| **2** |  | **Stack and Queues** |
| --- | --- | --- |
|  | 2.1 | Introduction, ADT of Stack, Operations on Stack, Array Implementation of Stack, Applications of Stack-Well form-ness of Parenthesis, Infix to Postfix Conversion and Postfix Evaluation, Recursion. |
|  | 2.2 | Introduction, ADT of Queue, Operations on Queue, Array Implementation of Queue, Types of Queue-Circular Queue, Priority Queue, Introduction of Double Ended Queue, Applications of Queue. |

***Stack***

Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO (last in first out) or FILO (first in last out).

Mainly the following four operations are performed in the stack :

1. Push: Adds an item in the stack. If the stack is full then it is set to be an overflow condition.

Syntax :

void push(int data)

{

if (top>=n-1)

{

printf(“Stack is Overfull! overflow”);

}

else

{

top ++;

stack[top]=data;

}

}

Pop: Removes an item from the stack the items are popped in the reverse order in which

they are Pushed if the stack is empty then it is set to be an underflow condition.

Syntax:

void pop(int data)

{

if (top<=n-1)

{

printf(“Stack is empty! underflow”);

}

else

{

printf(“The popped elements is %d”, Stack[top]);

top --;  
 }

}

Peek or Top: Return top element of a stack.

Syntax:

Int stack\_a(int stack[])

{

int data;

if (top== -1)

{

printf(“Stack is Empty”);

}

else

{

return stack [Top];

}

}

1. Is Empty: Returns true if the stack is empty, else false.

Syntax :

void isempty(int stack[])

{

If (top == -1)

{

printf (“Stack is empty);

}

else

{

printf (“Stack is not empty”);

}

}

Stack full: To check whether the stack is full, Returns true if the stack is full, else false

Syntax :

void full(int stake[])

{

if (top == n-1)

{

printf (“Stack is full”);

}

else

{

printf (“stack is not full”);

}

}

Stack is a linear data structure that follows a particular order in which the operations are

performed. The order may be LIFO (last in first out) or FILO (first in last out).

Mainly the following four operations are performed in the stack :

1. Push: Adds an item in the stack. If the stack is full then it is set to be an overflow condition.

Syntax :

void push(int data)

{

if (top>=n-1)

{

printf(“Stack is Overfull! overflow”);

}

else

{

top ++;

stack[top]=data;

}

}

1. Pop: Removes an item from the stack the items are popped in the reverse order in which they are Pushed if the stack is empty then it is set to be an underflow condition.

Syntax:

void pop(int data)

{

if (top<=n-1)

{

printf(“Stack is empty! underflow”);

}

else

{

printf(“The popped elements is %d”, Stack[top]);

top --;  
 }

}

1. Peek or Top: Return top element of a stack.

Syntax:

Int stack\_a(int stack[])

{

int data;

if (top== -1)

{

printf(“Stack is Empty”);

}

else

{

return stack [Top];

}

}

1. Is Empty: Returns true if the stack is empty, else false.

Syntax :

void isempty(int stack[])

{

If (top == -1)

{

printf (“Stack is empty);

}

else

{

printf (“Stack is not empty”);

}

}

1. Stack full: To check whether the stack is full, Returns true if the stack is full, else false

Syntax :

void full(int stake[])

{

if (top == n-1)

{

printf (“Stack is full”);

}

else

{

printf (“stack is not full”);

}

}

### **Double Ended Queue Classification**

**Deque can be classified as follows:**

**Input-restricted Deque: In input-restricted, deletion can be done from both the ends but insertion can be done only at the rear end of the queue.**

**Output-restricted Deque: In the output-restricted queue, insertion can be done from both the ends but deletion is done only at one end i.e. the front end of the queue.**

**We can also implement stacks and queues using deque.**