Programming Abstractions

CS106B

Cynthia Lee

Stanford University

Today's Topics

Abstract Data Types

- What is an ADT?
- Vector ADT
- Grid ADT
- *Next time:* Stack, Queue ADTs

ADTs

ADTs = “Abstract Data Types”

- **Language-independent models of common containers**
  - In other words, we try to focus on the aspects of the ADT that transcend whether we happen to be using it in C++, Java, Python, or some other language

- ADTs encompass both the nature of the data and ways of accessing it

- ADTs form a rich **vocabulary** of **nouns** (nature of the data) and **verbs** (ways of accessing it), often drawing on analogies to make their use intuitive
  - Skillful ADT use gives code added readability!

Types of ADTs

- When we say the “nature of the data,” we mean questions like:
  - Is the data **ordered** in some way?
    - Could/should you be able to say about the data that this element is the “first” one, and this other piece is the “tenth” one?
  - Is the data **paired or matched** in some way?
    - Could/should you be able to say about the data that this element A goes with element B (not D), and this element C goes with element D (not B)?
- When we say “ways of accessing it,” we mean questions like:
  - Is it important to be **able to add and remove data** during the course of use, or do we assume we have the “final” version from the beginning?
  - Is it important to be able to **search for any piece of data** in the collection, or is it enough to always take the first available one?

Types of ADTs

- When we say the "nature of the data," w… …ean questions like:
  - Is the data **ordered** in some way?
    - Could/should you be able to say a… …t the data… …hat this element is the "first" one, and this other piece is the "tenth" … …?
  - Is the data **paired or matched** in some way?
    - Could/should you be able to say about the da… …hat this element A goes with element B (not D), and this element C goes with element D (not B)?

We'll talk about ADTs in this category today and Wednesday.

We'll talk about ADTs in this category on Friday.

- When we say "ways of accessing it," we mean questions like:
  - Is it important to be **able to add and remove data** during the course of use, or do we assume we have the "final" version from the beginning?
  - Is it important to be able to **search for any piece of data** in the collection, or is it enough to always take the first available one?

Vector

OUR FIRST ADT!

Vector ADT

- ADT abstraction similar to an array or list
- You're probably thinking, "Hey, there was something like that in the language I studied before!"
  - This shouldn't be a surprise—remember that ADTs are defined as conceptual abstractions that are language-independent
- We will use **Stanford** library `Vector` (there is also an C++ STL `vector`, which will not use—watch out for capitalization!)

Stanford Library Vector

- We declare one like this:
  - `#include "vector.h"    // note quotes to mean Stanford version`
  - `Vector<string> lines; // note uppercase V here`
- This syntax is called **template** syntax
  - In C++, template containers must be **homogenous** (*all items the <u>same</u> type*)
  - The type goes in the < > after the class name Vector

```
// Example: initialize a vector containing 5 integers
Vector<int> nums {42, 17, -6,  0, 28};
```

index	*0*	*1*	*2*	*3*	*4*
value	42	17	-6	0	28

Vector

- Examples of declaring a Vector:
  - `Vector<int> pset3Scores;`
  - `Vector<double> measurementsData;`
  - `Vector<Vector<int>> allAssignmentScores;`

More on 2-D Vectors in a moment, with Grid ADT!

- Examples of using a Vector:
  - `pset3Scores.add(98);`
  - `pset3Scores.add(85);`
  - `pset3Scores.add(92);`
  - `cout << pset3Scores[0] << endl; // prints 98`
  - `cout << pset3Scores[pset3Scores.size() - 1] << endl; // prints 92`
  - `allAssignmentScores.add(pset3Scores);`
  - `cout << allAssignmentScores[0][1] << endl; // prints 85`

Vector Performance

A LITTLE PEEK AT HOW VECTORS WORK BEHIND THE SCENES

Performance Warning Vector insert/remove

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

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

→

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

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

→

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

- **These operations are <u>slower</u> the more elements they need to shift**

Performance Warning Vector insert/remove

- `v.insert(2, 42)`
  - shift elements right to ma

index	0	1	2	3	
value	3	8	9	7	5

index				3	4	5
value	3	8	42	9	7	5

> Pro tip: if possible in your situation, try to use `add()` (at the end) rather than `insert()` at the beginning

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

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

→

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

- **These operations are <u>slower</u> the more elements they need to shift**

Your turn: Vector performance

- **Warm-up question:** tell a neighbor what the contents of the vector look like at the end of each of OPTION 1 and at the end of OPTION 2. *(As shown,* v *starts out empty in both cases)*

```
Vector<int> v;
for (int i = 0; i < 100; i++) {
    v.insert(0, i); // OPTION 1
}
```

```
Vector<int> v;
for (int i = 0; i < 100; i++) {
    v.add(i); // OPTION 2
}
```

index	*0*	*1*	*2*	*3*	*4*	*…*
value	99	98	97	96	95	…

index	*0*	*1*	*2*	*3*	*4*	*…*
value	0	1	2	3	4	…

Your turn: Vector performance

- Compare how many times we write a number into one "box" of the Vector, in these two codes. Write can be the original write, or because it had to move over one place. *(As shown,* v *starts out empty in both cases)*

```
Vector<int> v;
for (int i = 0; i < 100; i++) {
    v.insert(0, i); // OPTION 1
}
```

```
Vector<int> v;
for (int i = 0; i < 100; i++) {
    v.add(i); // OPTION 2
}
```

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 5x as many times as the other
D. Something else!

