UNIT-II

ARRAY BASED LINEAR DATA STRUCTURES

DATA ABSTRACTION

• The process of hiding the inside detail of a data and providing the essential information.

• It is a programming design technique that relies on the separation of interface and implementation.

Why abstraction?

– Easier to design – hide what doesn't matter

– Security – prevent access to things that shouldn't be accessed

– Modifiability – can be modified without affecting users

Abstract Data Type

• An Abstract Data Type (ADT) can be defined as: – Is a specification of a data set and the operations on that data.

– An ADT is an externally defined data type that holds some kind of data.

– An ADT has built-in operations that can be performed on it or by it.

– Does not indicate any information about the internal representation of the data storage or implementation of the operations. – Independent of any programming language.

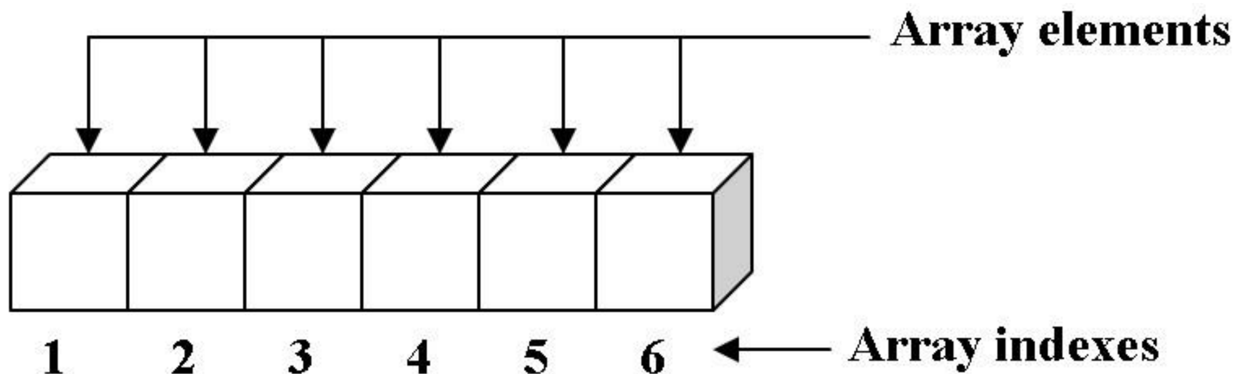
We are all used to dealing with the primitive data types as abstract data types.

- However, you know how to work with double values by adding them, multiplying them, truncating them to values of type int, inputting and outputting them, and so on. To work with values of type double, you only need to know how to use these operations and what they do. You don't need to know how they are implemented.

- When you create your own new data types—stacks, queues, trees, and even more complicated structures—you are necessarily deeply involved in the data representation and in the implementation of the operations.
- An abstract data type in a theoretical construct that consists of data as well as the operations to be performed on the data while hiding implementation.
- For example, a stack is a typical abstract data type. Items stored in a stack can only be added and removed in certain order – the last item added is the first item removed. We call these operations, pushing and popping. In this definition, we haven't specified have items are stored on the stack, or how the items are pushed and popped. We have only specified the valid operations that can be performed.
- For example, if we want to read a file, we wrote the code to read the physical file device. That is, we may have to write the same code over and over again.

ARRAY ADT

- The array is an **abstract data type (ADT)** that holds a collection of elements accessible by an index.



One-dimensional array with six elements

- The elements stored in an array can be anything from primitives types such as integers to more complex types like instances of classes.
- An element is stored in a given index and they can be retrieved at a later time by specifying the same index.

- The Array (ADT) have one property, they store and retrieve elements using an index. The array (ADT) is usually implemented by an Array (Data Structure).
- The way indices work is specific to the implementation, but you can usually think of them as the slot number in the array that the value occupies.
- To fulfill the ADT an array must implement at least a way to retrieve the data from a given index and to store the data in a given index.
- set(i, v) -> Sets the value of index i to v
- get(i) -> Returns the value of index i in the array V

Advantages

- Fast, random access of items or elements.
- Very memory efficient, very little memory is needed other than that needed to store the contents.

