DSA DA 1

Name – Tanay Doshi

Registration Number – 24BBS0104

question -1 menu driven code for stack

Pseudo code

```
1. Start
2. Initialize:
    • a[n] (stack array of size 5)
    • top = -1
3. Display menu options:
• 1: PUSH an element
• 2: POP an element
• 3: DISPLAY the stack
• 4: EXIT
4. If the user selects PUSH:
    • Check if top == n - 1 (stack is full, OVERFLOW).
        o If full, print: "Cannot add element to the stack: OVERFLOW".
    • If the stack is not full:
        o Increment top by 1.
        o Input the element and store it in a[top].
5. If the user selects POP:
    • Check if top == -1 (stack is empty, UNDERFLOW).
        o If empty, print: "No element to pop in the stack: UNDERFLOW".
    • If the stack is not empty:
        o Decrement top by 1.
6. If the user selects DISPLAY:
    • Print "Stack is:"
    • Check if top == -1 (stack is empty).
        o If empty, print: "The stack is empty".
    • If the stack is not empty:
```

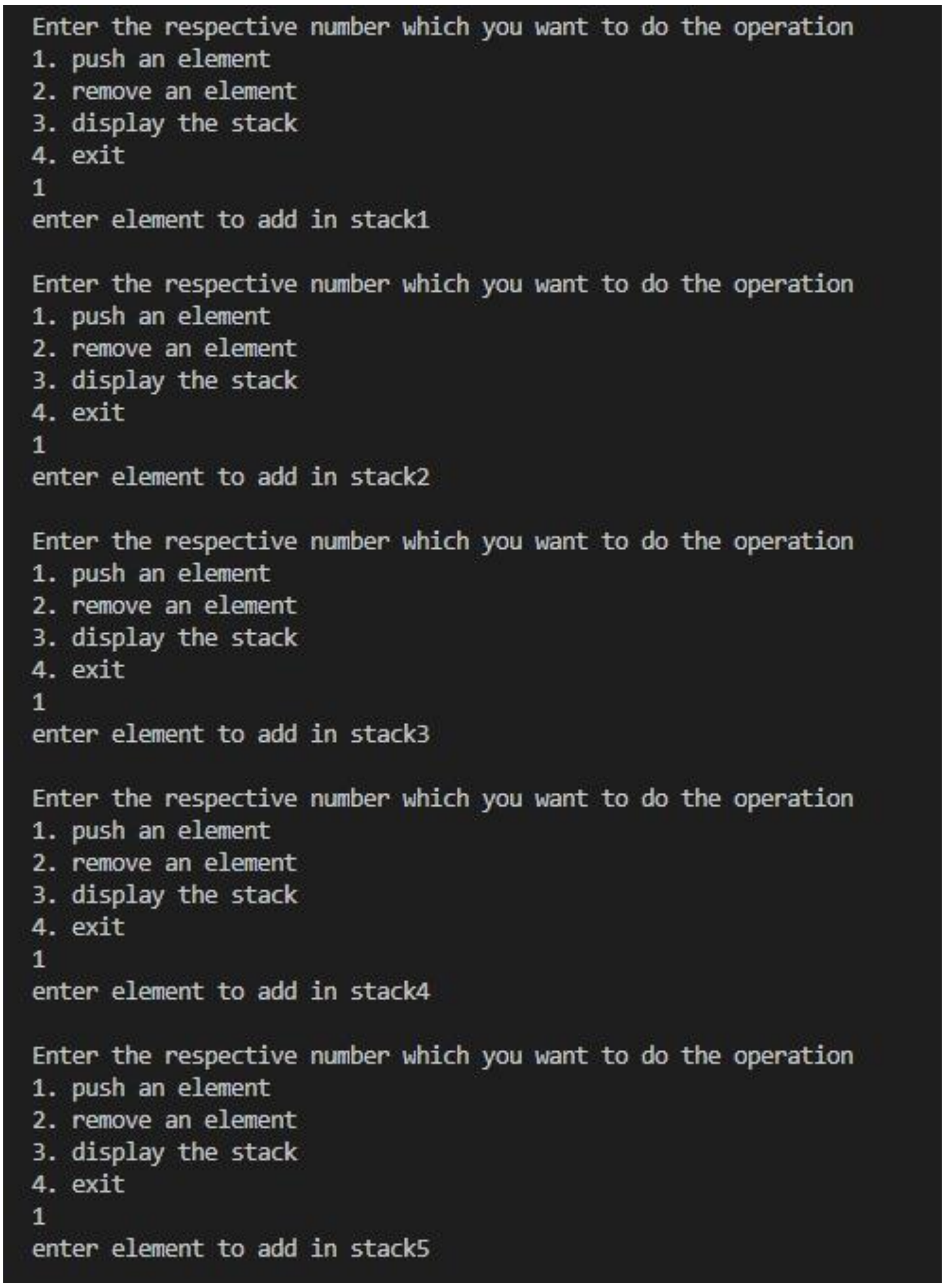
  - Loop from 0 to top.
  - Print each element in a.
7. **Repeat steps 3-6 until the user chooses EXIT.**
8. **End**

```
#include <stdio.h>

#include <stdlib.h>

#define n 5


int top=-1;

int a[n];


void push();

void pop();

void display();


int main()

{

  int choice;

  while(1)

  {

    printf("\nEnter the respective number which you want to do the operation \n1. push an element\n2. remove an element\n3. display the stack\n4. exit\n");

    scanf("%d", &choice);

    switch(choice)

    {


      case 4: exit(1);

          break;


      case 1: push();

          break;
```

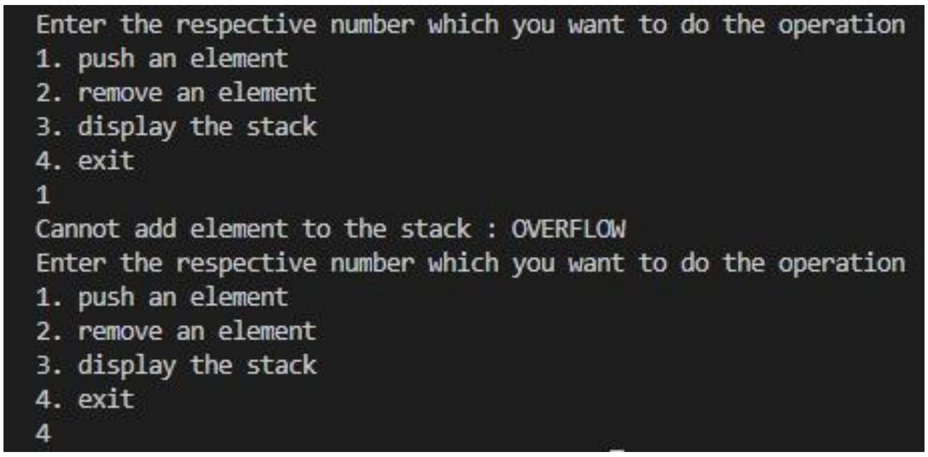
```
            case 2: pop();
                    break;

            case 3: display();
                    break;

            default: printf("enter a valid choice");
        }
    }
}

void push()
{
    if(top==n-1)
    {
        printf("Cannot add element to the stack : OVERFLOW");
    }

    else{
        top++;
        printf("enter element to add in stack");
        int x;
        scanf("%d", &x);
        a[top]=x;
    }
}

void pop()
{
    if(top==-1)
    {
        printf("No element to pop in the stack : UNDERFLOW");
    }

    else{
        top--;
    }
}

void display()
{
    printf("\nstack is : \n");

    if(top==-1)
    {
        printf("The stack is empty");
        return ;
    }

    for(int i=0; i<=top; i++)
    {
        printf("%d\t", a[i]);
    }
}
```

OUTPUT :

1) OVERFLOW

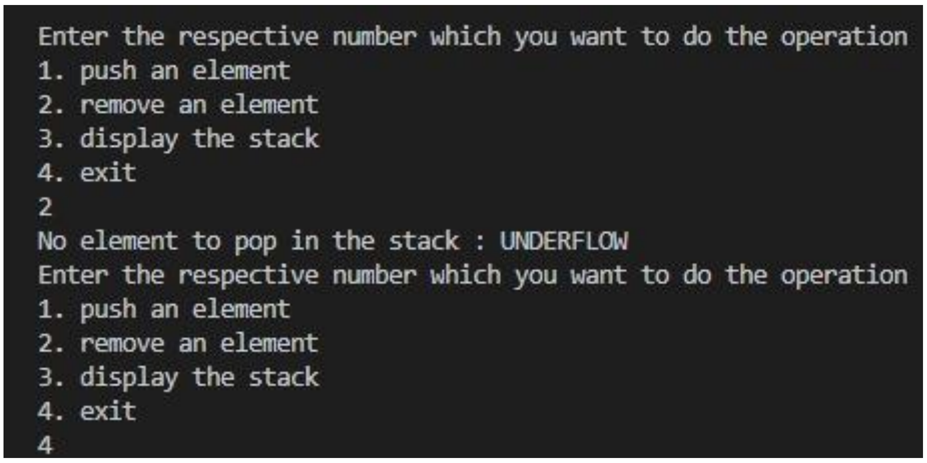


```
Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
1
enter element to add in stack1

Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
1
enter element to add in stack2

Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
1
enter element to add in stack3

Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
1
enter element to add in stack4

Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
1
enter element to add in stack5
```



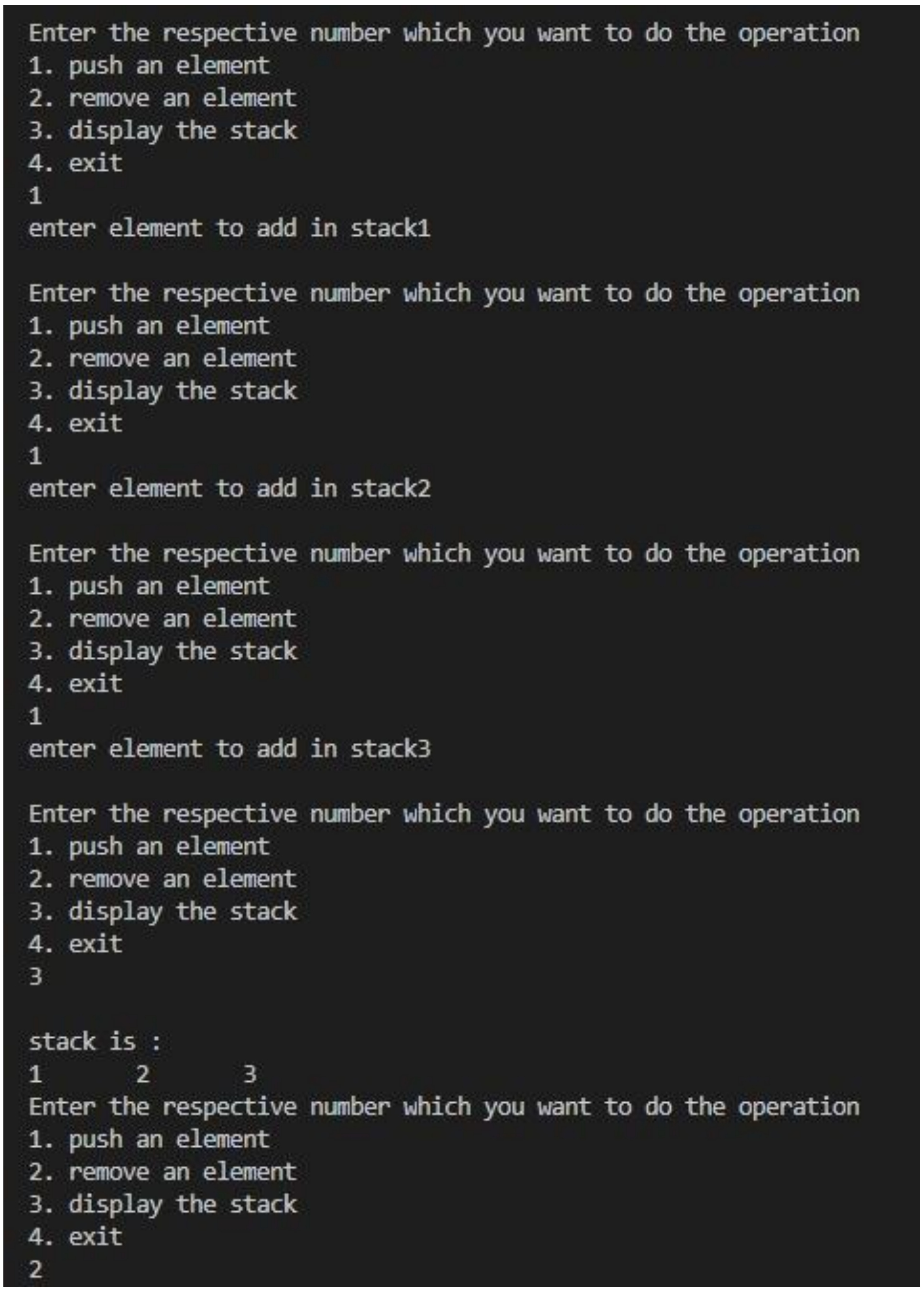
```
Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
1
Cannot add element to the stack : OVERFLOW
Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
4
```

2) Operation on an empty stack

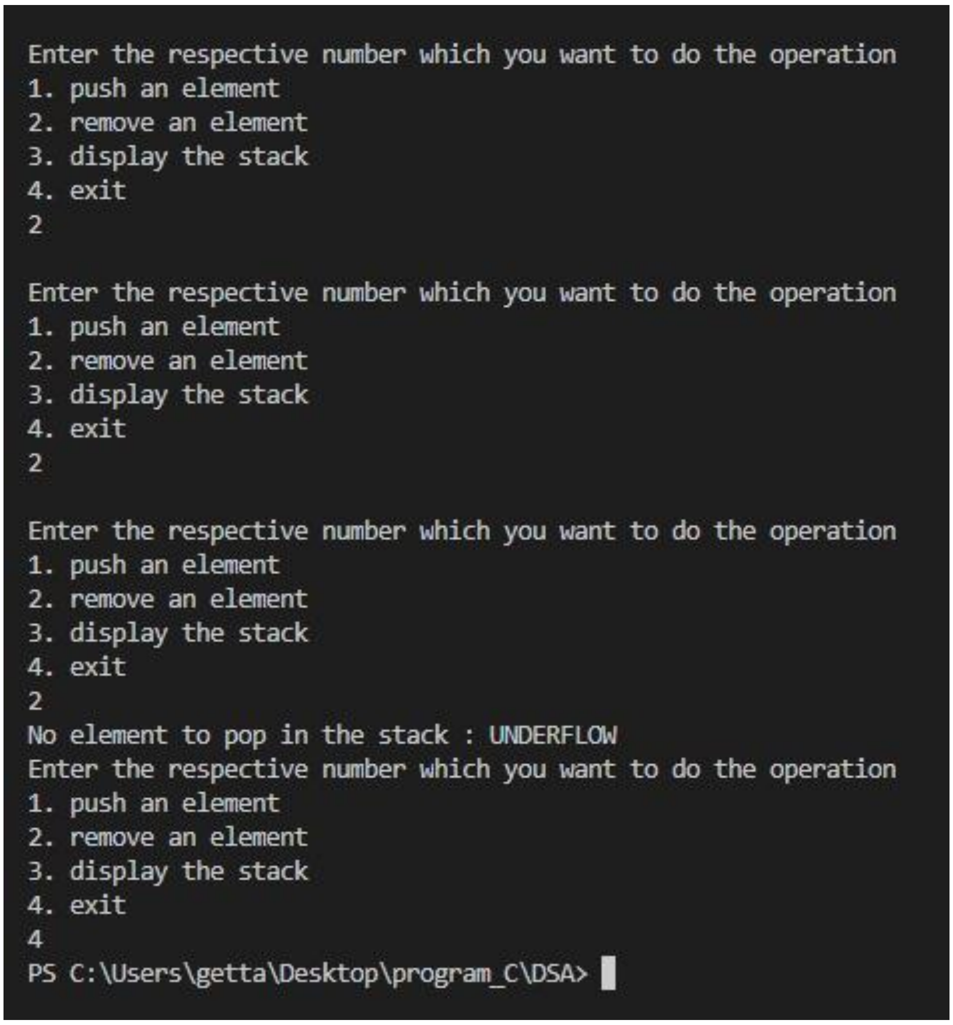


```
Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
2
No element to pop in the stack : UNDERFLOW
Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
4
```

