**AIM**

C program to find the sum of 'n' numbers.

**PROGRAM CODE**

#include<stdio.h>

int main()

{

int sum=0,n;

int a[10];

printf("enter the no of elements");

scanf("%d",&n);

printf("enter the array elements");

for(int i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

for(int i=0;i<n;i++)

{

sum=sum+a[i];

}

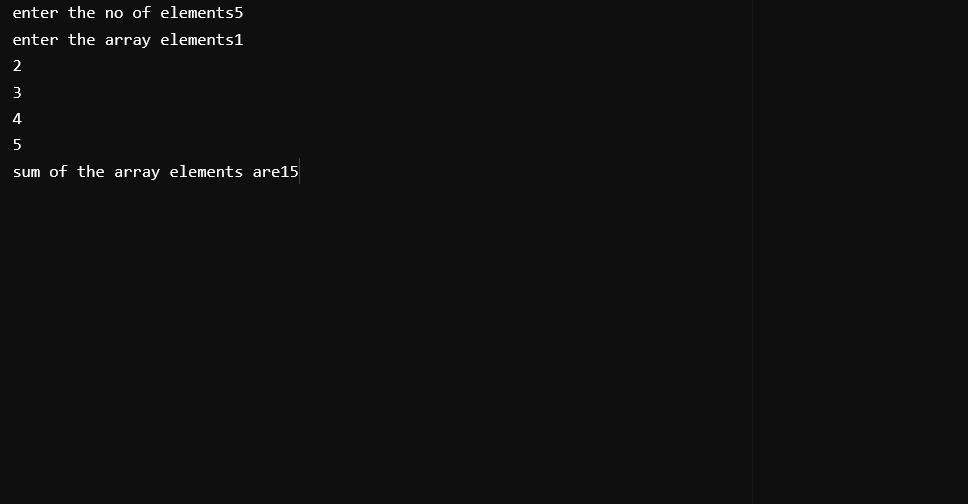
printf("sum of the array elements are");

printf("%d",sum);

return (0);

}

**Output**

****

**AIM**

C program to merge two sorted arrays.

**PROGRAM CODE**

#include<stdio.h>

int a[10], b[10], d[20];

int m, n;

int i, j;

void sort(int c[], int size) {

int temp;

for(i = 0; i < size - 1; i++) {

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

if(c[i] > c[j])

{

temp = c[i];

c[i] = c[j];

c[j] = temp;

}

}

}

}

void merge(int a[],int b[],int size1,int size2)

{

for(i = 0; i < size1; i++) {

d[i] = a[i];

}

for(i = 0; i < size2; i++) {

d[m + i] = b[i];

}

}

void display(int c[], int size) {

for(i = 0; i < size; i++) {

printf("%d\t", c[i]);

}

}

int main() {

printf("Enter the number of elements of array 1:\n");

scanf("%d", &m);

printf("Enter the elements of array 1:\n");

for(i = 0; i < m; i++) {

scanf("%d", &a[i]);

}

printf("Enter the number of elements of array 2:\n");

scanf("%d", &n);

printf("Enter the elements of array 2:\n");

for(i = 0; i < n; i++) {

scanf("%d", &b[i]);

}

printf("the entered unsorted array1\t");

display(a,m);

printf("\nthe entered unsorted array2\t");

display(b,n);

sort(a, m);

sort(b, n);

printf("\nSorted array 1:\n");

display(a, m);

printf("\nSorted array 2:\n");

display(b, n);

int merged\_size = m + n;

printf("\nthe merged array");

merge(a,b,m,n);

display(d, merged\_size);

sort(d, merged\_size);

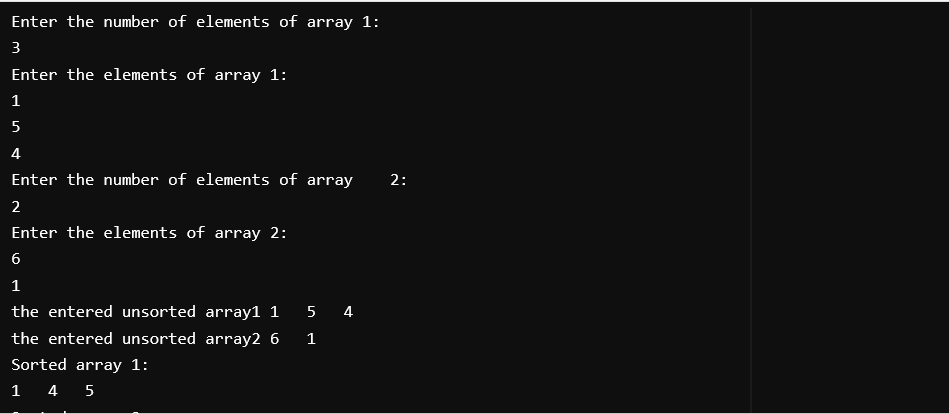
printf("\nMerged and sorted array:\n");

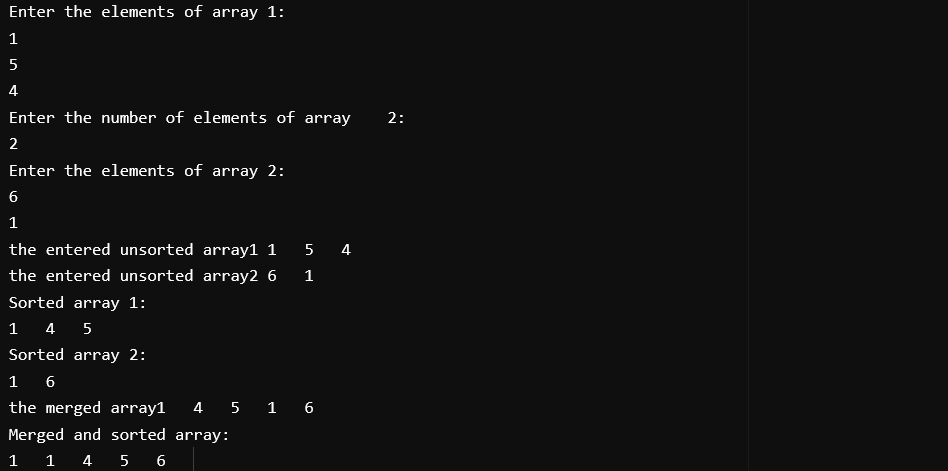
display(d, merged\_size);

return 0;

}

**Output**





**AIM**

C program to create a STACK using array data structure and implement the operations PUSH, POP and TRAVESE.

