VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT

On

Data Structures(23CS3PCDST)

Submitted by

VARSHA P (1BM22CC320)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

Faculty Incharge

Surabhi S
Assistant Professor

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Dec 2023- March 2024

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 **VARSHA P(1BM22CS320)**, who is a 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 2023-24. 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.

Prof. Surabhi S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index

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.

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

a) Push

b) Pop

c) Display

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

```
#include <stdio.h>
#include <stdlib.h>
#define max 10
int stack[max];
int top = -1;
int val;
void push(int);
void pop();
void display();
int main()
{
int choice = 0;
do
{
printf("Enter your choice\n");
printf("1.Push\n2.Pop\n3.Display\n4.Exit\n");
scanf("%d", &choice);
switch (choice)
{
case 1:
if (top == max - 1)
{
printf("Overflow\n");
}
else
{
printf("Enter value to be pushed\n");
scanf("%d", &val);
push(val);
printf("-----------------Push operation completed\n");
}
break;
case 2:
if (top == -1)
{
printf("Underflow\n");
}
```

```
else
{
pop();
printf("-------------------Pop operation completed\n");
}
break;
case 3:
if (top == -1)
{
printf("Stack is empty\n");
}
else
{
display();
}
break;
case 4:
printf("Exit\n");
exit(0);
default:
printf("Incorrect input\n");
}
} while (choice != 4);
return 0;
}
void push(int val)
{
if (top != max - 1)
{
top++;
stack[top] = val;
}
else
{
printf("Overflow\n");
}
}
void pop()
{
if (top != -1)
{
val = stack[top];
top--;
```

```
}
else
{
printf("Underflow\n");
}
}
void display()
{
    printf("---------------");
if (top != -1)
{
for (int i = 0; i <= top; i++)
printf("%d\t", stack[i]);
}
printf("\n");
}
```

OUTPUT:

```
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
1
Enter value to be pushed
3
-----------------Push operation completed
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
1
Enter value to be pushed
4
-----------------Push operation completed
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
1
Enter value to be pushed
5
-----------------Push operation completed
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
2
------------------Pop operation completed
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
3
--------------- 34
```

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

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
#define max 100
char st[max];
int top=-1;
void push(char st[], char);
char pop(char st[]);
void infixToPostfix(char source[], char target[]);
int getPriority(char);
int main()
{
char infix[100], postfix[100];
printf("Enter infix expression\n");
gets(infix);
strcpy(postfix," ");
infixToPostfix(infix, postfix);
printf("Postfix expression is-\n");
puts(postfix);
return 0;
}
void infixToPostfix(char source[], char target[])
{
int i=0, j=0;
char temp;
strcpy(target," ");
while (source[i]!='\0')
{
if(source[i]=='(')
{
push(st,source[i]);
i++;
}
else if(source[i]==')')
{
while((top!=-1)&&(st[top]!='('))
{
```

```
target[j]=pop(st);
j++;
}
if(top==-1)
{
printf("Incorrect expression\n");
exit(1);
}
temp=pop(st);
i++;
}
else if(isdigit(source[i])||isalpha(source[i]))
{
target[j]=source[i];
j++;
i++;
}
else
if(source[i]=='+'||source[i]=='-'||source[i]=='*'||source[i]=='/'||source[i]=='%')
{
while( (top!=-1) && (st[top]!='(') &&
(getPriority(st[top])>getPriority(source[i])) )
{
target[j]=pop(st);
j++;
}
push(st,source[i]);
i++;
}
else
{
printf("Incorrect element in expression\n");
exit(1);
}
}
while((top!=-1)&&(st[top]!='('))
{
target[j]=pop(st);
j++;
}
target[j]='\0';
}
int getPriority(char op)
```

```
{
if(op=='/'|| op=='*'||op=='%')
return 1;
else if(op=='+'||op=='-')
return 0;
}
void push(char st[],char val)
{
if(top==max-1)
{
printf("Stack overflow\n");
}
else
{
top++;
st[top]=val;
}
}
char pop(char st[])
{
char val = ' ';
if(top == -1)
{
printf("Stack underflow\n");
}
else
{
val = st[top];
top--;
}
return val;
}
```

OUTPUT:

```
Enter infix expression
A+B+C*D
Postfix expression is-
ABCD*++
```

3.Write a program to simulate the working of the queue of integers using an array. Provide the following operations: Insert, delete, display. The program should print appropriate messages for overflow and underflow conditions.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define max 100
int q[max], front=1, rear=-1;
void insert();
void delete();
void display();
void main()
{
while(1)
{
printf("Enter your choice\n");
printf("1.Enqueue\n 2.Dequeue\n 3.Display\n 4.Exit\n");
int choice=0;
scanf("%d", & choice);
switch(choice)
{
case 1:
insert();
break;
case 2:
delete();
break;
case 3:
display();
break;
case 4:
exit(0);
break;
default:
printf("Incorrect Input\n");
}
}
}
void insert()
{
int val=0;
printf("Enter value to be inserted\n");
scanf("%d", & val);
```

```
if(rear==max-1)
printf("Queue Overflow\n");
if (front==-1||rear==-1)
{
front=0;
rear=0;
}
else{
rear=rear+1;
}
q[rear]=val;
}
void delete()
{
if (front>rear||front==-1)
{
printf("Queue Underflow\n");
}
else{
printf("------------------%d has been deleted\n",q[front]);
front=front+1;
}
}
void display()
{
printf("The queue is-\n");
printf("--------------------");
for(int i=front; i<=rear; i++)
printf("%d\t",q[i]);
printf("\n");
}
```

OUTPUT:

```
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
1
Enter value to be inserted
2
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
1
Enter value to be inserted
4
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
1
Enter value to be inserted
6
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
2
------------------2 has been deleted
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
3
The queue is-
--------------------46
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
```

4.Write a program to simulate the working of a circular queue using an array. Provide the following operations: insert, delete& display. The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define max 20
int q[max], rear=-1, front=-1;
int isFull();
int isEmpty();
void insert (int element);
int delete();
void display();
void main()
{
int choice, element;
while(1)
{
printf("Enter choice\n");
printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n\n");
scanf("%d", &choice);
switch(choice)
{
case 1:
printf("Enter element to be inserted\n");
scanf("%d", & element);
insert(element);
break;
case 2:
element=delete();
break;
case 3:
display();
break;
case 4:
exit(0); break;
default:
printf("Incorrect Input\n");
break;
}
}
}
```

```
int isFull()
{
if((front==rear+1)||(front==0 && rear==max-1))
return 1;
return 0;
}
int isEmpty()
{
if(front==-1)
return 1;
return 0;
}
void insert(int element)
{
if (isFull())
{ printf("Overflow\n");
}
else
{
if(front==-1)
front=0;
rear=(rear+1)%max;
q[rear]=element;
}
}
int delete()
{
int value;
if (isEmpty())
{
printf("Underflow\n");
return -1;
}
else{
value=q[front];
if(front==rear)
{
front==-1;
rear==-1;
}
else
{
front=(front+1)%max;
```

```
}
return(value);
}}

void display()
{
int i;
if(isEmpty())
printf("Underflow\n");
else{
for(i=front;i!=rear;i=(i+1)%max)
printf("%d\t",q[i]);
printf("%d\t",q[i]);
printf("\n");
}
}
```

OUTPUT:

```
1.Insert
2.Delete
3.Display
4.Exit

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

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

1
Enter element to be inserted
6
Enter choice
1.Insert
2.Delete
3.Display
4.Exit

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

3
6
Enter choice
1.Insert
2.Delete
3.Display
4.Exit
```

5.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.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node* next;
};
struct Node* createNode(int newData) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = newData;
newNode->next = NULL;
return newNode;
}
struct Node* insertAtFirst(struct Node* head, int newData) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data=newData;
newNode->next = head;
return newNode;
}
struct Node* insertAtPosition(struct Node* head, int newData, int
position)
{
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data=newData;
if (position == 1) {
newNode->next = head;
return newNode;
}
struct Node* temp = head;
for (int i = 1; i < position - 1 && temp != NULL; i++) {
temp = temp->next;
}
if (temp == NULL) {
printf("Invalid position\n");
return head;
}
newNode->next = temp->next;
temp->next = newNode;
return head;
```

```
}
struct Node* insertAtEnd(struct Node* head, int newData)
{
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data=newData;
newNode->next=NULL;
if (head == NULL) {
return newNode;
}
struct Node* temp = head;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = newNode;
return head;
}
void displayList(struct Node* head) {
struct Node* temp = head;
while (temp != NULL) {
printf("%d -> ", temp->data);
temp = temp->next;
}
printf("NULL\n");
}
void main() {
struct Node* head = NULL;
head = insertAtEnd(head, 1);
head = insertAtEnd(head, 2);
head = insertAtEnd(head, 3);
printf("Linked List: ");
displayList(head);
head = insertAtFirst(head, 0);
printf("After insertion at first position: ");
displayList(head);
head = insertAtPosition(head, 4, 4);
printf("After insertion at position 4: ");
displayList(head);
head = insertAtEnd(head, 5);
printf("After insertion at end: ");
displayList(head);
}
```

