S.No: 1	Exp. Name: ***Operations on a Single Linked List***	**Date: 2025-08-08**

Aim:

Write a menu-driven C program that implements a singly linked list for the following operations:

1. Insertion at end.
2. Display list.
3. Sort the linked list.
4. Search for an element.

Note: The driver code for taking inputs is provided in the "SLLOps2Main.c", you need to complete the functions in the file "SLLOps2.c".

Source Code:

```
SLLOps2Main.c
```


ID: 24CS079
2024-2028-CSE-B
KPR Institute of Engineering and Technology

```
#include <stdio.h>
#include <stdlib.h>
#include "SLLOps2.c"

// Function prototypes
void insertAtEnd(struct Node** head, int data);
void display(struct Node* head);
void sortList(struct Node** head);
struct Node* search(struct Node* head, int key);

int main() {
    struct Node* head = NULL;
    int choice, data, key;
    struct Node* foundNode;

    printf("1. Insert\n");
    printf("2. Display\n");
    printf("3. Sort\n");
    printf("4. Search\n");
    printf("5. Exit\n");

    do {
        printf("Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Data: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 2:
                display(head);
                break;
            case 3:
                sortList(&head);
                break;
            case 4:
                printf("Key to search: ");
                scanf("%d", &key);
                foundNode = search(head, key);
                if (foundNode != NULL)
                    printf("%d found\n", foundNode->data);
                else
                    printf("%d not found\n", key);
                break;
```

```
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 5);

    return 0;
}
```

SLLOps2.c

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

// Function to insert a node at the end of the list
void insertAtEnd(struct Node** head, int data) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
        newNode->data = data;
        newNode->next = NULL;
        if(*head==NULL){
                *head = newNode;
                return;
        }
        struct Node* temp = *head;
        while(temp->next!=NULL){
                temp = temp->next;
        }
        temp->next = newNode;
}

// Function to display the linked list
void display(struct Node* head) {
        if(head == NULL){
                printf("List is empty\n");
                return;
        }
        struct Node* temp = head;
        while(temp!=NULL){
                printf("%d -> ", temp->data);
                temp = temp->next;
        }
        printf("NULL\n");

}

// Function to sort the linked list in ascending order
void sortList(struct Node** head) {
        if(*head == NULL){
                printf("List is empty\n");
                return;
        }
        struct Node *i, *j;
        int temp;
```

```
        for(i = *head; i!=NULL && i->next!=NULL; i = i->next){
                for(j = i->next; j!=NULL; j=j->next){
                        if(i->data>j->data){
                                temp = i->data;
                                i->data = j->data;
                                j->data = temp;
                        }
                }
        }

    printf("List sorted successfully\n");
}

// Function to search for a node with a given key
struct Node* search(struct Node* head, int key) {
        int p = 1;
        while(head!=NULL){
                if(head->data==key){
                        return head;
                }
                head = head->next;
                p++;
        }
        return NULL;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1. Insert
2. Display
3. Sort
4. Search
5. Exit
Choice:
2
List is empty
Choice:
3
List is empty

Choice:
4
Key to search:
0
0 not found
Choice:
53
Invalid choice
Choice:
1
Data:
1
Choice:
1
Data:
2
Choice:
1
Data:
3
Choice:
2
1 -> 2 -> 3 -> NULL
Choice:
5
Exiting...

Test Case - 2
User Output
1. Insert
2. Display
3. Sort
4. Search
5. Exit
Choice:
1
Data:
6

```
Choice:
1
Data:
5
Choice:
1
Data:
8
Choice:
1
Data:
2
Choice:
1
Data:
10
Choice:
2
6 -> 5 -> 8 -> 2 -> 10 -> NULL
Choice:
3
List sorted successfully
Choice:
2
2 -> 5 -> 6 -> 8 -> 10 -> NULL
Choice:
4
Key to search:
5
5 found
Choice:
4
Key to search:
12
12 not found
Choice:
6
Invalid choice
Choice:
```

```
5
```

```
Exiting...
```

S.No: 2	Exp. Name: ***C program to which performs all operations on singly linked list.***	**Date: 2025-08-12**

Aim:

Write a **C** program that uses functions to perform the following **operations on singly linked list.**

1. Insert at Begin.
2. Insert at End.
3. Insert node after the given node.
4. Delete first node.
5. Delete last node.
6. Delete node after the given node.
7. Delete at position.
8. Traversal.

Source Code:

SingleLL.c

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

#include "AllOperations.c"

void main() {
    NODE first = NULL;
    struct node * head = NULL;
    int x, pos, givenData, op;
    struct node * temp; // Declare temp variable here
    while (1) {
        printf("1.Insert At Begin\n2.Insert At End\n3.Insert node after
the given node\n");
        printf("4.Delete first node\n5.Delete last node\n6.Delete node
after the given node\n7.Delete at position\n");
        printf("8.Traverse the List\n9.Exit\n");
        printf("Enter your option: ");
        scanf("%d", & op);
        switch (op) {
        case 1:
            printf("Enter an element: ");
            scanf("%d", & x);
            first = insertAtBegin(first, x);
            break;
        case 2:
            printf("Enter an element: ");
            scanf("%d", & x);
            first = insertAtEnd(first, x);
            break;
        case 3:
            printf("Enter the data of the given node: ");
            scanf("%d", & givenData);
            printf("Enter the data to be inserted: ");
            scanf("%d", & x);
            temp = first; // Reuse temp variable here
            while (temp != NULL && temp -> data != givenData) {
                temp = temp -> next;
            }
            if (temp == NULL) {
                printf("Given node not found in the list.\n");
            } else {
                insertAfterNode(temp, x);
            }
            break;
        case 4:
            if (first == NULL) {
```

```
                printf("Single Linked List is empty so deletion is not
possible\n");
            } else {
                first = deleteAtBegin(first);
            }
            break;
        case 5:
            if (first == NULL) {
                printf("Single Linked List is empty so deletion is not
possible\n");
            } else {
                first = deleteAtEnd(first);
            }
            break;
        case 6:
            printf("Enter the data of the given node: ");
            scanf("%d", & givenData);
            temp = first; // Reuse temp variable here
            while (temp != NULL && temp -> data != givenData) {
                temp = temp -> next;
            }
            if (temp == NULL) {
                printf("Given node not found in the list.\n");
            } else {
                deleteAfterNode(temp);
            }
            break;
        case 7:
            if (first == NULL) {
                printf("Single Linked List is empty so deletion is not
possible\n");
            } else {
                printf("Enter position: ");
                scanf("%d", & pos);
                first = deleteAtPosition(first, pos);
            }
            break;
        case 8:
            if (first == NULL) {
                printf("Single Linked List is empty\n");
            } else {
                printf("The elements in SLL are: ");
                traverseList(first);
            }
            break;
        case 9:
```

```
            exit(0);
        }
    }
}
```

AllOperations.c

```
struct node {
    int data;
    struct node * next;
};
typedef struct node * NODE;

NODE createNode() {
    NODE temp;
    temp = (NODE) malloc(sizeof(struct node));
    temp -> next = NULL;
    return temp;
}