**PROGRAM CODE**

#include<stdio.h>

#define max 5

int stack[max];

int top=-1;

void push();

void pop();

void traversal();

int main()

{

int choice;

printf("1.push\n2.pop\n3.traversal\n4.exit\n");

do

{

printf("enter your choice");

scanf("%d",&choice);

switch(choice)

{

case 1:push();

break;

case 2:pop();

break;

case 3:traversal();

break;

case 4:printf("program exited\n");

break;

default:printf("wrong choice\n");

break;

}

}while(choice !=4);

}

void push()

{

int element;

if(top==max-1)

{

printf("stack overflow");

}

else

{

printf("enter the element");

scanf("%d",&element);

top++;

stack[top]=element;

}

}

void pop()

{

int element;

if(top==-1)

{

printf("STACK EMPTY");

}

else

{

element=stack[top];

top--;

printf("element deleted is %d",element);

}

}

void traversal()

{

int i;

for(i=0;i<=top;i++)

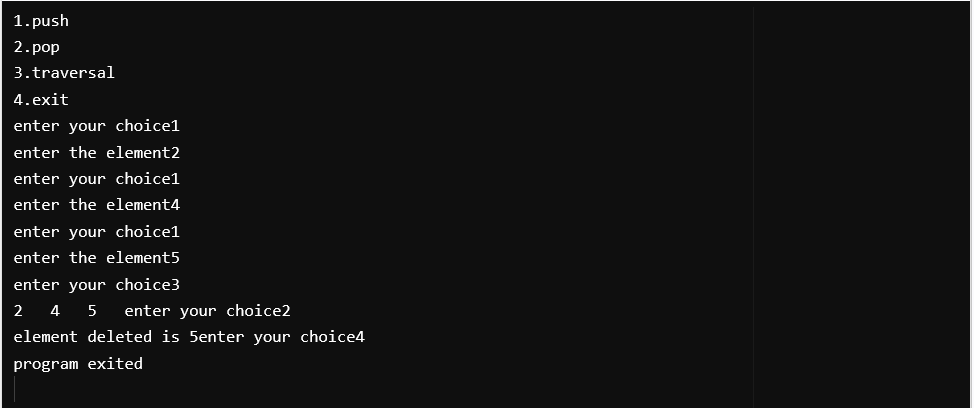
{

printf("%d",stack[i]);

}

}

**Output**



**AIM**

C program to create a QUEUE using array data structure and implement the operations ENQUEUE, DEQUEUE and TRAVERSE.

**PROGRAM CODE**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

int queue[SIZE];

int front = -1;

int rear = -1;

int isFull() {

return rear == SIZE - 1;

}

int isEmpty() {

return front == -1 || front > rear;

}

void enqueue(int value) {

if (isFull()) {

printf("Queue is full.\n");

} else

{

if (front == -1)

front = 0;

rear++;

queue[rear] = value;

printf("Enqueued %d.\n", value);

}

}

int dequeue() {

int value;

if (isEmpty()) {

printf("Queue is empty.\n");

} else {

value = queue[front];

front++;

printf("Dequeued %d.\n", value);

return value;

}

}

void display() {

if (isEmpty()) {

printf("Queue is empty.\n");

} else

{

printf("Queue elements are: ");

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]);

}

printf("\n");

}

}

int main() {

int choice, value;

while (1) {

printf("\n--- Linear Queue Menu ---\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter the value to enqueue: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid choice! Please select a valid option.\n");

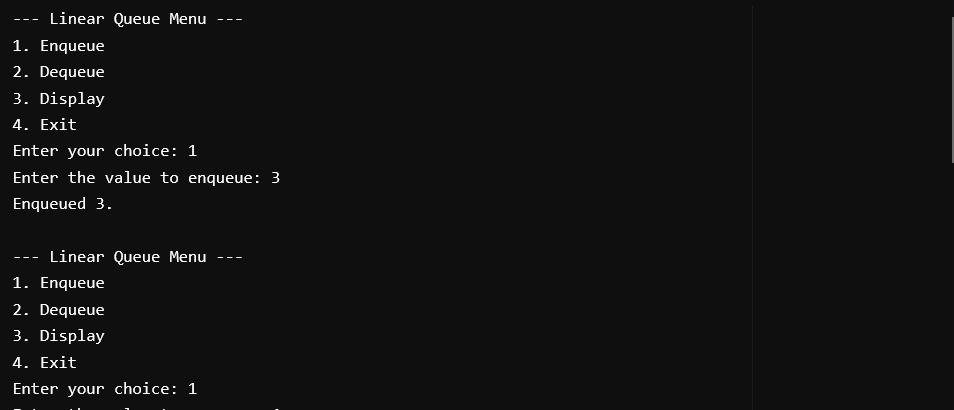
}

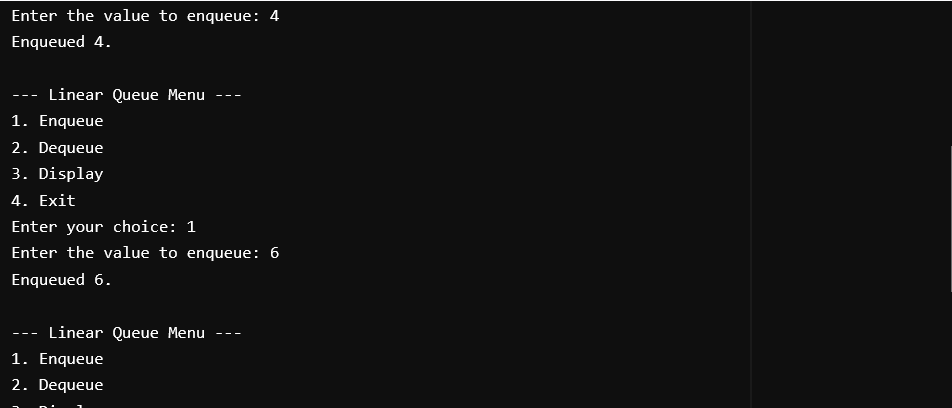
}

return 0;

}

**Output**









**AIM**

Implementation of Circular QUEUE Data Structure,C program to create a QUEUE using array data structure and implement the operations ENQUEUE, DEQUEUE and TRAVERSE.