OUTPUT:

```
Linked List: 1 -> 2 -> 3 -> NULL
After insertion at first position: 0 -> 1 -> 2 -> 3 -> NULL
After insertion at position 4: 0 -> 1 -> 2 -> 4 -> 3 -> NULL
After insertion at end: 0 -> 1 -> 2 -> 4 -> 3 -> 5 -> NULL

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

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

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node* next;
};
struct Node* insertAtEnd(struct Node* head, int newData)
{
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = newData;
newNode->next = NULL;
if (head == NULL)
{
return newNode;
}
struct Node* temp = head;
while (temp->next != NULL)
{
temp = temp->next;
}
temp->next = newNode;
return head;
}
struct Node* deleteFirst(struct Node* head)
{
if (head == NULL)
{
printf("List is Empty! Deletion not Possible");
return NULL;
}
struct Node* newHead = head->next;
free(head);
return newHead;
}
struct Node* deleteElement(struct Node* head, int target)
{
if (head == NULL)
```

```
{
printf("List is Empty, hence cannot Delete \n");
return NULL;
}
if (head->data == target)
{
struct Node* newHead = head->next;
free(head);
return newHead;
}
struct Node* temp = head;
while (temp->next != NULL && temp->next->data != target)
{
temp = temp->next;
}
if (temp->next == NULL)
{
printf("Element %d not found in the list \n", target);
return head;
}
struct Node* nodeToDelete = temp->next;
temp->next = temp->next->next;
free(nodeToDelete);
return head;
}
struct Node* deleteLast(struct Node* head)
{
if (head == NULL)
{
printf("List is Empty, hence cannot Delete \n");
return NULL;
}
if (head->next == NULL)
{
free(head);
return NULL;
}
struct Node* temp = head;
while (temp->next->next != NULL)
{
temp = temp->next;
}
free(temp->next);
```

```
temp->next = NULL;
return head;
}
void displayList(struct Node* head)
{
struct Node* temp = head;
while (temp != NULL)
{
printf(" %d ->", temp->data);
temp = temp->next;
}
printf("NULL \n");
}
int main()
{
struct Node* head = NULL;
head = insertAtEnd(head, 1);
head = insertAtEnd(head, 2);
head = insertAtEnd(head, 3);
printf("Linked List:");
displayList(head);
head = deleteFirst(head);
printf("After deleting the first element:");
displayList(head);
head = deleteElement(head, 2);
printf("After deleting the second element:");
displayList(head);
head = deleteLast(head);
printf("After deleting the last Element:");
displayList(head);
return 0;
}
```

OUTPUT

```
Linked List: 1 -> 2 -> 3 ->NULL
After deleting the first element: 2 -> 3 ->NULL
After deleting the second element: 3 ->NULL
After deleting the last Element:NULL

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

7.WAP using Single linked list
->Sorting
->Reversing
->Concatenation

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

// Node structure for the linked list
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node with the given data
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        fprintf(stderr, "Memory allocation error\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new node at the end of the linked list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);

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

// Function to display the linked list
void displayList(struct Node* head) {
```

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

// Function to sort the linked list using Bubble Sort
void sortList(struct Node* head) {
    struct Node *i, *j;
    int temp;

    for (i = head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                // Swap data of the two nodes
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}

// Function to reverse the linked list
void reverseList(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    *head = prev;
}

// Function to concatenate two linked lists
void concatenateLists(struct Node** list1, struct Node* list2) {
    if (*list1 == NULL) {
```

```
            *list1 = list2;
        } else {
            struct Node* current = *list1;
            while (current->next != NULL) {
                current = current->next;
            }
            current->next = list2;
        }
}

