DIGITAL ASSIGNMENT 1

NAME: Visvanathan Kamali

REG NO: 24BBS0218

1) **PSEUDOCODE:**

```
START
  DEFINE MAX = 5
  DEFINE stack[MAX]
  SET top = -1

  FUNCTION isFull():
    IF top == MAX - 1:
      RETURN TRUE
    ELSE:
      RETURN FALSE

  FUNCTION isEmpty():
    IF top == -1:
      RETURN TRUE
    ELSE:
      RETURN FALSE

  FUNCTION PUSH(value):
    IF isFull() == TRUE:
      PRINT "Stack Overflow! Cannot push"
    ELSE:
      top = top + 1
      stack[top] = value
      PRINT value "pushed to stack"
```

```
FUNCTION POP():
  IF isEmpty() == TRUE:
    PRINT "Stack Underflow! Nothing to pop"
  ELSE:
    PRINT stack[top] "popped from stack"
    top = top - 1

FUNCTION DISPLAY():
  IF isEmpty() == TRUE:
    PRINT "Stack is empty"
  ELSE:
    PRINT "Stack elements are:"
    FOR i = top DOWNTO 0:
      PRINT stack[i]

WHILE TRUE:
  PRINT "Menu:"
  PRINT "1. PUSH"
  PRINT "2. POP"
  PRINT "3. DISPLAY"
  PRINT "4. EXIT"
  GET choice from user

  IF choice == 1:
    GET value from user
    CALL PUSH(value)

  ELSE IF choice == 2:
    CALL POP()
```

```
    ELSE IF choice == 3:
      CALL DISPLAY()

    ELSE IF choice == 4:
      PRINT "Exiting program."
      EXIT

    ELSE:
      PRINT "Invalid choice. Try again."
END
```

PROGRAM – INPUT:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int stack[MAX];
int top = -1;
int isFull()
{
   if (top == MAX - 1)
      return 1;
   return 0;
}
int isEmpty()
{
   if (top == -1)
      return 1;
   return 0;
}
void PUSH(int value)
{
   if (isFull())
   {
```

```
        printf("Stack Overflow! Cannot push %d\n", value);
    }
    else
    {
        stack[++top] = value;
        printf("%d pushed to stack\n", value);
    }
}
void POP()
{
    if (isEmpty())
    {
        printf("Stack Underflow! Nothing to pop\n");
    }
    else
    {
        printf("%d popped from stack\n", stack[top--]);
    }
}
void Display()
{
    if (isEmpty())
```

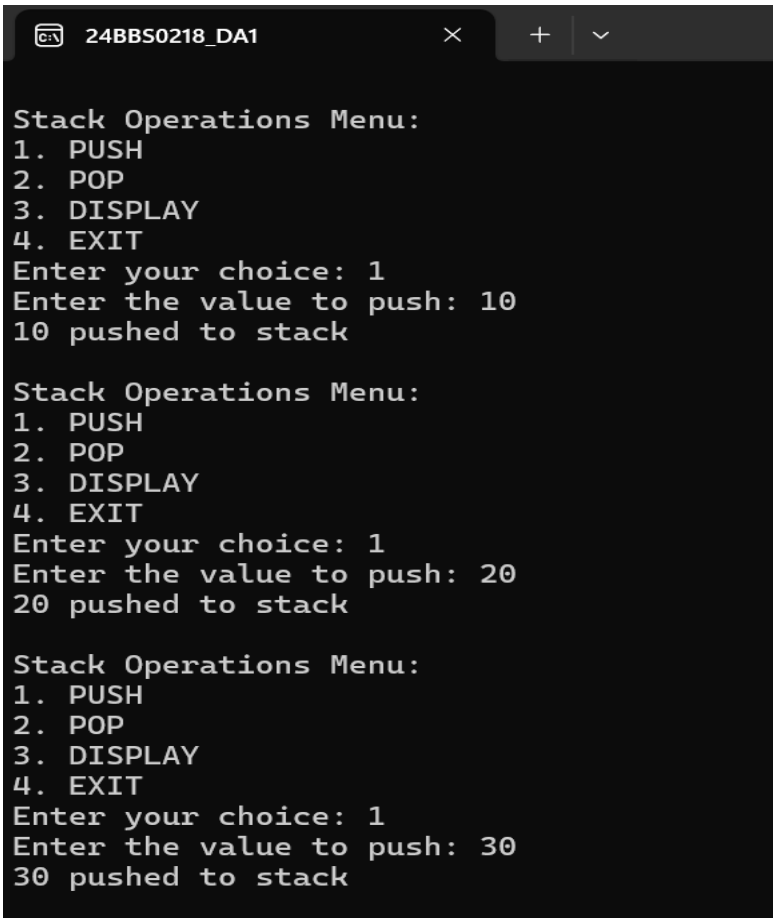
```
    {
        printf("Stack is empty!\n");
    }
    else
    {
        printf("Stack elements are: \n");
        for (int i = top; i >= 0; i--)
        {
            printf("%d\n", stack[i]);
        }
    }
}
int main()
{
    int choice, value;
    while (1)
    {
        printf("\nStack Operations Menu:\n");
        printf("1. PUSH\n");
        printf("2. POP\n");
        printf("3. DISPLAY\n");
        printf("4. EXIT\n");
```

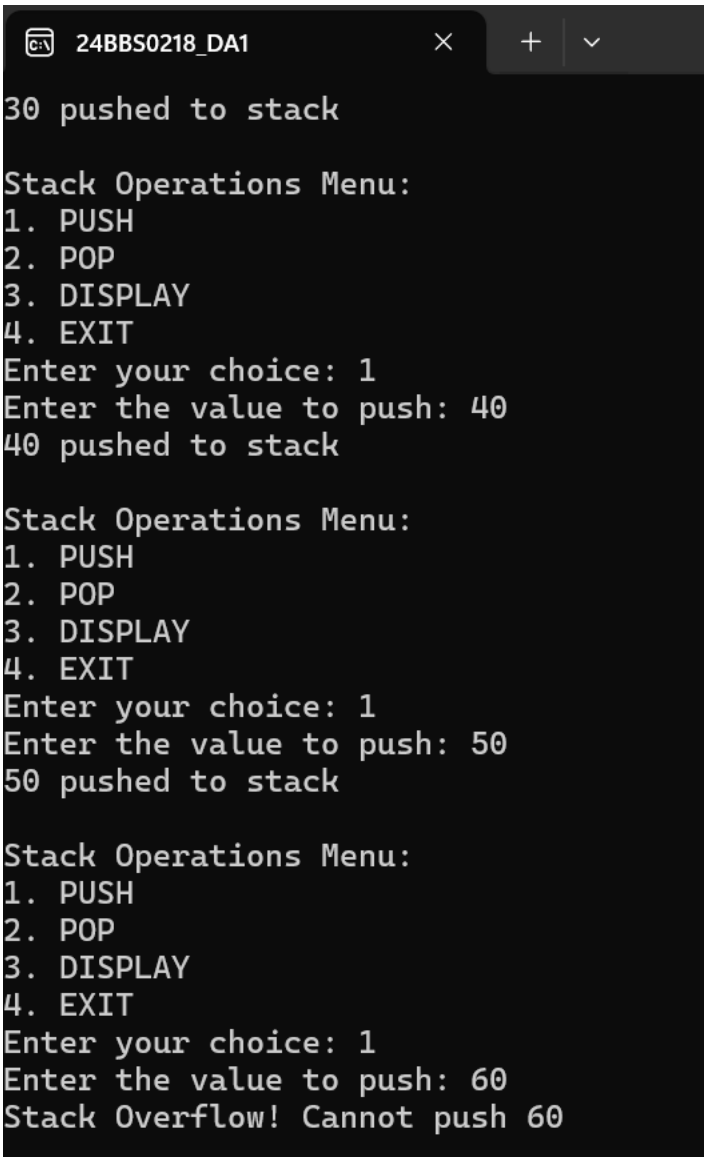
```
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice)
    {
        case 1:
            printf("Enter the value to push: ");
            scanf("%d", &value);
            PUSH(value);
            break;
        case 2:
            POP();
            break;
        case 3:
            Display();
            break;
        case 4:
            printf("Exiting program.\n");
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
    }
}
```

```
    return 0;
}
```

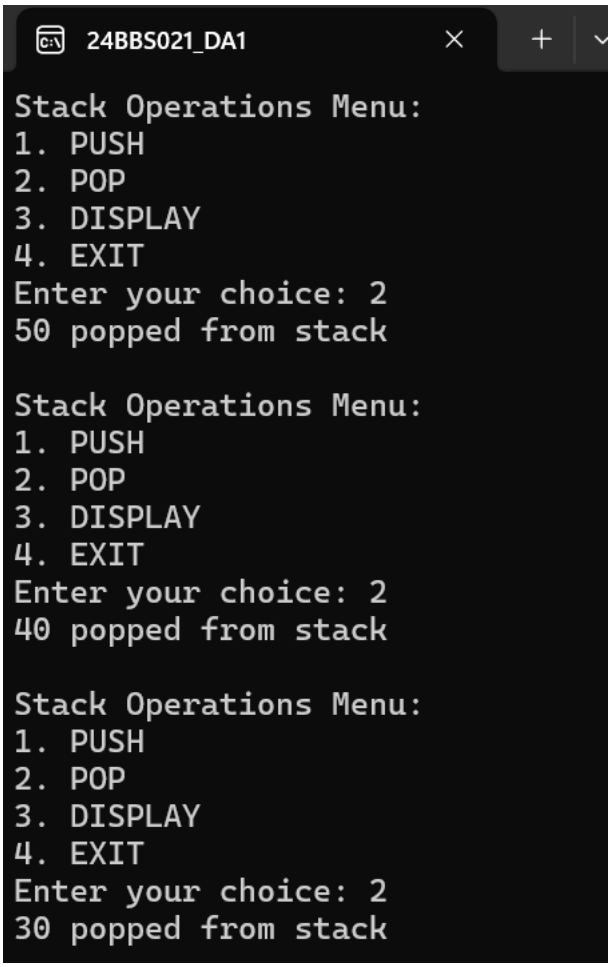
PROGRAM-OUTPUT:

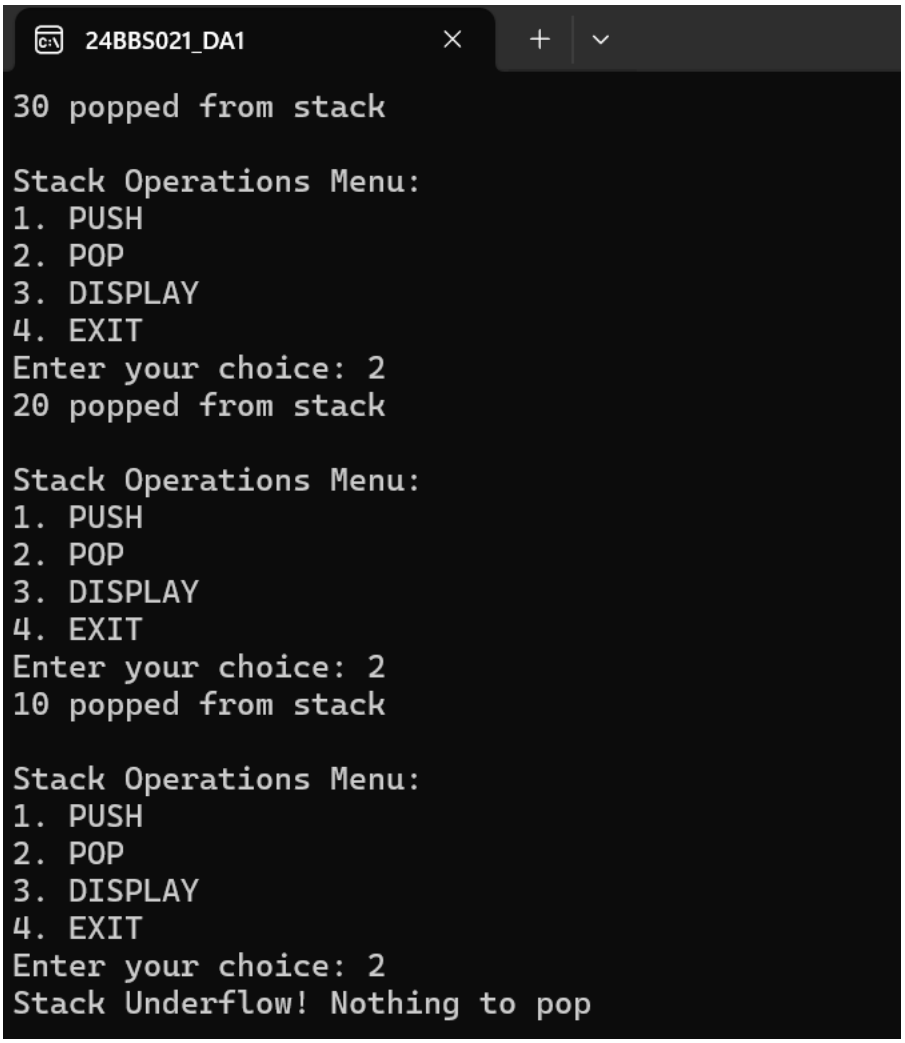
TEST CASE 1:



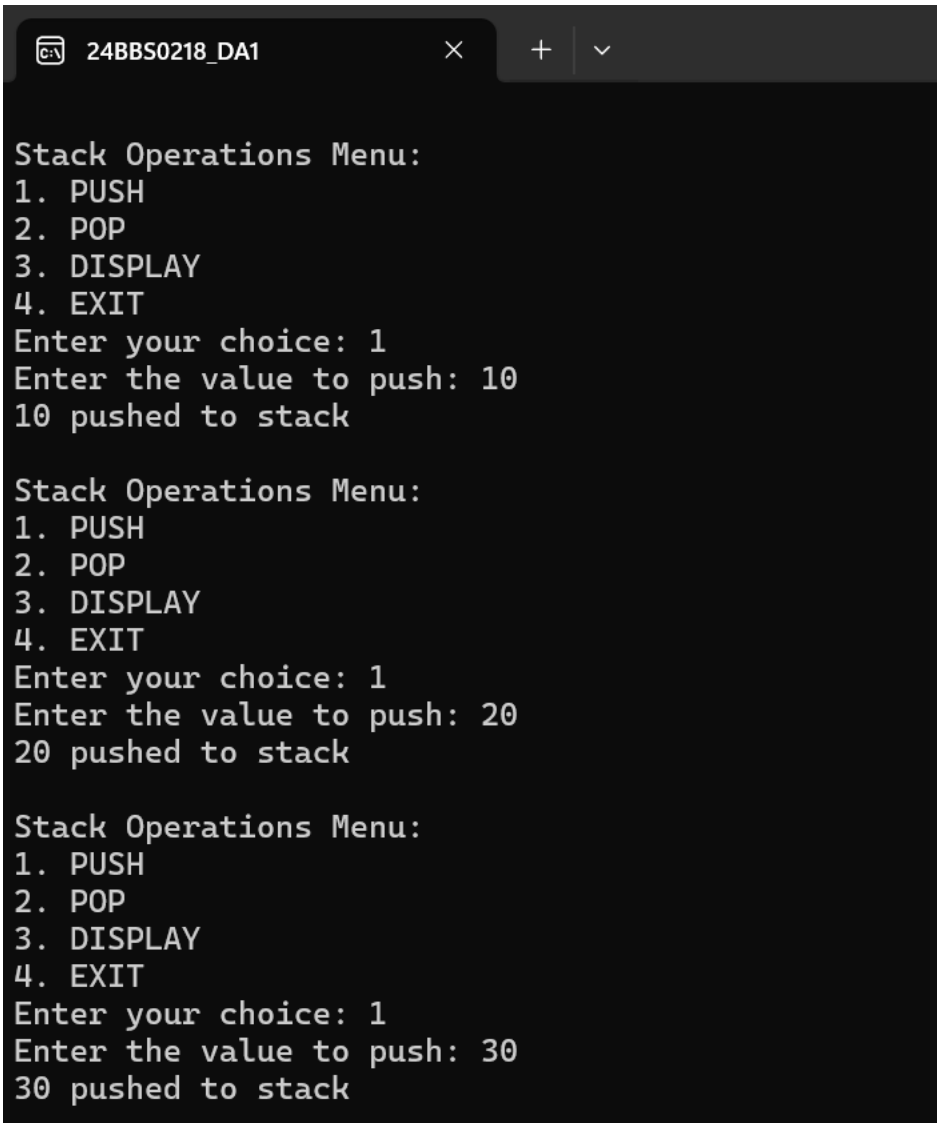

24BBS0218_DA1
30 pushed to stack
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to push: 40
40 pushed to stack
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to push: 50
50 pushed to stack
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to push: 60
Stack Overflow! Cannot push 60

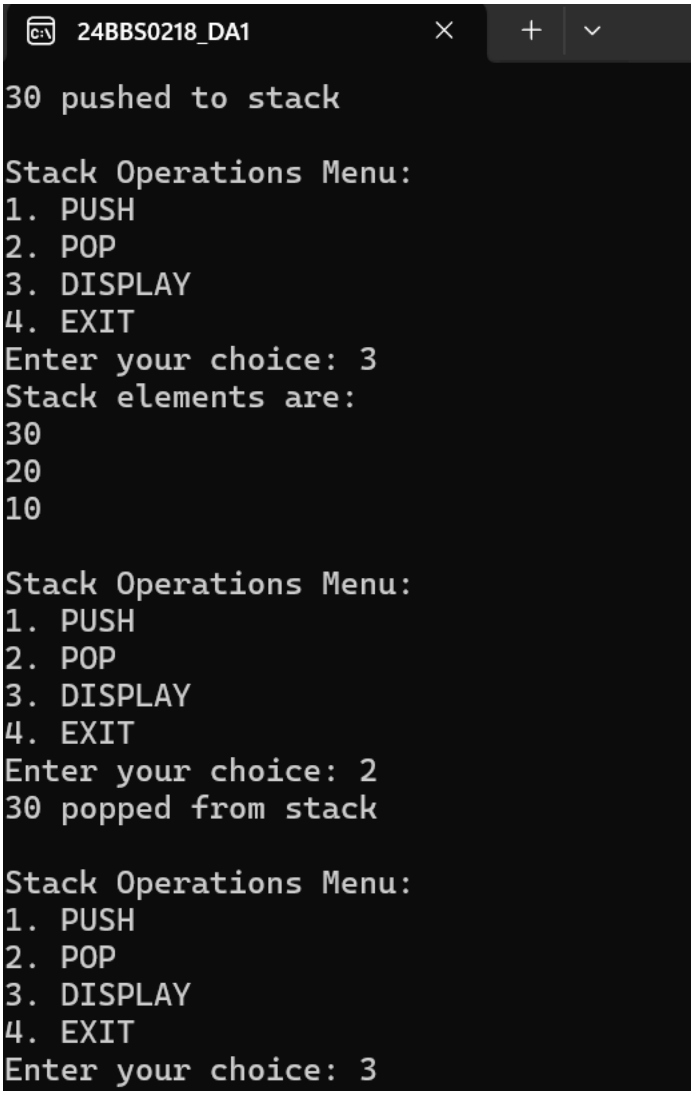
TEST CASE 2:


24BBS021_DA1
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 2
50 popped from stack
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 2
40 popped from stack
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 2
30 popped from stack


24BBS021_DA1
30 popped from stack
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 2
20 popped from stack
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 2
10 popped from stack
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 2
Stack Underflow! Nothing to pop

TEST CASE 3:


24BBS0218_DA1
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to push: 10
10 pushed to stack
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to push: 20
20 pushed to stack
Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to push: 30
30 pushed to stack




```
4. EXIT
Enter your choice: 3
Stack elements are:
20
10

Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 4
Exiting program.

