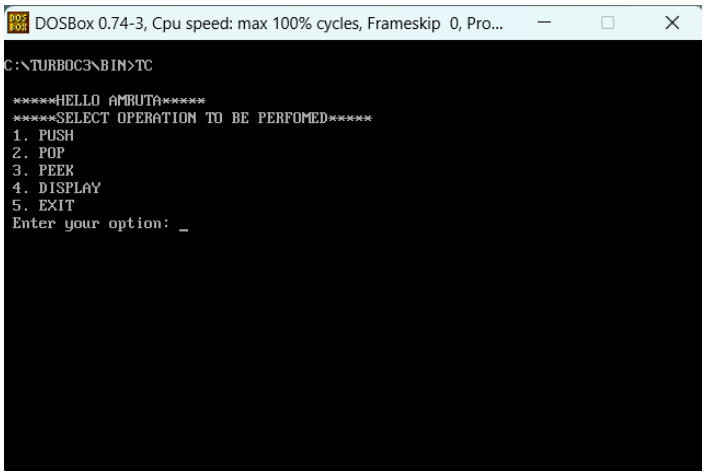
To implement a menu driven program to perform operations on the stack ADT using array.

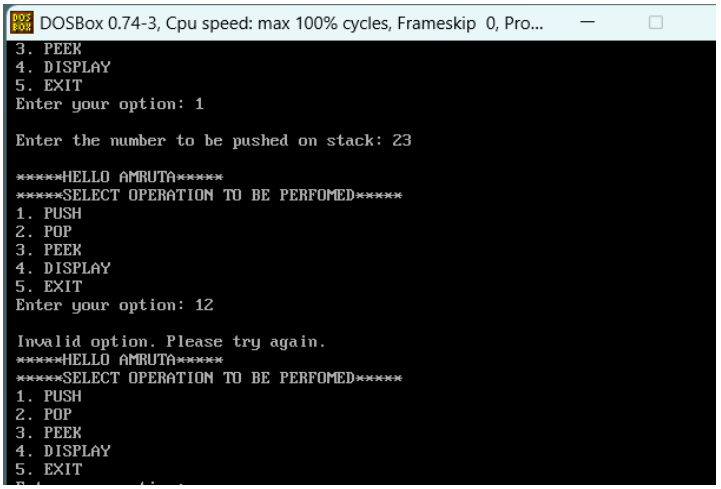
```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3
int st[MAX], top = -1;
void push(int st[], int val);
int pop(int st[]);
int peek(int st[]);
void display(int st[]);
int main() {
int val, option;
do {
printf("\n *****HELLO AMRUTA*****");
printf("\n *****SELECT OPERATION TO BE PERFOMED*****");
printf("\n 1. PUSH");
printf("\n 2. POP");
printf("\n 3. PEEK");
printf("\n 4. DISPLAY");
printf("\n 5. EXIT");
printf("\n Enter your option: ");
if (scanf("%d", &option) != 1) {
while (getchar() != '\n');
option = 0; // Set option to a value that does not match any valid case
printf("\n Invalid input. Please enter an integer.");
continue;
}
switch(option) {
case 1:
printf("\n Enter the number to be pushed on stack: ");
if (scanf("%d", &val) != 1) {
// Clear the input buffer in case of invalid input
while (getchar() != '\n');
printf("\n Invalid input. Please enter an integer.");
break;
}
push(st, val);
break;
case 2:
val = pop(st);
if(val != -1) {
printf("\n The value deleted from stack is: %d", val);
}
break;
case 3:
val = peek(st);
if(val != -1) {
```

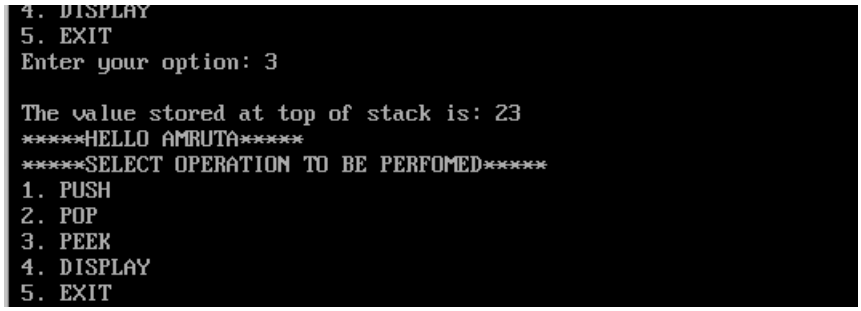
```
printf("\n The value stored at top of stack is: %d", val);
}
break;
case 4:
display(st);
break;
case 5:
printf("\n Exiting program.");
break;
default:
printf("\n Invalid option. Please try again.");
break;
}
} while(option != 5);
return 0;
}
void push(int st[], int val) {
if(top == MAX - 1) {
printf("\n STACK OVERFLOW");
} else {
top++;
st[top] = val;
}
}
int pop(int st[]) {
int val;
if(top == -1) {
printf("\n STACK UNDERFLOW");
return -1;
} else {
val = st[top];
top--;
return val;
}
}
void display(int st[]) {
int i;
if(top == -1) {
printf("\n STACK IS EMPTY");
} else {
printf("\n Stack elements are:");
for(i = top; i >= 0; i--) {
printf("\n %d", st[i]);
}
printf("\n"); // Added for formatting purposes
}
}
```

