NAME: PRATEEK SHARMA.

24BBS0057.

Q1.

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 100

int stack[SIZE], top = -1;
void pushElement(int num);
int popElement();
void showStack();

int main() {
    int option, num;
  while (1) {
  printf("\nOptions:\n");
  printf("1. Push\n");
  printf("2. Pop\n");
  printf("3. Show Stack\n");
  printf("4. Quit\n");
  printf("Choose an option: ");
  scanf("%d", &option);
      switch (option) {
           case 1:
            printf("Enter a number to push: ");
            scanf("%d", &num);
            pushElement(num);
           break;
         case 2:
           num = popElement();
          if (num != -1)
          printf("Popped: %d\n", num);
          break;
```

main.c

```c
        case 3:
        showStack();
        break;
          case 4:
          printf("Closing program...\n");
         exit(0);
            default:
           printf("Invalid option! Try again.\n");
        }
    }
    return 0;
}
void pushElement(int num) {
    if (top >= SIZE - 1) {
        printf("Stack Overflow! Cannot add %d.\n", num);
    } else {
        stack[++top] = num;
        printf("%d pushed successfully.\n", num);
    }
}
int popElement() {
    if (top < 0) {
        printf("Stack Underflow! No elements available to pop.\n");
        return -1;
    } else {
        return stack[top--];
    }
}
void showStack() {
    if (top < 0) {
```

main.c Toggle Source

```c
    return 0;
}
void pushElement(int num) {
    if (top >= SIZE - 1) {
        printf("Stack Overflow! Cannot add %d.\n", num);
    } else {
        stack[++top] = num;
        printf("%d pushed successfully.\n", num);
    }
}
int popElement() {
    if (top < 0) {
        printf("Stack Underflow! No elements available to pop.\n");
        return -1;
    } else {
        return stack[top--];
    }
}
void showStack() {
    if (top < 0) {
        printf("Stack is currently empty.\n");
    } else {
        printf("Stack contains: ");
        for (int i = top; i >= 0; i--) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}
```

Test case 1

```
Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 1
Enter a number to push: 21
21 pushed successfully.

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 1
Enter a number to push: 21
21 pushed successfully.

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 3
Stack contains: 21 21

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 2
Popped: 21

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 3
```

```
3. Show Stack
4. Quit
Choose an option: 1
Enter a number to push: 21
21 pushed successfully.

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 3
Stack contains: 21 21

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 2
Popped: 21

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 3
Stack contains: 21

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 4
Closing program...

Process returned 0 (0x0)   execution time : 75.718 s
Press any key to continue.
```

Testcase2:

```
Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 3
Stack is currently empty.

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 1
Enter a number to push: 69
69 pushed successfully.

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 1
Enter a number to push: 420
420 pushed successfully.

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 3
Stack contains: 420 69

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 2
```

```
2. Pop
3. Show Stack
4. Quit
Choose an option: 2
Popped: 420

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 3
Stack contains: 69

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 2
Popped: 69

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 3
Stack is currently empty.

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 4
Closing program...

Process returned 0 (0x0)   execution time : 40.944 s
Press any key to continue.
```

Testcase 3:

```
Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 1
Enter a number to push: 435
435 pushed successfully.

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 1
Enter a number to push: 5645
5645 pushed successfully.

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 1
Enter a number to push: 4677
4677 pushed successfully.

