DATA STRUCTURES AND ALGORITHMS

Name = SARTHAK AGGARWAL

Reg no = 24BBS0127

Code = CBS1003

Lab = L55+L56

Assignment-1

1. Write a menu driven program to implement the following operations on stack.

Algorithm :

a. PUSH()

* if ‘top’ == MAX – 1 // (MAX = size of the stack)
* print ‘Stack Overflow’ and exit
* else :
* increment ‘top’ by 1
* Add the element to the stack at the ‘top’ position

b. POP()

* if ‘top’ == -1
* print ‘Stack Underflow’ and exit
* else :
* Retrieve the element at top.
* Decrement top by 1.
* Return the element.

c. Display()

* if ‘top’ == -1
* print ‘Stack is empty’ and exit
* else :
* for i from ‘top’ to 0, print stack[i]

C Code:

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

#define MAX 5

int stack[MAX];
int top = -1;

void PUSH(int element) {
    if (top == MAX - 1) {
        printf("Stack Overflow\n");
    } else {
        top++;
        stack[top] = element;
        printf("Element %d pushed to stack\n", element);
    }
}

int POP() {
    if (top == -1) {
        printf("Stack Underflow\n");
        return -1;
    } else {
        int element = stack[top];
        top--;
        return element;
    }
}

void Display() {
    if (top == -1) {
        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, element;

    while (1) {
        printf("\nMenu:\n");
        printf("1. PUSH\n");
        printf("2. POP\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");

        if (scanf("%d", &choice) != 1) {
```

```
            printf("Invalid input. Exiting...\n");
            while (getchar() != '\n');  // Clear input buffer
            break;  // Exit if invalid input
        }

        switch (choice) {
            case 1:
                printf("Enter the element to push: ");
                if (scanf("%d", &element) != 1) {
                    printf("Invalid input. Exiting...\n");
                    while (getchar() != '\n');  // Clear input buffer
                    return 0;
                }
                PUSH(element);
                break;

            case 2:
                element = POP();
                if (element != -1) {
                    printf("Popped element: %d\n", element);
                }
                break;

            case 3:
                Display();
                break;

            case 4:
                printf("Exiting program...\n");
                return 0;

            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Code

PS C:\Users\sarthak aggarwal\.vscode\programs\.vscode\shadowhawk> cd "c:\Users\sarthak aggarwal\.vscode\programs\.vscode\shadowhawk\dsa\" ; if ($?) { gcc stack.c -o stack } ;
if ($?) { .\stack }
Enter the size of the stack (max 100): 2
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Code

 Stack Operations Menu:
 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT
 Enter your choice: 1
 Enter the element to push: 10
 Element 10 pushed onto the stack.

 Stack Operations Menu:
 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT
 Enter your choice: 1
 Enter the element to push: 20
 Element 20 pushed onto the stack.

 Stack Operations Menu:
 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT
 Enter your choice: 1
 Enter the element to push: 30
 Stack Overflow! Cannot push element.

 Stack Operations Menu:
 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT
 Enter your choice: 2
 Popped element: 20

Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 3
Stack elements: 10


Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 2
Popped element: 20

Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 3
Stack elements: 10

3. DISPLAY
4. EXIT
Enter your choice: 2
Popped element: 20

Stack Operations Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 3
Stack elements: 10