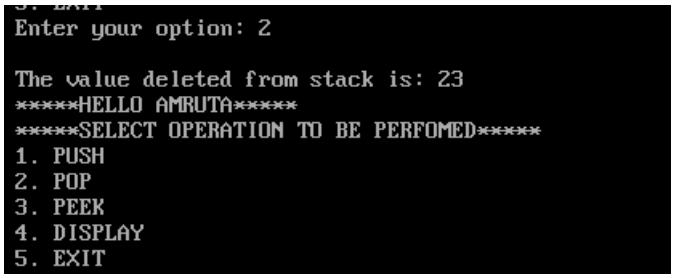
```
int peek(int st[]) {
if(top == -1) {
printf("\n STACK IS EMPTY");
return -1;
} else {
return st[top];
}
}
```

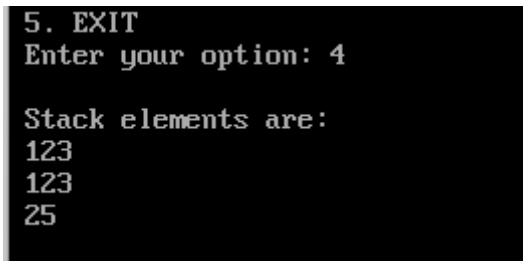
OUTPUT:











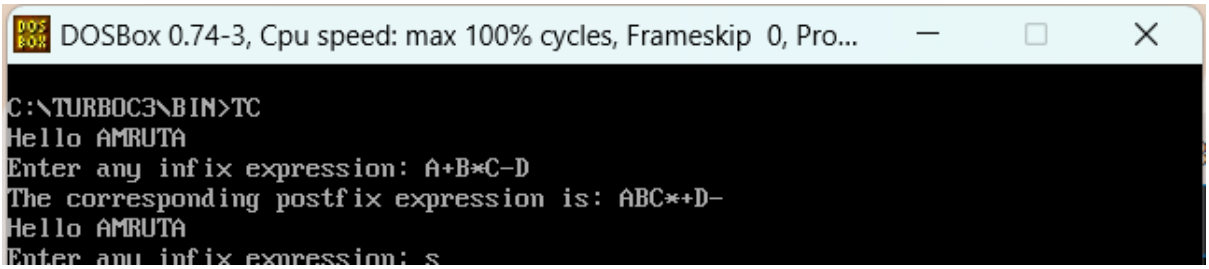
2. To write a program for infix to postfix conversion using stack.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
char st[MAX];
int top = -1;
void push(char st[], char);
char pop(char st[]);
void infixToPostfix(const char source[], char target[]);
int getPriority(char);
int main()
{
char infix[MAX], postfix[MAX];
printf("Hello AMRUTA\n");
printf("Enter any infix expression: ");
if (fgets(infix, sizeof(infix), stdin) != NULL) {
// Remove newline character if present
size_t length = strlen(infix);
if (infix[length - 1] == '\n') {
infix[length - 1] = '\0';
}
}
strcpy(postfix, "");
infixToPostfix(infix, postfix);
printf("The corresponding postfix expression is: ");
puts(postfix);
return 0;
}
void infixToPostfix(const char source[], char target[]) {
int i = 0, j = 0;
char temp;
strcpy(target, "");
while (source[i] != '\0') {
if (source[i] == '(') {
push(st, source[i]);
i++;
} else if (source[i] == ')') {
while ((top != -1) && (st[top] != '(')) {
target[j++] = pop(st);
```

```
}
if (top == -1) {
printf("INCORRECT EXPRESSION\n");
exit(1);
}
pop(st); // Remove left parenthesis
i++;
} else if (isdigit(source[i]) || isalpha(source[i])) {
target[j++] = source[i];
i++;
} else if (strchr("+-*/%", source[i])) {
while ((top != -1) && (st[top] != '(') && (getPriority(st[top]) >=
getPriority(source[i]))) {
target[j++] = pop(st);
}
push(st, source[i]);
i++;
} else {
printf("INCORRECT ELEMENT IN EXPRESSION\n");
exit(1);
}
}
while (top != -1) {
target[j++] = pop(st);
}
target[j] = '\0';
}
int getPriority(char op) {
switch (op) {
case '*':
case '/':
case '%':
return 2;
case '+':
case '-':
return 1;
default:
return 0;
}
}
void push(char st[], char val) {
```

