Today’s Topics

Abstract Data Types
- What is an ADT?
- Vector
- Grid
- Next time: Stack, Queue
ADTs
ADTs

- Programming language independent models of common containers
- They encompass not only the nature of the data, but ways of accessing it
- They form a rich **vocabulary** of **nouns** and **verbs**, often drawing on analogies to make their use intuitive, and to give code written in them a certain **literary** quality
Vector

- ADT abstraction similar to an array or list
- Many languages have a version of this (remember, ADTs are conceptual abstractions that are language-independent)
- We will use Stanford library Vector (there is also an STL vector, which will not use—watch out for capitalization)
- We declare one like this: `Vector<string> lines;`
- This syntax is called template syntax
  - Template containers must be homogenous (all items the same type)
  - The type goes in the <> after the class name Vector
  - // initialize a vector containing 5 integers
  - `Vector<int> nums {42, 17, -6, 0, 28};`
  -
<table>
<thead>
<tr>
<th>index</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>42</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>-6</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>28</td>
</tr>
</tbody>
</table>

#include "vector.h"
Vector

- Declaring a Vector:
  - `Vector<int> pset3Scores;`
  - `Vector<double> measurementsData;`
  - `Vector<Vector<int>> allAssignmentScores;`

- Using a Vector:
  - `pset3Scores.add(98);`
  - `pset3Scores.add(85);`
  - `pset3Scores.add(92);`
  - `cout << pset3Scores[0]; // prints 98`
  - `cout << pset3Scores[pset3Scores.size() - 1]; // prints 92`
*Performance Warning* Vector insert/remove

- **v.insert(2, 42)**
  - shift elements right to make room for the new element

- **v.remove(1)**
  - shift elements left to cover the space left by the removed element

- These operations are slower the more elements they need to shift
Your turn: Vector

- Compare how many times we write a number into one “box” of the Vector, in these two codes. (as shown, v starts out empty in both cases)

<table>
<thead>
<tr>
<th>Vector&lt;int&gt; v;</th>
<th>Vector&lt;int&gt; v;</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (int i = 0; i &lt; 100; i++) { v.insert(0, i); // OPTION 1 }</td>
<td>for (int i = 0; i &lt; 100; i++) { v.add(i); // OPTION 2 }</td>
</tr>
</tbody>
</table>

A. They both write in a box about the same number of times
B. One writes about 2x as many times as the other
C. One writes about 10x as many times as the other
D. Something else!

Since B and C don’t say which option writes more than the other, if you pick one of those, be sure to address that in your group discussion!
Grid container

ESSENTIALLY A MATRIX
(LINEAR ALGEBRA FANS CELEBRATE NOW)
Grid

- ADT abstraction similar to an array of arrays (matrix)
- Many languages have a version of this
  - (remember, ADTs are conceptual abstractions that are language-independent)

- In C++ we declare one like this:

```cpp
Grid<int> chessboard;
Grid<int> image;
Grid<double> realMatrix;
```
void printMe(const Grid<int>& grid, int row, int col) {
    for (int r = row - 1; r <= row + 1; r++) {
        for (int c = col - 1; c <= col + 1; c++) {
            if (grid.inBounds(r, c)) {
                cout << grid[r][c] << " ";
            }
        }
    }
    cout << endl;
}

How many 0’s does this print with input row = 2, col = 3? (and grid as shown on right)

(A) None or 1
(B) 2 or 3
(C) 4 or 5
(D) 6 or 7
Handy loop idiom: iterating over “neighbors” in a Grid

void printNeighbors(const Grid<int>& grid, int row, int col) {
    for (int r = row - 1; r <= row + 1; r++) {
        for (int c = col - 1; c <= col + 1; c++) {
            if (grid.inBounds(r, c)) {
                cout << grid[r][c] << " ";
            }
        }
    }
    cout << endl;
}

These nested for loops generate all the pairs in the cross product {-1,0,1} x {-1,0,1}, and we can add these as offsets to a (r,c) coordinate to generate all the neighbors (note: often want to test for and exclude the (0,0) offset, which is “myself” not a neighbor)
Stacks
template<typename ValueType> class Stack {
    public:
        Stack();
        ~Stack();
        int size();
        bool isEmpty();
        void clear();
        void push(ValueType value);
        ValueType pop();
        ValueType peek();
        string toString();

    private:
        -Redacted-

};

Source: http://www.flickr.com/photos/35237093334@N01/4094655578/
Author: http://www.flickr.com/people/35237093334@N01 Peter Kazanjian

Stanford University
Using a Stack to buffer file input

```cpp
void mystery(ifstream& infile) {
    Stack<string> lines;
    string line;
    while (getline(infile, line)) {
        lines.push(line);
    }
    infile.close();
    while (!lines.isEmpty()) {
        cout << lines.pop() << endl;
    }
}
```

What does this code do?