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    // Insert elements into the first linked list
    insertAtEnd(&list1, 5);
    insertAtEnd(&list1, 2);
    insertAtEnd(&list1, 8);

    // Display the original linked list
    printf("Original Linked List 1:\n");
    displayList(list1);

    // Sort the linked list 1
    sortList(list1);
    printf("Sorted Linked List 1:\n");
    displayList(list1);

    // Insert elements into the second linked list
    insertAtEnd(&list2, 3);
    insertAtEnd(&list2, 1);
    insertAtEnd(&list2, 7);

    // Display the original linked list 2
    printf("\nOriginal Linked List 2:\n");
    displayList(list2);

    // Concatenate the two linked lists
    concatenateLists(&list1, list2);

    // Display the concatenated linked list
    printf("\nConcatenated Linked List:\n");
    displayList(list1);
```

```c
    // Reverse the concatenated linked list
    reverseList(&list1);
    printf("\nReversed Linked List:\n");
    displayList(list1);

    // Free the memory used by the linked lists
    struct Node* temp;
    while (list1 != NULL) {
        temp = list1;
        list1 = list1->next;
        free(temp);
    }

    return 0;
}
```

OUTPUT:

```
Original Linked List 1:
5 -> 2 -> 8 -> NULL
Sorted Linked List 1:
2 -> 5 -> 8 -> NULL

Original Linked List 2:
3 -> 1 -> 7 -> NULL

Concatenated Linked List:
2 -> 5 -> 8 -> 3 -> 1 -> 7 -> NULL

Reversed Linked List:
7 -> 1 -> 3 -> 8 -> 5 -> 2 -> NULL

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

8a.Stack implementation using single linked list

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

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

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to push an element onto the stack
struct Node* push(struct Node* top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = top;
    return newNode;
}

// Function to pop an element from the stack
struct Node* pop(struct Node* top) {
    if (top == NULL) {
        printf("Stack underflow. Cannot pop.\n");
        return NULL;
    }

    struct Node* temp = top;
    top = top->next;
    free(temp);
    return top;
}

// Function to display the elements of the stack
```

```c
void displayStack(struct Node* top) {
    printf("Stack: ");
    while (top != NULL) {
        printf("%d ", top->data);
        top = top->next;
    }
    printf("\n");
}

// Function to free the memory allocated for the stack
void freeStack(struct Node* top) {
    while (top != NULL) {
        struct Node* temp = top;
        top = top->next;
        free(temp);
    }
}

int main() {
    struct Node* top = NULL;
    int choice, data;

    do {
        printf("\nMenu:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to push: ");
                scanf("%d", &data);
                top = push(top, data);
                break;

            case 2:
                top = pop(top);
                break;

```

```c
            case 3:
                displayStack(top);
                break;

            case 4:
                printf("Exiting the program.\n");
                break;

            default:
                printf("Invalid choice! Please enter a valid option.\n");
        }

    } while (choice != 4);

    // Free the memory allocated for the stack before exiting
    freeStack(top);

    return 0;
}
```

OUTPUT:

```
Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter data to push: 2

Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2

Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack: 2

Menu:
1. Push
2. Pop
3. Display
4. Exit
```

9.Queue implementation using single linked list

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

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

// Queue structure
struct Queue {
    struct Node* front;
    struct Node* rear;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to initialize an empty queue
struct Queue* initializeQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    if (queue == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    queue->front = queue->rear = NULL;
    return queue;
}

// Function to enqueue an element into the queue
void enqueue(struct Queue* queue, int data) {
    struct Node* newNode = createNode(data);
```

```c
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
        return;
    }

    queue->rear->next = newNode;
    queue->rear = newNode;
}

// Function to dequeue an element from the queue
void dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Queue underflow. Cannot dequeue.\n");
        return;
    }

    struct Node* temp = queue->front;
    queue->front = queue->front->next;

    // If front becomes NULL, update rear to NULL as well
    if (queue->front == NULL) {
        queue->rear = NULL;
    }

    free(temp);
}

// Function to display the elements of the queue
void displayQueue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Queue is empty.\n");
        return;
    }

    struct Node* current = queue->front;
    printf("Queue: ");
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