Popped element: 20
```

```
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 3
Stack elements: 10

3. DISPLAY
4. EXIT
Enter your choice: 3
Stack elements: 10

Stack elements: 10


Stack Operations Menu:
1. PUSH
2. POP
1. PUSH
2. POP
3. DISPLAY
2. POP
3. DISPLAY
3. DISPLAY
4. EXIT
Enter your choice: 4
Exiting program.
PS C:\Users\sarthak aggarwal\.vscode\programs\.vscode\shadowhawk\dsa>
```

2. Write a menu driven program to implement the following operations on Queue:

Algorithm :

a. Enqueue()

* if ‘rear’ == MAX – 1
* print ‘ Queue Overflow’ and exit.
* else :
* if the queue is empty (‘front’ == -1), set ‘front’ = 0.
* Increment ‘rear’ by 1.
* Add the element to the queue at the ‘rear’ position.

b. Dequeue()

* if ‘front’ == -1 or ‘front’ > ‘rear’, print “ Queue Underflow” and exit.
* else:
* Retrive the element at ‘front’.
* Increment ‘front’ by 1.
* if ‘front’ > ‘rear’(queue becomes empty), reset ‘front’ = -1 and ‘rear’ = -1
* Return element

c. Display()

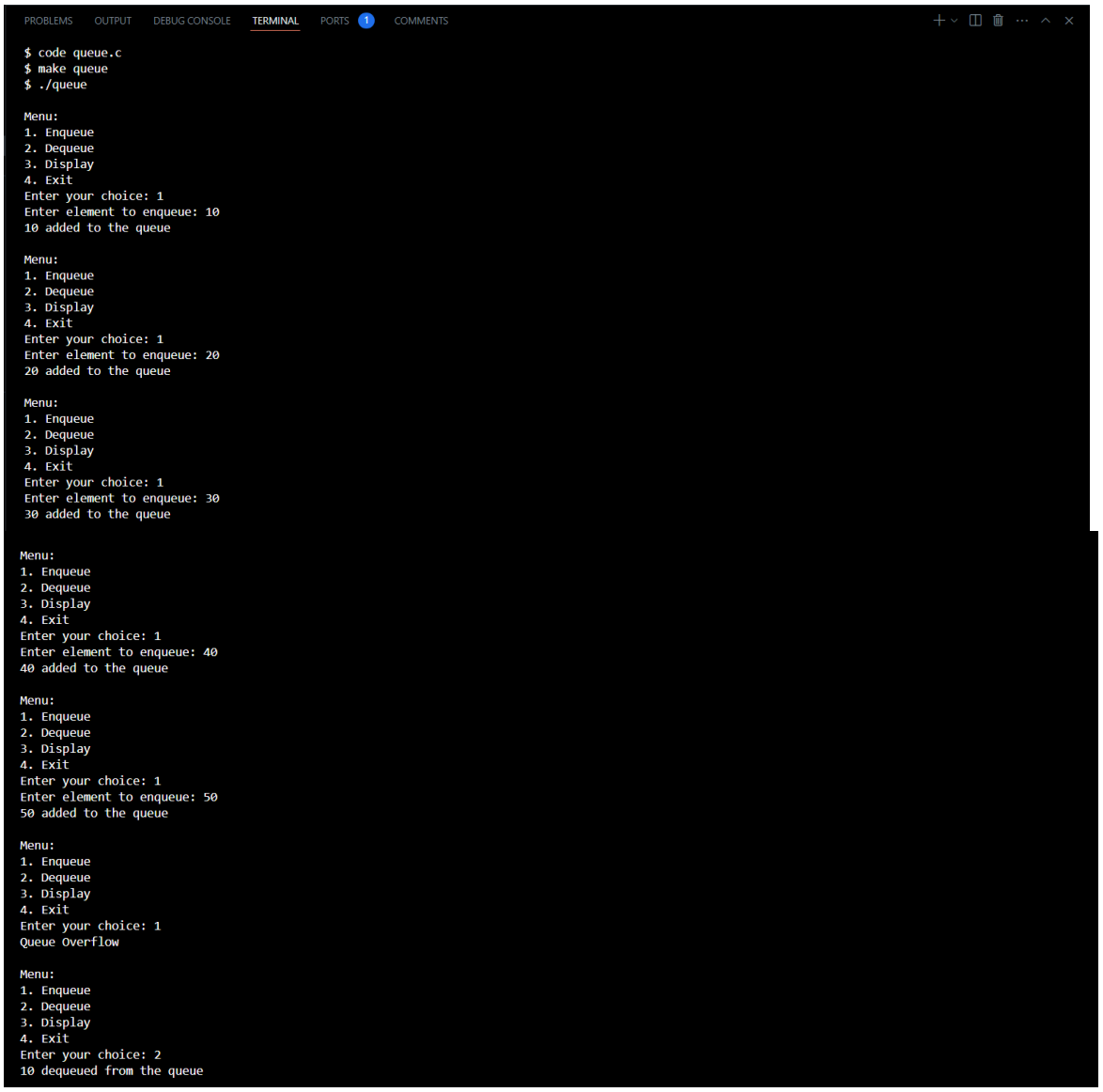
* if ‘front’ == -1, print “Queue is empty”, and exit.
* else:
* for i from ‘front’ to ‘rear’,
* print queue[i].

C Code:

```
#include <stdio.h>
#define MAX 5

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

void enqueue() {
    int value;
    if (rear == MAX - 1) {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1) {
        front = 0;
    }
    printf("Enter element to enqueue: ");
    scanf("%d", &value);
    rear++;
    queue[rear] = value;
    printf("%d added to the queue\n", value);
}

void dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue Underflow\n");
        return;
    }
    int value = queue[front];
    front++;
    if (front > rear) {
        front = rear = -1;
    }
    printf("%d dequeued from the queue\n", value);
}

void display() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

int main() {
    int choice;
    while (1) {
        printf("\nMenu:\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:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice, try again\n");
        }
    }
}
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
$ code queue.c
$ make queue
$ ./queue
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 10
10 added to the queue
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 20
20 added to the queue
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 30
30 added to the queue
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 40
40 added to the queue
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 50
50 added to the queue
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Queue Overflow
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
10 dequeued from the queue

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
20 dequeued from the queue

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
30 dequeued from the queue

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice:
2
40 dequeued from the queue

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 50
```

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 50

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 50

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

3. Write a menu driven program to implement the following operations on circular Queue:

Algorithm :

a. Enqueue()

* if (‘rear’ +1) % MAX == ‘front’, print “ Queue Overflow”(Queue is full) and exit.
* else:
* if the queue is empty(‘front’ == -1), set ‘front’ = 0 and ‘rear’ = 0.
* else:
* set ‘rear’ = (‘rear’ + 1) % MAX
* insert the new element at queue[rear].
*

b. Dequeue()

* if ‘front’ == -1, print “Queue underflow” (Queue is empty) and exit.
* else:
* Retrieve the element at queue[front].
* if ‘front’ == ‘rear’, set ‘front’ = -1 and ‘rear’ = -1 (queue becomes empty).
* else, set ‘front’ = (‘front’ + 1) % MAX.
* return element.

c. Disaply()

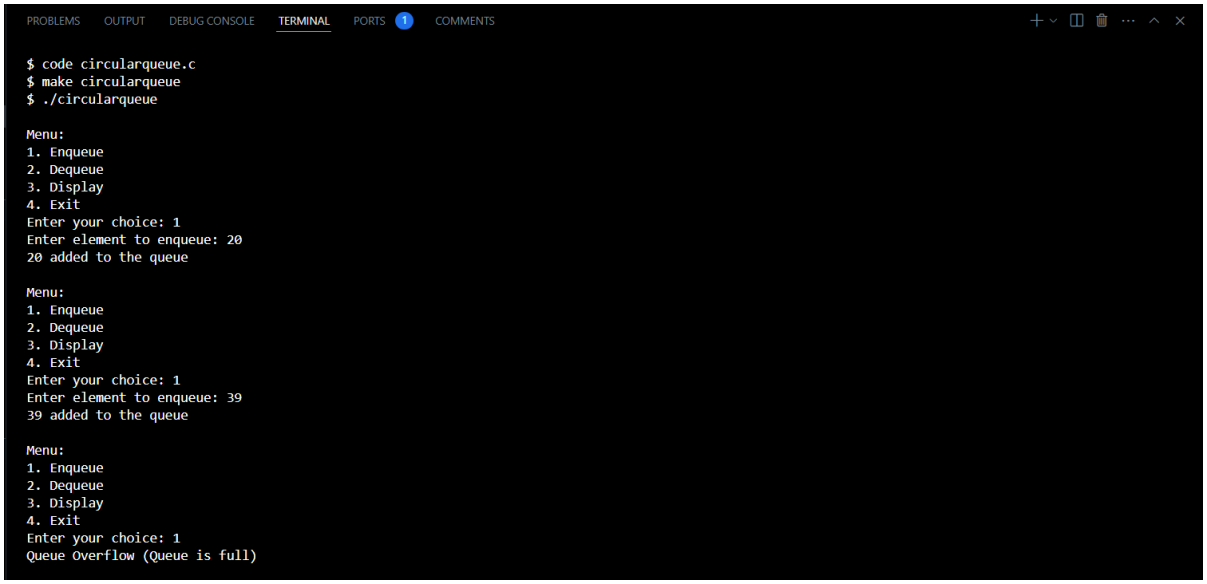
* if ‘front’ == -1, print ‘Queue is empty’ and exit.
* else:
* Start from ‘front’ and move to ‘rear’, wrapping around using % MAX.
* for i = ‘front’ to ‘rear’ (using [(i +1) %MAX])
* print queue[i].

C Code:

```c
#include <stdio.h>
#define MAX 2

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

void enqueue() {
   int value;
   if ((rear + 1) % MAX == front) {
      printf("Queue Overflow (Queue is full)\n");
      return;
   }
   if (front == -1) {
      front = 0;
      rear = 0;
   } else {
      rear = (rear + 1) % MAX;
   }
   printf("Enter element to enqueue: ");
   scanf("%d", &value);
   queue[rear] = value;
   printf("%d added to the queue\n", value);
}

void dequeue() {
   if (front == -1) {
      printf("Queue Underflow (Queue is empty)\n");
      return;
   }
   int value = queue[front];
   if (front == rear) {
      front = rear = -1; // Queue becomes empty
   } else {
      front = (front + 1) % MAX;
   }
   printf("%d dequeued from the queue\n", value);
}

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

int main() {
   int choice;
   while (1) {
```

```
        printf("\nMenu:\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:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice, try again\n");
        }
    }
}
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
$ code circularqueue.c
$ make circularqueue
$ ./circularqueue
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 20
20 added to the queue
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter element to enqueue: 39
39 added to the queue
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Queue Overflow (Queue is full)


Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
20 dequeued from the queue
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 39
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
$

4. Write a menu driven program to implement the following operations on singly linked list:

Algorithm

a. Insertion()

i. Beginning

```
Create a new node  'newNode'.
Assign the data to 'newNode' -> data.
Set newNode-> 'next' = 'head'.                //(current head of the list)
Update 'head' = 'newNode'.
exit.
```

ii. End:

```
create a new node newNode.
Set newNode -> data = value.
Set newNode -> next = NULL.
If 'head' == NULL:
        Set 'head' = newNode (list is empty)
        End.
Else:
        initialize 'temp' = 'head'.
        while temp-> next != NULL:
                Move temp = temp -> next.
        Set temp-> next = newNode      //(linked last node to new node)
```

iii. At a given position:

```
Create a new node newNode.
Set newNode->data = value.
 If position == 1:
     Set newNode->next = head.
     Update head = newNode.
     End.
Else:
      Initialize temp = head.
      For i = 1 to position - 1:
          If temp == NULL:
             Print "Invalid position".
             End.
          Move temp = temp->next.
       Set newNode->next = temp->next.
      Set temp->next = newNode.
 End.
```

b. deletion ()

i) Beginning

```
If head == NULL:
Print "List is empty".
End.
Set temp = head.
Update head = head->next.
Free(temp).
        End.
```

ii) At the end

```
If head == NULL:
    Print "List is empty".
    End.
If head->next == NULL:
    Set temp = head.
    Update head = NULL.
    Free(temp).
    End.
Initialize temp = head.
While temp->next != NULL:
    Set prev = temp.
    Move temp = temp->next.
Set prev->next = NULL.
Free(temp).
        End.
```

iii) At a given position

```
If head == NULL:
    Print "List is empty".
    End.
If position == 1:
    Set temp = head.
    Update head = head->next.
    Free(temp).
    End.
Initialize temp = head.
For i = 1 to position - 1:
    If temp == NULL or temp->next == NULL:
        Print "Invalid position".
        End.
        Move temp = temp->next.
Set toDelete = temp->next.
Update temp->next = temp->next->next.
Free(toDelete).
End.
```

c) Search(): search for the given element on the list

```
If head == NULL:
        Print "List is empty".
        End.
Initialize temp = head.
While temp != NULL:
        If temp->data == value:
Print "Element found".
                                        ii. End.
        Move temp = temp->next.
If temp == NULL:
        Print "Element not found".
End.
```

```
        Create a new node newNode.
        Set newNode->data = value.
        Set newNode->next = head.
        If head != NULL:
        Set head->prev = newNode.
Set head = newNode.
End.
```