Disadvantages

- Slow insertion and deletion of elements
- Array size must be known when the array is created and is fixed (static)

Problems with Array implementation of lists:

- Insertion and deletion are expensive. For example, inserting at position 0 (a new first element) requires first pushing the entire array down one spot to make room, whereas deleting the first element requires shifting all the elements in the list up one, so the worst case of these operations is O(n).
  - Even if the array is dynamically allocated, an estimate of the maximum size of the list is required. Usually this requires a high over-estimate, which wastes considerable space.
- Simple arrays are generally not used to implement lists. Because the running time for insertion and deletion is so slow and the list size must be known in advance.

Linear List ADT(Polynomials)

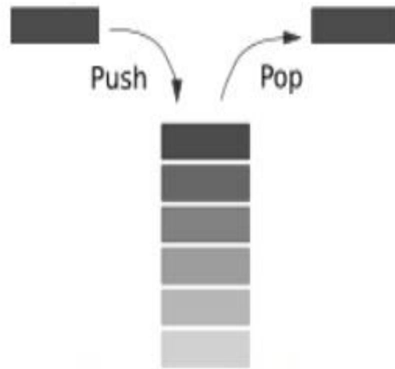
- One of the most simple and most used ADT is the linear list.
- A linear list is a sequence of $n \geq 0$ elements X1 , ..., Xn called nodes having the same base type T.
- The essential structural properties of a linear list are: – If n>0 then X1 is the first node and Xn is the last node. – If $1 < i < n$ then the i-th node Xi is preceded by Xi-1 and succeeded by Xi+1.
- To define lists as an ADT we must define a set of operators. Note that each set of operators defines a distinct ADT !
- There is a large variety of operators that can be defined on lists. Choosing the right set of operators is application dependent. Ex: – Provide the first or the last element of the list. – Compute the length of the list. – Append an element as the first or the last element of the list. – Check if the list is empty. – Remove the first or the last element of the list. –

STACK ADT - Linear Data Structures

"A stack is an ordered list in which all insertions and deletions are made at one end, called the top"stacks. are sometimes referred to as Last In First Out (LIFO) lists.

Stacks have some useful terminology associated with them:

Push To add an element to the stack
Pop To remove an element from the stock
Peek To look at elements in the stack without removing them
LIFO Refers to the last in, first out behavior of the stack
FILO Equivalent to LIFO



Simple representation of a stack

Given a stack S=(a[1],a[2],.......a[n]) then we say that a1 is the bottom most elemen

and element a[i]) is on top of element a[i-1], 1<i<=n.

Simple representation of a stack

Given a stack S=(a[1],a[2],.......a[n]) then we say that a1 is the bottom most element
and element a[i]) is on top of element a[i-1], 1<i<=n.

Implementation of stack :

1. array (static memory).
2. linked list (dynamic memory)

The operations of stack is

1. PUSH operations
2. POP operations
3. PEEK operations

The Stack ADT

A stack S is an abstract data type (ADT) supporting the following three methods:
push(n) : Inserts the item **n** at the top of stack

pop() : Removes the top element from the stack and returns that top element. An error occurs if the stack is empty.
peek(): Returns the top element and an error occurs if the stack is empty.

1. Adding an element into a stack. (called PUSH operations)

Adding element into the TOP of the stack is called PUSH operation.

Check conditions :

TOP = N , then **STACK FULL** where N is maximum size of the stack.

Adding into stack (PUSH algorithm)

procedure add(item : items);