Process returned 0 (0x0)   execution time : 62.883 s
Press any key to continue.
```

2) PSEUDOCODE:

```
START
  DEFINE MAX = 5
  DEFINE queue[MAX]
  SET front = -1
  SET rear = -1

  FUNCTION isFull():
    IF rear == MAX - 1:
      RETURN TRUE
    ELSE:
      RETURN FALSE

  FUNCTION isEmpty():
    IF front == -1:
      RETURN TRUE
    ELSE:
      RETURN FALSE

  FUNCTION Enqueue(value):
    IF isFull() == TRUE:
      PRINT "Queue Overflow! Cannot enqueue"
    ELSE:
      IF front == -1:
        front = 0
      rear = rear + 1
      queue[rear] = value
      PRINT value "enqueued to queue"

  FUNCTION Dequeue():
```

```
  IF isEmpty() == TRUE:
    PRINT "Queue Underflow! Nothing to dequeue"
  ELSE:
    PRINT queue[front] "dequeued from queue"
    IF front == rear:
      front = -1
      rear = -1
    ELSE:
      front = front + 1

FUNCTION Display():
  IF isEmpty() == TRUE:
    PRINT "Queue is empty"
  ELSE:
    PRINT "Queue elements are:"
    FOR i = front TO rear:
      PRINT queue[i]

WHILE TRUE:
  PRINT "Menu:"
  PRINT "1. Enqueue"
  PRINT "2. Dequeue"
  PRINT "3. Display"
  PRINT "4. Exit"
  GET choice from user

  IF choice == 1:
    GET value from user
    CALL Enqueue(value)
```

```
    ELSE IF choice == 2:
      CALL Dequeue()

    ELSE IF choice == 3:
      CALL Display()

    ELSE IF choice == 4:
      PRINT "Exiting program."
      EXIT

    ELSE:
      PRINT "Invalid choice. Try again."
END
```

PROGRAM – INPUT:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int queue[MAX];
int front = -1, rear = -1;
int isFull()
{
   return (rear == MAX - 1);
}
int isEmpty()
{
   return (front == -1);
}
void Enqueue(int value)
{
   if (isFull())
   {
      printf("Queue Overflow! Cannot enqueue %d\n", value);
   }
   else
   {
      if (front == -1)
      {
         front = 0;
      }
      rear++;
      queue[rear] = value;
      printf("%d enqueued to queue\n", value);
   }
```

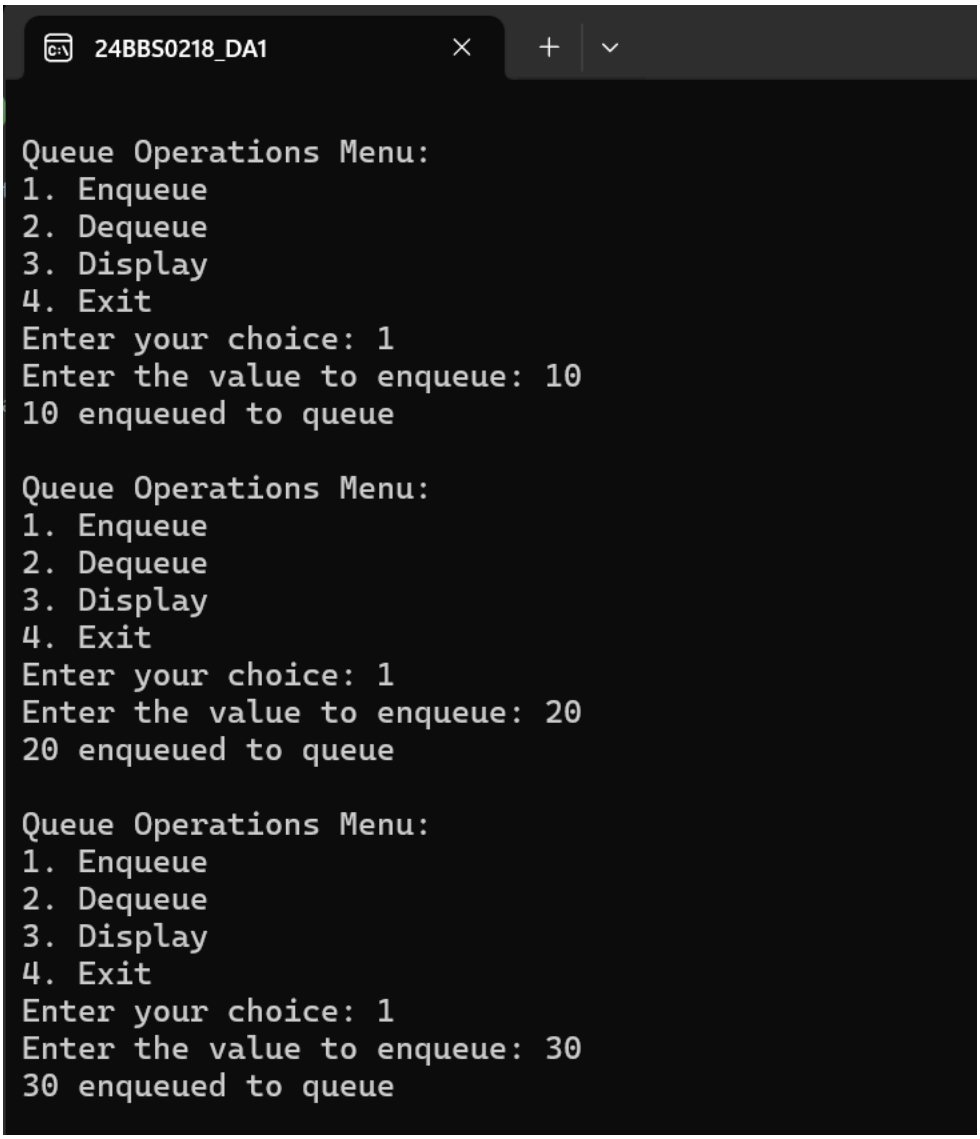
```
}
void Dequeue()
{
   if (isEmpty())
   {
      printf("Queue Underflow! Nothing to dequeue\n");
   }
   else
   {
      printf("%d dequeued from queue\n", queue[front]);
      if (front == rear)
      {
         front = rear = -1;
      }
      else
      {
         front++;
      }
   }
}
void Display()
{
   if (isEmpty())
   {
      printf("Queue is empty!\n");
   }
   else
   {
      printf("Queue elements are: \n");
      for (int i = front; i <= rear; i++)
```

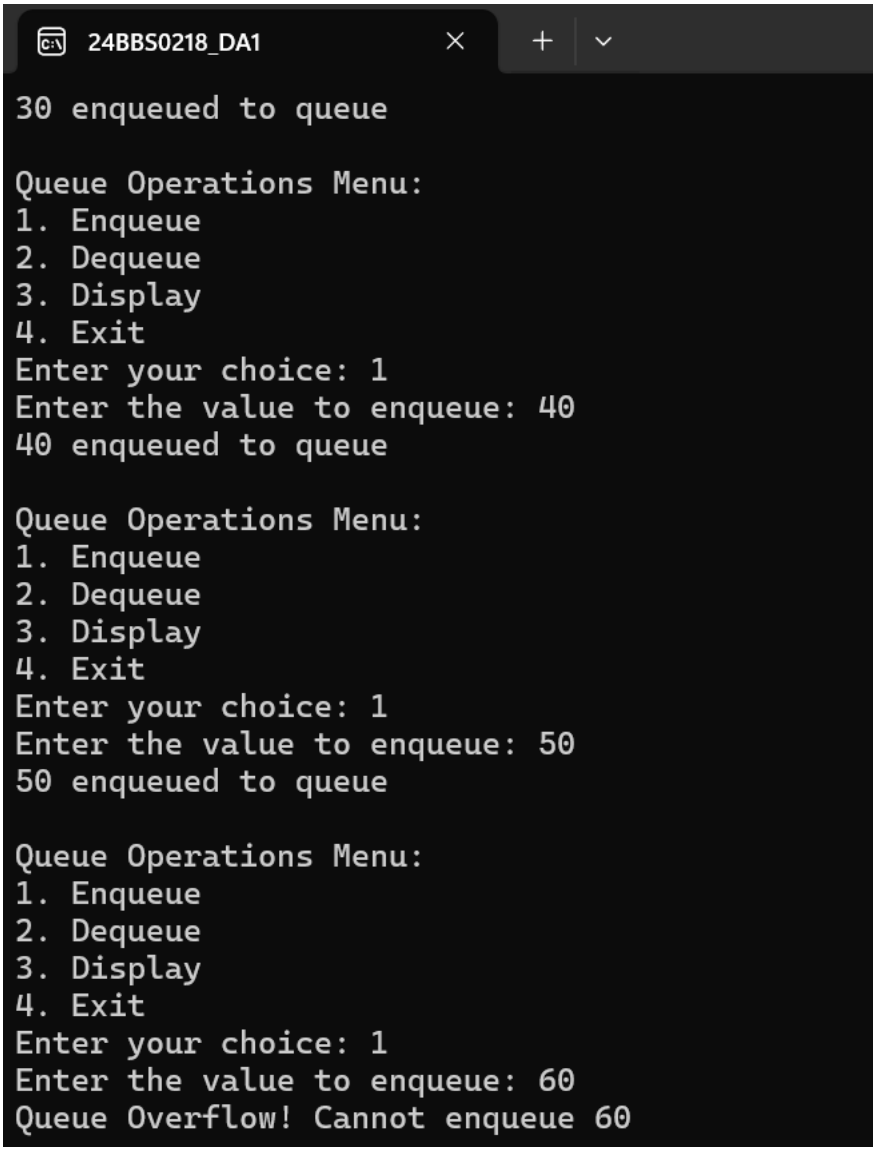
```
        {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}
int main()
{
    int choice, value;
    while (1)
    {
        printf("\nQueue Operations Menu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("Enter the value to enqueue: ");
                scanf("%d", &value);
                Enqueue(value);
                break;
            case 2:
                Dequeue();
                break;
            case 3:
                Display();
```

```
            break;
        case 4:
            printf("Exiting program.\n");
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }
   return 0;
}
```

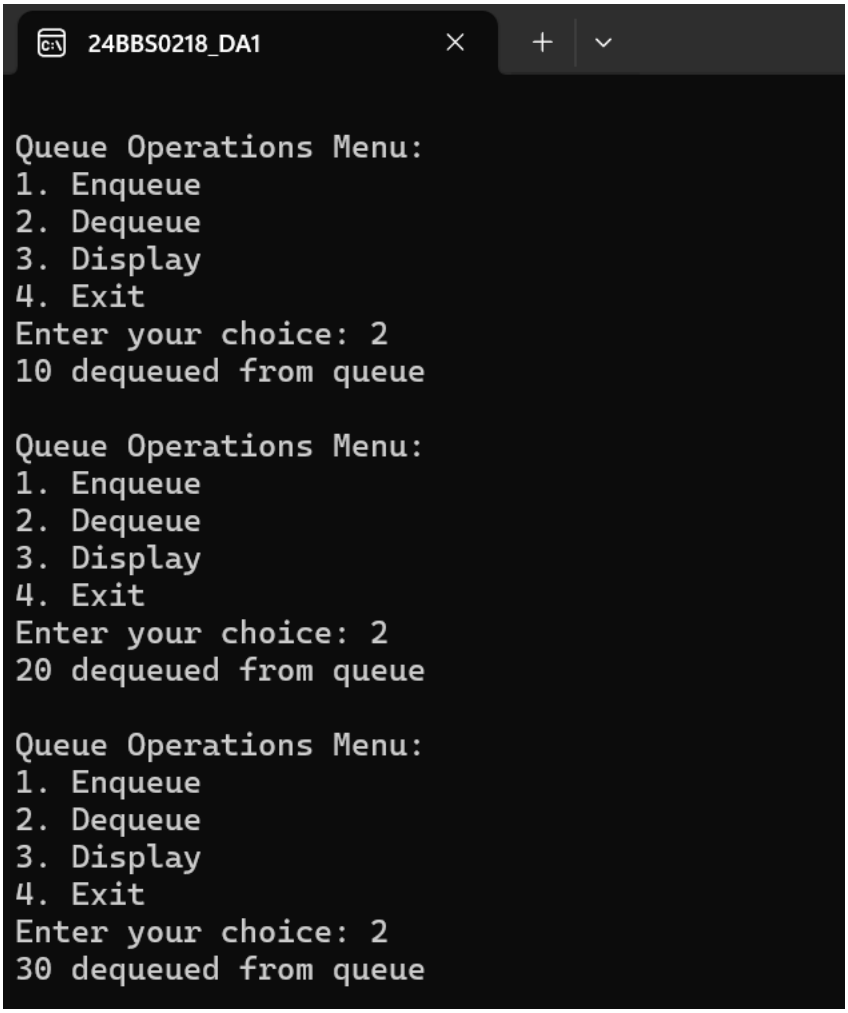
PROGRAM – OUTPUT:

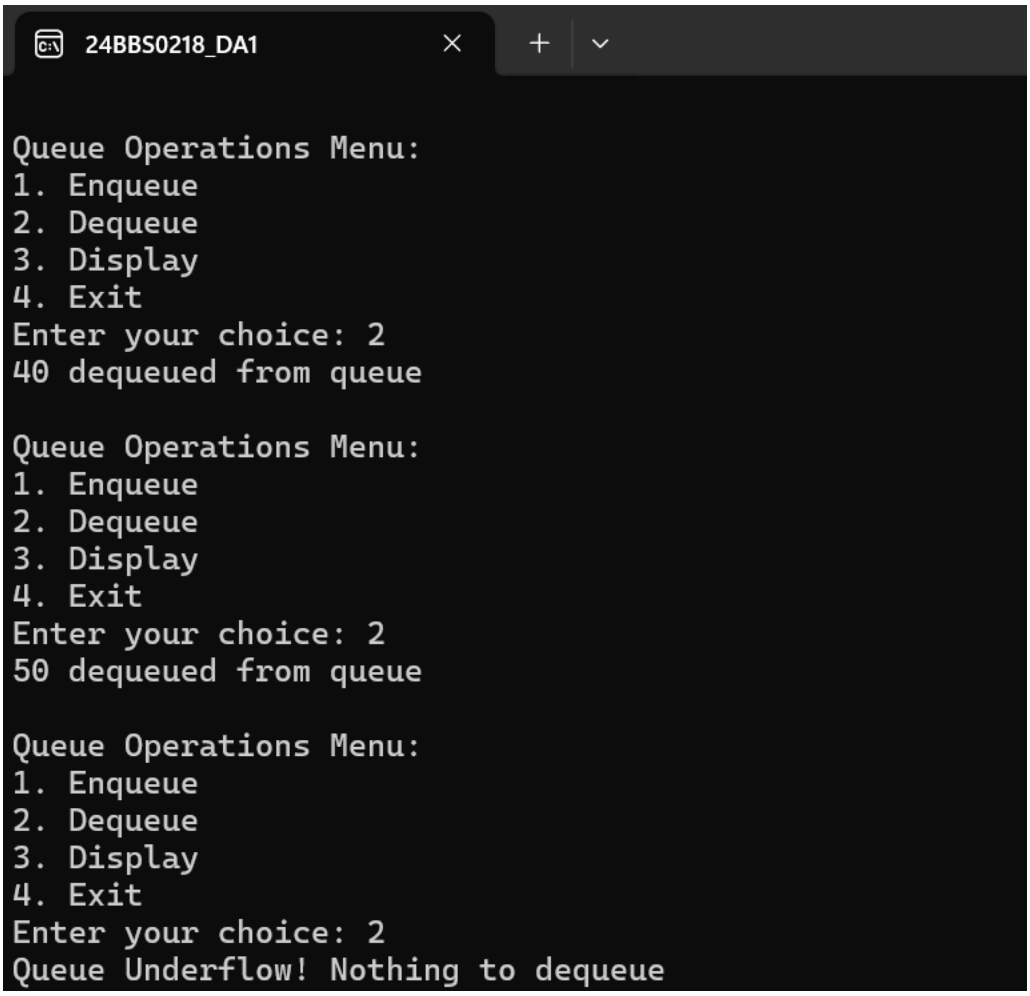
TEST CASE 1:



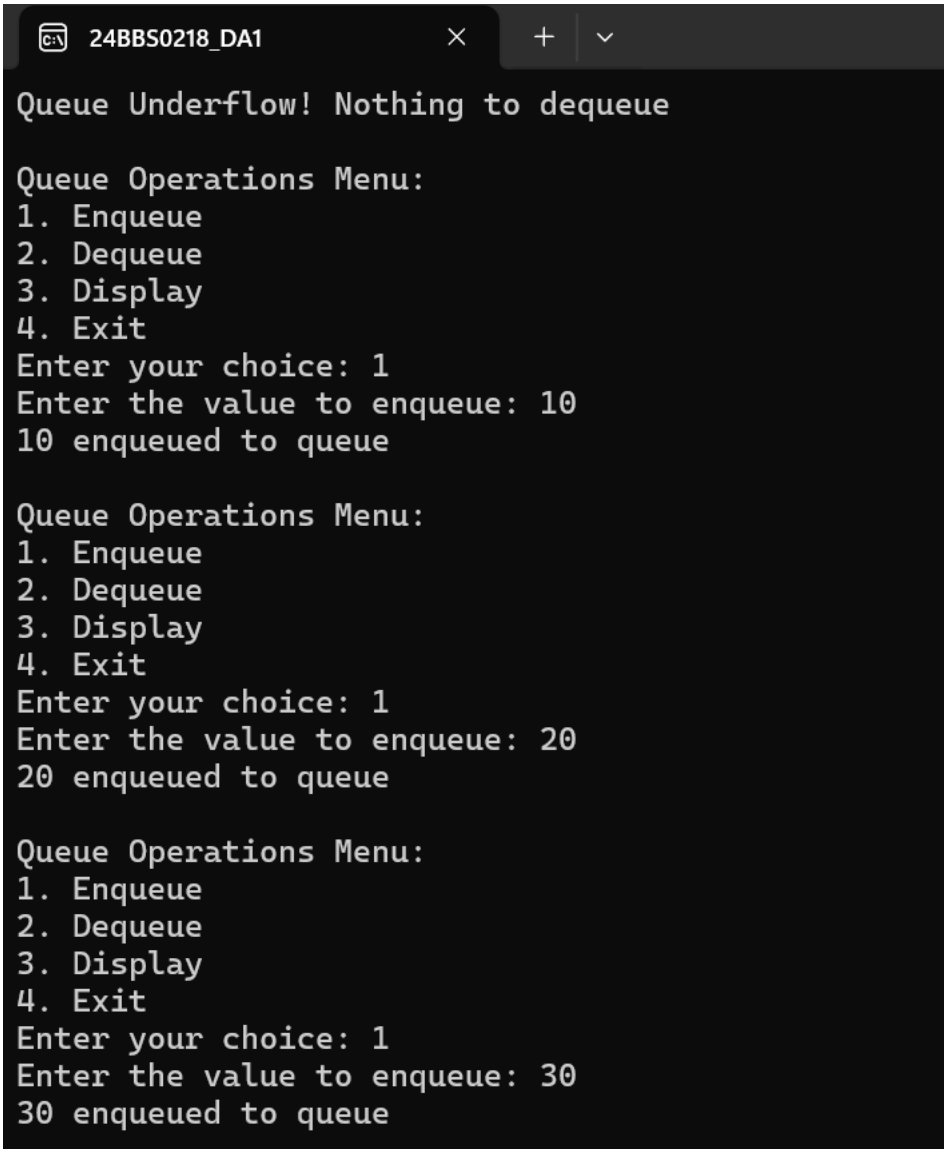

24BBS0218_DA1
30 enqueued to queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 40
40 enqueued to queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 50
50 enqueued to queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 60
Queue Overflow! Cannot enqueue 60

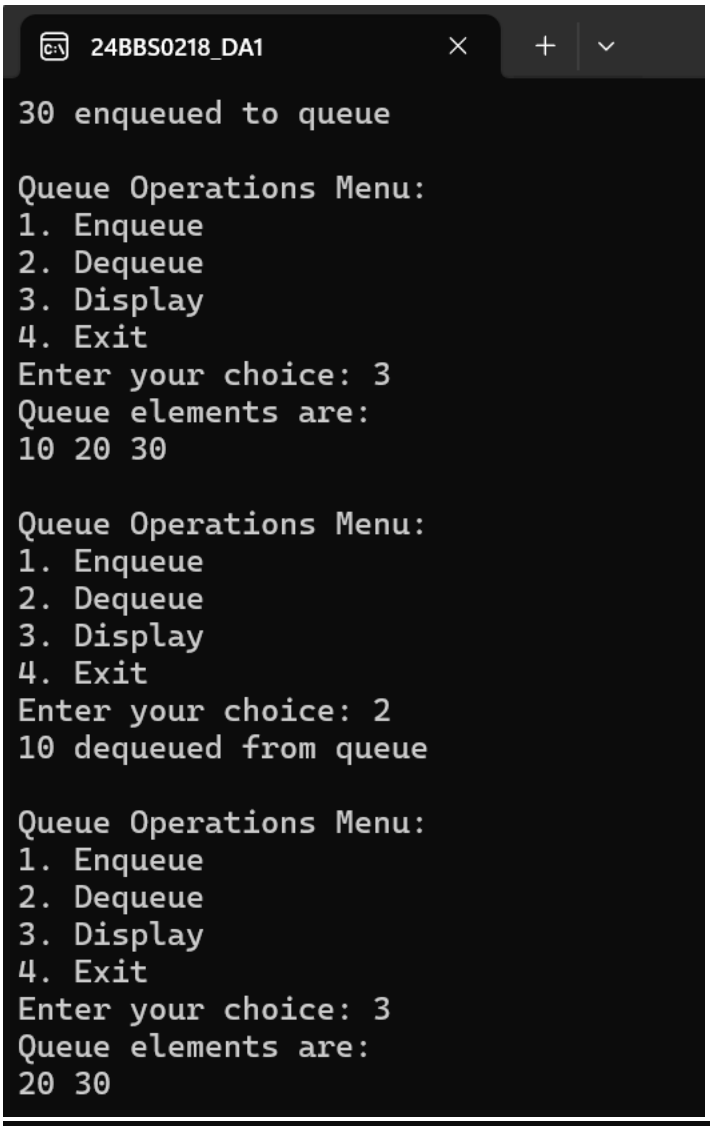
TEST CASE 2:

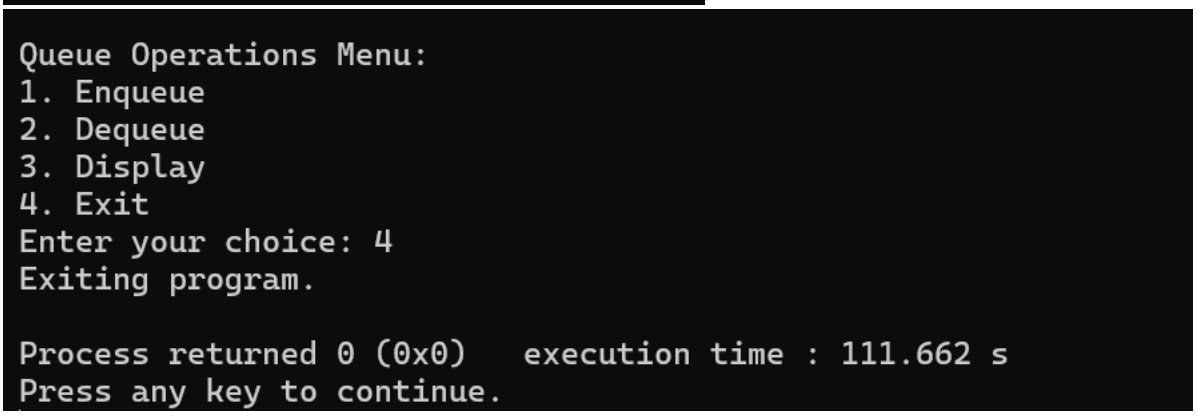

24BBS0218_DA1
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
10 dequeued from queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
20 dequeued from queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
30 dequeued from queue


24BBS0218_DA1
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
40 dequeued from queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
50 dequeued from queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow! Nothing to dequeue

TEST CASE 3:


24BBS0218_DA1
Queue Underflow! Nothing to dequeue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 10
10 enqueued to queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 20
20 enqueued to queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 30
30 enqueued to queue


24BBS0218_DA1
30 enqueued to queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are:
10 20 30
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
10 dequeued from queue
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are:
20 30


Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting program.
Process returned 0 (0x0) execution time : 111.662 s
Press any key to continue.

3) PSEUDOCODE:

```
START
  DEFINE MAX = 5
  DEFINE queue[MAX]
  SET front = -1
  SET rear = -1

  FUNCTION isFull():
    IF (rear + 1) % MAX == front:
      RETURN TRUE
    ELSE:
      RETURN FALSE

  FUNCTION isEmpty():
    IF front == -1:
      RETURN TRUE
    ELSE:
      RETURN FALSE

  FUNCTION Enqueue(value):
    IF isFull() == TRUE:
      PRINT "Queue Overflow! Cannot enqueue."
    ELSE:
      IF front == -1:
        front = 0  // Set front to 0 if queue is empty
      rear = (rear + 1) % MAX
      queue[rear] = value
      PRINT value "enqueued to queue"

  FUNCTION Dequeue():
```

```
  IF isEmpty() == TRUE:
    PRINT "Queue Underflow! Nothing to dequeue."
  ELSE:
    PRINT queue[front] "dequeued from queue"
    IF front == rear:
      front = rear = -1  // Reset to -1 when queue is empty
    ELSE:
      front = (front + 1) % MAX

FUNCTION Display():
  IF isEmpty() == TRUE:
    PRINT "Queue is empty."
  ELSE:
    PRINT "Queue elements are:"
    IF front <= rear:
      FOR i = front TO rear:
        PRINT queue[i]
    ELSE:
      FOR i = front TO MAX - 1:
        PRINT queue[i]
      FOR i = 0 TO rear:
        PRINT queue[i]

WHILE TRUE:
  PRINT "Menu:"
  PRINT "1. Enqueue"
  PRINT "2. Dequeue"
  PRINT "3. Display"
  PRINT "4. Exit"
  GET choice from user
```

```
    IF choice == 1:
      GET value from user
      CALL Enqueue(value)

    ELSE IF choice == 2:
      CALL Dequeue()

    ELSE IF choice == 3:
      CALL Display()

    ELSE IF choice == 4:
      PRINT "Exiting program."
      EXIT

    ELSE:
      PRINT "Invalid choice. Try again."
END
```

PROGRAM – INPUT:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int queue[MAX];
int front = -1, rear = -1;
int isFull()
{
   return ((rear + 1) % MAX == front);
}
int isEmpty()
{
   return (front == -1);
}
void Enqueue(int value)
{
   if (isFull())
   {
      printf("Queue Overflow! Cannot enqueue %d\n", value);
   }
   else
   {
      if (front == -1)
      {
         front = 0;
      }
      rear = (rear + 1) % MAX;
      queue[rear] = value;
      printf("%d enqueued to queue\n", value);
   }
```

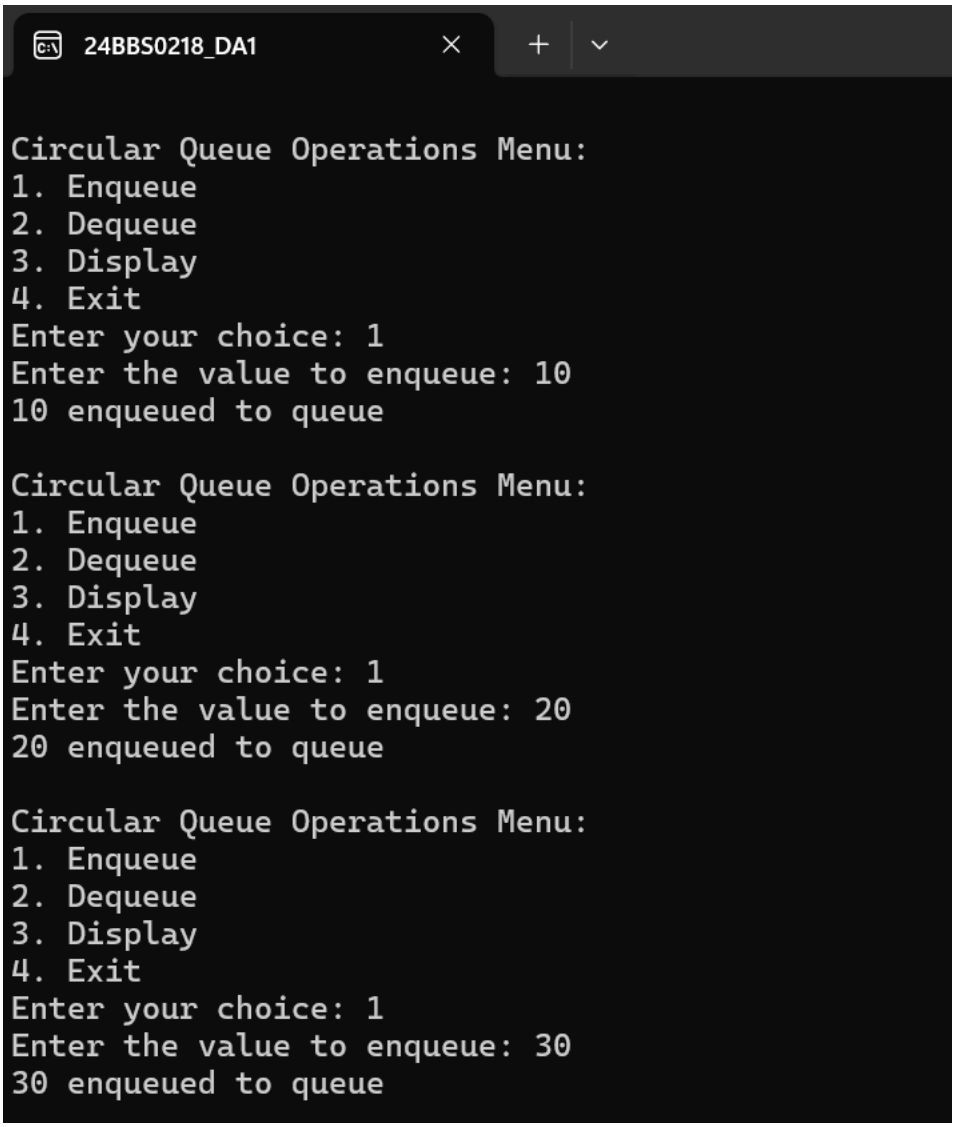
```
}
void Dequeue()
{
   if (isEmpty())
   {
      printf("Queue Underflow! Nothing to dequeue\n");
   }
   else
   {
      printf("%d dequeued from queue\n", queue[front]);
      if (front == rear) {
         front = rear = -1;
      }
      else
      {
         front = (front + 1) % MAX;
      }
   }
}
void Display()
{
   if (isEmpty())
   {
      printf("Queue is empty!\n");
   }
   else
   {
      printf("Queue elements are: \n");
      int i = front;
      while (i != rear)
```

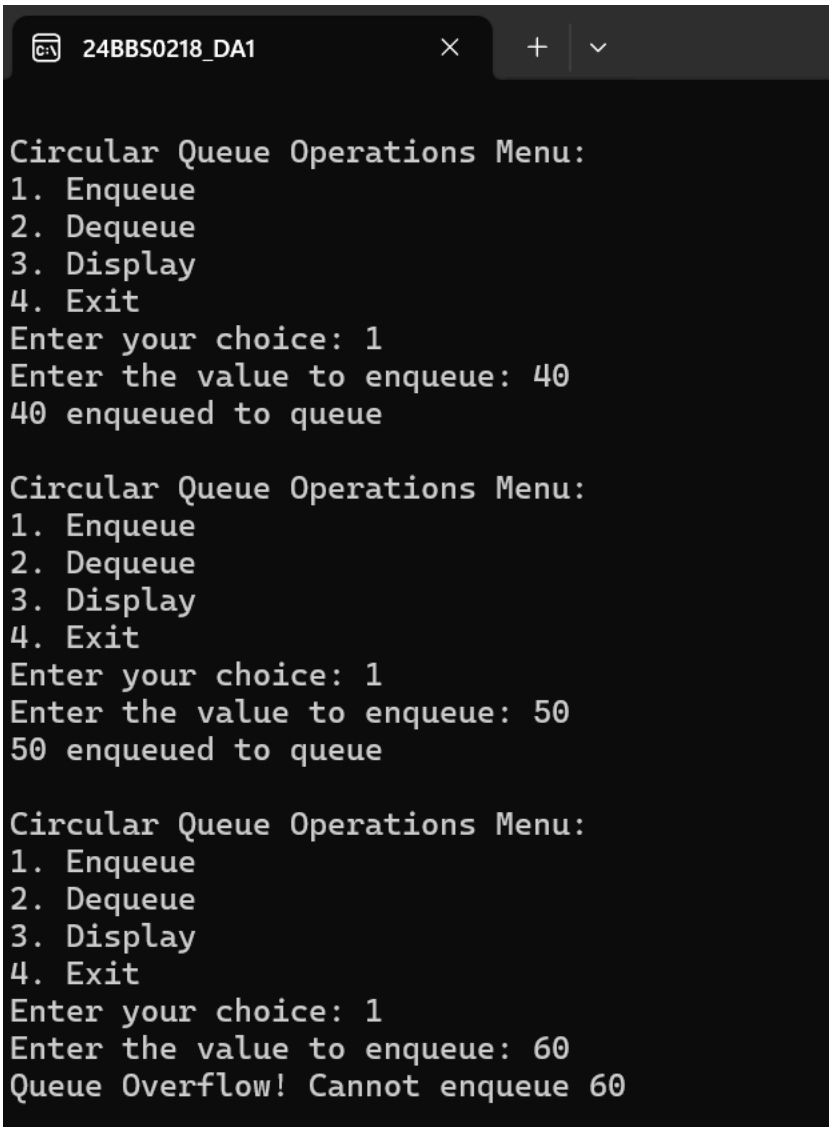
```
        {
            printf("%d ", queue[i]);
            i = (i + 1) % MAX;
        }
        printf("%d\n", queue[rear]);
    }
}
int main()
{
    int choice, value;
    while (1)
    {
        printf("\nCircular Queue Operations Menu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("Enter the value to enqueue: ");
                scanf("%d", &value);
                Enqueue(value);
                break;
            case 2:
                Dequeue();
                break;
            case 3:
```