```
// Function to free the memory allocated for the queue
void freeQueue(struct Queue* queue) {
    while (queue->front != NULL) {
        struct Node* temp = queue->front;
        queue->front = queue->front->next;
        free(temp);
    }
    free(queue);
}

int main() {
    struct Queue* queue = initializeQueue();
    int choice, data;

    do {
        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:
                printf("Enter data to enqueue: ");
                scanf("%d", &data);
                enqueue(queue, data);
                break;

            case 2:
                dequeue(queue);
                break;

            case 3:
                displayQueue(queue);
                break;

            case 4:
                printf("Exiting the program.\n");
                break;
```

```c
            default:
                printf("Invalid choice! Please enter a valid option.\n");
        }

    } while (choice != 4);

    // Free the memory allocated for the queue before exiting
    freeQueue(queue);

    return 0;
}
```

OUTPUT:

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter data to enqueue: 4

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter data to enqueue: 5

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

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue: 3 4 5
```

9. 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

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

// Node structure for doubly linked list
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new node to the left of the given node
void insertLeft(struct Node** head, struct Node* target, int value) {
    struct Node* newNode = createNode(value);

    if (target->prev != NULL)
        target->prev->next = newNode;
    else
        *head = newNode;

    newNode->prev = target->prev;
    newNode->next = target;
    target->prev = newNode;
}
```

```c
// Function to delete the node based on a specific value
void deleteNode(struct Node** head, int value) {
    struct Node* current = *head;

    // Find the node with the given value
    while (current != NULL && current->data != value) {
        current = current->next;
    }

    if (current == NULL) {
        printf("Node with value %d not found\n", value);
        return;
    }

    // Adjust the links to skip the current node
    if (current->prev != NULL)
        current->prev->next = current->next;
    else
        *head = current->next;

    if (current->next != NULL)
        current->next->prev = current->prev;

    // Free the memory occupied by the deleted node
    free(current);
}

// Function to display the doubly linked list
void displayList(struct Node* head) {
    printf("Doubly Linked List: ");
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;

    // Creating a doubly linked list
    head = createNode(1);
    head->next = createNode(2);
```

```
    head->next->prev = head;
    head->next->next = createNode(3);
    head->next->next->prev = head->next;

    // Displaying the original list
    displayList(head);

    // Inserting a new node to the left of the second node
    insertLeft(&head, head->next, 4);
    displayList(head);

    // Deleting a node with a specific value (e.g., 2)
    deleteNode(&head, 2);
    displayList(head);

    return 0;
}
```

OUTPUT:

```
Doubly Linked List: 1 -> 2 -> 3 -> NULL
Doubly Linked List: 1 -> 4 -> 2 -> 3 -> NULL
Doubly Linked List: 1 -> 4 -> 3 -> NULL

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

10.Write a program.

a. To construct Binary Search tree

b. Traverse the tree using inorder , postorder, preorder.

c. Display the elements in the tree.

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

// Node structure for Binary Search Tree
struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to insert a new node into the Binary Search Tree
struct Node* insert(struct Node* root, int value) {
    if (root == NULL) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}

// Function for inorder traversal of BST
void inorderTraversal(struct Node* root) {
```

```
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Function for preorder traversal of BST
void preorderTraversal(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Function for postorder traversal of BST
void postorderTraversal(struct Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

// Function to display elements in the Binary Search Tree
void displayTree(struct Node* root) {
    printf("Elements in the Binary Search Tree: ");
    inorderTraversal(root);
    printf("\n");
}

int main() {
    struct Node* root = NULL;

    // Constructing the Binary Search Tree
    root = insert(root, 10);
    insert(root, 5);
    insert(root, 15);
    insert(root, 7);
    insert(root, 12);

    // Displaying elements in the tree
```

```c
    displayTree(root);

    // Traversing the tree using inorder, preorder, and postorder
    printf("Inorder traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Preorder traversal: ");
    preorderTraversal(root);
    printf("\n");

    printf("Postorder traversal: ");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```

OUTPUT:

```
Doubly Linked List: 1 -> 2 -> 3 -> NULL
Doubly Linked List: 1 -> 4 -> 2 -> 3 -> NULL
Doubly Linked List: 1 -> 4 -> 3 -> NULL

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

11. WAP Breadth First Search

