VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Arjun Mallikarjun Banappanavar

(1BM24CS052)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
August-December 2025

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by Arjun Mallikarjun Banappanavar **(1BM24CS052)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2025-2026. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)** work prescribed for the said degree.

Ms. Manjula S
Assistant Professor
Department of ISE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Program 1:

Write a program to simulate the working of stack using an array with the following:

1) **a) Push**
2) **b) Pop**
3) **c) Display**

The program should print appropriate messages for stack overflow, stack underflow.

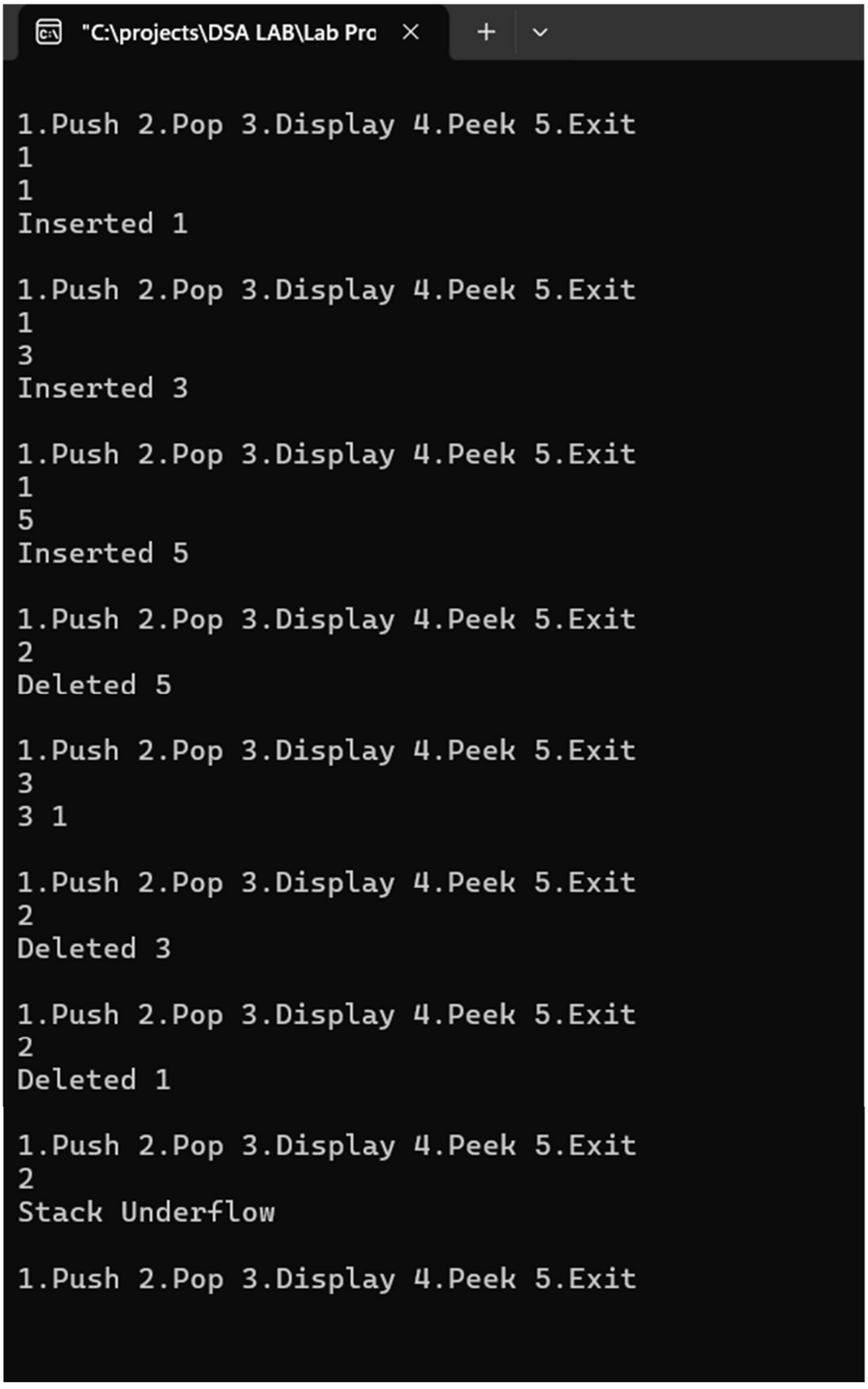
Code:

```
#include <stdio.h>

#include <stdlib.h>


#define SIZE 10


int stack[SIZE];

int top = -1;


void push(int value) {
    if (top == SIZE - 1) {
        printf("Stack Overflow\n");
    } else {
        stack[++top] = value;
        printf("Inserted %d\n", value);
    }
}


void pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
    } else {
        printf("Deleted %d\n", stack[top--]);
    }
}
```

```
void display() {
    if (top == -1) {
        printf("Stack is Empty\n");
    } else {
        for (int i = top; i >= 0; i--) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

void peek() {
    if (top == -1) {
        printf("Stack is Empty\n");
    } else {
        printf("Top element is %d\n", stack[top]);
    }
}

int main() {
    int choice, value;
    do {
        printf("\n1.Push 2.Pop 3.Display 4.Peek 5.Exit\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf("%d", &value);
                push(value);
                break;
```

```
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                peek();
                break;
            case 5:
                break;
            default:
                printf("Invalid Choice\n");
        }
    }while(choice != 5);
    return 0;
}
```

Output:


"C:\projects\DSA LAB\Lab Pro
1.Push 2.Pop 3.Display 4.Peek 5.Exit
1
1
Inserted 1
1.Push 2.Pop 3.Display 4.Peek 5.Exit
1
3
Inserted 3
1.Push 2.Pop 3.Display 4.Peek 5.Exit
1
5
Inserted 5
1.Push 2.Pop 3.Display 4.Peek 5.Exit
2
Deleted 5
1.Push 2.Pop 3.Display 4.Peek 5.Exit
3
3 1
1.Push 2.Pop 3.Display 4.Peek 5.Exit
2
Deleted 3
1.Push 2.Pop 3.Display 4.Peek 5.Exit
2
Deleted 1
1.Push 2.Pop 3.Display 4.Peek 5.Exit
2
Stack Underflow
1.Push 2.Pop 3.Display 4.Peek 5.Exit

Program 2:

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

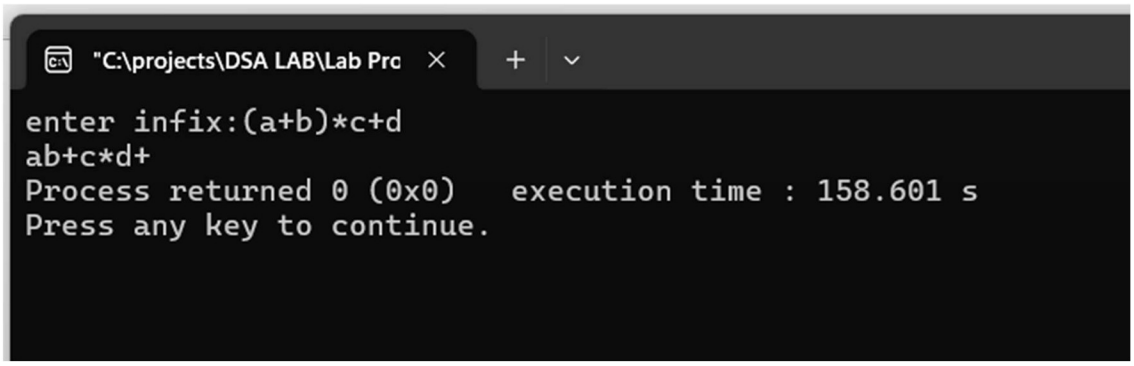
Code:

```
#include <stdio.h>

#include  <ctype.h>


#define SIZE 50


char stack[SIZE];

int top = -1;


void push(char x) {

   stack[++top] = x;

}


char pop() {

   return  stack[top--];

}


int pr(char x) {

   if (x == '+' || x == '-')

      return 1;

   if (x == '*' || x == '/')

      return 2;

   return 0;

}
```

```
int main() {
    char infix[50], postfix[50], ch;
    int i = 0, k = 0;
    printf("enter infix:");
    scanf("%s", infix);
    push('#');

    while ((ch = infix[i++]) != '\0') {
        if (ch == '(')
            push(ch);
        else if (isalnum(ch))
            postfix[k++] = ch;
        else if (ch == ')') {
            while (stack[top] != '('
                postfix[k++] = pop();
            pop();
        } else {
            while (pr(stack[top]) >= pr(ch))
                postfix[k++] = pop();
            push(ch);
        }
    }

    while (stack[top] != '#')
        postfix[k++] = pop();

    postfix[k] = '\0';
    printf("%s", postfix);
```

```
    return 0;
}
```

Output:



Program 3a:

a) **WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions.**

Code:

```
#include <stdio.h>

#define MAX 50


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


void insert() {
   int x;
   if (rear == MAX - 1) {
      printf("Queue Overflow\n");
      return;
   }
   if (front == -1)
      front = 0;
   printf("Enter element: ");
   scanf("%d", &x);
   q[++rear] = x;
   printf("Inserted %d\n", x);
}


void delete() {
   if (front == -1 || front > rear) {
```

```
        printf("Queue Underflow\n");
        return;
    }
    printf("Deleted element: %d\n", q[front]);
    front++;
}

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

int main() {
    int ch;
    while (1) {
        printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &ch);
```

```
        if (ch == 1)
            insert();
        else if (ch == 2)
            delete();
        else if (ch == 3)
            display();
        else if (ch == 4)
            break;
        else
            printf("Invalid choice\n");
    }
    return 0;
}
```

Output:

```
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 1
Enter element: 110
Inserted 110

1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 1
Enter element: 20
Inserted 20

1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 1
Enter element: 30
Inserted 30

1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 3
Queue elements: 110 20 30

1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 2
Deleted element: 110
```

```
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 2
Deleted element: 20

1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 2
Deleted element: 30

1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 2
Queue Underflow

1.Insert
2.Delete
3.Display
4.Exit
Enter choice:
```

