

(1) Write a program to push, pop and display the elements of a stack.

```

1. #include <stdio.h>
2. void Push();
3. void Pop();
4. void display();
5. int stack[50], p, q, n, option=0, top=-1;
6. void main()
7. {
8.     printf("Enter the number of elements:");
9.     scanf("%d", &n);
10.    while(option!=4)
11.    {
12.        printf("Choose one from the below option:\n");
13.        printf("(1)Push (2)Pop (3)Display (4) Exit");
14.        scanf("%d", &option);
15.        switch(option){
16.            case 1:
17.                push();
18.                break;
19.            case 2:
20.                pop();
21.                break;
22.            case 3:
23.                display();
24.                break;
25.            case 4:
26.                printf("Exit");
27.                break;
28.        } // End of switch case
29.    } // End of while loop
30. }
31. // Default : (1)Push

```

work 186 from 90% point ("Select the proper option.");
} - Now go to drawing tools

W.D. Pfeiffer & Schmid 917
? (1) 809 6.90v
? (1) 909 6.90v

```

void push() {
    if (top == n - 1)
        printf("In overflow");
}

```

Retrieving contact point ("Enter the value:");

1987 (Nineteen) Scanf ("%d", &value); // Reading

`top=topfigs[0, "balo"]` `figsize`

`Stack[top] = value;`

```
void pop() {
```

if (top == -1) {

~~point ("in UNDERFLOW");~~

3

use {

point f ("The popped element will be %od\n")

Stack [Top]);

$$\text{top} = \text{top} - 1;$$

3

~~void display() {~~

for ($i = \text{top}; i > 0$; $i--$) {

```
popoff ("0/od\n", stack[?]);
```

if $\text{top} == -1$ {
 cout << "Stack is empty.";

}

else

top = top - 1;

cout << top;

cout << "

Top is ";

O/P Enter the number of elements : 5

choose one from the below option: 1

1) push

2) pop

3) display

4) Exit

1

Enter the value : 8

choose one from the below option: 1

1) push

2) pop

3) display

4) exit

6

select the proper option and type

choose one from the below option: 1

1) push

2) pop

3) display

4) exit

2

8

choose one from the below option: 1

1) push

2) pop

3) display

4) exit

2

The popped element will be 89.

(Choose one from the below options)

1) Push

2) POP

3) DISPLAY

4) EXIT

4: Program ends with some words

EXIT

*Ques (2)
Gf. 09.24*
Write a program to convert an infix expression into its equivalent postfix notation.

I/P char infix[20], result[20]; stack[20]; int top=0; char result[20]; int pos=0;

void push(char);

char pop();

int order(char);

void main()

int i=0;

printf("Enter Infix:");

scanf("%s", infix);

while (infix[i] != '\0')

symbol = infix[i];

switch (symbol){

case '(': stack[++top] = '(';

push(symbol);

break;

else if (symbol == ')') {

while (stack[top] != '(')

result = pop();

post[post++]=result;

```

top --> i = 0; stack[0] = A; stack[1] = B; stack[2] = C; stack[3] = D; stack[4] = E; stack[5] = F; stack[6] = G; stack[7] = H; stack[8] = I; stack[9] = J; stack[10] = K; stack[11] = L; stack[12] = M; stack[13] = N; stack[14] = O; stack[15] = P; stack[16] = Q; stack[17] = R; stack[18] = S; stack[19] = T; stack[20] = U; stack[21] = V; stack[22] = W; stack[23] = X; stack[24] = Y; stack[25] = Z; stack[26] = a; stack[27] = b; stack[28] = c; stack[29] = d; stack[30] = e; stack[31] = f; stack[32] = g; stack[33] = h; stack[34] = i; stack[35] = j; stack[36] = k; stack[37] = l; stack[38] = m; stack[39] = n; stack[40] = o; stack[41] = p; stack[42] = q; stack[43] = r; stack[44] = s; stack[45] = t; stack[46] = u; stack[47] = v; stack[48] = w; stack[49] = x; stack[50] = y; stack[51] = z; stack[52] = .; stack[53] = ,; stack[54] = ;; stack[55] = (;
```

break;
 case '+':
 case '-':
 case '^':
 case '*':
 case '/':
 while (order(sym) < order(stack[top])) {
 result = pop();
 post[p++] = result;
 }
 if (order(sym) > order(stack[top])) {
 push(sym);
 }
 else if (order(sym) == order(stack[top])) {
 if (order(sym) == 1 || (order(sym) == 2)) {
 result = pop();
 post[p++] = result;
 push(sym);
 }
 else if (order(sym) == order(stack[top]) &&
 order(sym) == 3) {
 push(sym);
 }
 break;
 }
 i++;
} while (top > 0) {
 post[p++] = stack[top];
}

void Push (char x)

{

 push x onto stack

 top = top + 1

 stack [top] = x

}

char pop () {

{

 char x = stack [top];

 top = top - 1;

 return x;