NODE insertAtBegin(NODE first, int x) {
        NODE newNode = createNode();
        newNode->data = x;
        newNode->next = first;
        first = newNode;
        return newNode;

}

NODE insertAtEnd(NODE first, int x) {
        NODE newNode = createNode();
        newNode->data = x;

        if(first == NULL)









                return newNode;
        NODE temp = first;
        while(temp->next != NULL){
                temp = temp->next;
        }
        temp->next = newNode;
        return first;
```

```
}

void insertAfterNode(struct node * givenNode, int data) {
        if(givenNode == NULL){
                printf("Given node not found in the list.\n");
                return;
        }
        NODE newNode = createNode();
        newNode->data = data;
        newNode->next = givenNode->next;
        givenNode->next = newNode;

        printf("Inserted node with data %d after the given node.\n",
data);
}

NODE deleteAtBegin(NODE first) {
        if(first == NULL){
                printf("List is empty\n");
                return NULL;
        }
        NODE temp = first;
        first = first->next;
        printf("The deleted element from SLL : %d\n", temp->data);
        free(temp);
        return first;

}

NODE deleteAtEnd(NODE first) {
        if(first == NULL){
                printf("List is empty\n");
                return first;
        }
        if(first->next == NULL){
                printf("The deleted item from SLL : %d\n", first-
>data);
                free(first);
                return NULL;
        }
        NODE temp = first;
        while(temp->next->next!=NULL){
                temp = temp->next;
        }
        int dt = temp->next->data;
        free(temp->next);
```

```
        temp->next=NULL;
        printf("The deleted item from SLL : %d\n", dt);
        return first;

}

void deleteAfterNode(struct node * givenNode) {
        if(givenNode == NULL || givenNode->next == NULL){
                printf("Cannot delete. Given node is NULL or the next
node is NULL.\n");
                return;
        }
        NODE temp = givenNode->next;
        int dt = temp->data;
        givenNode->next = temp->next;
        printf("Deleted node with data: %d\n", dt);
        free(temp);

}

NODE deleteAtPosition(NODE first, int pos) {
        if(first == NULL){
                printf("List is empty\n");
                return NULL;
        }
        if(pos<0){
                printf("Invalid position\n");
        }
        if(pos == 1){
                return deleteAtBegin(first);
        }

        NODE temp = first;
        for(int i = 1; temp!=NULL && i<pos-1; i++){
                temp=temp->next;
        }
        if(temp == NULL || temp->next == NULL){
                printf("No such position in SLL so deletion is not
possible\n");
                return first;
        }
        NODE td = temp->next;
        int dv = td->data;
        temp->next = td->next;
        free(td);
        printf("The deleted element from SLL : %d\n", dv);
```

```
        return first;
}

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

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
3
Enter the data of the given node:
15
Enter the data to be inserted:
25
Given node not found in the list.
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position

8.Traverse the List
9.Exit
Enter your option:
6
Enter the data of the given node:
48
Given node not found in the list.
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
1
Enter an element:
101
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
1
Enter an element:
102
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List

```
9.Exit
Enter your option:
1
Enter an element:
103
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
8
The elements in SLL are: 103 --> 102 --> 101 --> NULL
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
2
Enter an element:
99
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
2
```

Enter an element:
98
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
2
Enter an element:
97
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
8
The elements in SLL are: 103 --> 102 --> 101 --> 99 --> 98 --> 97 --> NULL
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
3
Enter the data of the given node:
97

```
Enter the data to be inserted:
96
Inserted node with data 96 after the given node.
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
8
The elements in SLL are: 103 --> 102 --> 101 --> 99 --> 98 --> 97 -->
96 --> NULL
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
4
The deleted element from SLL : 103
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
5
The deleted item from SLL : 96
1.Insert At Begin
```

```
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
8
The elements in SLL are: 102 --> 101 --> 99 --> 98 --> 97 --> NULL
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
6
Enter the data of the given node:
97
Cannot delete. Given node is NULL or the next node is NULL.
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
6
Enter the data of the given node:
96
Given node not found in the list.
1.Insert At Begin
2.Insert At End
```

```
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
6
Enter the data of the given node:
101
Deleted node with data: 99
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
8
The elements in SLL are: 102 --> 101 --> 98 --> 97 --> NULL
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
7
Enter position:
5
No such position in SLL so deletion is not possible
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
```

```
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
7
Enter position:
6
No such position in SLL so deletion is not possible
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
7
Enter position:
4
The deleted element from SLL : 97
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
8
The elements in SLL are: 102 --> 101 --> 98 --> NULL
1.Insert At Begin
2.Insert At End
3.Insert node after the given node
4.Delete first node
```

5.Delete last node
6.Delete node after the given node
7.Delete at position
8.Traverse the List
9.Exit
Enter your option:
9

S.No: 3	Exp. Name: ***Stack using Arrays***	**Date: 2025-08-13**

Aim:

Write a C program to implement stack operations- push, pop, peek, display, isEmpty using arrays.

Source Code:

StackUsingArray.c

```
#include <stdio.h>
#include <stdlib.h>
#define STACK_MAX_SIZE 10
#include "StackOperations.c"

int main() {
        int op, x;
        while(1) {
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek
6.Exit\n");
                printf("Option: ");
                scanf("%d", &op);
                switch(op) {
                        case 1:
                                printf("element: ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                peek();
                                break;
                        case 6:
                                exit(0);
                }
        }
}
```

StackOperations.c

```
// declare the size of the array
int stack[STACK_MAX_SIZE];
int top = -1;
int i;
// define the top to -1

void push(int element) {
        if(top==STACK_MAX_SIZE - 1){
                printf("Stack is overflow\n");
        }else{
                stack[++top] = element;
                printf("Successfully pushed\n");
        }

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

}
void pop() {
        if(top==-1){
                printf("Stack is underflow\n");
        }else{
                printf("Popped value: %d\n", stack[top--]);
        }

}
void peek(){
        if(top==-1){
                printf("Stack is underflow\n");
        }else{
                printf("Peek value: %d\n", stack[top]);
        }

}
void isEmpty() {
        if(top==-1){
                printf("Stack is empty\n");
```

```
        }
        else{
                printf("Stack is not empty\n");
        }

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
4
Stack is empty
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
2
Stack is underflow
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
3
Stack is empty
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
5
Stack is underflow
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
25
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
26

Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
3
Elements: 26 25
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
2
Popped value: 26
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
4
Stack is not empty
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
5
Peek value: 25
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
1
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
2
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:

```
3
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
4
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
5
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
6
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
7
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
8
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
9
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
```

1
element:
10
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
11
Stack is overflow
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
6

S.No: 4	Exp. Name: ***Stack Implementation Using Linked Lists***	**Date: 2025-08-17**

Aim:

Write a program to implement a stack using linked lists.

Input Format:

The user is presented with a menu of options and provides input according to the desired operation:

1. **Push Operation:**
2. Input: Integer value to be pushed onto the stack.
3. **Pop Operation:**
4. No additional input is required.
5. **Display Operation:**
6. No additional input is required.
7. **Is Empty Operation:**
8. No additional input is required.
9. **Peek Operation:**
10. No additional input is required.
11. **Exit Operation:**
12. No additional input is required.

Output Format:

The output will vary depending on the selected option:

1. **Push Operation:**
2. If the stack is not full (no overflow), the output will be: "Successfully pushed."
3. **Pop Operation:**
4. If the stack is not empty, it will print: "Popped value = X" where X is the value removed from the stack. If the stack is empty, it will print: "Stack is underflow."
5. **Display Operation:**
6. If the stack is not empty, it will print: "Elements of the stack are : X Y Z ..." where X, Y, Z, etc., are the elements from top to bottom. If the stack is empty, it will print: "Stack is empty."
7. **Is Empty Operation:**
8. If the stack is empty, it will print: "Stack is empty." If the stack is not empty, it will print: "Stack is not empty."
9. **Peek Operation:**
10. If the stack is not empty, it will print: "Peek value = X" where X is the top element of the stack.If the stack is empty, it will print: "Stack is underflow."
11. **Exit Operation:**
12. The program terminates with no additional output.

Note:

- The partial code has been provided to you in the editor, you are required to fill in the missing code.

Source Code:

StackUsingLL.c

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

int main() {
        int op, x;
        while(1) {
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek
6.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                peek();
                                break;
                        case 6:
                                exit(0);
                }
        }
}
```

StackOperationsLL.c

```
struct stack {
        int data;
        struct stack *next;
};

typedef struct stack *stk;
stk top = NULL;

stk push(int x) {
        stk newNode = (stk )malloc(sizeof(stk));
        if(!newNode){
                printf("Stack overflow\n");
                return NULL;
        }
        newNode->data = x;
        newNode->next = top;
        top = newNode;
        printf("Successfully pushed.\n");
        return newNode;

}

stk pop() {
        if(!top){
                printf("Stack is underflow.\n");
                return NULL;
        }
        stk temp = top;
        int data = temp->data;
        top = top->next;
        printf("Popped value = %d\n", data);
        return temp;

}

void peek() {
        if(!top){
                printf("Stack is underflow.\n");
                return;
        }
        printf("Peek value = %d\n", top->data);

}

void isEmpty() {
        if(!top)
```

```
                printf("Stack is empty.\n");
        else
                printf("Stack is not empty.\n");

}

void display() {
        stk temp = top;
        if(temp == NULL) {
                printf("Stack is empty.\n");
        } else {
                printf("Elements of the stack are : ");
                while(temp != NULL) {
                        printf("%d ", temp -> data);
                        temp = temp -> next;
                }
                printf("\n");
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
33
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
22
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :

55
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
66
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 66 55 22 33
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 66
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 55
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 22 33
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 22
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is not empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option :
2
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
23
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
24
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 24 23
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 24
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 24
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option :
2
Popped value = 23
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

S.No: 5	Exp. Name: ***Operations on Queue using Arrays***	**Date: 2025-08-17**

Aim:

Write a C program to implement queue using **arrays**.

Input Format:

The program will repeatedly prompt the user with a menu to perform operations on the queue.

1. **Enqueue:** The user will be prompted to enter an integer value to be added to the queue.
2. **Dequeue:** No additional input is needed.
3. **Display:** No additional input is needed.
4. **Is Empty:** No additional input is needed.
5. **Size:** No additional input is needed.
6. **Exit:** No additional input is needed.

Output Format:

1. **Enqueue:**
2. If the queue is not full, the output will be: "Successfully inserted." If the queue is full, the output will be: "Queue is overflow."
3. **Dequeue:**
4. If the queue is not empty, the output will be: "Deleted element = X" where X is the element removed from the queue.If the queue is empty, the output will be: "Queue is underflow."
5. **Display:**
6. If the queue is empty, the output will be: "Queue is empty." Otherwise, the output will list the elements in the queue in the order from front to rear, e.g., "Elements in the queue : 10 20"
7. **Is Empty:**
8. If the queue is empty, the output will be: "Queue is empty." Otherwise, the output will be: "Queue is not empty."
9. **Size:**
10. If the queue is empty, the output will be: "Queue size : 0**"** Otherwise, the output will display the current number of elements, e.g., "Queue size : 3"
11. **Exit:**
12. The program terminates without any additional output.

Note:

- The driver code is provided; you need to implement the logic in the file **QueueOperations.c**.
- Refer to visible test cases to strictly match with input/output layout.

Source Code:

QueueUsingArray.c

```
#include <stdlib.h>
#include <stdio.h>
#include "QueueOperations.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size
6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

QueueOperations.c

```
//define a variable MAX to 10
#define MAX 10
//declare the size of the array to be MAX
int queue[MAX];
//define the front and rear to -1
int front = -1, rear = -1;
void enqueue(int x){
        // write your code here to enqueue an element
        if(rear == MAX){
                printf("Queue is overflow.\n");
                return;
        }
        if(front==-1) front=0;
        queue[++rear]=x;
        printf("Successfully inserted.\n");
}
void dequeue() {
        // write your code here to dequeue an element
        if(front==-1 || front>rear){
                printf("Queue is underflow.\n");
                return;
        }
        printf("Deleted element = %d\n", queue[front++]);
}
void display() {
        // write your code here to display the queue
        if(front == -1 || front > rear){
                printf("Queue is empty.\n");
                return;
        }
        printf("Elements in the queue : ");
        for(int i = front; i <= rear; i++){
                printf("%d ", queue[i]);
        }
        printf("\n");
}
void size() {
        // write your code here to get the size of the queue
        int s = 0;
        if(front == -1 || front>rear){
                printf("Queue size : 0\n");
                return;
        }
        for(int i = front; i<=rear; i++){
                s = s + 1;
        }
```

```
        printf("Queue size : %d\n", s);
}
void isEmpty() {
        // write your code here to check whether the queue is empty not
        if(front == -1 || front>rear)
                printf("Queue is empty.\n");
        else
                printf("Queue is not empty.\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
14
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :

1
Enter element :
78
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
53
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 14 78 53
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
25
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 25
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.

1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
65
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 65
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 65
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
63
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 1

1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

S.No: 6	Exp. Name: ***Queue using Linked Lists***	Date: 2025-08-17

Aim:

Write a program that allows users to perform the following operations on a queue:

1. Enqueue an element (add to the rear).
2. Dequeue an element (remove from the front).
3. Display all elements in the queue.
4. Check if the queue is empty.
5. Get the size of the queue.
6. Exit the program.

Input Format:

The program displays a menu with the following options:

1. Enqueue
2. Dequeue
3. Display
4. Is Empty
5. Size
6. Exit

The user selects an option by entering a number corresponding to the desired operation. For the "**Enqueue**" operation, the user is prompted to enter the integer element to be added to the queue.

Output Format:

For each operation, the program outputs the result:

1. For **Enqueue**, print: **Successfully inserted**
2. For **Dequeue**, print: **Deleted value: X** where X is the dequeued element, or **Queue is underflow** if the queue is empty.
3. For **Display**, print: **Elements: A B C ...** showing all elements in the queue or **Queue is empty** if there are no elements.
4. For **Is Empty**, print: **Queue is empty** or **Queue is not empty**.
5. For **Size**, print: **Queue size: N** where N is the number of elements in the queue.
6. For **Exit**, terminate the program without additional output.

Source Code:

QueueUsingLL.c