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
Choose an option: 3
Stack contains: 4677 5645 435

Options:
1. Push
2. Pop
3. Show Stack
4. Quit
```

PSEUDOCODE:

Initialize stack as an array of size SIZE(100)

Set top= -1

Function push(val):

If top=SIZE-a:

Print “stack overflow”

Else

++top

Stack[top]=val;

Print “value pushed”

Function pop():

If top= -1

Print “stack underflow”

Return -1

Else:

Value=stack[top]

--top

Return value

Functions display():

If top= -1

Print “stack is empty”

Else

For(i= top,i>=0,i--):

Print stack[i]

Main program:

While:

Print menu options(1:push,2:pop,3:display,4:exit)

Input choice

Switch choice

Case 1:

Input value

Push value

Case 2:

Pop()

Case 3:

Display()

Case 4:

Print “exiting”

Default:

Print “invalid”

Q2.

main.c

```c
#include <stdio.h>
#define MAX_SIZE 100

int queue[MAX_SIZE];
int front = -1, rear = -1;

int is_empty() {
    return front == -1;
}

int is_full() {
    return rear == MAX_SIZE - 1;
}

void enqueue(int value) {
    if (is_full()) {
        printf("Queue Overflow!\n");
    } else {
        if (is_empty()) {
            front = 0;
        }
        rear++;
        queue[rear] = value;
        printf("Enqueued %d\n", value);
    }
}

int dequeue() {
    if (is_empty()) {
        printf("Queue Underflow!\n");
```

main.c

```c
        return -1;
    } else {
        int value = queue[front];
        if (front == rear) {
            front = rear = -1;
        } else {
            front++;
        }
        return value;
    }
}

void display() {
    if (is_empty()) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue elements: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Enqueue\n");
```

main.c

```c
        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 value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                value = dequeue();
                if (value != -1) {
                    printf("Dequeued value: %d\n", value);
                }
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Test case 1:

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 20
Enqueued 20

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 10 20

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 10

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

```
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 10 20

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 10

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 20

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 20

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

Process returned 0 (0x0)   execution time : 27.956 s
Press any key to continue.
```

Testcase2:

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

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 450
Enqueued 450

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 345
Enqueued 345

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 69
Enqueued 69

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
```

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 450 345 69

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 450

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 345 69

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 345

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 69

Menu:
```

```
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 450

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 345 69

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 345

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 69

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

Process returned 0 (0x0)   execution time : 25.499 s
Press any key to continue.
```

Test case3:

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 420
Enqueued 420

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 420

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 420

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 690
Enqueued 690

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

```
Enter your choice: 3
Queue elements: 690

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 420
Enqueued 420

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 690 420

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 690

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 420

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
```

PSEUDOCODE

INIIALIZE queue as an array

Set front =-1,val= -1

Function enqueue(val):

If rear=max-1:

Print “queue overflow”

Else

If front=-1

Front =0

++rear

Queue(val)=val

Print “value enqued”

Function dequeue ()

If front=-1 or front>rear:
print “queue underflowed”

Return -1

Else:

Value=queue[front]

++front

Return value

Function display():

If front ==-1 or front>val

Print “empty”

Else:

For(i=front,i<rear,i++)

Print queue[i]

Main program:

While:

Print menu options(1;enqueue,2:dequeue,3:display,4:exit)

Input choice

Switch choice:

Case 1:

Input value

Enqueue(val)

Case 2:

Dequeue()

Case 3:

Display()

Case 4:

Print “existing”

Default:

Print “invalid”

Q3.

main.c

```c
#include <stdio.h>
#define MAX 100

int queue[MAX];
int front = -1, rear = -1;

void enqueue(int value) {
    if (rear == MAX - 1) {
        printf("Queue Overflow! Cannot enqueue %d.\n", value);
    } else {
        rear = (rear + 1) % MAX;
        if (front == -1) {
            front = rear;
        }
        queue[rear] = value;
        printf("Enqueued %d\n", value);
    }
}

int dequeue() {
    if (front == -1) {
        printf("Queue Underflow! No elements to dequeue.\n");
        return -1;
    } else {
        int value = queue[front];
        if (front == rear) {
            front = rear = -1;
        } else {
            front = (front + 1) % MAX;
        }
```

main.c

```c
        return value;
    }
}

void display() {
    if (front == -1) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue elements: ");
        int i = front;
        do {
            printf("%d ", queue[i]);
            i = (i + 1) % MAX;
        } while (i != (rear + 1) % MAX);
        printf("\n");
    }
}

int main() {
    int choice, value;

    do {
        printf("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);
```