}

int order (char symb) {

{

 if (symb == '+' || symb == '-' || symb == '*' || symb == '/')

 return 1;

 else if (symb == '^')

 return 2;

 else if (symb == '(' || symb == ')')

 return 3;

 else if (symb == '^' || symb == '*' || symb == '/')

 return 4;

 else if (symb == '+' || symb == '-')

 return 5;

 else if (symb == 'x' || symb == 'y' || symb == 'z')

 return 6;

 else if (symb == '1' || symb == '2' || symb == '3')

 return 7;

 else if (symb == '4' || symb == '5' || symb == '6')

 return 8;

 else if (symb == '7' || symb == '8' || symb == '9')

 return 9;

O/P Enter infix : A * B + C * D - E

Postfix is : A B * C D * + E -

Q1 Write a program to simulate the working of the queue of integers using an array. Provide the following operations: Insert, delete, display.

The program should print appropriate message for overflow and underflow condition.

I/P #include <stdio.h>

#define n=2

int q[n], rear=-1, front=-1;

void add();

void del();

void display();

void main()

int choice=0, num;

while(choice!=4)

printf("Enter choice:");

scanf("%d", &choice);

switch(choice){

case 1:

printf("Add number:");

scanf("%d", &num);

add(num);

break;

case 2:

printf("Delete");

num = del();

break;

case 3:

printf("Display");

display();

break;

case 4:

printf("Exit");

front = -1; rear = -1;

if (front == -1 & rear == -1) break;

else if (front < rear) cout << "Queue is empty now";

else if (front == rear) cout << "Queue is full now";

else cout << "Queue is not full and not empty now";

void add (int p) { queue[rear] = p; rear++; }

```
if (front == -1 & rear == -1) cout << "Queue is empty now";  
{  
    queue[front] = p;  
    front++;  
    cout << "Queue is not empty now";  
}  
else if (rear == n) cout << "Queue is full now";  
else {  
    queue[rear] = p;  
    rear++;  
    cout << "Queue is not full and not empty now";  
}
```

```
else if (rear == n)  
{  
    cout << "Queue is full now";  
}
```

```
cout << "Queue is now " << queue[front];
```

```
{  
    cout << "Queue is now " << queue[front];  
}
```

```
else cout << "Queue is not full and not empty now";
```

```
{  
    cout << "Queue is not full and not empty now";  
}
```

```
queue[rear] = p;
```

```
{  
    cout << "Queue is not full and not empty now";  
}
```

```
cout << "Queue is now " << queue[front];
```

```
{  
    cout << "Queue is now " << queue[front];  
}
```

```
void del () {  
    cout << "Queue is now " << queue[front];  
}
```

```
{  
    cout << "Queue is now " << queue[front];  
}
```

```
if (front == -1 & rear == -1)
```

```
{  
    cout << "Queue is empty now";  
}
```

```
cout << "Queue is now empty";
```

```
{  
    cout << "Queue is now empty";  
}
```

```
cout << "Queue is now empty now";
```

```
front = -1; rear = -1;
```

```
{  
    cout << "Queue is now empty now";  
}
```

```
else {  
    cout << "Queue is not empty now";  
}
```

~~frontiers of science & art~~

~~int k = Q[front];~~

point ("old is removed", v);

~~2010-09-20 Drosophilae, female, brownish red~~

Alphonse Moustier, a man from the area, with

void display()

int q;

If ($\text{eax} == -1$ || $\text{fout} == -1$) { good(); }

3

else {

```
for (i = front + 1; i <= rear; i++) {
```

present ("or odd", $\{f\}$) to $\{f\}$ above

3

point ("\\n");

۱

Digitized by Google on 2016-10-19 19:19:09

2

Op Enter choices 1

Adel number: 25

Enter choice: 1

Add number : 24

Enter choice: 3

display ~~display-block (gives block level)~~ ~~display-block (gives block level)~~

25 24 (most likely 25) Massachusetts 17

Enter choice 2

Delet e

Enter angle: 2

Peyote

Enter choice: 2 (Second + third) prints

Queue is empty now

Answers will vary. It would probably

Exit.

Q2 write a program to simulate the working of a circular queue using an array. provide the following operations: insert, delete & display. The program should print appropriate message for queue empty and queue overflow condition.

```

I/P #include <stdio.h>
#define n=3
void enqueue(int); int q[n];
void dequeue();
void display();
int rear=-1, front=-1;
main()
{
    enqueue(1);
    enqueue(2);
    enqueue(3);
    display();
    dequeue();
    enqueue(4);
    enqueue(5);
    display();
    dequeue();
    enqueue(6);
    display();
}

void enqueue(int p)
{
    if (rear==n-1 && front==0)
        printf("Queue is full");
    else if (front == -1)
        front = 0;
    rear++;
    q[rear] = p;
    if (rear == n-1)
        printf("Queue is full");
    else
        printf("%d", q[rear]);
}

void dequeue()
{
    if (front == -1)
        printf("Queue is empty");
    else
        front++;
}