Answer now on pollev.com/cs106b !

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!

Your turn: Vector performance

- **Answer: (D) Something else! (about 50x)**
  - In addition to analyzing the code and predicting number of writes needed, we can also time the code using our Stanford 106B test system.
  - **Check the code bundle for class today for runnable version!**

```
void runInsert(int size)
{
    Vector<int> v;
    for (int i = 0; i < size; i++) {
        v.insert(0, i);
    }
}

void runAdd(int size)
{
    Vector<int> v;
    for (int i = 0; i < size; i++) {
        v.add(i);
    }
}
```

```
/* * * * * * Test Cases * * * * * */
PROVIDED_TEST("Timing comparison")
{
    int size = 500000;
    TIME_OPERATION(size, runInsert(size));
    TIME_OPERATION(size, runAdd(size));
}
```

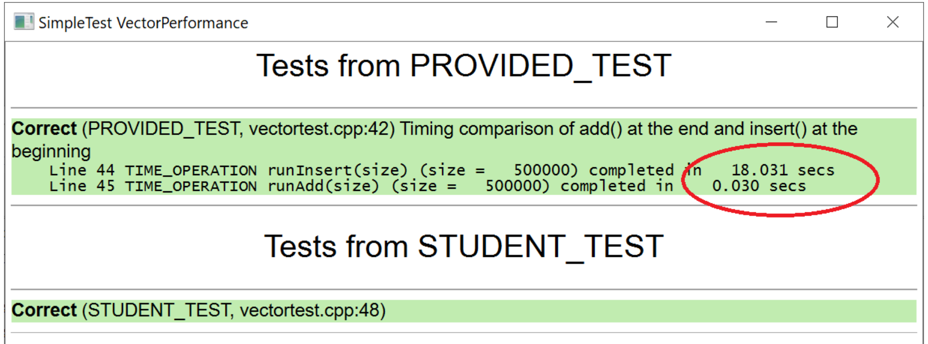
Your turn: Vector performance

- **Answer: (D) Something else! (about 50x)**
  - In addition to analyzing the code and predicting number of writes needed, we can also time the code using our Stanford 106B test system.
  - **Check the code bundle for class today for runnable version!**

```
void runInsert(int size)
{
    Vector<int> v;
    for (int i = 0; i < size; i++) {
        v.insert(0, i);
    }
}

void runAdd(int size)
{
    Vector<int> v;
    for (int i = 0; i < size; i++) {
        v.add(i);
    }
}
```

```
/* * * * * * Test Cases * * * * * */
PROVIDED_TEST("Timing comparison")
{
    int size = 500000;
    TIME_OPERATION(size, runInsert(size));
    TIME_OPERATION(size, runAdd(size));
}
```



Your turn: Vector performance

- **Answer: (D) Something else! (about 50x)**
  - Number of times a number is written in a box:
    - OPTION 1:
      - First loop iteration: 1 write
      - Next loop iteration: 2 writes … continued…
      - Formula for sum of numbers 1 to N = (N * (N + 1)) / 2
      - *(don’t worry if you don’t know this formula, we only expected a ballpark estimate)*
      - 100 * (100 + 1) / 2 = 10,100 / 2 = **5,050**
    - OPTION 2:
      - First loop iteration: 1 write
      - Next loop iteration: 1 write … continued…
      - **100**

Vector performance and parameter passing

- **Pro Tip:** always use pass-by-reference for containers like Vector (and Grid, which we'll see next) in this class!
  - For efficiency reasons—don't want to make a big copy every time with pass-by-value!

```
void printFirst(Vector<int>& input) {
    cout << input[0] << endl;
}

void printFirst100Times(Vector<int>& input) {
    for (int i = 0; i < input.size(); i++) {
        printFirst(input); // very expensive if not for &
    }
}
```

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:

```
#include "grid.h"

Grid<int> chessboard;
Grid<int> image;
Grid<double> realMatrix;
```

Code Reading Exercise: Grids and loops and loop

```
void printMe(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;
    }
}
```

2	1	2	0	0
1	0	2	1	2
0	0	0	1	1
2	2	2	2	2
1	1	0	1	1

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(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;
    }
}
```

row - 1 col – 1	row - 1 col + 0	row - 1 col + 1
row + 0 col - 1	row col	row + 0 col +1
row + 1 col - 1	row + 1 col + 0	row + 1 col + 1

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)