Program 3b:

b) **WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions**

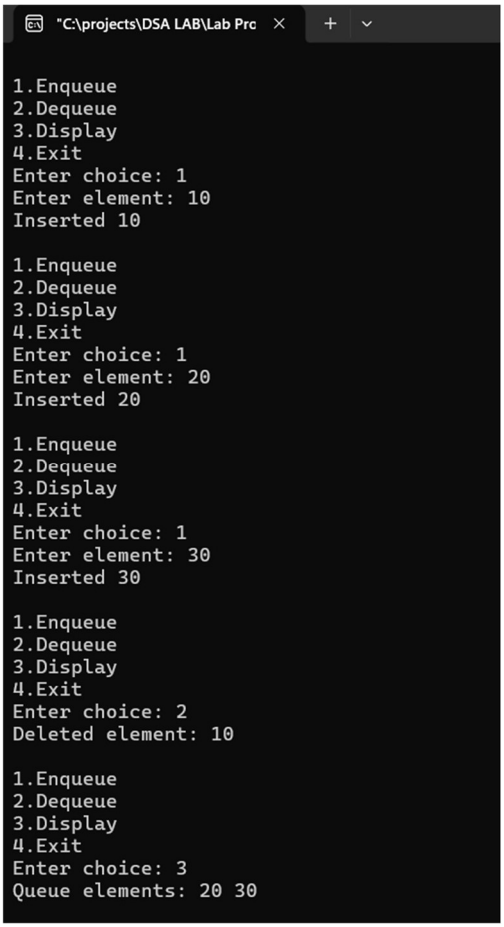
Code:

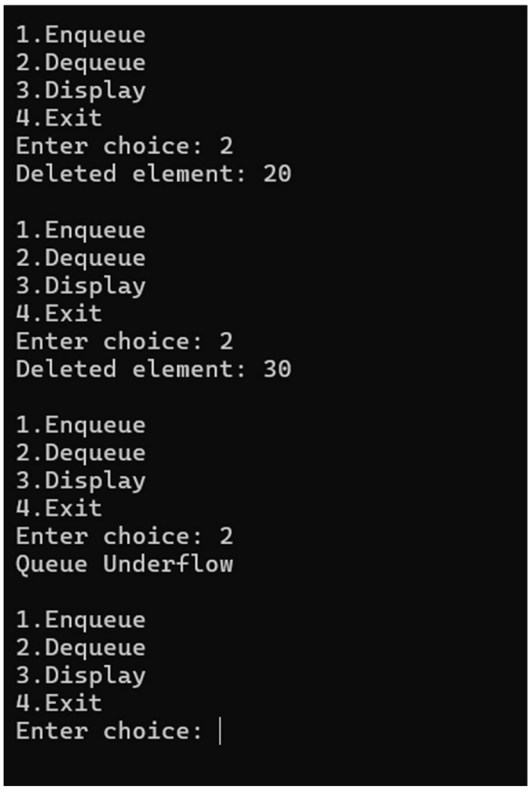
```
#include <stdio.h>

#define SIZE 5


int q[SIZE], front = -1, rear = -1;


int isFull() {
   return (front == (rear + 1) % SIZE);
}


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


void enqueue(int x) {
   if (isFull()) {
      printf("Queue Overflow\n");
      return;
   }
   if (front == -1)
      front = 0;
   rear = (rear + 1) % SIZE;
   q[rear] = x;
   printf("Inserted %d\n", x);
}


void dequeue() {
```

```
    if (isEmpty()) {
        printf("Queue Underflow\n");
        return;
    }
    printf("Deleted element: %d\n", q[front]);
    if (front == rear)
        front = rear = -1;
    else
        front = (front + 1) % SIZE;
}

void display() {
    int i;
    if (isEmpty()) {
        printf("Queue is Empty\n");
        return;
    }
    printf("Queue elements: ");
    for (i = front; i != rear; i = (i + 1) % SIZE)
        printf("%d ", q[i]);
    printf("%d\n", q[i]);
}

int main() {
    int ch, x;
    while (1) {
        printf("\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &ch);

        if (ch == 1) {
```

```
            printf("Enter element: ");
            scanf("%d", &x);
            enqueue(x);
        }
        else if (ch == 2)
            dequeue();
        else if (ch == 3)
            display();
        else if (ch == 4)
            break;
        else
            printf("Invalid choice\n");
    }
    return 0;
}
```

Output:


"C:\projects\DSA LAB\Lab Prc
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 1
Enter element: 10
Inserted 10
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 1
Enter element: 20
Inserted 20
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 1
Enter element: 30
Inserted 30
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 2
Deleted element: 10
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 3
Queue elements: 20 30


1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 2
Deleted element: 20
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 2
Deleted element: 30
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 2
Queue Underflow
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice:

Program 4:

WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list

Code:

```
#include <stdio.h>

#include <stdlib.h>


struct node {

   int data;

   struct node *next;

};


struct node *head = NULL;


struct node* create(int x) {

   struct node *n = malloc(sizeof(struct node));

   if (!n) exit(1);

   n->data = x;

   n->next = NULL;

   return n;

}


int count() {

   int c = 0;

   struct node *t = head;

   while (t) {

      c++;

      t = t->next;
```

```
    }
    return c;
}

void insert_begin(int x) {
    struct node *n = create(x);
    n->next = head;
    head = n;
    printf("Inserted %d at beginning\n", x);
}

void insert_end(int x) {
    struct node *n = create(x), *t = head;
    if (!head) {
        head = n;
        printf("Inserted %d as first node\n", x);
        return;
    }
    while (t->next) t = t->next;
    t->next = n;
    printf("Inserted %d at end\n", x);
}

void insert_pos(int x, int pos) {
    int i;
    struct node *t = head;

    if (pos < 1 || pos > count() + 1) {
        printf("Invalid position\n");
```

```
        return;
    }

    if (pos == 1) {
        insert_begin(x);
        return;
    }

    for (i = 1; i < pos - 1; i++)
        t = t->next;

    struct node *n = create(x);
    n->next = t->next;
    t->next = n;

    printf("Inserted %d at position %d\n", x, pos);
}

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

```
    printf("\n");
}

int main() {
    int ch, x, pos;
    while (1) {
        printf("\n1.Insert Begin\n2.Insert End\n3.Insert Position\n4.Display\n5.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &ch);

        if (ch == 1) {
            scanf("%d", &x);
            insert_begin(x);
        }
        else if (ch == 2) {
            scanf("%d", &x);
            insert_end(x);
        }
        else if (ch == 3) {
            scanf("%d %d", &x, &pos);
            insert_pos(x, pos);
        }
        else if (ch == 4)
            display();
        else if (ch == 5)
            break;
    }
    return 0;
}
```

Output:


"C:\projects\DSA LAB\Lab Prc
1.Insert Begin
2.Insert End
3.Insert Position
4.Display
5.Exit
Enter choice: 1
10
Inserted 10 at beginning
1.Insert Begin
2.Insert End
3.Insert Position
4.Display
5.Exit
Enter choice: 2
30
Inserted 30 at end
1.Insert Begin
2.Insert End
3.Insert Position
4.Display
5.Exit
Enter choice: 2
40
Inserted 40 at end
1.Insert Begin
2.Insert End
3.Insert Position
4.Display
5.Exit
Enter choice: 3
3
20
Invalid position
1.Insert Begin
2.Insert End
3.Insert Position
4.Display
5.Exit
Enter choice: 3
2
20
Invalid position
1.Insert Begin
2.Insert End
3.Insert Position
4.Display
5.Exit
Enter choice: 4
List: 10 30 40

Program 5:

WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

Code:

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

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

struct node *head = NULL;

struct node* create(int x) {
   struct node *n = malloc(sizeof(struct node));
   if (!n) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   n->data = x;
   n->next = NULL;
   return n;
}

void insert_begin(int x) {
   struct node *n = create(x);
   n->next = head;  head = n;
```

```
    printf("Inserted %d at beginning\n", x);
}

void delete_begin() {
    if (!head) {
        printf("List Underflow\n");
        return;
    }
    struct node *t = head;
    printf("Deleted %d from beginning\n", t->data);
    head = head->next;
    free(t);
}

void delete_end() {
    struct node *t = head, *p = NULL;
    if (!head) {
        printf("List Underflow\n");
        return;
    }
    if (!head->next) {
        printf("Deleted %d from end\n", head->data);
        free(head);
        head = NULL;
        return;
    }
    while (t->next) {
        p = t;
        t = t->next;
    }
    p->next = NULL;
```

```
    printf("Deleted %d from end\n", t->data);
    free(t);
}

void delete_key(int key) {
    struct node *t = head, *p = NULL;
    if (!head) {
        printf("List Underflow\n");
        return;
    }
    if (head->data == key) {
        delete_begin();
        return;
    }
    while (t && t->data != key) {
        p = t;
        t = t->next;
    }
    if (!t) {
        printf("Element %d not found\n", key);
        return;
    }
    p->next = t->next;
    printf("Deleted element %d\n", key);
    free(t);
}

void display() {
    struct node *t = head;
    if (!t) {
        printf("List is empty\n");
```

```
        return;
    }
    printf("List elements: ");
    while (t) {
        printf("%d ", t->data);
        t = t->next;
    }
    printf("\n");
}

int main() {
    int ch, x;
    while (1) {
        printf("\n1.Insert Begin\n2.Delete Begin\n3.Delete End\n4.Delete Key\n5.Display\n6.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &ch);

        if (ch == 1) {
            printf("Enter value: ");
            scanf("%d", &x);
            insert_begin(x);
        }
        else if (ch == 2)
            delete_begin();
         else if (ch == 3)
             delete_end();
        else if (ch == 4) {
            printf("Enter key: ");
            scanf("%d", &x);
            delete_key(x);
        }
```

```
        else if (ch == 5)
            display();
        else if (ch == 6)
            break;
        else
            printf("Invalid choice\n");
    }
    return 0;
}
```

Output:

```
1.Insert Begin
2.Delete Begin
3.Delete End
4.Delete Key
5.Display
6.Exit
Enter choice: 1
Enter value: 10
Inserted 10 at beginning

1.Insert Begin
2.Delete Begin
3.Delete End
4.Delete Key
5.Display
6.Exit
Enter choice: 1
Enter value: 20
Inserted 20 at beginning

1.Insert Begin
2.Delete Begin
3.Delete End
4.Delete Key
5.Display
6.Exit
Enter choice: 1
Enter value: 30
Inserted 30 at beginning

1.Insert Begin
2.Delete Begin
3.Delete End
4.Delete Key
5.Display
6.Exit
Enter choice: 2
Deleted 30 from beginning

1.Insert Begin
2.Delete Begin
3.Delete End
4.Delete Key
5.Display
6.Exit
Enter choice: 3
Deleted 10 from end

1.Insert Begin
2.Delete Begin
3.Delete End
4.Delete Key
5.Display
6.Exit
Enter choice: 4
Enter key: 1
Element 1 not found

1.Insert Begin
2.Delete Begin
3.Delete End
4.Delete Key
5.Display
6.Exit
Enter choice: 4
Enter key: 0
Element 0 not found

1.Insert Begin
2.Delete Begin
3.Delete End
4.Delete Key
5.Display
6.Exit
Enter choice: 4
Enter key: 20
Deleted 20 from beginning
```