C code:

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

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

struct Node* head = NULL;

void insertion_beginning() {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data: ");
    scanf("%d", &newNode->data);
    newNode->next = head;
    head = newNode;
    printf("Node inserted at the beginning\n");
}

void insertion_end() {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data: ");
    scanf("%d", &newNode->data);
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("Node inserted at the end\n");
}

void insertion_at_position() {
    int position, value;
    printf("Enter position: ");
    scanf("%d", &position);
    printf("Enter data: ");
    scanf("%d", &value);

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    if (position == 1) {
        newNode->next = head;
        head = newNode;
        printf("Node inserted at position 1\n");
    } else {
```

```
        struct Node* temp = head;
        for (int i = 1; i < position - 1; i++) {
            if (temp == NULL) {
                printf("Invalid position\n");
                return;
            }
            temp = temp->next;
        }
        newNode->next = temp->next;
        temp->next = newNode;
        printf("Node inserted at position %d\n", position);
    }
}

void deletion_beginning() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
    printf("Node deleted from the beginning\n");
}

void deletion_end() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    if (head->next == NULL) {
        free(head);
        head = NULL;
        printf("Node deleted from the end\n");
        return;
    }
    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    free(temp);
    printf("Node deleted from the end\n");
}

void deletion_at_position() {
    int position;
    printf("Enter position: ");
    scanf("%d", &position);

    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
```

```
    if (position == 1) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
        printf("Node deleted from position 1\n");
        return;
    }

    struct Node* temp = head;
    for (int i = 1; i < position - 1; i++) {
        if (temp == NULL || temp->next == NULL) {
            printf("Invalid position\n");
            return;
        }
        temp = temp->next;
    }

    struct Node* toDelete = temp->next;
    temp->next = temp->next->next;
    free(toDelete);
    printf("Node deleted from position %d\n", position);
}

void search() {
    int value;
    printf("Enter value to search: ");
    scanf("%d", &value);

    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = head;
    while (temp != NULL) {
        if (temp->data == value) {
            printf("Element %d found\n", value);
            return;
        }
        temp = temp->next;
    }
    printf("Element %d not found\n", value);
}

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

```
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice;
    while (1) {
        printf("\nMenu:\n");
        printf("1. Insertion at Beginning\n");
        printf("2. Insertion at End\n");
        printf("3. Insertion at Given Position\n");
        printf("4. Deletion at Beginning\n");
        printf("5. Deletion at End\n");
        printf("6. Deletion at Given Position\n");
        printf("7. Search an Element\n");
        printf("8. Display List\n");
        printf("9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: insertion_beginning(); break;
            case 2: insertion_end(); break;
            case 3: insertion_at_position(); break;
            case 4: deletion_beginning(); break;
            case 5: deletion_end(); break;
            case 6: deletion_at_position(); break;
            case 7: search(); break;
            case 8: display(); break;
            case 9: exit(0);
            default: printf("Invalid choice, try again.\n");
        }
    }
    return 0;
          }
```

Output :

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1 COMMENTS

```
$ ./singlylist

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 1
Enter data: 10
Node inserted at the beginning

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 2
Enter data: 30
Node inserted at the end

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 3
Enter position: 2
Enter data: 10
Node inserted at position 2

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 7
Enter value to search: 20
Element 20 not found

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 6
Enter position: 2
Node deleted from position 2

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 4
Node deleted from the beginning
```

```
Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 8
List elements: 30

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 5
Node deleted from the end

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 8
List is empty

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 9
$
```