```
#include <stdlib.h>
#include <stdio.h>
#include "QueueOperationsLL.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size
6.Exit\n");
                printf("Option: ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("element: ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

QueueOperationsLL.c

```
//write your code here
struct Node{
        int data;
        struct Node* next;
};
struct Node* front = NULL;
struct Node* rear = NULL;

void enqueue(int x){
        struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
        if(!temp){
                printf("Successfully inserted\n");
                return;
        }
        temp->data = x;
        temp->next = NULL;

        if(rear == NULL)
                        front = rear = temp;
        else{
                rear->next = temp;
                rear = temp;
        }
        printf("Successfully inserted\n");
}

void dequeue(){
        if(front == NULL){
                printf("Queue is underflow\n");
                return;
        }
        struct Node* temp = front;
        printf("Deleted value: %d\n", front->data);
        front = front->next;

        if(front == NULL)
                rear = NULL;

        free(temp);

}

void display(){
        if(front == NULL){
                printf("Queue is empty\n");
                return;
```

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

void isEmpty(){
        if(front == NULL)
                printf("Queue is empty\n");
        else
                printf("Queue is not empty\n");
}

void size(){
        int count = 0;
        struct Node* temp = front;
        while(temp!=NULL){
                count++;
                temp = temp->next;
        }
        printf("Queue size: %d\n", count);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Queue is underflow
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Queue is empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:

```
4
Queue is empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
5
Queue size: 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
44
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
55
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
66
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
67
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Elements: 44 55 66 67
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 44
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
```

2
Deleted value: 55
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
5
Queue size: 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
4
Queue is not empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
23
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
234
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
45
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:

456
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 23
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Elements: 234 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 234
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Elements: 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
4
Queue is not empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
5
Queue size: 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
6

S.No: 7	Exp. Name: ***C program that implements the quick sort***	**Date: 2025-09-16**

Aim:

Write a C program that implements the quick sort to sort a given list of integers in ascending order.

Input Format

- The program should prompt the user to enter the size of the array and the elements of the array.

Output Format

- After sorting the array, the program should display the sorted array.

Note: Partial code is already given to you in the editor, fill in the remaining code to achieve the task.

Source Code:

quickSort.c

```
#include <stdio.h>

void display(int arr[15], int n) {
        int i;
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
        printf("\n");
}
        void swap(int* a, int* b){
                int temp = *a;
                *a=*b;
                *b=temp;

}
int partition(int arr[15], int lb, int ub) {
        int p = arr[ub];
        int i = (lb-1);
        for(int j=lb; j<ub; j++){
                if(arr[j]<=p){
                        i++;
                        swap(&arr[i],&arr[j]);
                }
        }
        swap(&arr[i+1], &arr[ub]);
        return(i+1);



        //Type your content here







}
void quickSort(int arr[15], int low, int high) {
        int j;
        if (                low < high ) {
                j = partition(        arr,low,high   );
                quickSort(           arr,low,j-1    );
                quickSort(            arr,j+1,high   );
        }
}

void main() {
        int arr[15], i, n;
```

```
        printf("Enter array size : ");
        scanf("%d", &n);
        printf("Enter %d elements : ", n);
        for (i = 0; i < n; i++) {
                scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements are : ");
        display(arr, n);
        quickSort(arr, 0, n - 1);
        printf("After sorting the elements are : ");
        display(arr, n);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter array size :
6
Enter 6 elements :
6 2 8 10 36 14
Before sorting the elements are : 6 2 8 10 36 14
After sorting the elements are : 2 6 8 10 14 36

Test Case - 2
User Output
Enter array size :
8
Enter 8 elements :
95 14 10 23 36 2 5 35
Before sorting the elements are : 95 14 10 23 36 2 5 35
After sorting the elements are : 2 5 10 14 23 35 36 95

S.No: 8	Exp. Name: ***Heap Sort***	Date: 2025-09-16

Aim:

You're a software engineer at a financial services company responsible for developing algorithms to analyze market data and identify trading opportunities. Your team is exploring sorting techniques to efficiently process large datasets of stock prices. Your manager, David, provides the context:

As our trading platform handles a massive volume of market data, it's essential to implement efficient sorting algorithms to identify trends and trading opportunities quickly. One approach is to apply heap sort to arrange the stock prices in descending order, allowing traders to prioritize stocks with the highest potential returns. We need to develop a method to apply **heap sort** to the dataset and arrange the elements in **descending** order to support our trading strategy.

Constraints:

$1 <= N <= 1000$

$-100000 <= arr[i] <= 100000$

Use concept of heaps to sort the data.

Input Format:

The first line contains the integer N, the size of the list

The next line contains N space-separated integers.

Output Format:

Print the sorted list as a row of space-separated integers.

Source Code:

CTC29470.c

```
#include <stdio.h>
void heapify(int arr[], int n, int i){
        int la = i;
        int le = 2*i+1;
        int ri = 2*i+2;
        if(le<n && arr[le]>arr[la])
                la=le;
        if(ri<n && arr[ri]>arr[la])
                la=ri;
        if(la!=i){
                int t = arr[i];
                arr[i] = arr[la];
                arr[la] = t;
                heapify(arr,n,la);
        }
}
void heapsort(int arr[], int n){
        for(int i = n/2-1;i>=0;i--)
                heapify(arr,n,i);
        for(int i = n-1;i>=0;i--){
                int t = arr[0];
                arr[0] = arr[i];
                arr[i]=t;
                heapify(arr,i,0);
        }
}
void printd(int arr[],int n){
        for(int i = n-1; i>=0; i--){
                printf("%d ", arr[i]);
        }
        printf("\n");
}
int main(){
        int n;
        scanf("%d", &n);
        int arr[n];
        for(int i = 0; i<n; i++)
                scanf("%d", &arr[i]);
        heapsort(arr,n);
        printd(arr,n);
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
5
4 8 2 -3 0
8 4 2 0 -3

Test Case - 2
User Output
8
-6 -2 -8 -36 0 69 -5 -96
69 0 -2 -5 -6 -8 -36 -96

S.No: 9	Exp. Name: ***Shell Sort Algorithm***	**Date: 2025-10-18**

Aim:

You are given a task to sort an array of integers using the Shell Sort algorithm. Your program should follow these specifications:

Input Format:

- The first line contains a single integer **n**, which represents the number of elements in the array.
- The second line contains**n** space-separated integers, which are the elements of the array.

Output Format:

- Print the sorted array on a single line, with elements separated by spaces.
- If the number of elements does not match**n**, print **-1** and do not perform any sorting.

