DATA STRUCTURES AND ALGORITHMS LAB ASSIGNMENT – 3

Student Details

- Name: Pratyush Deo Singh
- Registration Number: 25BCE5101
- Course / Branch: B Tech. CSE (Core)
- Semester: 2nd
- Subject: Data Structures and Algorithms
- Faculty Name: Dr. Malini A

Program 1: Largest Element in an Array using Pointers

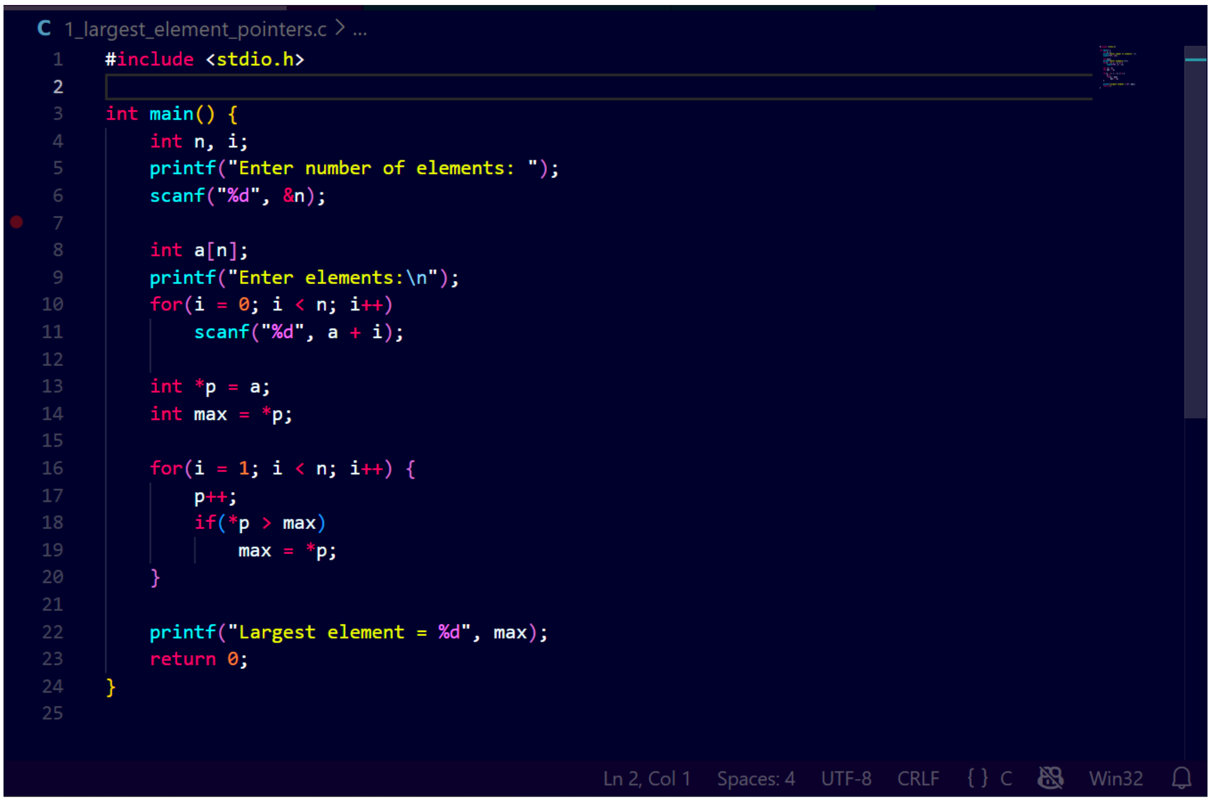
Aim

To write a C program to find the largest element in an integer array using pointers.

Algorithm

1. Read number of elements
2. Read array elements
3. Use a pointer to traverse the array
4. Compare elements and track the largest value
5. Display the largest element

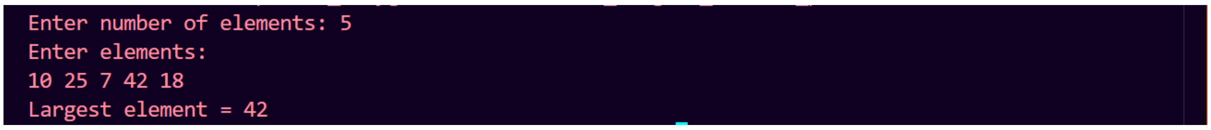
Program Code



C 1_largest_element_pointers.c > ...

```c
#include <stdio.h>

int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int a[n];
    printf("Enter elements:\n");
    for(i = 0; i < n; i++)
        scanf("%d", a + i);

    int *p = a;
    int max = *p;

    for(i = 1; i < n; i++) {
        p++;
        if(*p > max)
            max = *p;
    }

    printf("Largest element = %d", max);
    return 0;
}
```