3) Testing general push,pop and display function.Putting elements in stack and removing them one by one until its empty.



```
Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
1
enter element to add in stack1

Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
1
enter element to add in stack2

Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
1
enter element to add in stack3

Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
3

stack is :
1       2       3
Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
2
```



```
Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
2

Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
2

Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
2
No element to pop in the stack : UNDERFLOW
Enter the respective number which you want to do the operation
1. push an element
2. remove an element
3. display the stack
4. exit
4
PS C:\Users\getta\Desktop\program_C\DSA>
```

QUESTION – 2 menu driven code for queue

Pseudocode

```
1. Start
2. Initialize:
    • a[n] (queue array of size 3)
    • front = -1
    • rear = -1
3. Display menu options:
    • 1: ENQUEUE
    • 2: DEQUEUE
    • 3: DISPLAY
    • 4: EXIT
4. If the user selects ENQUEUE:
    • Check if rear == n - 1 (queue is full).
```

        - If full, print: "The queue is full!"
    - If the queue is not full:
        - If front == -1, set front = 0.
        - Increment rear by 1.
        - Input the element and store it in a[rear].
5. If the user selects DEQUEUE:
    - Check if front == -1 or front > rear (queue is empty).
        - If empty, print: "The queue is empty!"
    - If the queue is not empty:
        - Retrieve the element x = a[front].
        - Increment front by 1.
6. If the user selects DISPLAY:
    - Check if front == -1 or front > rear (queue is empty).
        - If empty, print: "The queue is empty!"
    - If the queue is not empty:
        - Loop from front to rear.
        - Print all elements of a in this range.
7. Repeat steps 3-6 until the user chooses EXIT.
8. End

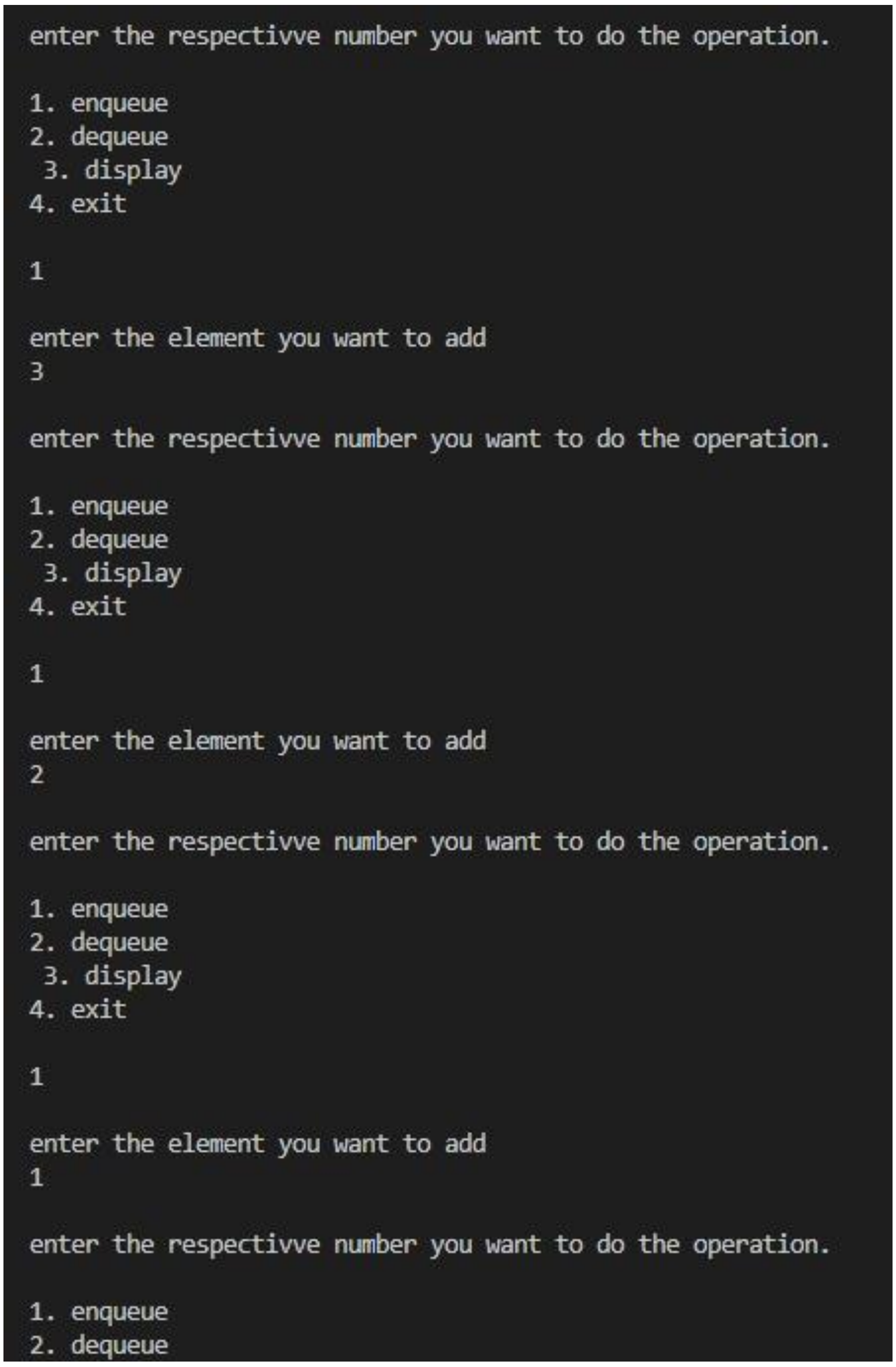
```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define n 3


int front=-1;

int rear=-1;


int a[n];


void enqueue();
void dequeue();
void display();
bool Isqueuefull();
bool Isqueueempty();

int main()
{
   int choice;
   while(1)
   {
      printf("\nenter the respectivve number you want to do the operation. \n");
      printf("\n1. enqueue\n2. dequeue\n 3. display\n4. exit\n\n");
      scanf("%d", &choice);
      switch(choice)
      {
         case 1: enqueue();
               break;

         case 2: dequeue();
               break;

         case 3: display();
               break;

         case 4: exit(1) ;

         default : printf("\nenter a valid choice\n");
      }
   }
   return 0;
}
```

```
bool Isqueuefull()
{
   if(rear==n-1)
   {
      return true;
   }
   else{
      return false;
   }
}

bool Isqueueempty()
{
   if(front==-1 || front>rear)
   {
      return true;
   }
   else{
      return false;
   }
}

void enqueue()
{
   if(Isqueuefull())
   {
      printf("\nThe queue is full !\n");
      return ;
   }
```

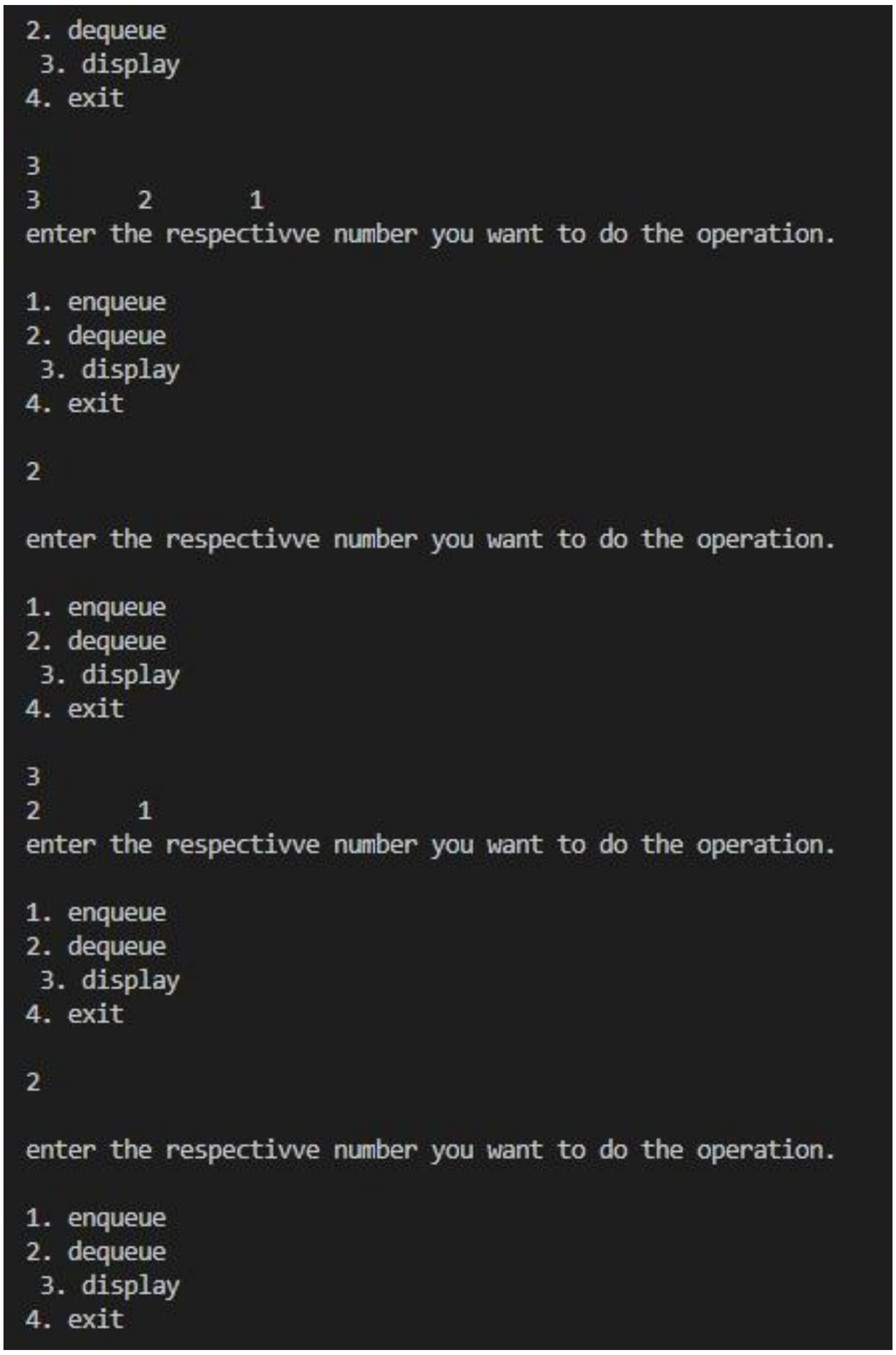
```
    if(front==-1)
    {
        front=0;
    }

    rear++;
    int x;
    printf("\nenter the element you want to add\n");
    scanf("%d", &x);
    a[rear]=x;
}

void dequeue()
{
    if(Isqueueempty())
    {
        printf("\nThe queue is empty\n");
        return ;
    }

    else{
        int x;
        x=a[front];
        front++;
    }
}

void display()
{
    if(Isqueueempty())
    {
        printf("\nThe queue is empty\n");
        return ;
    }

    else{
        for(int i=front; i<=rear; i++)
        {
            printf("%d\t", a[i]);
        }
    }
}
```

TEST CASES :

1). General operations on a queue

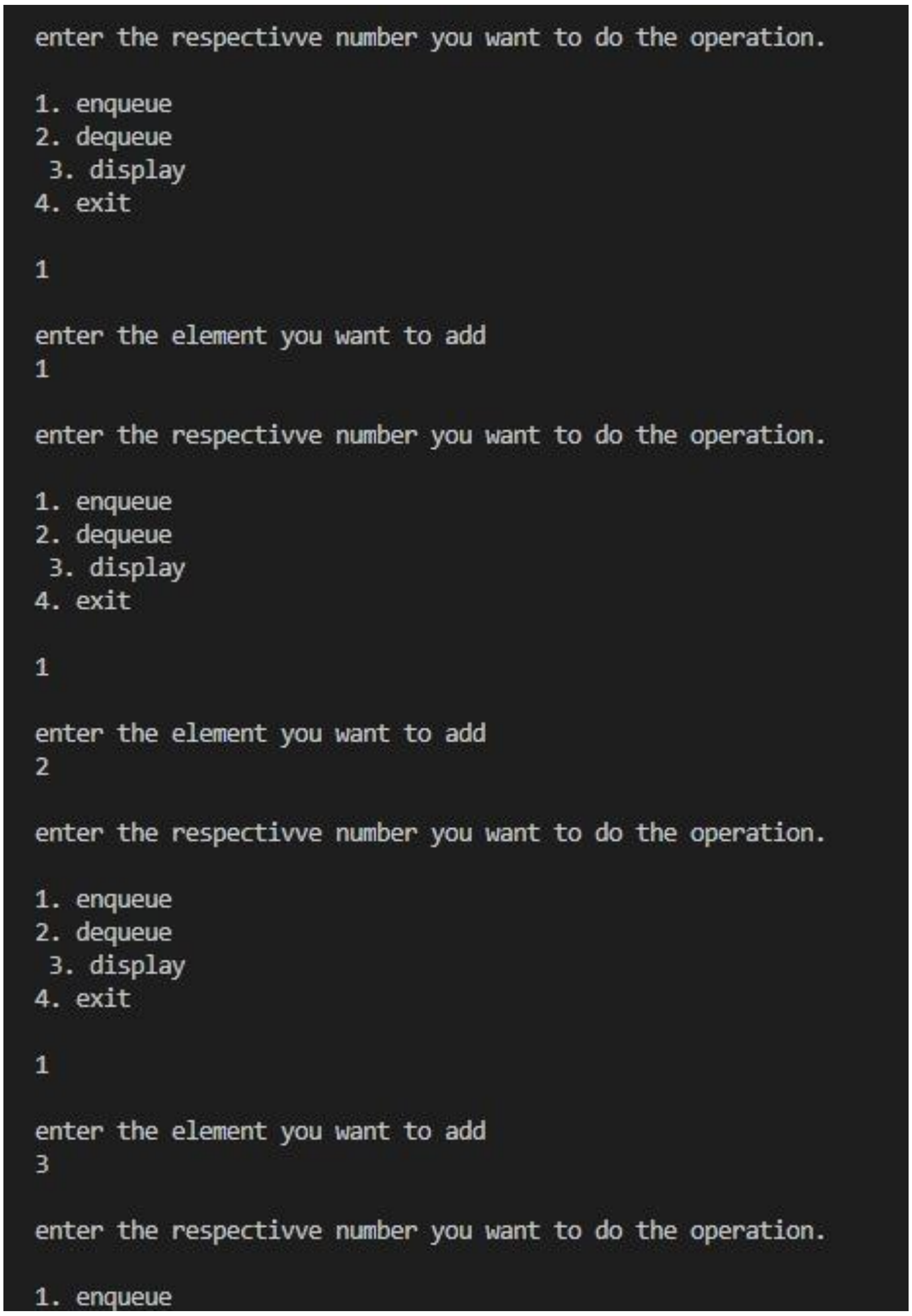


```
enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

1

enter the element you want to add
3

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

1

enter the element you want to add
2

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

1

enter the element you want to add
1

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
```

