DOUBLY LINKED LIST:

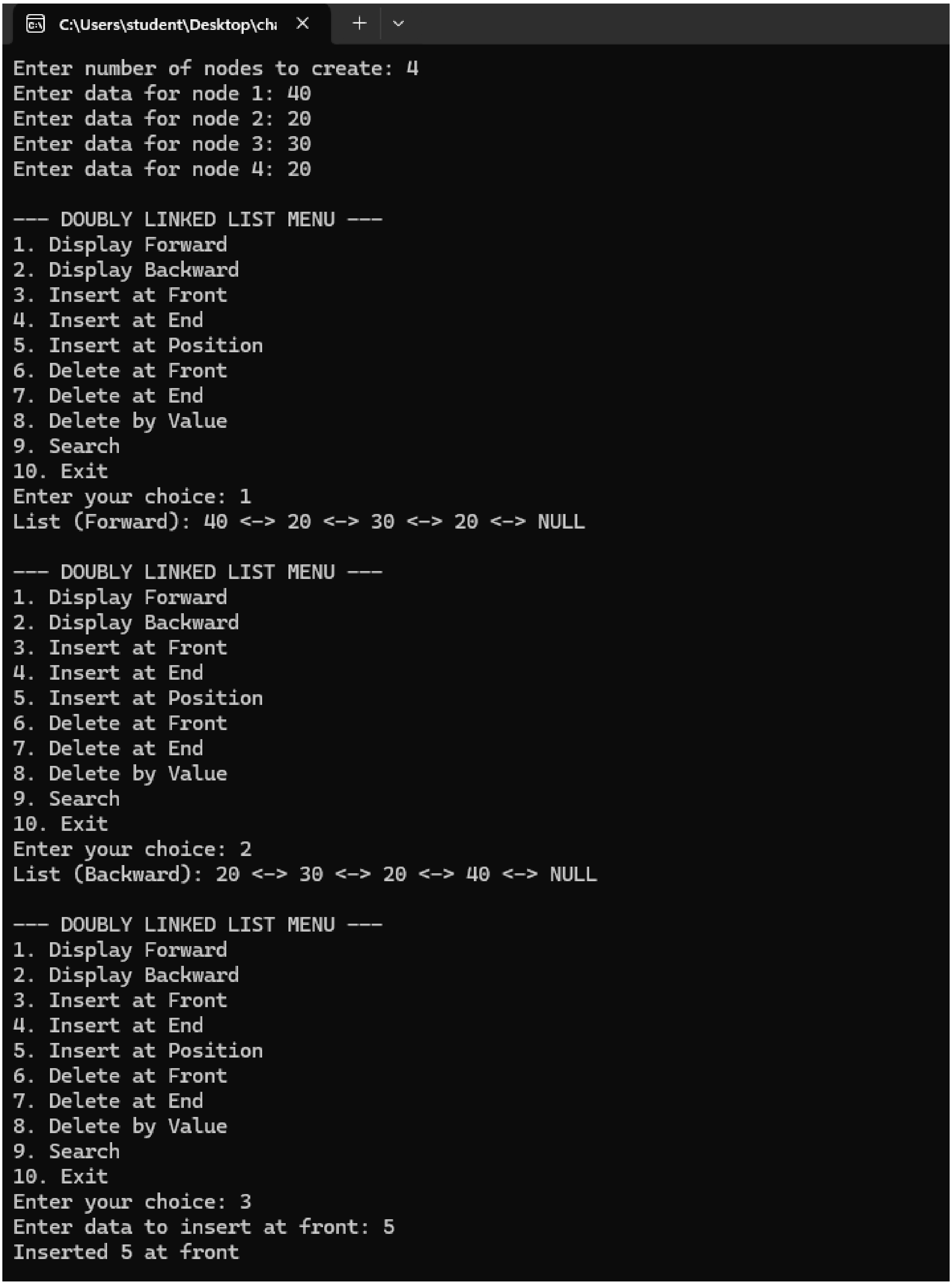
```
#include <stdio.h>
#include <stdlib.h>

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

struct Node *head = NULL, *tail = NULL;

void createList(int n) {
    int i, data;
    struct Node *newNode;
    for (i = 1; i <= n; i++) {
        printf("Enter data for node %d: ", i);
        scanf("%d", &data);
        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->prev = newNode->next = NULL;
        if (head == NULL) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }
}
```

```
void insertAtFront(int data) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = head;
    if (head == NULL)
        head = tail = newNode;
    else {
        head->prev = newNode;
        head = newNode;
    }
    printf("Inserted %d at front\n",data);
}

void insertAtEnd(int data) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = tail;
    if (tail == NULL)
        head = tail = newNode;
    else {
        tail->next = newNode;
        tail = newNode;
    }
    printf("Inserted %d at end\n",data);
}

void insertAtPosition(int data, int pos) {
    int i;
    struct Node *newNode, *temp = head;
    if (pos == 1) {
        insertAtFront(data);
        return;
    }
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    for (i = 1; i < pos - 1 && temp != NULL; i++)
        temp = temp->next;
    if (temp == NULL || temp->next == NULL) {
        insertAtEnd(data);
        return;
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    temp->next->prev = newNode;
    temp->next = newNode;
    printf("Inserted %d at position %d\n",data,pos);
}

void deleteAtFront() {
    struct Node *temp;
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    temp = head;
    head = head->next;
    if (head != NULL)
        head->prev = NULL;
    else
```

