

Collection Classes (Part 1)

Collection Classes (Part 1: Vectors, Grids, Stacks, and Queues)

Eric Roberts
CS 106B
January 12, 2015

Outline

1. Introduce the idea of collection classes
2. Introduce the **Vector** class
3. Use vectors to read an entire file
4. Introduce the **Grid** class
5. Introduce the **Stack** class
6. Use stacks to balance parentheses
7. Introduce the **Queue** class

The Collection Classes

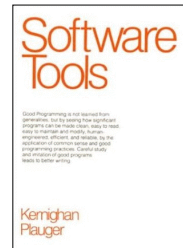
- For the rest of this week, we will be learning about the classes in Chapter 5. These classes contain other objects and are called *container* or *collection classes*.

Vector	Grid	Stack	Queue	Map	Set	Lexicon
--------	------	-------	-------	-----	-----	---------

- Here are some general guidelines for using these classes:
 - These classes represent *abstract data types* whose details are hidden.
 - Each class (except **Lexicon**) requires type parameters.
 - Declaring variables of these types always invokes a *constructor*.
 - Any memory for these objects is freed when its declaration scope ends.
 - Assigning one value to another *copies* the entire structure.
 - To avoid copying, these structures are usually passed by reference.

ADTs as Software Tools

- Over the relatively short history of software development, one of the clear trends is the increasing power of the tools available to you as a programmer.
- One of the best explanations of the importance of tools is the book *Software Tools* by Brian Kernighan and P. J. Plauger. Even though it was published in 1976, its value and relevance have not diminished over time.
- The primary theme of the book is that the best way to extend your reach in programming is to build on the tools of others.



Template Classes

- The collection classes are implemented as *template classes*, which make it possible for an entire family of classes to share the same code.
- Instead of using the class name alone, the collection classes require a type parameter that specifies the element type. For example, **Vector<int>** represents a vector of integers. Similarly, **Grid<char>** represents a two-dimensional array of characters.
- It is possible to nest classes, so that, for example, you could use the following definition to represent a list of chess positions:

```
Vector<Grid<char>> chessPositions;
```

Constructors for the **Vector<type>** Class

Vector<type> vec; Initializes an empty vector of the specified element type.
Vector<type> vec(n); Initializes a vector with n elements all set to the default value of the type.
Vector<type> vec(n, value); Initializes a vector with n elements all set to value .

Methods in the **vector<type>** Class

vec.size()	Returns the number of elements in the vector.
vec.isEmpty()	Returns true if the vector is empty.
vec.get(i) <i>or</i> vec[i]	Returns the i^{th} element of the vector.
vec.set(i, value) <i>or</i> vec[i] = value;	Sets the i^{th} element of the vector to value .
vec.add(value) <i>or</i> vec += value;	Adds a new element to the end of the vector.
vec.insert(index, value)	Inserts the value before the specified index position.
vec.remove(index)	Removes the element at the specified index.
vec.clear()	Removes all elements from the vector.

The **readEntireFile** Function

```
/*
 * Function: readEntireFile
 * Usage: readEntireFile(is, lines);
 * -----
 * Reads the entire contents of the specified input stream
 * into the string vector lines. The client is responsible
 * for opening and closing the stream
 */

void readEntireFile(istream & is, Vector<string> & lines) {
    lines.clear();
    string line;
    while (getline(is, line)) {
        lines.add(line);
    }
}
```

Methods in the **Grid<type>** Class

Grid<type> grid(nrows, ncols);	Constructs a grid with the specified dimensions.
grid.numRows()	Returns the number of rows in the grid.
grid.numCols()	Returns the number of columns in the grid.
grid[i][j]	Selects the element in the i^{th} row and j^{th} column.
resize(nrows, ncols)	Changes the dimensions of the grid and clears any previous contents.
inBounds(row, col)	Returns true if the specified row and column position is within the grid.

The Stack Metaphor

- A **stack** is a data structure in which the elements are accessible only in a **last-in/first-out** order.
- The fundamental operations on a stack are **push**, which adds a new value to the top of the stack, and **pop**, which removes and returns the top value.
- One of the most common metaphors for the stack concept is a spring-loaded storage tray for dishes. Adding a new dish to the stack pushes any previous dishes downward. Taking the top dish away allows the dishes to pop back up.



Methods in the **Stack<type>** Class

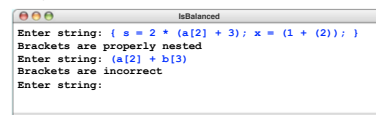
stack.size()	Returns the number of values pushed onto the stack.
stack.isEmpty()	Returns true if the stack is empty.
stack.push(value)	Pushes a new value onto the stack.
stack.pop()	Removes and returns the top value from the stack.
stack.peak()	Returns the top value from the stack without removing it.
stack.clear()	Removes all values from the stack.

Exercise: Stack Processing

Write a C++ program that checks whether the bracketing operators (parentheses, brackets, and curly braces) in a string are properly matched. As an example of proper matching, consider the string

```
{ s = 2 * (a[2] + 3); x = (1 + (2)); }
```

If you go through the string carefully, you discover that all the bracketing operators are correctly nested, with each open parenthesis matched by a close parenthesis, each open bracket matched by a close bracket, and so on.



Methods in the `Queue<type>` Class

<code>queue.size()</code>	Returns the number of values in the queue.
<code>queue.isEmpty()</code>	Returns <code>true</code> if the queue is empty.
<code>queue.enqueue(value)</code>	Adds a new value to the end of the queue (which is called its <i>tail</i>).
<code>queue.dequeue()</code>	Removes and returns the value at the front of the queue (which is called its <i>head</i>).
<code>queue.peek()</code>	Returns the value at the head of the queue without removing it.
<code>queue.clear()</code>	Removes all values from the queue.

The LaIR Queue

The LaIR Queue

The LaIR Queue

The LaIR Queue

Comparing Stacks and Queues