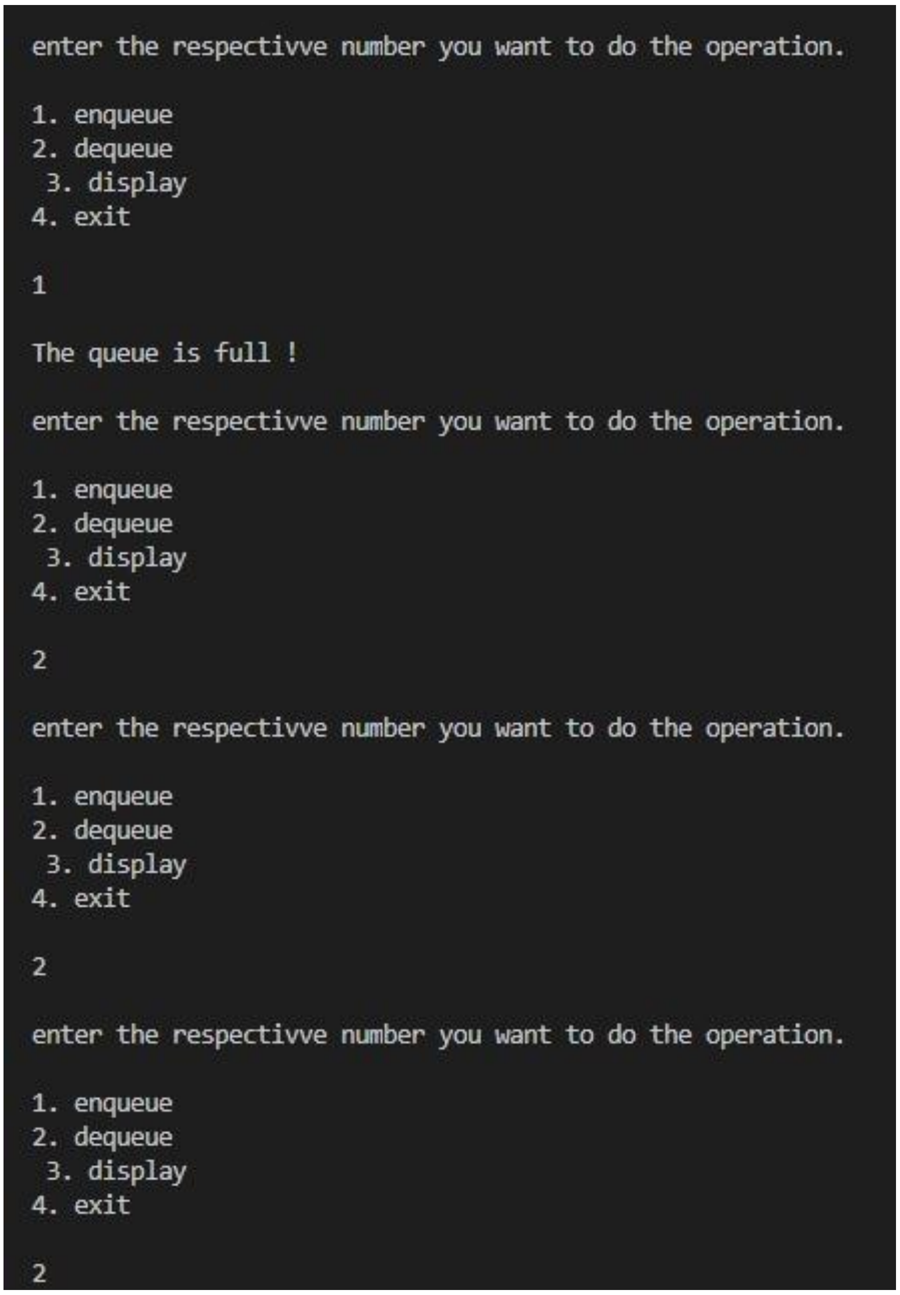


```
2. dequeue
 3. display
4. exit

3
3       2       1
enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

2

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

3
2       1
enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

2

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit
```

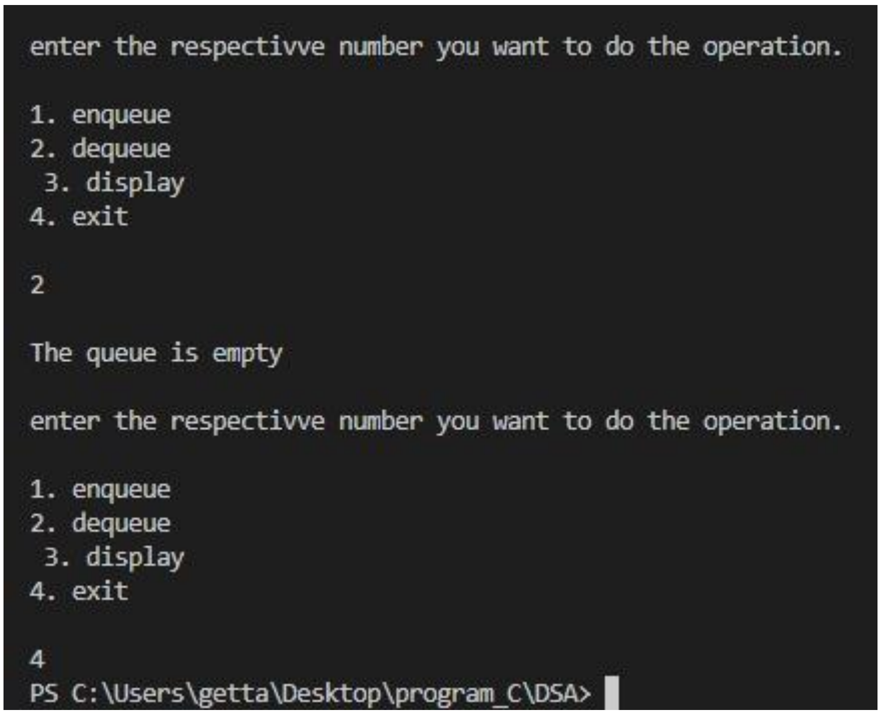
2). Overflowing the queue and deleting elements until its empty and displaying an empty queue



```
enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

1

enter the element you want to add
1

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

1

enter the element you want to add
2

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

1

enter the element you want to add
3

enter the respectivve number you want to do the operation.

1. enqueue
```

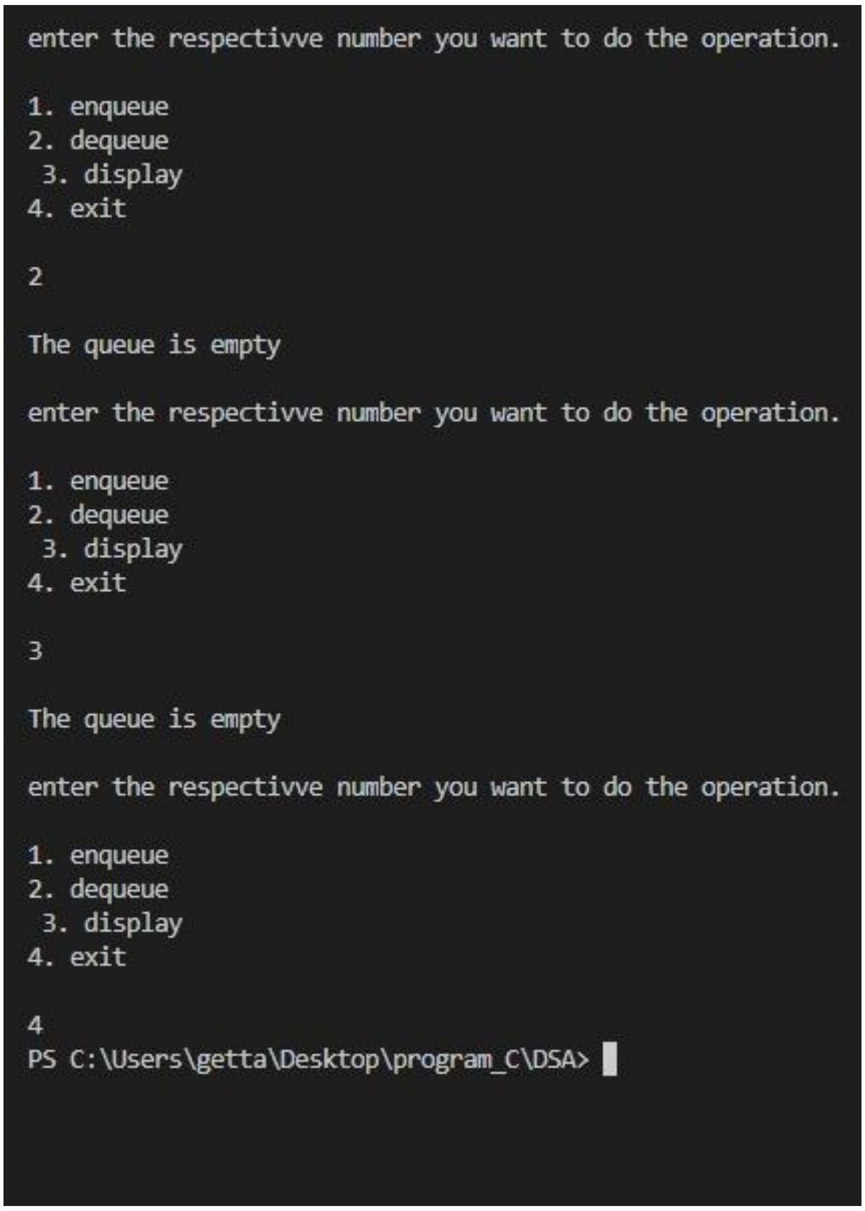


```
enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

1

The queue is full !

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

2

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

2

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

2
```



```
enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

2

The queue is empty

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

4
PS C:\Users\getta\Desktop\program_C\DSA>
```

3). Operations on an empty queue.



```
enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

2

The queue is empty

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

3

The queue is empty

enter the respectivve number you want to do the operation.

1. enqueue
2. dequeue
 3. display
4. exit

4
PS C:\Users\getta\Desktop\program_C\DSA> 
```

QUESTION – 3 Code for menu driven circular queue-

Pseudo code

```
1. Start
2. Initialize:
    o A (pointer to dynamically allocated queue)
    o front = -1
    o rear = -1
    o MAX_SIZE = 0
3. Input the size of the queue (MAX_SIZE):
    o If MAX_SIZE <= 0:
        ▪ Print: "Invalid size. Exiting program."
```

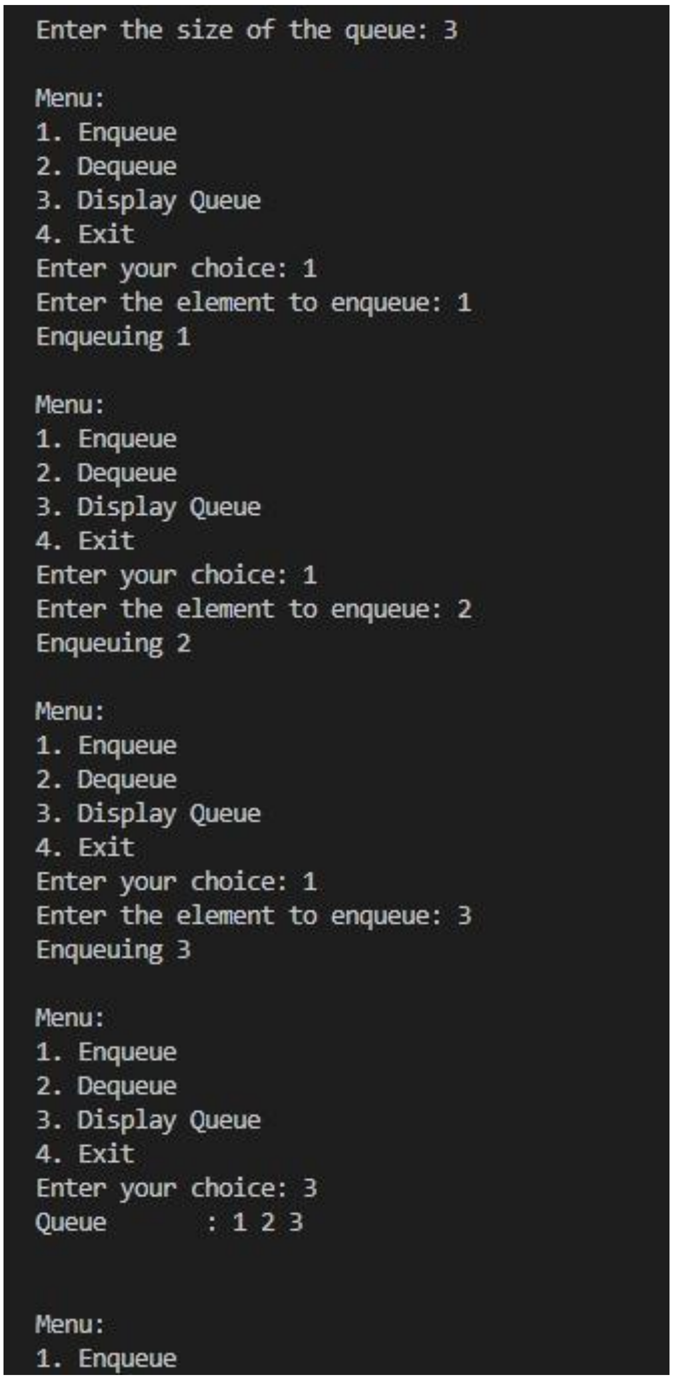
        - End the program.
    - Dynamically allocate memory for A with size MAX_SIZE.
    - If memory allocation fails:
        - Print: "Memory allocation failed. Exiting program."
        - End the program.
4. Display menu options:
    - 1: ENQUEUE
    - 2: DEQUEUE
    - 3: DISPLAY QUEUE
    - 4: EXIT
5. If the user selects ENQUEUE:
    - Input the element to enqueue (x).
    - Check if (rear + 1) % MAX_SIZE == front (queue is full):
        - If full, print: "Error: Queue is Full".
    - If the queue is not full:
        - If front == -1 and rear == -1 (queue is empty):
            - Set front = 0 and rear = 0.
        - Otherwise:
            - Set rear = (rear + 1) % MAX_SIZE.
        - Store the element x in A[rear].
6. If the user selects DEQUEUE:
    - Check if front == -1 and rear == -1 (queue is empty):
        - If empty, print: "Error: Queue is Empty".
    - If the queue is not empty:
        - Check if front == rear (only one element in the queue):
            - Set front = -1 and rear = -1.
        - Otherwise:
            - Set front = (front + 1) % MAX_SIZE.
7. If the user selects DISPLAY QUEUE:
    - Check if front == -1 and rear == -1 (queue is empty):
        - If empty, print: "Queue: Empty".