```
#include<stdio.h>
#include<conio.h>
void bfs(int a[20][20], int n, int src, int t[20][2], int s[])
{
int f,r,q[20],u,v,k=0,i;
for(i=1;i<=n;i++)
s[i]=0;
f=r=k=0;
q[r]=src;
s[src]=1;
while(f<=r)
{
u=q[f++];
for(v=1;v<=n;v++)
{
if(a[u][v]==1 && s[v]==0)
{
s[v]=1;
q[++r]=v;
t[k][0]=u;
t[k][1]=v;
k++;
}
}
}
}
void main()
{
int n,a[20][20],src,t[20][2],flag,s[20],i,j;

printf("Enter the number of nodes\n");
scanf("%d", &n);
printf("Enter the adjacency matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
scanf("%d", &a[i][j]);

}
printf("Enter the source\n");
```

```
scanf("%d", &src);
bfs(a,n,src,t,s);
flag=0;
for(i=0;i<n;i++)
{
if(s[i]==0)
{

printf("Vertex %d is not reachable\n", i);
flag=1;
}
else
printf("Vertex %d is reachable\n", i);

}
if(flag==1)
printf("Some nodes are not visited\n");
else
{
printf("The BFS traversal is\n");
for(i=0;i<n;i++)
printf("%d%d\n", t[i][0], t[i][1]);

}
getch();
}
```

OUTPUT:

```
Enter the number of nodes
5
Enter the adjacency matrix
0 1 1 0 0
1 0 1 1 0
1 1 0 0 1
0 1 0 0 1
0 0 1 1 0
Enter the source
0
Vertex 0 is reachable
Vertex 1 is reachable
Vertex 2 is reachable
Vertex 3 is reachable
Vertex 4 is reachable
The BFS traversal is
01
02
13
24
```

12.WAP for Depth First Search:

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

int a[20][20], s[20], n;

void dfs(int v) {
    int i;
    s[v] = 1;

    for (i = 1; i <= n; i++) {
        if (a[v][i] && !s[i]) {
            printf("\n %d->%d", v, i);
            dfs(i);
        }
    }
}

int main() {
    int i, j, count = 0;

    printf("\n Enter number of vertices:");
    scanf("%d", &n);

    for (i = 1; i <= n; i++) {
        s[i] = 0;

        for (j = 1; j <= n; j++) {
            a[i][j] = 0;
        }
    }

    printf("Enter the adjacency matrix:\n");

    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    for (i = 1; i <= n; i++) {
        if (!s[i]) {
```

```
            dfs(i);
        }
    }

    printf("\n");

    for (i = 1; i <= n; i++) {
        if (s[i]) {
            count++;
        }
    }

    if (count == n) {
        printf("Graph is connected");
    } else {
        printf("Graph is not connected");
    }

    return 0;
}
```

OUTPUT:

```
 Enter number of vertices:4
Enter the adjacency matrix:
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0

 1->2
 2->3
 3->4
Graph is connected
Process returned 0 (0x0)   execution time : 33.308 s
Press any key to continue.
```

13.

a.

856. Score of Parentheses

Medium | Topics | Companies

Given a balanced parentheses string `s`, return *the* ***score*** *of the string*.

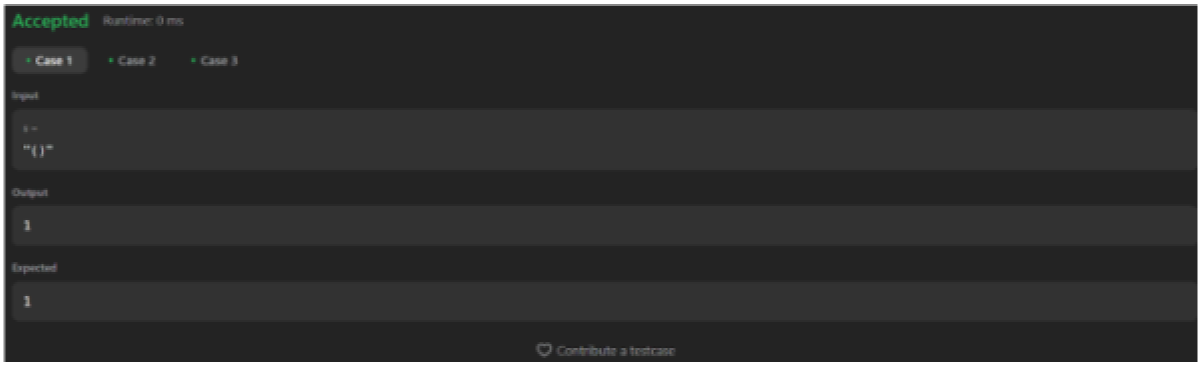
The **score** of a balanced parentheses string is based on the following rule:

- `"()"` has score `1`.
- `AB` has score `A + B`, where `A` and `B` are balanced parentheses strings.
- `(A)` has score `2 * A`, where `A` is a balanced parentheses string.