```

```
else {  
    rear = (rear + 1) % n;  
    q[rear] = p;  
}  
}  
void display() {  
    int t = q[front];  
    if (rear == -1 && front == -1)  
        printf("Queue is empty");  
    else if ((front + 1) % n == (rear + 1))  
        printf("Queue is empty now");  
    else  
        front = (front + 1) % n;  
        t = q[front];  
        printf("Old removed", t);  
}  
void display() {  
    int t = q[front];  
    if (rear == -1 && front == -1)  
        printf("Queue is empty");  
    else if (rear == front)  
        i = front + 1;  
        while (i != rear) {  
            printf("Old", q[i]);  
            i = (i + 1) % n;  
        }  
        printf("Old", q[front]);  
}
```

printf(" \n");

{

use {

i = front + 1;

while (i != rear) {

printf("odd", q[5973]get(6));

j = (i + 1) % n;

if (q[j] == 1) {

printf("odd", q[5973]get(6));

printf(" \n");

}

} (for loop) = main function 28 323

O/P Queue is full in main() Function

25 034045500 - front + 1 = rear

45 044045500 -

front = 0 (0 + 1) % 5 = 1

if (q[1] == 1) {

printf("odd", q[5973]get(6));

front = 1

if (q[1] == 1) {

printf("odd", q[5973]get(6));

front = 2

if (q[2] == 1) {

printf("odd", q[5973]get(6));

front = 3

if (q[3] == 1) {

printf("odd", q[5973]get(6));

front = 4

if (q[4] == 1) {

printf("odd", q[5973]get(6));

front = 5

- Q-5 write a program to implement singly linked list with following operations.
- (a) Create a Linked List.
 - (b) Insertion of a node at first position at any position and at end of list display the contents of linked list.

```

I/P: #include <stdio.h>
      #include <stdlib.h>
      struct Node {
          int data;
          struct Node * next;
      };
      struct Node * createnode(int data) {
          struct Node * newnode = (struct Node *)
              malloc(sizeof(struct Node));
          if (newnode == NULL) {
              printf("Memory allocation failed\n");
              exit(1);
          }
          newnode->data = data;
          newnode->next = NULL;
          return newnode;
      }
  
```

```

Struct Node * createLinkedList (int values[], int size) {
    struct Node * head = NULL;
    struct Node * tail = NULL;
    for (int i=0; i<size; i++) {
        struct Node * newNode = createnode(values[i]);
        if (head == NULL) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
}
  
```

13. 2023 sample else {
 tail → next = newNode; } pushNode
 tail = newNode; } else {
 pointer to previous first statement or An error at (1)
 void insertFirst (struct Node** head, int data) {
 struct Node* newNode = createNode(data);
 newNode → next = * head; ^{below this}
 * head = newNode; ^{below this}
 }

void insertFirst (struct Node** head, int data) {
 struct Node* newNode = createNode(data);
 newNode → next = * head; ^{below this}
 * head = newNode; ^{below this}
 }

void insertAtPosition (struct Node** head, int data,
 int position) {
 if (position == 0) {
 insertFirst(head, data);
 return; } else {
 struct Node* current = head;
 for (int i=0; i<position-1; i++) {
 if (current == NULL) {
 printf("Invalid position\n");
 return; } else {
 current = current → next; }
 }
 current → next = newNode; ^{below this}
 newNode → next = current → next; ^{below this}
 current → next = newNode; ^{below this}
 return; }
}

if (current == NULL) {
 printf("Invalid position\n");
 return; } else {
 struct Node* current = head;
 for (int i=0; i<position-1; i++) {
 if (current == NULL) {
 printf("Invalid position\n");
 return; } else {
 current = current → next; }
 }
 current → next = newNode; ^{below this}
 newNode → next = current → next; ^{below this}
 current → next = newNode; ^{below this}
 return; }
}

void insertEnd (struct Node** head, int data) {
 struct Node* newNode = createNode(data);
 if (*head == NULL) {
 *head = newNode; } else {
 struct Node* current = head;
 for (int i=0; i<position-1; i++) {
 if (current == NULL) {
 printf("Invalid position\n");
 return; } else {
 current = current → next; }
 }
 current → next = newNode; ^{below this}
 newNode → next = NULL; ^{below this}
 return; }
}

*head = now node; our working spot

return; after making link with it

}

else { N S C T }

start node *current = head; & add 0

while (current->next != NULL) {

current = current->next; (PAN & NH S S)