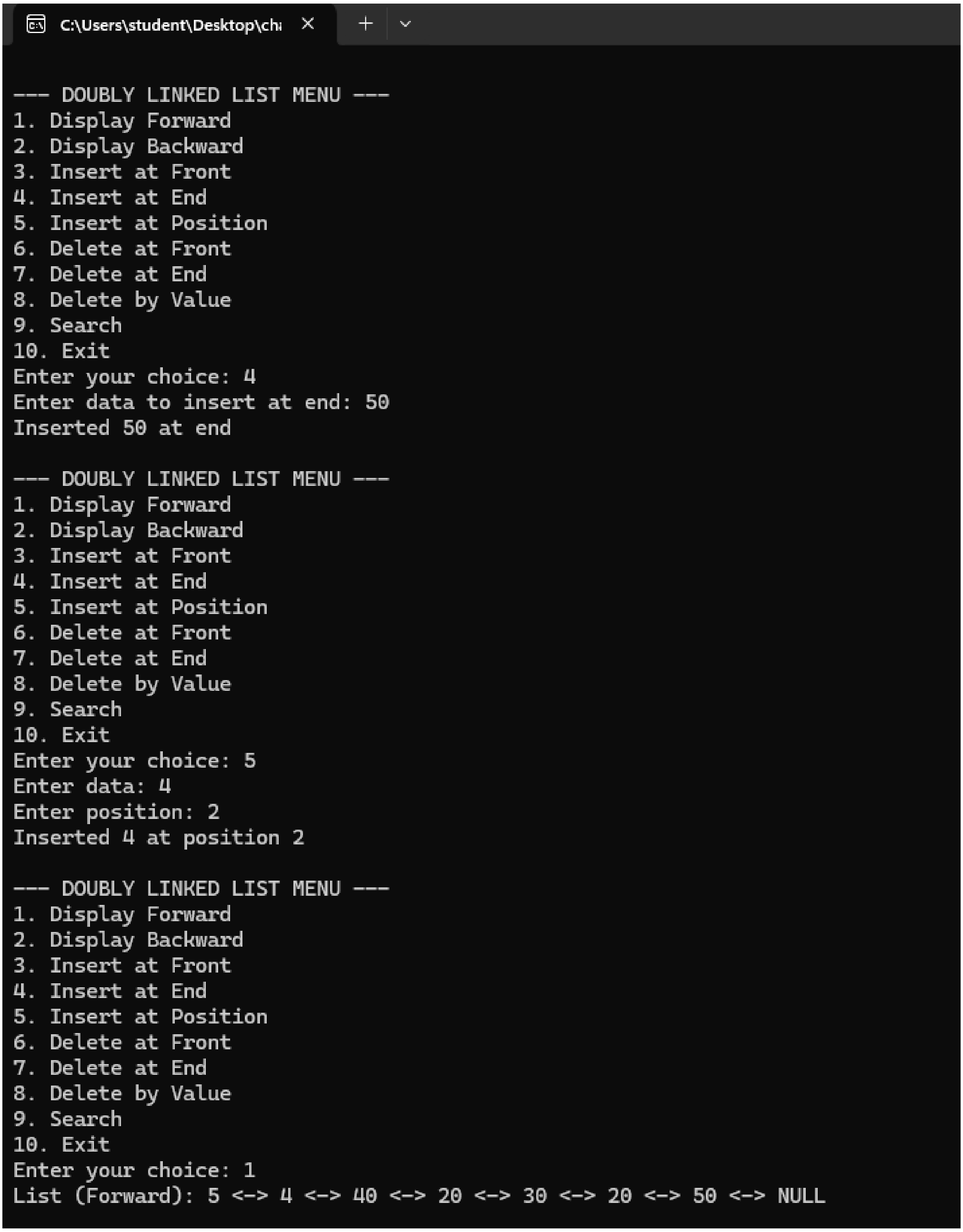
```
        tail = NULL;
        printf("Deleted %d at front\n",temp->data);
    free(temp);
}

void deleteAtEnd() {
    struct Node *temp;
    if (tail == NULL) {
        printf("List is empty!\n");
        return;
    }
    temp = tail;
    tail = tail->prev;
    if (tail != NULL)
        tail->next = NULL;
    else
        head = NULL;
        printf("Deleted %d at end\n",temp->data);
    free(temp);
}

void deleteByValue(int value) {
    struct Node *temp = head;
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    while (temp != NULL && temp->data != value)
        temp = temp->next;
    if (temp == NULL) {
        printf("Value not found!\n");
        return;
    }
    if (temp == head)
        deleteAtFront();
    else if (temp == tail)
        deleteAtEnd();
    else {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
        printf("Deleted %d \n",temp->data);
        free(temp);
    }
}

void search(int value) {
    struct Node *temp = head;
    int pos = 1;
    while (temp != NULL) {
        if (temp->data == value) {
            printf("Value %d found at position %d\n", value, pos);
            return;
        }
        temp = temp->next;
        pos++;
    }
    printf("Value %d not found in the list.\n", value);
}

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

void displayBackward() {
    struct Node *temp = tail;
    printf("List (Backward): ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->prev;
    }
    printf("NULL\n");
}

int main() {
    int choice, n, data, pos, value;

    printf("Enter number of nodes to create: ");
    scanf("%d", &n);
    createList(n);

    while (1) {
        printf("\n--- DOUBLY LINKED LIST MENU ---\n");