**PROGRAM CODE**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

int queue[SIZE];

int front = -1;

int rear = -1;

int isFull() {

return (front == (rear + 1) % SIZE);

}

int isEmpty() {

return (front == -1);

}

void enqueue(int value) {

if (isFull()) {

printf("Queue is full.\n");

} else

{

if (front == -1)

front = 0;

rear = (rear + 1) % SIZE;

queue[rear] = value;

printf("Enqueued %d.\n", value);

}

}

int dequeue() {

int value;

if (isEmpty()) {

printf("Queue is empty.\n");

return -1;

} else {

value = queue[front];

if (front == rear) {

front = rear = -1;

} else

{

front = (front + 1) % SIZE;

}

printf("Dequeued %d.\n", value);

return value;

}

}

void display() {

if (isEmpty()) {

printf("Queue is empty.\n");

} else {

int i;

printf("Queue elements are: ");

if (rear >= front) {

for (i = front; i <= rear; i++)

printf("%d ", queue[i]);

} else {

for (i = front; i < SIZE; i++)

printf("%d ", queue[i]);

for (i = 0; i <= rear; i++)

printf("%d ", queue[i]);

}

printf("\n");

}

}

int main() {

int choice, value;

while (1) {

printf("\n--- Circular Queue Menu ---\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to enqueue: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

printf("Exiting program.\n");

exit(0);

default:

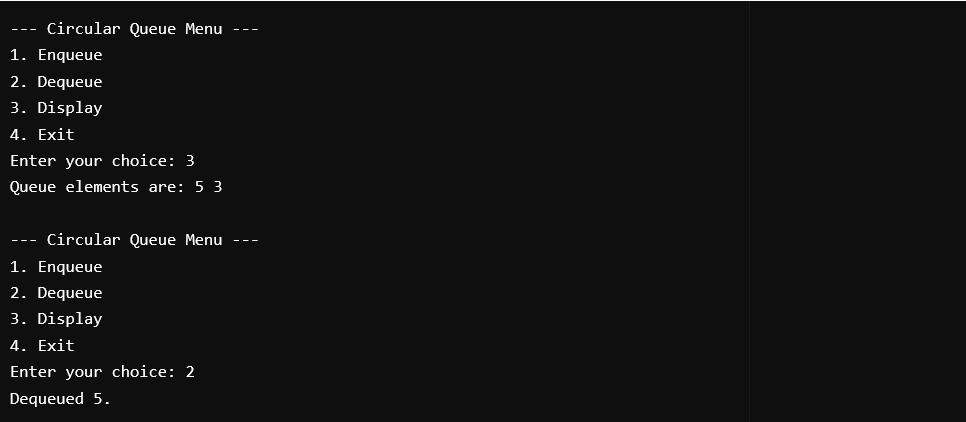
printf("Invalid choice! Please select a valid option.\n");

}

}

return 0;

}

****

****

**AIM**

Write a C program to implement Single Linked List with the following set of operations:

Insert At Front

Insert At End

Insert At Any

Delete At Front

Delete At End

Delete At Any

Search

Display/Traversal

**PROGRAM CODE**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL;

void insertAtFront(int new\_data) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = head;

head = new\_node;

}

void insertAtEnd(int new\_data) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = NULL;

if (head == NULL) {

head = new\_node;

} else {

struct Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = new\_node;

}

}

void insertAtPosition(int new\_data, int position) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->data = new\_data;

if (position == 0) {

new\_node->next = head;

head = new\_node;

return;

}

struct Node\* temp = head;

for (int i = 0; i < position - 1 && temp != NULL; i++) {

temp = temp->next;

}

if (temp == NULL) {

printf("Position out of bounds.\n");

free(new\_node);

} else {

new\_node->next = temp->next;

temp->next = new\_node;

}

}

void deleteAtFront() {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

struct Node\* temp = head;

head = head->next;

free(temp);

printf("Node deleted from the front.\n");

}

void deleteAtEnd() {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

struct Node\* temp = head;

struct Node\* prev = NULL;

if (head->next == NULL) {

free(head);

head = NULL;

printf("Node deleted from the end.\n");

return;

}

while (temp->next != NULL) {

prev = temp;

temp = temp->next;

}

prev->next = NULL;

free(temp);

printf("Node deleted from the end.\n");

}

void deleteAtPosition(int position) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

struct Node\* temp = head;

struct Node\* prev = NULL;

if (position == 0) {

head = head->next;

free(temp);

printf("Node deleted from position 0.\n");

return;

}

for (int i = 0; i < position && temp != NULL; i++) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) {

printf("Position out of bounds.\n");

} else {

prev->next = temp->next;

free(temp);

printf("Node deleted from position %d.\n", position);

}

}

void search(int value) {

struct Node\* temp = head;

int position = 0;

while (temp != NULL) {

if (temp->data == value) {

printf("Value %d found at position %d.\n", value, position);

return;

}

temp = temp->next;

position++;

}

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

}

void displayList() {

struct Node\* temp = head;

if (head == NULL) {

printf("The list is empty.\n");

} else {

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

}

int main() {

int choice, data, position, value;

while (1) {

printf("\nMenu:\n");

printf("1. Insert at the Front\n");

printf("2. Insert at the End\n");

printf("3. Insert at a Specific Position\n");

printf("4. Delete from the Front\n");

printf("5. Delete from the End\n");

printf("6. Delete from a Specific Position\n");

printf("7. Search for a Value\n");

printf("8. Display the List\n");

printf("9. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the data to insert at the front: ");

scanf("%d", &data);

insertAtFront(data);

break;

case 2:

printf("Enter the data to insert at the end: ");

scanf("%d", &data);

insertAtEnd(data);

break;

case 3:

printf("Enter the data to insert: ");

scanf("%d", &data);

printf("Enter the position to insert at: ");

scanf("%d", &position);

insertAtPosition(data, position);

break;

case 4:

deleteAtFront();

break;

case 5:

deleteAtEnd();

break;

case 6:

printf("Enter the position to delete from: ");

scanf("%d", &position);

deleteAtPosition(position);

break;

case 7:

printf("Enter the value to search for: ");

scanf("%d", &value);

search(value);

break;

case 8:

printf("The linked list is: ");

displayList();

break;

case 9:

printf("Exiting...\n");

exit(0);

default:

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

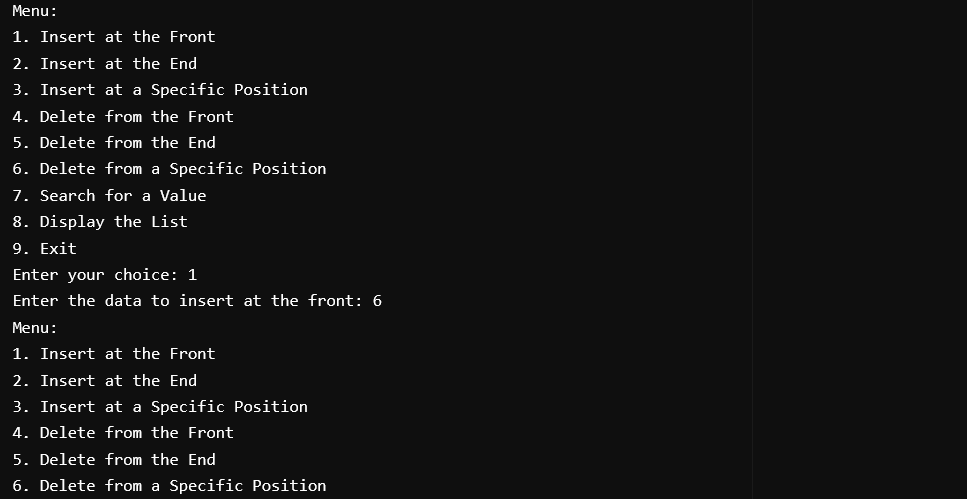
}

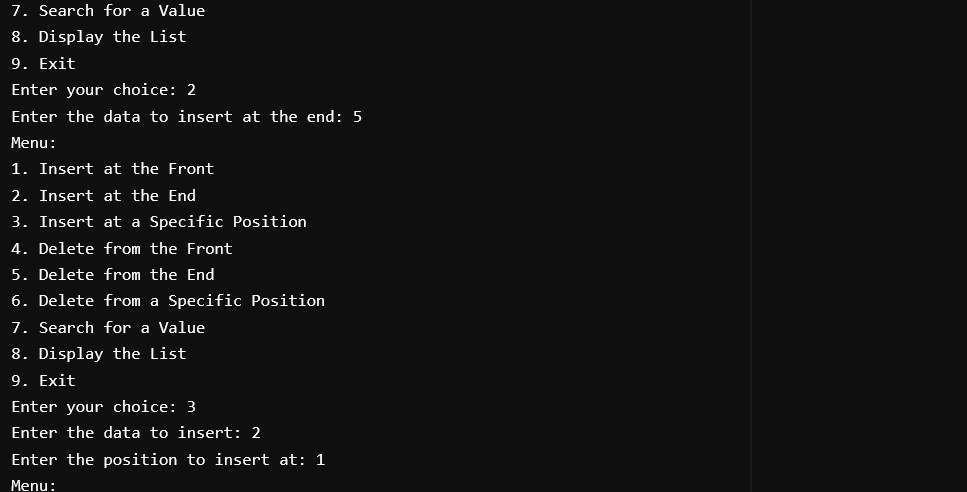
}

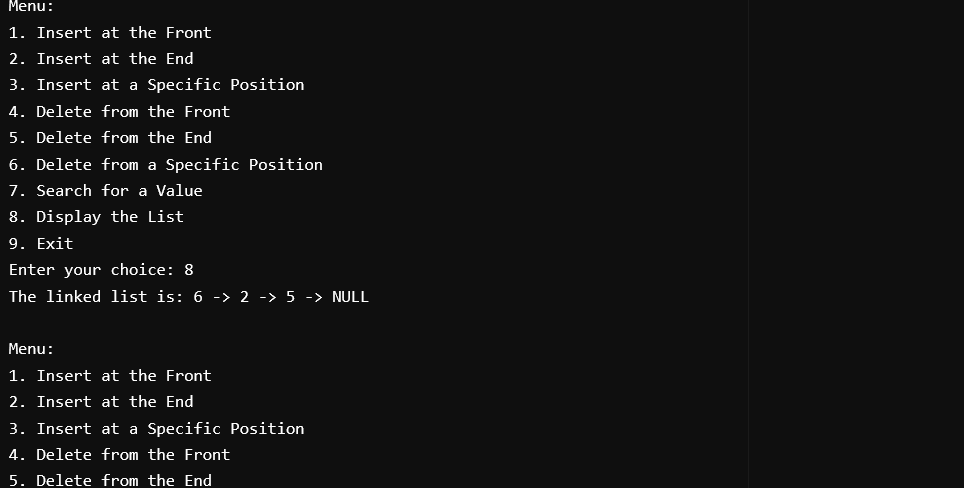
return 0;

