Name :- Varad Kedar Kulkarni

Roll no. :- BE19F05F032

Class :- SYCSE

Subject :- Data Structures

Assignment :- 1

Q.1] Write a program for insertion of node to list at the start and at the end of the list.

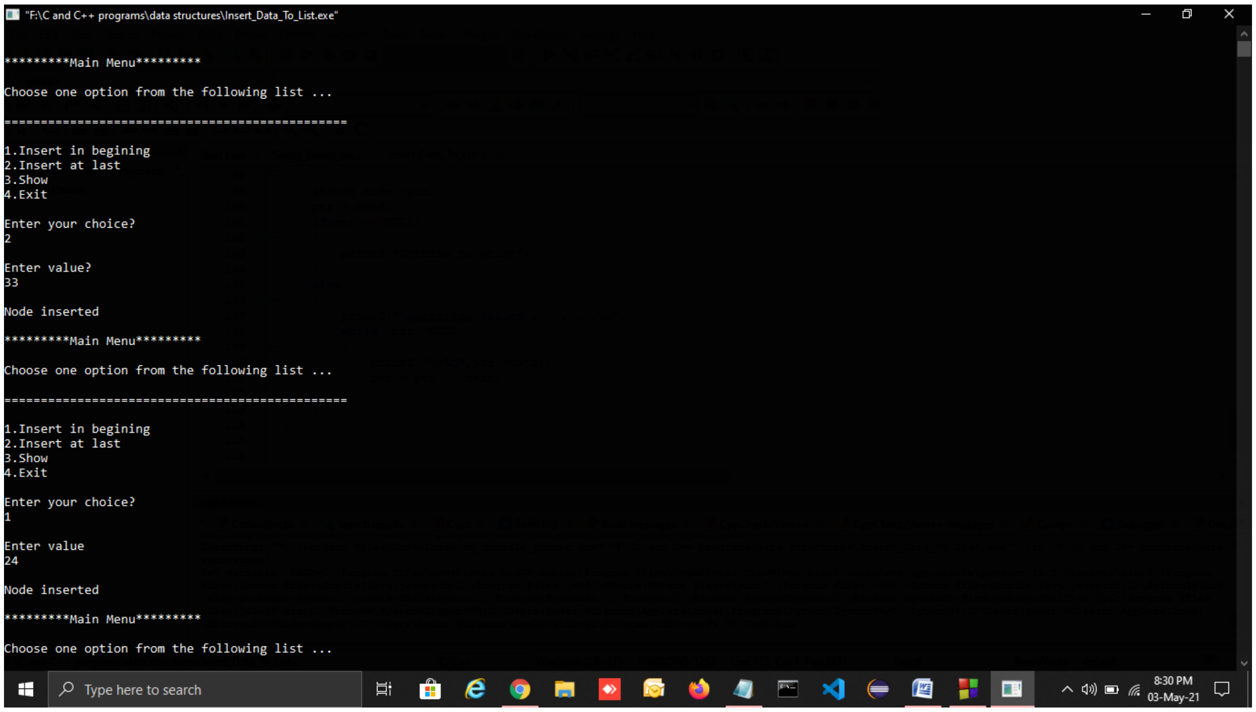
->
```
#include<stdio.h>

#include<stdlib.h>

struct node

{

   int data;

   struct node *next;

};

struct node *head;


void beginsert ();

void lastinsert ();

void display();

void main ()

{

   int choice =0;

   while(choice != 4)
```

```
{
  printf("\n\n*********Main Menu*********\n");
  printf("\nChoose one option from the following list ...\n");
  printf("\n==================================================\n");
  printf("\n1.Insert in begining\n2.Insert at last\n3.Show\n4.Exit\n");
  printf("\nEnter your choice?\n");
  scanf("\n%d",&choice);
  switch(choice)
  {
    case 1:
    beginsert();
    break;
    case 2:
    lastinsert();
    break;
    case 3:
    display();
    break;
    case 4:
    exit(0);
    break;
    default:
    printf("Please enter valid choice..");
  }
}
```