Program 6a:

a) **WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.**

Code:

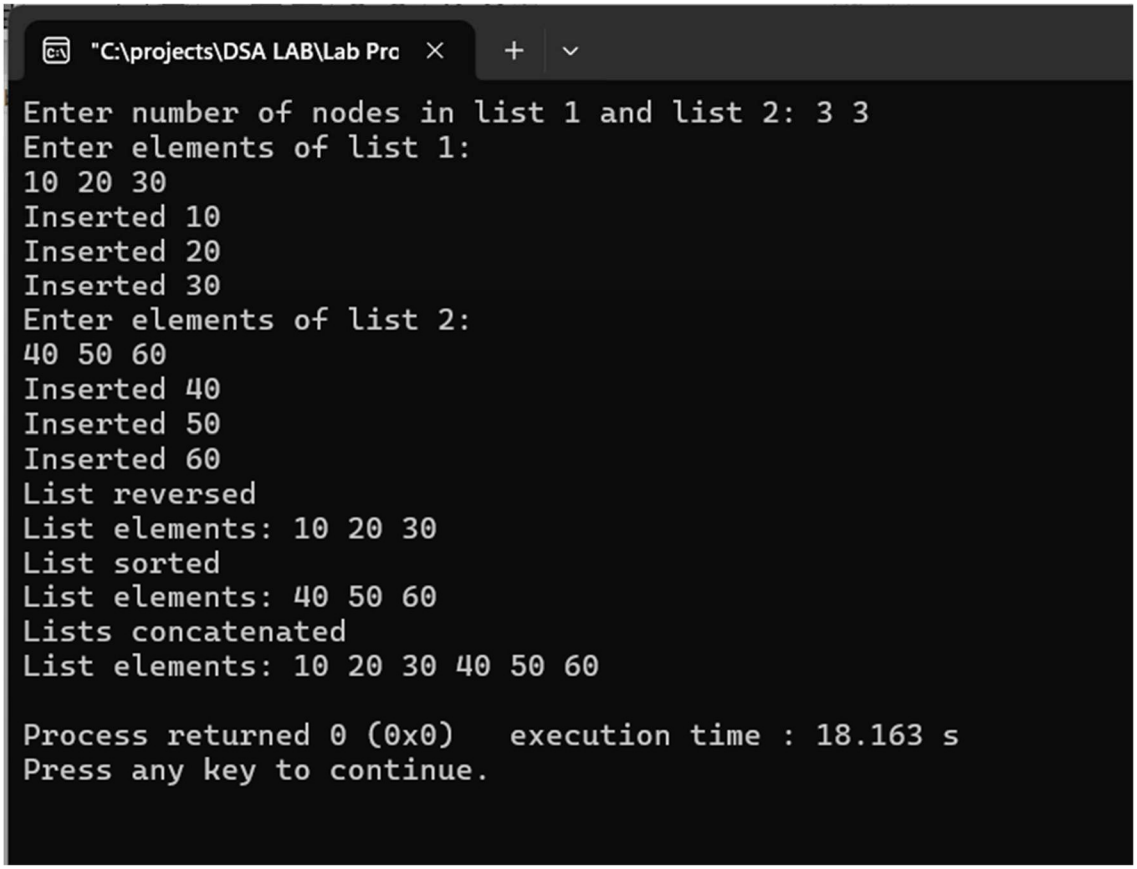
```
#include <stdio.h>

#include <stdlib.h>


struct node {

   int data;

   struct node *next;

};


struct node* create(int x) {

   struct node *n = malloc(sizeof(struct node));

   if (!n) {

      printf("Memory allocation failed\n");

      exit(1);

   }

   n->data = x;

   n->next = NULL;

   return n;

}


struct node* insert(struct node *head, int x) {

   struct node *n = create(x);

   n->next = head;

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

   return n;

}

void display(struct node *h) {
```

```
    if (!h) {
        printf("List is empty\n");
        return;
    }
    printf("List elements: ");
    while (h) {
        printf("%d ", h->data);
        h = h->next;
    }
    printf("\n");
}

struct node* reverse(struct node *h) {
    struct node *p = NULL, *c = h, *n;
    if (!h) {
        printf("List is empty. Cannot reverse\n");
        return h;
    }
    while (c) {
        n = c->next;
        c->next = p;
        p = c;
        c = n;
    }
    printf("List reversed\n");
    return p;
}

struct node* sort(struct node *h) {
    struct node *i, *j;
    int t;
```

```
    if (!h) {
        printf("List is empty. Cannot sort\n");
        return h;
    }
    for (i = h; i; i = i->next)
        for (j = i->next; j; j = j->next)
            if (i->data > j->data) {
                t = i->data;
                i->data = j->data;
                j->data = t;
            }
    printf("List sorted\n");
    return h;
}

struct node* concat(struct node *a, struct node *b) {
    struct node *t = a;
    if (!a && !b) {
        printf("Both lists are empty\n");
        return NULL;
    }
    if (!a) {
        printf("First list empty. Second list returned\n");
        return b;
    }
    while (t->next)
        t = t->next;
    t->next = b;
    printf("Lists concatenated\n");
    return a;
}
```

```
int main() {
   int m, n, x;
   struct node *h1 = NULL, *h2 = NULL;

   printf("Enter number of nodes in list 1 and list 2: ");
   scanf("%d %d", &m, &n);

   printf("Enter elements of list 1:\n");
   while (m--) {
      scanf("%d", &x);
      h1 = insert(h1, x);
   }

   printf("Enter elements of list 2:\n");
   while (n--) {
      scanf("%d", &x);
      h2 = insert(h2, x);
   }

   h1 = reverse(h1);
   display(h1);

   h2 = sort(h2);
   display(h2);

   h1 = concat(h1, h2);
   display(h1);

   return 0;
}
```

Output:


"C:\projects\DSA LAB\Lab Prc
Enter number of nodes in list 1 and list 2: 3 3
Enter elements of list 1:
10 20 30
Inserted 10
Inserted 20
Inserted 30
Enter elements of list 2:
40 50 60
Inserted 40
Inserted 50
Inserted 60
List reversed
List elements: 10 20 30
List sorted
List elements: 40 50 60
Lists concatenated
List elements: 10 20 30 40 50 60
Process returned 0 (0x0) execution time : 18.163 s
Press any key to continue.

Program 6b:

b) **WAP to Implement Single Link List to simulate Stack & Queue Operations.**

Code:

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

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

struct node *top = NULL, *front = NULL, *rear = NULL;

void push(int x) {
   struct node *n = malloc(sizeof(struct node));
   if (!n) {
      printf("Memory allocation failed\n");
      return;
   }
   n->data = x;
   n->next = top;
   top = n;
   printf("Pushed %d to stack\n", x);
}

void pop() {
   if (!top) {
      printf("Stack Underflow\n");
      return;
   }
```

```
    struct node *t = top;
    printf("Popped %d from stack\n", t->data);
    top = top->next;
    free(t);
}

void enqueue(int x) {
    struct node *n = malloc(sizeof(struct node));
    if (!n) {
        printf("Memory allocation failed\n");
        return;
    }
    n->data = x;
    n->next = NULL;
    if (!rear)
        front = rear = n;
    else {
        rear->next = n;
        rear = n;
    }
    printf("Enqueued %d to queue\n", x);
}

void dequeue() {
    if (!front) {
        printf("Queue Underflow\n");
        return;
    }
    struct node *t = front;
    printf("Dequeued %d from queue\n", t->data);
    front = front->next;
```

```
    if (!front) rear = NULL;
    free(t);
}

void display_stack() {
    struct node *t = top;
    if (!t) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack elements: ");
    while (t) {
        printf("%d ", t->data);
        t = t->next;
    }
    printf("\n");
}

void display_queue() {
    struct node *t = front;
    if (!t) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    while (t) {
        printf("%d ", t->data);
        t = t->next;
    }
    printf("\n");
}
```

```
int main() {
   int ch, x;
   while (1) {
      printf("\n1.Push\n2.Pop\n3.Display Stack\n4.Enqueue\n5.Dequeue\n6.Display
Queue\n7.Exit\n");
      printf("Enter choice: ");
      scanf("%d", &ch);

      if (ch == 1) {
         printf("Enter value: ");
         scanf("%d", &x);
         push(x);
      }
      else if (ch == 2)
         pop();
      else if (ch == 3)
         display_stack();
      else if (ch == 4) {
         printf("Enter value: ");
         scanf("%d", &x);
         enqueue(x);
      }
      else if (ch == 5)
         dequeue();
      else if (ch == 6)
         display_queue();
      else if (ch == 7)
         break;
      else
         printf("Invalid  choice\n");
```

```
    }

    return 0;

}
```

Output:

```
1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 1
Enter value: 10
Pushed 10 to stack

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 1
Enter value: 2
Pushed 2 to stack

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 3
Stack elements: 2 10

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 2
Popped 2 from stack

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 2
Popped 10 from stack

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 2
Stack Underflow
```

```
1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 2
Stack Underflow

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 4
Enter value: 1
Enqueued 1 to queue

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 6
Queue elements: 1

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 5
Dequeued 1 from queue

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice: 5
Queue Underflow

1.Push
2.Pop
3.Display Stack
4.Enqueue
5.Dequeue
6.Display Queue
7.Exit
Enter choice:
```

Program 7:

WAP to Implement doubly link list with primitive operations a) Create a doubly linked list. b) Insert a new node to the left of the node. c) Delete the node based on a specific value d) Display the contents of the list

Code:

```
#include <stdio.h>

#include <stdlib.h>


struct node {

   int data;

   struct node *prev, *next;

};


struct node *head = NULL;


struct node* create(int x) {

   struct node *n = malloc(sizeof(struct node));

   if (!n) {

      printf("Memory allocation failed\n");

      exit(1);

   }

   n->data = x;

   n->prev = n->next = NULL;

   return n;