        printf("1. Display Forward\n");
        printf("2. Display Backward\n");
        printf("3. Insert at Front\n");
        printf("4. Insert at End\n");
        printf("5. Insert at Position\n");
        printf("6. Delete at Front\n");
        printf("7. Delete at End\n");
        printf("8. Delete by Value\n");
        printf("9. Search\n");
        printf("10. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {

          case 1:
            displayForward();
            break;

          case 2:
            displayBackward();
            break;

          case 3:
            printf("Enter data to insert at front: ");
            scanf("%d", &data);
            insertAtFront(data);
            break;

          case 4:
            printf("Enter data to insert at end: ");
            scanf("%d", &data);
            insertAtEnd(data);
            break;

          case 5:
            printf("Enter data: ");
            scanf("%d", &data);
            printf("Enter position: ");
            scanf("%d", &pos);
            insertAtPosition(data, pos);
            break;

        case 6:
            deleteAtFront();
            break;

        case 7:
            deleteAtEnd();
            break;

        case 8:
            printf("Enter value to delete: ");
            scanf("%d", &value);
            deleteByValue(value);
            break;

        case 9:
            printf("Enter value to search: ");
            scanf("%d", &value);
            search(value);
            break;

        case 10:
            printf("Exiting program...\n");
            exit(0);
```

```
            default:
                printf("Invalid choice! Try again.\n");
        }
    }

    return 0;
}
```