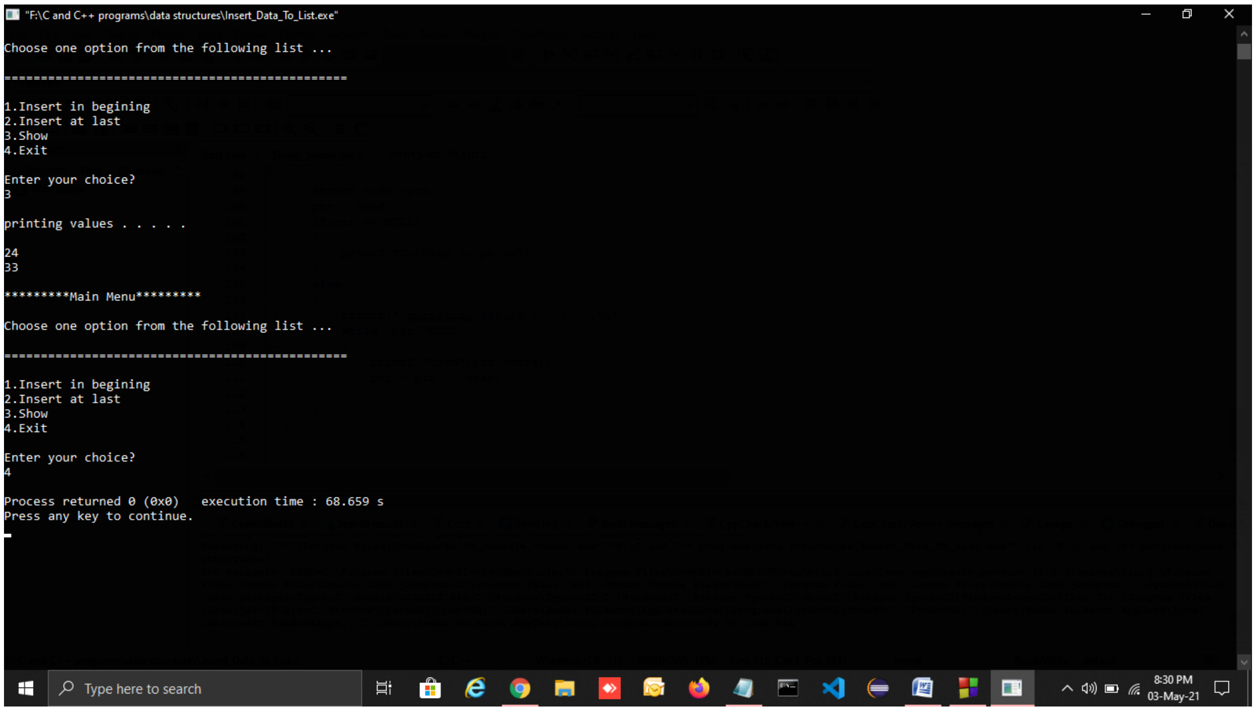
```
}

void beginsert()

{

    struct node *ptr;

    int item;

    ptr = (struct node *) malloc(sizeof(struct node *));

    if(ptr == NULL)

    {

        printf("\nOVERFLOW");

    }

    else

    {

        printf("\nEnter value\n");

        scanf("%d",&item);

        ptr->data = item;

        ptr->next = head;

        head = ptr;

        printf("\nNode inserted");

    }


}

void lastinsert()

{

    struct node *ptr,*temp;

    int item;
```

```
ptr = (struct node*)malloc(sizeof(struct node));
if(ptr == NULL)
{
  printf("\nOVERFLOW");
}
else
{
  printf("\nEnter value?\n");
  scanf("%d",&item);
  ptr->data = item;
  if(head == NULL)
  {
    ptr -> next = NULL;
    head = ptr;
    printf("\nNode inserted");
  }
  else
  {
    temp = head;
    while (temp -> next != NULL)
    {
      temp = temp -> next;
    }
    temp->next = ptr;
    ptr->next = NULL;
```

```
            printf("\nNode inserted");


        }
    }
}
void display()
{
    struct node *ptr;
    ptr = head;
    if(ptr == NULL)
    {
        printf("Nothing to print");
    }
    else
    {
        printf("\nprinting values . . . . .\n");
        while (ptr!=NULL)
        {
            printf("\n%d",ptr->data);
            ptr = ptr -> next;
        }
    }
}
```



```
"F:\C and C++ programs\data structures\Insert_Data_To_List.exe"

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in begining
2.Insert at last
3.Show
4.Exit

Enter your choice?
2

Enter value?
33

Node inserted

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in begining
2.Insert at last
3.Show
4.Exit

Enter your choice?
1

Enter value
24

Node inserted

*********Main Menu*********

Choose one option from the following list ...
```



```
"F:\C and C++ programs\data structures\Insert_Data_To_List.exe"

Choose one option from the following list ...

===============================================

1.Insert in begining
2.Insert at last
3.Show
4.Exit

Enter your choice?
3

printing values . . . . .

24
33

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in begining
2.Insert at last
3.Show
4.Exit

Enter your choice?
4

Process returned 0 (0x0)   execution time : 68.659 s
Press any key to continue.
```

Q.2] Write an algorithm to delete the node from the list and to insert the node at the end of the list.

-> To delete node from the list we have 2 methods :-

Iterative Method:

To delete a node from the linked list, we need to do the following steps.

1) Find the previous node of the node to be deleted.
2) Change the next of the previous node.
3) Free memory for the node to be deleted.

Recursive Method:

To delete a node of a linked list recursively we need to do the following steps.

1.We pass node* (node pointer) as a reference to the function (as in node* &head)

2.Now since current node pointer is derived from previous node's next (which is passed by reference) so now if the value of the current node pointer is changed, previous next node's value also gets changed which is the required operation while deleting a node (i.e points previous node's next to current node's (containing key) next).

3.Find the node containing the given value.

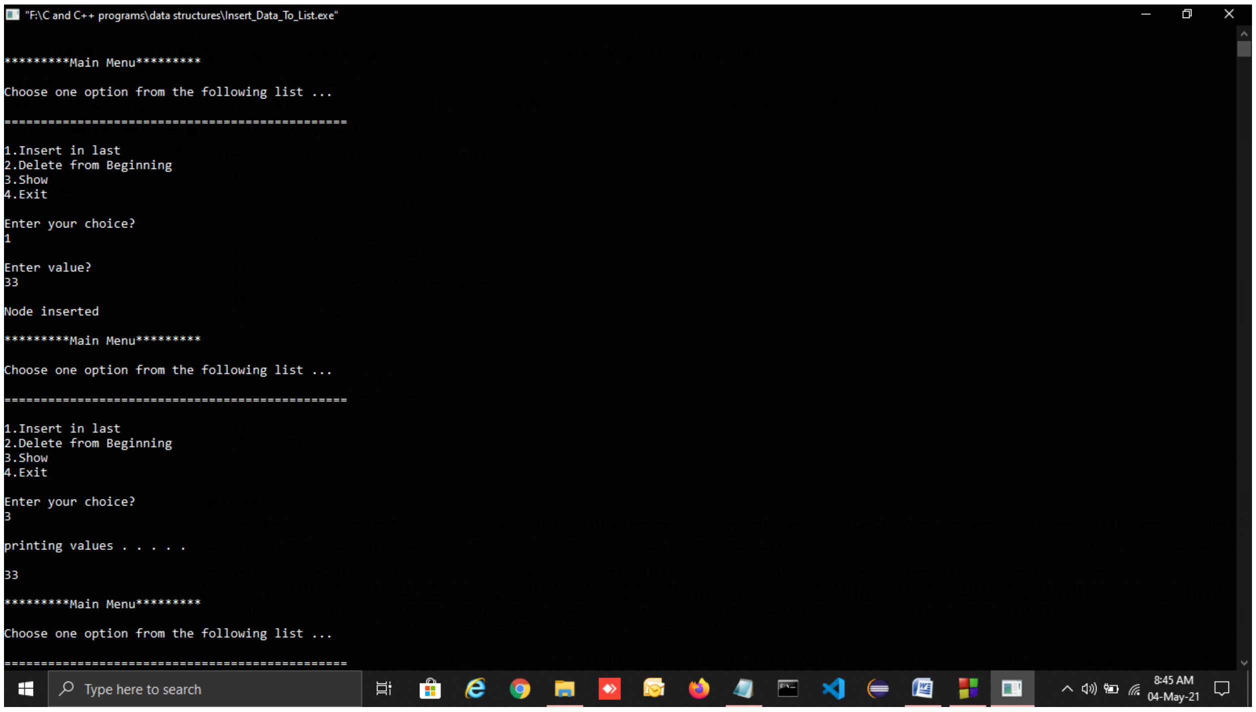
4.Store this node to deallocate it later using free() function.

5.Change this node pointer so that it points to it's next and by performing this previous node's next also get properly linked.