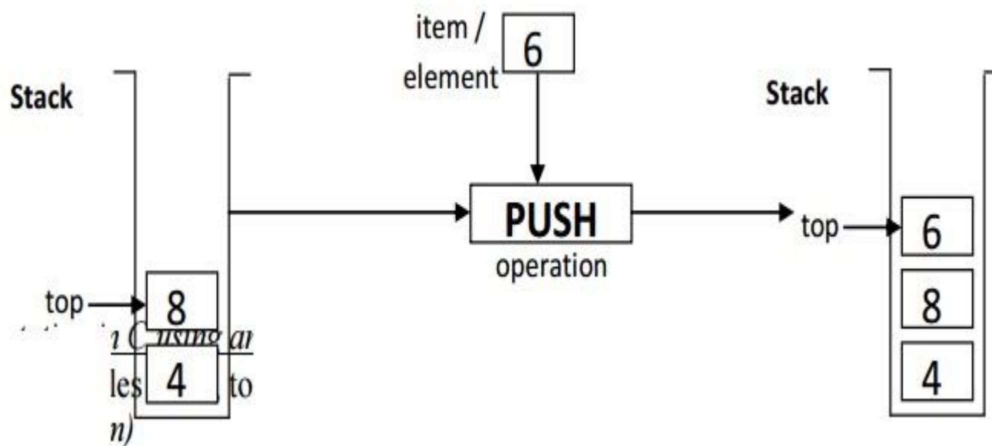
{add item to the global stack stack ; top is the current top of stack and n is its maximum size}

begin

if top = n **then** stackfull; top := top+1;

stack(top) := item;

end: {of add}



Implementation in C

/* here, the variables top and size are global variables */

void push (int item)

{

if (top == size-1)

printf(“Stack is Overflow”);

else

{

top = top + 1; stack[top] = item;
}

}

2. Deleting an element from a stack. (called POP operations)

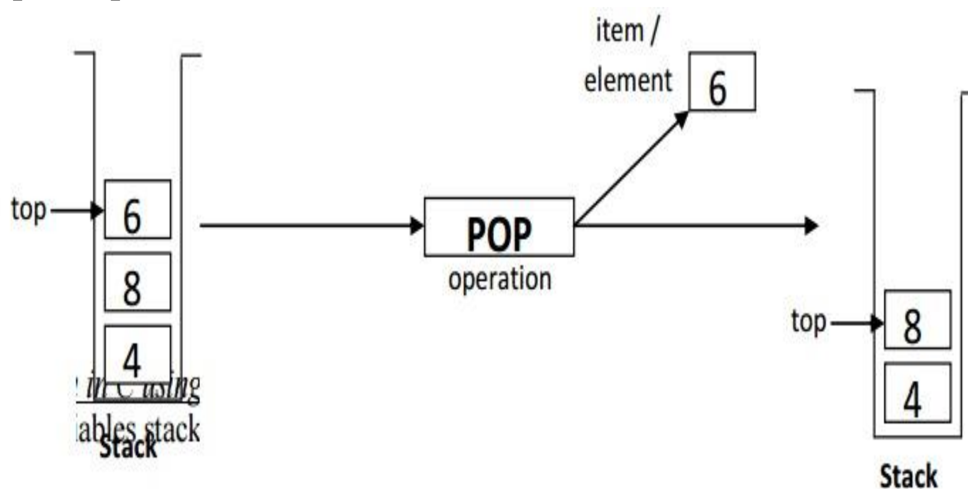
Deleting or Removing element from the TOP of the stack is called POP operations.

Check Condition:
TOP = 0, **then** STACK EMPTY
Deletion in stack (POP Operation)
procedure delete(**var** item : items);

{remove top element from the stack stack and put it in the item} **begin**

if top = 0 **then** stackempty; item := stack(top);

