UNIT-3 :-

Data Structures

Data structures :-

Representation of data in particular manner.

particular way of (or) organizing data in a computer. To reduce the space and time complexities of different tasks. Some popular data structures are

(i) Arrays
(ii) Linked list
(iii) Stack
(iv) Queue
(v) Binary Tree
(vi) Binary Search Tree
(vii) Graph
(viii) Heap
(ix) Hashing
(x) Misc

Stack :-

(i) Stack is ADT (Abstract Data Type).

(ii) Stack follows LIFO (Last In First Out). The element which is placed last, it is accessed first.

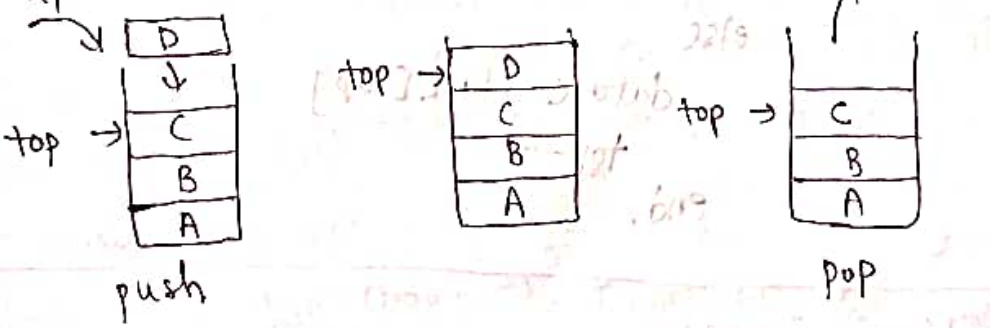
ex :- Dec of cards, piles of plates etc..

(iii) There are two ways to implement a stack.
- → Using arrays (Static, Dynamic)
- → Using Linked List.

(iv) Stack operations are
- → push() → pop() → peek() → isFull() → isEmpty()

(v) top pointer is used.

(vi) Representation.



Algorithms :-

push() :- pushing an element on the stack.

push operation steps ⇒

Step-1 :- Checks if the stack is full.

Step-2 :- If the stack is full produces an error and exit.

Step-3 :- If the stack is not full, Increment top

Step-4 :- Add data element to the stack location.

Step-5 :- Returns success.

Algorithm ⇒ begin

```
if top is equal to max-size
    display "stack is full".
else
    top++
    stack[top] ← data
end
```

---

pop() :- Removing an element from the stack (or) accessing an element from the stack.

pop operation steps ⇒

Step-1 :- Check if the stack is empty.

Step-2 :- If stack is empty, produce an error and exit.

Step-3 :- If stack is not empty, decrement top.

Step-4 :- Decreases the value of top by 1.

Step-5 :- Returns success.

Algorithm ⇒ begin

```
if top is equal to -1
    display "stack is empty".
else
    data ← stack[top]
    top--;
end.
```

Applications Queue :-

(i) CPU Scheduling

(ii) Handling website traffic

(iii) Maintaining the playlist in media player.

peek():- get the top data element of the stack, without removing it.

```
Algorithm ⇒ begin
              return stack[top]
            end.
```

---

isFull():- Checks if stack is Full. or not

```
Algorithm ⇒ begin
              if top == maxsize
                 return true
              else
                 return false.
            end
```

---

isEmpty():- Checks if stack is empty. or not

```
Algorithm ⇒ begin
              if top == -1
                 return true
              else
                 return false.
            end.
```

Applications of Stack:-

(i) Evaluation of expressions.
(ii) Backtracking
(iii) Undo Operations
(iv) mouse gaming

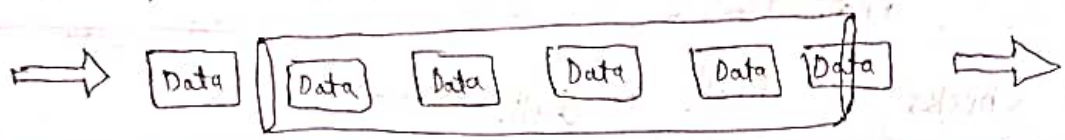
Queues:-

(i) Queue is ADT (Abstract Data Type)
(ii) Queue follows FIFO (First In First Out). The data item stored first will be accessed first.
ex:- single lane one way road, bus stops, ticket windows
(iii) There are two ways to implement queues.
   → Using arrays (static, dynamic)
   → Using Linked list

(iv) Queue operations are
→ enqueue() → dequeue() → peek() → isEmpty() → isFull()

(v) front, rear are ~~keywords~~ pointers in queue.

(vi) Representation.



Algorithms :-

enqueue() :- add an item to the queue.

enqueue operation steps ⇒
Step-1 :- Check if the queue is full
Step-2 :- If the queue is full, produce ~~stack~~ queue is full and exit.
Step-3 :- If the queue is not full, increment rear
Step-4 :- Add data element to the queue location, at which rear is pointing
Step-5 :- return success.

