CS 106B, Lecture 3
Vector and Grid

reading:
Programming Abstractions in C++, Chapter 4-5
Plan for Today

- Learn about two new "ADTs" or collections
  - Vector: a data structure for representing lists
  - Grid: a data structure ideal for representing two dimensional information
**STL vs. Stanford**

- **collection**: an object that stores data; a.k.a. "data structure"
  - the objects stored are called **elements**.
  - Also known as "ADTs" – abstract data types

- **Standard Template Library (STL)**: C++ built in standard library of collections.
  - **vector**, **map**, **list**, ...
  - Powerful but somewhat hard to use for new coders (messy syntax) – take 106L!

- **Stanford C++ library (SPL)**: Custom library of collections made for use in CS 106B/X.
  - **Vector**, **Grid**, **Stack**, **Queue**, **Set**, **Map**, ...
  - Similar to STL, but simpler interface and error messages.
  - Note the capitalized first letter
Vectors (Lists)

#include "vector.h"

• vector (aka list): a collection of elements with 0-based indexes
  – like a dynamically-resizing array (Java ArrayList or Python list)
  – Include the type of elements in the <> brackets

// initialize a vector containing 5 integers
//    index  0  1  2  3  4
Vector<int> nums {42, 17, -6, 0, 28};

Vector<string> names; // {}
names.add("Ashley");   // {"Ashley"}
names.add("Shreya");  // {"Ashley", "Shreya"}
names.insert(0, "Ed"); // {"Ed", "Ashley", "Shreya"}
Why not arrays?

// actual arrays in C++ are mostly awful
int nums[5] {42, 17, -6, 0, 28}; // no

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>42</td>
<td>17</td>
<td>-6</td>
<td>0</td>
<td>28</td>
</tr>
</tbody>
</table>

• **Arrays have fixed size** and cannot be easily resized.
  – In C++, an array doesn't even **know** its size. (no .length field)

• **C++ lets you index out of the array bounds** (garbage memory) **without** necessarily crashing or warning.

• **An array does not support many operations** that you'd want:
  – inserting/deleting elements into the front/middle/back of the array, reversing, sorting the elements, searching for a given value ...
## Vector members (5.1)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>v.add(value);</code> or <code>v += value;</code> or <code>v += v1, v2, ..., vN;</code></td>
<td>appends value(s) at end of vector</td>
</tr>
<tr>
<td><code>v.clear();</code></td>
<td>removes all elements</td>
</tr>
<tr>
<td><code>v[i]</code> or <code>v.get(i)</code></td>
<td>returns the value at given index</td>
</tr>
<tr>
<td><code>v.insert(i, value);</code></td>
<td>inserts given value just before the given index, shifting subsequent values to the right</td>
</tr>
<tr>
<td><code>v.isEmpty()</code></td>
<td>returns true if the vector contains no elements</td>
</tr>
<tr>
<td><code>v.remove(i);</code></td>
<td>removes/returns value at given index, shifting subsequent values to the left</td>
</tr>
<tr>
<td><code>v[i] = value;</code> or <code>v.set(i, value);</code></td>
<td>replaces value at given index</td>
</tr>
<tr>
<td><code>v.subList(start, length)</code></td>
<td>returns new vector of sub-range of indexes</td>
</tr>
<tr>
<td><code>v.size()</code></td>
<td>returns the number of elements in vector</td>
</tr>
<tr>
<td><code>v.toString()</code></td>
<td>returns a string representation of the vector such as <code>{3, 42, -7, 15}</code></td>
</tr>
<tr>
<td><code>ostr &lt;&lt; v</code></td>
<td>prints v to given output stream (e.g. <code>cout &lt;&lt; v</code>)</td>
</tr>
</tbody>
</table>
Iterating over a vector