        - If the queue is not empty:
            - Calculate the number of elements:
                - count = (rear + MAX_SIZE - front) % MAX_SIZE + 1.
            - Print all elements from front to rear using modulo indexing.
8. Repeat steps 4-7 until the user selects EXIT:
    - Print: "Exiting program."
9. Free the allocated memory for A.
10. End

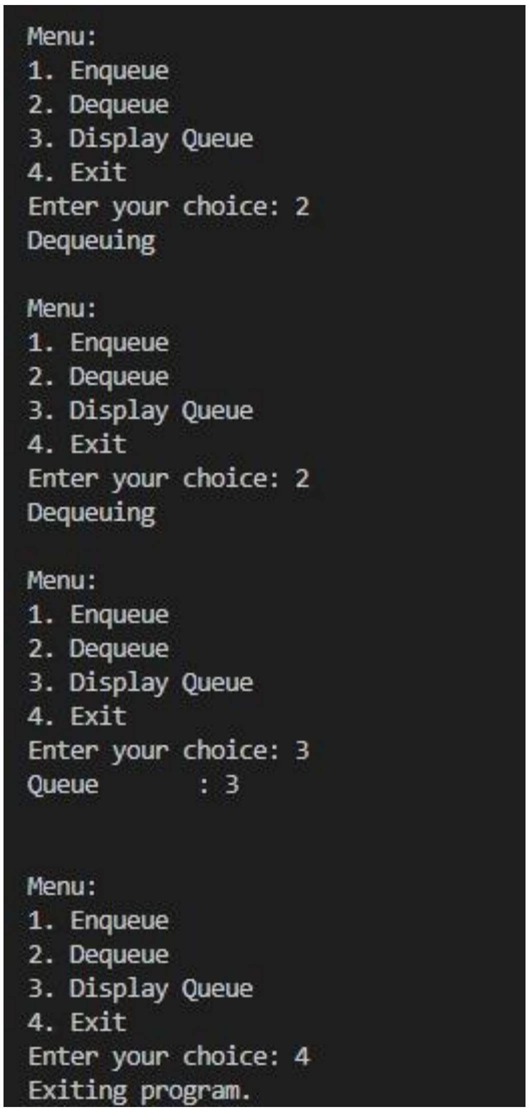
```
#include <stdio.h>

#include <stdlib.h>


int *A;

int front = -1;

int rear = -1;

int MAX_SIZE = 0;


int IsEmpty() {

    return (front == -1 && rear == -1);

}


int IsFull() {

    return (rear + 1) % MAX_SIZE == front;

}


void Enqueue(int x) {

    printf("Enqueuing %d\n", x);

    if (IsFull()) {

        printf("Error: Queue is Full\n");

        return;

    }

    if (IsEmpty()) {
        front = rear = 0;
    } else {
        rear = (rear + 1) % MAX_SIZE;
    }
    A[rear] = x;
}

void Dequeue() {
    printf("Dequeuing\n");
    if (IsEmpty()) {
        printf("Error: Queue is Empty\n");
        return;
    }
    if (front == rear) {
        front = rear = -1;
    } else {
        front = (front + 1) % MAX_SIZE;
    }
}

int Front() {
    if (IsEmpty()) {
        printf("Error: Cannot return front from empty queue\n");
        return -1;
    }
    return A[front];
}

void display() {
    int i = 0;
    if (IsEmpty()) {
        printf("Queue       : Empty\n\n");
        return;
    }
    int count = (rear + MAX_SIZE - front) % MAX_SIZE + 1;
    printf("Queue       : ");
    for (i = 0; i < count; i++) {
        int index = (front + i) % MAX_SIZE;
        printf("%d ", A[index]);
    }
    printf("\n\n");
}

int main() {
    printf("Enter the size of the queue: ");
    scanf("%d", &MAX_SIZE);

    if (MAX_SIZE <= 0) {
        printf("Invalid size. Exiting program.\n");
        return 0;
    }

    A = (int *)malloc(MAX_SIZE * sizeof(int));
    if (A == NULL) {
        printf("Memory allocation failed. Exiting program.\n");
        return 0;
    }

    int choice, value;

    do {
        printf("\nMenu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display Queue\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to enqueue: ");
                scanf("%d", &value);
                Enqueue(value);
                break;

            case 2:
                Dequeue();
                break;

            case 3:
                display();
                break;

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

            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);
    free(A);

    return 0;

}
```

TESTCASES :

1). General operation on circular queue

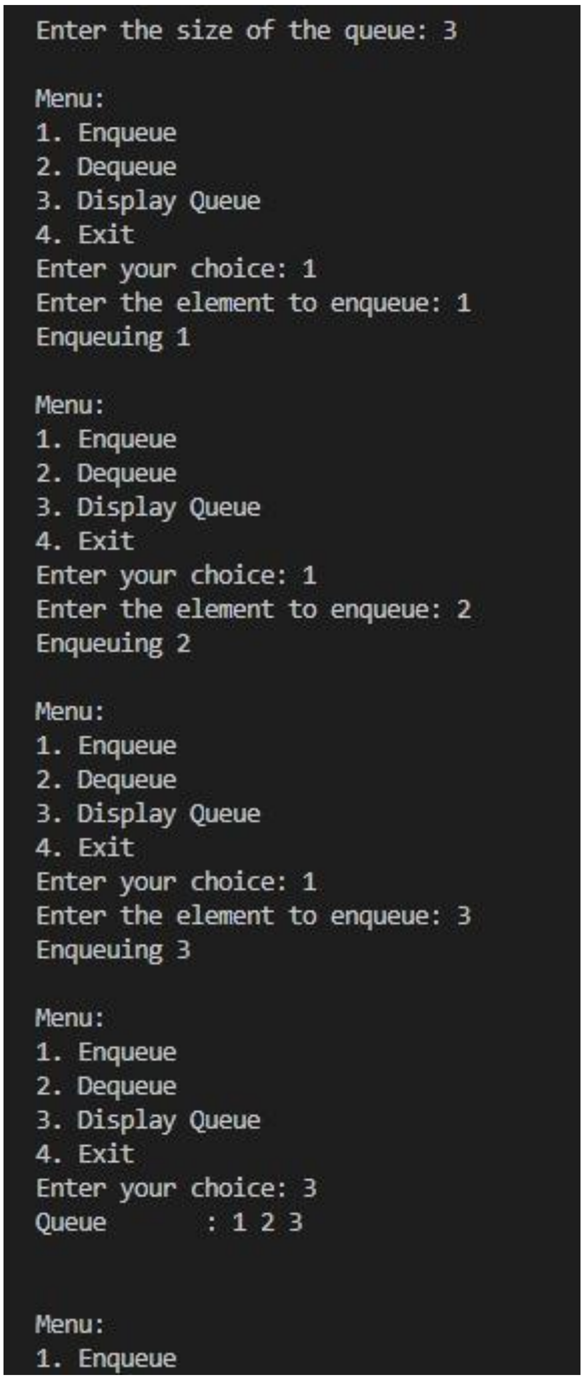


```
Enter the size of the queue: 3

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 1
Enqueuing 1

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 2
Enqueuing 2

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 3
Enqueuing 3

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 3
Queue       : 1 2 3


Menu:
1. Enqueue
```

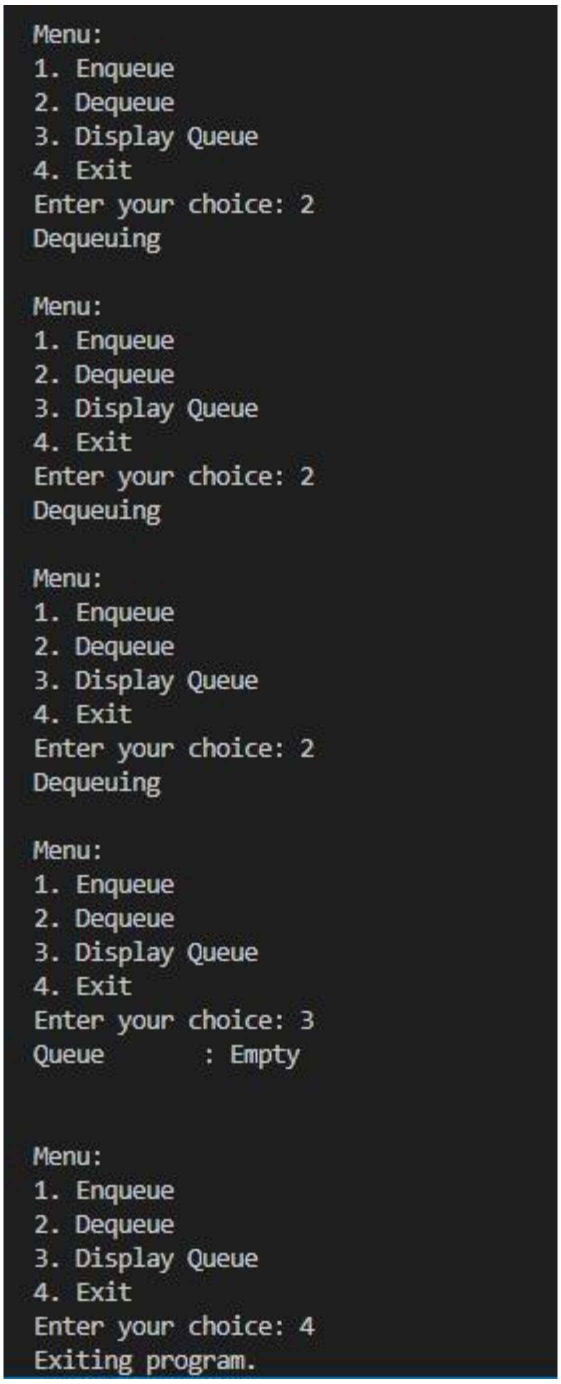


```
Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 2
Dequeuing

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 2
Dequeuing

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 3
Queue       : 3


Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 4
Exiting program.
```

2). Operation on an empty queue and overflowing it.

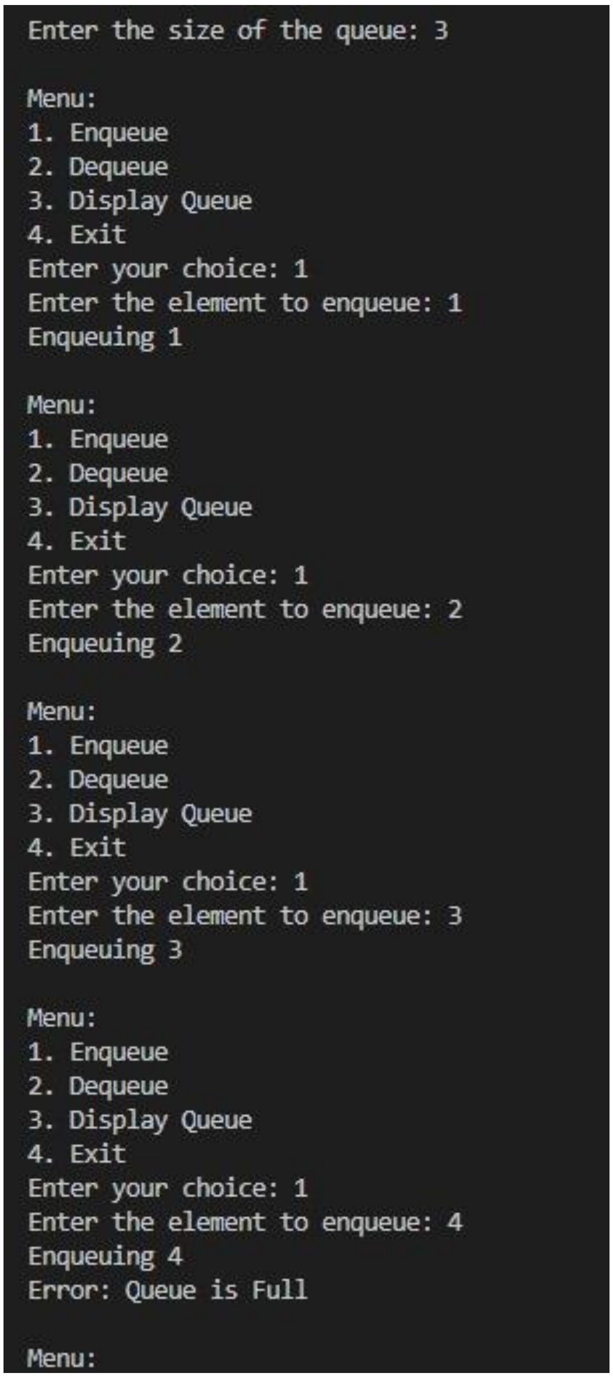


```
Enter the size of the queue: 3

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 1
Enqueuing 1

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 2
Enqueuing 2

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 3
Enqueuing 3

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 3
Queue       : 1 2 3


Menu:
1. Enqueue
```



```
Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 2
Dequeuing

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 2
Dequeuing

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 2
Dequeuing

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 3
Queue       : Empty


Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 4
Exiting program.
```

3). testing the circular nature of the queue


```
Enter the size of the queue: 3

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 1
Enqueuing 1

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 2
Enqueuing 2

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 3
Enqueuing 3

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 4
Enqueuing 4
Error: Queue is Full

Menu:
```

QUESTION – 4 Code for menu driven linked list

Pseudo Code

```
1. Start
2. Initialize:
    o head = NULL (pointer to the head of the linked list).
    o temp = NULL (temporary pointer for traversing or updating the list).
3. Display menu options repeatedly:
    o 1: Create nodes
```

- 2: Display linked list
- 3: Insert at the beginning
- 4: Insert at the end
- 5: Insert in the middle
- 6: Delete a node
- 7: Exit

4. If the user selects CREATE NODES:
    - Input the number of nodes (n).
    - If n < 1:
        - Print: "Enter a valid number of nodes (must be at least 1)".
        - Skip the operation.
    - For i = 1 to n:
        - Input the data for the node (data).
        - Create a new node using createnode(data).
        - If head == NULL:
            - Set head = newnode.
            - Set temp = head.
        - Else:
            - Set temp->next = newnode.
            - Update temp = temp->next.
5. If the user selects DISPLAY LINKED LIST:
    - Print the data in all nodes using the display function.
6. If the user selects INSERT AT BEGINNING:
    - Input the data for the new node (d).
    - Create a new node using insertbeg(head, d).
    - Update head with the returned value.
    - Print the updated linked list using display.
7. If the user selects INSERT AT END:
    - Input the data for the new node (d).
    - Create a new node using insertend(head, d).
    - Update head with the returned value.

