# 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.
  - **`V`**`ector`, **`G`**`rid`, **`S`**`tack`, **`Q`**`ueue`, **`S`**`et`, **`M`**`ap`, …
  - Similar to STL, but simpler interface and error messages.
  - Note the capitalized first letter

# Vectors (Lists)

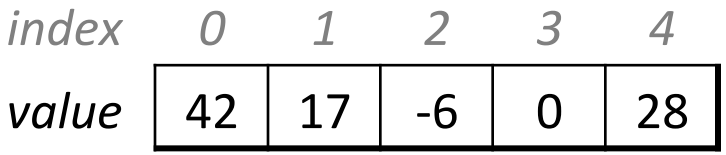
```
#include "vector.h"
```

- **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
```




- 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)

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

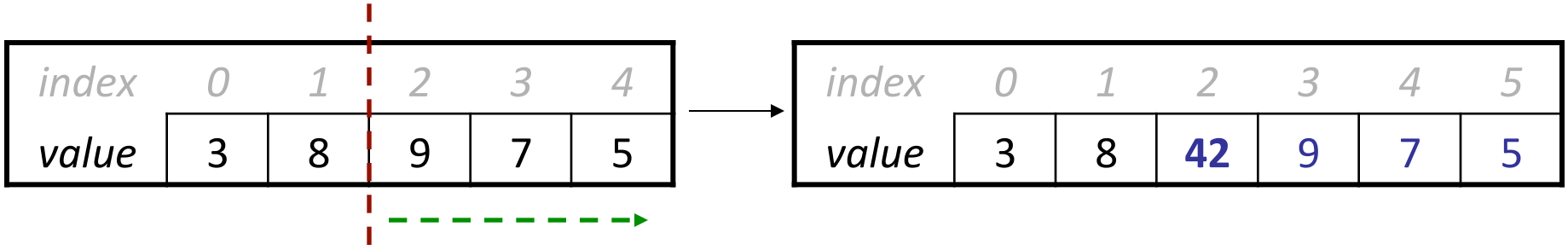
# Iterating over a vector

```
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

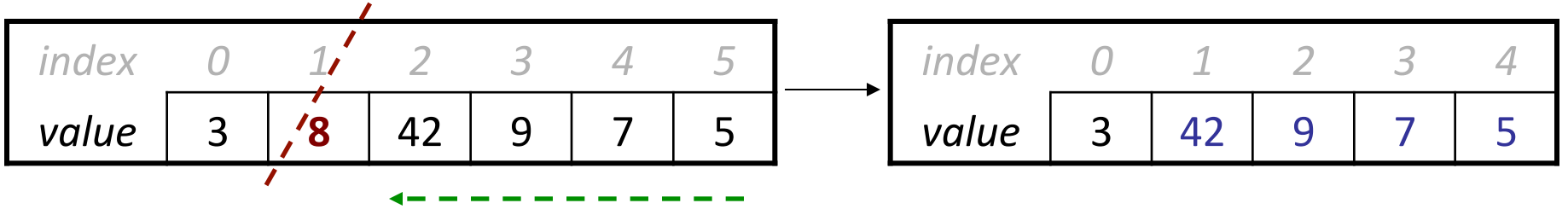
```
v.insert(2, 42);
```

- shift elements right to make room for the new element




```
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}
};
```

| row \ column | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 75 | 61 | 83 | 71 |
| 1 | 94 | 89 | 98 | 100 |
| 2 | 63 | 54 | 51 | 49 |

# Grid members (5.1)*

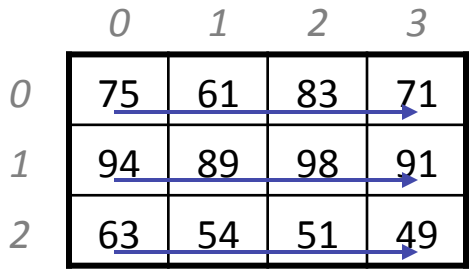
| | |
|---|---|
| `Grid<type> name(r, c);`<br>`Grid<type> name;` | create grid with given number of rows/cols; empty 0x0 grid if omitted |
| `g[r][c]` or `g.get(r, c)` | returns value at given row/col |
| `g.fill(value);` | set every cell to store the given value |
| `g.inBounds(r, c)` | returns `true`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