main.c

```c
        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:
                value = dequeue();
                if (value != -1) {
                    printf("Dequeued value: %d\n", value);
                }
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting.\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Test case1:

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to enqueue: 10
Enqueued 10
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to enqueue: 20
Enqueued 20
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue elements: 10 20
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued value: 10
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue elements: 20
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
```

```
Enter your choice: 3
Queue elements: 10 20
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued value: 10
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue elements: 20
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued value: 20
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue is empty.
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 4
Exiting.

Process returned 0 (0x0)   execution time : 17.161 s
Press any key to continue.
```

Test case 2:

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to enqueue: 69
Enqueued 69
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to enqueue: 429
Enqueued 429
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the value to enqueue: 320
Enqueued 320
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue elements: 69 429 320
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued value: 69
Menu:
1. ENQUEUE
2. DEQUEUE
```

```
Enter the value to enqueue: 320
Enqueued 320
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue elements: 69 429 320
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued value: 69
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue elements: 429 320
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued value: 429
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 4
Exiting.

Process returned 0 (0x0)   execution time : 18.883 s
Press any key to continue.
```

Test case 3:

PSEUDOCODE

initialize queue,

front = -1, rear = -1

enqueue(value):

if (rear + 1) % size == front

: print "Queue Overflow"

else: if front == -1:

front = 0 rear = (rear + 1) % size

queue[rear] = value dequeue():

if front == -1 or front > rear:

print "Queue Underflow"

return -1 else:

value = queue[front]

front = (front + 1) % size

if front == rear:

front = rear = -1

return value

display():

if front == -1:

print "Queue is empty"

else:

i = front

while i != rear:

print queue[i],

i = (i + 1) % size

print queue[rear] main():

do: display menu get user choice switch choice:

case 1: get value enqueue(value)

case 2: value = dequeue() if value != -1: print "Dequeued value:", value

case 3: display()

case 4: exit while choice is not 4

Q4.

main.c

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

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

struct Node *head = NULL;

void insertAtBeginning(int val) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = head;
    head = newNode;
}

void insertAtEnd(int val) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        struct Node *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
```

main.c

```
    }
}

void insertAtPosition(int pos, int val) {
    if (pos == 0) {
        insertAtBeginning(val);
    } else {
        struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = val;

        struct Node *temp = head;
        for (int i = 0; i < pos - 1 && temp != NULL; i++) {
            temp = temp->next;
        }

        if (temp == NULL) {
            printf("Invalid position.\n");
        } else {
            newNode->next = temp->next;
            temp->next = newNode;
        }
    }
}

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

main.c

```
        free(temp);
    }
}

void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty.\n");
    } else if (head->next == NULL) {
        free(head);
        head = NULL;
    } else {
        struct Node *temp = head;
        while (temp->next->next != NULL) {
            temp = temp->next;
        }
        free(temp->next);
        temp->next = NULL;
    }
}

void deleteFromPosition(int pos) {
    if (head == NULL) {
        printf("List is empty.\n");
    } else if (pos == 0) {
        deleteFromBeginning();
    } else {
        struct Node *temp = head;
        for (int i = 0; i < pos - 1 && temp != NULL; i++) {
            temp = temp->next;
        }
```

main.c