```
if (top == MAX - 1) {
printf("STACK OVERFLOW\n");
} else {
st[++top] = val;
}
}
char pop(char st[]) {
if (top == -1) {
printf("STACK UNDERFLOW\n");
return ' ';
} else {
return st[top--];
}
}
```

OUTPUT:



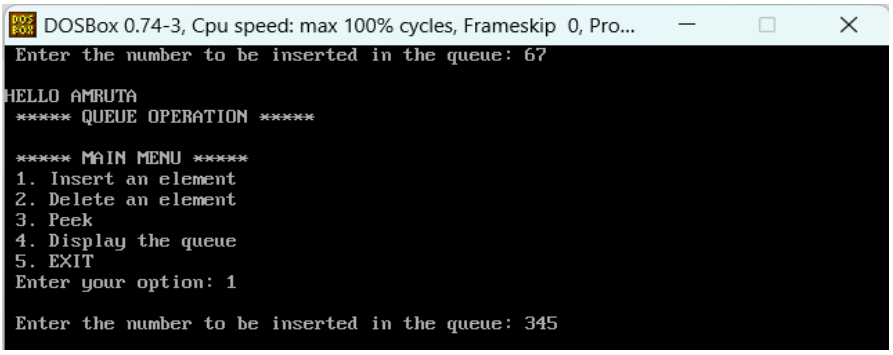
4: To implement Linear Queue ADT using array.

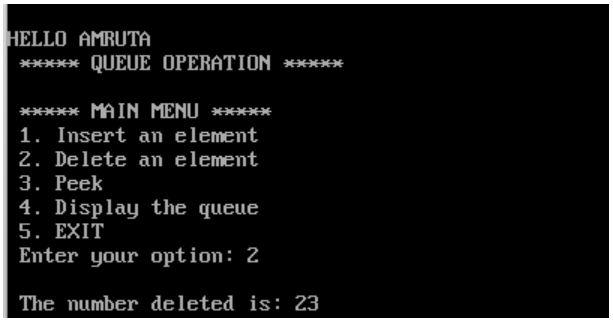
```
#include <stdio.h>
#define MAX 10
int queue[MAX];
int front = -1, rear = -1;
void insert(void);
int delete_element(void);
int peek(void);
void display(void);
int main() {
int option, val;
do {
printf("\nHELLO AMRUTA");
printf("\n ***** QUEUE OPERATION *****");
printf("\n\n ***** MAIN MENU *****");
printf("\n 1. Insert an element");
printf("\n 2. Delete an element");
printf("\n 3. Peek");
printf("\n 4. Display the queue");
printf("\n 5. EXIT");
printf("\n Enter your option: ");
scanf("%d", &option);
switch(option) {
case 1:
insert();
break;
case 2:
val = delete_element();
if (val != -1) {
printf("\n The number deleted is: %d", val);
}
break;
case 3:
val = peek();
if (val != -1) {
printf("\n The first value in queue is: %d", val);
}
break;
case 4:
display();
```

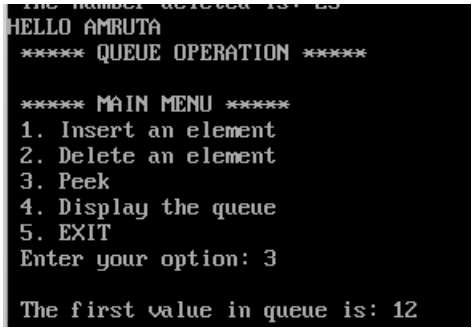
```
break;
}
} while(option != 5);
return 0;
}
void insert() {
int num;
   printf("\n Enter the number to be inserted in the queue: ");
   scanf("%d", &num);
   if(rear == MAX - 1) {
      printf("\n OVERFLOW");
   } else if(front == -1 && rear == -1) {
      front = rear = 0;
   } else {
      rear++;
   }
   queue[rear] = num;
}
int delete_element() {
   int val;
   if(front == -1 || front > rear) {
      printf("\n UNDERFLOW");
      return -1;
   } else {
      val = queue[front];
      front++;
      if(front > rear) {
         front = rear = -1; // Reset the queue after the last element is removed
      }
      return val;
   }
}
int peek() {
   if(front == -1 || front > rear) {
      printf("\n QUEUE IS EMPTY");
      return -1;
   } else {
      return queue[front];
   }
}
```

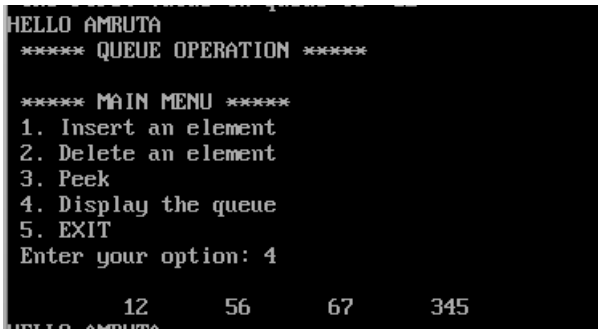
```
void display() {
   int i;
   printf("\n");
   if(front == -1 || front > rear) {
      printf("\n QUEUE IS EMPTY");
   } else {
      for(i = front; i <= rear; i++) {
         printf("\t %d", queue[i]);
      }
   }
}
```

OUTPUT:









5. To implement Circular Queue ADT using array.

```
#include <stdio.h>
#define MAX 10
int queue[MAX];
int front = -1, rear = -1;
void insert(void);
int delete_element(void);
int peek(void);
void display(void);
int main() {
int option, val;
do
{
printf("\n\n ***** HELLO AMRUTA *****");
printf("\n ***** MAIN MENU *****");
printf("\n 1. Insert an element");
printf("\n 2. Delete an element");
printf("\n 3. Peek");
printf("\n 4. Display the queue");
printf("\n 5. EXIT");
printf("\n Enter your option: ");
scanf("%d", &option);
switch(option)
{
case 1:
insert();
break;
case 2:
val = delete_element();
if(val != -1) {
printf("\n The number deleted is: %d", val);
}
break;
case 3:
val = peek();
if(val != -1) {
printf("\n The first value in queue is: %d", val);
}
break;
case 4:
```