Constraints:

- The number of elementsn is between 1 and 1000.
- Each integer in the array is between -1,000,000 and 1,000,000.

Source Code:

CTC38982.c

```
#include <stdio.h>
void main(){
        int n;
        if(scanf("%d", &n)!=1){
                printf("-1\n");
                return;
        }
        int arr[n];
        int c = 0;
        for(int i = 0; i<n; i++){
                if(scanf("%d", &arr[i])==1)
                        c++;
        }
        int co;
        while((co=getchar())!='\n' && co!= EOF)
                c++;
        if(c!=n){
                printf("-1\n");
                return;
        }
        for(int gap = n/2; gap>0; gap/=2){
                for(int i = gap; i<n; i++){
                        int temp = arr[i];
                        int j;
                        for(j = i; j>=gap && arr[j-gap]>temp; j = j -
gap){
                                arr[j] = arr[j-gap];
                        }
                        arr[j] = temp;
                }
        }
        for(int i = 0; i<n; i++){
                printf("%d", arr[i]);
                if(i<n-1)
                        printf(" ");
        }
        printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output
6
10 5 3 8 6 7
3 5 6 7 8 10

Test Case - 2
User Output
3
2 4 6 8
-1

S.No: 10	Exp. Name: ***Bubble Sort***	Date: 2025-10-18

Aim:

Write a C program that reads n integer numbers and arrange them in ascending order using Bubble Sort.

Input Format

- The first line contains an integer n, representing the size of the array.
- The second line contains n space-separated integers, representing the elements of the array.

Output Format

- The program displays the original array before sorting.
- The program displays the sorted array in ascending order.

Note :

- Refer to the visible test cases to strictly match with input/output layout.
- After each output, ensure that a new line is printed.

Source Code:

bubbleSort.c

```c
#include <stdio.h>
void main(){
        int n;
        printf("n: ");
        scanf("%d", &n);
        int arr[n];
        printf("Elements: ");
        for(int i = 0; i<n; i++)
                scanf("%d", &arr[i]);
        printf("Before sorting: ");
        for(int i = 0; i<n; i++)
                printf("%d ", arr[i]);
        printf("\n");
        for(int i = 0; i<n-1; i++){
                for(int j = 0; j<n-i-1; j++){
                        if(arr[j]>arr[j+1]){
                                int t = arr[j];
                                arr[j] = arr[j+1];
                                arr[j+1] = t;
                        }
                }
        }
        printf("After sorting: ");
        for(int i = 0; i<n; i++){
                printf("%d ", arr[i]);
        }
        printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
n:
4
Elements:
44 22 66 11
Before sorting: 44 22 66 11
After sorting: 11 22 44 66

Test Case - 2
User Output
n:
5
Elements:
9 2 7 1 6
Before sorting: 9 2 7 1 6
After sorting: 1 2 6 7 9

S.No: 11	Exp. Name: ***Selection Sort***	**Date: 2025-10-18**

Aim:

Write a C program that prompts the user to enter the size of an array and its elements. The program should then sort the array in ascending order using the Selection Sort algorithm and display the sorted array.

Input Format:

- The first line of input contains an integer $size$ representing the size of the array.
- The second line of input contains $size$ integers separated by spaces, representing the elements of the array.

Output Format:

- The program should display the original array followed by the sorted array, each on a separate line.
- The elements of the array should be separated by spaces.

Source Code:

selection.c

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

void selectionSort(int arr[], int size) {
    for ( int i = 0; i<size-1; i++ ) {
        int minIndex = i;

        for ( int j = i+1; j<size; j++  ) {

            if ( arr[j]<arr[minIndex] ) {

                minIndex = j;
            }
        }

        int temp = arr[i];

        arr[i] = arr[minIndex];

        arr[minIndex] = temp;

    }
}

void displayArray(int arr[], int size) {
    for(int i = 0; i<size; i++)
                printf("%d ", arr[i]);
        printf("\n");
}

int main() {
    int size;

    printf("Enter the size of the array: ");
    scanf("%d", &size);

    int *arr = (int *)malloc(size * sizeof(int));

    printf("Enter %d elements of the array: ", size);
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Original array: ");
    displayArray(arr, size);
```

```
    selectionSort(arr, size);
    printf("Sorted array: ");
    displayArray(arr, size);
    free(arr);

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the size of the array:
4
Enter 4 elements of the array:
2 1 6 9
Original array: 2 1 6 9
Sorted array: 1 2 6 9

Test Case - 2
User Output
Enter the size of the array:
5
Enter 5 elements of the array:
67 3 0 23 7
Original array: 67 3 0 23 7
Sorted array: 0 3 7 23 67

S.No: 12	Exp. Name: ***Hash Table***	Date: 2025-10-18

Aim:

You are entrusted with the task of developing a program in C to manage employee records efficiently. The program operates on a set of **N** employee records, each uniquely identified by a four-digit key.

The program relies on a Hash Table (HT) that is responsible for maintaining these employee records in memory. The Hash Table comprises **m** memory locations, and the set **L** represents the memory addresses, each a two-digit value. Each of these addresses corresponds to one of the four locations within the Hash Table.

Resolve the collision (if any) using **linear probing**.

Input Format:

First line of input contains two space separated integers N, m respectively
Second line of input contains N space separated integer representing the four digit key values

Output Format:

Print the hash table constructed as per the test cases given

Source Code:

HashTable1.c

```
// Type your content here...
#include <stdio.h>
#include<stdlib.h>
int main(){
        int n, m;
        if(scanf("%d %d", &n, &m)!=2) return 0;
        int *keys = NULL;

        if(n>0){
                keys = (int *)malloc(sizeof(int)*n);
                for(int i = 0; i<n; i++)
                        scanf("%d", &keys[i]);
        }

        int *hash = (int *)malloc(sizeof(int)*m);
        for(int i = 0; i<m; i++)
                hash[i]=-1;
        int fail = -1;
        for(int k = 0; k<n;k++){
                int key = keys[k];
                int index = key%m;
                int inserted = 0;
                int pos;
                int probe = 0;
                while(probe<m){
                        pos = (index+probe)%m;
                        if(hash[pos]==-1){
                                hash[pos]=key;
                                inserted=1;
                                break;
                        }
                        probe++;
                }
                if(!inserted){
                        fail=k+1;
                        break;
                }
        }
        if(fail!=-1){
                printf("Hash table is full. Cannot insert records from
key %d\n", fail);
        }
        int empty = 1;
        for(int i = 0; i<m; i++){
                if(hash[i]!=-1){
                        empty = 0;
```

```
                        break;
                }
        }
        if(empty)
                printf("Hash Table is empty\n");
        else{
                for(int i = 0; i<m; ++i){
                        if(hash[i]==-1)
                                printf("T[%d] -> -1\n", i);
                        else
                                printf("T[%d] -> %d\n", i, hash[i]);
                }
        }
        free(keys);
        free(hash);
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
5 5
1596 3574 9512 7532 4268
T[0] -> 4268
T[1] -> 1596
T[2] -> 9512
T[3] -> 7532
T[4] -> 3574

Test Case - 2
User Output
6 4
4587 6589 4521 6523 5824 6548
Hash table is full. Cannot insert records from key 5
T[0] -> 6523
T[1] -> 6589
T[2] -> 4521