{ 2 forms are to handle of making link

current->next = new_node; & L & 2

}

void display (start node *head) { for show - S-S

while (head != NULL) { always print next

printf ("odd->" & head->data);

then make back track head = head->next; & making C

if nothing left { 2 cases, if no element then, break

printf ("NULL in"); after for loop for

}

int main () {

start node * linkedList = createLinkedList ();

int data; for both threads

printf ("Enter data to insert at beginning:");

scanf ("%d", & data); else thread

insertFirst (& linkedList, data); - f

int position; for popping & decreasing & other thread

(* printf ("Enter data to insert at a specific

position:")); then

scanf ("%d", & data); & breaking if

scanf ("Enter position:");

scanf ("%d", & position);

insertAtPosition (& linkedList, data, position);

printf ("Enter data to insert at end:");

scanf ("%d", & data); & break if

insertEnd (& linkedList, data);

display (linkedList);

return 0;

Q1P

Enter number of elements: 4

Enter the elements:

1 2 3 4

Enter data to insert at the beginning: 5

Enter data to insert at a specific position: 34

Enter the position: 2

Enter data to insert at the end: 66

5 → 1 → 34 → 2 → 3 → 4 → 66 → NULL

Q-2

WAP to implement singly linked list with following operations (labour) 9 P.M.

(a) Create a linked list.

(b) Deletion of first element, specified element and last element in list, display the contents of linked list.

I/P

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

```
#include <stdlib.h> // for dynamic memory allocation
```

```
struct Node {
```

```
    int data; // data part of node
```

```
    struct Node *next; // next part of node
```

```
}; // closing brace of struct Node
```

```
struct Node *createNode(int data) {
```

```
    struct Node *newNode = (struct Node *)
```

```
    malloc(sizeof(struct Node));
```

```
    if (newNode == NULL) {
```

```
        printf("Memory allocation failed");
```

```
        exit(1);
```

```
    newNode->data = data; // data part
```

```
    newNode->next = NULL; // next part
```

```
    return newNode; // returning address of head
```

```
}
```

```

struct Node* create_LinkedList() {
    struct Node* head = NULL;
    struct Node* tail = NULL;
    int size, data; // for loop
    printf("Enter the number of elements:");
    scanf("%d", &size);
    printf("Enter the elements:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &data);
        struct Node* newNode = createNode(data);
        if (head == NULL) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}

void deleteFirst(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Nothing to delete.");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

void deerefement(struct Node** head, int key) {
    if (*head == NULL) {
        printf("List is empty. Nothing to delete.");
    }
}

```

return; // if current == head, then list is empty

else { // list is not empty & head != null

struct Node *current = *head;

struct Node *prev = NULL;

if (current != NULL) { // current->data != head

prev = current; // prev

if (current->next == NULL) { // next

current->next = head; // head

if (current == NULL) { // head

printf("Element is not found in list\n");
return; // head == head

else { // element found

if (prev == NULL) { // first node

*head = current->next; // head = current->next

else { // not first node

prev->next = current->next; // previous node's next = current->next

free(current); // free current

} // end if (prev != NULL)

else free(current); // free current

void deregList(struct Node **head) {

if (*head == NULL) { // list is empty

printf("List is empty. Nothing to delete\n");

else { // list is not empty

return; // head == head

if ((*head)->next == NULL) { // head

free(*head); // free head

*head = NULL; // head = NULL

return; // head == head

else { // list is not empty & head != null

struct Node *prev = NULL;

while (current->next != NULL) {

prev = current; *in every node, prev is the previous node* 982

current = current -> next; *move current*

}

if (prev == 1)

prev -> next = NULL; *make first node*

free(current); *free current*

}

void display(struct Node * head) { *function to print list*

while (head != NULL) { *traverse list*

printf(" %d => " , head-> data);

head = head -> next; *move to next*

}

printf("\nNULL\n"); *print NULL at end*

}

int main() {

struct Node * linkedList = createLinkedList();

printf(" Linked list before deleting first element: \n");

display(linkedList);

deleteFirst(& linkedList);

printf(" Linked list after deleting first element: \n");

display(linkedList);

int key;

printf(" Enter the element to delete: ");

scanf("%d", &key);

deleteElement(& linkedList, key);

printf(" Linked list after deleting specified element: \n");

display(linkedList);

deleteLast(& linkedList);

printf(" Linked list after deleting last element: \n");

display(linkedList);

return 0;

}

Q.P

Enter number of elements? $4 \rightarrow 1 = 1989$

Enter the elements? $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \text{NULL}$

Linked list before deletion: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \text{NULL}$

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \text{NULL}$ (linked list diagram)

Linked list after deleting first element:

$2 \rightarrow 3 \rightarrow 4 \rightarrow \text{NULL}$ (linked list diagram)

Entered the after deleting first element:

Enter the element to delete? 4

Linked list after deleting specified element:

$2 \rightarrow 3 \rightarrow \text{NULL}$

Linked list after deleting last element:

$(2 \rightarrow \text{NULL})$ both ends have been nullified

Q-1) Write a program to sort, reverse and concatenate a linear linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
};
void insert_at_begin(struct node *head, int data) {
    struct node *newnode = (struct node*) malloc(sizeof(struct node));
    newnode->data = data;
    newnode->next = head;
    head = newnode;
}
void printlist(struct node *head) {
    while (head != NULL) {
        printf("%d\n", head->data);
        head = head->next;
    }
    printf("\n");
}
void sortlist(struct node **head) {
    struct node *current, *nextnode;
    int temp;
    current = *head;
    while (current != NULL) {
        nextnode = current->next;
        while (nextnode != NULL) {
            if (current->data > nextnode->data) {
                temp = current->data;
                current->data = nextnode->data;
                nextnode->data = temp;
            }
            nextnode = nextnode->next;
        }
        current = current->next;
    }
}
```

current->data = nextnode->data;
 nextnode->data = temp; 22, 20
 }
 nextnode = nextnode->next; 21
 }
 woren = current->next; 20, 21
 }
 void reverseList (struct node** headRef) {
 struct node * prev, * woren, * nextnode;
 prev = NULL;
 if (*headRef == NULL) {
 *headRef = woren; 21
 woren->next = prev; 20
 woren = nextnode; 21
 if (head->data == "bob") 21
 *headRef = prev; 20
 }
 void concatenateLists (struct node** list1, struct node** list2);
 if (*list1 == NULL) {
 *list1 = list2; 21
 if (list2->next != NULL) 21
 struct node * temp = list1->next; 21
 while (temp->next != NULL) {
 temp = temp->next; 21
 temp->next = list2; 21
 list2 = temp->next; 21
 }
 int main() {
 }

struct node * list 1 = NULL;

struct node * list 2 = NULL;

int choice;
int data;

printf("1. Insert info. list 1\n");

printf("2. Insert info. list 2\n");

printf("3. Sort list 1\n");

printf("4. Reverse list 1\n");

printf("5. Concatenate list 1 & 2\n");

printf("6. Print list 1\n");

printf("0. Exit\n");

while(1){

 printf("Enter choice:");

 scanf("%d", &choice);

 switch(choice){

 case 1:

 printf("Enter data into list 1\n");

 scanf("%d", &data);

 insert at beginning(&list 1, data);

 break;

 case 2:

 printf("Enter data to insert into list 2\n");

 scanf("%d", &data);

 insert at begin(&list 2, data);

 break;

 case 3:

 sortList(&list 1);

 printf("list 1 sorted\n");

 break;

 case 4:

 reverseList(&list 1);

 printf("list 1 reversed\n");

 break;

case 5: Insert & Print concatenated lists

concatenateList(& list1, list2);
printf("List is concatenated \n").
break;

case 6: Print list 1

printf("List \n"),

printList(list1),

if(!list1) printf("List 2 \n"),

printList(list2),

break; // break and go to switch

case 7: Print list 1

printf("List \n"),

if(!list1) break;

printList(list1),

break; // break and go to switch

option 0: Exit program

if(0) exit(0);

else printf("List \n"),

① Insert into list 1 in front

② Insert into list 2

③ Sort list 1

④ Reverse list 2 in front

⑤ concatenate list 1 & list 2

⑥ print list 1 & list 2

⑦ Exit.

Enter your choice : 1

Enter data to insert into list 1 : 23

Enter your choice : 2

Enter data to insert into list 2 : 34

Enter your choice : 1

Enter data to insert into list 1 : 2

Enter data to insert into list 2 : 45

Enter your choice: 6

List 1: 2 23

List 2: 45 34

Enter your choice: 4

List 1 reversed:

Enter your choice: 1

List 1: 23 2

List 2: 45 34

Enter your choice: 5

List concatenated

Enter your choice: 6

List 1: 23 2 45 34

List 2: 45 34

Enter your choice: 0

Exit

Q-2 Write a C program to implement Stack using linked list. (not the library function)

I/P #include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node * next;

};

struct node * top = NULL;

void push(int x) {

struct node * newnode;

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

newnode->data = x;

newnode->next = top;

top = newnode;

```
void pop() {
```

```
    struct node *temp;
```

```
    temp = top;
```

```
    if (top == 20) {
```

```
        printf("List is empty");
```

```
}
```

```
else {
```

```
    top = top->next;
```

```
    free(temp);
```

```
}
```

```
void main() {
```

```
    int choice = -1, i = 0;
```

```
    while (choice != 4) {
```

```
        printf("Enter choice:");
```

```
        scanf("%d", &choice);
```

```
        switch (choice) {
```

```
            case 1: push(i);
```

```
            case 2: printf("Add at top");
```

```
            case 3: printf("Add at bottom");
```

```
            case 4: push(i);
```

```
            break;
```

```
            case 5: printf("pop from top");
```

```
            pop();
```

```
            break;
```

```
            case 6: printf("pop from bottom");
```

```
            pop();
```

```
            break;
```

```
            case 7: printf("Display");
```

```
            display();
```

```
            break;
```

```
            case 8: exit(0);
```

```
            break;
```

```
void display() {  
    struct node *temp; // pointer to node  
    temp = top; // top points to first node  
    while (temp != NULL) {  
        printf("%d", temp->data); // print data  
        temp = temp->next; // move to next node  
    }  
}
```

Q1P Enter choice: 1

Add at top: 42

Enter choice: 2

Add at top: 43

Enter choice: 3

43 42

Enter choice: 2

Pop from top

Enter choice: 3

42

Enter choice: 4

exit.

Q2 Queue Implementation using Linked List

