- - - - - Data Structures - - - - -

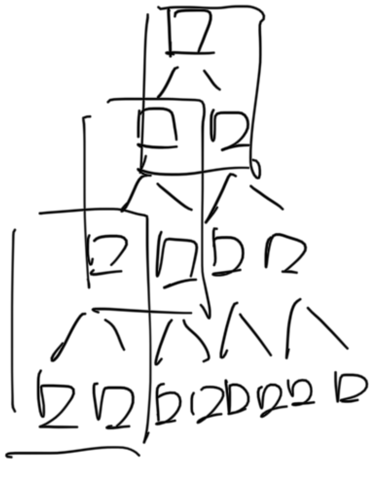
— Recursion

Recursion :
- Break problem into smaller identical problems.

Iteration :
- The repetition of a process in order.

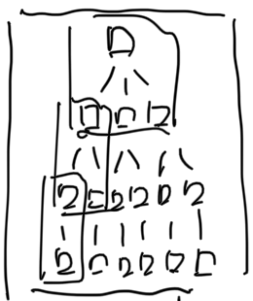
Ex : Binary Search

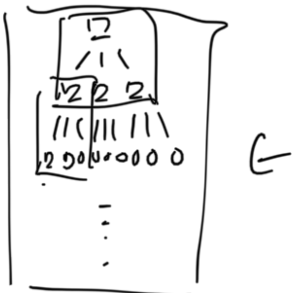
```
BS ( in anArrayType : ArrayType, in value : ItemType)
- if ( anArray's item = value )
    // If finding an item in anArray that
    // has equal value to value

  else
    // Find the midpoint of anArray
    // Determine if the midpoint is value
    //              which half of anArray holds value

      if (value in first half) ->
          BS ( first half of anArray, value )
      else ->
          BS ( second half of anArray, value )
```



↳)

Each recursive function calls itself and solve an identical, smaller problem.





These two involved iteration.

Other example: Tower of Hanoi
Fibonacci series
Combinatorial numbers

Summary

→ Define the problem in term of smaller problems and see if it reduces the problem size.

→ For every case it should reach a base case in the end.

— Object-Oriented Programming

Class ⇒ data member + member functions

data member ↓ Typically data

member functions ↓ decide how data are operated

Three characteristics of OOP:

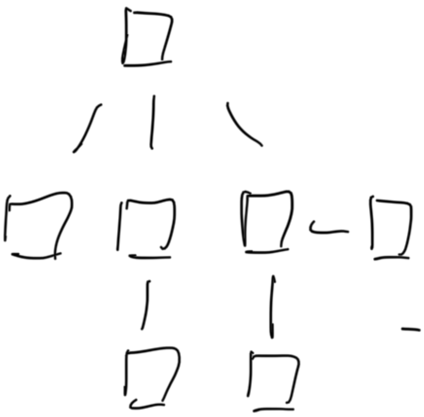
Encapsulation:
Objects combine data and operations
Hides inner details

Inheritance:
Classes can inherit properties from other classes
Existed class can be reused

Polymorphism:
Objects can determine appropriate operations at execution time

execution time

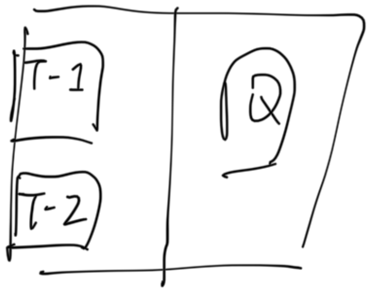
Motives:



A modularized program is easier to write/read, making it easier to maintain.

- Separates the purpose and use of a module from its implementation
- Make unnecessary details inaccessible from outside the module.
- A module's specifications should detail how it behaves and be independant of the module's implementation

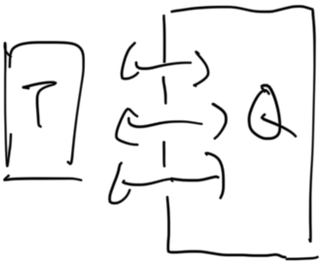
Concepts:



Isolated tasks: the implementation of task T doesn't affect task Q

The isolation of modules is not total, a function's specification or contract governs how it interacts with other

Goals:



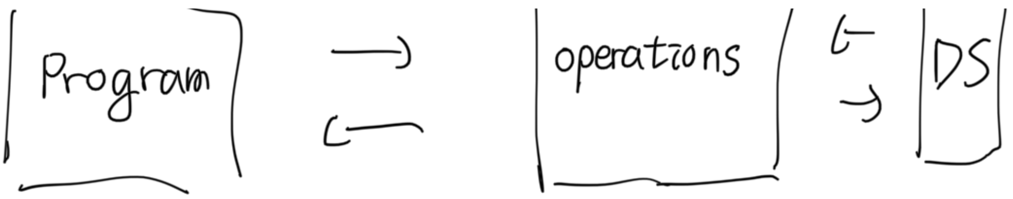
Typical operations on data

Add/Remove data to a data collection

Access the information of a data in a data collection

ADT (Abstract Data Type)

Composed of a collection of data and a set of operations on that data



Notice that Program does not directly access to DS.