```
T[3] -> 4587
```

Test Case - 3
User Output
0 5
Hash Table is empty

S.No: 13	Exp. Name: ***Quadratic Probing Technique***	**Date: 2025-10-18**

Aim:

Smith is enhancing the record management system to use quadratic probing for handling collisions while hashing with a fixed-size hash table. The records are identified by four-digit keys, and Smith wants to efficiently insert these records into the hash table. Your task is to help Smith to write a C program which creates a hash table, inserts records using quadratic probing, and displays the contents.

Input Format:

- First line of input contains two space separated integers N, m representing number if keys and size of the hash table
- Second line of input contains N space separated four-digit integers representing the values.

Output Format:

- If the hash table is empty after all operations, print:

`Hash Table is empty`

- Otherwise, for each index i of the hash table (from 0 to m-1), print:

`T[i] -> <value>`

where value is the value stored at index i.

- If the hash table becomes full and a key cannot be inserted, print:

`Hash table is full. Cannot insert records from key {X}`

where X is the 1-based position of the key that couldn't be inserted.

Note:

- Refer to the sample test cases for better understanding of input and output formats.

Source Code:

`HashTable1.c`

```
// Type your content here...
#include <stdio.h>
#include <stdlib.h>
void insert(int ht[], int m, int key, int count){
        int index = key % m;
        int orgi = index;
        while(ht[index]!=-1){
                index = (index+1)%m;
                if(index == orgi){
                        printf("Hash table is full. Cannot insert
records from key %d\n", count);
                        return;
                }
        }
        ht[index] = key;
}
int main(){
        int n,m;
        int count = 0;
        scanf("%d %d", &n, &m);
        if(n==0){
                printf("Hash Table is empty\n");
                return 0;
        }
        int keys[n];
        for(int i = 0; i<n; i++){
                scanf("%d", &keys[i]);
        }
        int ht[m];
        for(int i = 0; i<m; i++)
                ht[i]=-1;
        for(int i = 0; i<n; i++){
                count++;
                insert(ht, m, keys[i], count);
                if(count > m)
                        break;
        }
        for(int i = 0; i<m; i++){
                printf("T[%d] -> %d\n", i, ht[i]);
        }
        printf("\n");

        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
5 5
1596 3574 9512 7532 4268
T[0] -> 4268
T[1] -> 1596
T[2] -> 9512
T[3] -> 7532
T[4] -> 3574

Test Case - 2
User Output
6 4
4587 6589 4521 6523 5824 6548
Hash table is full. Cannot insert records from key 5
T[0] -> 6523
T[1] -> 6589
T[2] -> 4521
T[3] -> 4587

Test Case - 3
User Output
0 5
Hash Table is empty

S.No: 14	Exp. Name: ***Hashing using modulo division***	**Date: 2025-10-18**

Aim:

Write a c program to implement hashing using modulo division.

Input Format:

The first line reads an integer representing the key value.
The second line reads an integer representing the table size.

Output Format:

The output is an integer representing the hash value modulo division.

Source Code:

ModuloDivisionHashing.c

```
#include <stdio.h>

// Function to perform hash using modulo division
int hashModuloDivision( int key, int tableSize ) {
        //complete the function..
        if(tableSize==0)
                return 0;
        return key%tableSize;
}

int main() {
    int key, tableSize;

    // Input the key and table size from the user
    printf("Key: ");
    scanf("%d", &key);

    printf("Table size: ");
    scanf("%d", &tableSize);

    // Perform hashing using modulo division
    int hashValue = hashModuloDivision(key, tableSize);

    // Display the hash value
    printf("Modulo Division: %d\n", hashValue);

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Key:
10
Table size:
6
Modulo Division: 4

Test Case - 2
User Output
Key:
55
Table size:
11
Modulo Division: 0

Test Case - 3
User Output
Key:
-15
Table size:
4
Modulo Division: -3

S.No: 15	Exp. Name: ***Linear search***	Date: 2025-10-18

Aim:

Write a C program to check whether the given element is present or not in the array of elements using linear search.

Input Format:

- The first line contains an integer n, representing the size of the array.
- The second line contains n space-separated integers, representing the elements of the array.
- The third line contains an integer key, representing the search element.

Output Format:

- If the search element is found, the program displays the message **"Found at position pos"**, where **pos** is the position of the element in the array (0-indexed).
- If the search element is not found, the program displays the message **"key is not found"**, where **key** is the search element.

Note:

- Add new line char \n at the end of the output.
- Refer to the visible test cases to strictly match with input/output layout.

Source Code:

SearchEle.c

ID: 24CS079
2024-2028-CSE-B
KPR Institute of Engineering and Technology

```
// Type Content here...
#include <stdio.h>
int main(){
        int n;
        printf("Enter size: ");
        scanf("%d", &n);
        int arr[n];
        printf("Enter %d element: ", n);
        for(int i = 0; i<n; i++)
                scanf("%d", &arr[i]);
        int k;
        printf("Enter search element: ");
        scanf("%d", &k);
        int f = 0;
        int p = -1;
        for(int i = 0; i<n; i++){
                if(arr[i]==k){
                        f=1;
                        p=i;
                        break;
                }
        }
        if(f)
                printf("Found at position %d\n", p);
        else
                printf("%d is not found\n", k);

        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter size:
6
Enter 6 element:
2 4 8 1 3 5
Enter search element:
6
6 is not found

Test Case - 2
User Output
Enter size:
6
Enter 6 element:
2 4 8 1 3 5
Enter search element:
2
Found at position 0

Test Case - 3
User Output
Enter size:
6
Enter 6 element:
2 4 8 1 3 5
Enter search element:
9
9 is not found

S.No: 16	Exp. Name: ***Non-recursive Binary search***	**Date: 2025-10-18**

Aim:

Write a C program that use non-recursive functions to perform the Binary search operation for a Key value in a given list of integers.

Input Format

- The first line contains an integer n, representing the size of the array.
- The second line contains n space-separated integers, representing the elements of the array in sorted order.
- The third line contains an integer key, representing the search element.

Output Format

- If the search element is found, the program displays the message **"found at pos"**, where **pos** is the position of the element in the array (1-indexed).
- If the search element is not found, the program displays the message **"not found"**.