Output



```
Enter number of elements: 5
Enter elements:
10 25 7 42 18
Largest element = 42
```

Program 2: Sum of Array Elements using Pointers

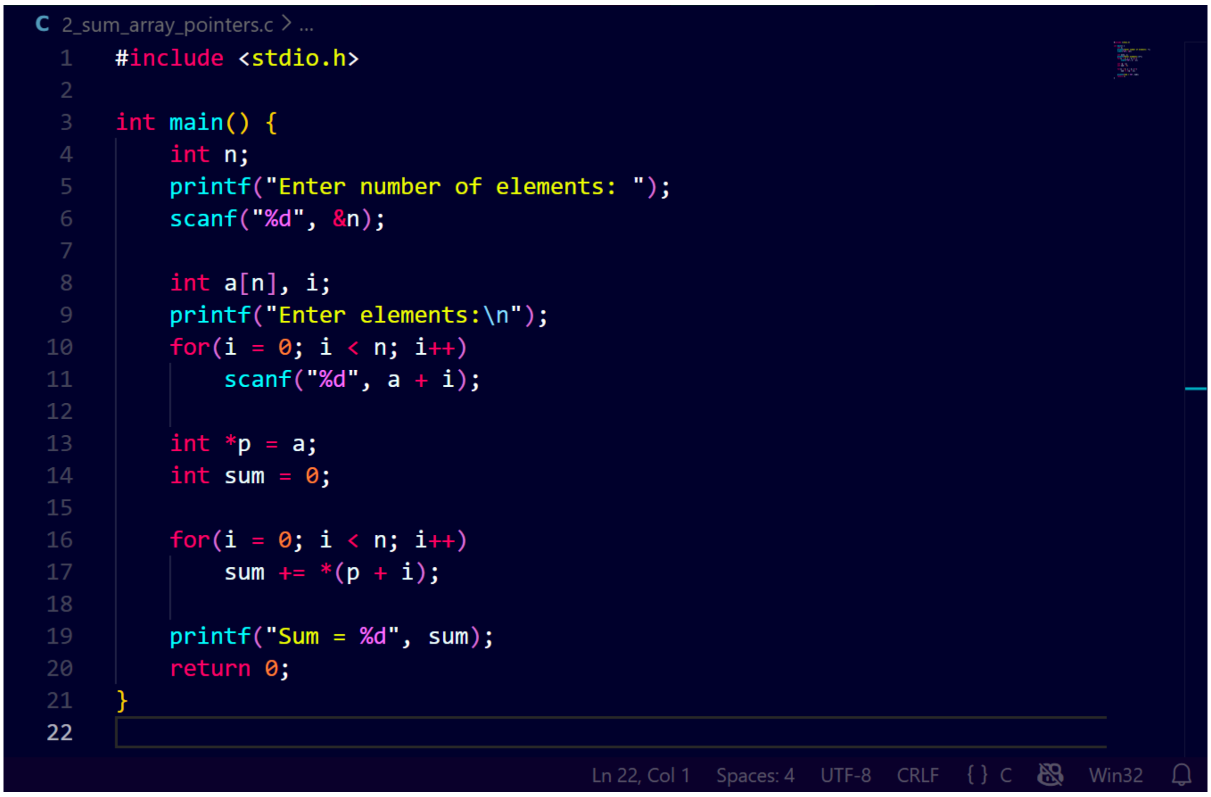
Aim

To write a C program to compute the sum of all elements in an array using pointers (without indexing).

Algorithm

1. Read number of elements
2. Read array elements
3. Use a pointer to access each element
4. Add values to sum
5. Display the sum

Program Code

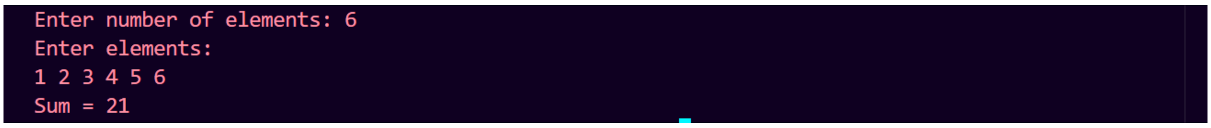


2_sum_array_pointers.c > ...

```c
#include <stdio.h>

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int a[n], i;
    printf("Enter elements:\n");
    for(i = 0; i < n; i++)
        scanf("%d", a + i);

    int *p = a;
    int sum = 0;

    for(i = 0; i < n; i++)
        sum += *(p + i);

    printf("Sum = %d", sum);
    return 0;
}

```

Ln 22, Col 1   Spaces: 4   UTF-8   CRLF   { } C   Win32

Output



```
Enter number of elements: 6
Enter elements:
1 2 3 4 5 6
Sum = 21
```