```
            Display();
            break;
        case 4:
            printf("Exiting program.\n");
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }
   return 0;
}
```

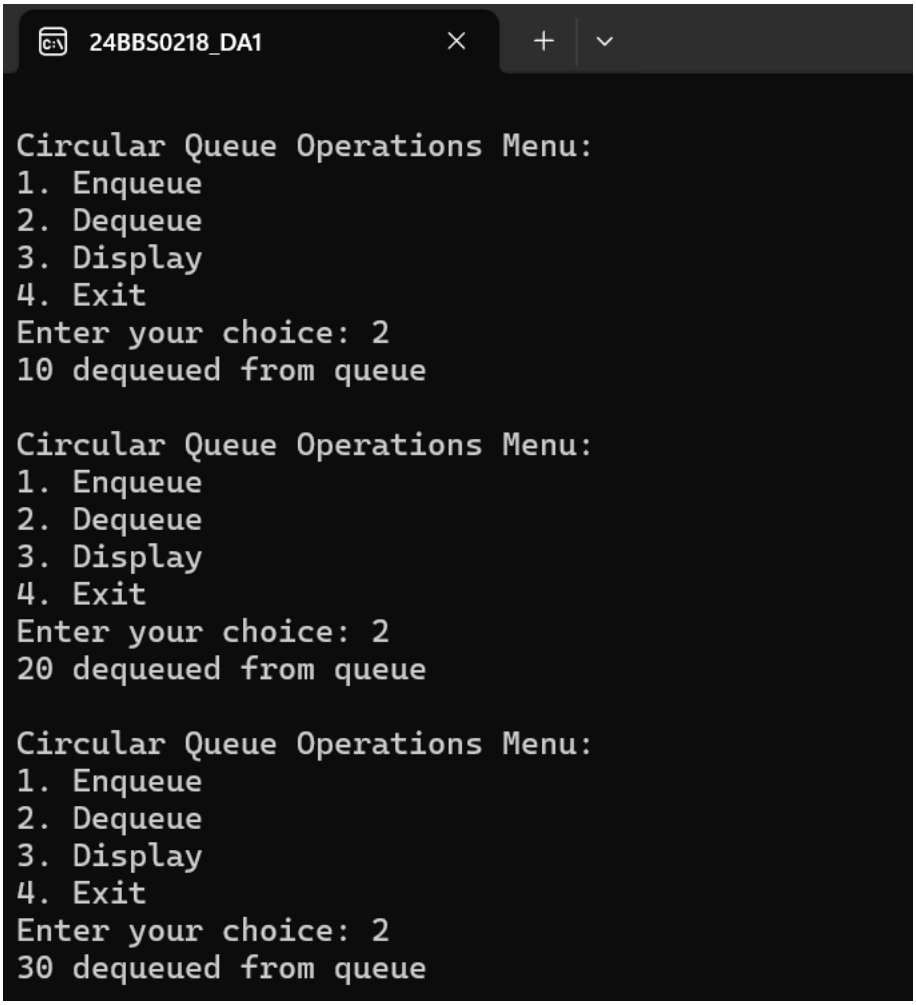
PROGRAM – OUTPUT:

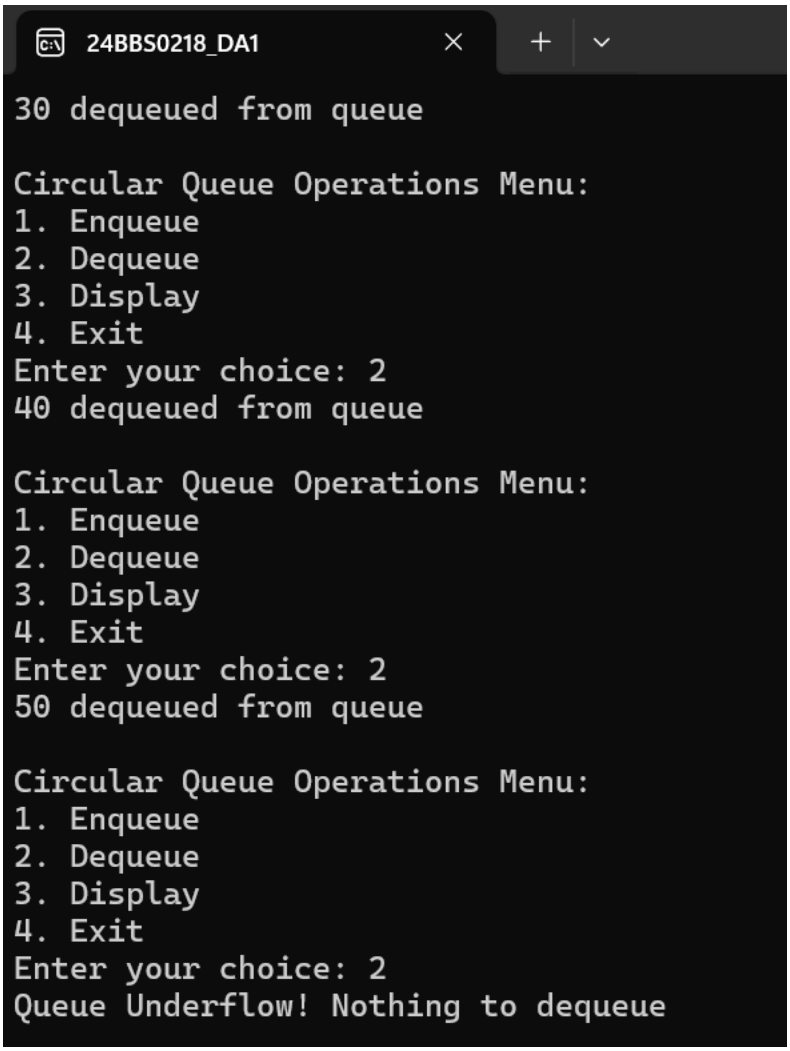
TEST CASE 1:



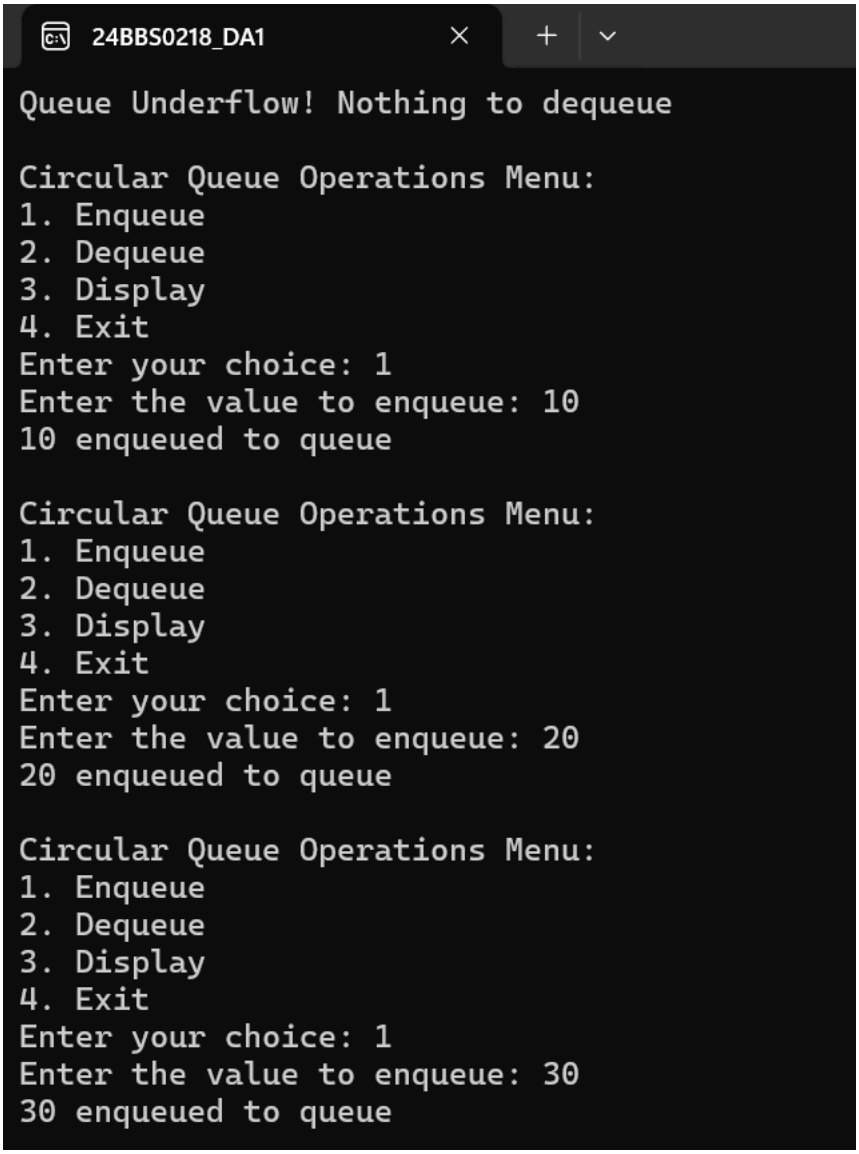

24BBS0218_DA1
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 40
40 enqueued to queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 50
50 enqueued to queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 60
Queue Overflow! Cannot enqueue 60

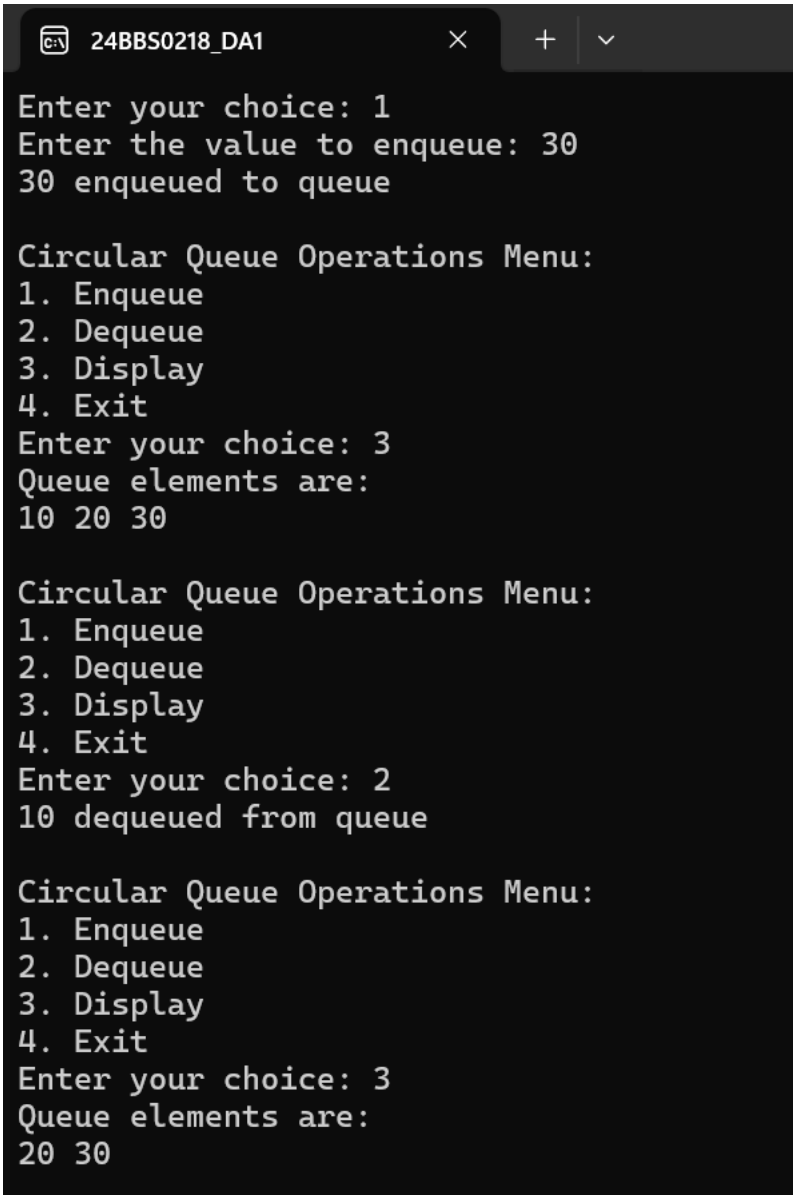
TEST CASE 2:

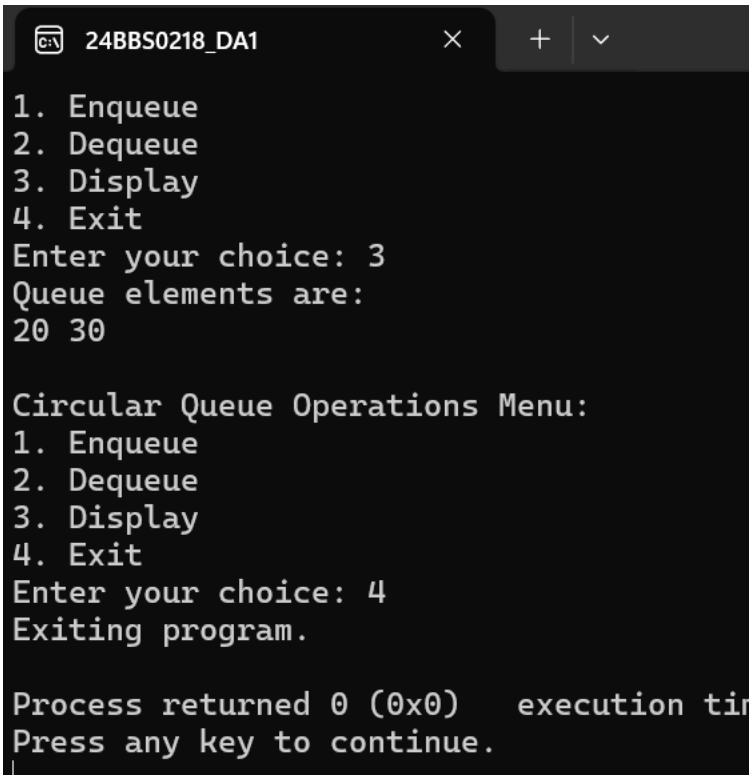

24BBS0218_DA1
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
10 dequeued from queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
20 dequeued from queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
30 dequeued from queue


24BBS0218_DA1
30 dequeued from queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
40 dequeued from queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
50 dequeued from queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow! Nothing to dequeue

TEST CASE 3:


24BBS0218_DA1
Queue Underflow! Nothing to dequeue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 10
10 enqueued to queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 20
20 enqueued to queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 30
30 enqueued to queue


24BBS0218_DA1
Enter your choice: 1
Enter the value to enqueue: 30
30 enqueued to queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are:
10 20 30
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
10 dequeued from queue
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are:
20 30


24BBS0218_DA1
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are:
20 30
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting program.
Process returned 0 (0x0)   execution tir
Press any key to continue.

4) <u>PSEUDOCODE:</u>

```
START
   DEFINE Node with fields: data, next
   DEFINE head as NULL

   FUNCTION InsertAtBeginning(data):
      CREATE newNode
      SET newNode.data = data
      SET newNode.next = head
      SET head = newNode

   FUNCTION InsertAtEnd(data):
      CREATE newNode
      SET newNode.data = data
      SET newNode.next = NULL

      IF head == NULL:
         SET head = newNode
      ELSE:
         SET current = head
         WHILE current.next != NULL:
            SET current = current.next
         SET current.next = newNode

   FUNCTION InsertAtPosition(data, position):
      CREATE newNode
      SET newNode.data = data
      SET current = head
      SET count = 1
```

```
    IF position == 1:
      CALL InsertAtBeginning(data)
    ELSE:
      WHILE current != NULL AND count < position-1:
        SET current = current.next
        SET count = count + 1

      IF current == NULL:
        PRINT "Position is out of range"
      ELSE:
        SET newNode.next = current.next
        SET current.next = newNode

  FUNCTION DeleteAtBeginning():
    IF head == NULL:
      PRINT "List is empty!"
    ELSE:
      SET head = head.next

  FUNCTION DeleteAtEnd():
    IF head == NULL:
      PRINT "List is empty!"
    ELSE:
      SET current = head
      WHILE current.next != NULL AND current.next.next !=
NULL:
        SET current = current.next
      SET current.next = NULL

  FUNCTION DeleteAtPosition(position):
```

```
    IF head == NULL:
        PRINT "List is empty!"
    ELSE:
        SET current = head
        SET count = 1

        IF position == 1:
            CALL DeleteAtBeginning()
        ELSE:
            WHILE current != NULL AND count < position-1:
                SET current = current.next
                SET count = count + 1

            IF current == NULL OR current.next == NULL:
                PRINT "Position is out of range"
            ELSE:
                SET current.next = current.next.next

FUNCTION Search(value):
    SET current = head
    SET position = 1
    WHILE current != NULL:
        IF current.data == value:
            PRINT "Element found at position", position
            RETURN
        SET current = current.next
        SET position = position + 1
    PRINT "Element not found in the list"

FUNCTION Display():
```

```
    IF head == NULL:
        PRINT "List is empty"
    ELSE:
        SET current = head
        WHILE current != NULL:
            PRINT current.data
            SET current = current.next

