CSI 1430
Week 13 Resource

Colin Burdine

4/11/21

Major Topics:

1. STL Containers
2. Operations on Objects

Keywords: *searching, objects, STL containers*

Reminder: Group tutoring sessions for this course will be held online every Monday from 7:00pm to 8:00pm CDT. For more information on how to sign up for these sessions, go to: https://www.baylor.edu/tutoring.

1 Notes on Major Topics:

1.1 STL Containers

Last week we discussed some methods for searching and sorting arrays of elements using methods such as linear search, binary search, bubblesort, insertionsort, etc. This week we will briefly examine some of the built-in methods which we can use to search and sort data structures of types, whether those types are C++ primitives (i.e. `int`, `double`) or even objects. The way we can accomplish this in C++ is through the *standard template library (STL)*. We have already encountered one data structure provided by the STL, namely the `vector` class. For a list of data structures supplied by the STL, you can examine the documentation here. These containers are quite useful because they are *template types* in C++, meaning we can adapt their implementation based on the *types* that they contain. For example, we can create both instances of `vector<int>` and `vector<double>` and expect similar behavior for element access and modification (though the return types and

certain parameters may vary with the type).

One operation that the STL provides for (almost) all containers is *iteration*, which allows us to perform some operation for every element in a container. Previously, we achieved this through a `for` loop using an index variable:

```
vector<int> myIntVector = ...;

// iterate over a vector with a for loop:
for(int i = 0; i < myIntVector.size(); ++i){
    int element = myIntVector.at(i);

    /* do something with the element here */
}
```

Using STL containers, we do the exact same thing but with cleaner code using the *range-based for loop*, which is a feature introduced in C++11 (you read more about this in the docs here). We can used range-based for loops to access elements of a container by value or by reference with the following syntax:

```
vector<string> myStrVector = ...;

// iterate over a vector with a for loop (by value):
for(string element : myStrVector){

    /* do something with the element here */
}

// iterate over a vector with a for loop (by reference):
for(string& element : myStrVector){

    /* do something with the element here */
}
```

The range-based for loop is one of the many convenient features provided by the STL. once again, I would implore you to read through the documentation, as there are many features documented there that are very useful in practice (but are not required knowledge for this course).

1.2 Operations on Objects

One feature of STL containers is that if we define certain built-in operations associated with objects (such as `operator==` and `operator<`), we can perform operations such as searching and sorting. In particular, we can show that **it is sufficient to define only `operator<` in order for STL containers to be able to sort any type**! This is a

consequence of the fact that if we can compute the boolean value of $(a < b)$, the we can define equality using the following identity:

$$(a == b) \quad \text{is equivalent to} \quad !(a < b)\ \&\&\ !(b < a)$$

Searching Objects

So, in order for the STL to know how to sort a vector of objects, we simply need to define `operator<` for a given type, and then make a call to the `sort()` function in the `<algorithm>` library. For example, if we define an `Event` class representing some real-world event taking place in a given year, month and day, we can create a vector of `Event`s and sort them chronologically as follows:

```
#include <algorithm> // needed for sort()
#include <vector>
#include <string>
using namespace std;

// simple Event struct:
struct Event {
    string name;
    int year, month, day;
};

// compare Events chronologically:
bool operator<(Event& l, Event& r){
    bool lt = false;
    if(l.year != r.year){
        lt = (l.year < r.year);
    } else if(l.month < r.month){
        lt = (l.month < r.month);
    } else {
        lt = (l.day < r.day);
    }
    return lt;
}

int main(){

    // create a vector of Events:
    vector<Event> timeline = vector<Event>();

    // sort Events chronologically:
    sort(timeline.begin(), timeline.end());
    return 0;
}
```

2 Programming Example Problem

To illustrate how we can use searching and sorting with the STL in C++, consider the following example programming problem:

Create a struct or class that represents a `word`. A `word` is simply a string that can be compared with other strings in a case-insensitive manner. Any two `word`s are considered equal if and only if they are case-insensitively equal.

Write a program that reads a long sequence of words from the console terminated by a string simply consisting of the `#` symbol. The program should sort the words alphabetically and print them (one per line) to the console, excluding the last `"#"` string.

My solution to this problem is on the next page. You can look at my answer if you are stuck, but I would strongly encourage you to sketch out a design first, and then proceed to write your solution in C++. This exercise is intended to check your understanding of the concepts from this week.

3 Closing Remarks

I hope that the resources I shared with you this semester are helpful and efficient in addressing your academic needs. As we approach the end of the fall semester, I would like to let you know that all resources for the course content this semester can be found here: https://www.baylor.edu/support_programs/index.php?id=967950

Group tutoring sessions for this course will be held online every Monday evening from 7:00pm to 8:00pm CT. For more information on how to sign up for these sessions, go to: https://www.baylor.edu/tutoring.

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

// Word struct:
struct Word {
    string text; // original text:
    string lcText; // lowercase text:

    Word(string w){
        text = w;
        lcText = w;

        // convert to lower case:
        for(char& c : lcText){
            c = tolower(c);
        }
    }
};

// compare words based on lower case text:
bool operator<(Word& l, Word& r){
    return l.lcText < r.lcText;
}

// read words and sort case-insensitively:
int main(){

    vector<Word> words = vector<Word>();
    string line;

    while(cin >> line && line != "#"){
        words.push_back(Word(line));
    }

    // sort words:
    sort(words.begin(), words.end());

    // print out sorted words (in original case):
    for(Word& w : words){
        cout << w.text << endl;
    }

    return 0;
}
```