Algorithm to insert node at the end of the list :-

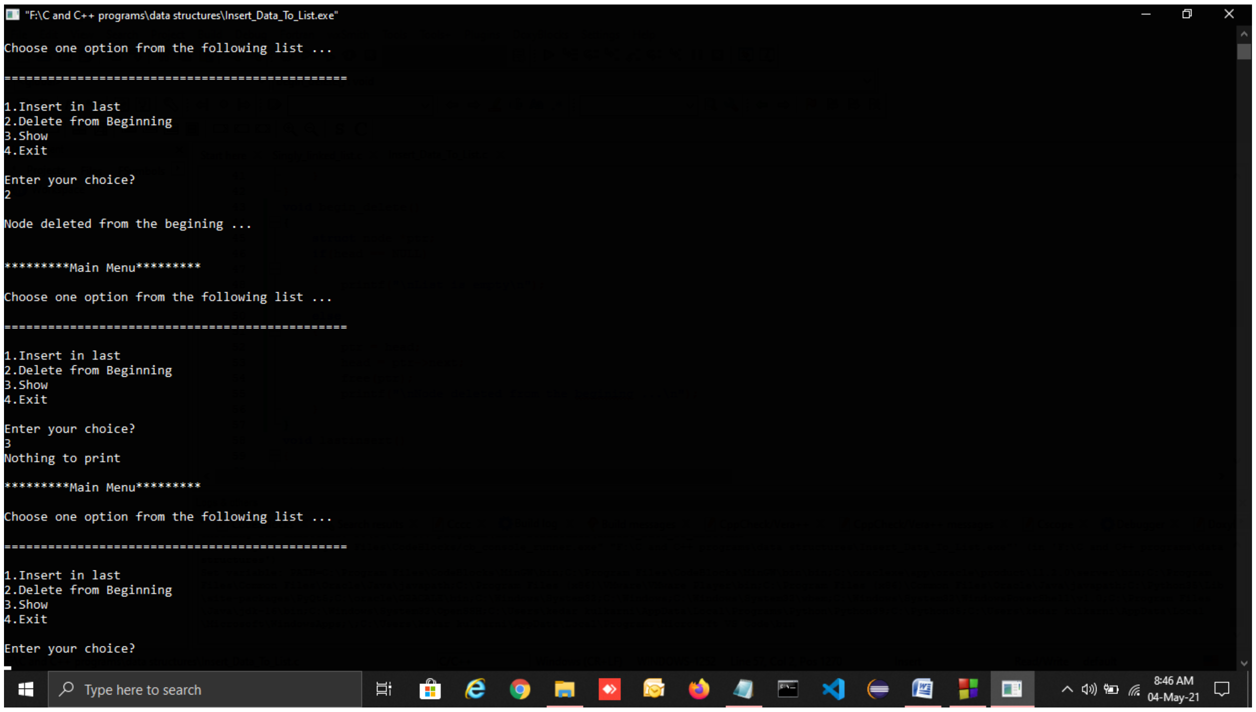
```
1: IF PTR = NULL Write OVERFLOW
    Go to Step 1
   [END OF IF]

2: SET NEW_NODE = PTR

3: SET PTR = PTR - > NEXT

4: SET NEW_NODE - > DATA = VAL

5: SET NEW_NODE - > NEXT = NULL

6: SET PTR = HEAD

7: Repeat Step 8 while PTR - > NEXT != NULL
```

```
8: SET PTR = PTR - > NEXT
[END OF LOOP]

9: SET PTR - > NEXT = NEW_NODE

10: EXIT
```



"F:\C and C++ programs\data structures\Insert_Data_To_List.exe"

```
*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in last
2.Delete from Beginning
3.Show
4.Exit

Enter your choice?
1

Enter value?
33

Node inserted

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in last
2.Delete from Beginning
3.Show
4.Exit

Enter your choice?
3

printing values . . . . .

33

*********Main Menu*********

Choose one option from the following list ...

===============================================
```



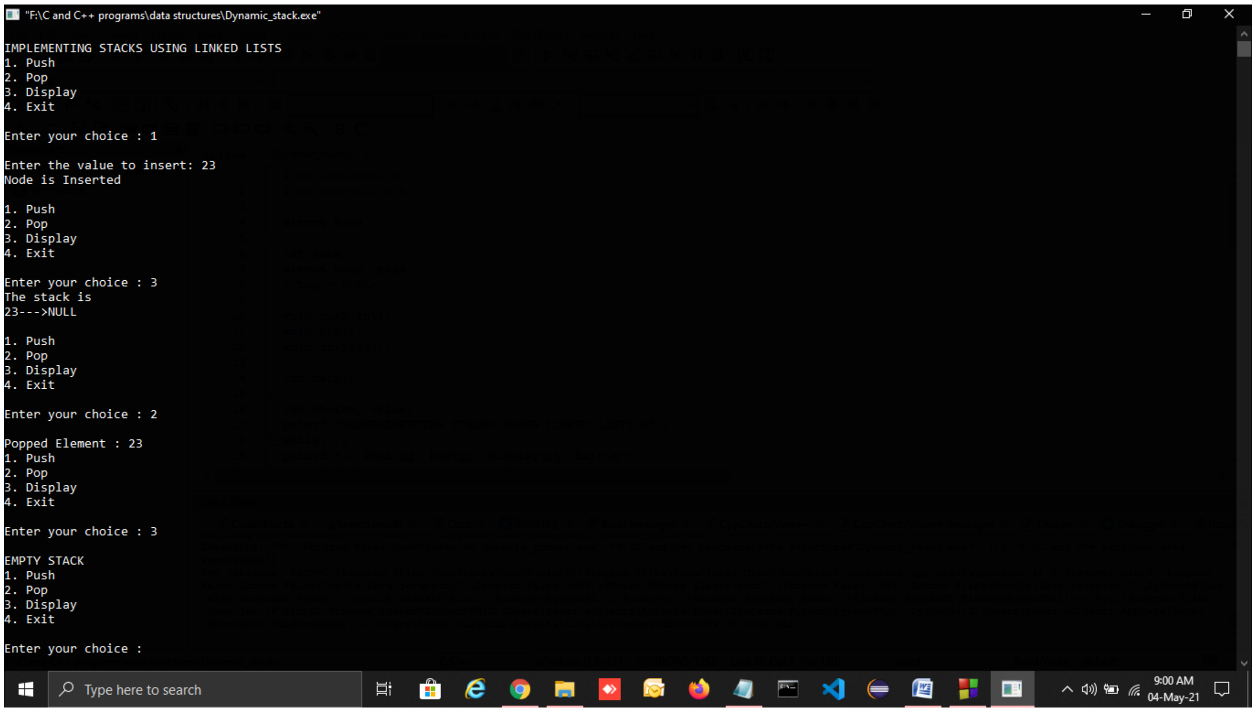
```
"F:\C and C++ programs\data structures\Insert_Data_To_List.exe"
Choose one option from the following list ...

===============================================

1.Insert in last
2.Delete from Beginning
3.Show
4.Exit

Enter your choice?
2

Node deleted from the begining ...

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in last
2.Delete from Beginning
3.Show
4.Exit

Enter your choice?
3
Nothing to print

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in last
2.Delete from Beginning
3.Show
4.Exit

Enter your choice?
```