}


void insert_end(int x) {

   struct node *n = create(x), *t = head;

   if (!head) {
```

```
        head = n;
        printf("Inserted %d as first node\n", x);
        return;
    }
    while (t->next)
        t = t->next;
    t->next = n;
    n->prev = t;
    printf("Inserted %d at end\n", x);
}

void insert_left(int x, int key) {
    struct node *t = head;
    while (t && t->data != key)
        t = t->next;

    if (!t) {
        printf("Key %d not found. Insertion failed\n", key);
        return;
    }

    struct node *n = create(x);
    n->next = t;
    n->prev = t->prev;

    if (t->prev)
        t->prev->next = n;
    else
        head = n;
```

```
    t->prev = n;
    printf("Inserted %d to the left of %d\n", x, key);
}

void delete_key(int key) {
    struct node *t = head;

    if (!head) {
        printf("List Underflow\n");
        return;
    }

    while (t && t->data != key)
        t = t->next;

    if (!t) {
        printf("Key %d not found\n", key);
        return;
    }

    if (t->prev)
        t->prev->next = t->next;
    else
        head = t->next;

    if (t->next)
        t->next->prev = t->prev;
```

```
    printf("Deleted node with value %d\n", key);
    free(t);
}

void display() {
    struct node *t = head;
    if (!t) {
        printf("List is empty\n");
        return;
    }
    printf("Doubly Linked List: ");
    while (t) {
        printf("%d ", t->data);
        t = t->next;
    }
    printf("\n");
}

int main() {
    int ch, x, k;
    while (1) {
        printf("\n1.Insert End\n2.Insert Left\n3.Delete Key\n4.Display\n5.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &ch);

        if (ch == 1) {
            printf("Enter value: ");
            scanf("%d", &x);
            insert_end(x);
```

```
        }
        else if (ch == 2) {
            printf("Enter new value and key: ");
            scanf("%d %d", &x, &k);
            insert_left(x, k);
        }
        else if (ch == 3) {
            printf("Enter key to delete: ");
            scanf("%d", &k);
            delete_key(k);
        }
        else if (ch == 4)
            display();
        else if (ch == 5)
            break;
        else
            printf("Invalid  choice\n");
    }
    return 0;
}
```

Output:

```
1.Insert End
2.Insert Left
3.Delete Key
4.Display
5.Exit
Enter choice: 1
Enter value: 10
Inserted 10 as first node

1.Insert End
2.Insert Left
3.Delete Key
4.Display
5.Exit
Enter choice: 1
Enter value: 20
Inserted 20 at end

1.Insert End
2.Insert Left
3.Delete Key
4.Display
5.Exit
Enter choice: 2
Enter new value and key: 30 20
Inserted 30 to the left of 20

1.Insert End
2.Insert Left
3.Delete Key
4.Display
5.Exit
Enter choice: 4
Doubly Linked List: 10 30 20

1.Insert End
2.Insert Left
3.Delete Key
4.Display
5.Exit
Enter choice: 3
Enter key to delete: 3
Key 3 not found

1.Insert End
2.Insert Left
3.Delete Key
4.Display
5.Exit
Enter choice: 3
Enter key to delete: 20
Deleted node with value 20
```

```
1.Insert End
2.Insert Left
3.Delete Key
4.Display
5.Exit
Enter choice: 4
Doubly Linked List: 10 30

1.Insert End
2.Insert Left
3.Delete Key
4.Display
5.Exit
Enter choice:
```

Program 8:

Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in-order, preorder and post order c) To display the elements in the tree.

Code:

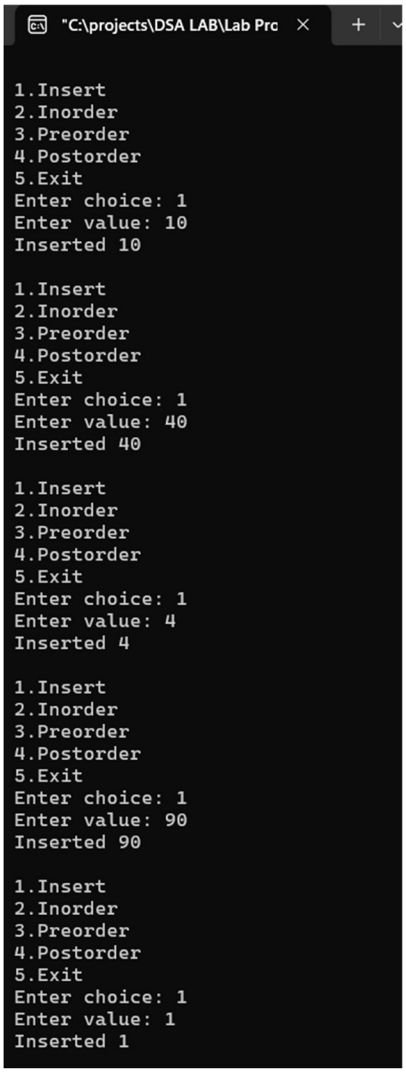
```
#include <stdio.h>
#include <stdlib.h>

struct node {
   int data;
   struct node *l, *r;
};

struct node* create(int x) {
   struct node *n = malloc(sizeof(struct node));
   if (!n) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   n->data = x;
   n->l = n->r = NULL;
   return n;
}

struct node* insert(struct node *r, int x) {
   if (!r) {
      printf("Inserted %d\n", x);
      return create(x);
   }
   if (x < r->data)          r->l = insert(r->l, x);
```

```
    else if (x > r->data)
        r->r = insert(r->r, x);
    else
        printf("Duplicate value %d not inserted\n", x);
    return r;
}

void inorder(struct node *r) {
    if (!r) return;
    inorder(r->l);
    printf("%d ", r->data);
    inorder(r->r);
}

void preorder(struct node *r) {
    if (!r) return;
    printf("%d ", r->data);
    preorder(r->l);
    preorder(r->r);
}

void postorder(struct node *r) {
    if (!r) return;
    postorder(r->l);
    postorder(r->r);
    printf("%d ", r->data);
}

int main() {
    int ch, x;
    struct node *root = NULL;
```

```
while (1) {
    printf("\n1.Insert\n2.Inorder\n3.Preorder\n4.Postorder\n5.Exit\n");
    printf("Enter choice: ");
    scanf("%d", &ch);

    if (ch == 1) {
        printf("Enter value: ");
        scanf("%d", &x);
        root = insert(root, x);
    }
    else if (ch == 2) {
        if (!root) printf("Tree is empty\n");
        else {
            printf("Inorder traversal: ");
            inorder(root);
            printf("\n");
        }
    }
    else if (ch == 3) {
        if (!root) printf("Tree is empty\n");
        else {
            printf("Preorder traversal: ");
            preorder(root);
            printf("\n");
        }
    }
    else if (ch == 4) {
        if (!root) printf("Tree is empty\n");
        else {
            printf("Postorder traversal: ");
```

```
            postorder(root);
            printf("\n");
          }
        }
        else if (ch == 5)
          break;
        else
          printf("Invalid choice\n");
    }
    return 0;
}
```

Output:


"C:\projects\DSA LAB\Lab Pro
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter choice: 1
Enter value: 10
Inserted 10
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter choice: 1
Enter value: 40
Inserted 40
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter choice: 1
Enter value: 4
Inserted 4
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter choice: 1
Enter value: 90
Inserted 90
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter choice: 1
Enter value: 1
Inserted 1

```
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter choice: 2
Inorder traversal: 1 4 10 40 90

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter choice: 3
Preorder traversal: 10 4 1 40 90

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter choice: 4
Postorder traversal: 1 4 90 40 10

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter choice:
```

Program 9a:

a) **Write a program to traverse a graph using BFS method.**

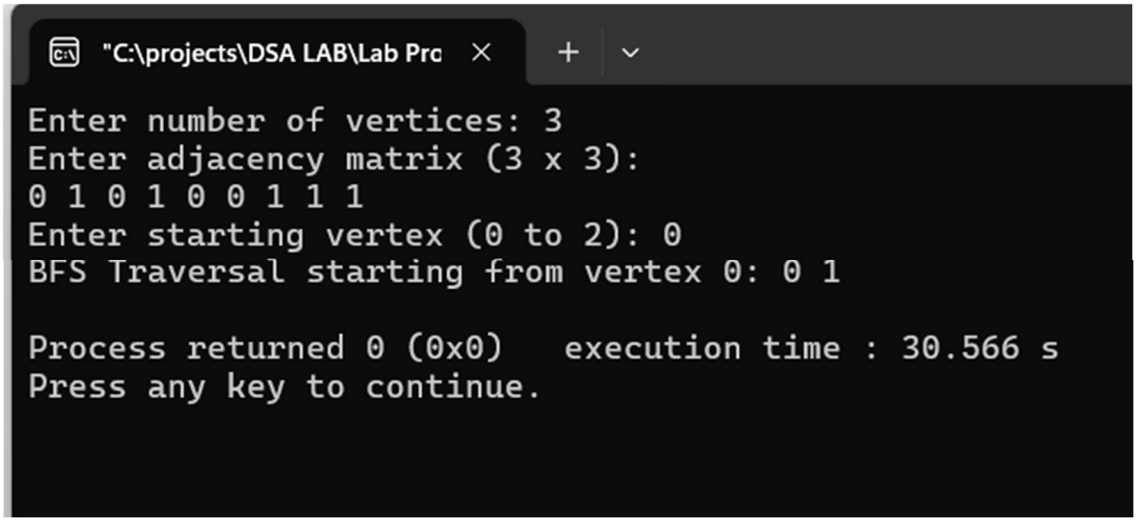
Output:

```
#include <stdio.h>

#define MAX 20

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

int visited[MAX];


void enqueue(int vertex) {
   if (rear == MAX - 1) {
      printf("Queue Overflow!\n");
      return;
   }
   if (front == -1) front = 0;
   queue[++rear] = vertex;
}


int dequeue() {
   if (front == -1 || front > rear) {
      return -1;
   }
   return queue[front++];
}


int isEmpty() {
   return (front == -1 || front > rear);
}


void BFS(int adj[MAX][MAX], int n, int start) {
   for (int i = 0; i < n; i++) visited[i] = 0;
```