FUNCTION MainMenu():
    PRINT "Menu:"
    PRINT "1. Insert at Beginning"
    PRINT "2. Insert at End"
    PRINT "3. Insert at Position"
    PRINT "4. Delete at Beginning"
    PRINT "5. Delete at End"
    PRINT "6. Delete at Position"
    PRINT "7. Search for Element"
    PRINT "8. Display List"
    PRINT "9. Exit"
    GET user choice
    IF choice == 1:
        GET data
        CALL InsertAtBeginning(data)
    ELSE IF choice == 2:
        GET data
        CALL InsertAtEnd(data)
    ELSE IF choice == 3:
        GET data, position
        CALL InsertAtPosition(data, position)
    ELSE IF choice == 4:
```

```
        CALL DeleteAtBeginning()
    ELSE IF choice == 5:
        CALL DeleteAtEnd()
    ELSE IF choice == 6:
        GET position
        CALL DeleteAtPosition(position)
    ELSE IF choice == 7:
        GET value
        CALL Search(value)
    ELSE IF choice == 8:
        CALL Display()
    ELSE IF choice == 9:
        EXIT

END
```

PROGRAM – INPUT:

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node* next;
};
struct Node* head = NULL;
void InsertAtBeginning(int data)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
   newNode->data = data;
   newNode->next = head;
   head = newNode;
   printf("%d inserted at the beginning.\n", data);
}
void InsertAtEnd(int data)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
   struct Node* temp = head;
   newNode->data = data;
   newNode->next = NULL;
   if (head == NULL)
   {
      head = newNode;
   }
   else
```

```
    {
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("%d inserted at the end.\n", data);
}
void InsertAtPosition(int data, int position)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    struct Node* temp = head;
    newNode->data = data;
    if (position == 1)
    {
        newNode->next = head;
        head = newNode;
        printf("%d inserted at position %d.\n", data, position);
    }
    else
    {
        for (int i = 1; i < position - 1 && temp != NULL; i++)
        {
            temp = temp->next;
        }
        if (temp == NULL)
        {
            printf("Position out of range.\n");
```

```
        }
        else
        {
            newNode->next = temp->next;
            temp->next = newNode;
            printf("%d inserted at position %d.\n", data, position);
        }
    }
}
void DeleteAtBeginning()
{
    if (head == NULL)
    {
        printf("List is empty.\n");
    }
    else
    {
        struct Node* temp = head;
        head = head->next;
        free(temp);
        printf("Node deleted from the beginning.\n");
    }
}
void DeleteAtEnd()
{
    if (head == NULL)
    {
        printf("List is empty.\n");
    }
    else
```

```
    {
        struct Node* temp = head;
        while (temp->next != NULL && temp->next->next != NULL)
        {
            temp = temp->next;
        }
        free(temp->next);
        temp->next = NULL;
        printf("Node deleted from the end.\n");
    }
}
void DeleteAtPosition(int position)
{
    if (head == NULL)
    {
        printf("List is empty.\n");
    }
    else
    {
        struct Node* temp = head;
        if (position == 1)
        {
            head = head->next;
            free(temp);
            printf("Node deleted from position %d.\n", position);
        }
        else
        {
            for (int i = 1; i < position - 1 && temp != NULL; i++)
            {
```

```
            temp = temp->next;
        }
        if (temp == NULL || temp->next == NULL)
        {
            printf("Position out of range.\n");
        }
        else
        {
            struct Node* nodeToDelete = temp->next;
            temp->next = temp->next->next;
            free(nodeToDelete);
            printf("Node deleted from position %d.\n", position);
        }
    }
  }
}
void Search(int value)
{
  struct Node* temp = head;
  int position = 1;
  while (temp != NULL)
  {
    if (temp->data == value)
    {
      printf("Element %d found at position %d.\n", value,
position);
      return;
    }
    temp = temp->next;
    position++;
```

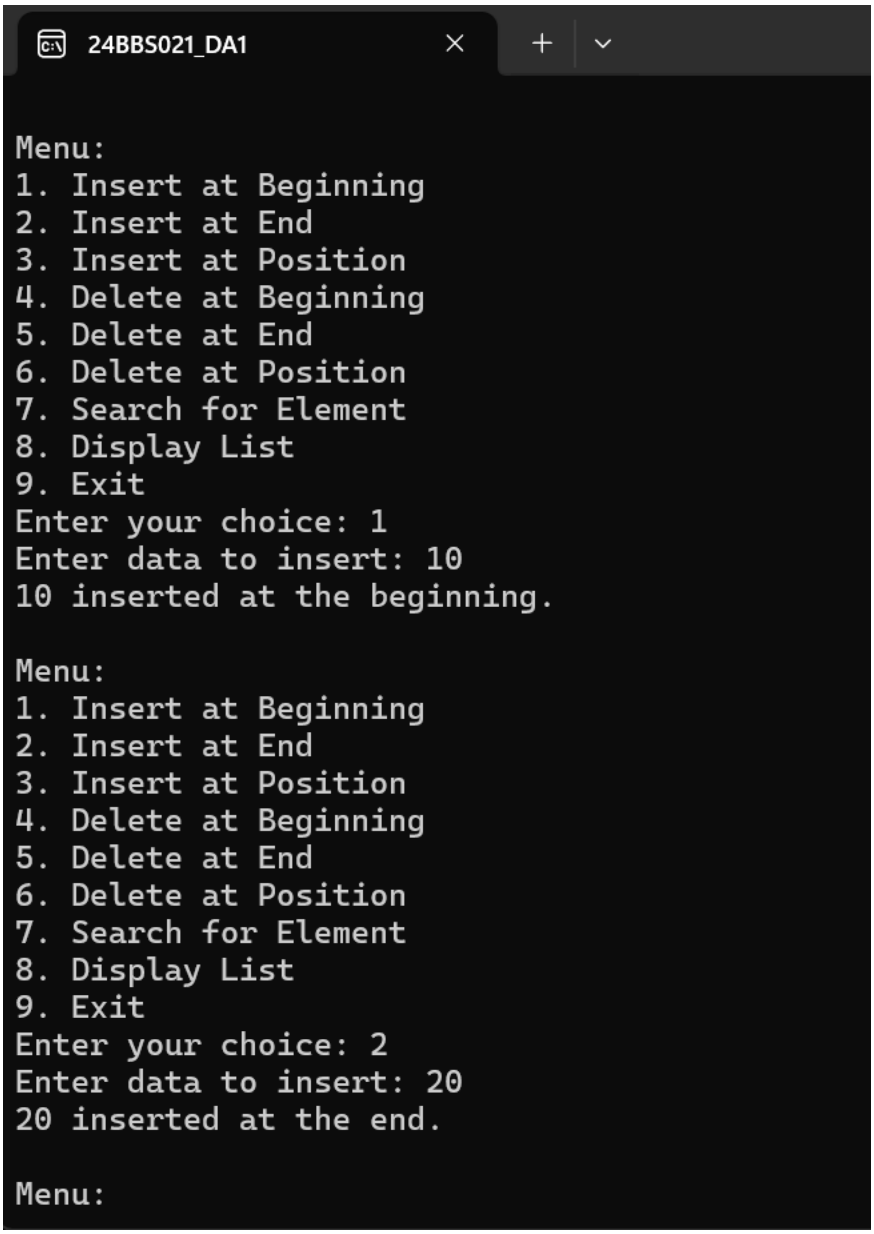
```
    }
    printf("Element %d not found.\n", value);
}
void Display()
{
    if (head == NULL)
    {
        printf("List is empty.\n");
    }
    else
    {
        struct Node* temp = head;
        while (temp != NULL)
        {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
int main()
{
    int choice, data, position;
    while (1)
    {
        printf("\nMenu:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete at Beginning\n");
```

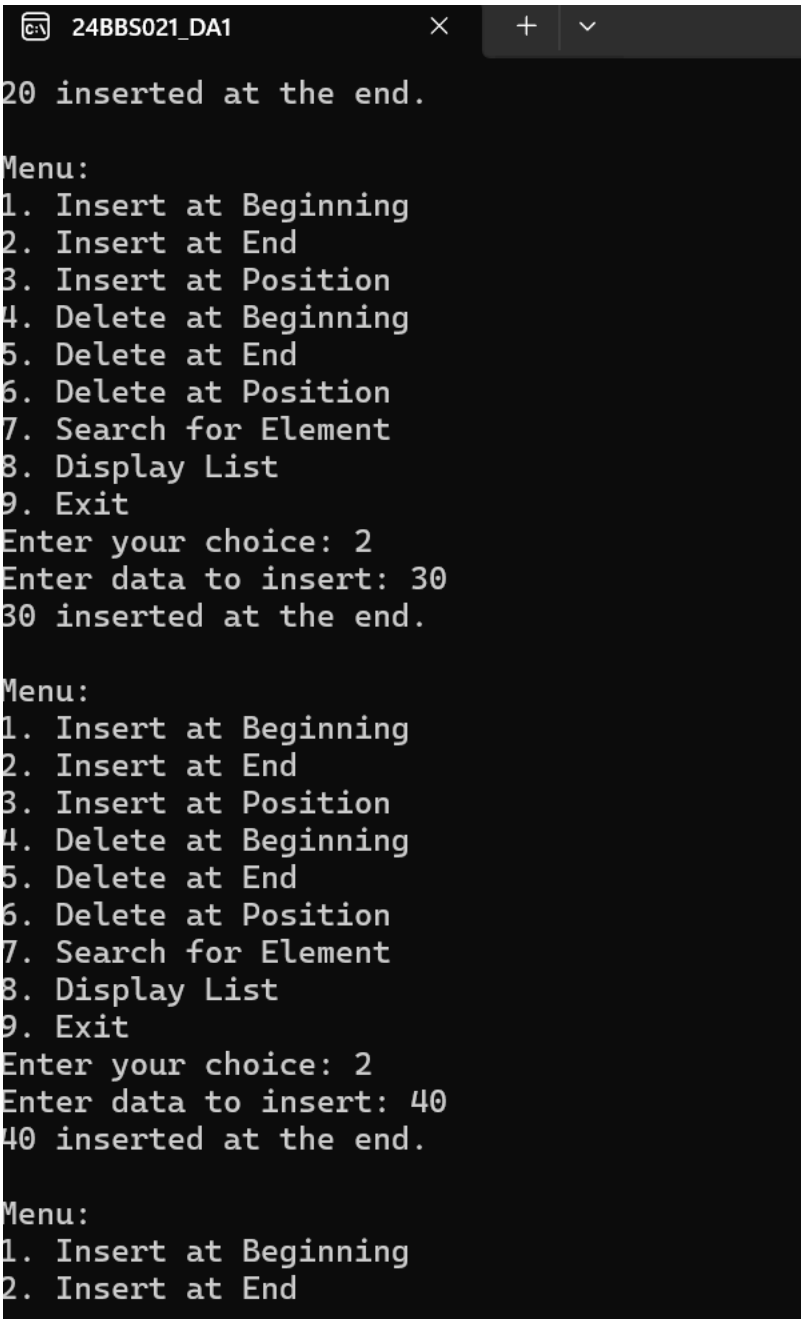
```
printf("5. Delete at End\n");
printf("6. Delete at Position\n");
printf("7. Search for Element\n");
printf("8. Display List\n");
printf("9. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice)
{
   case 1:
      printf("Enter data to insert: ");
      scanf("%d", &data);
      InsertAtBeginning(data);
      break;
   case 2:
      printf("Enter data to insert: ");
      scanf("%d", &data);
      InsertAtEnd(data);
      break;
   case 3:
      printf("Enter data and position to insert: ");
      scanf("%d %d", &data, &position);
      InsertAtPosition(data, position);
      break;
   case 4:
      DeleteAtBeginning();
      break;
   case 5:
      DeleteAtEnd();
      break;
```

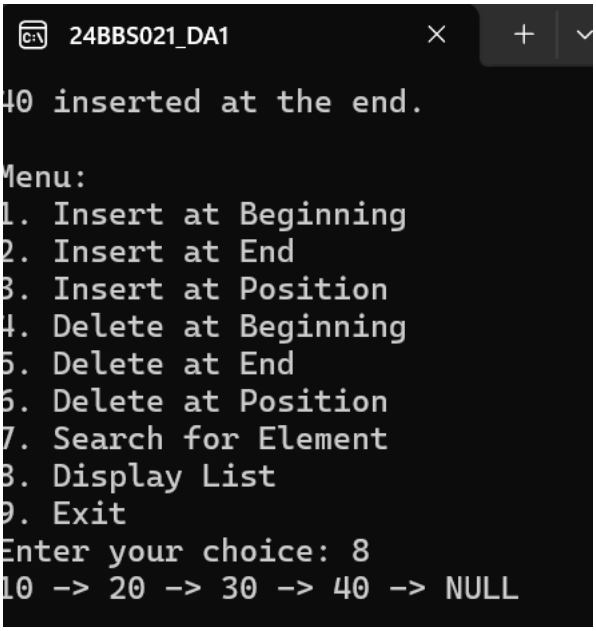
```
            case 6:
                printf("Enter position to delete: ");
                scanf("%d", &position);
                DeleteAtPosition(position);
                break;
            case 7:
                printf("Enter element to search: ");
                scanf("%d", &data);
                Search(data);
                break;
            case 8:
                Display();
                break;
            case 9:
                exit(0);
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
    return 0;
}
```

PROGRAM – OUTPUT:

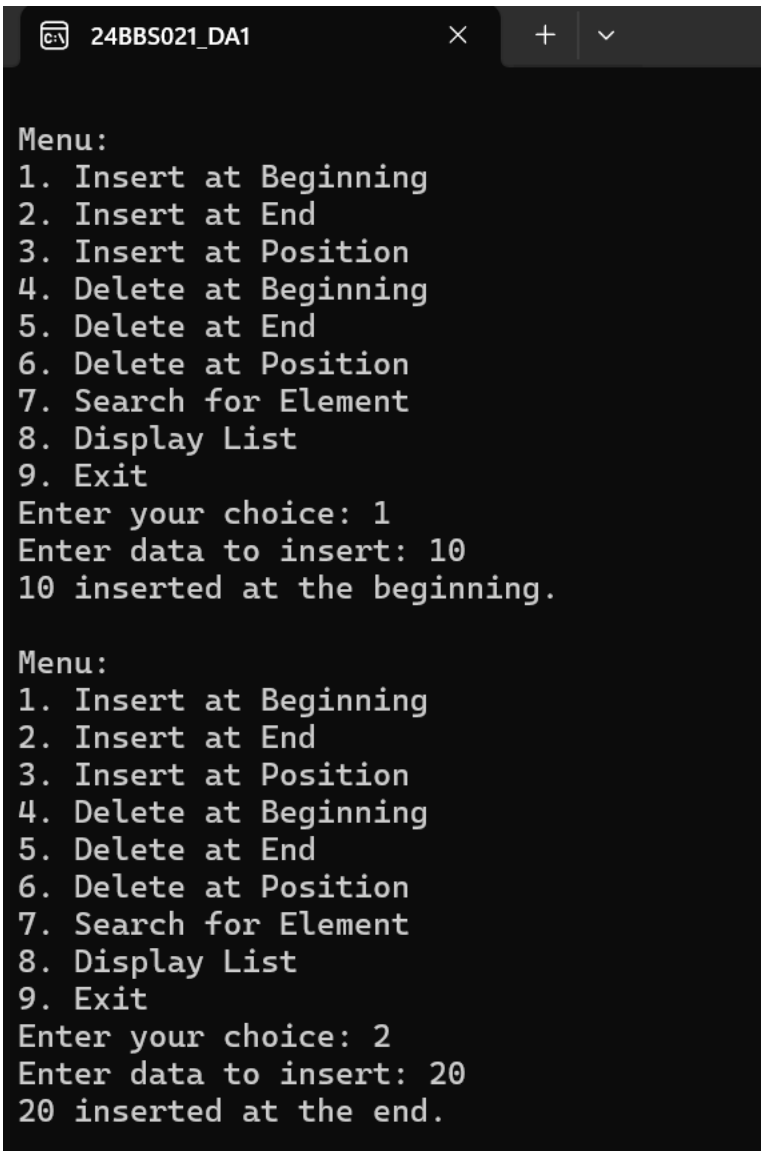
TEST CASE 1:

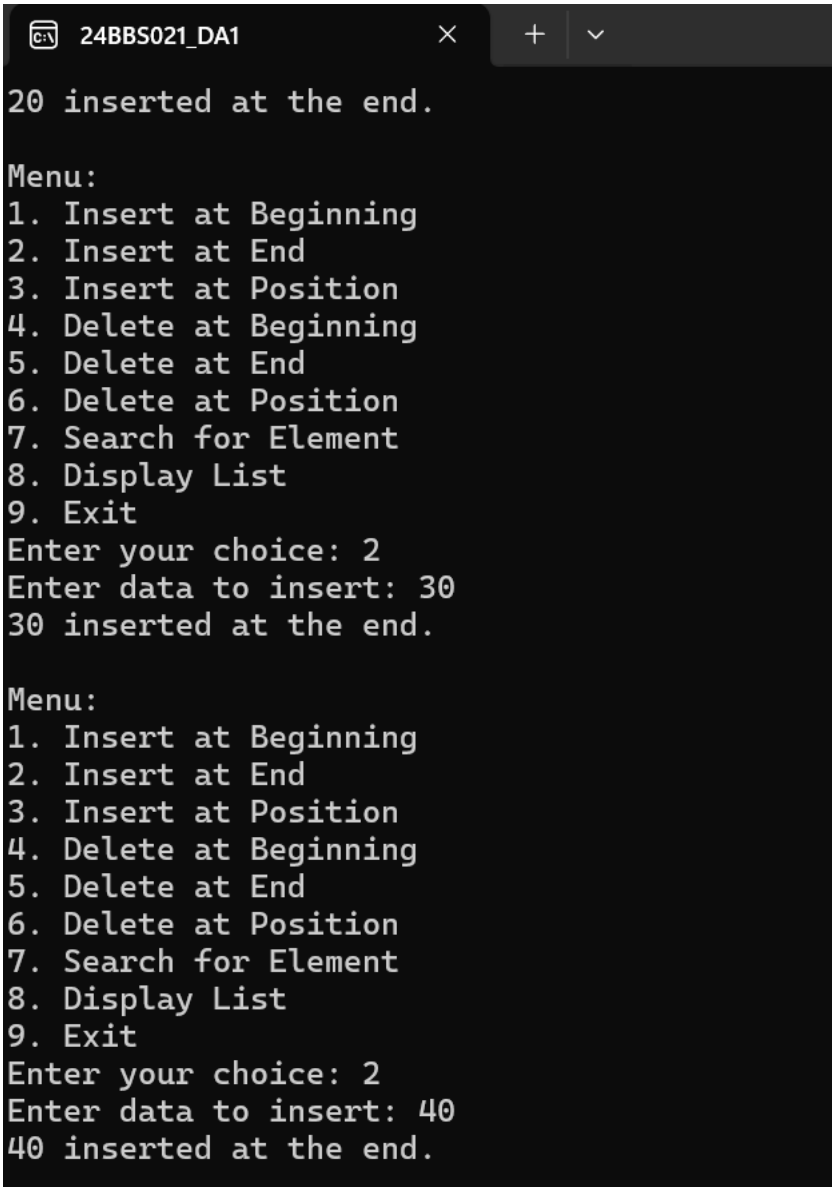


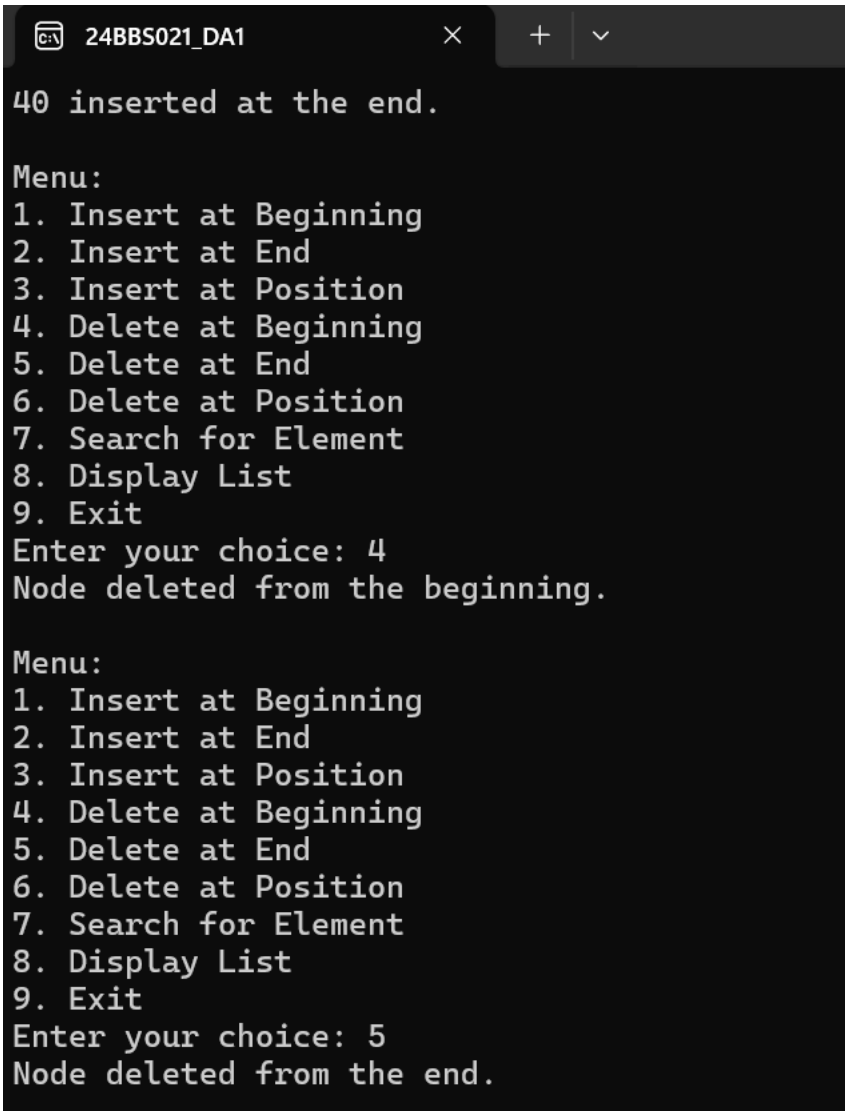

24BBS021_DA1
20 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 30
30 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 40
40 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End


24BBS021_DA1
40 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 8
10 -> 20 -> 30 -> 40 -> NULL

TEST CASE 2:

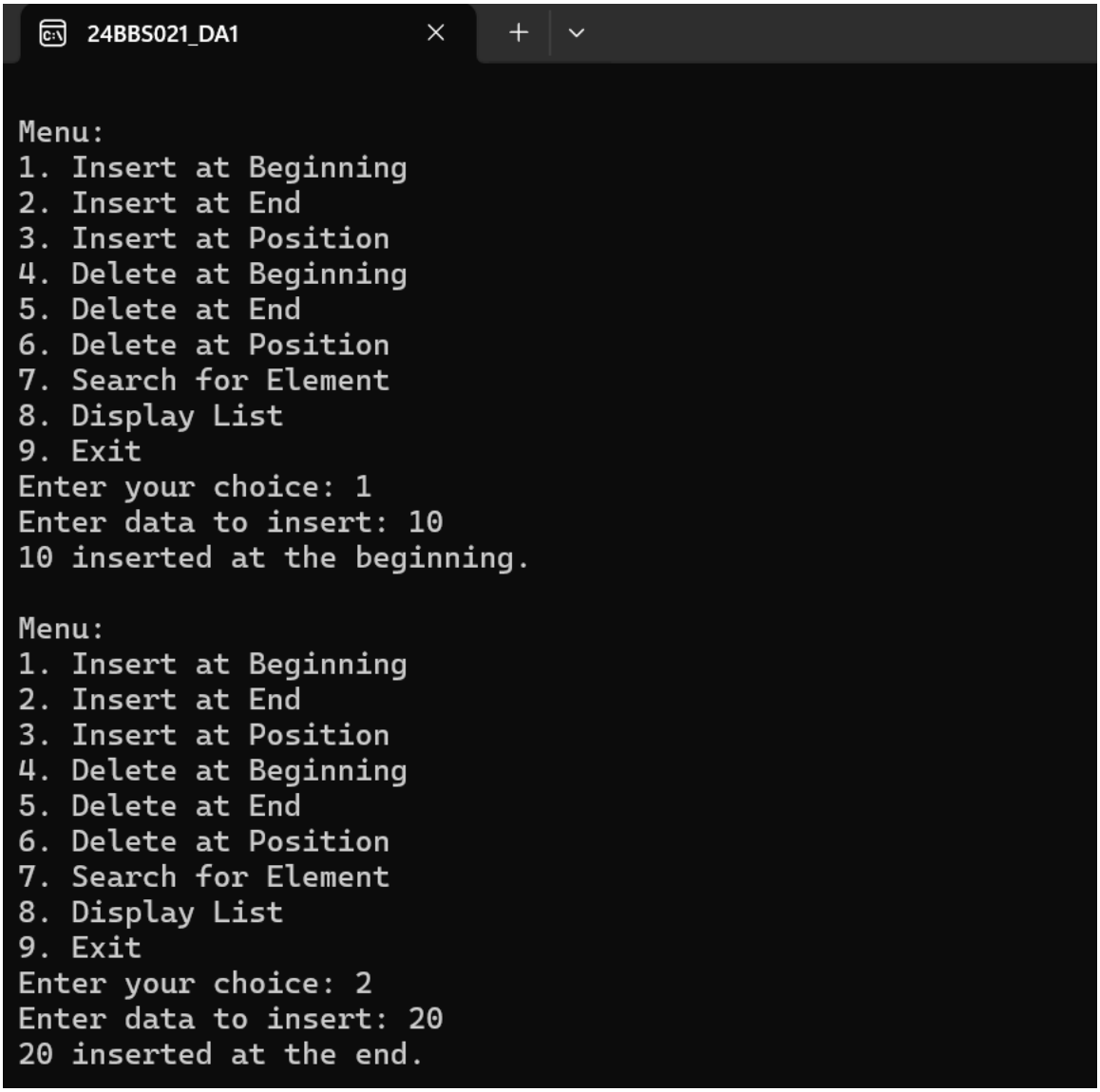

24BBS021_DA1
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 1
Enter data to insert: 10
10 inserted at the beginning.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 20
20 inserted at the end.

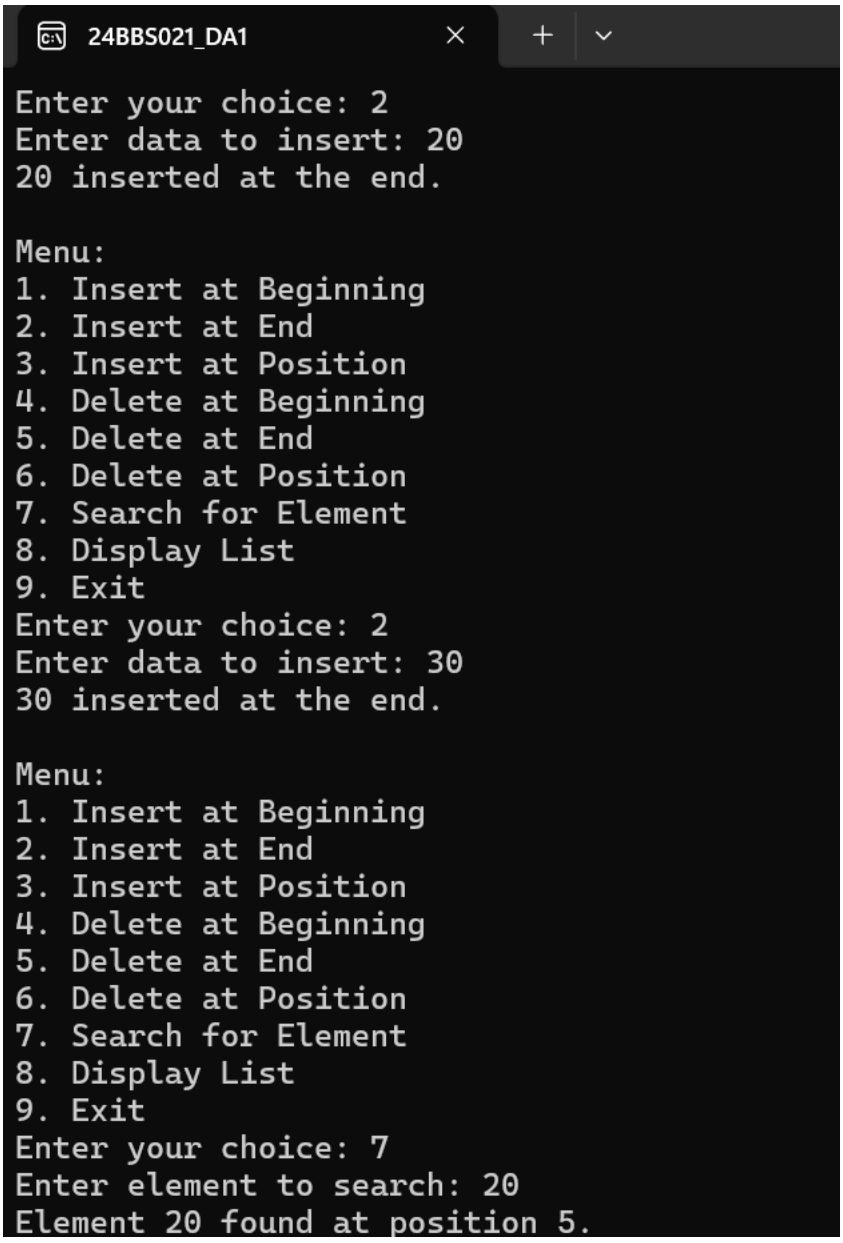

24BBS021_DA1
20 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 30
30 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 40
40 inserted at the end.

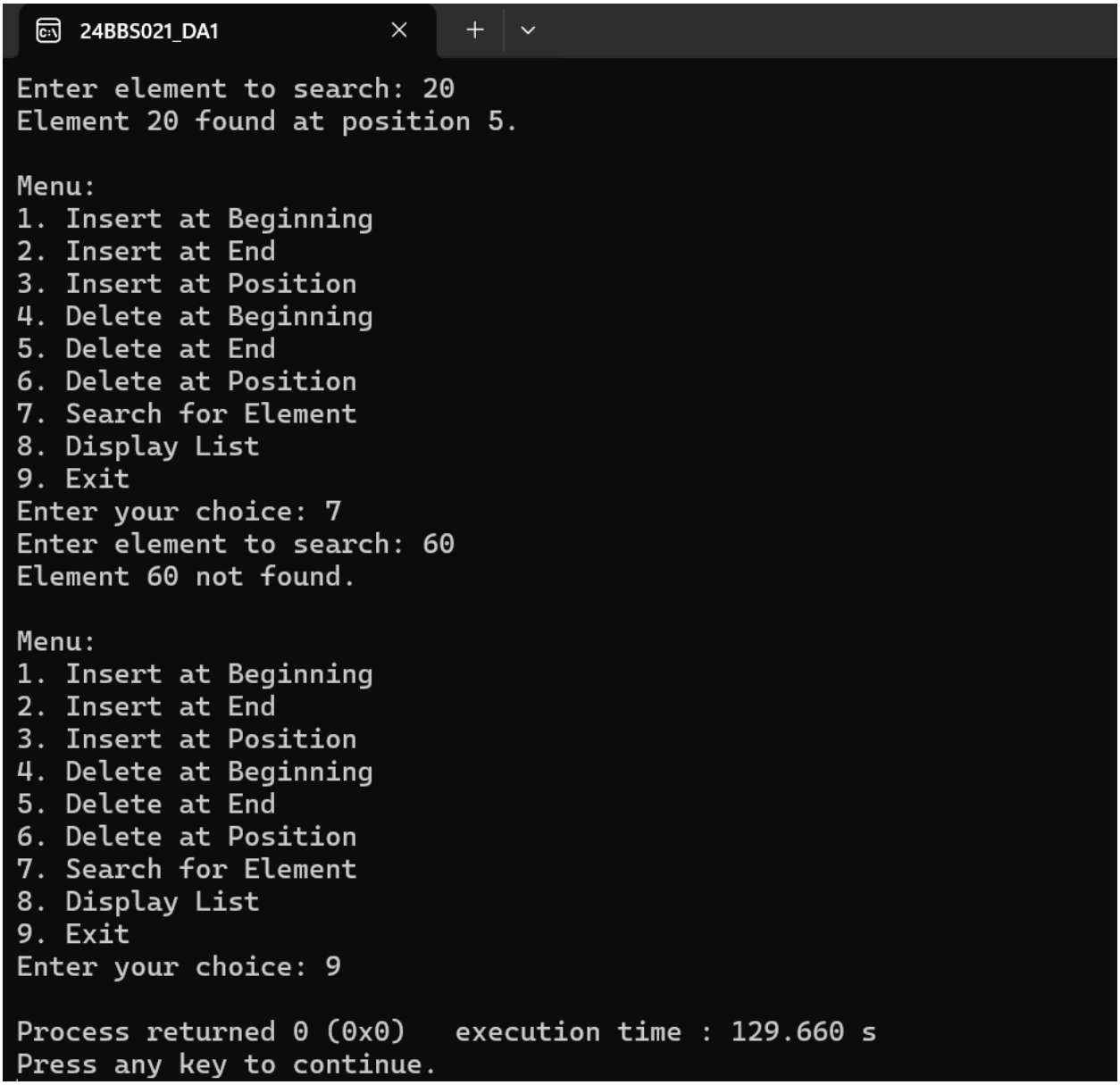

24BBS021_DA1
40 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 4
Node deleted from the beginning.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 5
Node deleted from the end.


24BBS021_DA1
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 6
Enter position to delete: 2
Node deleted from position 2.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 8
10 -> 30 -> 40 -> 20 -> 30 -> NULL

TEST CASE 3:


24BBS021_DA1
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 1
Enter data to insert: 10
10 inserted at the beginning.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 20
20 inserted at the end.


24BBS021_DA1
Enter your choice: 2
Enter data to insert: 20
20 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 30
30 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 7
Enter element to search: 20
Element 20 found at position 5.


24BBS021_DA1
Enter element to search: 20
Element 20 found at position 5.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 7
Enter element to search: 60
Element 60 not found.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 9
Process returned 0 (0x0) execution time : 129.660 s
Press any key to continue.