.

```
int scoreOfParentheses(char* s)
{
int len = strlen(s);
int score = 0;
int depth = 0;
for (int i = 0; i < len; i++)
{
if (s[i] == '(')
{
depth++;
}
else
{
depth--;
if (s[i - 1] == '(')
{
score += 1 << depth;
}
}
}
49| Page
return score;
}
```

OUTPUT:



Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
"()"
Output
1
Expected
1
Contribute a testcase

b.

2095. Delete the Middle Node of a Linked List

Medium | Topics | Companies | Hint

You are given the `head` of a linked list. **Delete** the **middle node**, and return *the* `head` *of the modified linked list*.

The **middle node** of a linked list of size `n` is the `⌊n / 2⌋`th node from the **start** using **0-based indexing**, where `⌊x⌋` denotes the largest integer less than or equal to `x`.

- For `n` = `1`, `2`, `3`, `4`, and `5`, the middle nodes are `0`, `1`, `1`, `2`, and `2`, respectively.

```
struct ListNode* deleteMiddle(struct ListNode* head)
{
int count=0, middleNode, i=0;
struct ListNode* temp=head;
struct ListNode* node=head;
struct ListNode* prev;
struct ListNode* next=head;
while(temp!=NULL)
{
count++;
temp=temp->next;
}
middleNode=count/2;
if(middleNode==0)
{
struct ListNode* newHead=head->next;
free(head);
return newHead;
}
while(i<middleNode)
{
prev=node;
node=node->next;
next=node->next;
i++;
}
prev->next=next;
free(node);
return head;
```

```
}
```

OUTPUT:

Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
head =
[2,1]
Output
[2]
Expected
[2]

c.

328. Odd Even Linked List

Medium · Topics · Companies

Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

```c
struct ListNode* oddEvenList(struct ListNode* head)
{
if(head==NULL||head->next==NULL)
{
return head;
}
struct ListNode* oddTemp=head;
struct ListNode* evenTemp=head->next;
struct ListNode* evenHead=evenTemp;
while(evenTemp!=NULL && evenTemp->next!=NULL)
{
oddTemp->next=evenTemp->next;
oddTemp=oddTemp->next;
evenTemp->next=oddTemp->next;
evenTemp=evenTemp->next;
}
oddTemp->next=evenHead;
return head;
}
```

OUTPUT:

Accepted
Runtime: 2 ms
Case 1
Case 2
Input
head =
[1,2,3,4,5]
Output
[1,3,5,2,4]
Expected
[1,3,5,2,4]

d.

450. Delete Node in a BST

Medium | Topics | Companies

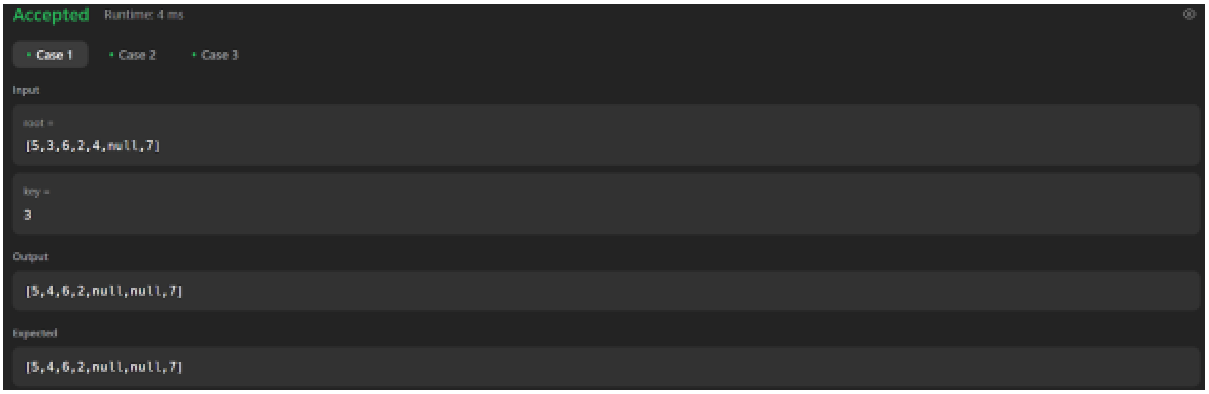
Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return *the* ***root node reference*** *(possibly updated) of the BST*.