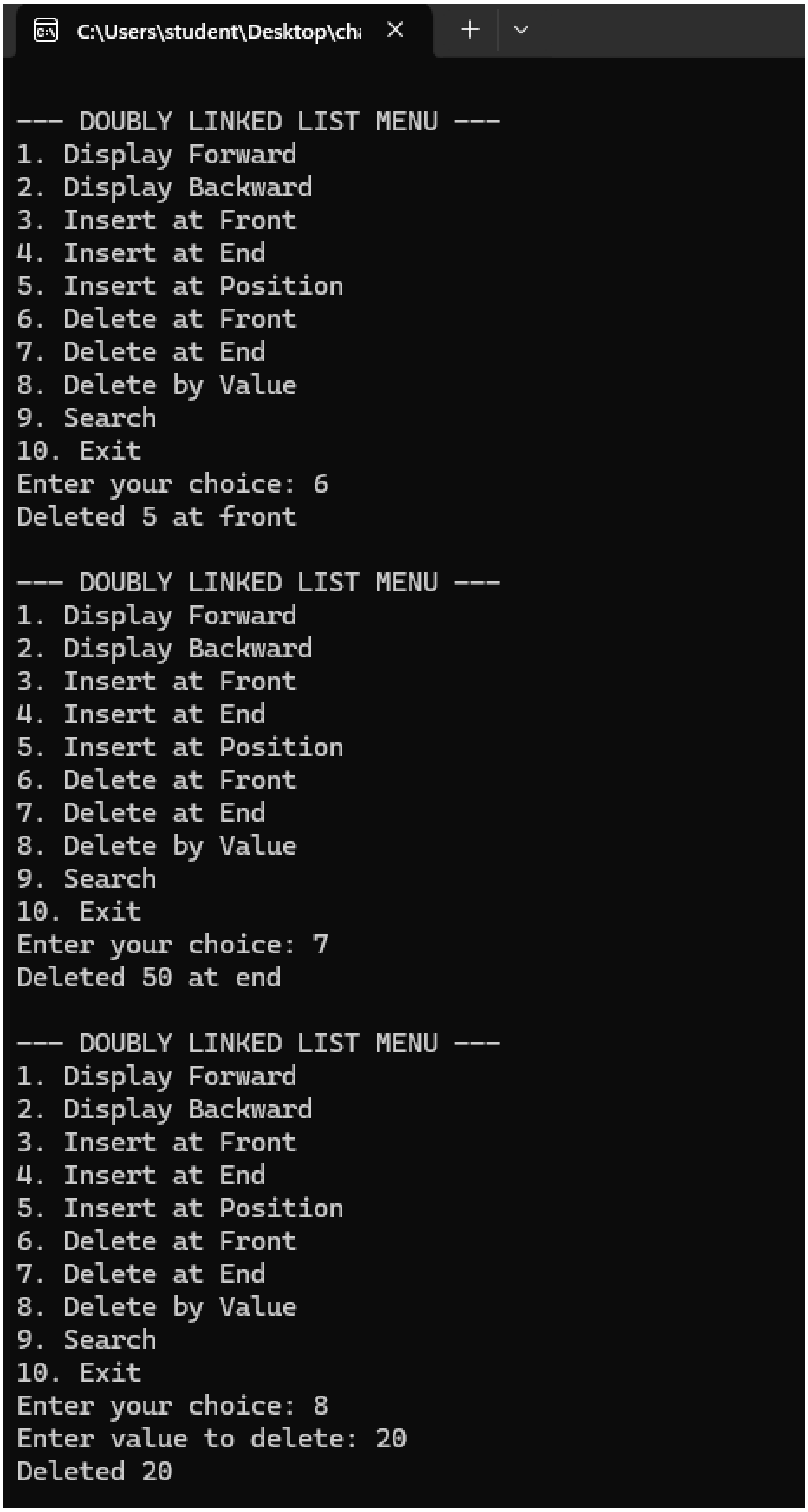
OUTPUT:



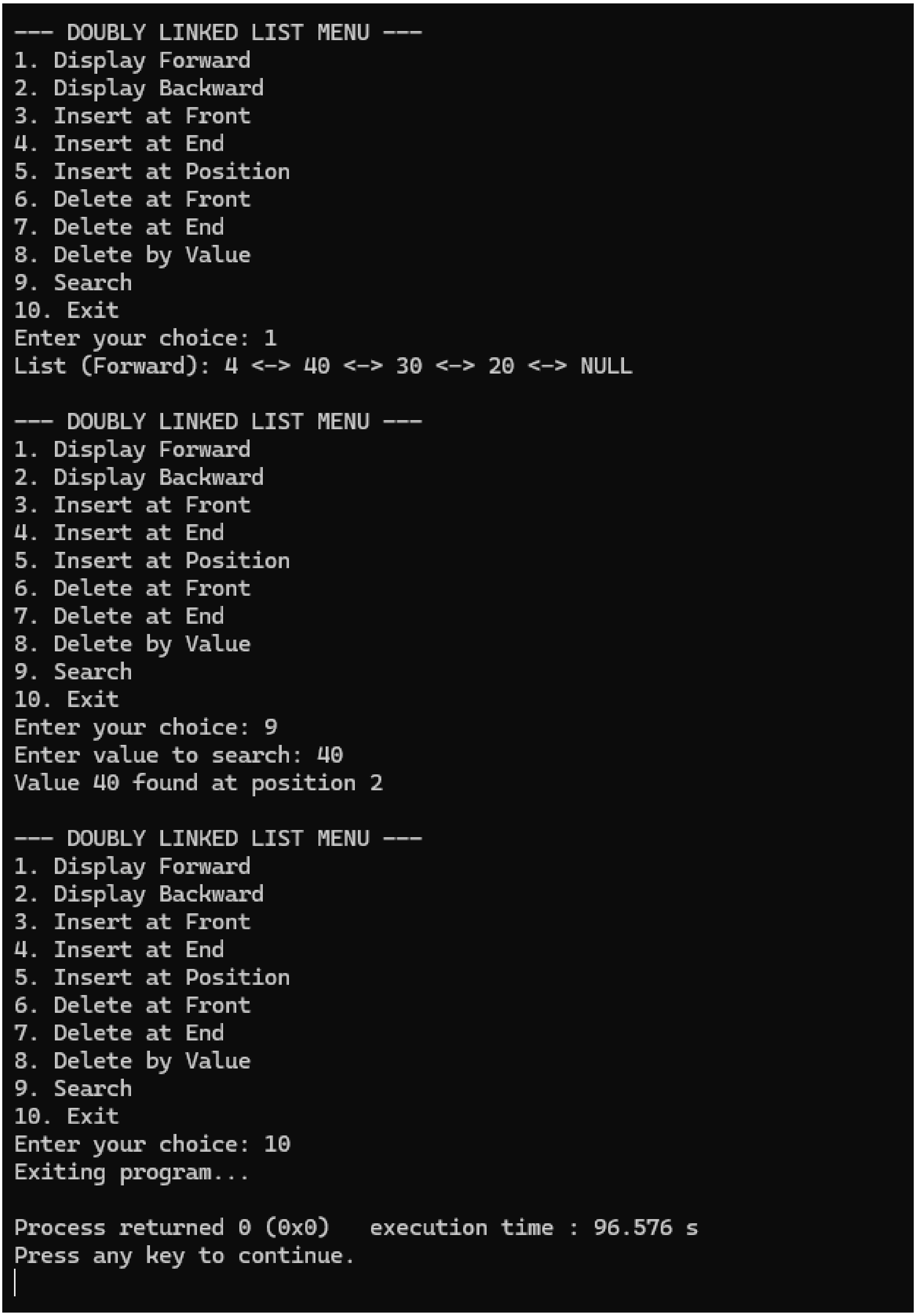
```
Enter number of nodes to create: 4
Enter data for node 1: 40
Enter data for node 2: 20
Enter data for node 3: 30
Enter data for node 4: 20

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 1
List (Forward): 40 <-> 20 <-> 30 <-> 20 <-> NULL

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 2
List (Backward): 20 <-> 30 <-> 20 <-> 40 <-> NULL

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 3
Enter data to insert at front: 5
Inserted 5 at front
```



```
--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 4
Enter data to insert at end: 50
Inserted 50 at end

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 5
Enter data: 4
Enter position: 2
Inserted 4 at position 2

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 1
List (Forward): 5 <-> 4 <-> 40 <-> 20 <-> 30 <-> 20 <-> 50 <-> NULL
```



```

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 6
Deleted 5 at front

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 7
Deleted 50 at end

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 8
Enter value to delete: 20
Deleted 20
```



```
--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 1
List (Forward): 4 <-> 40 <-> 30 <-> 20 <-> NULL

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 9
Enter value to search: 40
Value 40 found at position 2

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 10
Exiting program...

Process returned 0 (0x0)   execution time : 96.576 s
Press any key to continue.
```