Q.3] Write the program to implement the stack using linked list.

```
->#include<stdio.h>

#include<stdlib.h>


struct Node

{

int data;

struct Node *next;

}*top = NULL;


void push(int);

void pop();

void display();
```

```
int main()

{

int choice, value;

printf("\nIMPLEMENTING STACKS USING LINKED LISTS\n");

while(1){

printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1: printf("\nEnter the value to insert: ");

scanf("%d", &value);

push(value);

break;


case 2: pop();

break;


case 3: display();

break;


case 4: exit(0);

break;


default: printf("\nInvalid Choice\n");
}}}

void push(int value)

{

struct Node *newNode;

newNode = (struct Node*)malloc(sizeof(struct Node));

newNode->data = value;

if(top == NULL)

newNode->next = NULL;

else

newNode->next = top;

top = newNode;

printf("Node is Inserted\n\n");

}

void pop()

{

if(top == NULL)

printf("\nEMPTY STACK\n");

else{

struct Node *temp = top;

printf("\nPopped Element : %d", temp->data);

printf("\n");

top = temp->next;

free(temp);
```

```
}}


void display()

{

if(top == NULL)

printf("\nEMPTY STACK\n");

else

{

printf("The stack is \n");

struct Node *temp = top;

while(temp->next != NULL){

printf("%d--->",temp->data);

temp = temp -> next;

}

printf("%d--->NULL\n\n",temp->data);

}}
```



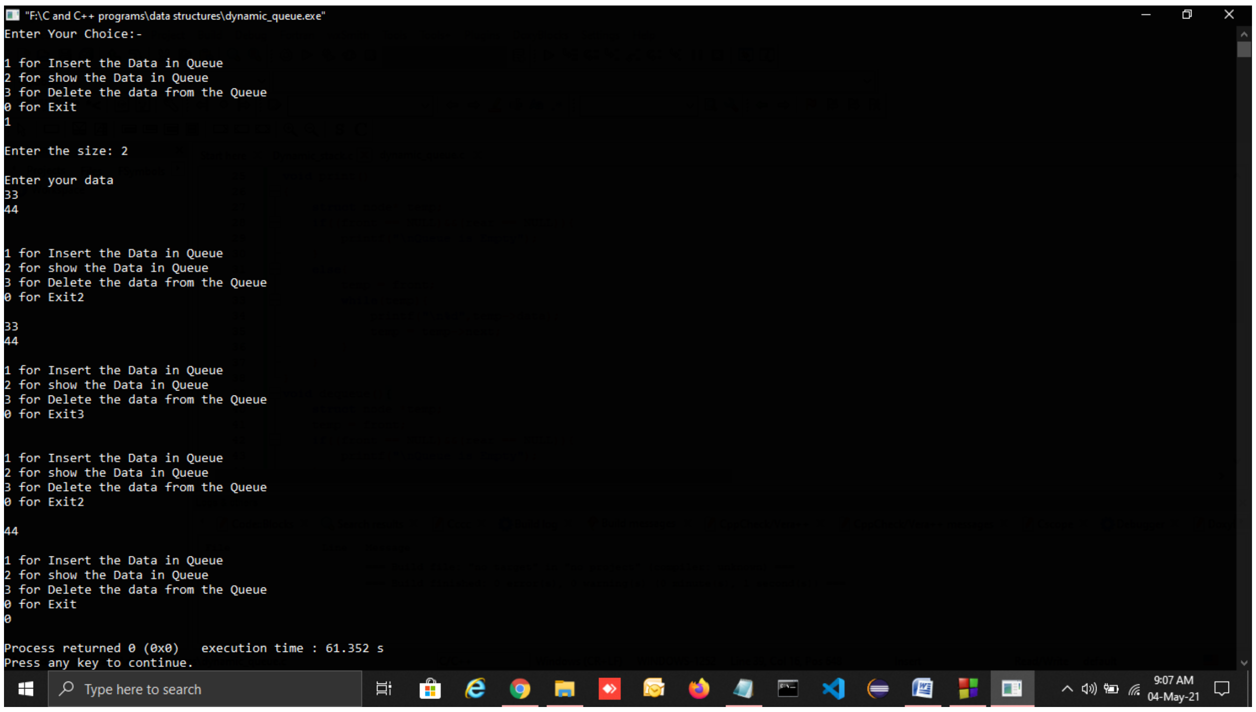
"F:\C and C++ programs\data structures\Dynamic_stack.exe"

```
IMPLEMENTING STACKS USING LINKED LISTS
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 23
Node is Inserted

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 3
The stack is
23--->NULL

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 2

Popped Element : 23
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 3

EMPTY STACK
1. Push
2. Pop
3. Display
4. Exit

Enter your choice :
```

Q.4] Write the program to implement the queue using linked list.