```

        if (temp == NULL || temp->next == NULL) {
            printf("Invalid position.\n");
        } else {
            struct Node *toDelete = temp->next;
            temp->next = toDelete->next;
            free(toDelete);
        }
    }
}

void search(int val) {
    struct Node *temp = head;
    int pos = 0;

    while (temp != NULL) {
        if (temp->data == val) {
            printf("Value %d found at position %d.\n", val, pos);
            return;
        }
        temp = temp->next;
        pos++;
    }

    printf("Value %d not found in the list.\n", val);
}

void display() {
    struct Node *temp = head;
    printf("List: ");
```

main.c

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

int main() {
    int choice, val, pos;

    do {
        printf("\nMenu:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete from Beginning\n");
        printf("5. Delete from End\n");
        printf("6. Delete from Position\n");
        printf("7. Search\n");
        printf("8. Display\n");
        printf("9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &val);
                insertAtBeginning(val);
                break;
```

main.c

```
            case 2:
                printf("Enter the value to insert: ");
                scanf("%d", &val);
                insertAtEnd(val);
                break;
            case 3:
                printf("Enter the position to insert: ");
                scanf("%d", &pos);
                printf("Enter the value to insert: ");
                scanf("%d", &val);
                insertAtPosition(pos, val);
                break;
            case 4:
                deleteFromBeginning();
                break;
            case 5:
                deleteFromEnd();
                break;
            case 6:
                printf("Enter the position to delete: ");
                scanf("%d", &pos);
                deleteFromPosition(pos);
                break;
            case 7:
                printf("Enter the value to search: ");
                scanf("%d", &val);
                search(val);
                break;
            case 8:
                display();
```

main.c

```
            case 4:
                deleteFromBeginning();
                break;
            case 5:
                deleteFromEnd();
                break;
            case 6:
                printf("Enter the position to delete: ");
                scanf("%d", &pos);
                deleteFromPosition(pos);
                break;
            case 7:
                printf("Enter the value to search: ");
                scanf("%d", &val);
                search(val);
                break;
            case 8:
                display();
                break;
            case 9:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 9);

    return 0;
}
```

Testcase1:

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1
Enter the value to insert: 10

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 2
Enter the value to insert: 20

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 3
Enter the position to insert: 2
Enter the value to insert: 30

Menu:
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 4

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 20 30

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1
Enter the value to insert: 69

Menu:
1. Insert at Beginning
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 7
Enter the value to search: 30
Value 30 found at position 2.

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 69 20 30

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 9
Exiting...

Process returned 0 (0x0)   execution time : 65.154 s
```

Test case 2:

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1
Enter the value to insert: 420

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 2
Enter the value to insert: 690

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 3
Enter the position to insert: 2
Enter the value to insert: 450
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 3
Enter the position to insert: 2
Enter the value to insert: 460

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 420 690 460 450

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 5

Menu:
1. Insert at Beginning
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 420 690 460

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 5

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 420 690

Menu:
1. Insert at Beginning
2. Insert at End
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 5

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 420 690

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 9
Exiting...

Process returned 0 (0x0)   execution time : 77.041 s
Press any key to continue.
```

Test case3:

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1
Enter the value to insert: 1

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1
Enter the value to insert: 2

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 2 1

Menu:
1. Insert at Beginning
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 3
Enter the position to insert: 3
Enter the value to insert: 504
Invalid position.

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 6
Enter the position to delete: 1

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 2
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 3
Enter the position to insert: 3
Enter the value to insert: 2309
Invalid position.

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 2

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 7
Enter the value to search: 2
Value 2 found at position 0.
```

PSEUDOCODE:

Head=null

Function insert b(val)

Create new node

New node.data=val

New node.next=head

Head=new node

Print “value inserted in beginning”

Function insert(val)

Create new node

Newnode.date=value

New node.next=NULL

Head=new node

Else:

Set temp= head

While temp.next !=null’

Temp=temp.next

Temp.next=new node

Print “value inserted at end”

```
Function insert N(val pos)
Create new Node
New Node=value
If position=1
Newnode.next=head
Head=newnode

Else
Set temp=head
For i=1 to position =2
Temp=temp.next
If temp=NULL
Print “insert”
Return

New node.next=temp.next
Temp.net=new node
Print “value inserted”
Function delete():
 If HEAD=NULL;
Print “empty”
Else:
Set temp=head
Head=head.next
Delete temp
Print “delete value from the beginning”