```
display();
break;
}
} while(option != 5);
return 0;
}
void insert()
{
int num;
printf("\n Enter the number to be inserted in the queue: ");
scanf("%d", &num);
if((front == 0 && rear == MAX - 1) || (front == rear + 1)) {
printf("\n OVERFLOW");
} else if(front == -1 && rear == -1) {
front = rear = 0;
queue[rear] = num;
} else if(rear == MAX - 1 && front != 0) {
rear = 0;
queue[rear] = num;
} else {
rear++;
queue[rear] = num;
}
}
int delete_element()
{
int val;
if(front == -1 && rear == -1) {
printf("\n UNDERFLOW");
return -1;
}
val = queue[front];
if(front == rear) {
front = rear = -1;
} else {
if(front == MAX - 1) {
front = 0;
} else {
front++;
}
}
```

```
return val;
}
int peek()
{
if(front == -1 && rear == -1) {
printf("\n QUEUE IS EMPTY");
return -1;
} else {
return queue[front];
}
}
void display()
{
int i;
printf("\n");
if(front == -1 && rear == -1) {
printf("\n QUEUE IS EMPTY");
} else {
if(front <= rear) {
for(i = front; i <= rear; i++) {
printf("\t %d", queue[i]);
}
} else {
for(i = front; i < MAX; i++) {
printf("\t %d", queue[i]);
}
for(i = 0; i <= rear; i++) {
printf("\t %d", queue[i]);
}
}
}
}
```

OUTPUT:

```
***** HELLO AMRUTA *****
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option: 2

The number deleted is: 56
```

```
***** HELLO AMRUTA *****
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option: 3

The first value in queue is: 56
```

```
***** HELLO AMRUTA *****
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option: 4

       56      67      78      89      90      69
```

```
***** HELLO AMRUTA *****
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option: 2