top := top-1; **end;** {of delete}

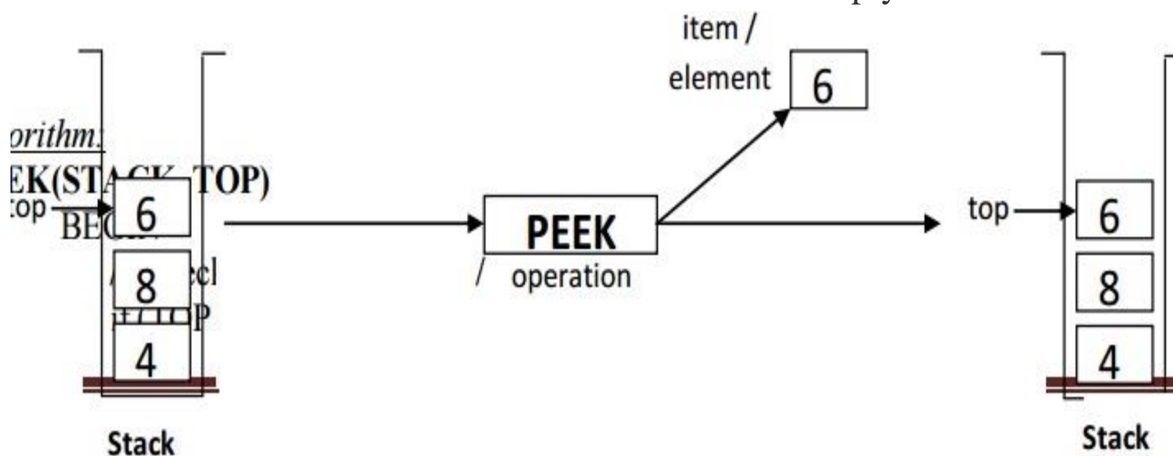


Implementation in using array:

```
/* here, the variables stack, and top are global variables
*/
int pop ( )
{

if (top == -1)
{

printf("Stack is Underflow"); return (0);

}
else

{
return (stack[top--]);

}
}
```

3. Peek Operation:

- Returns the item at the top of the stack but does not delete it.
- This can also result in **underflow** if the stack is empty.



Algorithm:

PEEK(STACK, TOP)

```
BEGIN

/* Check, Stack is empty? */
if (TOP == -1) then

print "Underflow" and return 0. else

item = STACK[TOP] / * stores the top element into a local variable */
return item          / * returns the top element to the user */

END
```

Applications of Stack:

1. It is very useful to evaluate arithmetic expressions. (Postfix Expressions)
2. Infix to Postfix Transformation
3. It is useful during the execution of recursive programs
4. A Stack is useful for designing the compiler in operating system to store local variables inside a function block.
5. A stack (memory stack) can be used in function calls including recursion.
6. Reversing Data
7. Reverse a List
8. Convert Decimal to Binary
9. Parsing –It is a logic that breaks into independent pieces for further processing
10. Backtracking

Note :

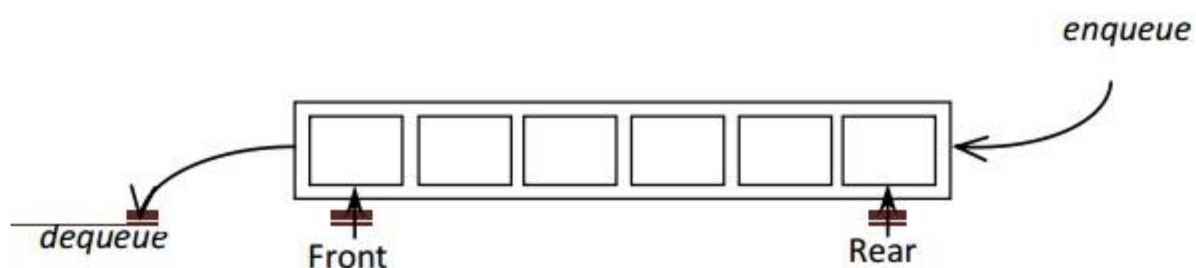
1. Infix notation A+(B*C)
equivalent Postfix notation ABC*+
2. Infix notation (A+B)*C
Equivalent Postfix notation AB+C*

Queue ADT - Linear Data Structures

"A queue is an ordered list in which all inse at another end called FRONT". squeue are some times referred to as First In First Out (FIFO) lists.

QUEUE ADT

"A queue is an ordered list in which all inse at another end called FRONT". s*queue* are some times referred to as First In First Out (FIFO) lists.



Example

1. The people waiting in line at a bank cash counter form a queue.

2. In computer, the jobs waiting in line to use the processor for execution. This queue is

   called ***Job Queue***.

Operations Of Queue

There are two basic queue operations. They are,

Enqueue –Inserts an item / element at the rear end of the queue. An error occurs if the queue is full.