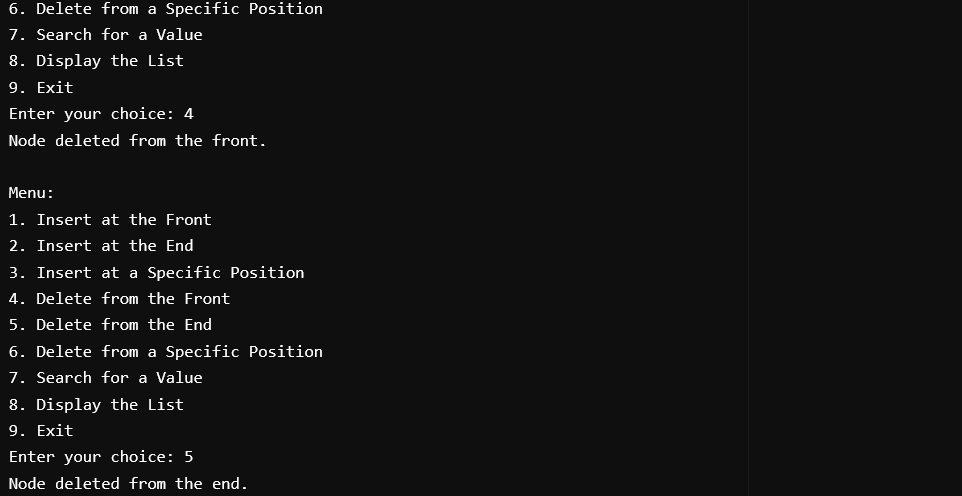
}

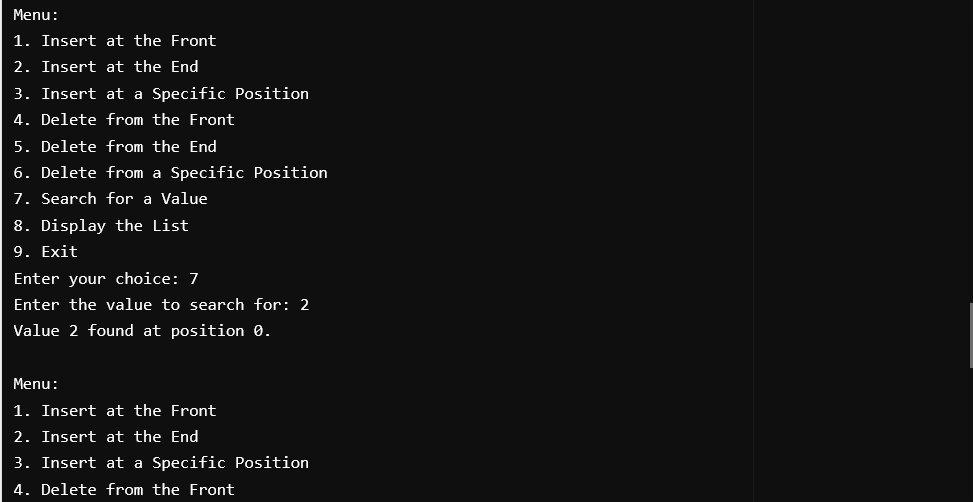
**output**

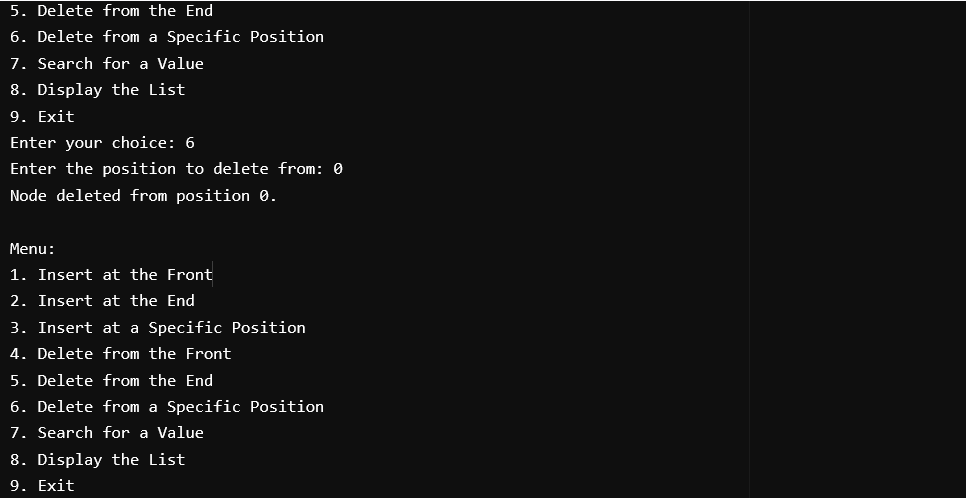


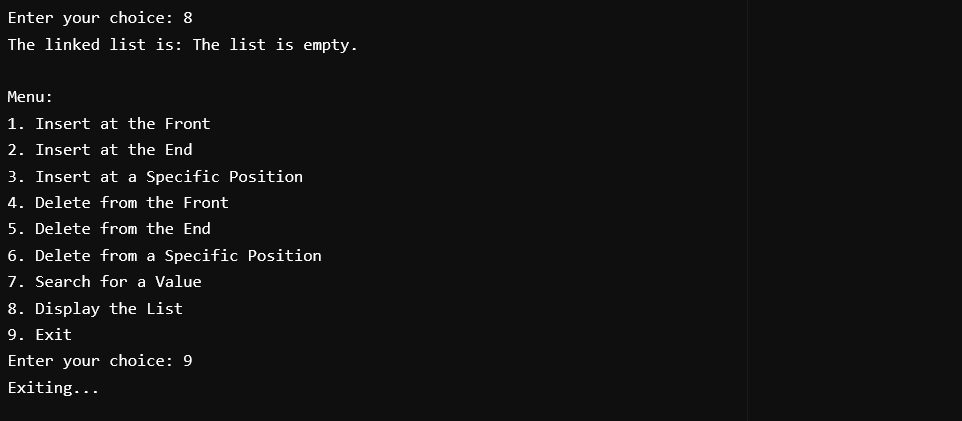












**AIM**Write a C program to implement Double Linked List with the following set of operations:

1.Insert At Front

2.Insert At End

3.Insert At Any

4.Delete At Front

5.Delete At End

6.Delete At Any

7.Search

8.Display/Traversal

**PROGRAM CODE**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*Llink;

struct Node \*Rlink;

};

struct Node \*header = NULL;

struct Node \*CreateNode(int data)

{

struct Node \*newnode;

newnode = (struct Node\*)malloc(sizeof(struct Node));

newnode->data = data;

newnode->Llink = NULL;

newnode->Rlink = NULL;

return newnode;

}

void insertatfront(int data)

{

struct Node \*newnode;

newnode = CreateNode(data);

if (header == NULL)

{

header = newnode;

}

else

{

newnode->Rlink = header;

header->Llink = newnode;

header = newnode;

}

}

void insertatend(int data)

{

struct Node \*ptr;

struct Node \*newnode;

newnode = CreateNode(data);

if (header == NULL)

{

header = newnode;

}

else

{

ptr = header;

while (ptr->Rlink != NULL)

{

ptr = ptr->Rlink;

}

ptr->Rlink = newnode;

newnode->Llink = ptr;

}

}

void insertatany(int key, int data)

{

struct Node \*ptr = header;

struct Node \*newnode;

while (ptr != NULL && ptr->data != key)

{

ptr = ptr->Rlink;

}

if (ptr == NULL)

{

printf("Key not found, insertion not possible\n");

return;

}

newnode = CreateNode(data);

newnode->Rlink = ptr->Rlink;

if (ptr->Rlink != NULL)

{

ptr->Rlink->Llink = newnode;

}

ptr->Rlink = newnode;

newnode->Llink = ptr;

}

void deleteatfront()

{

struct Node \*ptr;

if (header == NULL)

{

printf("List is empty\n");

}

else

{

ptr = header;

header = header->Rlink;

if (header != NULL)

{

header->Llink = NULL;

}

free(ptr);

printf("Node deleted from the front\n");

}

}

void deleteatend()

{

if (header == NULL)

{

printf("List is empty\n");

}

else

{

struct Node \*ptr;

struct Node \*ptr1;

ptr = header;

while (ptr->Rlink != NULL)

{

ptr1 = ptr;

ptr = ptr->Rlink;

}

ptr1->Rlink = NULL;

free(ptr);

printf("Node deleted from the end\n");

}

}

void deleteAtAny(int key)

{

if (header == NULL)

{

printf("List is empty\n");

return;

}

struct Node \*ptr = header;

struct Node \*prev = NULL;

while (ptr != NULL && ptr->data != key)