The number deleted is: 56
```

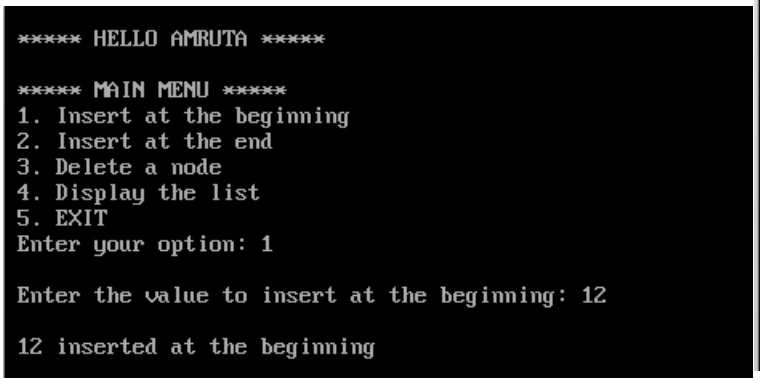
6. To implement Singly linked list ADT.

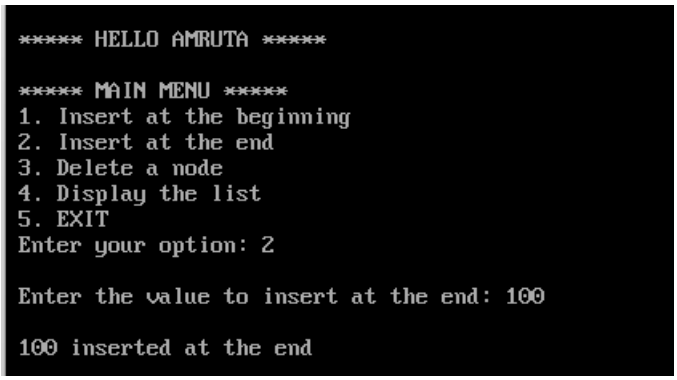
```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node* next;
};
struct Node* head = NULL;
void insertAtBeginning(int value);
void insertAtEnd(int value);
void deleteNode(int value);
void displayList();
int main() {
int option, value;
do {
printf("\n\n ***** HELLO AMRUTA *****");
printf("\n\n ***** MAIN MENU *****");
printf("\n 1. Insert at the beginning");
printf("\n 2. Insert at the end");
printf("\n 3. Delete a node");
printf("\n 4. Display the list");
printf("\n 5. EXIT");
printf("\n Enter your option: ");
scanf("%d", &option);
switch(option) {
case 1:
printf("\n Enter the value to insert at the beginning: ");
scanf("%d", &value);
insertAtBeginning(value);
break;
case 2:
printf("\n Enter the value to insert at the end: ");
scanf("%d", &value);
insertAtEnd(value);
break;
case 3:
printf("\n Enter the value to delete: ");
scanf("%d", &value);
deleteNode(value);
break;
```

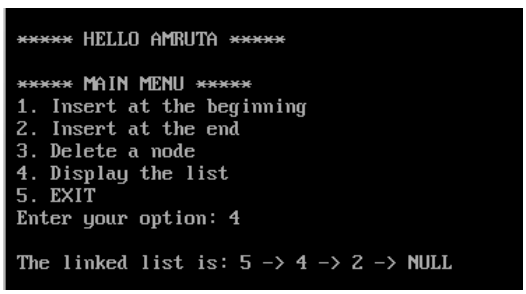
```
case 4:
displayList();
break;
}
} while(option != 5);
return 0;
}
void insertAtBeginning(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = head;
head = newNode;
printf("\n %d inserted at the beginning", value);
}
void insertAtEnd(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;
if (head == NULL) {
head = newNode;
} else {
struct Node* temp = head;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = newNode;
}
printf("\n %d inserted at the end", value);
}
void deleteNode(int value) {
struct Node* temp = head;
struct Node* prev = NULL;
if (temp != NULL && temp->data == value) {
head = temp->next;
free(temp);
printf("\n Node with value %d deleted", value);
return;
}
while (temp != NULL && temp->data != value) {
prev = temp;
temp = temp->next;
```

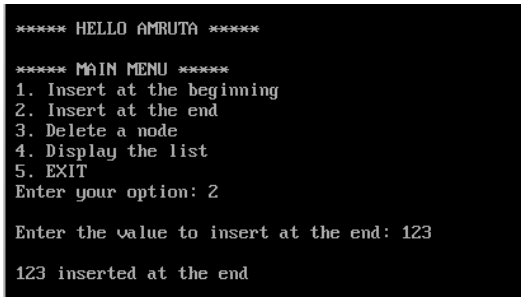
```
}
if (temp == NULL) {
printf("\n Node with value %d not found", value);
return;
}
prev->next = temp->next;
free(temp);
printf("\n Node with value %d deleted", value);
}
void displayList() {
struct Node* temp = head;
if (head == NULL) {
printf("\n The list is empty");
return;
}
printf("\n The linked list is: ");
while (temp != NULL) {
printf("%d -> ", temp->data);
temp = temp->next;
}
printf("NULL");
}
```

OUTPUT:









7. To implement Circular linked list ADT.

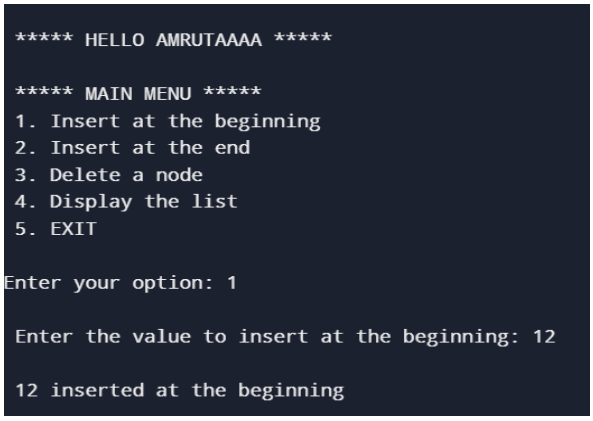
```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
int data;
struct Node* next;
};
struct Node* head = NULL;
void insertAtBeginning(int value);
void insertAtEnd(int value);
void deleteNode(int value);
void displayList();
int main() {
int option, value;
do
{
printf("\n\n ***** HELLO SHRAVANI *****");
printf("\n\n ***** MAIN MENU *****");
printf("\n 1. Insert at the beginning");
printf("\n 2. Insert at the end");
printf("\n 3. Delete a node");
printf("\n 4. Display the list");
printf("\n 5. EXIT");
printf("\n Enter your option: ");
scanf("%d", &option);
switch(option)
{
case 1:
printf("\n Enter the value to insert at the beginning: ");
scanf("%d", &value);
insertAtBeginning(value);
break;
case 2:
printf("\n Enter the value to insert at the end: ");
scanf("%d", &value);
insertAtEnd(value);
break;
case 3:
printf("\n Enter the value to delete: ");
```

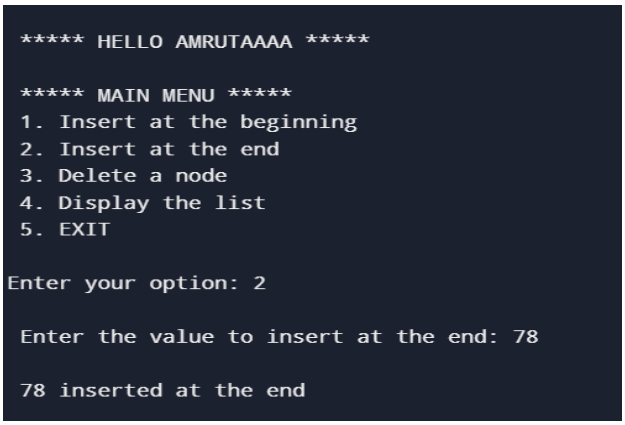
```
scanf("%d", &value);
deleteNode(value);
break;
case 4:
displayList();
break;
}
} while(option != 5);
return 0;
}
void insertAtBeginning(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
if (head == NULL) {
newNode->next = newNode; // Point to itself when there's only one node
head = newNode;
} else {
struct Node* temp = head;
while (temp->next != head) {
temp = temp->next;
}
newNode->next = head;
temp->next = newNode;
head = newNode;
}
printf("\n %d inserted at the beginning", value);
}
void insertAtEnd(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
if (head == NULL) {
newNode->next = newNode; // First node points to itself
head = newNode;
} else {
struct Node* temp = head;
while (temp->next != head) {
temp = temp->next;
}
temp->next = newNode;
newNode->next = head;
}
```

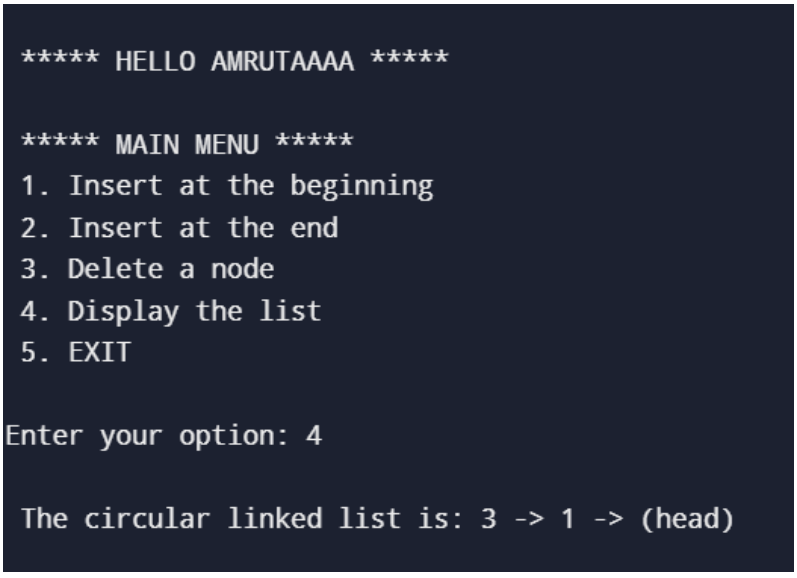
```
printf("\n %d inserted at the end", value);
}
void deleteNode(int value) {
if (head == NULL) {
printf("\n List is empty");
return;
}
struct Node *temp = head, *prev = NULL;
if (temp->data == value && temp->next == head) {
free(temp);
head = NULL;
printf("\n Node with value %d deleted", value);
return;
}
if (temp->data == value) {
while (temp->next != head) {
temp = temp->next;
}
temp->next = head->next;
free(head);
head = temp->next;
printf("\n Node with value %d deleted", value);
return;
}
while (temp->next != head && temp->data != value) {
prev = temp;
temp = temp->next;
}
if (temp->data == value) {
prev->next = temp->next;
free(temp);
printf("\n Node with value %d deleted", value);
} else {
printf("\n Node with value %d not found", value);
}
}
void displayList() {
struct Node* temp = head;
if (head == NULL) {
printf("\n The list is empty");
return;
```