```
#include <cs50.h>
```

```
#include <stdlib.h>
```

```
struct node {
```

```
    int data;
```

```
    struct node *next;
```

```
};
```

```
struct node *front = NULL, *rear = NULL;
```

```
void enqueue(int x) {
```

```
    new_node = (struct node *) malloc(sizeof(struct node));
```

newnode->data = 2; newnode->next = NULL;

if (front == 0 & rear == 0) {

front = rear = newnode; }

} else { front = rear = newnode; }

else { front = rear = newnode; }

rear->next = newnode; }

rear = newnode; }

} else { front = rear = newnode; }

void display () {

struct node * temp;

if (front == 0 & rear == 0) {

printf("Queue is empty"); }

else { front = rear = newnode; }

temp = front; }

while (temp != NULL) {

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

temp = temp->next; }

void deQueue() {

struct node * temp;

temp = front; }

if (front == 0 & rear == 0) {

printf("Empty queue"); }

else { front = front->next; }

front = front->next; }

free(temp); }

rear = front; }

else { front = front->next; }

~~O/P~~ faster choice: 1

Add element i/2

Enter choice: 1

Add element: 13

After Chivis 3

~~Add element 65~~

Enter employee 3

DISPLAY

12 13 15

Enter employee 2

DELETE ELEMENT

12 13 15

Enter employee 3

DISPLAY

13 15

Enter employee 4

DELETE ELEMENT

13 15

Enter employee 5

DISPLAY

13 15

Enter employee 6

DISPLAY

13 15

Enter employee 7

DISPLAY

13 15

Enter employee 8

DISPLAY

13 15

Enter employee 9

DISPLAY

13 15

Enter employee 10

DISPLAY

13 15

Enter employee 11

DISPLAY

13 15

Enter employee 12

- Q-1) write a program to implement doubly linked list with primitive operations
- (a) Create a doubly linked list.
 - (b) Insert a new node to the left of node.
 - (c) Delete the node based on a spell (C name).

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
    struct node *prev;
};

struct node *head = NULL, *newnode;
struct node *temp;

void create(int data);
void insert_at_left(int data);
void delete_at_val();
void display();

void create(int data) {
    newnode = (struct node *) malloc(sizeof(struct node));
    newnode->data = data;
    newnode->next = NULL;
    newnode->prev = NULL;
    if (head == NULL) {
        head = temp = newnode;
    } else {
        temp->next = newnode;
        newnode->prev = temp;
        temp = newnode;
    }
}

void insert_at_left(int data) {
}
```

int p=0, qnt_Pes, struct node * current = head;

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

newnode->next = NULL;

newnode->prev = NULL;

printf("Enter position: ");

scanf("%d", &Pes);

while (p < Pes - 1)

current = current->next;

p++

}

newnode->next = current->next;

(current->next)->prev = newnode;

current->next = newnode;

newnode->prev = current;

}

void delete_at_value()

struct node * nextnode;

int value; struct node * current = head;

printf("Enter value to be deleted: ");

scanf("%d", &value);

if (current->next == NULL) {

nextnode = current->next;

if (nextnode->data == value) {

current = (current->next);

continue;

}

else {

wornt->next = nextnode->next;

(nextnode->next)->prev = current;

free(nextnode);

break;

}

if (wornt->next == NULL) to break loop

Q1P

Enter choice : 1 for insertion & 2 for deletion : 1-0

Enter data for node : 5

Enter choice : 2 for insertion & 1 for deletion : 1-1

Enter data : 21

Enter position : 0 for left & 1 for right : 0

Enter choice : 1 for insertion & 2 for deletion : 1-0

Enter data for Node : 6

Enter choice : 3

Enter value to be deleted : 5

Enter choice : 4 for left & 6 for right : 6

Display

21 6

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

choice = 1 for insertion & 2 for deletion : 1

Q-1 Delete the middle node of a linked list \rightarrow LeetCode
https://leetcode.com/problems/delete-middle-node-of-a-linked-list-with-single-pass/

```

I/P struct ListNode* deleteMiddle (struct ListNode* head) {
    int n=0; // count number of nodes in list
    struct ListNode *current = head;
    struct ListNode *nextnode;
    while (current != NULL) {
        n++;
        current = current->next;
    }
    if (n==1) {
        free(head);
        head = NULL;
        return head;
    }
    int mid = n/2; // mid point of list
    if (mid == 0) {
        head = head->next;
        free(head);
        return head;
    }
    else if (mid == 1) {
        prevnode = current;
        current = current->next;
        prevnode->next = current->next;
        free(current);
        return head;
    }
    else {
        current = current->next;
        nextnode = current->next;
        p++;
        current->next = nextnode->next;
        free(nextnode);
        return head;
    }
}