->
```
#include <stdio.h>

#include <stdlib.h>

struct node

{

	int data;

	struct node* next;

};

struct node *front = NULL;

struct node *rear = NULL;

void enqueue(int d)

{

	struct node* new_n;

	new_n = (struct node*)malloc(sizeof(struct node));

	new_n->data = d;
```

```
        new_n->next = NULL;
        if((front == NULL)&&(rear == NULL)){
                front = rear = new_n;
        }
        else{
                rear->next = new_n;
                rear = new_n;
        }


}
void print()
{
        struct node* temp;
        if((front == NULL)&&(rear == NULL)){
                printf("\nQueue is Empty");
        }
        else{
                temp = front;
                while(temp){
                        printf("\n%d",temp->data);
                        temp = temp->next;
                }
        }
}
void dequeue(){
        struct node *temp;
        temp = front;
        if((front == NULL)&&(rear == NULL)){
                printf("\nQueue is Empty");
        }
        else{
                front = front->next;
                free(temp);
        }
}
int main()
{
        int opt,n,i,data;
        printf("Enter Your Choice:-");
        while(opt!=0){
                printf("\n\n1 for Insert the Data in Queue\n2 for show the Data in Queue \n3 for Delete the data
from the Queue\n0 for Exit");
                scanf("%d",&opt);
                switch(opt){
                        case 1:
                                printf("\nEnter the size: ");
                                scanf("%d",&n);
                                printf("\nEnter your data\n");
                                i=0;
                                while(i<n){
                                        scanf("%d",&data);
                                        enqueue(data);
                                        i++;
                                }
                                break;
                        case 2:
                                print();
                                break;
                        case 3:
                                dequeue();
                                break;
                        case 0:
                                break;
                        default:
                                printf("\nIncorrect Choice");

                }
        }
}
```



"F:\C and C++ programs\data structures\dynamic_queue.exe"

```
Enter Your Choice:-

1 for Insert the Data in Queue
2 for show the Data in Queue
3 for Delete the data from the Queue
0 for Exit
1

Enter the size: 2

Enter your data
33
44


1 for Insert the Data in Queue
2 for show the Data in Queue
3 for Delete the data from the Queue
0 for Exit2

33
44

1 for Insert the Data in Queue
2 for show the Data in Queue
3 for Delete the data from the Queue
0 for Exit3


1 for Insert the Data in Queue
2 for show the Data in Queue
3 for Delete the data from the Queue
0 for Exit2

44

1 for Insert the Data in Queue
2 for show the Data in Queue
3 for Delete the data from the Queue
0 for Exit
0

Process returned 0 (0x0)   execution time : 61.352 s
Press any key to continue.
```

Q.5] Compare array implementation of list and dynamic implementation of list. Also mention how nodes are declared in both the implementation.

Write an algorithm to add long positive integer using circular lists.

-> **Difference between array implementation and dynamic implementation of list :-**

Size: Since data can only be stored in contiguous blocks of memory in an array, its size cannot be altered at runtime due to risk of overwriting over other data. However in a linked list, each node points to the next one such that data can exist at scattered (non-contiguous) addresses; this allows for a dynamic size which can change at runtime.
Memory allocation: For arrays at compile time and at runtime for linked lists. but, dynamically allocated array also allocates memory at runtime.
Memory efficiency: For the same number of elements, linked lists use more memory as a reference to the next node is also stored along with the data. However, size flexibility in linked lists may make them use less memory overall; this is useful when there is uncertainty about size or there are large variations in the size of data elements; memory equivalent to the upper limit on the size has to be allocated (even if not all of it is being used) while using arrays, whereas linked lists can increase their sizes step-by-step proportionately to the amount of data.
Execution time: Any element in an array can be directly accessed with its index; however in case of a linked list, all the previous elements must be traversed to reach any element. Also, better cache locality in arrays (due to contiguous memory allocation) can significantly improve performance. As a result, some operations (such as modifying a certain element) are faster in arrays, while some other (such as inserting/deleting an element in the data) are faster in linked lists.

Creating nodes in array implementation :-

Array declaration :

```
int A[1000];
int *A = (int *) malloc(1000 * sizeof(int));
```