Function deleteb(){
If head=null
Print “list is empty”
If head.next=NULL
```

Delete HEAD

VALUE

Function (search tree(val)):

Set temp=head

While temp.data=value

While{

Temp!=null

}

Return temp=temp.next

Print “Value find a”

Temp=temp.net

Display()

If head=null

Print “list is empty”

Else:

If head !=number

If link temp=null;

While temp=Null

Print “temp.data”

Temp=temp.

}

Main program

while (1)

{ print menu (Insert, Delete, Search, Display and exit)

input choice

switch (choice) {

case 1: input val Insert_A(val)

; case 2: input val Insert_B(val)

case 3: input val, position Insert_C(val, position)

case 4: Delete_A()

case 5: Delete_B()

case 6: input position Delete_position(position)

case 7: input val Search(val) break;

case 8: Display() break; case 9: exit break; default: Print "Invalid choice" } }

Q5.

main.c

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

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

struct Node *head = NULL;

void insertAtBeginning(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}

void insertAtEnd(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        struct Node *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
```

main.c

```c
    }
}

void insertAtPosition(int position, int value) {
    if (position == 0) {
        insertAtBeginning(value);
    } else {
        struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
        newNode->data = value;

        struct Node *temp = head;
        for (int i = 0; i < position - 1 && temp != NULL; i++) {
            temp = temp->next;
        }

        if (temp == NULL) {
            printf("Invalid position.\n");
        } else {
            newNode->next = temp->next;
            temp->next = newNode;
        }
    }
}

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

main.c

```c
        free(temp);
    }
}

void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty.\n");
    } else if (head->next == NULL) {
        free(head);
        head = NULL;
    } else {
        struct Node *temp = head;
        while (temp->next->next != NULL) {
            temp = temp->next;
        }
        free(temp->next);
        temp->next = NULL;
    }
}

void deleteFromPosition(int position) {
    if (head == NULL) {
        printf("List is empty.\n");
    } else if (position == 0) {
        deleteFromBeginning();
    } else {
        struct Node *temp = head;
        for (int i = 0; i < position - 1 && temp != NULL; i++) {
            temp = temp->next;
        }
```

main.c

```
91
92          if (temp == NULL || temp->next == NULL) {
93              printf("Invalid position.\n");
94          } else {
95              struct Node *toDelete = temp->next;
96              temp->next = toDelete->next;
97              free(toDelete);
98          }
99      }
100 }
101
102 void search(int value) {
103     struct Node *temp = head;
104     int position = 0;
105
106     while (temp != NULL) {
107         if (temp->data == value) {
108             printf("Value %d found at position %d.\n", value, position);
109             return;
110         }
111         temp = temp->next;
112         position++;
113     }
114
115     printf("Value %d not found in the list.\n", value);
116 }
117
118 void display() {
119     struct Node *temp = head;
120     printf("List: ");
```

main.c

```
121     while (temp != NULL) {
122         printf("%d ", temp->data);
123         temp = temp->next;
124     }
125     printf("\n");
126 }
127
128 int main() {
129     int choice, value, position;
130
131     do {
132         printf("\nMenu:\n");
133         printf("1. Insert at Beginning\n");
134         printf("2. Insert at End\n");
135         printf("3. Insert at Position\n");
136         printf("4. Delete from Beginning\n");
137         printf("5. Delete from End\n");
138         printf("6. Delete from Position\n");
139         printf("7. Search\n");
140         printf("8. Display\n");
141         printf("9. Exit\n");
142         printf("Enter your choice: ");
143         scanf("%d", &choice);
144
145         switch (choice) {
146             case 1:
147                 printf("Enter the value to insert: ");
148                 scanf("%d", &value);
149                 insertAtBeginning(value);
150                 break;
```

main.c