{

prev = ptr;

ptr = ptr->Rlink;

}

if (ptr == NULL)

{

printf("Key not found, deletion not possible\n");

return;

}

if (prev == NULL)

{

header = ptr->Rlink;

}

else

{

prev->Rlink = ptr->Rlink;

}

if (ptr->Rlink != NULL)

{

ptr->Rlink->Llink = prev;

}

free(ptr);

printf("Node with key %d deleted\n", key);

}

void traversal()

{

struct Node \*ptr = header;

if (ptr == NULL)

{

printf("List is empty\n");

return;

}

while (ptr != NULL)

{

printf("%d ", ptr->data);

ptr = ptr->Rlink;

}

printf("\n");

}

void searchNode(int key)

{

struct Node \*ptr = header;

while (ptr != NULL)

{

if (ptr->data == key)

{

printf("Key %d found in the list\n", key);

return;

}

ptr = ptr->Rlink;

}

printf("Key %d not found in the list\n", key);

}

int main()

{

int choice, data, key;

do

{

printf("\nMenu:\n");

printf("1. Insert at front\n");

printf("2. Insert at end\n");

printf("3. Insert at any position\n");

printf("4. Delete from front\n");

printf("5. Delete from end\n");

printf("6. Delete at any position\n");

printf("7. Traverse the list\n");

printf("8. Search a node\n");

printf("9. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter data to insert at front: ");

scanf("%d", &data);

insertatfront(data);

break;

case 2:

printf("Enter data to insert at end: ");

scanf("%d", &data);

insertatend(data);

break;

case 3:

printf("Enter key to insert after: ");

scanf("%d", &key);

printf("Enter data to insert: ");

scanf("%d", &data);

insertatany(key, data);

break;

case 4:

deleteatfront();

break;

case 5:

deleteatend();

break;

case 6:

printf("Enter key to delete: ");

scanf("%d", &key);

deleteAtAny(key);

break;

case 7:

traversal();

break;

case 8:

printf("Enter key to search: ");

scanf("%d", &key);

searchNode(key);

break;

case 9:

printf("Exiting...\n");

break;

default:

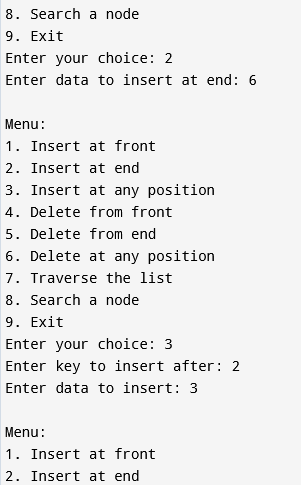
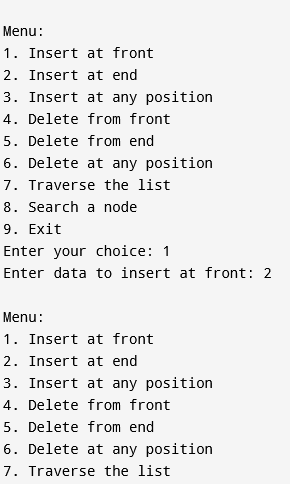
printf("Invalid choice. Please try again.\n");

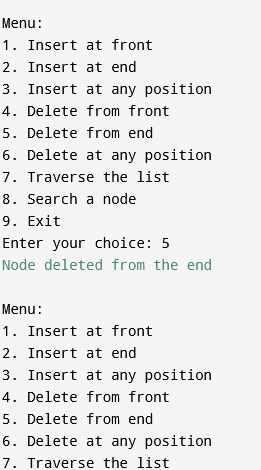
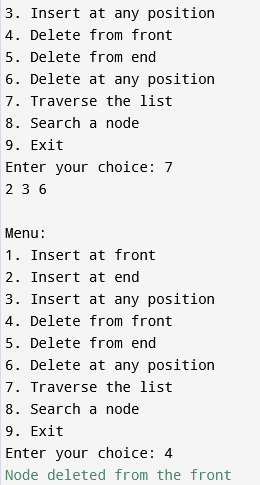
}

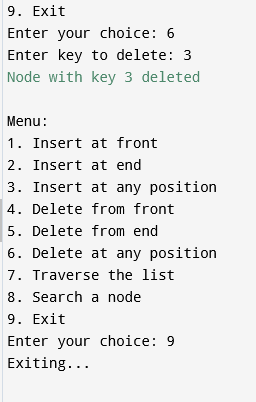
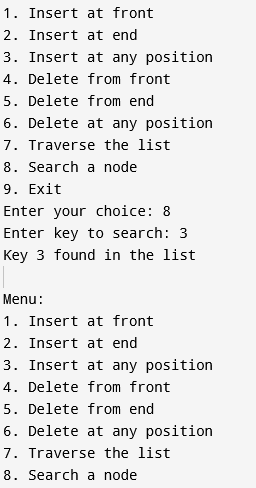
} while (choice != 9);

return 0;

}







**AIM**

Write a C program to implement Stack using Single Linked List

**PROGRAM CODE**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* top = NULL;

int size = 0;

int maxSize = 5;

int isEmpty() {

return size == 0;

}

int isFull() {

return size == maxSize;

}

void push(int data) {

if (isFull()) {

printf("Stack Overflow! Cannot push %d\n", data);

return;

}

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

printf("Memory allocation failed\n");

return;

}

newNode->data = data;

newNode->next = top;

top = newNode;

size++;

printf("Pushed %d onto the stack\n", data);

}

int pop() {

if(isEmpty()) {

printf("Stack Underflow! Cannot pop\n");

return -1;

}

struct Node\* temp = top;

int poppedData = temp->data;

top = temp->next;

free(temp);

size--;

return poppedData;

}

void display() {

if (isEmpty()) {

printf("Stack is empty\n");

return;

}

struct Node\* current = top;

printf("Stack elements: ");

while (current) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

int main() {

int choice, element;

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 the element to push: ");

scanf("%d", &element);

push(element);

break;

case 2:

element = pop();

if (element != -1) {

printf("Popped element: %d\n", element);

}

break;

case 3:

display();

break;

case 4:

printf("Exiting...\n");

break;

default:

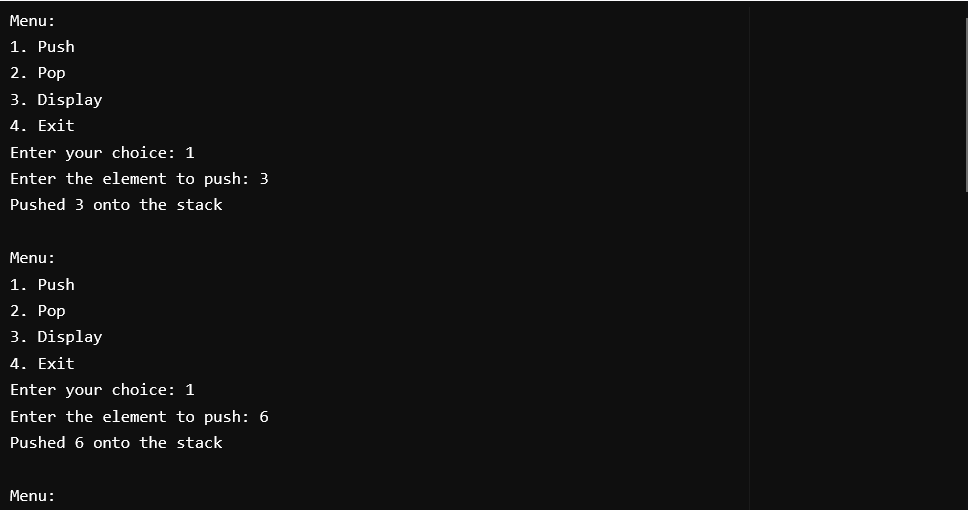
printf("Invalid choice! Please try again.\n");

}

}while (choice != 4);

return 0;

}





**AIM**

Write a C program to implement Set data structure using Bit Vector representation and perform operations such Union, Intersection, Set Complement and Set Difference.

**PROGRAM CODE**

#include <stdio.h>

#include <string.h>

#define MAX 32

void initializeBitVector(int bitVector[], int size) {

for (int i = 0; i < size; i++) {

bitVector[i] = 0;

}

}

void displayBitVector(int bitVector[], int size) {

for (int i = 0; i < size; i++) {

printf("%d", bitVector[i]);

}

printf("\n");

}

void displaySet(int bitVector[], int size, char universalSet[]) {

printf("{ ");

for (int i = 0; i < size; i++) {

if (bitVector[i] == 1)

printf("%c ", universalSet[i]);

}

printf("}\n");

}

void unionSet(int set1[], int set2[], int result[], int size) {

for (int i = 0; i < size; i++) {

result[i] = set1[i] | set2[i];

}

}

void intersectionSet(int set1[], int set2[], int result[], int size) {

for (int i = 0; i < size; i++) {

result[i] = set1[i] & set2[i];

}

}

void complementSet(int set[], int result[], int size) {

for (int i = 0; i < size; i++) {

if (set[i] == 0)

result[i] = 1;

else

result[i] = 0;

}

}

void differenceSet(int set1[], int set2[], int result[], int size) {

for (int i = 0; i < size; i++) {

result[i] = set1[i] & !set2[i];

}

}

void setToBitVector(char set[], int bitVector[], int setSize, int universalSize, char universalSet[]) {

initializeBitVector(bitVector, universalSize);

for (int i = 0; i < setSize; i++) {

for (int j = 0; j < universalSize; j++) {

if (set[i] == universalSet[j]) {

bitVector[j] = 1;

break;

}

}

}

}

int main() {

char universalSet[MAX], set1[MAX], set2[MAX];

int bitVector1[MAX], bitVector2[MAX], result[MAX];

int sizeU, size1, size2;

int choice;

printf("Enter the size of the universal set (max %d): ", MAX);

scanf("%d", &sizeU);

if (sizeU > MAX || sizeU <= 0) {

printf("Invalid size for the universal set. Exiting.\n");

return 1;

}

printf("Enter the elements of the universal set (characters): ");

for (int i = 0; i < sizeU; i++) {

scanf(" %c", &universalSet[i]);

}

printf("Enter the size of Set 1: ");

scanf("%d", &size1);

printf("Enter the elements of Set 1 (characters): ");

for (int i = 0; i < size1; i++) {

scanf(" %c", &set1[i]);

}

printf("Enter the size of Set 2: ");

scanf("%d", &size2);

printf("Enter the elements of Set 2 (characters): ");

for (int i = 0; i < size2; i++) {

scanf(" %c", &set2[i]);

}

setToBitVector(set1, bitVector1, size1, sizeU, universalSet);

setToBitVector(set2, bitVector2, size2, sizeU, universalSet);

printf("\nBit Vector Representation of Set 1: ");

displayBitVector(bitVector1, sizeU);

printf("Set 1: ");

displaySet(bitVector1, sizeU, universalSet);

printf("\nBit Vector Representation of Set 2: ");

displayBitVector(bitVector2, sizeU);

printf("Set 2: ");

displaySet(bitVector2, sizeU, universalSet);

do {

printf("\nMenu:\n");

printf("1. Union\n");

printf("2. Intersection\n");

printf("3. Complement of Set 1\n");

printf("4. Complement of Set 2\n");

printf("5. Difference (Set 1 - Set 2)\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

unionSet(bitVector1, bitVector2, result, sizeU);

printf("Union Bit Vector: ");

displayBitVector(result, sizeU);

printf("Union Set: ");

displaySet(result, sizeU, universalSet);

break;

case 2:

intersectionSet(bitVector1, bitVector2, result, sizeU);

printf("Intersection Bit Vector: ");

displayBitVector(result, sizeU);

printf("Intersection Set: ");

displaySet(result, sizeU, universalSet);

break;

case 3:

complementSet(bitVector1, result, sizeU);

printf("Complement of Set 1 Bit Vector: ");

displayBitVector(result, sizeU);

printf("Complement of Set 1: ");

displaySet(result, sizeU, universalSet);

break;

case 4:

complementSet(bitVector2, result, sizeU);

printf("Complement of Set 2 Bit Vector: ");

displayBitVector(result, sizeU);

printf("Complement of Set 2: ");

displaySet(result, sizeU, universalSet);

break;

case 5:

differenceSet(bitVector1, bitVector2, result, sizeU);

printf("Difference (Set 1 - Set 2) Bit Vector: ");

displayBitVector(result, sizeU);

printf("Difference (Set 1 - Set 2): ");

displaySet(result, sizeU, universalSet);

break;

case 6:

printf("Exiting program.\n");

break;

default:

printf("Invalid choice. Please try again.\n");