Note : Fill in the missing code

Source Code:

```
recursiveBinarySearch.c
```

```c
#include <stdio.h>

int binarysearch(int a[], int low, int high, int key) {
    int mid;
    while ( low <= high ) {

        mid = (low + high)/2;

        if ( a[mid]==key ) {


            return mid;

        } else if ( key<a[mid] ) {
                high = mid - 1;


        } else if ( key>a[mid] ){
                        low = mid + 1;
                }
    }
    return -1;
}

int main() {
    int a[20], i, n, key, pos;

    printf("size: ");
    scanf("%d", & n);
    printf("elements: ");
    for (i = 0; i < n; i++) {
        scanf("%d", & a[i]);
    }
    printf("search element: ");
    scanf("%d", & key);
    pos = binarysearch(a, 0, n - 1, key);
    if (pos >= 0) {
        printf("found at %d", pos + 1);
    } else {
        printf("not found");
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
size:
3
elements:
3 6 9
search element:
6
found at 2

Test Case - 2
User Output
size:
3
elements:
3 6 9
search element:
2
not found

S.No: 17	Exp. Name: ***Expression Trees***	Date: 2025-10-24

Aim:

Write a C program that implements Expression Tree

Source Code:

expressiontree.c

```
#include <stdlib.h>
#include<stdio.h>
#include<malloc.h>
struct tree {
        char data;
        struct tree *left;
        struct tree *right;
};
typedef struct tree * ENODE;
ENODE stack[30];
int top = -1;
ENODE newnode(char ch) {
        ENODE temp;
        temp = (ENODE)malloc(sizeof(struct tree));
        temp->data = ch;
        temp->left = NULL;
        temp->right = NULL;
        return(temp);
}
void push(ENODE temp) {
        if(top>=98){
                printf("Stack Overflow");
                exit(1);
        }
        stack[++top] = temp;
}
ENODE pop() {
        if(top<0)
                return NULL;
        return stack[top--];
}
void inorder(ENODE t) {
        if(t!=NULL){
                inorder(t->left);
                printf("%c", t->data);
                inorder(t->right);
        }

}
void preorder(ENODE t){
        if(t!=NULL){
                printf("%c", t->data);
                preorder(t->left);
                inorder(t->right);
        }
}
```

```
void postorder(ENODE t) {
        if(t!=NULL){
                postorder(t->left);
                postorder(t->right);
                printf("%c", t->data);
        }

}
int isOperator(char ch){
        return (ch == '+' || ch == '-' || ch == '*' || ch == '/');
}
/*
void main() {
        char postfix_exp[20];
        ENODE temp;
        int j,i;
        printf("Enter a postfix expression : ");
        scanf("%s",postfix_exp);
        printf("Inorder Traversal of expression tree : ");
        inorder(temp);
        printf("\n");
        printf("Preorder Traversal of expression tree : ");
        preorder(temp);
        printf("\n");
        printf("Postorder Traversal of expression tree : ");
        postorder(temp);
        printf("\n");
}
*/
void main(){
        char postfix_exp[20];
        ENODE temp;
        int j, i;
        printf("Enter a postfix expression : ");
        scanf("%s", postfix_exp);
        for(i = 0; postfix_exp[i]!='\0'; i++){
                char ch = postfix_exp[i];
                temp = newnode(ch);
                if(isOperator(ch)){
                        ENODE right = pop();
                        ENODE left = pop();
                        temp -> right = right;
                        temp ->left = left;
                }
                push(temp);
        }
```

```
        printf("Inorder Traversal of expression tree : ");
        inorder(temp);
        printf("\n");
        printf("Preorder Traversal of expression tree : ");
        preorder(temp);
        printf("\n");
        printf("Postorder Traversal of expression tree : ");
        postorder(temp);
        printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter a postfix expression :
234*-
Inorder Traversal of expression tree : 2-3*4
Preorder Traversal of expression tree : -23*4
Postorder Traversal of expression tree : 234*-

Test Case - 2
User Output
Enter a postfix expression :
1258++*
Inorder Traversal of expression tree : 1*2+5+8
Preorder Traversal of expression tree : *12+5+8
Postorder Traversal of expression tree : 1258++*

Test Case - 3
User Output
Enter a postfix expression :
234*-56**
Inorder Traversal of expression tree : 2-3*4*5*6
Preorder Traversal of expression tree : *-23*45*6
Postorder Traversal of expression tree : 234*-56**

S.No: 18	Exp. Name: ***Prim's Algorithm***	Date: 2025-10-20

Aim:

Given an **adjacency matrix** of a weighted and undirected graph **G**. Find the cost of the minimum spanning tree that covers all the nodes using Prim's Algorithm. Take the elements of the adjacency matrix as a string of integers and spaces.

Constraints:

- 0 < no_of_vertices <= 100

Note:

- The elements of the matrix will always be **N * N**
- Always start from the first node to find the minimum cost
- Take the nodes names as 0,1,2,3,....,N always

Sample Test Case-1:

Input:

4
0 1 2 4 1 0 0 3 2 0 0 5 4 3 5 0

Output:

6

Sample Test Case-2:

Input:

5
0 3 0 7 8 3 0 1 4 0 0 1 0 2 0 7 4 2 0 3 8 0 0 3 0

Output:

9

Source Code:

CTC36469.c

```
#include <stdio.h>
#include <limits.h>
int main(){
        int n;
        scanf("%d", &n);
        int graph[100][100];
        int i, j, k = 0;
        for(i = 0; i<n; i++){
                for(j = 0; j<n; j++){
                        scanf("%d", &graph[i][j]);
                }
        }
        int selected[100] = {0};
        selected[0] = 1;

        int edges = 0;
        int total_cost = 0;

        while(edges < n-1){
                int min = INT_MAX;
                int x=0, y=0;
                for(i = 0; i<n; i++){
                        if(selected[i]){
                                for(j = 0; j<n; j++){
                                        if(!selected[j]&&graph[i]
[j]&&graph[i][j]<min){
                                                min = graph[i][j];
                                                x = i;
                                                y = j;

                                        }
                                }
                        }
                }
                selected[y] = 1;
                total_cost += graph[x][y];
                edges++;
        }
        printf("%d\n", total_cost);

        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
4
0 1 2 4 1 0 0 3 2 0 0 5 4 3 5 0
6

Test Case - 2
User Output
5
0 3 0 7 8 3 0 1 4 0 0 1 0 2 0 7 4 2 0 3 8 0 0 3 0
9

S.No: 19	Exp. Name: ***Basic banking system using goto***	**Date: 2025-10-24**

Aim:

write a C program that simulates a basic banking system? The program should display a menu for users to check their balance, deposit money, withdraw funds, or exit the system. Use the goto statement to facilitate user interactions, and ensure that deposit and withdrawal operations are valid and the process should be carried until the user selects exit option.

Source Code:

banking.c

```
#include <stdio.h>

int main() {
    double balance;
    int choice;
    double amount;

    printf("Enter the amount:");
    scanf("%lf", &balance);
        start:
    printf("Simple Banking System\n");
    printf("Menu:\n");
    printf("1.Check Balance\n");
    printf("2.Deposit Money\n");
    printf("3.Withdraw Money\n");
    printf("4.Exit\n");
    printf("Enter your choice:");
    scanf( "%d", &choice );

    switch ( choice ) {
        case 1:
            printf("balance:%.2f\n",  balance);
                        goto start;

        case 2:
            printf("amount:");
            scanf("%lf", &amount  );
            if ( amount > 0 ) {
                balance = balance + amount;
                printf("new balance:%.2f\n", balance );
            } else {
                printf("Invalid amount\n");
            }
            goto start;
        case 3:
            printf("amount:");
            scanf("%lf", &amount );
            if (amount <= balance ) {
                balance = balance - amount;

                printf("Withdrawal successful! Your new balance
is:%.2f\n", balance  );
            } else {
                printf("Invalid amount or insufficient balance\n");
            }
            goto start;
```