Algorithm ⇒ begin
if rear == maxsize
display "Queue is full".
else
rear++;
queue[rear] ← data
end

---

dequeue() :- remove an item from the queue.

dequeue operation steps ⇒
Step-1 :- Check if the queue is empty.
Step-2 :- If the queue is empty, produce ~~stack~~ queue is empty and exit.
Step-3 :- If the queue is not empty, access the data where front is pointing.
Step-4 :- Increment front to the next available data element.
Step-5 :- return success.

Algorithm ⇒ begin
if front == rear == -1
display "Queue is empty".
else
data ← queue[front]
front++
end

Peek():- To see the data at the front of the queue.

Algorithm ⇒ begin
return queue[front]
end.

---

isFull():- Checks if queue is full or not

Algorithm ⇒ begin
if rear == maxsize
return true
else
return false.
end

---

isEmpty:- Checks if queue is empty or not

Algorithm ⇒ begin
if front == rear == -1
return true
else
return false
end.

Arithmetic Expression Evaluation:-

Evaluation of Expressions:-

All expressions are usually represented in infix notation.

There are three types of expressions

(i) Infix notation ex:- A+B

(ii) pre-fix notation (polish notation) ex:- +AB

(iii) post fix notation (Reverse polish notation) ex:- AB+

There are 3 levels of precedence for 5 binary operators.

→ Highest :- Exponentiation (^)

→ Next highest :- Multiplication (*) and division (/)

→ Lowest :- Addition (+) and subtraction (-).

ex:- Infix notation = $(A-B)^{\wedge}[C/(D+E)+F]$

Postfix notation = $AB-CDE+/F+^{\wedge}$

The procedure for getting the result is

(i) Convert the expression to reverse polish notation

(ii) Push the operands into the stack

(iii) When any operator encounter then pop two topmost operands (or) elements.

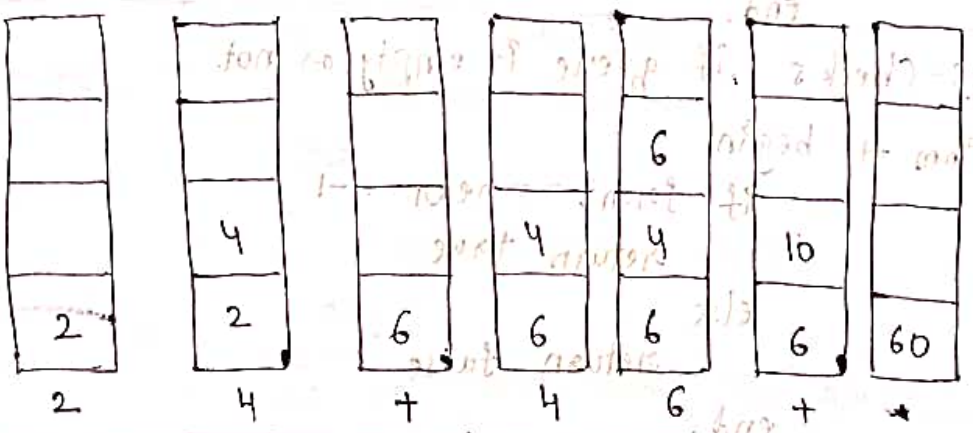
(iv) After push the result into the stack.

(v) After the complete execution the final result remains on the top of the stack.

---

ex:- $(2+4)^{\wedge}(4+6) = 60$

$2\,4\,+\,4\,6\,+^{\wedge} = 60$



---

ex:- $4\,5\,6\,{}^{*}\,+$

Step	I/P Symbol	Operation	Stack	Calculation
1	4	push	4	
2	5	push	4,5	
3	6	push	4,5,6	
4	*	pop 2 elements & Evaluate	4	5*6 = 30
5		pop result	4,30	
6	+	pop 2 elements & Evaluate		4+30 = 34
7		push result	34	
8		pop		34 (result)

Stack Operations:-

```
#include <stdio.h>
#include <conio.h>
int stack[100], choice, n, top=0, value, i;
void main()
{
  clrscr();
  printf(" Enter the size of stack\n");
  scanf ("%d", &n);
  printf(" Stack operations are \n 1. push\n 2. pop\n
          3. display\n 4. exit\n");
  while (choice != 4)
  { printf(" Enter the choice\n");
    scanf ("%d", &choice);
    switch (choice)
    {
      case 1 : if (top >= n)
                 printf(" Stack is full\n");
               else
               { printf (" Enter the value\n");
                 scanf ("%d", &value);
                 top++;
                 stack[top] = value;
               }
               break;
      case 2 : if (top <= 0)
                 printf (" Stack is empty\n");
               else
               { printf (" value removed \n");
                 top--;
               }
               break;
```

```
case 3 : if(top>=0)
         {printf("The stack elements are\n");
          for(i=top;i>=0;i--)
          {printf("%d\n",stack[i]);
          }
         }
         else
         {printf("stack is empty\n");
         }
         break;
case 4 : printf("Exiting\n"); break;
default : printf("Invalid choice\n");
}
}
getch();
}
```