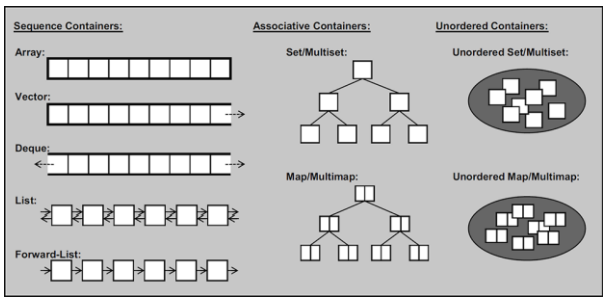
Herbert Wertheim
College of Engineering
Department of Computer & Information Science & Engineering
UNIVERSITY of FLORIDA

Data Structures – Part I

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

Data Structures

- A **data structure** is a particular way of organizing data in a computer so that it can be used effectively
  - Arrays (Fixed size)
  - Dynamic Arrays
  - Ordered set/map
  - Unordered set/map
  - Heap and Priority Queue
  - Stack / Queue / Deque
  - Linked list
  - Pair / Tuple
  - Customized data structure



- Different DS suits different problems
  - use C++ STL, Java API and/or standard libraries included with Python 3

Pic from https://oi-wiki.org/

Data Structures – Cont.

	C++	Java	Python
Dynamic Arrays	vector	ArrayList (slightly faster than Vector)	list
Ordered set Ordered map	set map	TreeSet TreeMap	OrderedSet OrderedDict
Unordered set Unordered map	unordered_set unordered_map	HashSet HashMap	set Hash table
Heap and Priority Queue	priority_queue	PriorityQueue	queue.PriorityQueue
Stack	stack	Stack	stack
Queue / Deque	queue deque	Queue Deque	queue.Queue deque
Linked list	list	LinkedList	collections.deque
Pair / Tuple	pair<a, b> tuple<a, b, c>	AbstractMap.SimpleEntry	tuple class

Operator Overloading Example

```
struct student {
  string name;
  int grade;
}
```

```
bool operator<(const student& a, const student& b) {
  return a.grade < b.grade || (a.grade == b.grade && a.name > b.name);
}
```

```
priority_queue<student> pq;
int main() {
  // …
  // set priority queue
  for (int i = 1; i <= n; i++) {
    // get name and grade;
    pq.push({name, grade});
  }
  // now pq.top() contains student with highest grade (or if tie, student whose name is the smallest Lexicographic order)
  // …
}
```

Stack

- Last In First Out (LIFO)
  - O(1) for insertion (push) and O(1) deletion (pop) from the top
  - Used in Recursion, Evaluation of Postfix Expressions, Bracket Matching
- Example:
  - Bracket matching: UVA 551 - Nesting a Bunch of Brackets
    - Read the brackets one by one from left to right. Every time we encounter a close bracket, we need to match it with the latest open bracket
      - Special attention: (* *)

551 Nesting a Bunch of Brackets

In this problem we consider expressions containing brackets that are properly nested. These expressions are obtained by juxtaposition of properly netsted expressions in a pair of matching brackets, the left one an opening and the right one a closing bracket.

(a + $ (b =) (a)) is properly nested

(a + $) b =) (a ( ) is not

In this problem we have several pairs of brackets, so we have to impose a second condition on the expression: the matching brackets should be of the same kind. Consequently '(())' is OK, but '([])' is not. The pairs of brackets are:

```
(   )
[   ]
{   }
<   >
(*  *)
```

The two characters '(*' should be interpreted as one symbol, not as an opening bracket '(' followed immediately by an asterisk, and similarly for '*)'. The combination '(*)' should be interpreted as '(*' followed by ')'.

Write a program that checks wheter expressions are properly nested. If the expression is not properly nested your program should determine the position of the offending bracket, that is the length of the shortest prefix of the expression that can not be extended to a properly nested expression. Don't forget '(*' counts as one, as does '*)'. The characters that are not brackets also count as one.

Postfix Notation

- Postfix notation is a notation for writing arithmetic expressions in which the operands appear before their operators
  - LeetCode 150. Evaluate Reverse Polish Notation

1. Read postfix expression from left to right till the end
   - If wordListber
     - push it onto Stack
   - If operator
     i. Pop two items
        - A <- Top item
        - B <- Next to Top item

     ii. Evaluate B operator A
        - push calculation result onto Stack
2. result <- Top element
3. END

923*-

3

- Extend: Infix to Postfix Conversion Using Stacks

Dynamic arrays

- Vector / ArrayList
  - Efficiently add/remove elements at the end of the structure. In general, vector is almost as fast as using an ordinary array in O(1) time
- Deque
  - Dynamic array that can be efficiently manipulated at both ends of the structure
  - O(1) average time with a larger constant factor
  - Used in algorithms for ‘Sliding Window’ problem, etc
- Queue
  - Used in Breadth First Search, Topological Sort, etc
- Example:
  - https://codeforces.com/problemset/problem/879/B