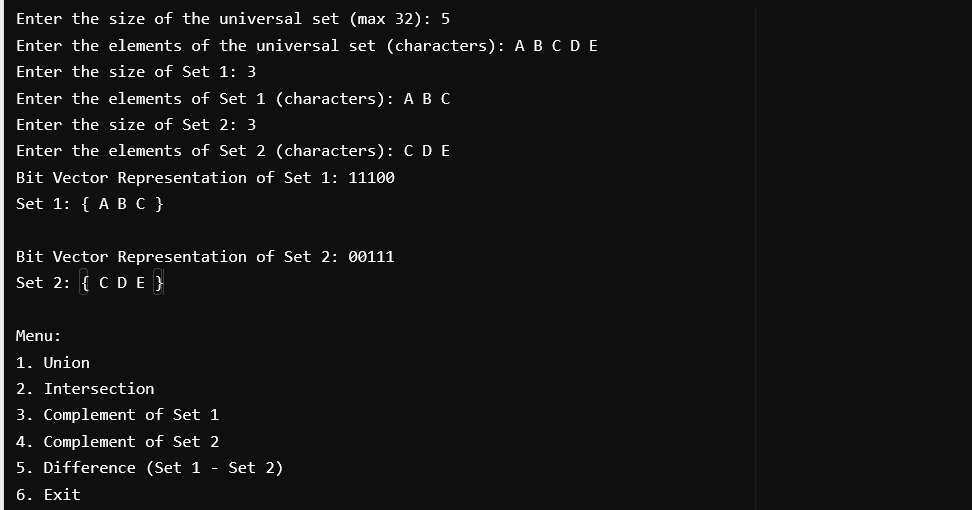
}

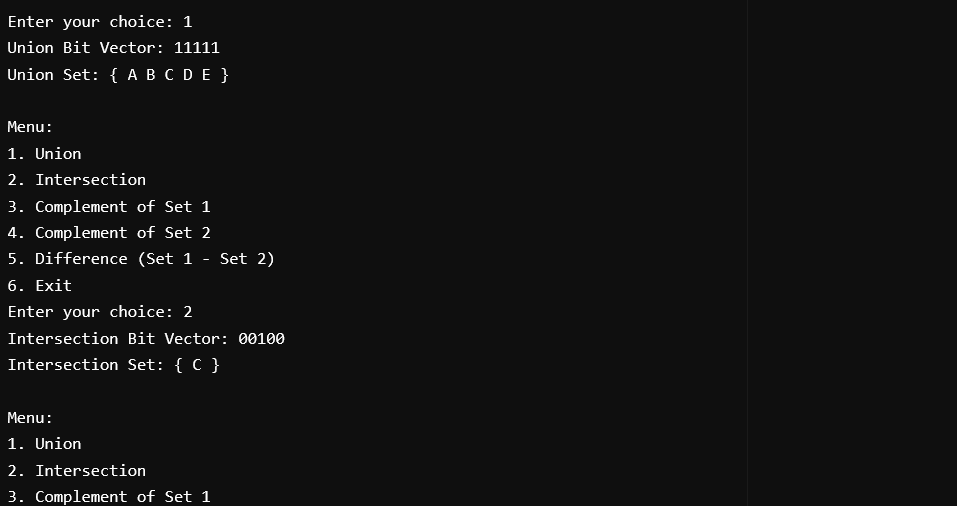
} while (choice != 6);

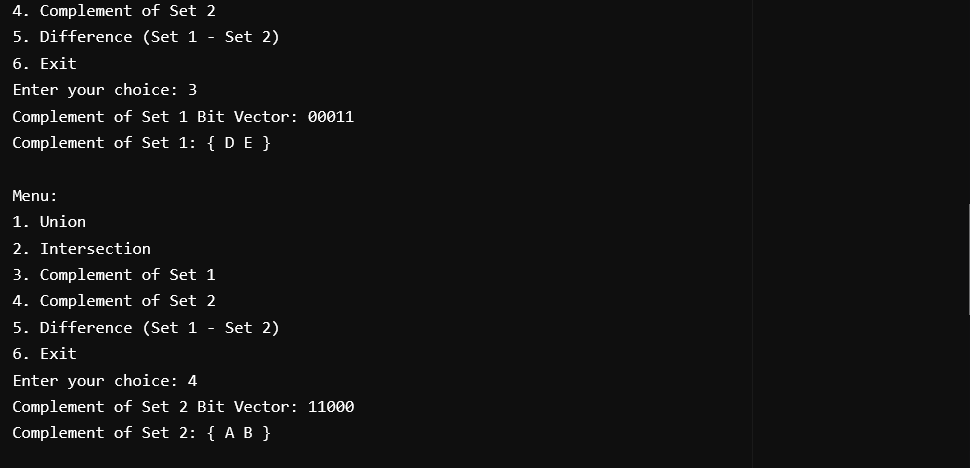
return 0;

}

**OUTPUT**









**AIM**

Write a C program to implement Disjoint data structure using an array and perform operations such as MakeSet, FindSet and Union (using union-rank algorithm)

**PROGRAM CODE**

#include <stdio.h>

#define MAX 100

int rank[MAX];

void makeset(int n) {

for (int i = 1; i <= n; i++) {

rank[i] = -1;

}

printf("after makeset rank matrix is :");

for (int i = 1; i <= n; i++) {

printf("%d",rank[i]);

}

}

int findset(int x) {

if (rank[x] < 0) {

return x;

}

rank[x] = findset(rank[x]);

return rank[x];

}

void unionByRank(int x, int y) {

int rootX = findset(x);

int rootY = findset(y);

if (rootX != rootY) {

if (rank[rootX] <= rank[rootY]) {

rank[rootX] += rank[rootY];

rank[rootY] = rootX;

} else {

rank[rootY] += rank[rootX];

rank[rootX] = rootY;

}

}

}

void printRanks(int n) {

printf("Rank Array: ");

for (int i = 1; i <= n; i++) {

printf("%d ", rank[i]);

}

printf("\n");

}

int main() {

int n, choice, x, y;

printf("Enter the number of elements: ");

scanf("%d", &n);

while (1) {

printf("\nMenu:\n");

printf("1. make-Set\n");

printf("2. Find-Set\n");

printf("3. Union\n");

printf("4. Print Rank Array\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:makeset(n);

break;

case 2:

printf("Enter element to find its set: ");

scanf("%d", &x);

printf("Representative of %d is %d\n", x, findset(x));

break;

case 3:

printf("Enter two elements to perform union: ");

scanf("%d %d", &x, &y);

unionByRank(x, y);

break;

case 4:

printRanks(n);

break;

case 5:

printf(“exiting”);

break;

default:

printf("Invalid choice! Try again.\n");

}

}

return 0;

}

**Output**