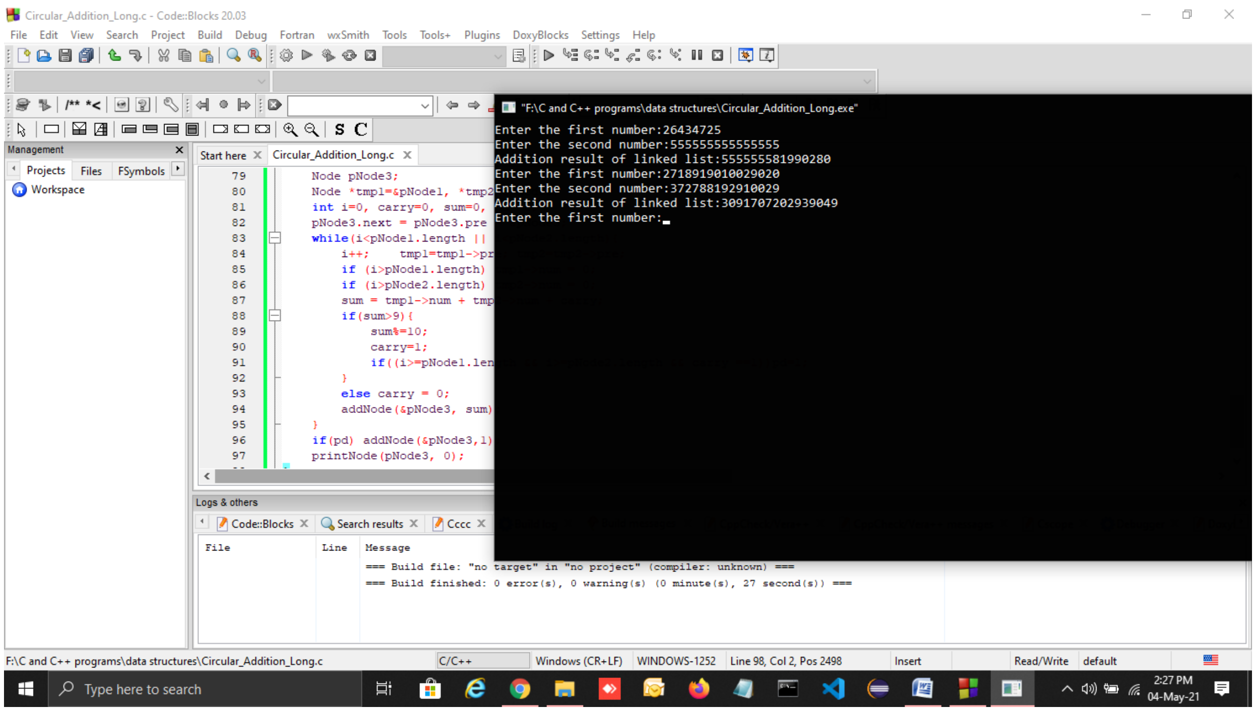
Accessing ith element :

```
A[i]
```

Creating a Node in dynamic implementation :-

.

```
typedef struct LinkedList *node;
node createNode(){
     node temp;
     temp = (node)malloc(sizeof(struct LinkedList));     temp->next = NULL
return temp;
}
```



```
    Node pNode3;
    Node *tmp1=&pNode1, *tmp2
    int i=0, carry=0, sum=0,
    pNode3.next = pNode3.pre
    while(i<pNode1.length ||
        i++;    tmp1=tmp1->pr
        if (i>pNode1.length)
        if (i>pNode2.length)
        sum = tmp1->num + tmp
        if(sum>9){
            sum%=10;
            carry=1;
            if((i>=pNode1.len
        }
        else carry = 0;
        addNode(&pNode3, sum)
    }
    if(pd) addNode(&pNode3,1)
    printNode(pNode3, 0);
```

```
"F:\C and C++ programs\data structures\Circular_Addition_Long.exe"
Enter the first number:26434725
Enter the second number:555555555555555
Addition result of linked list:555555581990280
Enter the first number:2718919010029020
Enter the second number:372788192910029
Addition result of linked list:3091707202939049
Enter the first number:_
```

```
File        Line    Message
                    === Build file: "no target" in "no project" (compiler: unknown) ===
                    === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 27 second(s)) ===
```

Circular_Addition_Long.c - Code::Blocks 20.03
F:\C and C++ programs\data structures\Circular_Addition_Long.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 98, Col 2, Pos 2498
Insert
Read/Write
default
2:27 PM
04-May-21

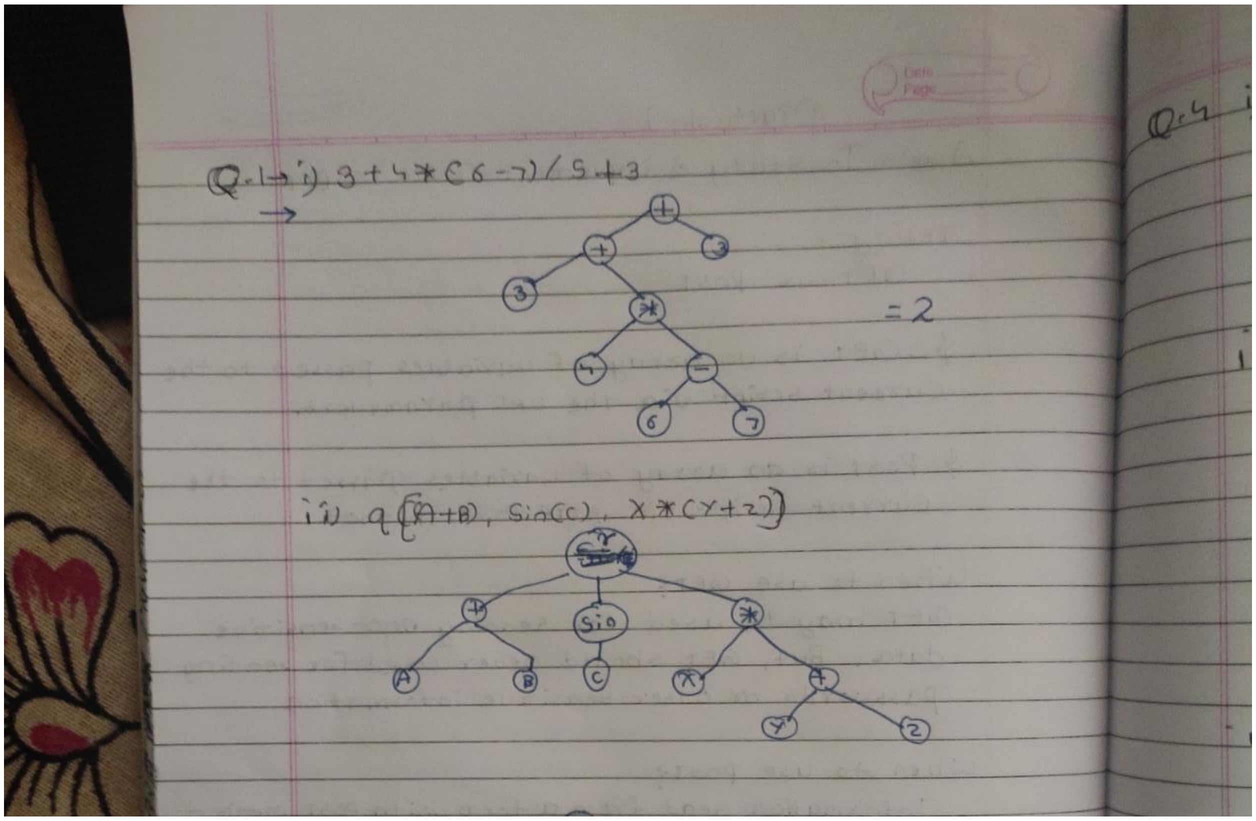
Assignment :- 2

Q.1] Construct binary tree for the following expression

I] 3+4*(6-7)/5+3

II] v(A+B, sin (C), X *(Y+Z))

->



Q.2] Write a program to create binary search tree.