A. Prints all lines of a file to cout
B. Prints only the first line of a file to cout
C. Prints only the last line of a file to cout
D. Prints all lines of a file to cout in reverse
E. All/ none/ more
Using a Stack to buffer file input

```cpp
void mystery(ifstream& infile) {
    Stack<string> lines;
    string line;
    while (getline(infile, line)) {
        lines.push(line);
    }
    infile.close();
    while (!lines.isEmpty()) {
        cout << lines.pop() << endl;
    }
}
```

What does this code do?

A. Prints all lines of a file to cout
B. Prints only the first line of a file to cout
C. Prints only the last line of a file to cout
D. Prints all lines of a file to cout in reverse
E. All/ none/ more

Why do I need Stack? I could have done that with a Vector!
void mystery(ifstream& infile) {
    Stack<string> lines;  // Use a stack
    string line;
    while (getline(infile, line)) {
        lines.push(line);
    }
    infile.close();
    while (!lines.isEmpty()) {
        cout << lines.pop() << endl;
    }
}

Vector<string> lines;  // Use a vector
lines.add(line);
cout << lines[lines.size() - 1] << endl;
lines.remove(lines.size() - 1);
Vector version

```cpp
void mystery(ifstream& infile) {
    Vector<string> lines;
    string line;
    while (getline(infile, line)) {
        lines.add(line);
    }
    infile.close();
    while (!lines.isEmpty()) {
        cout << lines[lines.size() - 1] << endl;
        lines.remove(lines.size() - 1);
    }
}
```

This code isn’t terrible, but it is harder to read quickly, and is probably more error prone.

For example, it would be easy to forget the “-1” when you remove “lines.size()-1”.

Stanford University
Applications of Stacks

We’ve seen one (buffering input and giving it back in reverse—LIFO—order). What else are Stacks good for?
Operator Precedence and Syntax Trees

Ignoring operator precedence rules, how many distinct results are there to the following arithmetic expression?

- $3 \times 3 + 3 \times 3$
Reverse Polish Notation

Ambiguities don’t exist in RPN
Also called “postfix” because the operator goes after the operands

Postfix (RPN):
- $4 \times 3 * 4 \times 3 * +$

Equivalent Infix:
- $(4\times3) + (4\times3)$

Reverse Polish Notation

Ambiguities don't exist in RPN. Also called "postfix because the operator goes after the operands.

Postfix (RPN):
- $4 \times 3 + 4 \times 3$

Equivalent Infix:
- $(4 \times 3) + (4 \times 3)$

[Image of Hewlett-Packard 48GX Scientific Graphing Calculator]

[Image of a girl using a computer]
Reverse Polish Notation

This postfix expression:

- \(4 \ 3 \ * \ 7 \ 2 \ 5 \ * \ + \ +\)

Is equivalent to this infix expression:

A. \(((4*3) + (7*2)) + 5\)
B. \((4*3) + ((7+2) + 5)\)
C. \((4*3) + (7 + (2*5))\)
D. Other/none/more than one
Stacks and RPN

- Evaluate this expression with the help of a stack
  - Encounter a **number**? **PUSH** it
  - Encounter an **operator**? **POP** two numbers and **PUSH** result

- $4 \times 3 + 7 - 2 + 5 \times +$

Contents of the stack, reading from top down:
(A) 7, 12
(C) 10, 7, 12
(D) 10, 5, 2, 7, 12
(E) Other
Evaluate this expression with the help of a stack

- **Encounter a number**: `PUSH` it
- **Encounter an operator**: `POP` two numbers and `PUSH` result

```
/* Note: this code assumes numbers are all 1 digit long */
bool evaluate(Stack<int>& memory, string instruction) {
    for (int i = 0; i < instruction.size(); i++) {
        if (isdigit(instruction[i])) {
            int value = instruction[i] - '0'; //convert char->int
            memory.push(value);
        } else if (isSupportedOperator(instruction[i])) {
            if (memory.size() < 2) return false;
            int second = memory.pop();
            int first = memory.pop();
            int result = compute(first, instruction[i], second);
            memory.push(result);
        } else {
            return false;
        }
    }
    return memory.size() == 1; //validity check
}
```

43*725*++