   - Print the updated linked list using display.
8. If the user selects INSERT IN THE MIDDLE:
   - Check if head == NULL:
     - If true, print: "List is empty. Cannot insert in the middle."
     - Skip the operation.
   - Input the data (d) and position (pos) after which to insert the new node.
   - Count the number of nodes (count) in the list.
   - If pos > count:
     - Print: "Invalid position! Cannot perform the function."
     - Skip the operation.
   - Insert the new node using insertmiddle(head, d, pos, count).
   - Update head with the returned value.
   - Print the updated linked list using display.
9. If the user selects DELETE A NODE:
   - Input the position of the node to delete (pos).
   - Delete the node at the specified position using deletenode(head, pos).
   - Update head with the returned value.
   - Print the updated linked list using display.
10. If the user selects EXIT:
    - Print: "THANK YOU!"
    - Terminate the program.
11. End

```c
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node *next;

};
```

```
struct Node *head = NULL;
struct Node *temp = NULL;

struct Node *createnode(int d) {
  struct Node *newnode = (struct Node *)malloc(sizeof(struct Node));
  if (newnode == NULL) {
    printf("\nMemory allocation failed!\n");
    exit(1);
  }
  newnode->data = d;
  newnode->next = NULL;
  return newnode;
}

void display(struct Node *head) {
  if (head == NULL) {
    printf("List is empty.\n");
    return;
  }
  temp = head;
  while (temp != NULL) {
    printf("%d -> ", temp->data);
    temp = temp->next;
  }
  printf("NULL\n");
}

struct Node *insertbeg(struct Node *head, int d) {
  struct Node *beg = createnode(d);
  beg->next = head;
  head = beg;
```

```
    return head;
}

struct Node *insertend(struct Node *head, int d) {
    struct Node *end = createnode(d);
    if (head == NULL) {
        return end;  // If the list is empty, the new node becomes the head.
    }
    temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = end;
    return head;
}

struct Node *insertmiddle(struct Node *head, int d, int pos, int n) {
    if (head == NULL) {
        printf("List is empty. Cannot insert in the middle.\n");
        return head;
    }
    if (pos > n) {
        printf("\nInvalid position! Cannot perform the function.\n");
        return head;
    }
    struct Node *mid = createnode(d);
    temp = head;
    int i;
    for (i = 0; i < pos - 1; i++) {
        temp = temp->next;
    }
    mid->next = temp->next;
    temp->next = mid;
    return head;
}

struct Node *deletenode(struct Node *head, int pos) {
    if (head == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return head;
    }
    if (pos == 1) {
        struct Node *toDelete = head;
        head = head->next;
        free(toDelete);
        return head;
    }
    temp = head;
    int i;
    for (i = 1; i < pos - 1 && temp->next != NULL; i++) {
        temp = temp->next;
    }
    if (temp->next == NULL) {
        printf("Invalid position! Nothing to delete.\n");
        return head;
    }
    struct Node *toDelete = temp->next;
    temp->next = toDelete->next;
    free(toDelete);
    return head;
}
```

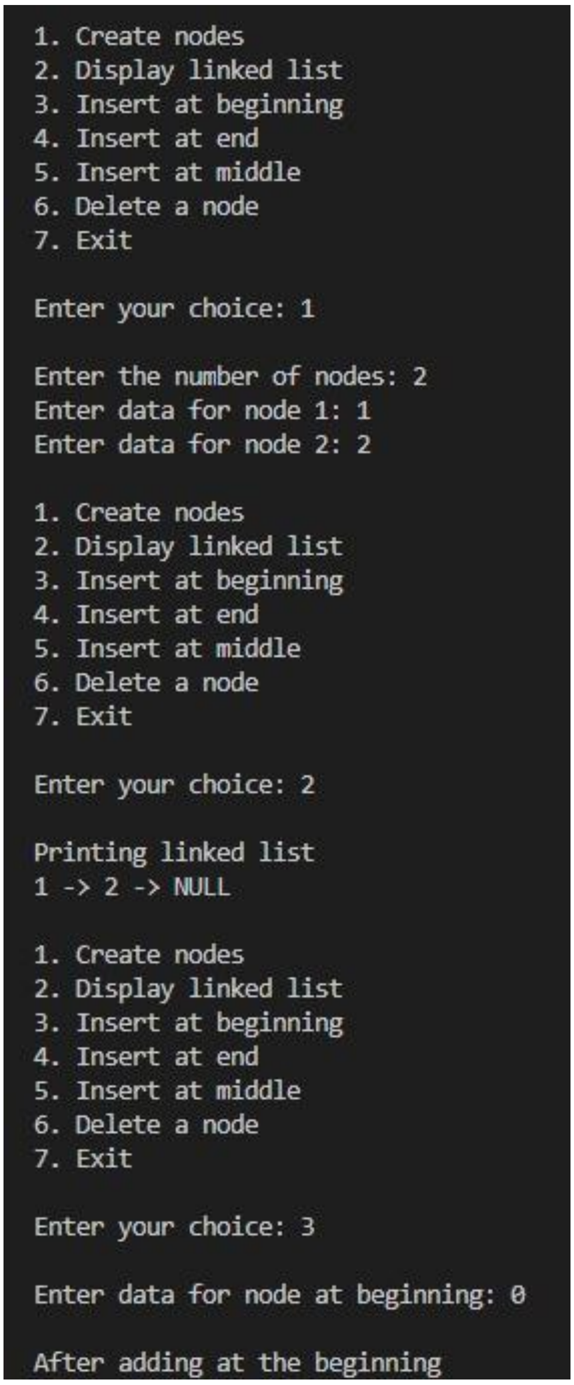
```
int main() {
  int num = 0, i;
  while (num != 7) {
    printf("\n1. Create nodes\n2. Display linked list\n3. Insert at beginning\n4. Insert at end\n5.
Insert at middle\n6. Delete a node\n7. Exit\n");
    printf("\nEnter your choice: ");
    scanf("%d", &num);

    switch (num) {
      case 1: {
        printf("\nEnter the number of nodes: ");
        int n;
        scanf("%d", &n);
        if (n < 1) {
          printf("Enter a valid number of nodes (must be at least 1).\n");
          break;
        }
        for (i = 0; i < n; i++) {
          printf("Enter data for node %d: ", (i + 1));
          int data;
          scanf("%d", &data);
          struct Node *newnode = createnode(data);
          if (head == NULL) {
            head = newnode;
            temp = head;
          } else {
            temp->next = newnode;
            temp = temp->next;
          }
        }
        break;
```

```
}
case 2:
   printf("\nPrinting linked list\n");
   display(head);
   break;
case 3: {
   printf("\nEnter data for node at beginning: ");
   int d;
   scanf("%d", &d);
   head = insertbeg(head, d);
   printf("\nAfter adding at the beginning\n");
   display(head);
   break;
}
case 4: {
   printf("\nEnter data for node at the end: ");
   int d;
   scanf("%d", &d);
   head = insertend(head, d);
   printf("\nAfter adding at the end\n");
   display(head);
   break;
}
case 5: {
   if (head == NULL) {
      printf("List is empty. Cannot insert in the middle.\n");
      break;
   }
   printf("\nEnter data for node in middle and the position after which to insert: ");
   int d, pos;
   scanf("%d%d", &d, &pos);
```

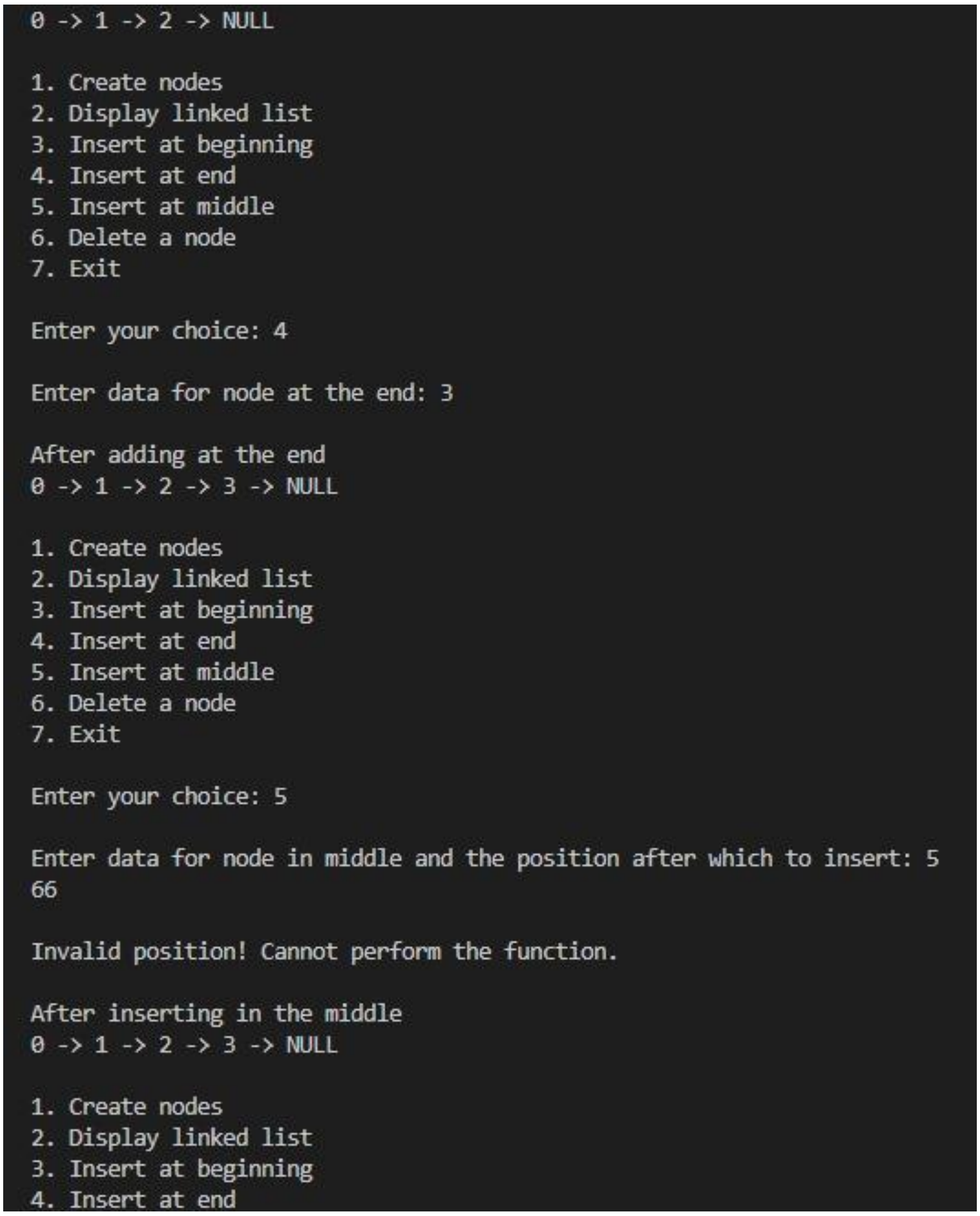
```
            int count = 0;
            temp = head;
            while (temp != NULL) {
                count++;
                temp = temp->next;
            }
            head = insertmiddle(head, d, pos, count);
            printf("\nAfter inserting in the middle\n");
            display(head);
            break;
        }
        case 6: {
            printf("\nEnter the position of the node you want to delete: ");
            int pos;
            scanf("%d", &pos);
            head = deletenode(head, pos);
            printf("\nAfter deletion\n");
            display(head);
            break;
        }
        case 7:
            printf("THANK YOU!\n");
            break;
        default:
            printf("Enter a valid number.\n");
    }
  }
  return 0;
}
```