Vector<string> names {"Ed", "Hal", "Sue"};

for (int i = 0; i < names.size(); i++) {
    cout << names[i] << endl; // for loop
} // Ed Hal Sue

for (int i = names.size() - 1; i >= 0; i--) {
    cout << names[i] << endl; // for loop, backward
} // Sue Hal Ed

for (string name : names) {
    cout << name << endl; // "for-each" loop
} // Ed Hal Sue

// Can't edit (insert/delete) in for-each loop
Vector insert/remove

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

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

index | 0 | 1 | 2 | 3 | 4 | 5 |
value | 3 | 8 | 42 | 9 | 7 | 5 |

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.)
Announcements

• Assignment 0 due Friday
  – Fill out the exam survey by **5PM on Friday**
  – If you need help with Qt stop by LaIR or Ashley's office hours (last chance to get help is 12:15PM on Thursday)

• Sections start today! Should have received an email from cs198@cs.stanford.edu
  – You can switch your section or sign up late at cs198.stanford.edu
  – Email **Shreya** at shreya@cs.stanford.edu if you were assigned a different section than your partner
Grid (5.1)

#include "grid.h"

• like a 2D array, but more powerful
• Good for board games, matrices, images, city maps, etc.
• must specify element type in < > (a template or a type parameter)

// constructing a Grid
Grid<int> matrix(3, 4);
matrix[0][0] = 75;
...

// or specify elements in {}
Grid<int> matrix = {
    {75, 61, 83, 71},
    {94, 89, 98, 100},
    {63, 54, 51, 49}
};

<table>
<thead>
<tr>
<th>row</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>75</td>
<td>61</td>
<td>83</td>
<td>71</td>
</tr>
<tr>
<td>1</td>
<td>94</td>
<td>89</td>
<td>98</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>63</td>
<td>54</td>
<td>51</td>
<td>49</td>
</tr>
</tbody>
</table>
Grid members (5.1)*

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Grid&lt;type&gt; name(r, c);</code></td>
<td>create grid with given number of rows/cols; empty 0x0 grid if omitted</td>
</tr>
<tr>
<td><code>Grid&lt;type&gt; name;</code></td>
<td></td>
</tr>
<tr>
<td><code>g[r][c]</code> or <code>g.get(r, c)</code></td>
<td>returns value at given row/col</td>
</tr>
<tr>
<td><code>g.fill(value);</code></td>
<td>set every cell to store the given value</td>
</tr>
<tr>
<td><code>g.inBounds(r, c)</code></td>
<td>returns true if given position is in the grid</td>
</tr>
<tr>
<td><code>g.numCols()</code> or <code>g.width()</code></td>
<td>returns number of columns</td>
</tr>
<tr>
<td><code>g.numRows()</code> or <code>g.height()</code></td>
<td>returns number of rows</td>
</tr>
<tr>
<td><code>g.resize(nRows, nCols);</code></td>
<td>resizes grid to new size, discarding old contents</td>
</tr>
<tr>
<td><code>g[r][c] = value;</code> or <code>g.set(r, c, value);</code></td>
<td>stores value at given row/col</td>
</tr>
<tr>
<td><code>g.toString()</code></td>
<td>returns a string representation of the grid such as &quot;{{3, 42}, {-7, 1}, {5, 19}}&quot;</td>
</tr>
<tr>
<td><code>ostr &lt;&lt; g</code></td>
<td>prints, e.g. {{3, 42}, {-7, 1}, {5, 19}}</td>
</tr>
</tbody>
</table>

* (a partial list; see [http://stanford.edu/~stepp/cppdoc/](http://stanford.edu/~stepp/cppdoc/))
Looping over a grid

• Row-major order:
  for (int r = 0; r < grid.numRows(); r++) {
    for (int c = 0; c < grid.numCols(); c++) {
      do something with grid[r][c];
    }
  }

  // "for-each" loop (also row-major)
  for (int value : grid) {
    do something with value;
  }