5. Write a menu driven program to implement the following operations on Doubly linked list: (algorithm)

a. Insertion()

i. Beginning

```
Create a new node newNode.
Set newNode->data = value.
Set newNode->next = head.
If head != NULL:
    Set head->prev = newNode.
Set head = newNode.
End.
```

ii. End

```
Create a new node newNode.
Set newNode->data = value.
Set newNode->next = NULL.
If head == NULL:
        Set head = newNode.
        End.
Else:
        Initialize temp = head.
        While temp->next != NULL:
Move temp = temp->next.
        Set temp->next = newNode.
        Set newNode->prev = temp.
End.
```

iii) Insertion at a Given Position

```
Create a new node newNode.
Set newNode->data = value.
If position == 1:
       Set newNode->next = head.
        If head != NULL:
              Set head->prev = newNode.
         Update head = newNode.
          End.
Else:
     Initialize temp = head.
     For i = 1 to position - 1:
           If temp == NULL:
           Print "Invalid position".
           End.
        Move temp = temp->next.
 Set newNode->next = temp->next.
  If temp->next != NULL:
            Set temp->next->prev = newNode.
   Set temp->next = newNode.
  Set newNode->prev = temp.
  End.
```

b) Deletion():

i) Beginning

```
If head is NULL (list is empty)
   Print "List is empty"
   End
Set temp = head
Update head = head->next
If head is not NULL
   Set head->prev = NULL
Free temp
        End
```

ii) End

```
If head is NULL (list is empty)
   Print "List is empty"
   End
If head->next is NULL (only one node in the list)
   Set temp = head
   Update head = NULL
Else
   Set temp = head
   While temp->next != NULL
      Move temp = temp->next
   End
   Set temp->prev->next = NULL
Free temp
End
```

iii) At any given position

```
If head is NULL (list is empty)
   Print "List is empty"
   End
If position is 1
   Set temp = head
   Update head = head->next
   If head is not NULL
      Set head->prev = NULL
Else
   Set temp = head
   For i = 1 to position - 1
      If temp is NULL or temp->next is NULL
         Print "Invalid position"
         End
      Move temp = temp->next
   End
   Set toDelete = temp->next
   Update temp->next = temp->next->next
   If temp->next != NULL
      Set temp->next->prev = temp
   Free toDelete
End
```

c) Search(): search for the given element on the list

```
If head is NULL (list is empty)
   Print "List is empty"
   End
Set temp = head
While temp is not NULL
   If temp->data == value
      Print "Element found"
      End
   Move temp = temp->next
End
Print "Element not found"
End
```

C code :

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

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

struct Node* head = NULL;

void insertion_beginning() {
    int value;
    printf("Enter data: ");
    scanf("%d", &value);

    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = value;
    newNode->next = head;
    newNode->prev = NULL;

    if (head != NULL) {
        head->prev = newNode;
    }

    head = newNode;
    printf("Node inserted at the beginning\n");
}

void insertion_end() {
    int value;
    printf("Enter data: ");
    scanf("%d", &value);

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

    if (head == NULL) {
        newNode->prev = NULL;
        head = newNode;
        printf("Node inserted at the end\n");
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
        printf("Node inserted at the end\n");
    }
```

```
}

void insertion_at_position() {
    int value, position;
    printf("Enter position: ");
    scanf("%d", &position);
    printf("Enter data: ");
    scanf("%d", &value);

    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = value;

    if (position == 1) {
        newNode->next = head;
        newNode->prev = NULL;
        if (head != NULL) {
            head->prev = newNode;
        }
        head = newNode;
        printf("Node inserted at position 1\n");
    } else {
        struct Node* temp = head;
        for (int i = 1; i < position - 1; i++) {
            if (temp == NULL) {
                printf("Invalid position\n");
                return;
            }
            temp = temp->next;
        }

        newNode->next = temp->next;
        if (temp->next != NULL) {
            temp->next->prev = newNode;
        }
        temp->next = newNode;
        newNode->prev = temp;
        printf("Node inserted at position %d\n", position);
    }
}

void deletion_beginning() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = head;
    head = head->next;

    if (head != NULL) {
        head->prev = NULL;
    }
```

```
    free(temp);
    printf("Node deleted from the beginning\n");
}

void deletion_end() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    if (head->next == NULL) {
        free(head);
        head = NULL;
        printf("Node deleted from the end\n");
        return;
    }

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

    temp->prev->next = NULL;
    free(temp);
    printf("Node deleted from the end\n");
}

void deletion_at_position() {
    int position;
    printf("Enter position: ");
    scanf("%d", &position);