```
151             case 2:
152                 printf("Enter the value to insert: ");
153                 scanf("%d", &value);
154                 insertAtEnd(value);
155                 break;
156             case 3:
157                 printf("Enter the position to insert: ");
158                 scanf("%d", &position);
159                 printf("Enter the value to insert: ");
160                 scanf("%d", &value);
161                 insertAtPosition(position, value);
162                 break;
163             case 4:
164                 deleteFromBeginning();
165                 break;
166             case 5:
167                 deleteFromEnd();
168                 break;
169             case 6:
170                 printf("Enter the position to delete: ");
171                 scanf("%d", &position);
172                 deleteFromPosition(position);
173                 break;
174             case 7:
175                 printf("Enter the value to search: ");
176                 scanf("%d", &value);
177                 search(value);
178                 break;
179             case 8:
180                 display();
```

main.c

```c
            case 4:
                deleteFromBeginning();
                break;
            case 5:
                deleteFromEnd();
                break;
            case 6:
                printf("Enter the position to delete: ");
                scanf("%d", &position);
                deleteFromPosition(position);
                break;
            case 7:
                printf("Enter the value to search: ");
                scanf("%d", &value);
                search(value);
                break;
            case 8:
                display();
                break;
            case 9:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 9);

    return 0;
}
```

int printf(const char* __restrict__ _Format, ...)

Test 1:

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1
Enter the value to insert: 10

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 2
Enter the value to insert: 20

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 3
Enter the position to insert: 2
Enter the value to insert: 35
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 10 20 35

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 4

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 20 35

Menu:
1. Insert at Beginning
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 4

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 20 35

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 9
Exiting...

Process returned 0 (0x0)   execution time : 231.659 s
Press any key to continue.
```

Test case 2:

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1
Enter the value to insert: 56

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1
Enter the value to insert: 45

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 7
Enter the value to search: 44
Value 44 not found in the list.
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 5

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 6
Enter the position to delete: 1
Invalid position.

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 45

Menu:
```

```
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 6
Enter the position to delete: 1
Invalid position.

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8
List: 45

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 9
Exiting...

Process returned 0 (0x0)   execution time : 58.066 s
Press any key to continue.
```

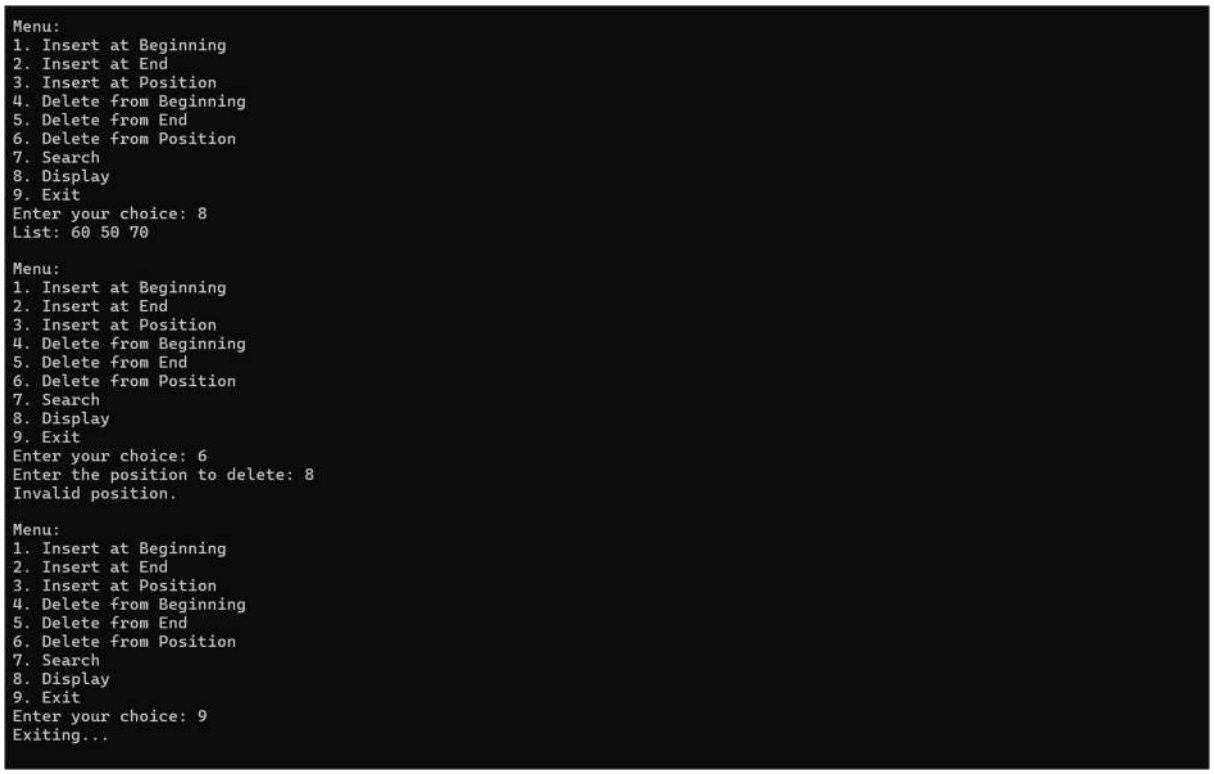
Test case 3:

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 2
Enter the value to insert: 50

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1
Enter the value to insert: 60

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 3
Enter the position to insert: 2
Enter the value to insert: 70

Menu:
```