• Column-major order:
  for (int c = 0; c < grid.numCols(); c++) {
    for (int r = 0; r < grid.numRows(); r++) {
      do something with grid[r][c];
    }
  }
Grid as parameter

• When a Grid is passed by value, C++ makes a copy of its contents.
  – Copying is slow; you should **pass by reference** with &
  – If the code won't modify the grid, also pass it as **const**

// Which one is best?
A) int computeSum(Grid<int> g) {
B) int computeSum(Grid<int>& g) {
C) int computeSum(const Grid<int> g) {
D) int computeSum(const Grid<int>& g) {

// Which one is best?
A) void invert(Grid<double> matrix) {
B) void invert(Grid<double>& matrix) {
C) void invert(const Grid<double> matrix) {
D) void invert(const Grid<double>& matrix) {
Grid exercise

- Write a function `knightCanMove` that accepts a grid and two row/column pairs \((r_1, c_1), (r_2, c_2)\) as parameters, and returns `true` if there is a knight at chess board square \((r_1, c_1)\) that can legally move to empty square \((r_2, c_2)\).
- Recall that a knight makes an "L" shaped move, going 2 squares in one dimension and 1 square in the other.
- `knightCanMove(board, 1, 2, 2, 4)` returns `true`
bool knightCanMove(Grid<string>& board, int r1, int c1, int r2, int c2) {
    if (!board.inBounds(r1, c1) || !board.inBounds(r2, c2)) {
        return false;
    }
    if (board[r1][c1] != "knight" || board[r2][c2] != ") {
        return false;
    }
    int dr = abs(r1 - r2);
    int dc = abs(c1 - c2);
    if(!(dr == 1 && dc == 2) || (dr == 2 && dc == 1)) {
        return false;
    }
    return true;
}
bool knightCanMove(Grid<string>& board, int r1, int c1, int r2, int c2) {
    int dr = abs(r1 - r2), dc = abs(c1 - c2);
    return board.inBounds(r1, c1) && board.inBounds(r2, c2)
        && board[r1][c1] == "knight" && board[r2][c2] == ""
        && ((dr == 1 && dc == 2) || (dr == 2 && dc == 1));
}
Overflow (extra) slides
#include <sstream>

• An istream lets you tokenize a string.

    // read specific word tokens from a string
    istringstream input("Jenny Smith 8675309");
    string first, last;
    int phone;
    input >> first >> last; // first="Jenny", last="Smith"
    input >> phone;        // 8675309

    // read all tokens from a string
    istringstream input2("To be or not to be");
    string word;
    while (input2 >> word) {
        cout << word << endl; // To \n be \n or \n not \n ...
    }
#include <sstream>

• An **ostringstream** lets you write output into a string buffer.
  – Use the **str** method to extract the string that was built.

```cpp
// produce a formatted string of output
int age = 42, iq = 95;
ostringstream output;
output << "Zoidberg's age is " << age << endl;
output << " and his IQ is " << iq << "!" << endl;
string result = output.str();

// result = "Zoidberg's age is 42\nand his IQ is 95!\n"
Bug: Mix lines/tokens

cout << "How old are you? ";
int age;
cin >> age;

cout << "And what's your name? ";
string name;
getline(cin, name);
cout << "Wow, " << name << " is " << age << "!" << endl;

// output:
// How old are you: 17
// And what's your name: Stuart
// Wow, is 17!

– Advice: Don't mix getline and >> on the same input stream.
– Advice: Always use Stanford getXxx methods to read from cin.
Exercise: inputStats2

• Write a function **inputStats2** that prints statistics about the data in a file. Example file, carroll.txt:

```
1 Beware the Jabberwock, my son,
2 the jaws that bite, the claws that catch,
3
4 Beware the JubJub bird and shun
5 the frumious bandersnatch.
```

• The call of inputStats2("carroll.txt"); should print:

```
Line 1: 30 chars, 5 words
Line 2: 41 chars, 8 words
Line 3: 0 chars, 0 words
Line 4: 31 chars, 6 words
Line 5: 26 chars, 3 words
longest = 41, average = 25.6
```
inputStats2 solution

/* Prints length/count statistics about data in the given file. */
void inputStats2(string filename) {
    ifstream input;
    input.open(filename);

    int lineCount = 0, longest = 0, totalChars = 0;
    string line;
    while (getline(input, line)) {
        lineCount++;
        totalChars += line.length();
        longest = max(longest, line.length());
        int wordCount = countWords(line);  // on next slide
        cout << "Line " << lineCount << " line length(): " << line.length()
            << " chars, " << wordCount << " words" << endl;
    }
    double average = (double) totalChars / lineCount;
    cout << longest = " " << longest
        << "", average = " " << average << endl;
}
inputStats2 solution

/* Returns the number of words in the given string. */
int countWords(string line) {
    istringstream words(line);
    int wordCount = 0;
    string word;
    while (words >> word) {
        wordCount++;
    }
    return wordCount;
}
#include <iomanip>
– helps produce formatted output, a la printf

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setw( (n) )</td>
<td>right-aligns next token in a field ( n ) chars wide</td>
</tr>
<tr>
<td>setfill( (ch) )</td>
<td>sets padding chars inserted by setw to the given char (default ' ')</td>
</tr>
<tr>
<td>setbase( (b) )</td>
<td>prints future numeric tokens in base-( b )</td>
</tr>
<tr>
<td>left, right</td>
<td>left- or right-aligns tokens if setw is used</td>
</tr>
<tr>
<td>setprecision( (d) )</td>
<td>prints future doubles with ( d ) digits after decimal</td>
</tr>
<tr>
<td>fixed</td>
<td>prints future doubles with a fixed number of digits</td>
</tr>
<tr>
<td>scientific</td>
<td>prints future doubles in scientific notation</td>
</tr>
</tbody>
</table>

```cpp
for (int i = 2; i <= 2000; i *= 10) {
    cout << left << setw(4) << i // 2 1.41
        << right << setw(8) << fixed // 20 4.47
        << setprecision(2) << sqrt(i) << endl; // 200 14.14
        << setprecision(2) << sqrt(i) << endl; // 2000 44.72
}
```
Exercise: Hours

• Given hours.txt of section leader hours worked, in this format:

<table>
<thead>
<tr>
<th>#</th>
<th>ID</th>
<th>Name</th>
<th>Hours</th>
<th>Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123</td>
<td>Alex</td>
<td>2 4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>Jessica</td>
<td>8.5 1.5</td>
<td>5 5 10 6</td>
</tr>
<tr>
<td>3</td>
<td>7289</td>
<td>Erik</td>
<td>3 6 4 4.68</td>
<td>4</td>
</tr>
</tbody>
</table>

• Write code to output hours worked by each SL in this format:

Alex (ID# 123) worked 10.0 hours (2.50/day)
Jessica (ID# 46) worked 36.0 hours (6.00/day)
Erik (ID# 7289) worked 21.7 hours (4.34/day)
/* This program computes the ... */
#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>
using namespace std;

int main() {
    ifstream input;
    input.open("hours.txt");

    string line;
    while (getline(input, line)) {
        // "7289 Erik 3 6 4 4.68 4"
        istringstream tokens(line);
        int id;       // 7289
        string name;  // "Erik"
        tokens >> id >> name;

        // rest of tokens are days
        double totalHours = 0.0;
        int days = 0;
        ...

        double hours;
        while (tokens >> hours) {
            totalHours += hours;
            days++;
        }

        // Erik (ID# 7289) worked
        // 21.7 hours (4.34/day)
        cout << left << setw(9) << name << " (ID#"
             << right << setw(5) << id << ") worked "
             << fixed << setprecision(1) << totalHours << " hours (" 
             << setprecision(2) << totalHours/days
             << "/day)" << endl;
    }
    return 0;
}