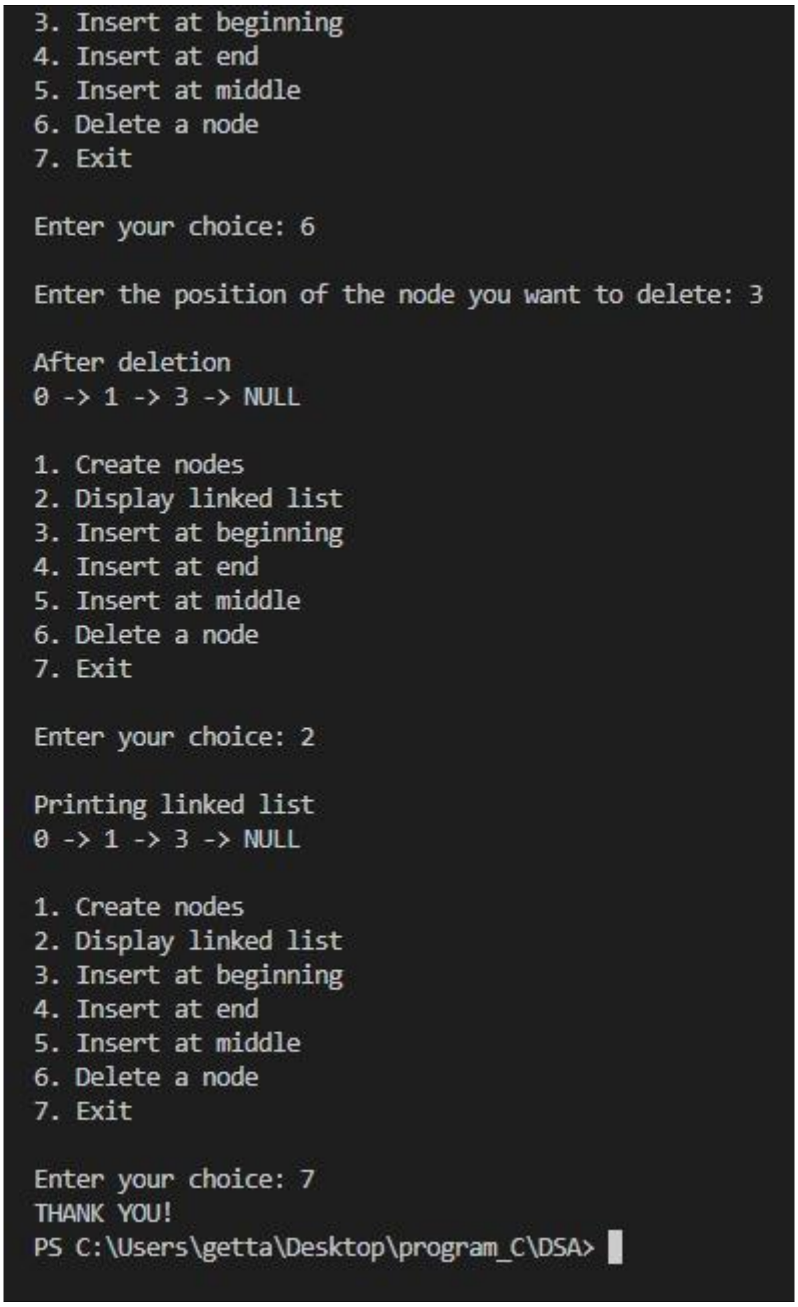
TESTCASES-

1. General operations on a linked list



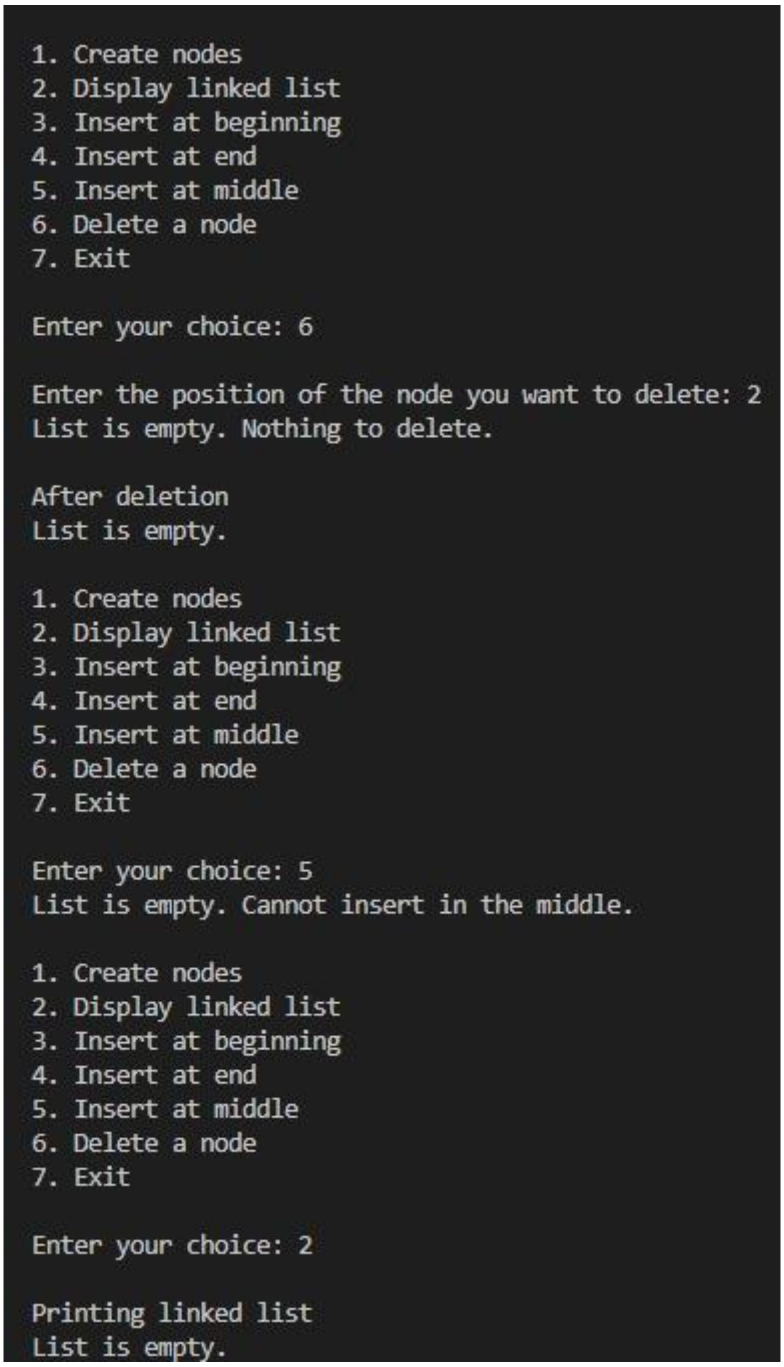
```
1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 1

Enter the number of nodes: 2
Enter data for node 1: 1
Enter data for node 2: 2

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 2

Printing linked list
1 -> 2 -> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 3

Enter data for node at beginning: 0

After adding at the beginning
```


```
0 -> 1 -> 2 -> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 4

Enter data for node at the end: 3

After adding at the end
0 -> 1 -> 2 -> 3 -> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 5

Enter data for node in middle and the position after which to insert: 5
66

Invalid position! Cannot perform the function.

After inserting in the middle
0 -> 1 -> 2 -> 3 -> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
```

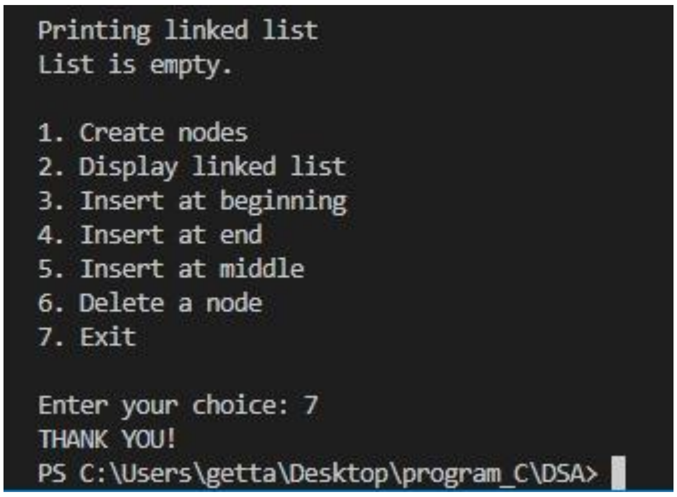


```
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 6

Enter the position of the node you want to delete: 3

After deletion
0 -> 1 -> 3 -> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 2

Printing linked list
0 -> 1 -> 3 -> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 7
THANK YOU!
PS C:\Users\getta\Desktop\program_C\DSA>
```

2). Operations on empty linked list

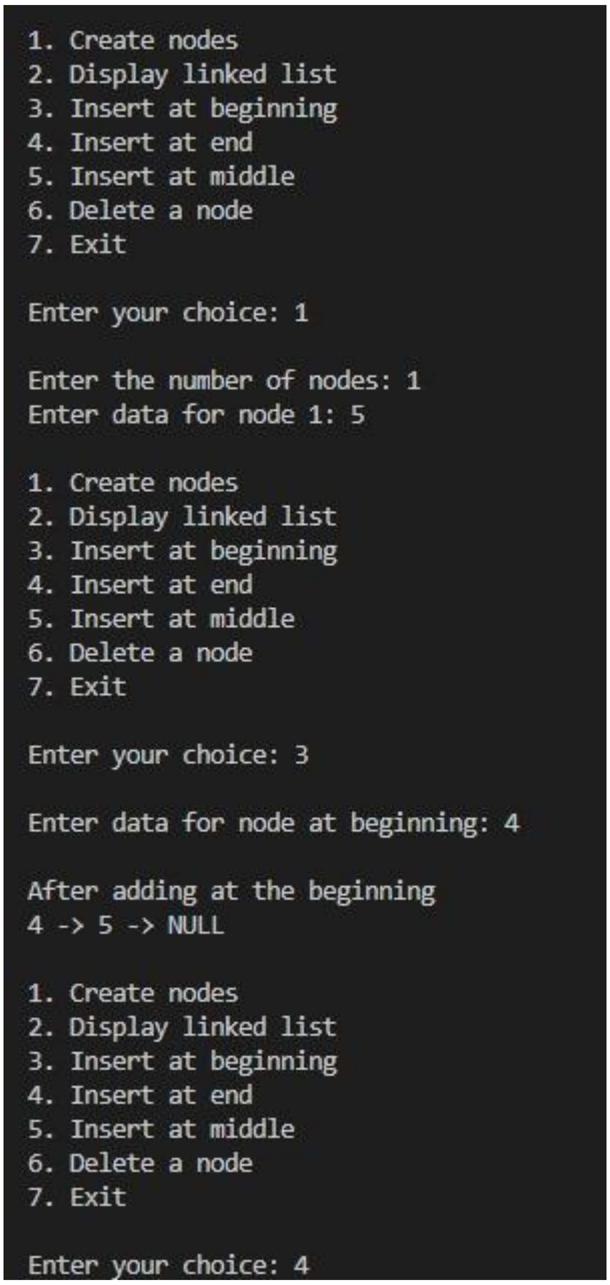


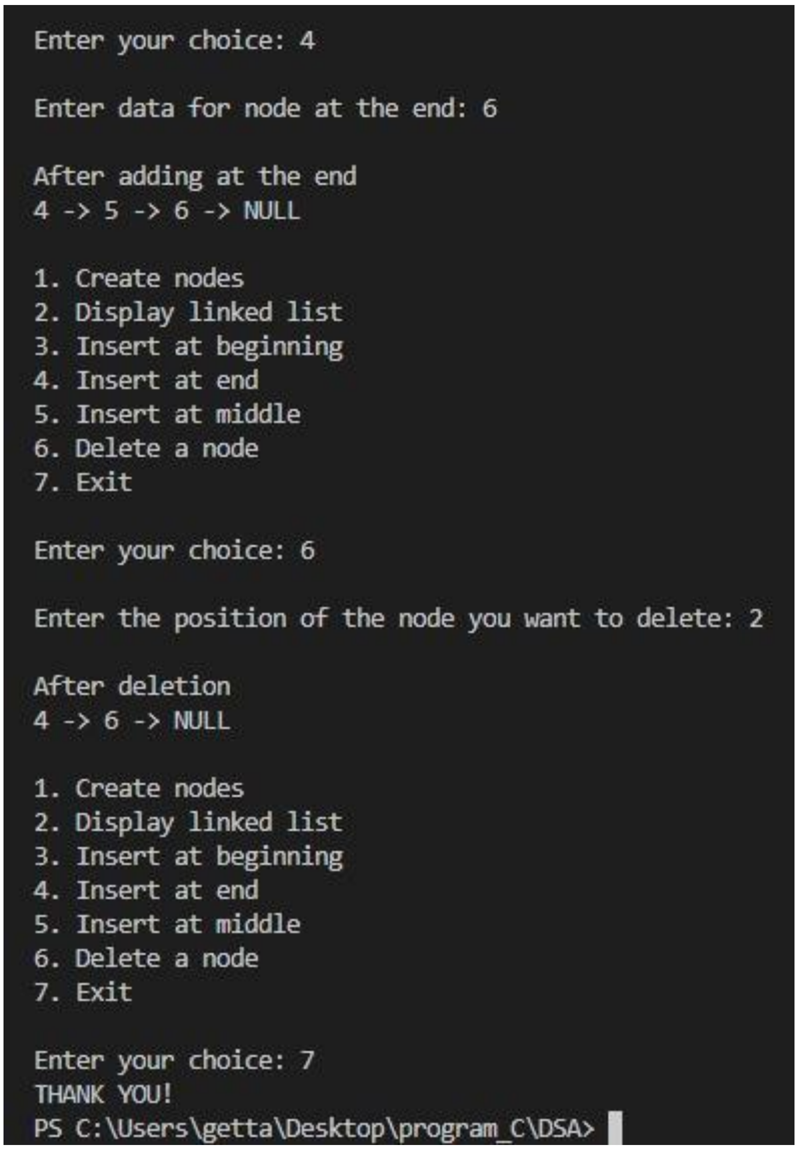
```
1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 6

Enter the position of the node you want to delete: 2
List is empty. Nothing to delete.

After deletion
List is empty.

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 5
List is empty. Cannot insert in the middle.

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 2

Printing linked list
List is empty.
```



```
Printing linked list
List is empty.

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 7
THANK YOU!
PS C:\Users\getta\Desktop\program_C\DSA> 
```

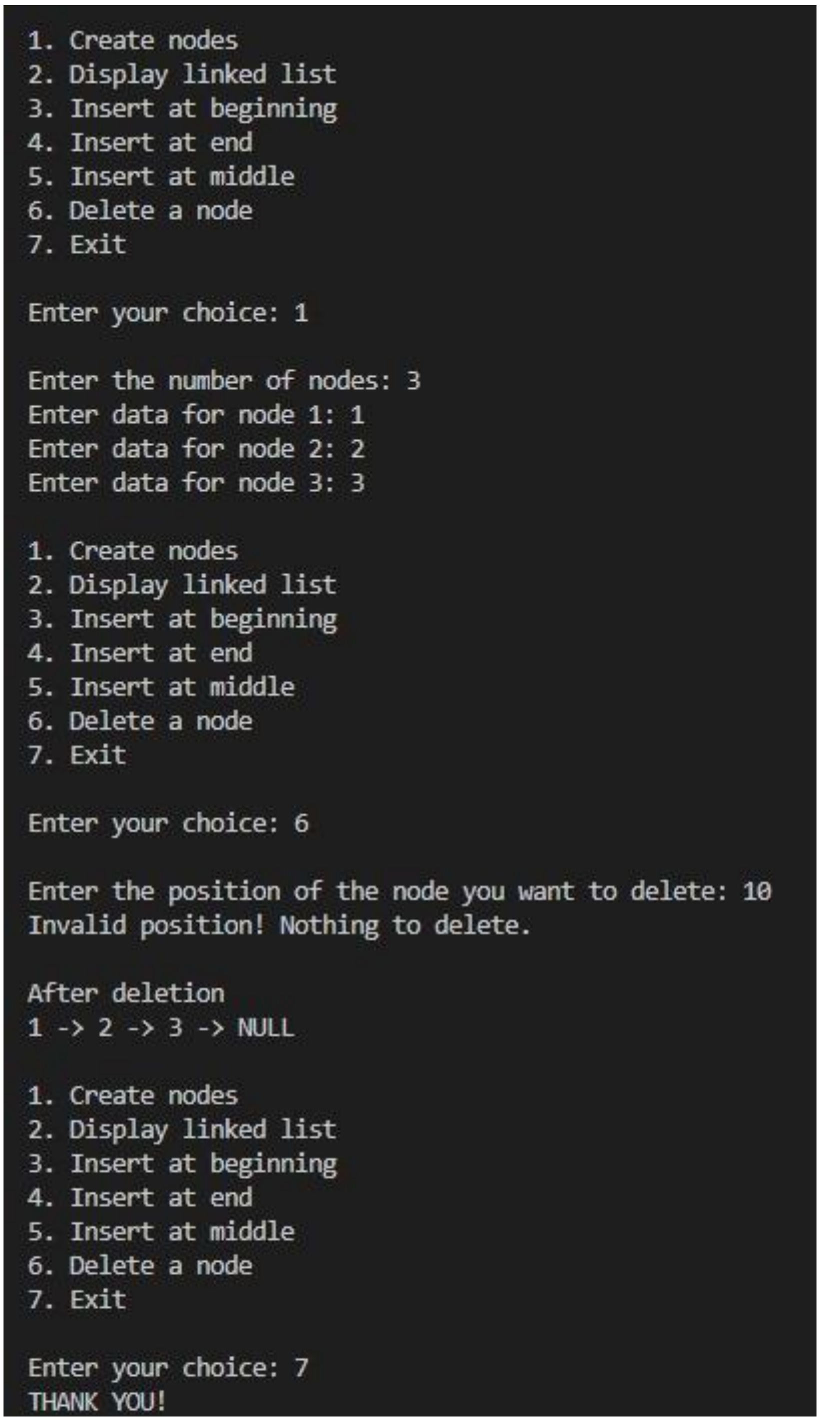
3). Operations on a single linked list.


```
1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 1

Enter the number of nodes: 1
Enter data for node 1: 5

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 3

Enter data for node at beginning: 4

After adding at the beginning
4 -> 5 -> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 4
```



```
Enter your choice: 4

Enter data for node at the end: 6

After adding at the end
4 -> 5 -> 6 -> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 6

Enter the position of the node you want to delete: 2

After deletion
4 -> 6 -> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 7
THANK YOU!
PS C:\Users\getta\Desktop\program_C\DSA>
```

4). Edge case to detect incorrect delete position



```
1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 1

Enter the number of nodes: 3
Enter data for node 1: 1
Enter data for node 2: 2
Enter data for node 3: 3

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 6

Enter the position of the node you want to delete: 10
Invalid position! Nothing to delete.

After deletion
1 -> 2 -> 3 -> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 7
THANK YOU!
```

QUESTION - 5 code for doubly linked list

Pseudo Code

```
1. Start
2. Initialize:
    o head = NULL (pointer to the head of the linked list).
    o temp = NULL (temporary pointer for traversing or updating the list).
3. Display menu options repeatedly:
```

   - 1: Create nodes
   - 2: Display linked list
   - 3: Insert at beginning
   - 4: Insert at end
   - 5: Insert in middle
   - 6: Delete a node
   - 7: Exit
4. If user selects CREATE NODES:
   - Input the number of nodes (n).
   - If n < 1:
     - Print: "Enter a valid number".
     - Skip the operation.
   - For i = 1 to n:
     - Input the data for the node (data).
     - Create a new node using createnode(data).
     - If head == NULL:
       - Set head = newnode.
       - Set temp = head.
     - Else:
       - Set temp->next = newnode.
       - Set newnode->prev = temp.
       - Update temp = temp->next.
5. If user selects DISPLAY LINKED LIST:
   - Traverse the list starting from head.
   - For each node, print node->data followed by "<->".
   - After the last node, print "NULL".
6. If user selects INSERT AT BEGINNING:
   - Input the data for the new node (d).
   - Create a new node using insertbeg(head, d).
   - Update head with the returned value.
   - Print the updated linked list using display(head).