Also, operations can be used in an application without the knowledge of how the operations will be implemented

When designing an ADT, we should know what data or its operations a problem require.

—Linked lists

Pointer: Contain the address in memory of a memory cell, initially undefined (NOT NULL)

#Static allocation

*p represents the memory cell to which p points

& is the address-of operator

To place the address of a variable into a pointer use p = &x or p = new int

Dynamic allocation of a memory cell that can contain an integer

delete operator returns dynamically allocated memory to the system for reuse

Notes that it also leaves it undefined.

Dynamic Allocation of Arrays:

- You can use the new operator to allocate an array dynamically

int size = 50; double *anArray = new double[size]

- An array name is a pointer to its first element

- An array name is a pointer

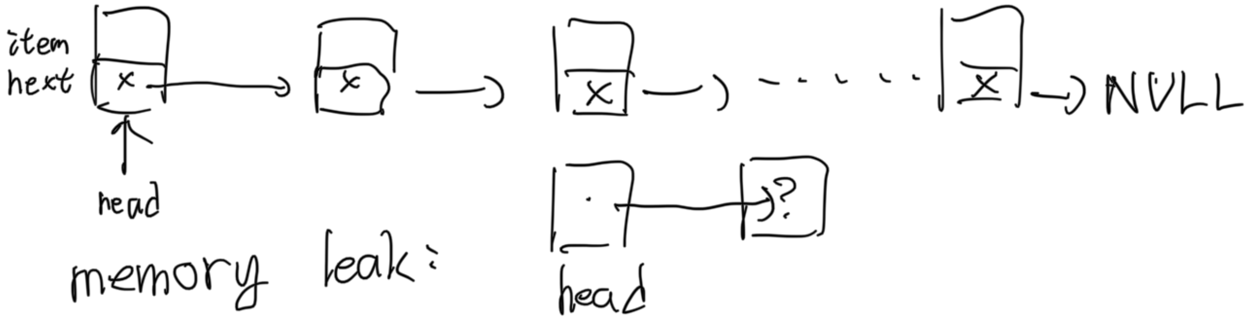
anArray[2] ≡ *(anArray+2)

- The size of a dynamically allocated array can be increased

double *oldArray = anArray;

anArray = new double[3*size]

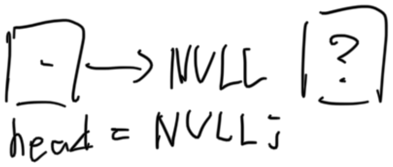
Pointer-Based Linked Lists:



memory leak:

Execute head = NULL while head is point to something result in a lost cell

head = new Node;



-> operator is to reference the item pointer is pointing.

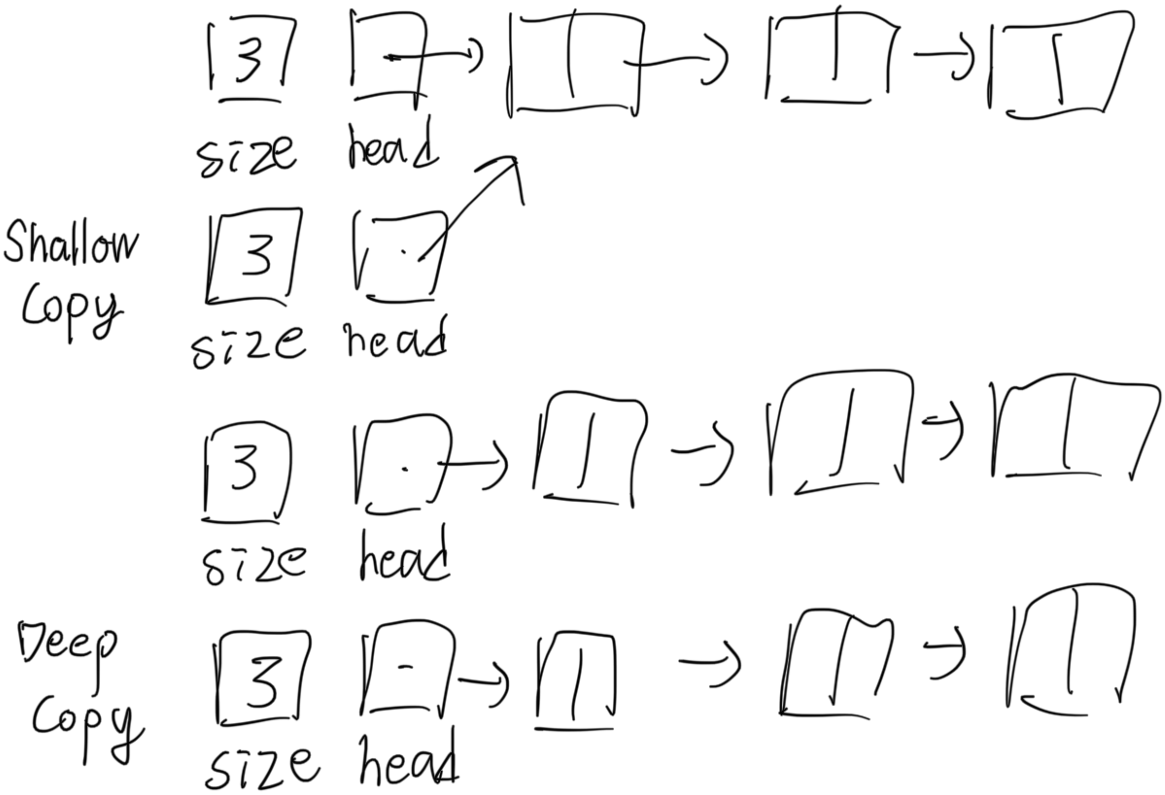
Constructors and Destructors:

