
        cout << words[i] << endl;
    cout << endl << endl;

    // Sort the WordData object vector into descending order
    sort (words.rbegin(), words.rend());

    // Print the vector of Word Data objects using an iterator
    for (vector<WordData>::iterator itr = words.begin();
         itr != words.end(); itr++)
        cout << *itr << endl;
    cout << endl << endl;

    return 0;
}
```

5. Using the make utility, try to compile your program. You should simply enter “make” at the Linux prompt.
6. Your program probably did not compile. If you carefully parse the error messages, you will see the statement “no match for operator <”; the class needs a less-than (<) operator for the algorithm sort function to compile. Add a less-than operator prototype to the WordData class description in worddata.h and the implementation of the operator to your worddata.cpp file.
 - a. The prototype should be similar to the prototype for the == operator.
 - b. The implementation should use the strcmp function to compare the words in the implicit and explicitly passed objects. If the word in *this strictly follows the other word, the operator should return true; otherwise it should return false.
7. Try executing your program without designating an input file:
lab7
The program should produce an error message and exit.
8. Try executing your program with a non-existent input file:
lab7 words.txt
The program should produce an error message and exit.
9. Copy the words.txt and manywords.txt test files from lab 6 to Lab7.2.
10. Execute the program using words.txt as the input file:
lab7 words.txt
Again, you should see 2 lists of word data elements, the first sorted in ascending order, the second in descending order.
11. Execute the program using manywords.txt as the input file:
lab7 manywords.txt
The lists produced by this execution of lab7 should be much longer than the lists from the previous execution.
12. When the program is executing correctly, drop a copy of your worddata header and implementation files into the CS 215 drop box as **yourlastnameL7.h** and **yourlastnameL7.cpp** respectively.