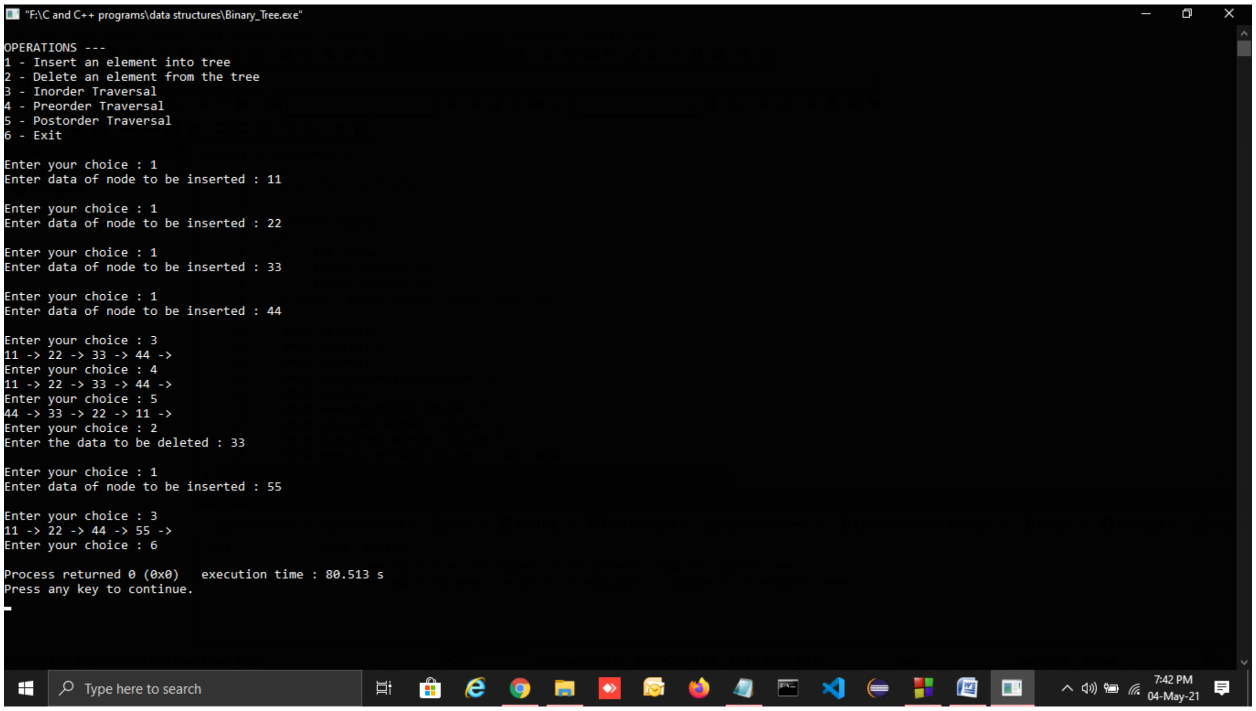
->

```
#include <stdio.h>

#include <stdlib.h>


struct btnode

{

   int value;

   struct btnode *l;

   struct btnode *r;

}*root = NULL, *temp = NULL, *t2, *t1;


void delete1();
void insert();
void delete();
void inorder(struct btnode *t);
void create();
void search(struct btnode *t);
void preorder(struct btnode *t);
void postorder(struct btnode *t);
void search1(struct btnode *t,int data);
int smallest(struct btnode *t);
int largest(struct btnode *t);

int flag = 1;

void main()
{
   int ch;

    printf("\nOPERATIONS ---");
    printf("\n1 - Insert an element into tree\n");
    printf("2 - Delete an element from the tree\n");
    printf("3 - Inorder Traversal\n");
    printf("4 - Preorder Traversal\n");
    printf("5 - Postorder Traversal\n");
    printf("6 - Exit\n");
    while(1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            insert();
            break;
        case 2:
            delete();
            break;
        case 3:
            inorder(root);
            break;
        case 4:
            preorder(root);
            break;
        case 5:
            postorder(root);
            break;
        case 6:
            exit(0);
        default :
            printf("Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}
```

```
void insert()
{
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

void create()
{
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}

void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
    else if ((temp->value < t->value) && (t->l != NULL))
```

```
        search(t->l);
    else if ((temp->value < t->value) && (t->l == NULL))
        t->l = temp;
}

void inorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    if (t->l != NULL)
        inorder(t->l);
    printf("%d -> ", t->value);
    if (t->r != NULL)
        inorder(t->r);
}

void delete()
{
    int data;

    if (root == NULL)
    {
        printf("No elements in a tree to delete");
        return;
    }
    printf("Enter the data to be deleted : ");
    scanf("%d", &data);
    t1 = root;
    t2 = root;
    search1(root, data);
}

void preorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d -> ", t->value);
    if (t->l != NULL)
        preorder(t->l);
    if (t->r != NULL)
        preorder(t->r);
}

void postorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display ");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d -> ", t->value);
}

void search1(struct btnode *t, int data)
{
    if ((data>t->value))
    {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value))
    {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value))
    {
        delete1(t);
    }
}
```

```
void delete1(struct btnode *t)
{
    int k;

    if ((t->l == NULL) && (t->r == NULL))
    {
        if (t1->l == t)
        {
            t1->l = NULL;
        }
        else
        {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
        return;
    }

    else if ((t->r == NULL))
    {
        if (t1 == t)
        {
            root = t->l;
            t1 = root;
        }
        else if (t1->l == t)
        {
            t1->l = t->l;


        }
        else
        {
            t1->r = t->l;
        }
        t = NULL;
        free(t);
        return;
    }


    else if (t->l == NULL)
    {
        if (t1 == t)
        {
            root = t->r;
            t1 = root;
        }
        else if (t1->r == t)
            t1->r = t->r;
        else
            t1->l = t->r;
        t == NULL;
        free(t);
        return;
    }


    else if ((t->l != NULL) && (t->r != NULL))
    {
        t2 = root;
        if (t->r != NULL)
        {
            k = smallest(t->r);
            flag = 1;
        }
        else
        {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }


}


int smallest(struct btnode *t)
{
    t2 = t;
    if (t->l != NULL)
    {
        t2 = t;
        return(smallest(t->l));
    }
    else
        return (t->value);
}


int largest(struct btnode *t)
{
    if (t->r != NULL)
    {
        t2 = t;
        return(largest(t->r));
    }
    else
        return(t->value);
}
```



```
"F:\C and C++ programs\data structures\Binary_Tree.exe"

OPERATIONS ---
1 - Insert an element into tree
2 - Delete an element from the tree
3 - Inorder Traversal
4 - Preorder Traversal
5 - Postorder Traversal
6 - Exit

Enter your choice : 1
Enter data of node to be inserted : 11

Enter your choice : 1
Enter data of node to be inserted : 22

Enter your choice : 1
Enter data of node to be inserted : 33

Enter your choice : 1
Enter data of node to be inserted : 44

Enter your choice : 3
11 -> 22 -> 33 -> 44 ->
Enter your choice : 4
11 -> 22 -> 33 -> 44 ->
Enter your choice : 5
44 -> 33 -> 22 -> 11 ->
Enter your choice : 2
Enter the data to be deleted : 33

Enter your choice : 1
Enter data of node to be inserted : 55

Enter your choice : 3
11 -> 22 -> 44 -> 55 ->
Enter your choice : 6

Process returned 0 (0x0)   execution time : 80.513 s
Press any key to continue.
```