```
        case 4:
            printf("Exiting\n");
            break;
        default:
            printf("Invalid choice\n");
            goto start;
    }

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the amount:
10000
Simple Banking System
Menu:
1.Check Balance
2.Deposit Money
3.Withdraw Money
4.Exit
Enter your choice:
1
balance:10000.00
Simple Banking System
Menu:
1.Check Balance
2.Deposit Money
3.Withdraw Money
4.Exit
Enter your choice:
2
amount:
10000
new balance:20000.00
Simple Banking System
Menu:

1.Check Balance
2.Deposit Money
3.Withdraw Money
4.Exit
Enter your choice:
4
Exiting

Test Case - 2
User Output
Enter the amount:
40000
Simple Banking System
Menu:
1.Check Balance
2.Deposit Money
3.Withdraw Money
4.Exit
Enter your choice:
5
Invalid choice
Simple Banking System
Menu:
1.Check Balance
2.Deposit Money
3.Withdraw Money
4.Exit
Enter your choice:
1
balance:40000.00
Simple Banking System
Menu:
1.Check Balance
2.Deposit Money
3.Withdraw Money
4.Exit
Enter your choice:
4
Exiting

S.No: 20	Exp. Name: ***Shortest Path from source to vertex***	**Date: 2025-10-24**

Aim:

You are required to write a C program that computes the shortest paths from a source vertex to all other vertices in a weighted, directed graph. The program should utilize Dijkstra's algorithm to solve this problem.

The program must perform the following tasks:

Input the Number of Vertices:

- Print: **"Enter the number of vertices: "**
- Read an integer V representing the number of vertices in the graph.

Input the Adjacency Matrix:

- Print: **"adjacency matrix:"**
- For each pair of vertices (i, j), print: **"Weight from i to j: "**
- Read the weight of the edge between vertex i and vertex j. If there is no edge between them, input 0.

Input the Source Vertex:

- Print: **"Enter the source vertex: "**
- Read the source vertex src.

Validation:

- Ensure that the number of vertices does not exceed 100. If it does, print: **"Number of vertices exceeds maximum allowed (100)"** and exit the program.
- Ensure that the source vertex is within the valid range (0 to V-1). If it is not, print: **"Invalid source vertex"** and exit the program.

Compute Shortest Paths:

- Use Dijkstra's algorithm to compute the shortest paths from the source vertex to all other vertices.

Refer to sample test cases for better understanding

Source Code:

Path.c

```c
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define MAX_V 100

int minDistance(int dist[], bool sptSet[], int V) {

        // Write your code here...
        int min = INT_MAX, min_index = -1;
        for(int v = 0; v<V; v++){
                if(!sptSet[v] && dist[v] <= min){
                        min = dist[v];
                        min_index = v;
                }
        }

        return min_index;

}

void printSolution(int dist[], int V) {
    printf("Vertex \t Distance from Source\n");
    for (int i = 0; i < V; i++) {
        printf("%d \t %d\n", i, dist[i]);
    }
}

void dijkstra(int graph[MAX_V][MAX_V], int src, int V) {

        // Write your code here...
        int dist[MAX_V];
        bool sptSet[MAX_V];

        for(int i = 0; i<V; i++){
                dist[i] = INT_MAX;
                sptSet[i] = false;
        }

        dist[src] = 0;

        for(int count = 0; count < V - 1; count++){
                int u = minDistance(dist, sptSet, V);
                if(u == -1){
                        break;
                }
```

```
                sptSet[u] = true;
                for(int v = 0; v<V; v++){
                        if(!sptSet[v] && graph[u][v] > 0 && dist[u]!=
INT_MAX && dist[u] + graph[u][v] < dist[v]){
                                dist[v] = dist[u] + graph[u][v];
                        }
                }
        }

        printSolution(dist,V);


}

int main() {
    int graph[MAX_V][MAX_V];
    int V, src;

    printf("number of vertices: ");
    scanf("%d", &V);

    if (V > MAX_V) {
        printf("Number of vertices exceeds maximum allowed (%d)\n",
MAX_V);
        return 1;
    }

    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            graph[i][j] = 0;
        }
    }

    printf("adjacency matrix:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            printf("Weight from %d to %d: ", i, j);
            scanf("%d", &graph[i][j]);
        }
    }

    printf("Enter the source vertex: ");
    scanf("%d", &src);

    if (src < 0 || src >= V) {
        printf("Invalid source vertex\n");
```

```
        return 1;
    }

    dijkstra(graph, src, V);

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
number of vertices:
3
adjacency matrix:
Weight from 0 to 0:
0
Weight from 0 to 1:
5
Weight from 0 to 2:
10
Weight from 1 to 0:
5
Weight from 1 to 1:
0
Weight from 1 to 2:
2
Weight from 2 to 0:
10
Weight from 2 to 1:
2
Weight from 2 to 2:
0
Enter the source vertex:
0
Vertex Distance from Source
0 0
1 5

2	7

S.No: 21	Exp. Name: ***Book Inventory***	**Date: 2025-10-24**

Aim:

A library is managing its book inventory. Each book has the following information: title, author, publication year, and quantity in stock. Implement a structure to represent a book and write a function to display the details of a book.

Input Format:

- The first line is a string representing the title of the book (can contain spaces).
- The second line is a string representing the author of the book (can contain spaces).
- The third line is an integer representing the publication year.
- The fourth line is an integer representing the quantity in stock.

Output Format:

The output displays the details of the book in the following order:

- Title of the book.
- Author of the book.
- Publication year.
- Quantity in stock.

Example:

Input:

```
Changing India
Dr. Manmohan Singh
2018
15
```

Output:

```
Changing India
Dr. Manmohan Singh
2018
15
```

Note:

- The main function is provided to you in the editor. You just need to fill in the required code.
- Refer to the sample test cases for a better understanding of the input and output format

Source Code:

bookInventory.c

```c
#include <stdio.h>

// Definition of the Book structure
struct Book {
   char title[100];
char author[100];
int publicationYear;
int quantityInStock;

};

// Function to display the details of a book
void displayBookDetails( struct Book book1 ) {
        printf("%s\n", book1.title);
        printf("%s\n", book1.author);
        printf("%d\n", book1.publicationYear);
        printf("%d\n", book1.quantityInStock);

}


int main() {
        struct Book book1;


        scanf(" %[^\n]s", book1.title);
        scanf(" %[^\n]s", book1.author);

        scanf("%d", &book1.publicationYear);
        scanf("%d", &book1.quantityInStock);

        displayBookDetails(book1);

        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Changing India
Dr. Manmohan Singh

2018
15
Changing India
Dr. Manmohan Singh
2018
15

Test Case - 2
User Output
The Call of History
Peter Baker
2019
6
The Call of History
Peter Baker
2019
6