C Programming and Data Structures

Computer Science and Technology-B

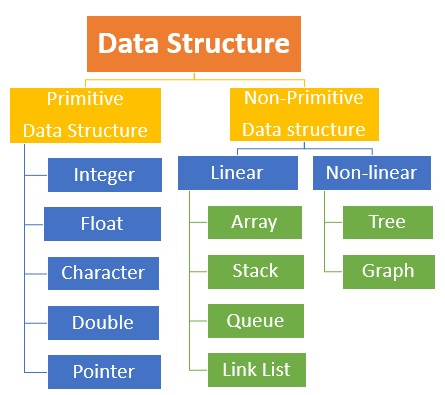
UNIT IV Data Structures

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Data Structures:

A Data Structure is a data organization, management, and storage format that enables efficient access and modification.

Classification of Data Structures:

**

In this unit we are going to learn about STACK and QUEUE.

STACK

*Stack is a linear data structure which follows a particular order in which the operations are performed.*

*Stack Follows:*

*LIFO (Last In First Out)   
FILO (First In Last Out)*

*Real Time Examples:*

**

**

**

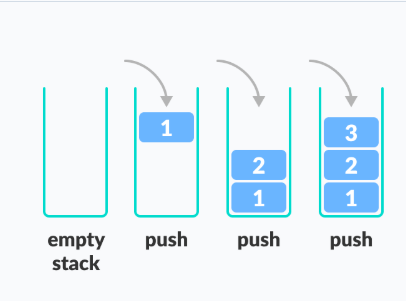
Stack Operations:

*Four types of stack operations.*

*They are:*

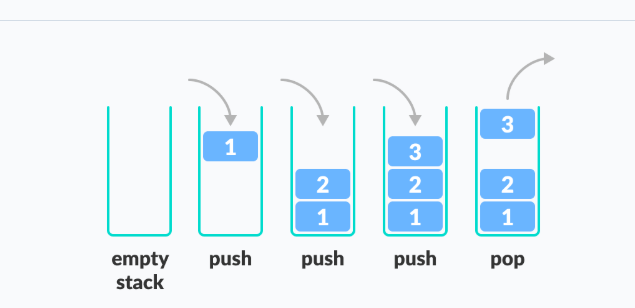
* + - * + *Push*
        + *Pop*
        + *Peek or Top*
        + *isEmpty*

PUSH:

*Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.*

*POP:*

*Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.*

**

PEEK OR TOP:

*Returns top element of stack.*

*isEmpty:*

*Returns true if stack is empty, else false*

Working of Stack:

* *A pointer called TOP is used to keep track of the top element in the stack.*
* *When initializing the stack, we set its value to -1 so that we can check if the stack is empty by comparing TOP == -1.*
* *On pushing an element, we increase the value of TOP and place the new element in the position pointed to by TOP.*
* *On popping an element, we return the element pointed to by TOP and reduce its value.*
* *Before pushing, we check if the stack is already full*
* *Before popping, we check if the stack is already empty*

Stack implementation using array:

#include<stdio.h>

#include<stdlib.h>

int top=-1;

int stack[100];

void push();

void pop();

void display();

void main()

{

int a;

printf("\*\*\*MENU\*\*\*");

printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");

while(a!=4)

{

printf("Enter your Choice:");

scanf("%d",&a);

switch(a)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("Enter a valid choice\n");

break;

}

}

}

void push()

{

int n;

if(top==99)

{

printf("Stack Overflow\n");

}

else

{

printf("Enter a value:");

scanf("%d",&n);

top=top+1;

stack[top]=n;

}

}

void pop()

{

if(top==-1)

{

printf("Stack Underflow\n");