```
}
printf("\n The circular linked list is: ");
do {
printf("%d -> ", temp->data);
temp = temp->next;
} while (temp != head);
printf("(head)");
}
```

OUTPUT:







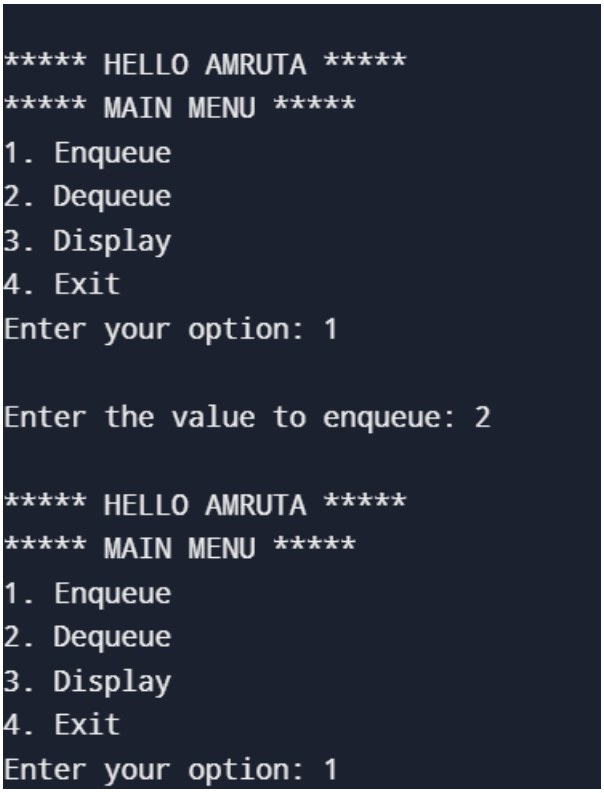
8. To implement Linear Queue ADT using linked list.

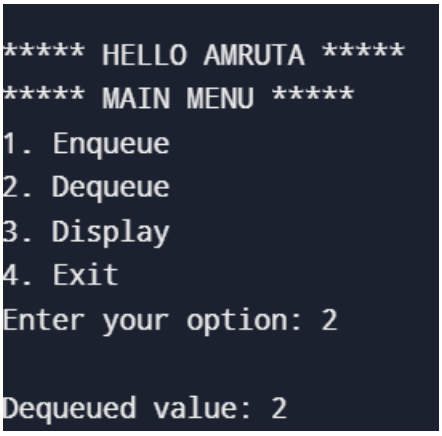
```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;
   struct Node* next;
};

struct Node* front = NULL;
struct Node* rear = NULL;

// Function to enqueue a value
void enqueue(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = NULL;
   if (rear == NULL) {
      front = rear = newNode;
   } else {
      rear->next = newNode;
      rear = newNode;
   }
}

// Function to dequeue a value
int dequeue() {
   if (front == NULL) {
      printf("\nQueue is empty\n");
      return -1;
   }
   int value = front->data;
   struct Node* temp = front;
   front = front->next;
   if (front == NULL) {
      rear = NULL;
   }
   free(temp);
   return value;
}

// Function to display the queue
void display() {
   struct Node* temp = front;
   if (front == NULL) {
      printf("\nQueue is empty\n");
```