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

# Looping over a grid

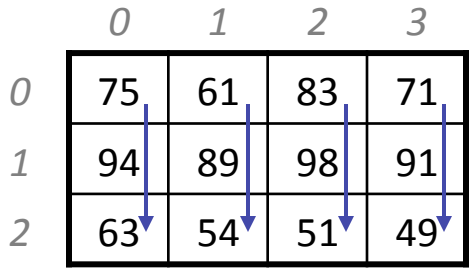
- Row-major order:

```
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

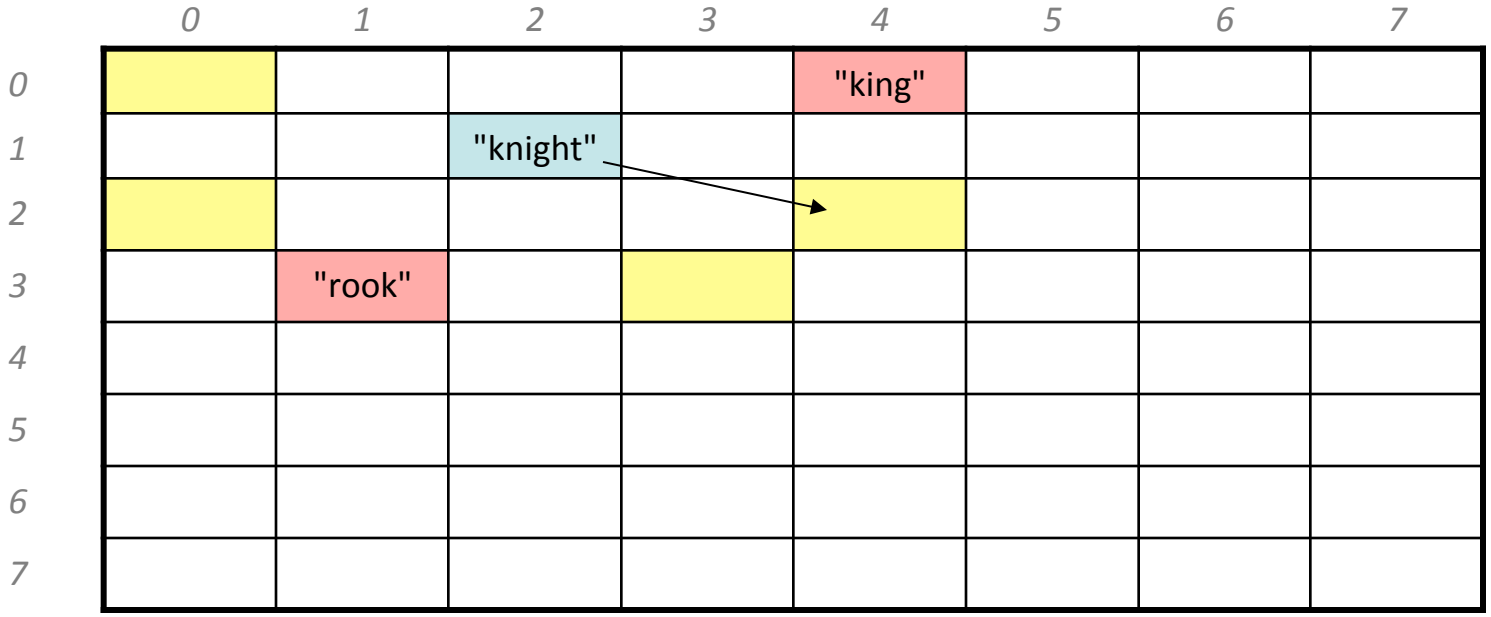
- When a `Grid` is passed by value, C++ makes a copy of its contents.
  - Copying is slow; you should **pass by reference** with **&**
  - If the code won't modify the grid, also pass it as **`const`**

```
// 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

knightCanMove

- Write a function **`knightCanMove`** that accepts a grid and two row/column pairs (*r1*, *c1*), (*r2*, *c2*) as parameters, and returns `true` if there is a knight at chess board square (*r1*, *c1*) that can legally move to empty square (*r2*, *c2*).
  - 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`



# Grid exercise solution

```
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;
}
```

# Grid solution 2

```
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

# istringstream

```
#include <sstream>
```

- An **istringstream** 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 ...
}
```

# ostringstream

```
#include <sstream>
```

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

```
// 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!
```

user input:

**17**\n
**Stuart**\n

– *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()
             << " 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;
}
```

# Formatted I/O

```
#include <iomanip>
```

– helps produce formatted output, a la `printf`

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

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

# Exercise: Hours

hoursWorked

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

```
1  123 Alex 3 2 4 1
2  46 Jessica 8.5 1.5 5 5 10 6
3  7289 Erik 3 6 4 4.68 4
```

- 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)
```

# Hours solution

```
/* 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;
}
```