}

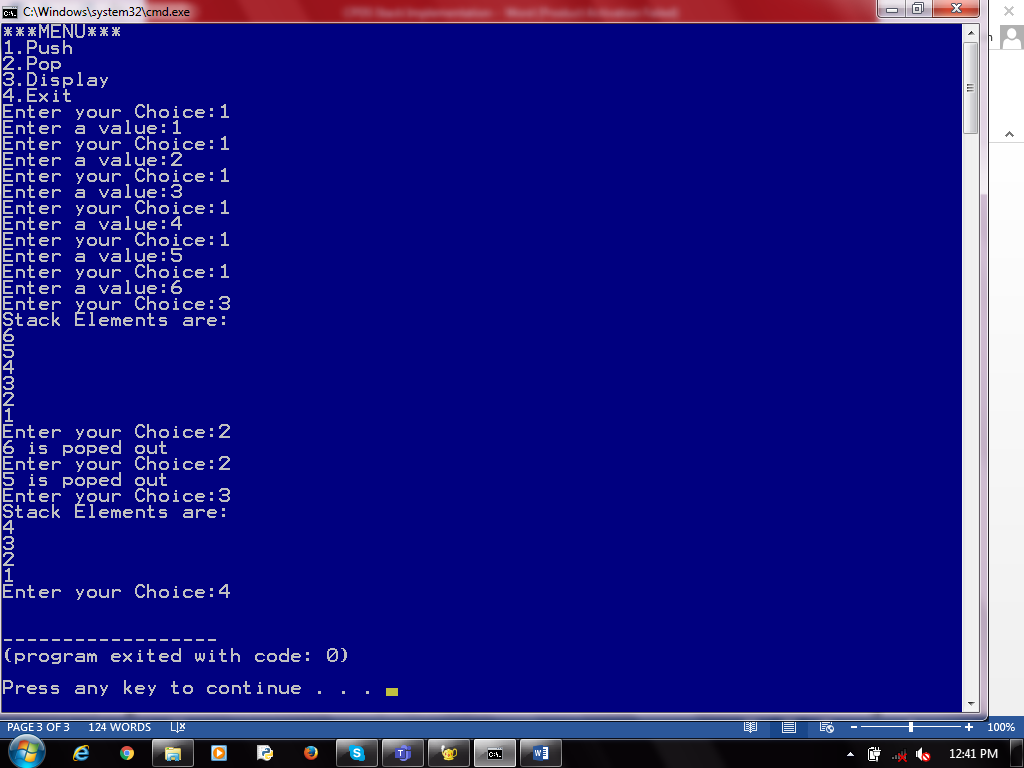
else

{

printf("%d is poped out\n",stack[top]);

top=top-1;

}

**}

void display()

{

int i;

if(top==-1)

{

printf("Stack is Empty\n");

}

else

{

printf("Stack Elements are:\n");

for(i=top;i>=0;i--)

{

printf("%d\n",stack[i]);

}

}

}

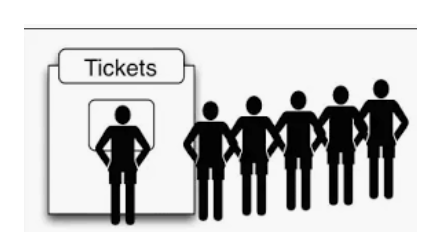
*Queue*

*A Queue is a linear structure which follows a particular order in which the operations are performed.*