```
    enqueue(start);
    visited[start] = 1;
    printf("BFS Traversal starting from vertex %d: ", start);

    while (!isEmpty()) {
        int v = dequeue();
        printf("%d ", v);
        for (int i = 0; i < n; i++) {
            if (adj[v][i] == 1 && !visited[i]) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}

int main() {
    int n, adj[MAX][MAX], start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix (%d x %d):\n", n, n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }
    printf("Enter starting vertex (0 to %d): ", n - 1);
    scanf("%d", &start);
    BFS(adj, n, start);
    return 0;
```

}

Output:


"C:\projects\DSA LAB\Lab Prc
Enter number of vertices: 3
Enter adjacency matrix (3 x 3):
0 1 0 1 0 0 1 1 1
Enter starting vertex (0 to 2): 0
BFS Traversal starting from vertex 0: 0 1
Process returned 0 (0x0) execution time : 30.566 s
Press any key to continue.

Program 9b:

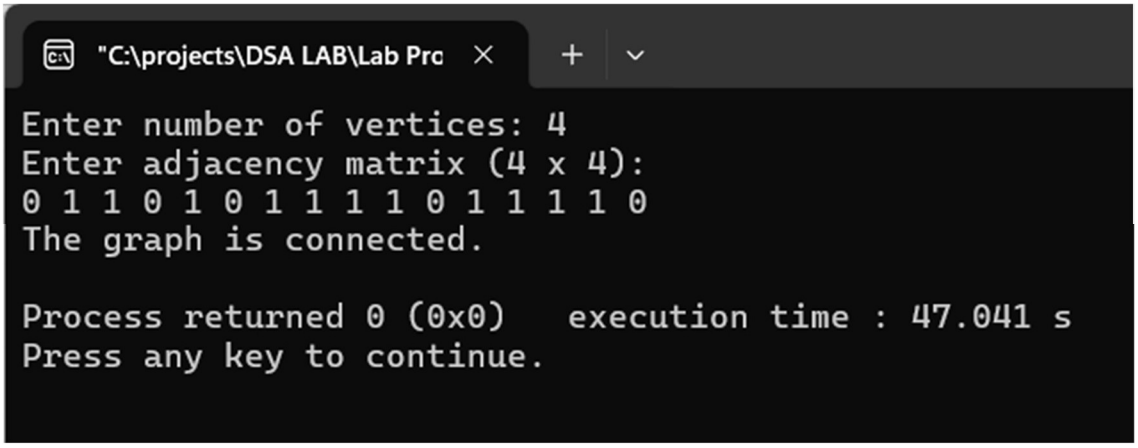
b) **Write a program to check whether given graph is connected or not using DFS method.**

Code:

```
#include <stdio.h>

#define MAX 20

int visited[MAX];


void DFS(int adj[MAX][MAX], int n, int start) {
   visited[start] = 1;
   for (int i = 0; i < n; i++) {
      if (adj[start][i] == 1 && !visited[i]) {
         DFS(adj, n, i);
      }
   }
}


int main() {
   int n, adj[MAX][MAX], start = 0;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter adjacency matrix (%d x %d):\n", n, n);
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         scanf("%d", &adj[i][j]);
      }
   }
   for (int i = 0; i < n; i++) visited[i] = 0;
   DFS(adj, n, start);
   int connected = 1;
   for (int i = 0; i < n; i++) {
```

```
        if (!visited[i]) {
            connected = 0;
            break;
        }
    }
    if (connected)
        printf("The graph is connected.\n");
    else
        printf("The graph is NOT connected.\n");
    return 0;
}
```

Output:



Program 10:

Given a File of N employee records with a set K of Keys(4 digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

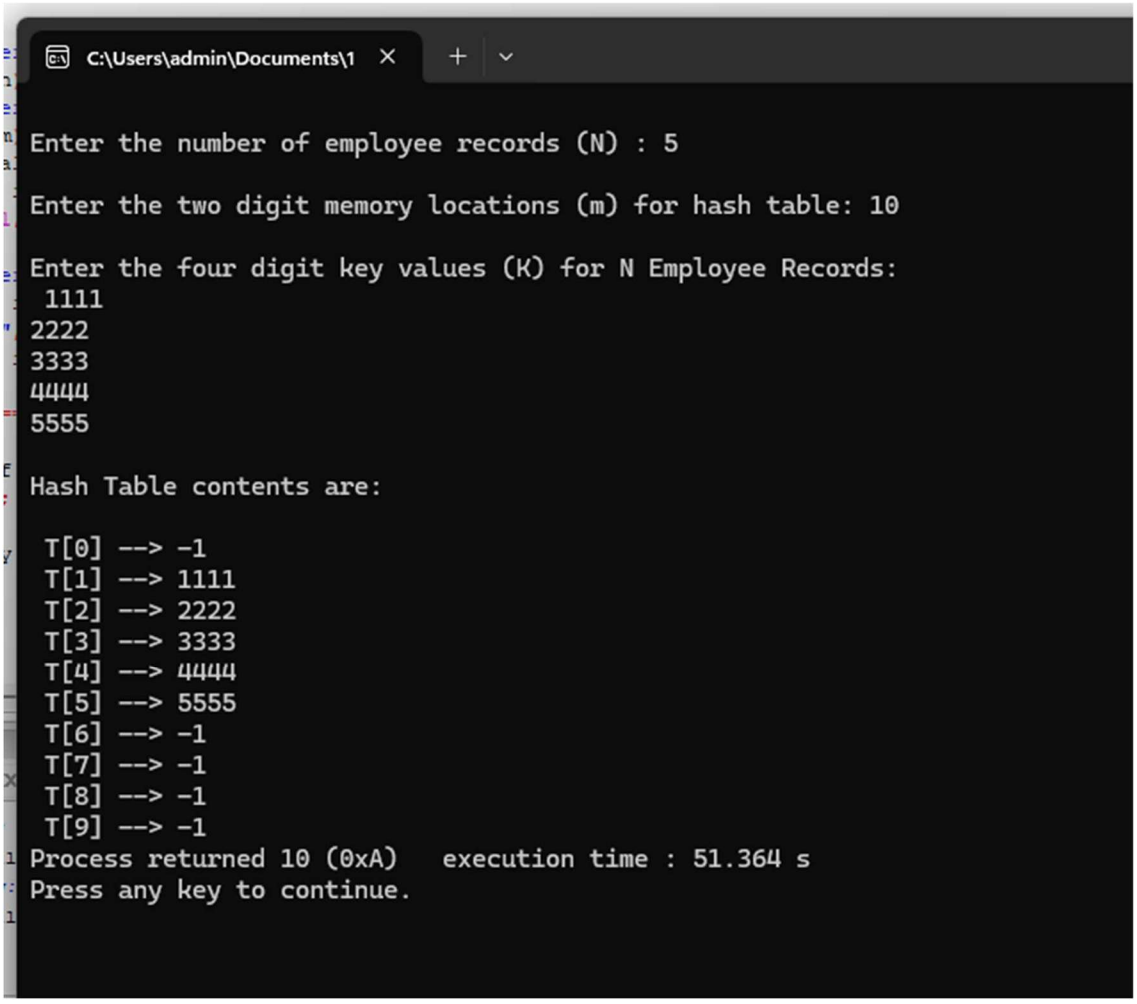
Code:

```
#include<stdio.h>

#include<stdlib.h>

int key[20], n, m;

int *ht, index;

int count = 0;

void insert(int key)

{

index = key % m;

while(ht[index] != -1)

{

index = (index+1)%m;

}

ht[index] = key;

count++;

}

void display()

{

int i;

if(count == 0)

{

printf("\nHash Table is empty");

return;
```

```
}

printf("\nHash Table contents are:\n ");
for(i=0; i<m; i++)
printf("\n T[%d] --> %d ", i, ht[i]);
}
void main()
{
int i;
printf("\nEnter the number of employee records (N) : ");
scanf("%d", &n);
printf("\nEnter the two digit memory locations (m) for
hash table: ");
scanf("%d", &m);
ht = (int *)malloc(m*sizeof(int));
for(i=0; i<m; i++)
ht[i] = -1;

printf("\nEnter the four digit key values (K) for N
Employee Records:\n “);
for(i=0; i<n; i++)
scanf("%d", &key[i]);
for(i=0; i<n; i++)
{
if(count == m)
{
printf("\nHash table is full. Cannot insert the
record %d key", i+1);
break;
```

```
}
insert(key[i]);
}
display();
}
```

Output:

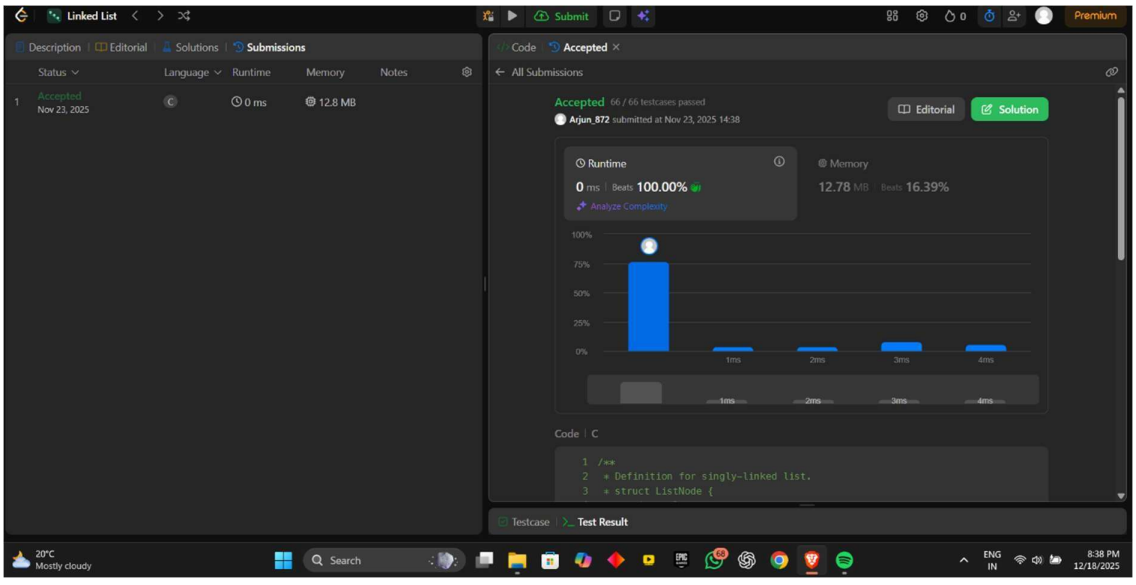


Leetcode Problem:

1) **203 – Remove Linked List Elements Given the head of a linked list and an integer val, remove all the nodes of the linked list that have Node.val == val, and return the new head. Example 1: Input: head = [1,2,6,3,4,5,6], val = 6 Output: [1,2,3,4,5] Example 2: Input: head = [], val = 1 Output: [] Example 3: Input: head = [7,7,7,7], val = 7 Output: []**

Code:

```
 * };
 */
struct ListNode* removeElements(struct ListNode* head, int val) {
    struct ListNode *t1=head,*t2;
    while (t1 != NULL) {
        if (t1->val == val) {
            if (t1 == head ) {
                head = head -> next;
                free (t1);
                t1 = head;
            }
            else {
                t2 -> next = t1 -> next;
                free (t1);
                t1 = t2 ->next;
            }
        }
        else {
            t2 = t1;
            t1 = t1 -> next;
        }
    }
    return head;
}