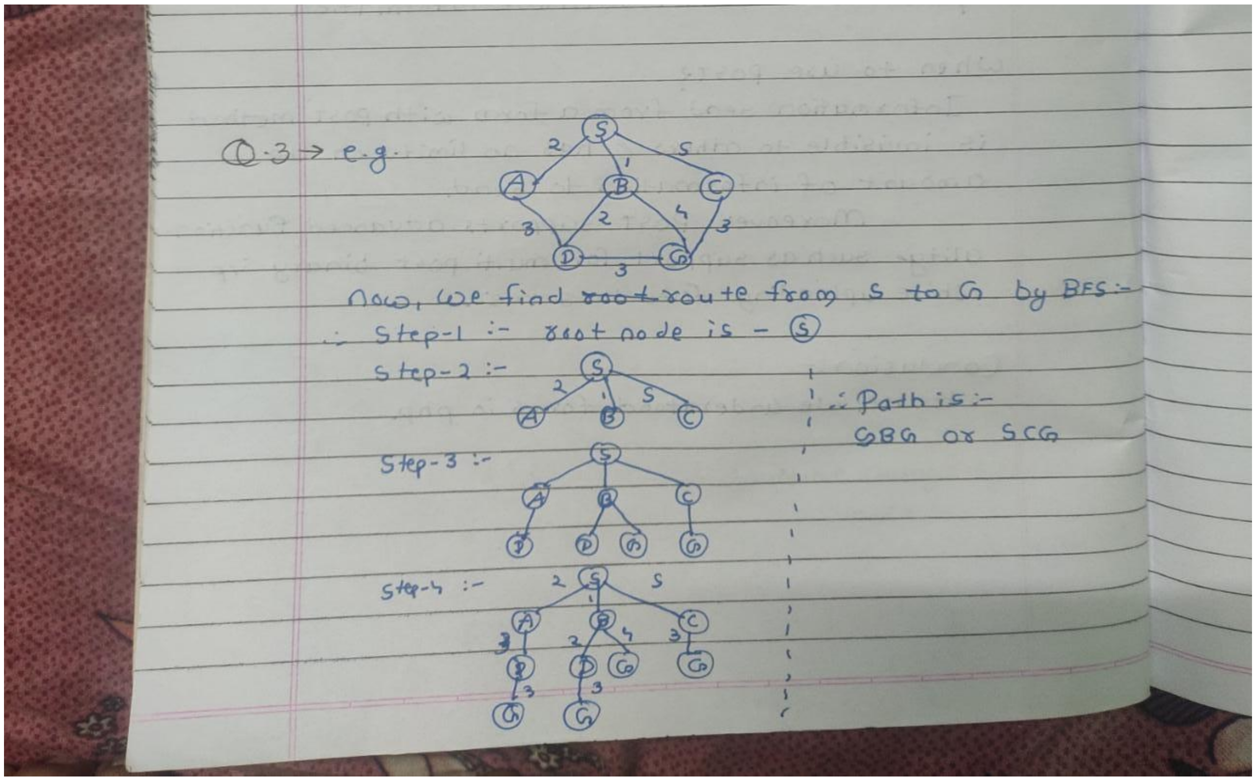
Q.3] Describe Breadth First Search Traversal Technique for graph with example.

->Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighbouring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal.

The algorithm of breadth first search is given below. The algorithm starts with examining the node A and all of its neighbours. In the next step, the neighbours of the nearest node of A are explored and process continues in the further steps. The algorithm explores all neighbours of all the nodes and ensures that each node is visited exactly once and no node is visited twice.

Algorithm

- **Step 1:** SET STATUS = 1 (ready state)
  for each node in G
- **Step 2:** Enqueue the starting node A
  and set its STATUS = 2
  (waiting state)
- **Step 3:** Repeat Steps 4 and 5 until
  QUEUE is empty
- **Step 4:** Dequeue a node N. Process it
  and set its STATUS = 3
  (processed state).
- **Step 5:** Enqueue all the neighbours of
  N that are in the ready state
  (whose STATUS = 1) and set
  their STATUS = 2
  (waiting state)
  [END OF LOOP]
- **Step 6:** EXIT



Q.3 → e.g.

Now, we find ~~root~~ route from S to G by BFS :-

∴ Step-1 :- root node is - (S)

Step-2 :-

Step-3 :-

Step-4 :-

∴ Path is :-
SBG or SCG

Q.4] Describe the following terms with example

i)DAG ii) Weighted graph iii) Acyclic graph iv) in degree & v)out degree

->DAG representation :-

A DAG for basic block is a directed acyclic graph with the following labels on nodes:

1. The leaves of graph are labeled by unique identifier and that identifier can be variable names or constants.
2. Interior nodes of the graph is labeled by an operator symbol.
3. Nodes are also given a sequence of identifiers for labels to store the computed value.

- DAGs are a type of data structure. It is used to implement transformations on basic blocks.
- DAG provides a good way to determine the common sub-expression.
- It gives a picture representation of how the value computed by the statement is used in subsequent statements.

Weighted graphs :-

A **weighted graph** refers to a simple graph that has weighted edges. These weighted edges can be used to compute the shortest path. It consists of:

- A set of vertices *V*.
- A set of edges *E*.
- And a number *w* (with *w* meaning weight) that's assigned to each edge. Weights might represent things such as costs, lengths, or capacities.

In a simple graph, the assumption is that the sum of all the weights is equal to 1.

Acyclic graphs :-

An acyclic graph is a graph without cycles (a cycle is a complete circuit). When following the graph from node to node, you will never visit the same node twice.

In-degree and Out-degree :-

Indegree of a Graph

- Indegree of vertex V is the number of edges which are coming into the vertex V.
- **Notation** − $\deg^{-}(V)$.

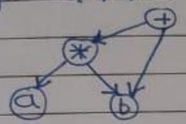
Outdegree of a Graph

- Outdegree of vertex V is the number of edges which are going out from the vertex V.
- **Notation** − $\deg^{+}(V)$.

Consider the following examples.

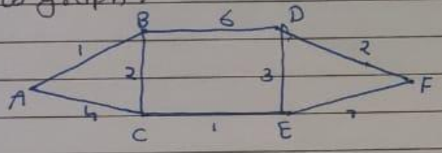
Q.4 i) DAG :- (Direct Acyclic Graph)

e.g. a * b + b



ii) Weighted graph :-

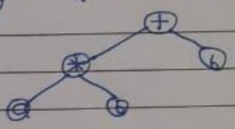
e.g.



for weighted graph shortest path from A to F is :- ABCEDF or ABDF & which is equals to 9.

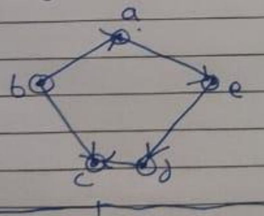
iii) Acyclic graph :-

e.g. a * b + b



iv) In-degree & out-degree vertex of graph :-

e.g.



Vertex	a	b	c	d	e
Indegree	1	0	2	1	1
Outdegree	1	2	0	1	1