Basically, the deletion can be divided into two stages:

1. Search for a node to remove.
2. If the node is found, delete the node.

```
struct TreeNode* deleteNode(struct TreeNode* root, int key)
{
if (root == NULL) return root;
if (key < root->val)
{
root->left = deleteNode(root->left, key);
}
else if (key > root->val)
{
root->right = deleteNode(root->right, key);
}
else
{
if (root->left == NULL) {
struct TreeNode* temp = root->right;
free(root);
return temp;
} else if (root->right == NULL) {
struct TreeNode* temp = root->left;
free(root);
return temp;
}
struct TreeNode* temp = root->right;
while (temp && temp->left != NULL)
temp = temp->left;
root->val = temp->val;
root->right = deleteNode(root->right, temp->val);
}
return root;
}
```

OUTPUT:


Accepted
Runtime: 4 ms
Case 1
Case 2
Case 3
Input
root =
[5,3,6,2,4,null,7]
key =
3
Output
[5,4,6,2,null,null,7]
Expected
[5,4,6,2,null,null,7]

e.

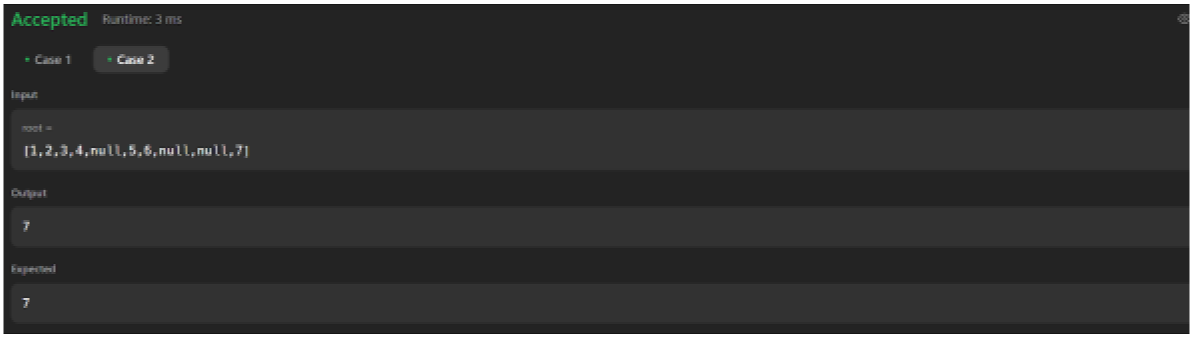
513. Find Bottom Left Tree Value

Medium Topics Companies

Given the `root` of a binary tree, return the leftmost value in the last row of the tree.

```
int findBottomLeftValue(struct TreeNode* root)
{
if (root == NULL)
return -1; // No nodes in the tree
struct TreeNode** queue = (struct TreeNode**)malloc(sizeof(struct TreeNode*)
* 10000);
int front = 0, rear = 0, nextLevelCount = 0, currentLevelCount = 1;
int leftmostValue = root->val;
queue[rear++] = root;
while (front < rear) {
struct TreeNode* current = queue[front++];
currentLevelCount--;
if (current->left != NULL) {
queue[rear++] = current->left;
nextLevelCount++;
}
if (current->right != NULL) {
queue[rear++] = current->right;
nextLevelCount++;
}
if (currentLevelCount == 0) {
if (nextLevelCount > 0)
leftmostValue = queue[front]->val;
currentLevelCount = nextLevelCount;
nextLevelCount = 0;
}
}
free(queue);
return leftmostValue;
}
```

OUTPUT:


Accepted
Runtime: 3 ms
Case 1
Case 2
Input
root =
[1,2,3,4,null,5,6,null,null,7]
Output
7
Expected
7