View less

Write your notes here
```



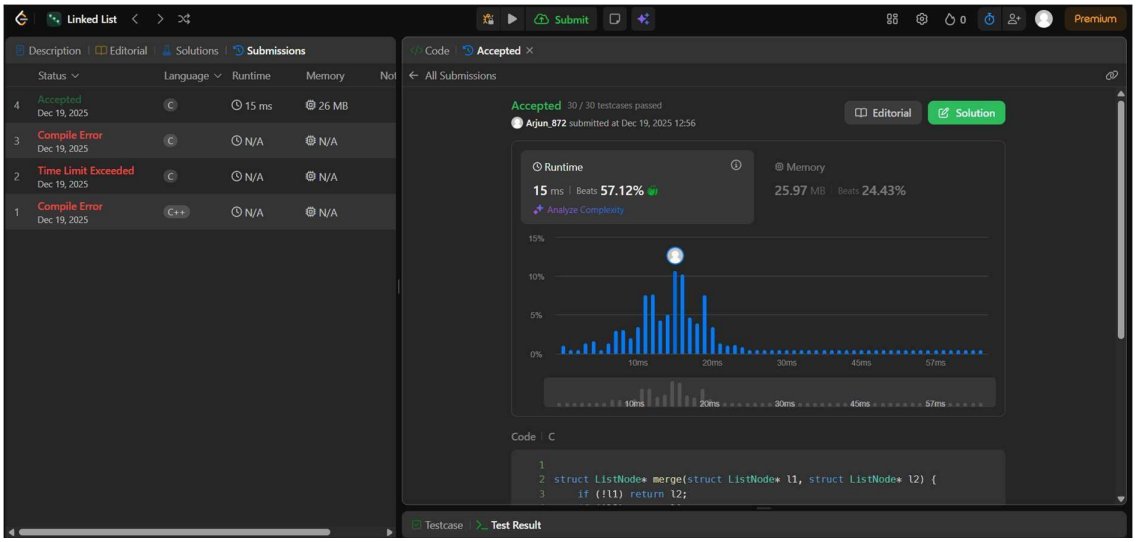
2) **148 – Sort List Given the head of a linked list, return the list after sorting it in ascending order. Example 1: Input: head = [4,2,1,3] Output: [1,2,3,4] Example 2: Input: head = [-1,5,3,4,0] Output: [-1,0,3,4,5] Example 3: Input: head = [] Output: []**

Code:

```
1
2  struct ListNode* merge(struct ListNode* l1, struct ListNode* l2) {
3      if (!l1) return l2;
4      if (!l2) return l1;
5
6      if (l1->val <= l2->val) {
7          l1->next = merge(l1->next, l2);
8          return l1;
9      } else {
10         l2->next = merge(l1, l2->next);
11         return l2;
12     }
13 }
14
15 struct ListNode* getMiddle(struct ListNode* head) {
16     struct ListNode *slow = head, *fast = head, *prev = NULL;
17
18     while (fast && fast->next) {
19         prev = slow;
20         slow = slow->next;
21         fast = fast->next->next;
22     }
23
24     if (prev)
25         prev->next = NULL;  // split the list
26
27     return slow;
28 }
```

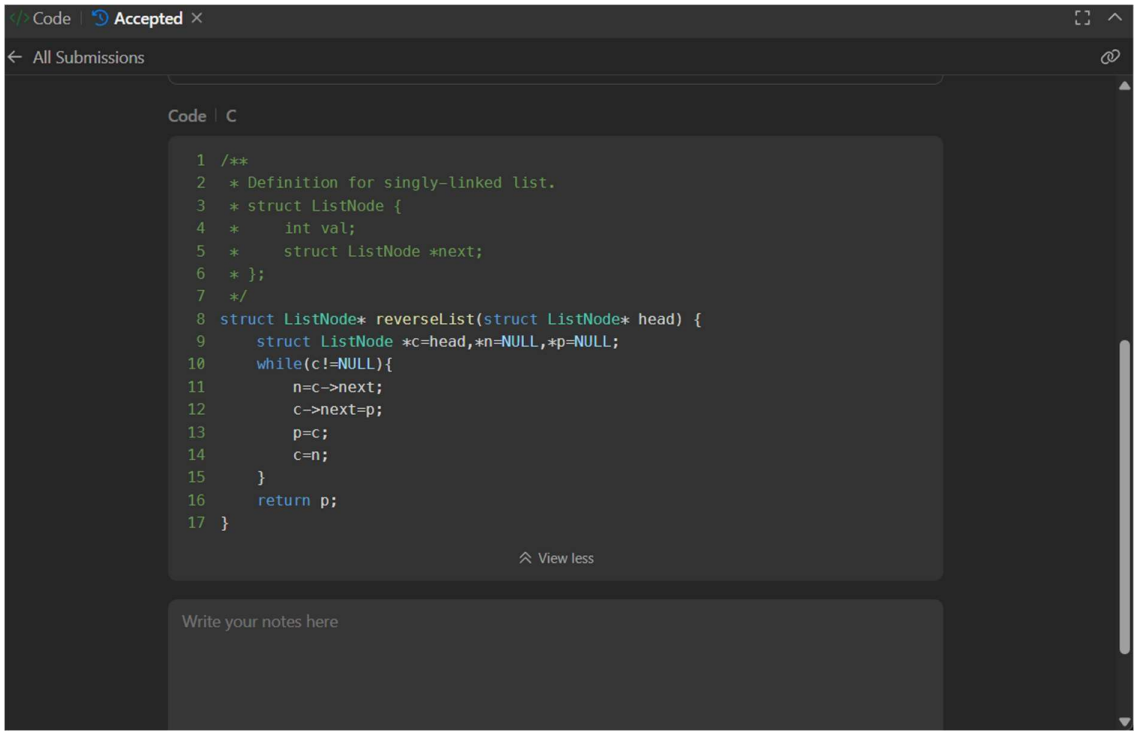
```
29
30 struct ListNode* sortList(struct ListNode* head) {
31     if (!head || !head->next)
32         return head;
33
34     struct ListNode* mid = getMiddle(head);
35
36     struct ListNode* left = sortList(head);
37     struct ListNode* right = sortList(mid);
38
39     return merge(left, right);
40 }
```

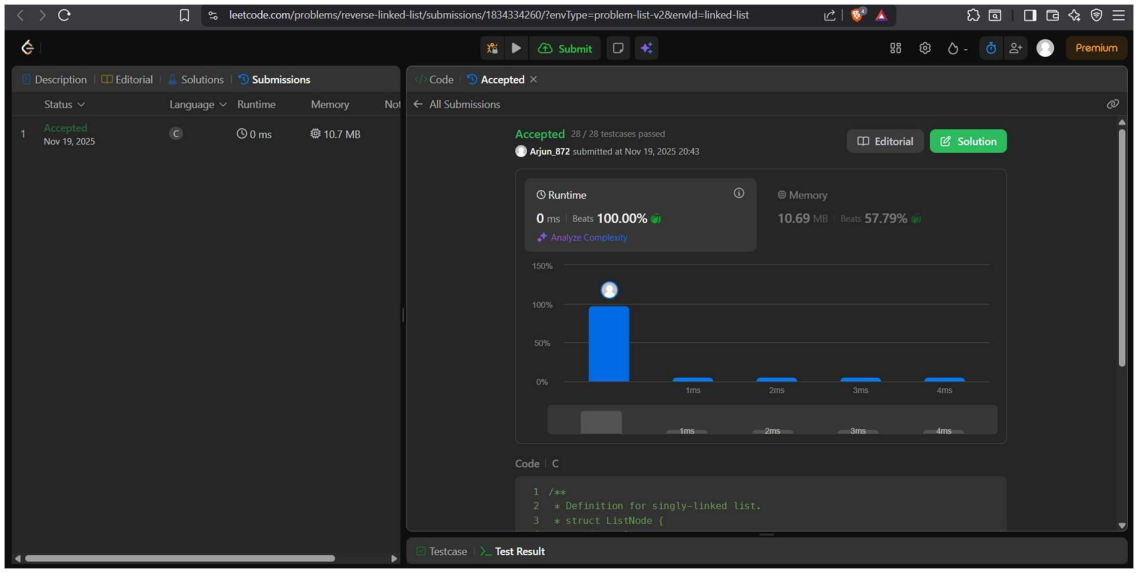
View less


Linked List
Submit
Premium
Description
Editorial
Solutions
Submissions
Status
Language
Runtime
Memory
4 Accepted Dec 19, 2025 C 15 ms 26 MB
3 Compile Error Dec 19, 2025 C N/A N/A
2 Time Limit Exceeded Dec 19, 2025 C N/A N/A
1 Compile Error Dec 19, 2025 C++ N/A N/A
Code
Accepted
All Submissions
Accepted 30 / 30 testcases passed
Arjun_872 submitted at Dec 19, 2025 12:56
Editorial
Solution
Runtime
15 ms | Beats 57.12%
Analyze Complexity
Memory
25.97 MB | Beats 24.43%
15%
10%
5%
0%
10ms
20ms
30ms
45ms
57ms
Code | C
struct ListNode* merge(struct ListNode* l1, struct ListNode* l2) {
if (!l1) return l2;
Testcase
Test Result

3) **Given the head of a singly linked list, reverse the list, and return the reversed list. Example 1: Input: head = [1,2,3,4,5] Output: [5,4,3,2,1] Example 2: Input: head = [1,2] Output: [2,1] Example 3: Input: head = [] Output: []**

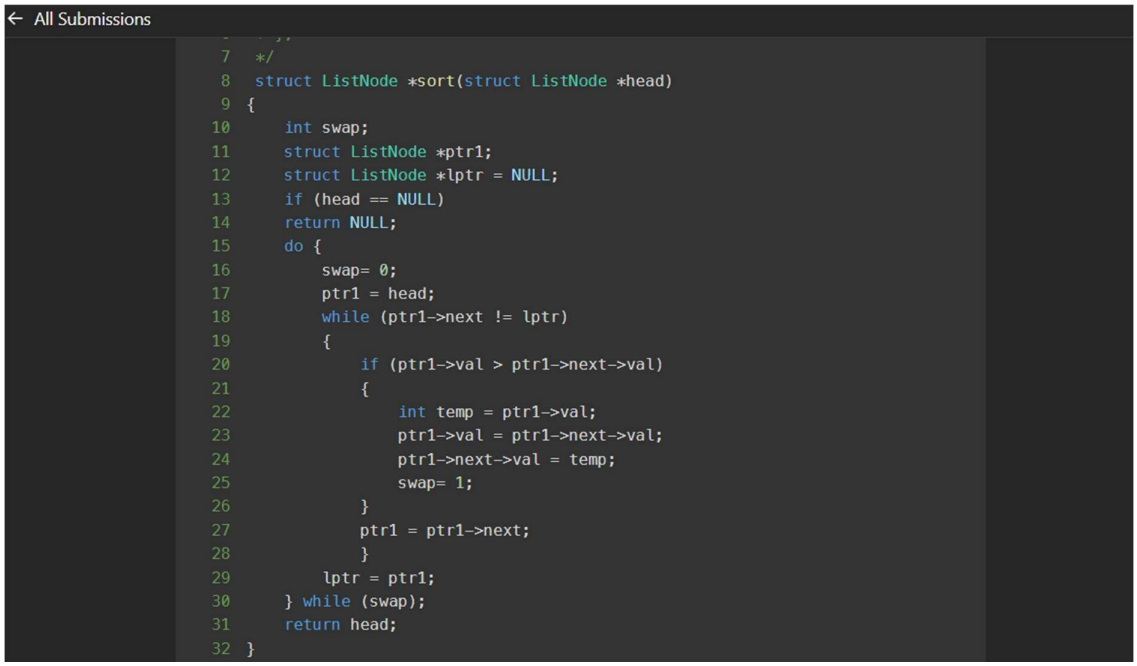
Code:

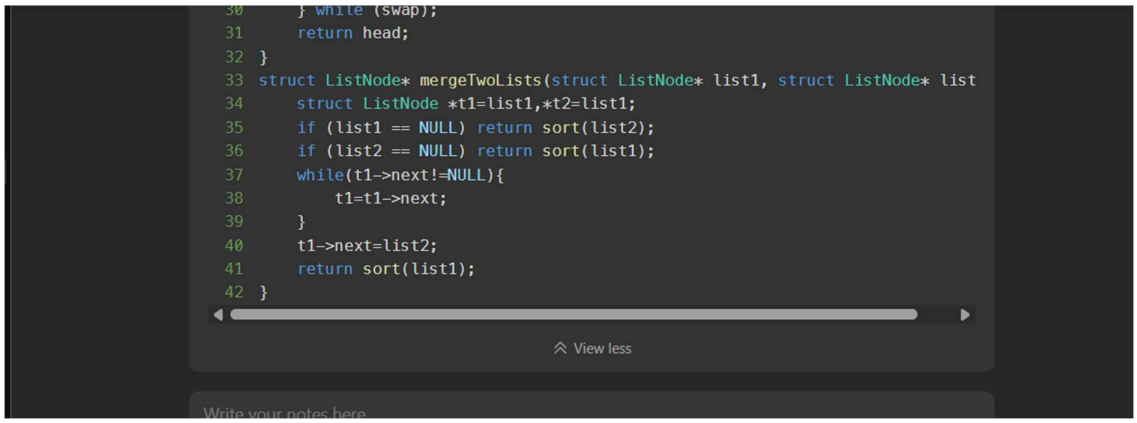


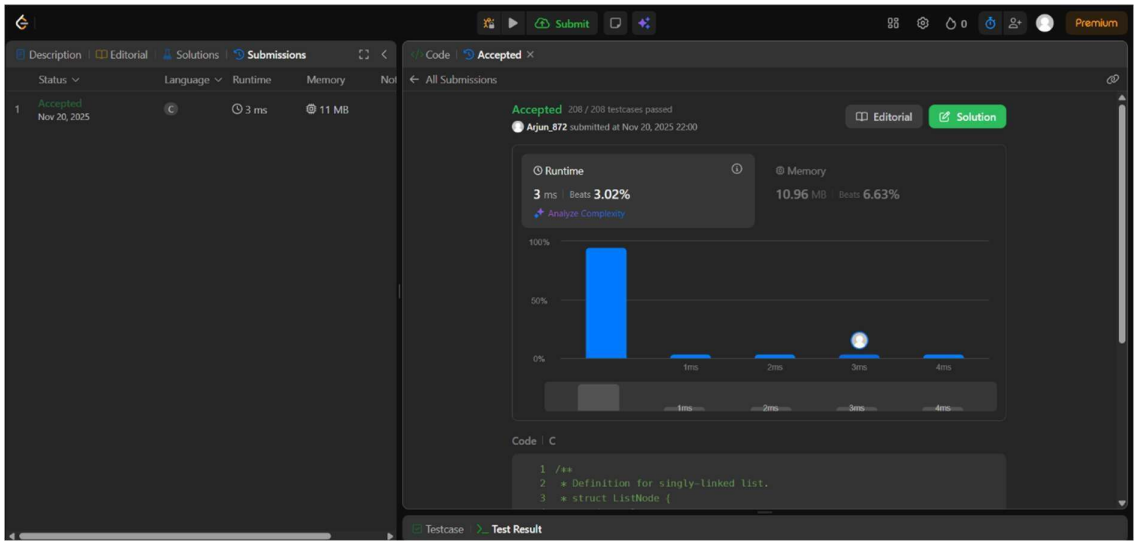


4) **21 – Merge Two Sorted Lists You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list. Example 1: Input: list1 = [1,2,4], list2 = [1,3,4] Output: [1,1,2,3,4,4] Example 2: Input: list1 = [], list2 = [] Output: [] Example 3: Input: list1 = [], list2 = [0] Output: [0]**

Code:

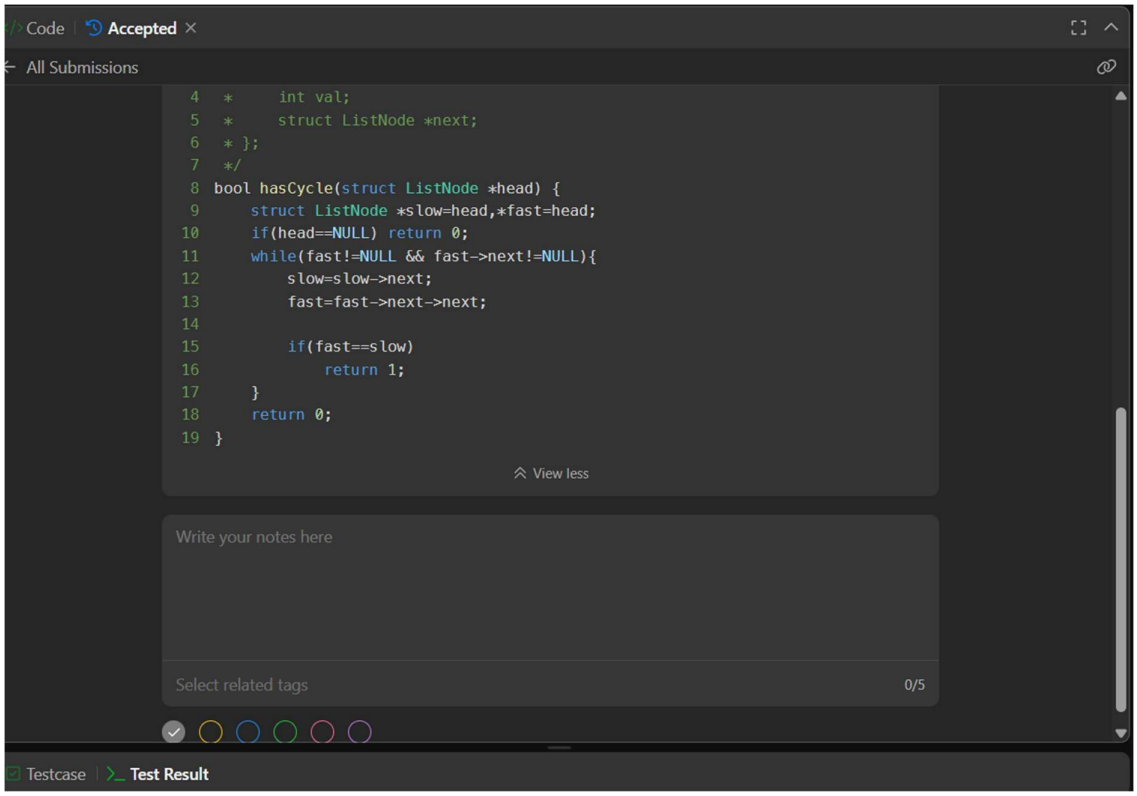


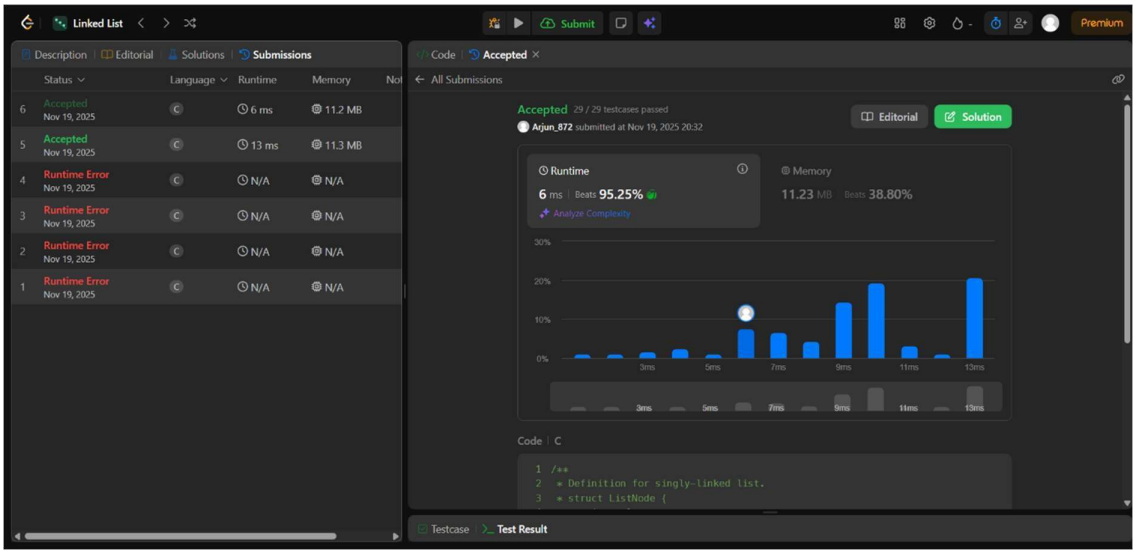



Submit
Premium
Description
Editorial
Solutions
Submissions
Status
Language
Runtime
Memory
1
Accepted
Nov 20, 2025
3 ms
11 MB
Code
Accepted
All Submissions
Accepted 208 / 208 testcases passed
Arjun_872 submitted at Nov 20, 2025 22:00
Editorial
Solution
Runtime
3 ms | Beats 3.02%
Analyze Complexity
Memory
10.96 MB | Beats 6.63%
100%
50%
0%
1ms
2ms
3ms
4ms
Code | C
1 /**
2 * Definition for singly-linked list.
3 * struct ListNode {
Testcase
Test Result

5) **141 – Linked List Cycle Given head, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that the tail's next pointer is connected to. Note that pos is not passed as a parameter. Return true if there is a cycle in the linked list. Otherwise, return false. Example 1: Input: head = [3,2,0,-4], pos = 1 Output: true Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0- indexed). Example 2: Input: head = [1,2], pos = 0 Output: true Explanation: There is a cycle in the linked list, where the tail connects to the 0th node. Example 3: Input: head = [1], pos = -1 65| Page Output: false Explanation: There is no cycle in the linked list.**

Code:




Linked List
Submit
Premium
Description
Editorial
Solutions
Submissions
Status
Language
Runtime
Memory
6 Accepted Nov 19, 2025 C 6 ms 11.2 MB
5 Accepted Nov 19, 2025 C 13 ms 11.3 MB
4 Runtime Error Nov 19, 2025 C N/A N/A
3 Runtime Error Nov 19, 2025 C N/A N/A
2 Runtime Error Nov 19, 2025 C N/A N/A
1 Runtime Error Nov 19, 2025 C N/A N/A
Code
Accepted
All Submissions
Accepted 29 / 29 testcases passed
Arjun_872 submitted at Nov 19, 2025 20:32
Editorial
Solution
Runtime
6 ms | Beats 95.25%
Analyze Complexity
Memory
11.23 MB Beats 38.80%
30%
20%
10%
0%
3ms
5ms
7ms
9ms
11ms
13ms
Code | C
1 /**
2 * Definition for singly-linked list.
3 * struct ListNode {
Testcase
Test Result

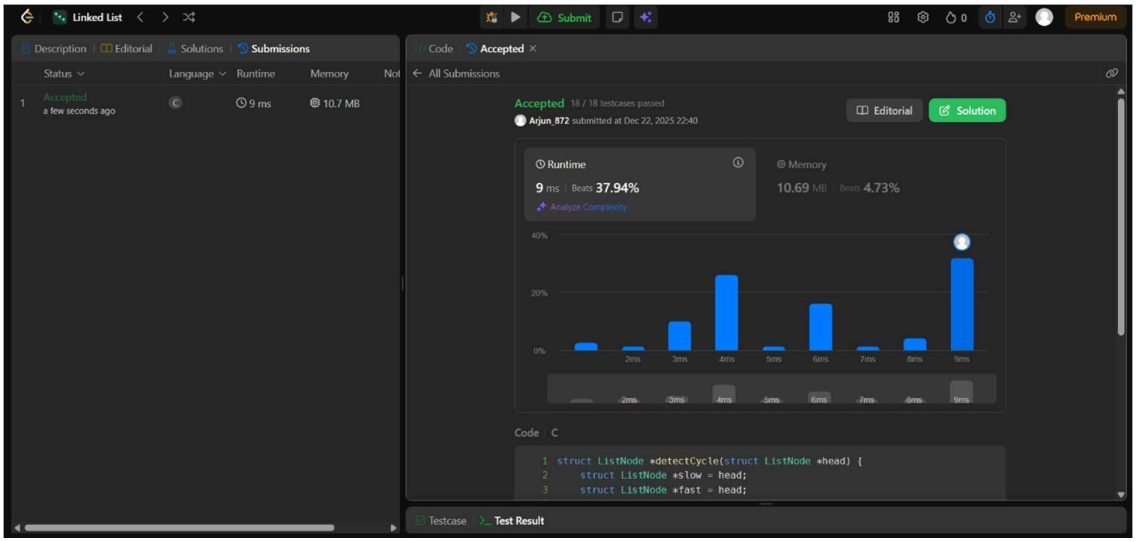
6) **142 – Linked List Cycle II Given the head of a linked list, return the node where the cycle begins. If there is no cycle, return null. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index (0-indexed) of the node that the tail's next pointer is connected to. If pos is -1, then there is no cycle. Note that pos is not passed as a parameter. Do not modify the linked list. Example 1: Input: head = [3,2,0,-4], pos = 1 Output: tail connects to node index 1 Explanation: There is a cycle in the linked list, where the tail connects to the second node. Example 2: Input: head = [1,2], pos = 0 Output: tail connects to node index 0 Explanation: There is a cycle in the linked list, where the tail connects to the first node. Example 3: Input: head = [1], pos = -1 Output: no cycle Explanation: There is no cycle in the linked list.**

Code:

Code | C