Dequeue –Removes an item / element from the front end of the queue, and returns it to the user. An error occurs if the queue is empty.

1. Addition into a queue

```
procedure addq (item : items);
{add item to the queue q}
begin

   if rear=n then queuefull
   else begin

      rear :=rear+1; q[rear]:=item;

   end;
   end;
   {of addq}
```

2. Deletion in a queue

```
procedure deleteq (var item : items);
{delete from the front of q and put into item}
begin

   if front = rear then queueempty
   else begin

      front := front+1
      item := q[front];

   end;
   end
```

Uses of Queues (Application of queue)

Queues remember things in first-in-first-out (FIFO) order. Good for fair (first come first served) ordering of actions.

Examples of use: (Application of stack)

1• scheduling (processing of GUI events printing request)
2• simulation orders the events (models real life queues (e.g. supermarkets checkout, phone calls on hold).

Evaluation of Expressions:

Calculators employing reverse Polish notation (also known as postfix notation)use a stack structure to hold values.

- Expressions can be represented in prefix, postfix or infix notations. Conversion from one form of the expression to another form needs a stack.
- Many compilers use a stack for parsing the syntax of expressions, program blocks etc. before translating into low level code.
- Most of the programming languages are context-free languages allowing them to be parsed with stack based machines. Note that natural languages are context sensitive languages and stacks alone are not enough to interpret their meaning.
- **Infix, Prefix and Postfix Notation.**
- We are accustomed to write arithmetic expressions with the operation between the two operands: **a+b** or **c/d**. If we write **a+b*c**, however, we have to apply precedence rules to avoid the ambiguous evaluation (add first or multiply first?).

- There's no real reason to put the operation between the variables or values. They can just as well precede or follow the operands. You should note the advantage of prefix and postfix: the need for precedence rules and parentheses are eliminated.

Infix	Prefix	Postfix
a + b	+ a b	a b +
a + b * c	+ a * b c	a b c * +
(a + b) * (c - d)	* + a b - c d	a b + c d - *
b * b - 4 * a * c		
40 - 3 * 5 + 1		

An arithmetic expression will have operands and operators. Operator precedence listed below:

Highest : (*) and (/)

Lowest : (+) and (-)

For most common arithmetic operations, the operator symbol is placed in between its two operands. This is called ***infix notation***.

*Example: A + B , E * F*

Parentheses can be used to group the operations.

*Example: (A + B) * C*

Accordingly, the order of the operators and operands in an arithmetic expression does not uniquely determine the order in which the operations are to be performed.

Polish notation refers to the notation in which the operator symbol is placed before its two operands. This is called ***prefix notation***.

*Example: +AB, *EF*

The fundamental property of polish notation is that the order in which the operations are to be performed is completely determined by the positions of the operators and operands in the expression.

Accordingly, one never needs parentheses when writing expressions in Polish notation.

Reverse Polish Notation refers to the analogous notation in which the operator symbol is placed after its two operands. This is called ***postfix notation.***

*Example: AB+, EF**

Here also the parentheses are not needed to determine the order of the operations.

The computer usually evaluates an arithmetic expression written in infix notation in two steps,

1. *It converts the expression to* ***postfix notation****.*

2. *It evaluates the postfix expression.*

In each step, the stack is the main tool that is used to accomplish the given task.

(Algorithm) Infix expressions are often translated into postfix form in which the operators appear after their operands.

Steps:

1. Initialize an empty stack.
2. Scan the Infix Expression from left to right.
3. If the scannned character is an operand, add it to the Postfix Expression.
4. If the scanned character is an operator and if the stack is empty, then push the character to stack.
5. If the scanned character is an operator and the stack is not empty, Then
   (a) Compare the precedence of the character with the operator on the top of the stack.

   (b) While operator at top of stack has higher precedence over the scanned character & stack is not empty.

(i) POP the stack.

(ii) Add the Popped character to Postfix String.

(c) Push the scanned character to stack.

6. Repeat the steps 3-5 till all the characters

7. While stack is not empty,

(a) Add operator in top of stack

(b) Pop the stack.

8. Return the Postfix string.