```
        return;
    }
    printf("\nQueue: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int option, value;

    do {
        printf("\n***** HELLO AMRUTA *****");
        printf("\n***** MAIN MENU *****");
        printf("\n1. Enqueue");
        printf("\n2. Dequeue");
        printf("\n3. Display");
        printf("\n4. Exit");
        printf("\nEnter your option: ");
        scanf("%d", &option);

        switch (option) {
            case 1:
                printf("\nEnter the value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;

            case 2:
                value = dequeue();
                if (value != -1) {
                    printf("\nDequeued value: %d", value);
                }
                break;

            case 3:
                display();
                break;

            case 4:
                printf("\nExiting the program...\n");
                break;

            default:
                printf("\nInvalid option, please try again.\n");
```

```
        }
    } while (option != 4);

    return 0;
}
```

OUTPUT:





9. To implement Graph traversal techniques :

i)Depth first search (DFS)

```
#include <stdio.h>
#define MAX 5
void depth_first_search(int adj[][MAX], int visited[], int start) {
int stack[MAX];
int top = -1, i;
printf("%c-", start + 65);
visited[start] = 1;
stack[++top] = start;
while (top != -1) {
start = stack[top];
for (i = 0; i < MAX; i++) {
if (adj[start][i] && visited[i] == 0) {
stack[++top] = i;
printf("%c-", i + 65);
visited[i] = 1;
break;
}
}
if (i == MAX) {
top--;
}
}
}
int main() {
int adj[MAX][MAX];
int visited[MAX] = {0};
int i, j;
printf("\nHello Shravani");
printf("\nEnter the adjacency matrix:\n");
for (i = 0; i < MAX; i++) {
for (j = 0; j < MAX; j++) {
scanf("%d", &adj[i][j]);
}
}
printf("DFS Traversal: ");
depth_first_search(adj, visited, 0);
printf("\n");
return 0;
}
```

OUTPUT:

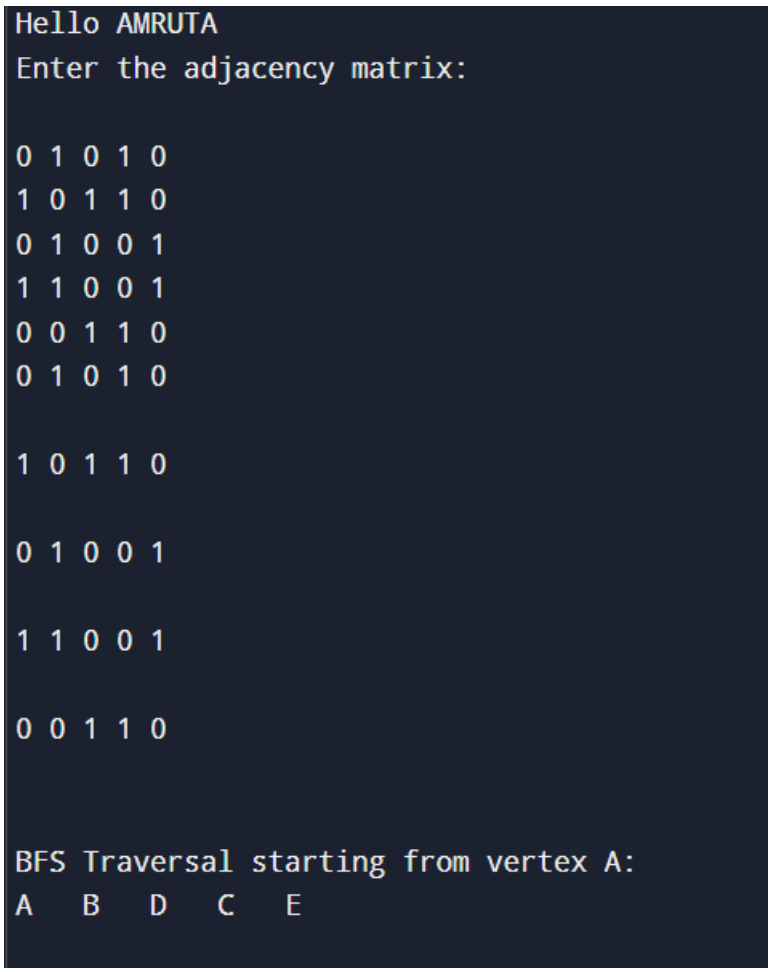
```
Hello AMRUTA
Enter the adjacency matrix:

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

1 0 1 1 0

0 1 0 0 1

1 1 0 0 1

0 0 1 1 0


DFS Traversal: A-B-C-E-D-
```

ii) Breadth first search (BFS)

```
#include <stdio.h>
#define MAX 5
void breadth_first_search(int adj[][MAX], int visited[], int start) {
int queue[MAX];
int rear = -1, front = -1, i;
// Enqueue the start vertex
queue[++rear] = start;
visited[start] = 1;
front++;
printf("BFS Traversal starting from vertex %c:\n", start + 65);
while (front <= rear) {
// Dequeue a vertex and print it
start = queue[front++];
printf("%c\t", start + 65);
// Enqueue all adjacent vertices of the dequeued vertex
for (i = 0; i < MAX; i++) {
if (adj[start][i] == 1 && visited[i] == 0) {
queue[++rear] = i;
```

```
visited[i] = 1;
}
}
}
printf("\n");
}
int main() {
int visited[MAX] = {0};
int adj[MAX][MAX];
int i, j;
printf("Hello Shravani\n");
printf("Enter the adjacency matrix:\n", MAX, MAX);
for (i = 0; i < MAX; i++) {
for (j = 0; j < MAX; j++) {
scanf("%d", &adj[i][j]);
}
}
breadth_first_search(adj, visited, 0); // Start BFS from vertex 0
return 0;
}
```

OUTPUT:



10. To implement Binary Search Tree ADT using linked list.