5) PSEUDOCODE:

```
START
  DEFINE Node with fields: data, next, prev
  DEFINE head as NULL

  FUNCTION InsertAtBeginning(data):
    CREATE newNode
    SET newNode.data = data
    SET newNode.next = head
    SET newNode.prev = NULL
    IF head != NULL:
      SET head.prev = newNode
    SET head = newNode

  FUNCTION InsertAtEnd(data):
    CREATE newNode
    SET newNode.data = data
    SET newNode.next = NULL
    IF head == NULL:
      SET head = newNode
    ELSE:
      SET current = head
      WHILE current.next != NULL:
        SET current = current.next
      SET current.next = newNode
      SET newNode.prev = current

  FUNCTION InsertAtPosition(data, position):
    CREATE newNode
    SET newNode.data = data
```

```
    SET current = head
    SET count = 1

    IF position == 1:
        CALL InsertAtBeginning(data)
    ELSE:
        WHILE current != NULL AND count < position-1:
            SET current = current.next
            SET count = count + 1

        IF current == NULL:
            PRINT "Position is out of range"
        ELSE:
            SET newNode.next = current.next
            SET newNode.prev = current
            IF current.next != NULL:
                SET current.next.prev = newNode
            SET current.next = newNode

FUNCTION DeleteAtBeginning():
    IF head == NULL:
        PRINT "List is empty!"
    ELSE:
        SET temp = head
        SET head = head.next
        IF head != NULL:
            SET head.prev = NULL
        FREE temp
        PRINT "Node deleted from the beginning"
```

```
FUNCTION DeleteAtEnd():
    IF head == NULL:
        PRINT "List is empty!"
    ELSE:
        SET current = head
        WHILE current.next != NULL:
            SET current = current.next
        IF current.prev != NULL:
            SET current.prev.next = NULL
        FREE current
        PRINT "Node deleted from the end"

FUNCTION DeleteAtPosition(position):
    IF head == NULL:
        PRINT "List is empty!"
    ELSE:
        SET current = head
        SET count = 1

        IF position == 1:
            CALL DeleteAtBeginning()
        ELSE:
            WHILE current != NULL AND count < position:
                SET current = current.next
                SET count = count + 1

            IF current == NULL:
                PRINT "Position is out of range"
            ELSE:
                IF current.prev != NULL:
```

```
            SET current.prev.next = current.next
        IF current.next != NULL:
            SET current.next.prev = current.prev
        FREE current
        PRINT "Node deleted from position", position

FUNCTION Search(value):
    SET current = head
    SET position = 1
    WHILE current != NULL:
        IF current.data == value:
            PRINT "Element found at position", position
            RETURN
        SET current = current.next
        SET position = position + 1
    PRINT "Element not found in the list"

FUNCTION Display():
    IF head == NULL:
        PRINT "List is empty"
    ELSE:
        SET current = head
        WHILE current != NULL:
            PRINT current.data
            SET current = current.next

FUNCTION MainMenu():
    PRINT "Menu:"
    PRINT "1. Insert at Beginning"
    PRINT "2. Insert at End"
```

```
PRINT "3. Insert at Position"
PRINT "4. Delete at Beginning"
PRINT "5. Delete at End"
PRINT "6. Delete at Position"
PRINT "7. Search for Element"
PRINT "8. Display List"
PRINT "9. Exit"
GET user choice
IF choice == 1:
   GET data
   CALL InsertAtBeginning(data)
ELSE IF choice == 2:
   GET data
   CALL InsertAtEnd(data)
ELSE IF choice == 3:
   GET data, position
   CALL InsertAtPosition(data, position)
ELSE IF choice == 4:
   CALL DeleteAtBeginning()
ELSE IF choice == 5:
   CALL DeleteAtEnd()
ELSE IF choice == 6:
   GET position
   CALL DeleteAtPosition(position)
ELSE IF choice == 7:
   GET value
   CALL Search(value)
ELSE IF choice == 8:
   CALL Display()
ELSE IF choice == 9:
```

EXIT

END

PROGRAM – INPUT:

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node* next;
   struct Node* prev;
};
struct Node* head = NULL;
void InsertAtBeginning(int data)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
   newNode->data = data;
   newNode->next = head;
   newNode->prev = NULL;
   if (head != NULL)
   {
      head->prev = newNode;
   }
   head = newNode;
   printf("%d inserted at the beginning.\n", data);
}
void InsertAtEnd(int data)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
   struct Node* temp = head;
   newNode->data = data;
```

```
    newNode->next = NULL;
    if (head == NULL)
    {
        head = newNode;
        newNode->prev = NULL;
    }
    else
    {
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
    printf("%d inserted at the end.\n", data);
}
void InsertAtPosition(int data, int position)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    struct Node* temp = head;
    int count = 1;
    newNode->data = data;
    if (position == 1)
    {
        InsertAtBeginning(data);
    }
    else
    {
```

```
        while (temp != NULL && count < position - 1)
        {
            temp = temp->next;
            count++;
        }
        if (temp == NULL)
        {
            printf("Position out of range.\n");
        }
        else
        {
            newNode->next = temp->next;
            newNode->prev = temp;
            if (temp->next != NULL)
            {
                temp->next->prev = newNode;
            }
            temp->next = newNode;
            printf("%d inserted at position %d.\n", data, position);
        }
    }
}
void DeleteAtBeginning()
{
    if (head == NULL)
    {
        printf("List is empty.\n");
    }
    else
    {
```

```
        struct Node* temp = head;
        head = head->next;
        if (head != NULL)
        {
            head->prev = NULL;
        }
        free(temp);
        printf("Node deleted from the beginning.\n");
    }
}
void DeleteAtEnd()
{
    if (head == NULL)
    {
        printf("List is empty.\n");
    }
    else
    {
        struct Node* temp = head;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        if (temp->prev != NULL)
        {
            temp->prev->next = NULL;
        }
        free(temp);
        printf("Node deleted from the end.\n");
    }
```

```
}
void DeleteAtPosition(int position) {
   if (head == NULL) {
      printf("List is empty.\n");
   } else {
      struct Node* temp = head;
      int count = 1;
      if (position == 1) {
         DeleteAtBeginning();
      } else {
         while (temp != NULL && count < position) {
            temp = temp->next;
            count++;
         }
         if (temp == NULL) {
            printf("Position out of range.\n");
         } else {
            if (temp->prev != NULL) {
               temp->prev->next = temp->next;
            }
            if (temp->next != NULL) {
               temp->next->prev = temp->prev;
            }
            free(temp);
            printf("Node deleted from position %d.\n", position);
         }
      }
   }
}
void Search(int value) {
```

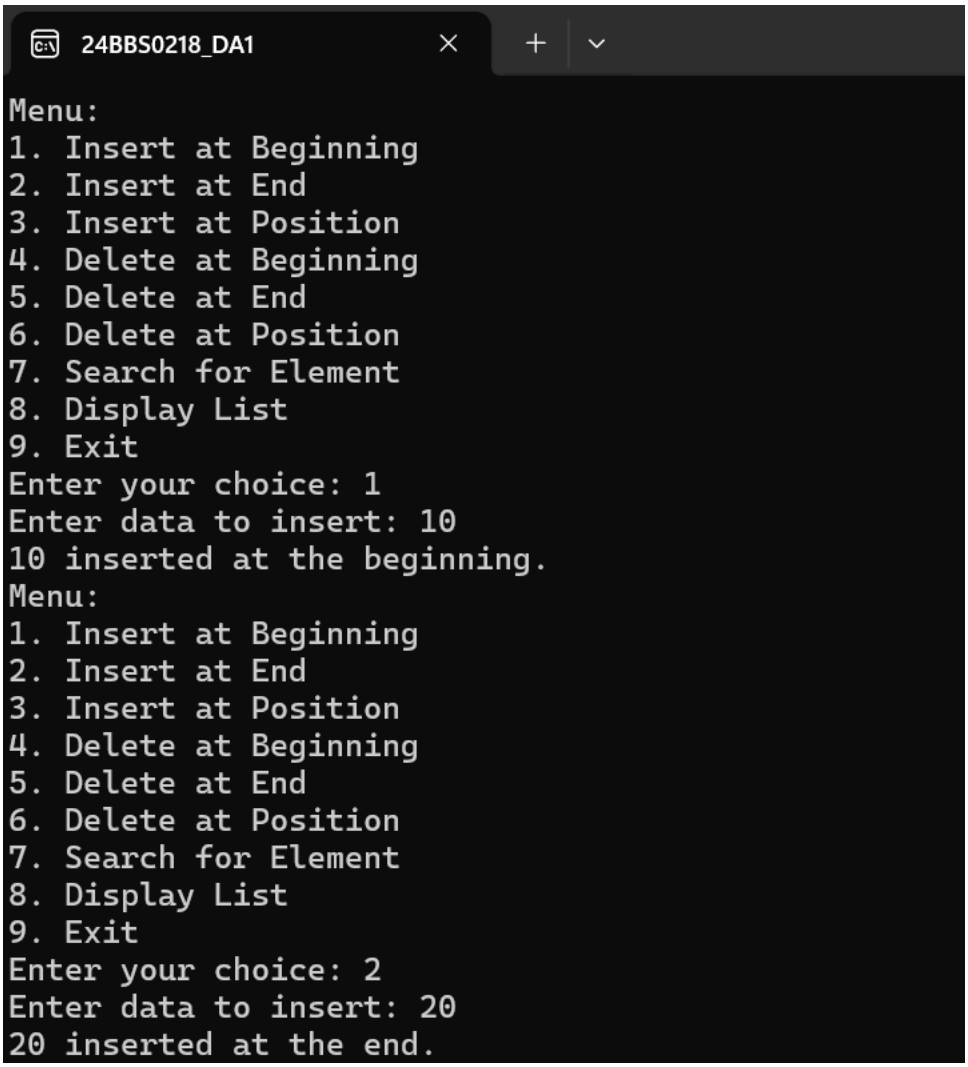
```
    struct Node* temp = head;
    int position = 1;
    while (temp != NULL) {
        if (temp->data == value) {
            printf("Element found at position %d.\n", position);
            return;
        }
        temp = temp->next;
        position++;
    }
    printf("Element not found in the list.\n");
}
void Display() {
    if (head == NULL) {
        printf("List is empty.\n");
    } else {
        struct Node* temp = head;
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}
int main() {
    int choice, data, position;
    while (1) {
        printf("Menu:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
```

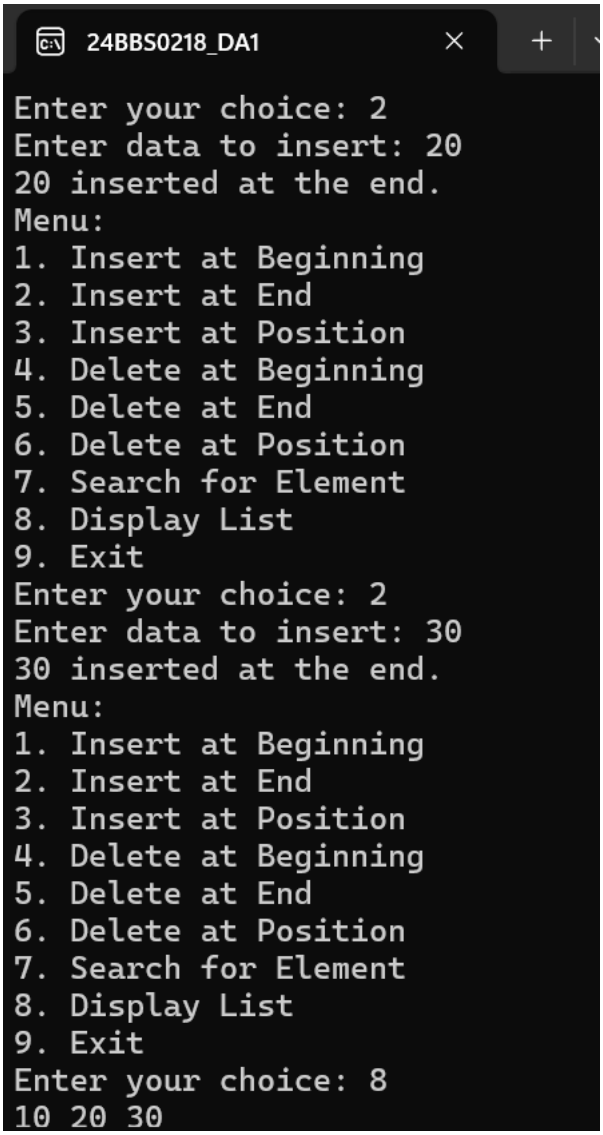
```
printf("3. Insert at Position\n");
printf("4. Delete at Beginning\n");
printf("5. Delete at End\n");
printf("6. Delete at Position\n");
printf("7. Search for Element\n");
printf("8. Display List\n");
printf("9. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
   case 1:
      printf("Enter data to insert: ");
      scanf("%d", &data);
      InsertAtBeginning(data);
      break;
   case 2:
      printf("Enter data to insert: ");
      scanf("%d", &data);
      InsertAtEnd(data);
      break;
   case 3:
      printf("Enter data and position to insert: ");
      scanf("%d %d", &data, &position);
      InsertAtPosition(data, position);
      break;
   case 4:
      DeleteAtBeginning();
      break;
   case 5:
      DeleteAtEnd();
```