```c
struct ListNode *detectCycle(struct ListNode *head) {
    struct ListNode *slow = head;
    struct ListNode *fast = head;

    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;

        if (slow == fast) {
            struct ListNode *entry = head;
            while (entry != slow) {
                entry = entry->next;
                slow = slow->next;
            }
            return entry;
        }
    }
    return NULL;
}

```

View less

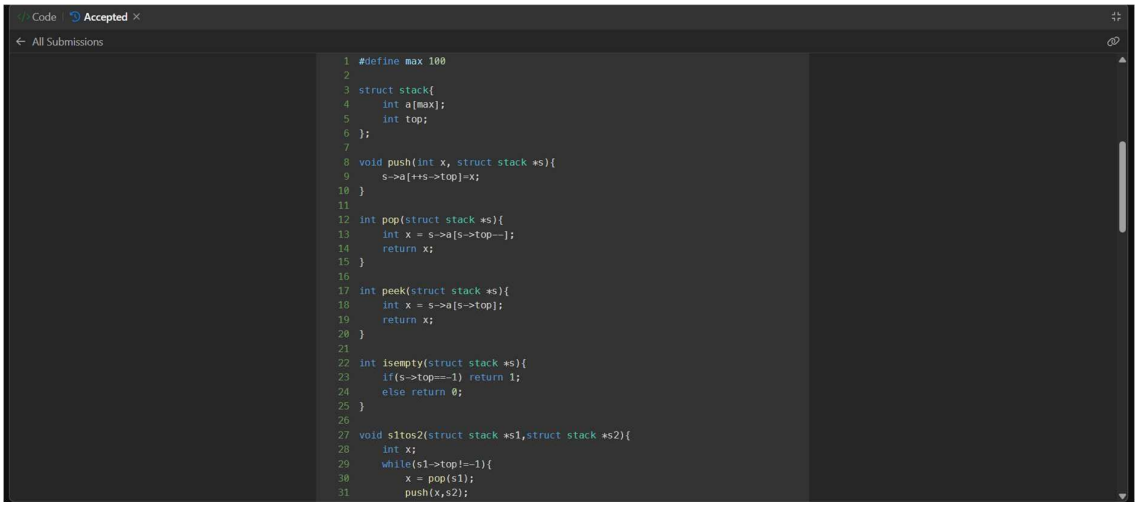

Linked List
Submit
Premium
Description
Editorial
Solutions
Submissions
Status
Language
Runtime
Memory
1
Accepted
a few seconds ago
9 ms
10.7 MB
Code
Accepted
All Submissions
Accepted 18 / 18 testcases passed
Arjun_872 submitted at Dec 22, 2025 22:40
Editorial
Solution
Runtime
9 ms | Beats 37.94%
Analyze Complexity
Memory
10.69 MB | Beats 4.73%
40%
20%
0%
2ms
3ms
4ms
5ms
6ms
7ms
8ms
9ms
Code | C
1 struct ListNode *detectCycle(struct ListNode *head) {
2 struct ListNode *slow = head;
3 struct ListNode *fast = head;
Testcase
Test Result

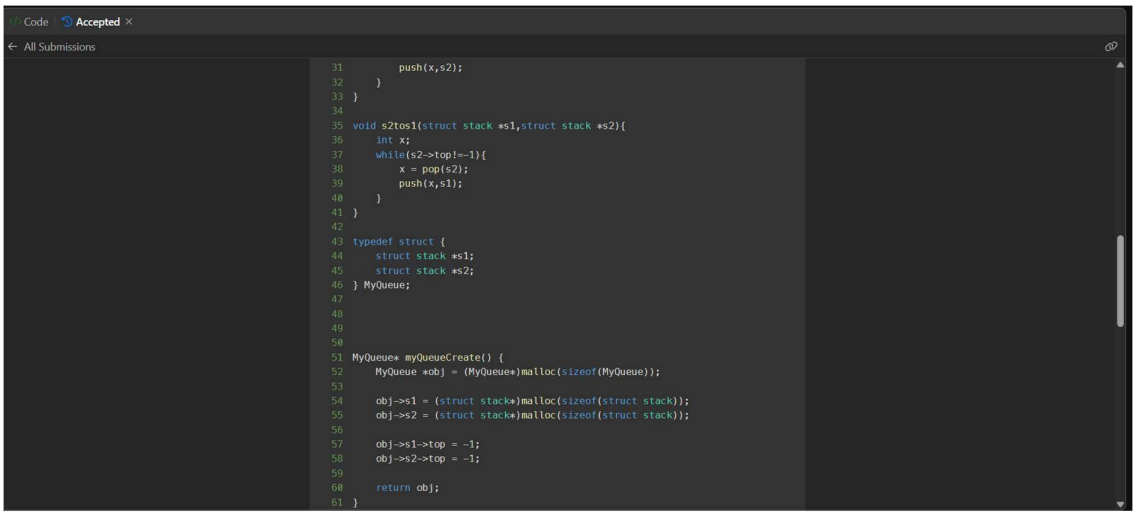
7) **232. Implement Queue using Stacks** Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

Implement the MyQueue class:

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.
- int peek() Returns the element at the front of the queue.
- boolean empty() Returns true if the queue is empty, false otherwise.

Code:





Code | Accepted

All Submissions

```
void myQueuePush(MyQueue* obj, int x) {
    push(x,obj->s1);
}

int myQueuePop(MyQueue* obj) {
    s1tos2(obj->s1,obj->s2);
    int x = pop(obj->s2);
    s2tos1(obj->s1,obj->s2);
    return x;
}

int myQueuePeek(MyQueue* obj) {
    s1tos2(obj->s1,obj->s2);
    int x = peek(obj->s2);
    s2tos1(obj->s1,obj->s2);
    return x;
}

bool myQueueEmpty(MyQueue* obj) {
    int x = isempty(obj->s1);
    if(x == 1) return true;
    else return false;
}

void myQueueFree(MyQueue* obj) {
    free(obj->s1);
    free(obj->s2);
    free(obj);
}
```