```
#include <stdio.h>
#include <stdlib.h>
// Define the structure for a node in the BST
typedef struct Node {
int data;
struct Node* left;
struct Node* right;
} Node;
// Function to create a new node
Node* create_node(int data) {
Node* new_node = (Node*)malloc(sizeof(Node));
new_node->data = data;
new_node->left = NULL;
new_node->right = NULL;
return new_node;
}
// Function to insert a new node into the BST
Node* insert(Node* root, int data) {
if (root == NULL) {
return create_node(data);
}
if (data < root->data) {
root->left = insert(root->left, data);
} else {
root->right = insert(root->right, data);
}
return root;
}
// Function to search for a value in the BST
Node* search(Node* root, int data) {
if (root == NULL || root->data == data) {
return root;
}
if (data < root->data) {
return search(root->left, data);
} else {
return search(root->right, data);
}
}
// Function to perform in-order traversal of the BST
void in_order_traversal(Node* root) {
if (root != NULL) {
in_order_traversal(root->left);
printf("%d ", root->data);
in_order_traversal(root->right);
```

```
}
}
// Main function
int main() {
Node* root = NULL;
int choice, value;
do
{
printf("\nHello AMRUTA");
printf("\n Select operation to be perfomed:");
printf("\n1. Insert a value");
printf("\n2. Search for a value");
printf("\n3. In-order traversal");
printf("\n4. Exit");
printf("\nEnter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter the value to insert: ");
scanf("%d", &value);
root = insert(root, value);
break;
case 2:
printf("Enter the value to search: ");
scanf("%d", &value);
Node* result = search(root, value);
if (result != NULL) {
printf("Value %d found in the BST.\n", value);
} else {
printf("Value %d not found in the BST.\n", value);
}
break;
case 3:
printf("In-order traversal of the BST: ");
in_order_traversal(root);
printf("\n");
break;
case 4:
printf("Exiting...\n");
break;
default:
printf("Invalid choice. Please enter a valid option.\n");
break;
}
} while (choice != 4);
return 0;
```

}

OUTPUT:

```
Hello AMRUTA
 Select operation to be perfomed:
1. Insert a value
2. Search for a value
3. In-order traversal
4. Exit
Enter your choice: 1
Enter the value to insert: 50

Hello AMRUTA
 Select operation to be perfomed:
1. Insert a value
2. Search for a value
3. In-order traversal
4. Exit
Enter your choice: 1
Enter the value to insert: 30

Hello AMRUTA
 Select operation to be perfomed:
1. Insert a value
2. Search for a value
3. In-order traversal
```

```
Hello AMRUTA
 Select operation to be perfomed:
1. Insert a value
2. Search for a value
3. In-order traversal
4. Exit
Enter your choice: 2
Enter the value to search: 30
Value 30 found in the BST.
```

```
Hello AMRUTA
 Select operation to be perfomed:
1. Insert a value
2. Search for a value
3. In-order traversal
4. Exit
Enter your choice: 3
In-order traversal of the BST: 20 30 50
```

11. To write a program based on the application of Binary Search Technique.

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

#define SIZE 10

int smallest(int arr[], int k, int n);
void selection_sort(int arr[], int n);

int main() {
   int arr[SIZE], num, i, n, beg, end, mid, found = 0;
   printf("Hello AMRUTA\n");
   printf("Enter the number of elements in the array (max %d): ", SIZE);
   scanf("%d", &n);

   if (n > SIZE) {
      printf("Number of elements exceeds the maximum allowed size of %d.\n", SIZE);
      return 1;
   }

   printf("Enter the elements:\n");
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   selection_sort(arr, n);

   printf("The sorted array is:\n");
   for (i = 0; i < n; i++) {
      printf("%d\t", arr[i]);
   }
   printf("\n");

   printf("Enter the number to be searched: ");
   scanf("%d", &num);

   beg = 0;
   end = n - 1;
   while (beg <= end) {
      mid = (beg + end) / 2;
      if (arr[mid] == num) {
         printf("%d is present in the array at position %d\n", num, mid + 1);
         found = 1;
         break;
      } else if (arr[mid] > num) {
```

```
            end = mid - 1;
        } else {
            beg = mid + 1;
        }
    }
if (!found) {
printf("%d does not exist in the array\n", num);
}
return 0;
}
int smallest(int arr[], int k, int n) {
int pos = k, small = arr[k], i;
for (i = k + 1; i < n; i++) {
if (arr[i] < small) {
small = arr[i];
pos = i;
}
}
return pos;
}
void selection_sort(int arr[], int n) {
int k, pos, temp;
for (k = 0; k < n; k++) {
pos = smallest(arr, k, n);
temp = arr[k];
arr[k] = arr[pos];
arr[pos] = temp;
}
}
```

OUTPUT:

```
Hello AMRUTA
Enter the number of elements in the array (max 10): 5
Enter the elements:
1 2 5 3 4
The sorted array is:
1   2   3   4   5
Enter the number to be searched:
```