OBSERVATION:

Page No:
Date.

1/10/25

Prg-7a

WAP to implement doubly Linked List with primitive operations.

a Create doubly Linked list.
b Insert a newNode to the left of the node
c Delete a node based on specificed value
d Display the contents of list.

Ans Pseudocode:

```
Struct node {
        Struct node *prev, *next;
         int data;
};

insertatFront (value)
-> void insertAtFront (int data) {
          Struct node * newNode {
          newNode->data = data
          newNode->prev = NULL
          newNode->next = head
         if (head == NULL) head = tail = newNode
         else   head -> prev = newNode
                  head = newNode

-> void insertAtEnd (int data) {
        Struct node * new Node {
         newNode-> data = data
         newNode-> prev = tail
         newNode-> next = NULL
         if (tail==NULL)   head = tail = newNode
         else  tail->next = newNode, tail = newNode.
```

```
-> void insertatpos (int data, pos){
          int i,
          struct node * newNode
   case 1:  If pos=1
               insertatfront (data)
   case 2:  for(i=1; i<pos-1 && temp!=NULL, i++)
            temp = temp->next.
              if (temp==NULL || temp->next = NULL)
                    insertAt End(data)
      newNode->next = temp->next
      newNode->prev = temp
      temp->next -> prev = newNode
       temp->next = newNode

-> void deleteAtfront ()
          struct node * temp
           if (head == NULL) printf (empty)
            temp = head
            head = head->next
            if (head != NULL)  head->prev = NULL
             else   tail = null
            free (temp)

-> void deleteAt end()
          struct  node * temp
           if (tail == NULL) print (Empty)
            temp = tail,
            tail = tail-> prev
            if (tail != NULL) tail -> next = null
             else   head = NULL
             free (temp).
```

```
void deletebyvalue (int value) {
    struct node * temp = head
    if (head = = NULL) print( (empty)
     while (temp != NULL && temp -> data != value)
          temp = temp -> next
      if (temp = = NULL ) print( (element not
                                     found)
       if (temp = = head ) deleteAtfront()
        if (temp = = tail ) delete At end ()
       else
            temp -> prev -> next = temp -> next
            temp -> next -> prev = temp -> prev
             free (temp)

void display ()
     struct node * temp = head
      while (temp != tail )
               print (%d , temp -> data)
                temp = temp -> next
        return.
```

prg:

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

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

Page No:
Date.

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

void createList (int n) {
      int i, data;
      struct node * newNode;
      for (i=1; i<=n; i++) {
            printf ("Enter data for node %d," i);
            scanf ("%d", &data);
            newNode = (struct Node *) mallac
                        (size of (struct Node));
            newNode -> data = data;
            newNode -> prev = newNode -> next = NULL;
            if (head == NULL) {
                  head = tail = newNode;
            }
            else {
                  tail -> next = newNode;
                  newNode -> prev = tail;
                  tail = newNode;
            }
      }
}

void insertAtFront (int data) {
      struct Node * newNode = (struct Node *)
                        mall (size of (struct node));
      newNode -> data = data;
      newNode -> prev = NULL;
      newNode -> next = head;
      if (head == NULL)
            head = tail = newNode;
      else {
```

```
    head->prev = newNode;
     head = newNode;
  }
 }printf ("Inserted %d at font\n", data);

void insertAtEnd (int data) {
    Struct Node * newNode = (Struct node*) mallac
                          (siz of (Struct node));
     newNode->data = data;
      newNode->next = NULL;
      newNode->prev = tail;
      if (tail == NULL)
            head = tail = newNode;
      else {
        tail->next = newNode;
         tail = newNode;
      }
     printf ("Inserted %d at end \n", data);
 }

void  insertAtPosition (int data, int pos) {
     int i;
     Struct Node* newNode, *temp = head;
     if (pos == 1) {
          insertAtFront (data);
           return;
      }
     newNode = (struct Node*) mallac (size of
                         (Struct node));
      newNode -> data = data;
      for (i = 1; i < pos - 1 && temp ! =
              NULL ; i++)
```

```
temp = temp-> next;
if (temp == NULL || temp->next = NULL) {
    ~~temp~~ insertAtEnd (data);
        return;
  }
newNode->next = temp-> next;
 newNode-> prev = temp;
 temp-> next -> prev = newNode;
  temp -> next = newNode;
 printf ("Inserted %d at position %d\n",
                        data, pos);
}

void deleteAtFront () {
     Struct Node * temp;
      if (head == NULL) {
            printf ("List is empty\n");
            return;
      }
     temp = head;
     head = head -> next;
     if (head != NULL)
            head -> prev = NULL;
     else
          tail = NULL;
          printf (" Deleted %d at front\n",
                         temp->data);
          free (temp);
  }
```