*Queue follows:*

*FIFO (First In First Out)*

*Real Time Examples:*

**

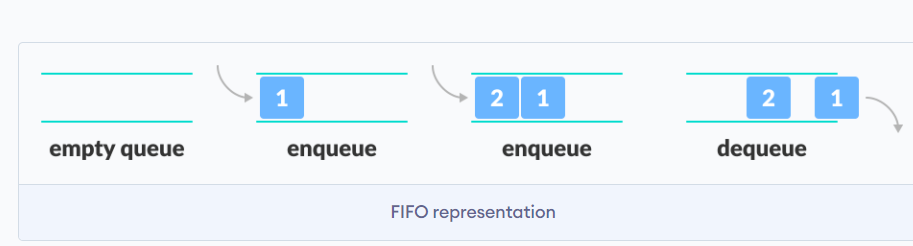
*Queue Operations:*

* *Enqueue:*

*Add an element to the end of the queue*

* *Dequeue:*

*Remove an element from the front of the queue*

**

* *IsEmpty:*

*Check if the queue is empty*

* *IsFull:*

*Check if the queue is full*

* *Peek:*

*Get the value of the front of the queue without removing it*

*Working of Queue:*

* *Two pointers called FRONT and REAR are used to keep track of the first and last elements in the queue.*
* *When initializing the queue, we set the value of FRONT and REAR to -1.*
* *On enqueuing an element, we increase the value of REAR index and place the new element in the position pointed to by REAR.*
* *On dequeuing an element, we return the value pointed to by FRONT and increase the FRONT index.*
* *Before enqueuing, we check if the queue is already full.*
* *Before dequeuing, we check if the queue is already empty.*
* *When enqueuing the first element, we set the value of FRONT to 0.*
* *When dequeuing the last element, we reset the values of FRONT and REAR to -1.*

*Queue Array Implementation:*

#include<stdio.h>

#include<stdlib.h>

int front=-1;

int rare=-1;

int queue[5]; //Global Variable Declaration

void enqueue();

void dequeue(); //Function prototype

void display();

void main()

{

int a;

printf("\*\*\*MENU\*\*\*");

printf("\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

while(a!=4)

{

printf("Enter your Choice:");

scanf("%d",&a);

switch(a)

{

case 1:

enqueue(); //function calling

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("Enter a valid choice\n");

break;

}

}

}

void enqueue() //function defination

{

int n;

if(rare==5-1)

{

printf("Queue is Full\n");

}

else

{

if(front==-1)

{

front=0;

}

printf("Enter a value:");

scanf("%d",&n);

rare=rare+1;

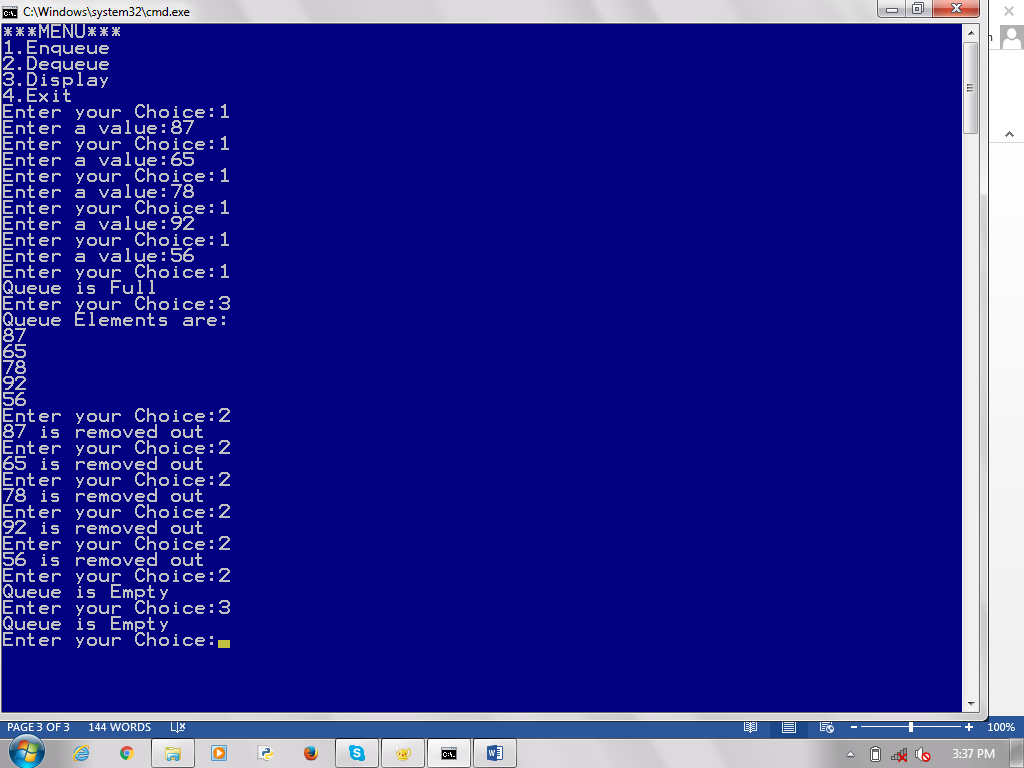
queue[rare]=n;

