Programming Abstractions

CS106B

Cynthia Lee
Today’s Topics

Abstract Data Types (ADTs)
- Vector
- Grid

C++ Functions and Parameters
- “Pass by reference” for containers like Vector and Grid

Fauxtoshop
- Assignment 1 orientation
- Grid practice
ADTs

VECTOR, GRID
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
- Many languages have a version of this
  - (remember, ADTs are conceptual abstractions that are language-independent)

- In C++ we declare one like this: `Vector<string> lines;`
- This syntax is called template syntax
  - Vectors can hold many things, but they all have to be the same type
  - The type goes in the <> after the class name Vector
    - `Vector<int> assignment3Scores;`
    - `Vector<double> measurementsData;`
    - `Vector<Vector<int>> allAssignmentScores;`
Vector

- Declaring a Vector:
  - `Vector<int> assignment3Scores;`
  - `Vector<double> measurementsData;`
  - `Vector<Vector<int>> allAssignmentScores;`

- Using a Vector:
  - `assignment3Scores.append(98);`
  - `assignment3Scores.append(85);`
  - `assignment3Scores.append(92);`
  - `cout << assignment3Scores[0]; // prints 98`
  - `cout << assignment3Scores[2]; // prints 92`
Parameter Passing in C++

IMPORTANT C++ DETAIL FOR WORKING WITH CONTAINERS LIKE VECTOR
int main(){
    int n = 5;
    foo(n);
    cout << n << endl;
    return 0;
}

void foo(int n) {
    n++;
}
"Pass by reference"

```c++
int main()
{
    int n = 5;
    foo(n);
    cout << n << endl;
    return 0;
}

void foo(int &n) {
    n++;
}
```

What is printed?

A. 5  
B. 6  
C. Error or something else
Often used when you would want to “return” several values from a function (but there is only one return value allowed)

#include "random.h" //Stanford library for random numbers

void pickLotto(int& first, int& second, int& third) {
    first = randomInteger(0,10);
    second = randomInteger(0,10);
    third = randomInteger(0,10);
}

int main(){
    int lotto1 = 0;
    int lotto2 = 0;
    int lotto3 = 0;
    pickLotto(lotto1, lotto2, lotto3);
    cout << lotto1 << " " << lotto2 << " " << lotto3 << endl;
    return 0;
}
Pass by reference

**Always** pass containers like Vector (and Grid, which we’ll see next) by reference in this class!

- For efficiency reasons—don’t want to make a big copy every time!
- If you aren’t interested in actually changing the object (i.e., by reference is purely for efficiency), then this is a little bit overkill because it does allow the object to be changed
  › Use “const” to communicate that you won’t change the structure

```cpp
void printFirst(const Vector<int>& input) {
    cout << input[0] << endl; // "read-only" access
    // input[0] = 5; // this would not be allowed by compiler
}
```
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;
```
Code example: Fauxtoshop

“FAUX”—PRONOUNCED “FOH”—IS A FRENCH WORD THAT MEANS FAKE OR COUNTERFEIT
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 (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

```cpp
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 (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)