```
            break;
        case 6:
            printf("Enter position to delete: ");
            scanf("%d", &position);
            DeleteAtPosition(position);
            break;
        case 7:
            printf("Enter element to search: ");
            scanf("%d", &data);
            Search(data);
            break;
        case 8:
            Display();
            break;
        case 9:
            exit(0);
        default:
            printf("Invalid choice. Try again.\n");
      }
   }
   return 0;
}
```

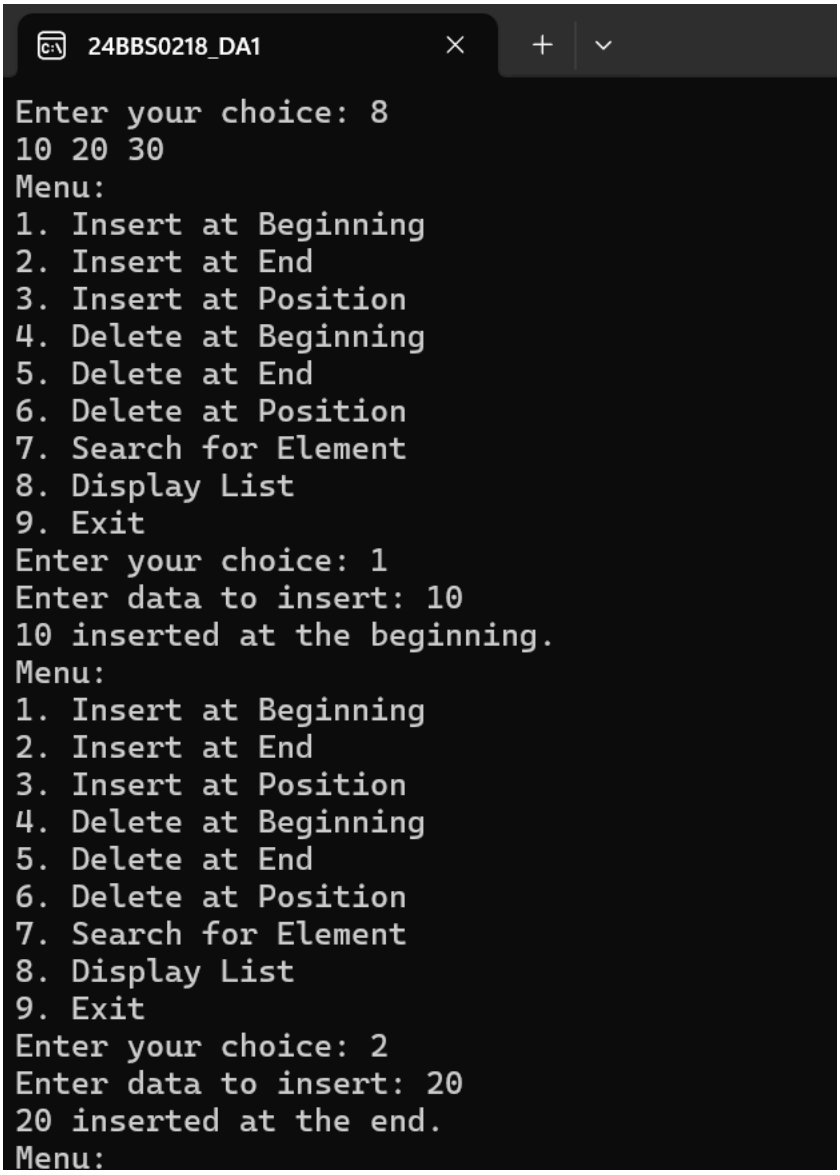
PROGRAM – OUTPUT:

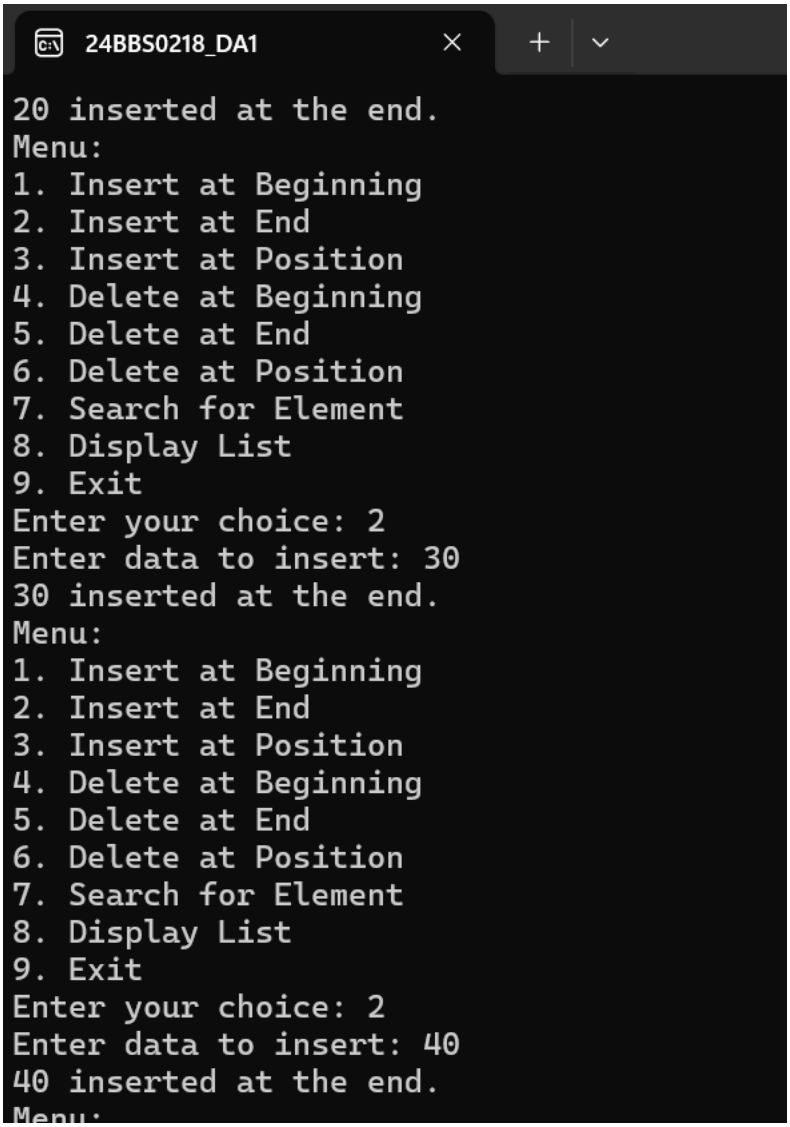
TEST CASE 1:

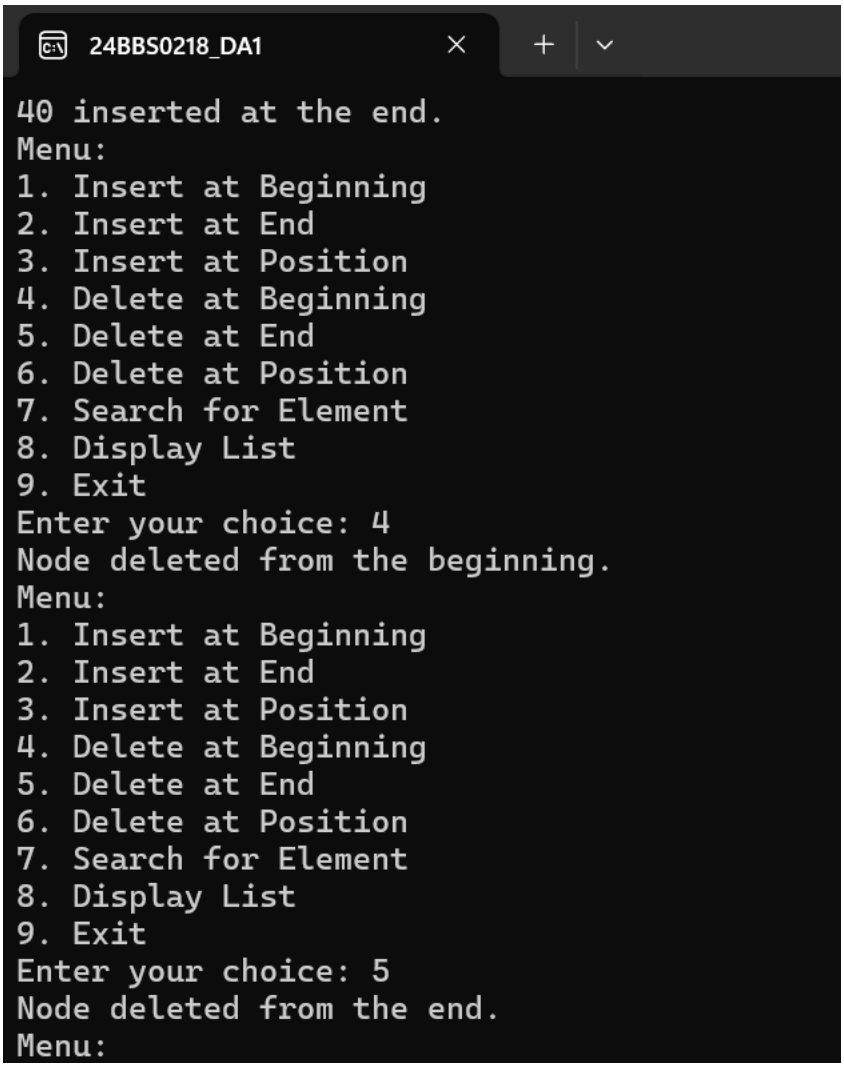


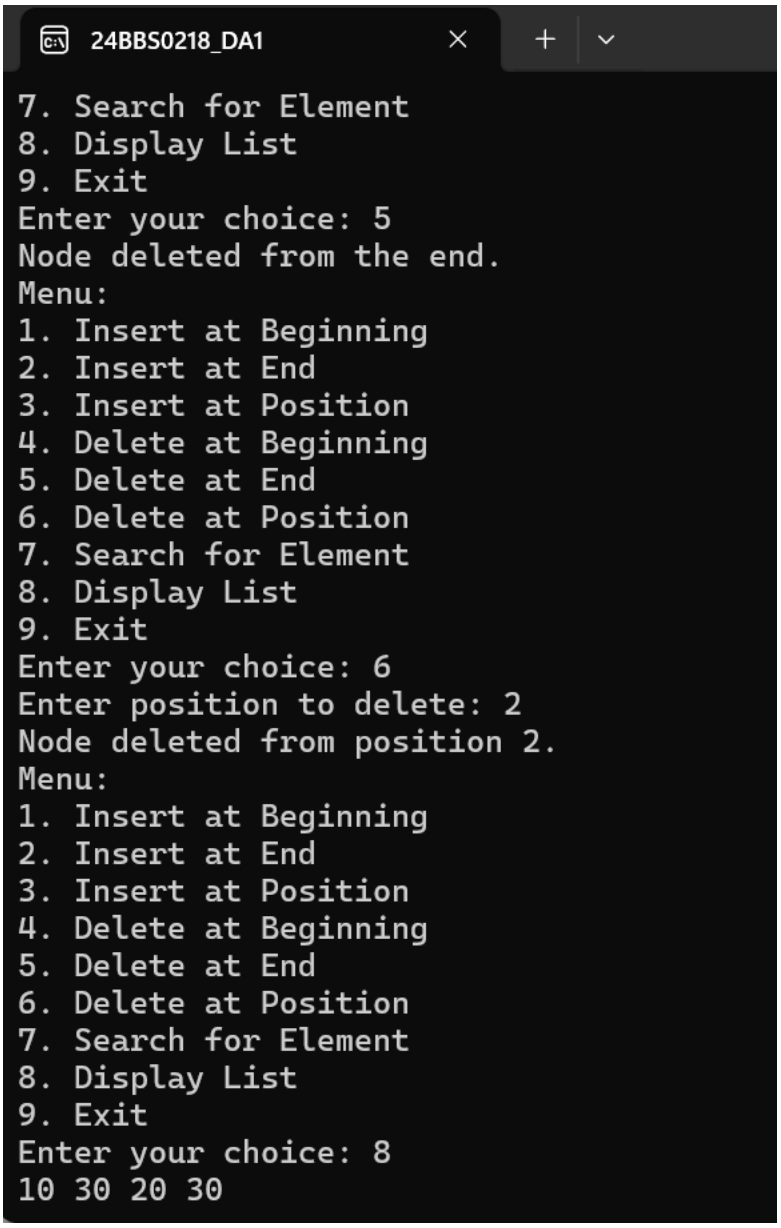

24BBS0218_DA1
Enter your choice: 2
Enter data to insert: 20
20 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 30
30 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 8
10 20 30

TEST CASE 2:

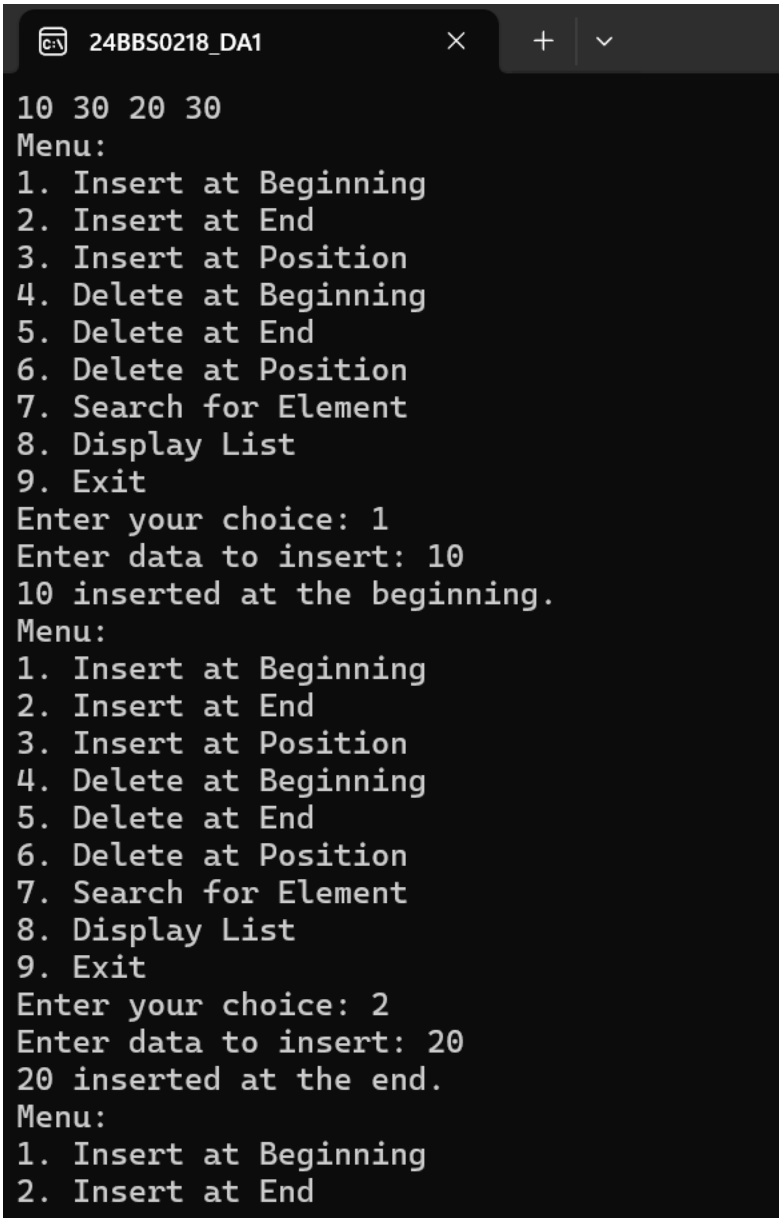

24BBS0218_DA1
Enter your choice: 8
10 20 30
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 1
Enter data to insert: 10
10 inserted at the beginning.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 20
20 inserted at the end.
Menu:

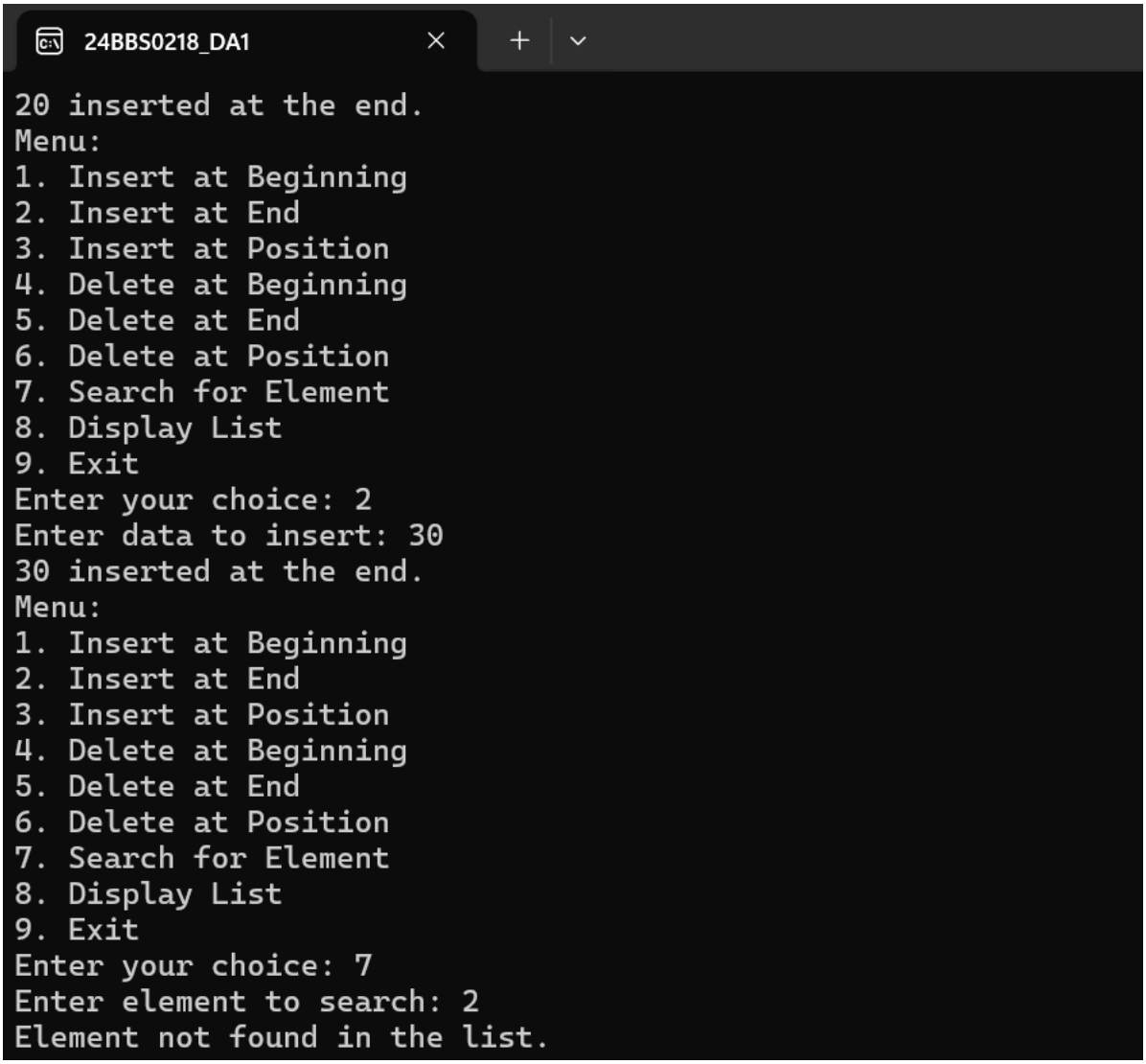

24BBS0218_DA1
20 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 30
30 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 40
40 inserted at the end.
Menu:

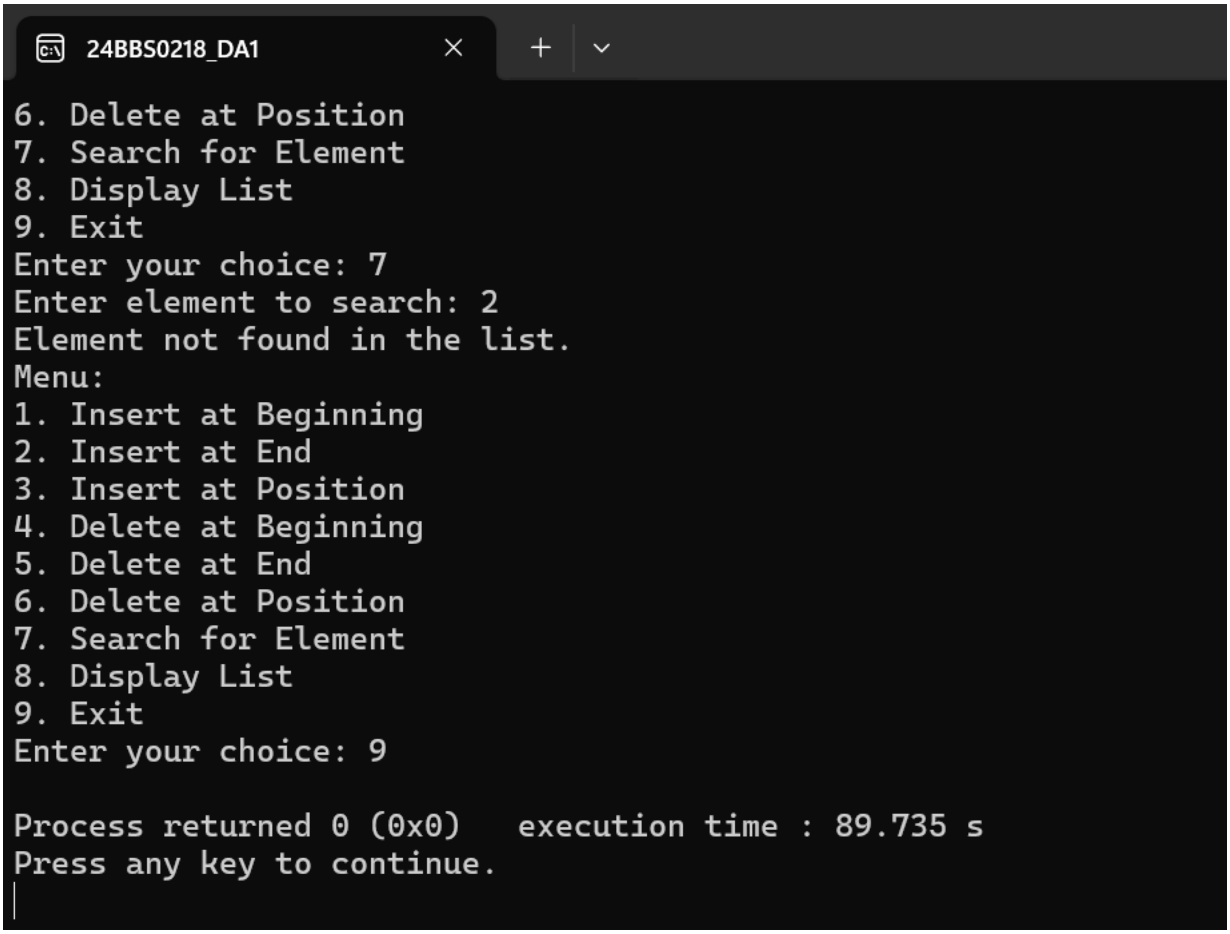

24BBS0218_DA1
40 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 4
Node deleted from the beginning.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 5
Node deleted from the end.
Menu:


24BBS0218_DA1
7. Search for Element
8. Display List
9. Exit
Enter your choice: 5
Node deleted from the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 6
Enter position to delete: 2
Node deleted from position 2.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 8
10 30 20 30

TEST CASE 3:


24BBS0218_DA1
10 30 20 30
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 1
Enter data to insert: 10
10 inserted at the beginning.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 20
20 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End


24BBS0218_DA1
20 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 2
Enter data to insert: 30
30 inserted at the end.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 7
Enter element to search: 2
Element not found in the list.


24BBS0218_DA1
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 7
Enter element to search: 2
Element not found in the list.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 9
Process returned 0 (0x0) execution time : 89.735 s
Press any key to continue.