}

}

void dequeue()

{

 if(front==-1 || front>rare)

{

printf("Queue is Empty\n");

}

else

{

printf("%d is removed out\n",queue[front]);

front=front+1;

}

}

void display()

{

int i;

if(front==-1|| front>rare)

{

printf("Queue is Empty\n");

}

else

{

printf("Queue Elements are:\n");

for(i=front;i<=rare;i++)

{

printf("%d\n",queue[i]);

}

}

}

*Types of Queue:*

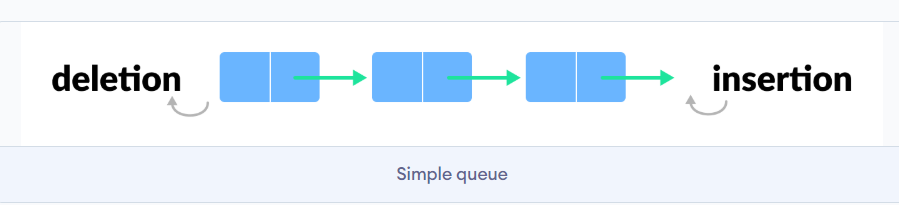
*They are four types of queues,*

*They are:-*

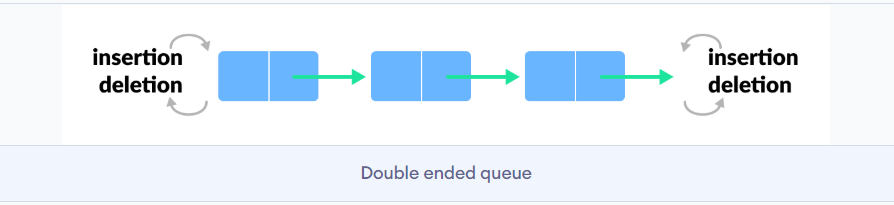
* *Simple Queue*
* *Circular Queue*
* *Priority Queue*
* *Double Ended Queue*

Simple Queue:

*In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows FIFO rule.*

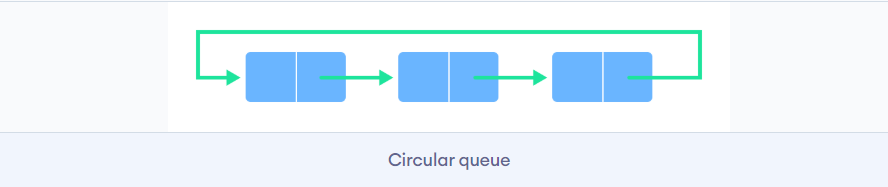
**

*Circular Queue:*

* *In a circular queue, the last element points to the first element making a circular link.*
* *The main advantage of a circular queue over a simple queue is better memory utilization.*

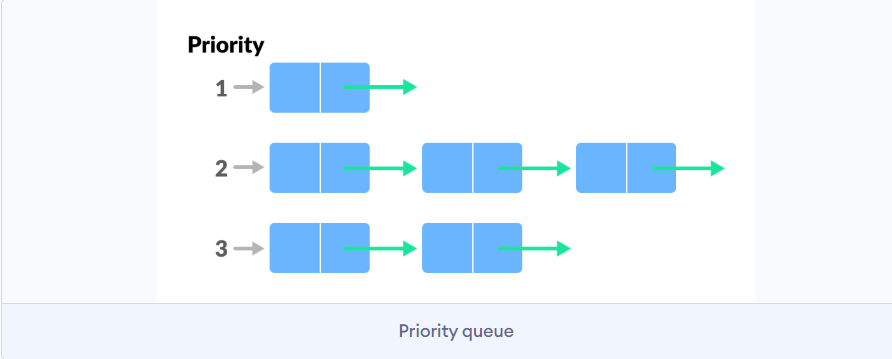
*Priority Queue:*

*A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority. If elements with the same priority occur, they are served according to their order in the queue.*