Default constructor initializes size and head

A destructor is required for dynamically allocated memory

Copy constructor creates a deep copy

If you omit a copy constructor, the compiler generates one, but it is only sufficient for implementations that use statically allocated arrays.



Arrays vs Linked Lists

Size

- Increasing the size of a resizable array can waste storage and time
- Linked list grows and shrink as necessary

Storage requiremenes

- In the ADT, A pointer-based implemeation require more memory for each item.

Retrieval

- The time to access the ith item

  array → Constant (i)

  pointer → Depends on i

Insertion and deletion

array → shifting data

pointer → traversal

Remarks

array -based lists use an implicit ordering scheme
pointer explicit

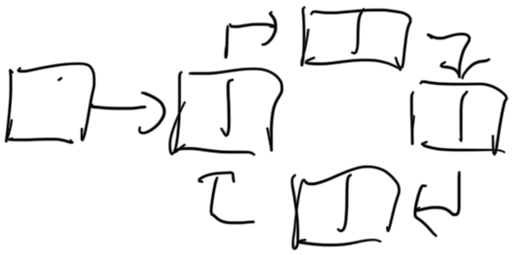
array enable direct access of an element

Linked list require a traversal

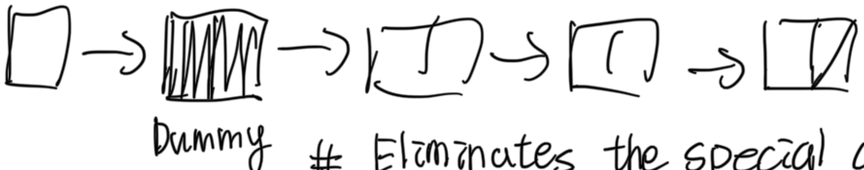
Increasing size of an array involves copying

Variations:

Circular Linked Lists:

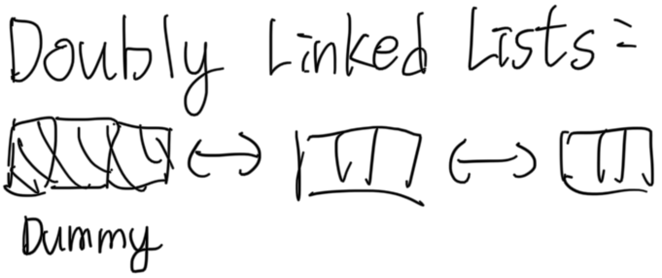


Dummy Head Node:



Eliminates the special cases for insertion into and deletion from the start of a linked list

Doubly Linked Lists:



(not necessary)

— Solution collections

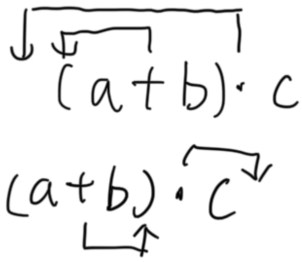
A language is a set of strings of symbols

A grammar is the rules for forming the

strings in a language

Algebraic Expressions =

Infix = $a + b$	$(a+b)\cdot c$	
Prefix = $+ a b$	$\cdot + a b c$	
Postfix = $a b +$	$a b + c \cdot$	



advantages of prefix and postfix expressions

- No precedence / association rules
- No parentheses - Simple grammars
- Straightforward recognition and evaluation algorithms

Backtracking - a strategy for guessing at a solution and backing up when an impasse is reached, can be combined to solve problems