    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    if (position == 1) {
        struct Node* temp = head;
        head = head->next;

        if (head != NULL) {
            head->prev = NULL;
        }

        free(temp);
        printf("Node deleted from position 1\n");
        return;
    }

    struct Node* temp = head;
    for (int i = 1; i < position - 1; i++) {
```

```
        if (temp == NULL || temp->next == NULL) {
            printf("Invalid position\n");
            return;
        }
        temp = temp->next;
    }

    struct Node* toDelete = temp->next;
    temp->next = temp->next->next;

    if (temp->next != NULL) {
        temp->next->prev = temp;
    }

    free(toDelete);
    printf("Node deleted from position %d\n", position);
}

void search() {
    int value;
    printf("Enter value to search: ");
    scanf("%d", &value);

    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = head;
    while (temp != NULL) {
        if (temp->data == value) {
            printf("Element %d found\n", value);
            return;
        }
        temp = temp->next;
    }

    printf("Element %d not found\n", value);
}

void display() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

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

```
}

int main() {
   int choice;
   while (1) {
      printf("\nMenu:\n");
      printf("1. Insertion at Beginning\n");
      printf("2. Insertion at End\n");
      printf("3. Insertion at Given Position\n");
      printf("4. Deletion at Beginning\n");
      printf("5. Deletion at End\n");
      printf("6. Deletion at Given Position\n");
      printf("7. Search an Element\n");
      printf("8. Display List\n");
      printf("9. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1: insertion_beginning(); break;
         case 2: insertion_end(); break;
         case 3: insertion_at_position(); break;
         case 4: deletion_beginning(); break;
         case 5: deletion_end(); break;
         case 6: deletion_at_position(); break;
         case 7: search(); break;
         case 8: display(); break;
         case 9: exit(0);
         default: printf("Invalid choice, try again.\n");
      }
   }
   return 0;
}
```

Output:

```
$ code doublylist.c
$ make doublylist
$ ./doublylist

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 1
Enter data: 10
Node inserted at the beginning

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 2
Enter data: 30
Node inserted at the end

 Menu:
 1. Insertion at Beginning
 2. Insertion at End
 3. Insertion at Given Position
 4. Deletion at Beginning
 5. Deletion at End
 6. Deletion at Given Position
 7. Search an Element
 8. Display List
 9. Exit
 Enter your choice: 3
 Enter position: 2
 Enter data: 20
 Node inserted at position 2

 Menu:
 1. Insertion at Beginning
 2. Insertion at End
 3. Insertion at Given Position
 4. Deletion at Beginning
 5. Deletion at End
 6. Deletion at Given Position
 7. Search an Element
 8. Display List
 9. Exit
 Enter your choice: 8
 List elements: 10 20 30
```

```
 Menu:
 1. Insertion at Beginning
 2. Insertion at End
 3. Insertion at Given Position
 4. Deletion at Beginning
 5. Deletion at End
 6. Deletion at Given Position
 7. Search an Element
 8. Display List
 9. Exit
 Enter your choice: 7
 Enter value to search: 30
 Element 30 found

 Menu:
 1. Insertion at Beginning
 2. Insertion at End
 3. Insertion at Given Position
 4. Deletion at Beginning
 5. Deletion at End
 6. Deletion at Given Position
 7. Search an Element
 8. Display List
 9. Exit
 Enter your choice: 6
 Enter position: 2
 Node deleted from position 2
```

```
 Menu:
 1. Insertion at Beginning
 2. Insertion at End
 3. Insertion at Given Position
 4. Deletion at Beginning
 5. Deletion at End
 6. Deletion at Given Position
 7. Search an Element
 8. Display List
 9. Exit
 Enter your choice: 4
 Node deleted from the beginning

 Menu:
 1. Insertion at Beginning
 2. Insertion at End
 3. Insertion at Given Position
 4. Deletion at Beginning
 5. Deletion at End
 6. Deletion at Given Position
 7. Search an Element
 8. Display List
 9. Exit
 Enter your choice: 5
 Node deleted from the end

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 7
Enter value to search: 20
List is empty

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 8
List is empty

Menu:
1. Insertion at Beginning
2. Insertion at End
3. Insertion at Given Position
4. Deletion at Beginning
5. Deletion at End
6. Deletion at Given Position
7. Search an Element
8. Display List
9. Exit
Enter your choice: 9
$
```