```

Q-2 Odd Even Linked List \rightarrow Left Codes

IP struct listnode * oddEvenGet(struct listnode * head);
if (!head) head->next = (1 + head->next->next); } }

return head;

{ }

return head;

struct listnode * oddHead = head, * evenHead;

partipating * head->next; partipating * listnode * head;

struct listnode * oddHead = head, * evenHead;
while (evenHead < even->next) { }

odd->next = even->next;

odd = odd->next;

even = even->next;

even = even->next;

{ }

return oddHead;

return oddHead;

{ }

return oddHead;

Q-3 Write C program to implement binary tree

(a) to construct Binary Search Tree

(b) Traverse the tree using Inorder, Post-order, Preorder.

(c) Display the elements present in

IP

#include<stdio.h> // for standard input output

#include<stdlib.h>

struct TreeNode {

int val;

struct TreeNode * left;

struct TreeNode * right;

{ }

struct TreeNode * createNode(int val) { }

Struct TreeNode * newnode = (Struct TreeNode *) malloc
 size of (Struct TreeNode));

newnode->val = val;

{ if (root->left == NULL) { } else { }

 newnode->left = NULL;

 newnode->right = NULL;

 return newnode;

BroadFirstSearch (Struct TreeNode * root);

Struct TreeNode * insert (Struct TreeNode * root, int val);

if (root == NULL) { } else { }

 return createNode (val);

 { newnode->left = insert (root->left, val); }

 { newnode->right = insert (root->right, val); }

 { if (root->left == NULL) { } else { }

 { if (val > root->val) { }

 { if (root->right == NULL) { }

 { root->right = insert (root->right, val); }

 { if (root->right == NULL) { }

 { if (val > root->val) { }

 { if (root->right == NULL) { }

 { root->right = insert (root->right, val); }

 { if (root->right == NULL) { }

preorder traversal (root \rightarrow left);
}

void postorder traversal (struct Tree Node *root) {
if (root == NULL) {
return;
}
postorder traversal (root \rightarrow left);
postorder traversal (root \rightarrow right);
printf (" %d ", root->val);
}

void displayTree (struct Tree Node *root) {
printf (" Elements in tree: ");
inorder traversal (root);
printf ("\n");
}

int main () {

struct Tree Node *root = NULL;

int choice = -1, val;

printf (" 1. Insert element \n ");

printf (" 2. Display tree (inorder) \n ");

printf (" 3. Display tree (preorder) \n ");

printf (" 4. Display tree (postorder) \n ");

printf (" 5. Exit \n ");

while (choice != 5) {

printf (" Enter your choice: ");

scanf ("%d", &choice);

switch (choice) {

case 1:

printf (" Enter value to insert: ");

scanf ("%d", &val);

root = insert (root, val);

break;

case 2:

display tree (root);

break;

choice case 1: current selection 1st.

print ("elements in tree (preorder):");

preorder traversal (root);

print ("in");

choice case 2: break; return;

choice case 3: current selection 2nd.

print ("elements in tree (postorder):");

postorder traversal (root);

choice case 4: break; return;

choice case 5: insertion "1" tree;

insert (return 0); insert & return 1;

choice case 6: insertion "2" tree;

insert (return 1); insert & return 2;

choice case 7: insertion "3" tree;

enter your choice : 1 * insertion tree;

enter value to insert : 33 (= 33 0 0 0) tree;

enter your choice : 1 (= 1 0 0 0) tree;

enter value to insert : 44 (= 44 0 0 0) tree;

enter your choice : 2 (= 2 0 0 0) tree;

elements in tree : 33 44 up & "1" tree;

enter your choice : 3 (= 3 0 0 0) tree;

elements in tree (preorder) : 33 44 99 66 55

enter your choice : 5 (= 5 0 0 0) tree;

exit : 0 (= 0 0 0 0) tree;

* (= 0 0 0 0) tree;

choice case 8: deletion "1" tree;

choice case 9: deletion "2" tree;

choice case 10: deletion "3" tree;

choice case 11: deletion "4" tree;

Q-1 Implement DFS traversal.

```
#include <stdio.h>
#include <iostream.h>
#define MAX_VERTICES 20
int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES]; int n;
void dfs(int start) {
    int i;
    visited[start] = 1;
    printf("visited %d\n", start);
    for (i = 0; i < n; i++) {
        if (graph[start][i] && !visited[i]) {
            dfs(i);
        }
    }
}
```

```
int main() {
    int n, i, start;
    printf("Enter the number of vertices : ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix : ");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf(" %d", &graph[i][j]);
        }
    }
}
```

```
printf("Enter the starting vertex for DFS : ");
scanf(" %d", &start);
for (i = 0; i < n; i++) {
    visited[i] = 0;
}
dfs(start);
```

return 0;

}

Q/P Enter the number of vertices: 4
Enter the adjacency matrix of graph:

0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0

0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0

1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Entering time & starting vertex) for DFS: 0

visited 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Visited 3 a. {0,1,2,3,Browsing} 98

2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Q-2 Implement BFS.