7. If user selects INSERT AT END:
    - Input the data for the new node (d).
    - Create a new node using insertend(head, d).
    - Update head with the returned value.
    - Print the updated linked list using display(head).
8. If user selects INSERT IN THE MIDDLE:
    - Input the data (d) and position (pos) after which to insert the new node.
    - Traverse the list to count the number of nodes (count).
    - If pos > count or pos < 1:
        - Print: "Invalid position".
        - Skip the operation.
    - Insert the new node using insertmiddle(head, d, pos, count).
    - Update head with the returned value.
    - Print the updated linked list using display(head).
9. If user selects DELETE A NODE:
    - Input the position of the node to delete (pos).
    - Delete the node at the specified position using deletenode(head, pos).
    - Update head with the returned value.
    - Print the updated linked list using display(head).
10. If user selects EXIT:
    - Print: "THANK YOU!"
    - Terminate the program.
11. End

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

struct Node {
   int data;
   struct Node *prev;
   struct Node *next;
```

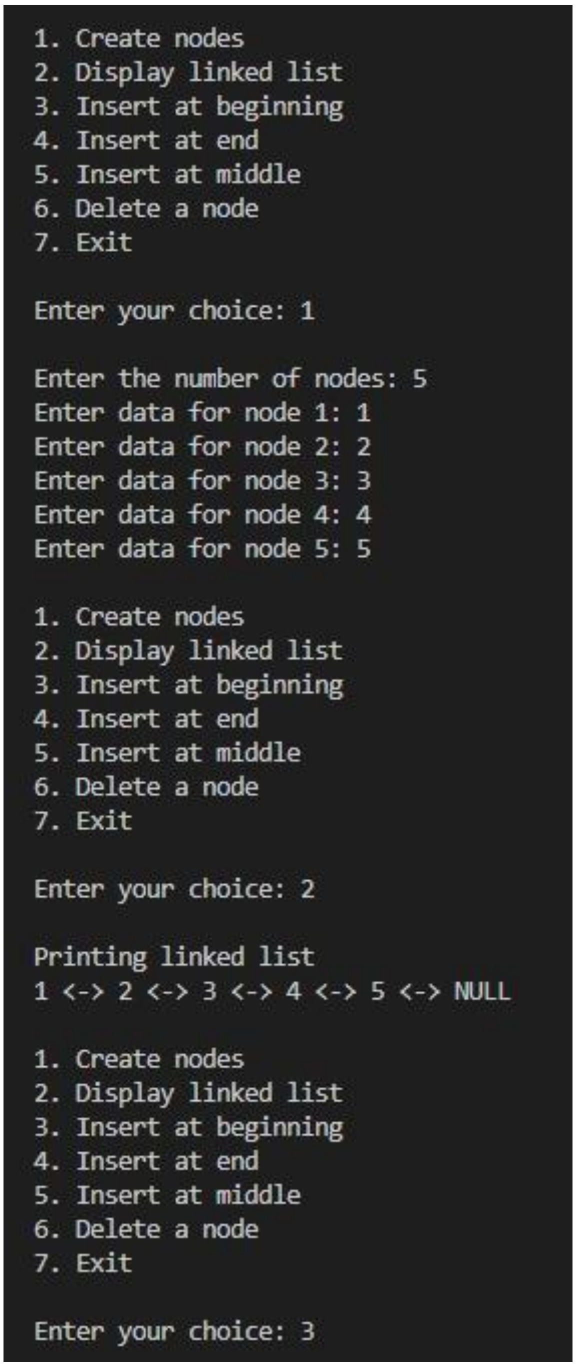
```
};

struct Node *head = NULL;
struct Node *temp = NULL;

struct Node *createnode(int d) {
   struct Node *newnode = (struct Node *)malloc(sizeof(struct Node));
   if (newnode == NULL) {
      printf("\nMemory allocation failed!\n");
      exit(1);
   }
   newnode->data = d;
   newnode->prev = NULL;
   newnode->next = NULL;
   return newnode;
}

void display(struct Node *head) {
   temp = head;
   while (temp != NULL) {
      printf("%d <-> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}

struct Node *insertbeg(struct Node *head, int d) {
   struct Node *beg = createnode(d);
   if (head == NULL) {
      head = beg;
   } else {
        beg->next = head;
        head->prev = beg;
        head = beg;
    }
    return head;
}

struct Node *insertend(struct Node *head, int d) {
    struct Node *end = createnode(d);
    if (head == NULL) {
        head = end;
    } else {
        temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = end;
        end->prev = temp;
    }
    return head;
}

struct Node *insertmiddle(struct Node *head, int d, int pos, int n)
{
       int i ;
    if (pos > n || pos < 1) {
        printf("\nInvalid position!\n");
        return head;
    }
    struct Node *mid = createnode(d);
    temp = head;
    for (i = 1; i < pos; i++) {
        temp = temp->next;
    }
    mid->next = temp->next;
    if (temp->next != NULL) {
        temp->next->prev = mid;
    }
    mid->prev = temp;
    temp->next = mid;
    return head;
}

struct Node *deletenode(struct Node *head, int pos)
{
       int i;
    if (head == NULL) {
        printf("\nList is empty, nothing to delete.\n");
        return head;
    }
    if (pos == 1) {
        temp = head;
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
        }
        free(temp);
        return head;
    }
    temp = head;
    for (i = 1; i < pos && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("\nPosition out of range.\n");
        return head;
    }
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }
    free(temp);
    return head;
}

int main()
{
    int i;
    int num = 0;
    while (1) {
        printf("\n1. Create nodes\n2. Display linked list\n3. Insert at beginning\n4. Insert at end\n5.
Insert at middle\n6. Delete a node\n7. Exit\n");
        printf("\nEnter your choice: ");
        scanf("%d", &num);

        switch (num) {
            case 1:
                printf("\nEnter the number of nodes: ");
                int n;
                scanf("%d", &n);
                if (n < 0) {
```

```
        printf("Enter a valid number\n");
        break;
    }
    for (i = 0; i < n; i++) {
        printf("Enter data for node %d: ", (i + 1));
        int data;
        scanf("%d", &data);
        struct Node *newnode = createnode(data);
        if (head == NULL) {
            head = newnode;
            temp = head;
        } else {
            temp->next = newnode;
            newnode->prev = temp;
            temp = temp->next;
        }
    }
    break;

case 2:
    printf("\nPrinting linked list\n");
    display(head);
    break;

case 3:
    printf("\nEnter data for node at beginning: ");
    int d;
    scanf("%d", &d);
    head = insertbeg(head, d);
    printf("\nAfter adding at the beginning\n");
    display(head);
    break;

case 4:
    printf("\nEnter data for node at the end: ");
    scanf("%d", &d);
    head = insertend(head, d);
    printf("\nAfter adding at the end\n");
    display(head);
    break;

case 5:
    printf("\nEnter data for node in middle and the position after which to insert: ");
    int pos;
    scanf("%d%d", &d, &pos);
    int count = 0;
    temp = head;
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    head = insertmiddle(head, d, pos, count);
    display(head);
    break;

case 6:
    printf("\nEnter the position of the node you want to delete: ");
    int pos_del;
    scanf("%d", &pos_del);
    head = deletenode(head, pos_del);
    display(head);
    break;
```

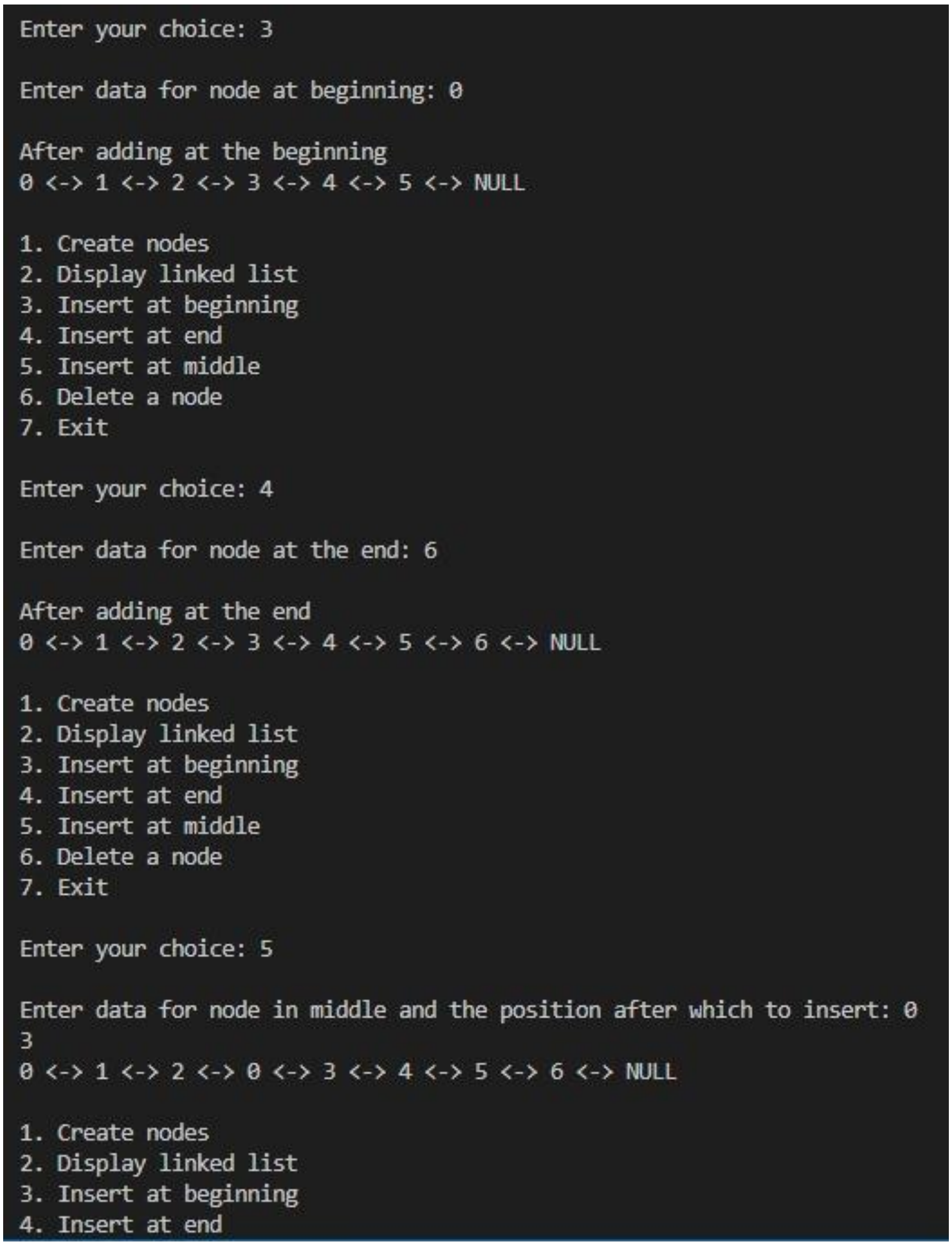
```
            case 7:
                printf("THANK YOU!\n");
                exit(0);