PSEUDOCODE:

head = NULL

insertAtBeginning(value):

create new node

new_node.data = value

new_node.next = head

head = new_node

```
insertAtEnd(value):
 create new node
 new_node.data = value
 new_node.next = NULL

 if head is NULL:
  head = new_node
 else:
  temp = head
  while temp.next is not NULL:
   temp = temp.next
  temp.next = new_node

insertAtPosition(position, value):
 if position is 0:
  insertAtBeginning(value)
 else:
  create new node
  new_node.data = value

  temp = head
  for i from 0 to position-1:
   if temp is NULL:
    print "Invalid position"
    return
   temp = temp.next

  new_node.next = temp.next
  temp.next = new_node
```

```
deleteFromBeginning():
 if head is NULL:
  print "List is empty"
  return

 temp = head
 head = head.next
 free temp

deleteFromEnd():
 if head is NULL:
  print "List is empty"
  return

 if head.next is NULL:
  free head
  head = NULL
  return

 temp = head
 while temp.next.next is not NULL:
  temp = temp.next
 free temp.next
 temp.next = NULL

deleteFromPosition(position):
 if head is NULL:
  print "List is empty"
  return

 if position is 0:
```

```
    deleteFromBeginning()
    return

  temp = head
  for i from 0 to position-1:
    if temp is NULL or temp.next is NULL:
      print "Invalid position"
      return
    temp = temp.next

  to_delete = temp.next
  temp.next = to_delete.next
  free to_delete

search(value):
  temp = head
  position = 0

  while temp is not NULL:
    if temp.data is equal to value:
      print "Value", value, "found at position", position
      return

    temp = temp.next
    position = position + 1

  print "Value", value, "not found in the list"



display():
  temp = head
```

```
 print "List: "
 while temp is not NULL:
  print temp.data, " "
  temp = temp.next
 print
do:
 display menu with options (insert, delete, search, display, exit)
 get user choice

 switch choice:
  case 1:
   get value
   insertAtBeginning(value)
  case 2:
   get value
   insertAtEnd(value)
  case 3:
   get position
   get value
   insertAtPosition(position, value)
  case 4:
   deleteFromBeginning()
  case 5:
   deleteFromEnd()
  case 6:
   get position
   deleteFromPosition(position)
  case 7:
   get value
   search(value)
  case 8:
```

```
    display()
  case 9:
    exit

while choice is not 9
```