I/P #include <stdio.h>

void bfs(int arr[10], int n, int v) { /* arr = 0 1 2 3 4 5 6 7 8 9 */

int f, r, q[10], v; /* f=front, r=rear */

q[0] = v; /* initial value */ f = 0; r = 0;

printf("The nodes visited from 0(0)", v);

for (f = 0; f < r; f++) { /* visit the front node */ } /* loop */

f = f + 1; /* increment front pointer */ r = r + 1; /* increment rear pointer */

q[f] = v; /* add new node */

if (arr[v] == 1) { /* if node is visited */ }

printf("0(0)", v);

while (f <= r) {

/* visit the front node */ v = q[f++]; /* increment front pointer */

for (v = 0; v < n; v++) { /* loop */ }

if ((arr[v] == 1) && (sv == 0)) {

printf("0(0)", v);

sv = 1;

q[f] = v; /* add new node */

```
pointf ("n");  
void main () {  
    int n, a[10][10], source, i, j, sign = 1;  
    pointf ("nEnter no. of nodes:");  
    scanf ("%d", &n);  
    pointf ("nEnter the adjacency matrix: \n");  
    for (i = 0; i < n; i++) {  
        for (j = 0; j < n; j++) {  
            scanf ("%d", &a[i][j]);  
        }  
    }  
    for (source = 0; source < n; source++) {  
        if (sign) {  
            for (i = 0; i < n; i++) {  
                if (a[source][i] == 1) {  
                    printf ("%d ", i);  
                }  
            }  
            sign = 0;  
        } else {  
            for (i = 0; i < n; i++) {  
                if (a[source][i] == 1) {  
                    printf ("%d ", i);  
                }  
            }  
            sign = 1;  
        }  
    }  
}
```

O/P Enter no. of nodes: 3

Enter the adjacency matrix:

1 1 1

1 0 0

0 1 0

The node visited from 0 0 1 2

The node visited from 1 1 0 2

The node visited from 2 0 2 1 0

Q-3 Delete a node from BST as linked code

```
struct TreeNode * deleteNode (struct TreeNode * root, int key) {  
    if (root == NULL)  
        return root;  
    if (key < root->val)  
        root->left = deleteNode (root->left, key);  
    if (key > root->val)  
        root->right = deleteNode (root->right, key);  
    if (root->left == NULL && root->right == NULL)  
        free (root);  
    else if (root->left == NULL)  
        root = root->right;  
    else if (root->right == NULL)  
        root = root->left;  
    else {  
        struct TreeNode * temp = root->right;  
        while (temp->left != NULL)  
            temp = temp->left;  
        root->val = temp->val;  
        root->right = deleteNode (root->right, temp->val);  
    }  
    return root;  
}
```

```
else if (key > root->val) { // left child
    root->right = deleteNode (root->right, key);
}
else {
    if (root->left == NULL) {
        struct TreeNode* temp = root->right;
        free (root);
        return temp;
    }
    else if (root->right == NULL) {
        struct TreeNode* temp = root->left;
        free (root);
        return temp;
    }
    else {
        struct TreeNode* temp = minValueNode (root->right);
        root->val = temp->val;
        root->right = deleteNode (root->right, temp->val);
    }
}
return root;
```

```
void inorder (struct TreeNode* root) {
    if (root != NULL) {
        inorder (root->left);
        printf ("%d ", root->val);
        inorder (root->right);
    }
}
```

```
struct TreeNode* minValueNode (struct TreeNode* node) {
    struct TreeNode* current = node;
    while (current->left != NULL) {
        current = current->left;
    }
    return current;
}
```

Q4 Bottom left free (value -> root code).

Q5 bottom left free (value -> root code).

```
int findBottomLeftValue (struct TreeNode* root) {
    struct TreeNode* queue[100];
    int front = 0, rear = 0;
    queue[rear++] = root;
    int debtMostValue = root->val;
    while (front < rear) {
        int levelSize = rear - front;
        for (int i = 0; i < levelSize; i++) {
            struct TreeNode* current = queue[front++];
            if (i == 0) {
                debtMostValue = current->val;
            }
            else if (current->left != NULL) {
                queue[rear++] = current->left;
            }
            else if (current->right != NULL) {
                queue[rear++] = current->right;
            }
        }
        return debtMostValue;
    }
}
```

8am
4/3/24