            default:
                printf("Enter a valid number.\n");
        }
    }
    return 0;
}
```

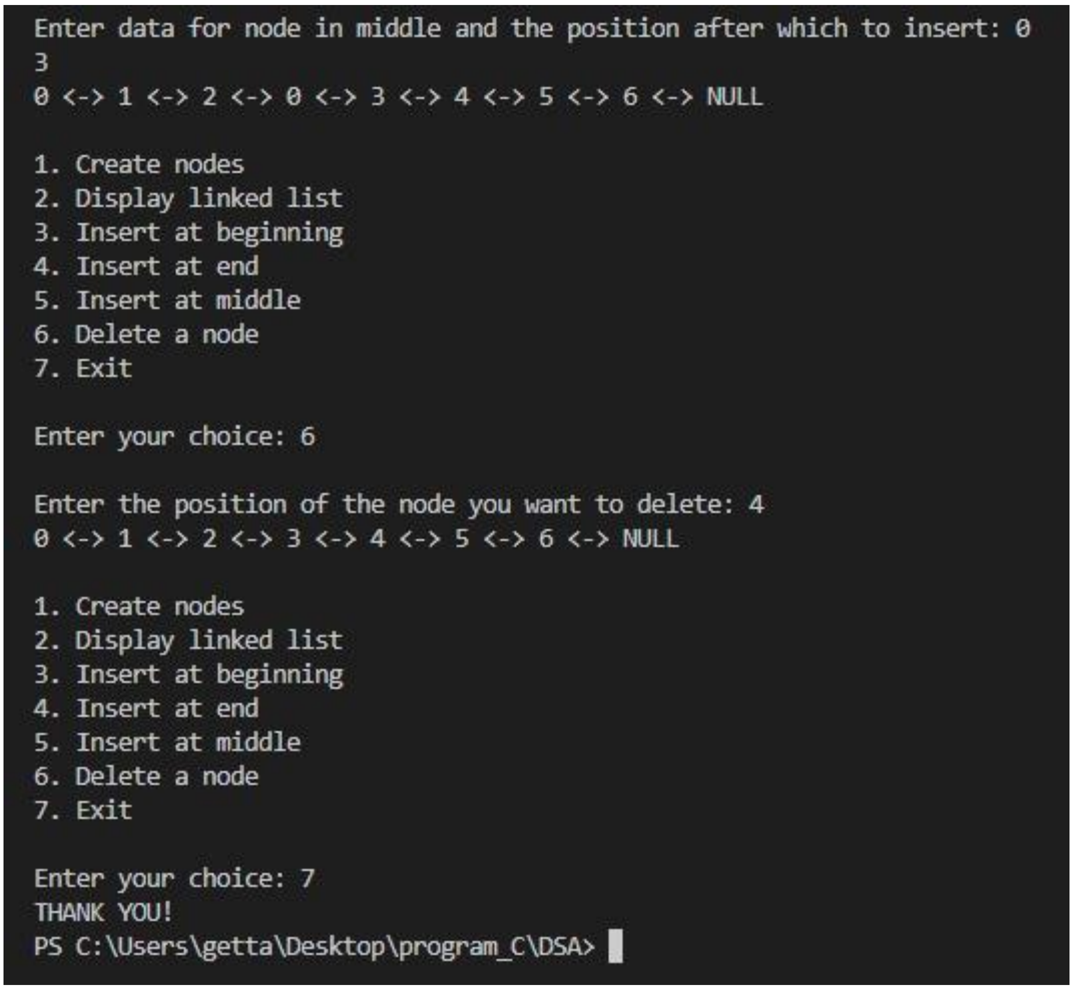
Testcases :

1). Operation on doubly linked list



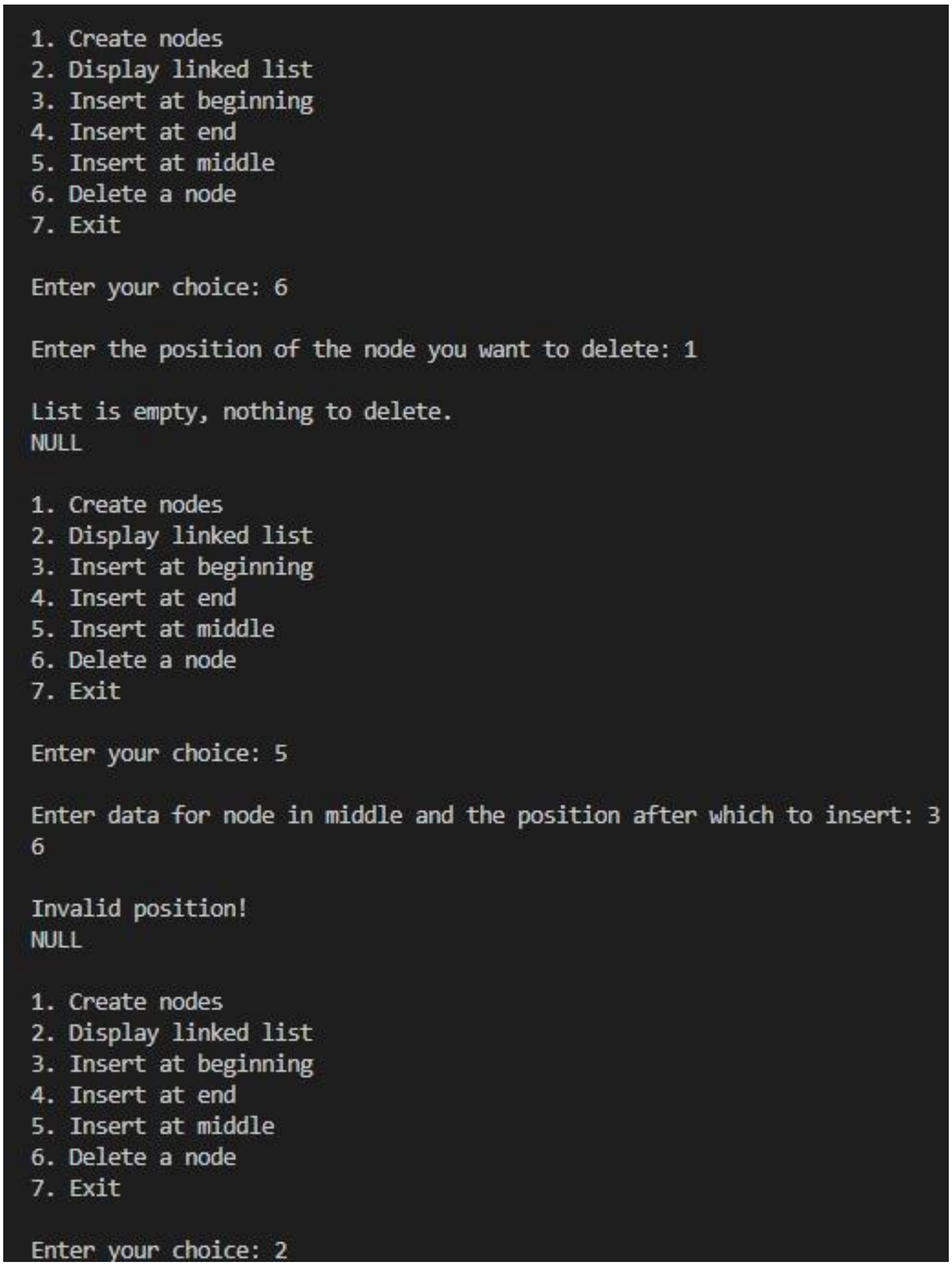
```
1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 1

Enter the number of nodes: 5
Enter data for node 1: 1
Enter data for node 2: 2
Enter data for node 3: 3
Enter data for node 4: 4
Enter data for node 5: 5

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 2

Printing linked list
1 <-> 2 <-> 3 <-> 4 <-> 5 <-> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 3
```


```
Enter your choice: 3

Enter data for node at beginning: 0

After adding at the beginning
0 <-> 1 <-> 2 <-> 3 <-> 4 <-> 5 <-> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 4

Enter data for node at the end: 6

After adding at the end
0 <-> 1 <-> 2 <-> 3 <-> 4 <-> 5 <-> 6 <-> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 5

Enter data for node in middle and the position after which to insert: 0
3
0 <-> 1 <-> 2 <-> 0 <-> 3 <-> 4 <-> 5 <-> 6 <-> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
```

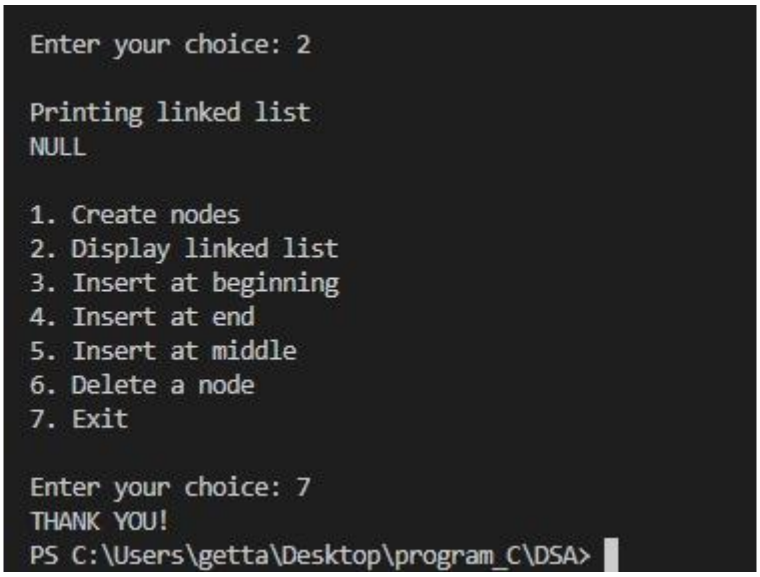


```
Enter data for node in middle and the position after which to insert: 0
3
0 <-> 1 <-> 2 <-> 0 <-> 3 <-> 4 <-> 5 <-> 6 <-> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 6

Enter the position of the node you want to delete: 4
0 <-> 1 <-> 2 <-> 3 <-> 4 <-> 5 <-> 6 <-> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 7
THANK YOU!
PS C:\Users\getta\Desktop\program_C\DSA>
```

2. Operation on empty doubly linked list-

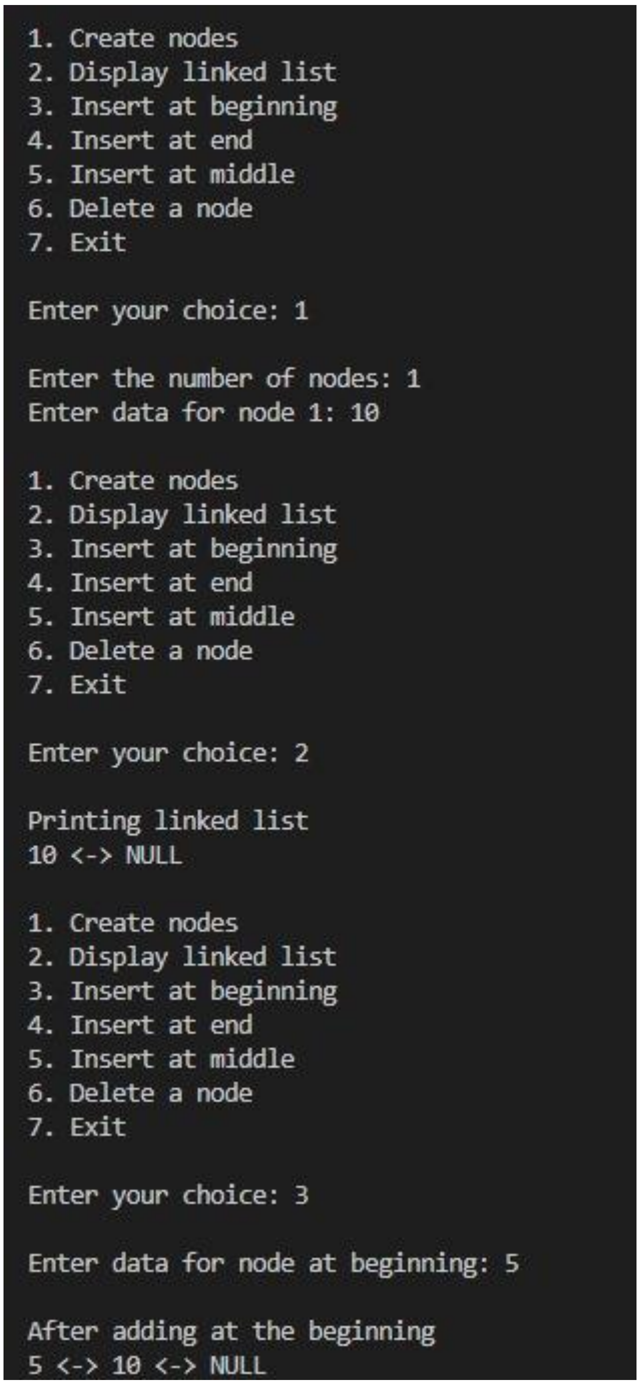


```
1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 6

Enter the position of the node you want to delete: 1

List is empty, nothing to delete.
NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 5

Enter data for node in middle and the position after which to insert: 3
6

Invalid position!
NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 2
```

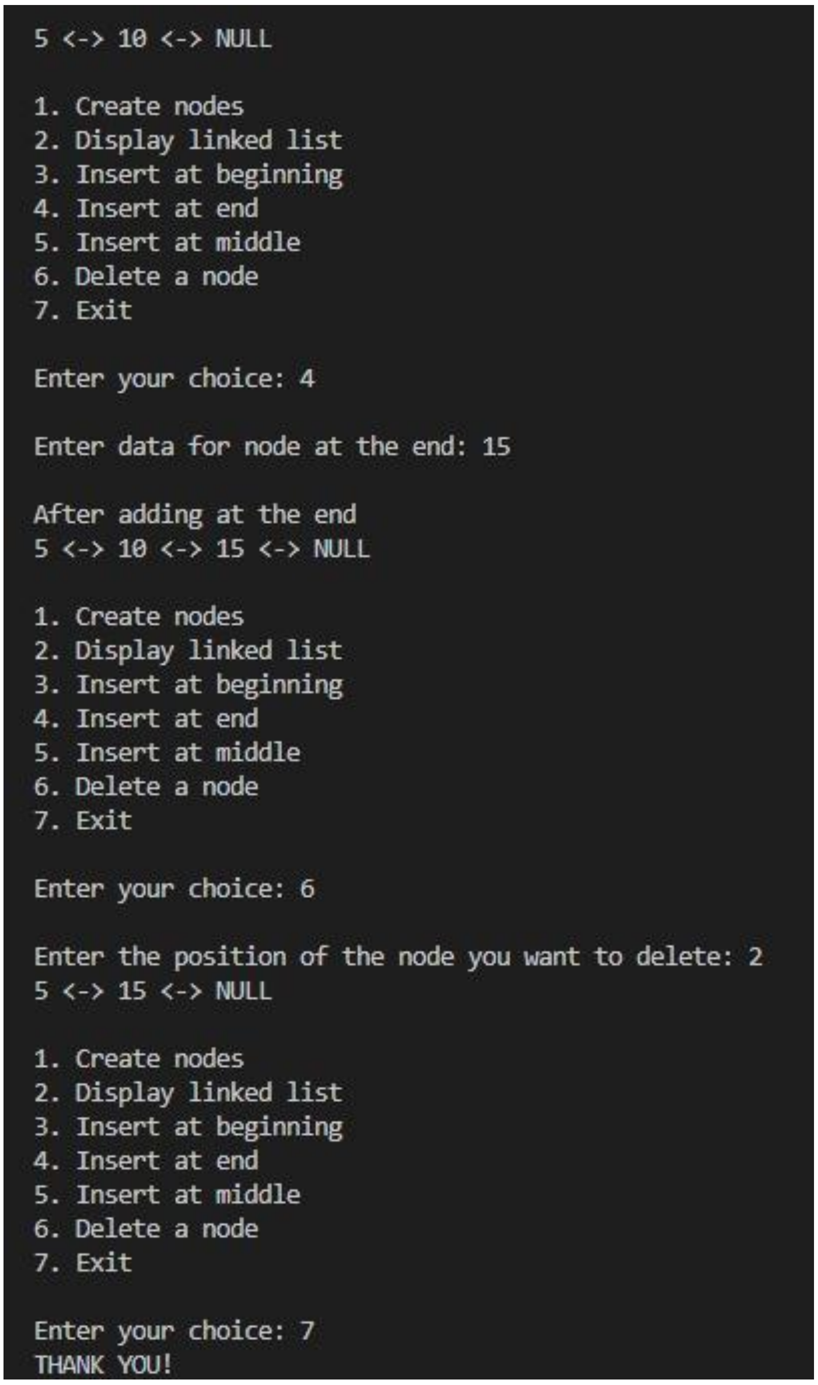


```
Enter your choice: 2

Printing linked list
NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 7
THANK YOU!
PS C:\Users\getta\Desktop\program_C\DSA>
```

3). Operations on single element doubly linked list.



```
1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 1

Enter the number of nodes: 1
Enter data for node 1: 10

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 2

Printing linked list
10 <-> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 3

Enter data for node at beginning: 5

After adding at the beginning
5 <-> 10 <-> NULL
```



```
5 <-> 10 <-> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 4

Enter data for node at the end: 15

After adding at the end
5 <-> 10 <-> 15 <-> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 6

Enter the position of the node you want to delete: 2
5 <-> 15 <-> NULL

1. Create nodes
2. Display linked list
3. Insert at beginning
4. Insert at end
5. Insert at middle
6. Delete a node
7. Exit

Enter your choice: 7
THANK YOU!
```