Chapter 20 - Part 1

Intro to Data Structures

The Big Picture

Abstract Data Type

(Logical View)

(Concept/Specifications)

- An ADT is an organized collection of information and a set of operations used to manage that information.
- The operations define its interface and behavior.
- As long as the ADT fulfills the conditions of the interface, it doesn't really matter how the ADT is implemented.

Data Structure

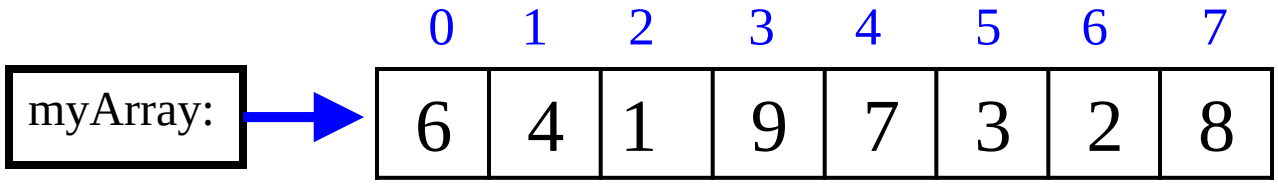
(Physical View)

(Implementation)

- It is the implementation of an ADT.
- It uses programming constructs and primitive data types supported by the language.
- It provides an implementation-independent view of the data, allowing us to change implementation details without changing how the users interact with the data structure.

What is a Data Structure?

- A data structure is a collection of data elements organized in some fashion. The organization reflects the relationship among the data elements.
- A data structure supports both storing the data elements and the operations for manipulating the data in the structure.
- For example, an ***array*** is a data structure that holds sequential collection of data element of the same type. Array operations allows us to find the size of the array, store, retrieve, and modify elements in the array.



Example - 1

ADT ***List*** Specifications

- A collection of data elements stored sequentially. It supports insertion and deletion anywhere in the list.

ADT ***List*** Implementation

- Static Array: Predetermined and contiguous memory locations.
- Dynamic Array: Expandable contiguous memory locations (e.g., Java ArrayList).
- Linked-List: Non-contiguous memory locations linked with pointers.

ADT List Specification

Definition

The **ADT List** is a linear sequence of data elements, together with a list of methods.

Methods

createList( )

// post: Create an empty list

add(index, item)

// post: Insert item at position index of a list if
// 1<=index<= size()+1. If index <= size(), items at position
// index onwards are shifted one position to the right

remove(index)

// post: Remove item at position index of a list if
// 1<=index<= size(). Items at position index+1 onwards are
// shifted one position to the left

isEmpty( )

// post: Determine if a list is empty

get(index)

// post: Returns item at position index of
// a list if 1<=index<=size()

size( )

// post: Returns number of items in a list

Others methods…

Note: Items in the list must be of the same type

ADT List Example

Consider a grocery list: *milk, eggs, butter, apples, bread*

1. How can we construct this list?
   - Create an empty list
   - Use series of add operations

```
myList.createList( )
myList.add(1,milk)
myList.add(2,eggs)
myList.add(3,butter)
. . .
milk, eggs, butter, apples, bread
```

2. How do we insert a new item into a certain position?
   - Use add operation

```
myList.add(4, nuts)

milk, eggs, butter, nuts, apples, bread
```

3. How do we delete an item from a certain position?
   - Use remove operation

```
myList.remove(5)

milk, eggs, butter, nuts, bread
```

Example - 2

ADT ***Stack*** Specifications

- A collection of data elements stored sequentially such that insertion and deletion of data elements take place only at one end of the list (called top of the stack).

ADT ***Stack*** Implementation

- Static Array: Predetermined and contiguous memory locations.
- Dynamic Array: Expandable contiguous memory locations (e.g., Java ArrayList).
- Linked-List: Non-contiguous memory locations lined with pointers.

ADT Stack Specification - 1

Definition

- The **ADT Stack** is a linear sequence of data elements stored sequentially, together with access procedures.
- The access procedures permit **insertion** and **deletion** of items only at one end of the sequence (the "**top**").
- The stack is a list structure, sometimes called a **last-in-first-out** (or LIFO) list (stack behavior).
- A stack is either empty, or it consists of a sequence of items. Access is limited to the "top" item on the stack at all times.

ADT Stack Specification - 2

Methods

createStack( ) // post: creates an empty Stack

push(newItem) // post: adds newItem at the top of the stack.

top( ) // post: returns the top item of the stack. It does not change the stack.

isEmpty( ) // post: determines whether a stack is empty

pop( ) // post: changes the stack by removing the top item.

// Other methods…

Classic Data Structures

- **Linear data structure:** A data structure in which data elements are traversed (processed) in sequential fashion. Examples include lists, stacks, and queues.
- **Non-Linear data structure:** Every data item is attached to several other data items to reflect the relationship between the data elements. The data items are not arranged in a sequential structure. Examples include Trees and Graphs.

- **List:** A collection of data elements stored sequentially. Lists support insertion and deletion anywhere in the list.
- **Stack:** A special type of the list where insertions and deletions take place only at the one end, referred to as the top of a stack.
- **Queue:** A waiting list, where insertions take place at the back (known as *Tail*) of the queue and deletions take place from the front (known as *Head*) of the queue.
- **Graph:** A set of nodes connected with edges. One node may connect with multiple nodes.
- **Tree:** A special graph such that it has a *root* node and every other node has one *parent* node.

Linear vs. Non-Linear

- Non-Linear data structures use memory efficiently since it does not require contiguous memory space.
- Non-Linear data structures do not require knowing the length of the data items (structure size) prior to allocation.
- Non-Linear data structures have no restriction on the length.
- Non-Linear data structures overhead is the *link* to the next data item in the structure.

Think Objects

- In object-oriented thinking, a data structure is an object that stores other objects, referred to as data or elements.
- So some people refer a data structure as a *container object* or a *collection object*.
- To **define** a data structure is essentially to declare a class. The class for a data structure should use data fields to store data and provide methods to support operations such as insertion and deletion.
- To **create** a data structure is to create an instance/object from the class. Then apply the methods on the instance to manipulate the data structure such as insertion, deletion, sorting, search, and others.

End of Slides