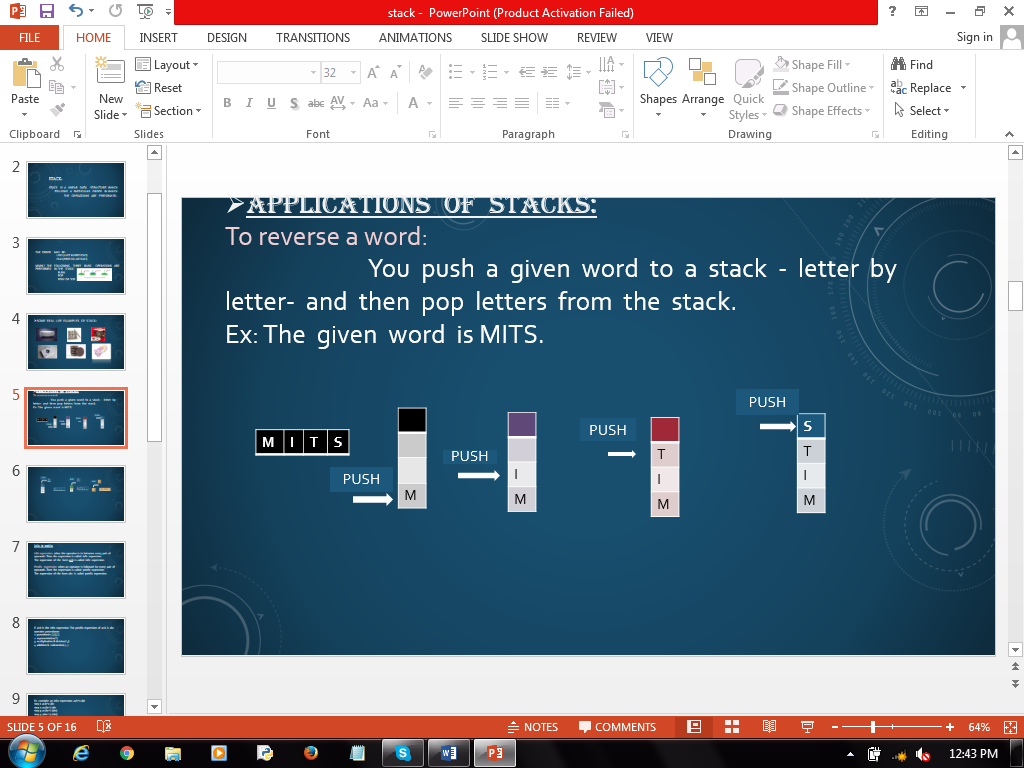
**

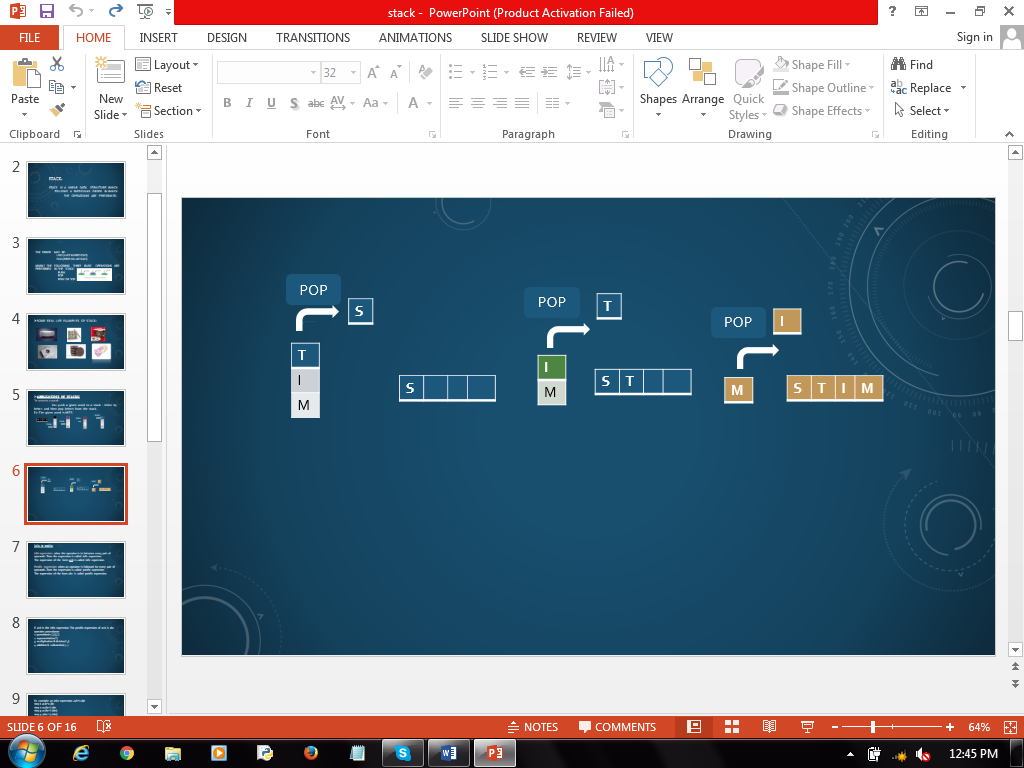
*Double Ended Queue or deque:*

*Double Ended Queue is a type of queue in which insertion and removal of elements can be performed from either from the front or rear. Thus, it does not follow FIFO rule (First In First Out).*

**

***Applications of Stack:***

*To reverse a word:   
 You push a given word to a stack - letter by letter- and then pop letters from the stack.  
Ex: The given word is MITS.*



*Infix to Postfix:*

***Infix expression****: When the operator is in between every pair of operands. Then the expression is called infix expression.  
The expression of the form* ***a+b*** *is called infix expression.* ***Postfix expression:*** *When an operator is followed for every pair of operands. Then the expression is called postfix expression.  
The expression of the form* ***ab+*** *is called postfix expression****.***

*If a+b is the infix expression. The postfix expression of a+b is ab+*

*Operator precedence:   
1. Parenthesis ( ),{ },[ ]  
2. Exponential (^)  
3. Multiplication & Division (\*,/)  
4. Addition & Subtraction (+,-)*

***Ex : Consider an infix expression a+b\*c-d/e*** *step 1: a+b\*c-d/e  
step 2: a+(bc\*)-d/e  
step 3: a+(bc\*)-(de/)  
step 4: (abc\*+)-(de/)  
step 5: (abc\*+)(de/)-  
step 6: abc\*+de/-*

*The postfix expression is abc\*+de/-*

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Infix*** | ***Stack*** | ***Postfix*** |
| ***Step 1*** | *a+b\*c-d/e* | *+ \** | *abc* |
| ***Step 2*** | *a+b\*c-d/e* | *null* | *abc\*+* |
| ***Step 3*** | *a+b\*c-d/e* | *- /* | *abc\*+de* |
| ***Step 4*** | *a+b\*c-d/e* | *null* | *abc\*+de/-* |

*The postfix expression is* ***abc\*+de/-***

* *Another application of stack is an “undo” mechanism in text editors ; this operation is accomplished by keeping all text changes in a stack.*

***Applications of Queue:****1. In real life scenario, call center phone systems uses queues to hold people calling them in an order , until a service representative is free.  
  
2. In printers and computer systems we use queues the first paper inserted will printed first.  
  
3. Key press sequence in keyboard.*