```
void deleteAtFront () {
    struct Node *temp;
    if (tail == NULL) {
        printf ("List is empty.\n");
        return;
    }
    temp = tail;
    tail = tail->prev;
    if (tail != NULL)
        tail->next = NULL;
    else
        head = NULL;
        printf ("Deleted %d at end\n",
                temp->data);
        free (temp);
}


void deleteByValue (int value) {
    struct Node *temp = head;
    if (head == NULL) {
        printf ("List is empty\n");
        return;
    }
    while (temp != NULL && temp->data != value)
        temp = temp->next;
    if (temp == NULL) {
        printf ("Value not found!\n");
        return;
    }
    if (temp == head)
        deleteAtFront();
```

```
else if (temp == tail)
        deleteAtEnd ();
else {
    temp->prev->next = temp->next;
    temp->next->prev = temp->next;
    printf ("Deleted %d \n", temp->data);
     fee (temp);
  }
}

void search (int value) {
    struct Node *temp = head;
    int pos = 1;
    while (temp != NULL) {
        if (temp->data == value) {
            printf ("Value %d found at
                    position %d\n", value,
                            pos);
            return;
        }
        temp = temp->next;
        pos++;
    }
    printf ("Value %d not found in the
             list \n", value);
}


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


void displayBackward () {
        struct Node *temp = ~~head~~tail;
        printf (" List (Backward): ");
        while (temp != NULL) {
            printf ("%d <-> ", temp->data);
             temp = temp -> prev;
        }
        printf ("NULL\n");
    }


int main () {
    int choice, n ,data, pos, value;

    printf (" Enter no. of nodes to create: ");
     scanf (" %d ", &n);
     createList (n);

    while (1) {
        printf ("\n----DOUBLY LINKED LIST---");
        printf (" 1. Display Forward \n");
        printf (" 2. Display Backward \n");
        printf (" 3. ~~Display~~ Insert at Front \n");
        printf ("4. Insert at End \n");
        printf (" 5. Insert at Position \n");
printf ("6. Delete at Front \n");
printf ("7. Delete at End \n");
printf ("8. Delete by Value \n");
printf ("9. Search \n");
printf ("10. Exit \n");
printf ("Enter your choice : ");
scanf ("%d", &choice);

switch (choice) {

   case 1:
      displayForward();
      break;

   case 2:
      displayBackward();

   case 3:
      printf ("Enter data to insert
                 at Front : ");
      scanf ("%d", &data);
      insertAtFront (data);
      break;

   case 4:
      printf ("Enter data to insert
                 at End : ");
      scanf("%d", &data);
      insertAtEnd (data);
      break;

   case 5:
```

```
printf ("Enter data: ");
 scanf ("%d", &data);
 printf ("Enter position: ");
 scanf ("%d", &pos);
 insertAtPosition (data, pos);
 break;

case 6:
    deleteAtFront ();
    break;

case 7:
    deleteAt~~Front~~End ();
    break;

case 8:
    printf ("Enter value to delete: ");
    scanf ("%d", &value);
    deleteByValue (value);
     break;

case 9:
    printf (" Enter value to search: ");
    scanf ("%d", &value);
    search(value);
    break;

case 10:
    printf ("Exiting program... ");
    exit (0);

default {:
        printf(" Invalid choice! Try agin.\n");
     }
   }
~~print(o~~ return 0;
}.
```

Output :-

```
Enter number of nodes to create : 4
 Enter data for node 1 : 40
  Enter data for node 2 : 20
  Enter data for node 3 : 30
   Enter data for node 4 : 20

----Doubly Linked List Menu----
1. Display forward.
2  Display backward
3. Insert at Front
4.  Insert at End
5.   Insert at Position
6.  Delete at Front
7   Delete at End
8.  Delete by value
9   Search
10.  Exit
Enter your choice : 1
 List (Forward) : 40 <-> 20<-> 30<-> 20 <->NULL

Enter your choice : 2
List (Backward): 20 <-> 30 <-> 20 <-> 40 <-> NULL
```

```
Enter your choice: 3
Enter data to insert at front: 5
 Inserted 5 at Front.

Enter your choice: 4
 Enter data to insert at End: 50
 Inserted 50 at end.

Enter your choice: 5
 Enter data: 4
 Enter position: 2
Inserted 4 at position 2.

Enter your choice: 6
 Deleted 5 at front

Enter your choice: 7
Deleted 50 at end.

 Enter your choice: 8
Enter value to delete: 20
 Deleted 20.

 Enter your choice: 9
Enter value to search: 40
 Value 40 found at position 2.

 Enter your choice: 10
 Exiting program....
```

8/11/25