Program 3: Count Even and Odd Elements using Pointers

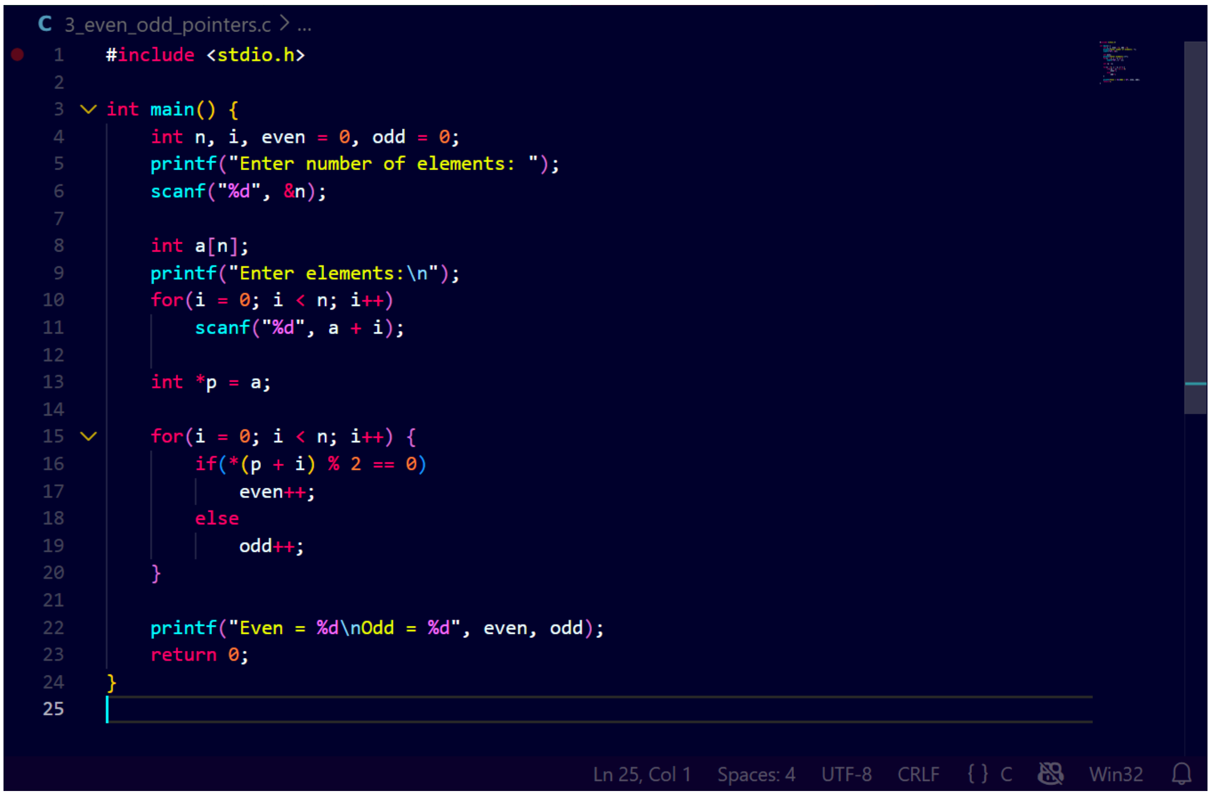
Aim

To write a C program to count how many elements in an array are even and odd using pointers.

Algorithm

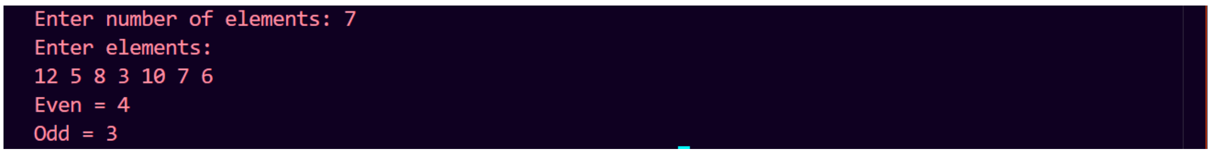
1. Read number of elements
2. Read array elements
3. Traverse array using pointer
4. Check each value for even or odd
5. Count and display results

Program Code



```c
#include <stdio.h>

int main() {
    int n, i, even = 0, odd = 0;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int a[n];
    printf("Enter elements:\n");
    for(i = 0; i < n; i++)
        scanf("%d", a + i);

    int *p = a;

    for(i = 0; i < n; i++) {
        if(*(p + i) % 2 == 0)
            even++;
        else
            odd++;
    }

    printf("Even = %d\nOdd = %d", even, odd);
    return 0;
}
```

Output



```
Enter number of elements: 7
Enter elements:
12 5 8 3 10 7 6
Even = 4
Odd = 3
```

Program 4: Dynamic Memory Allocation and Average (malloc and free)

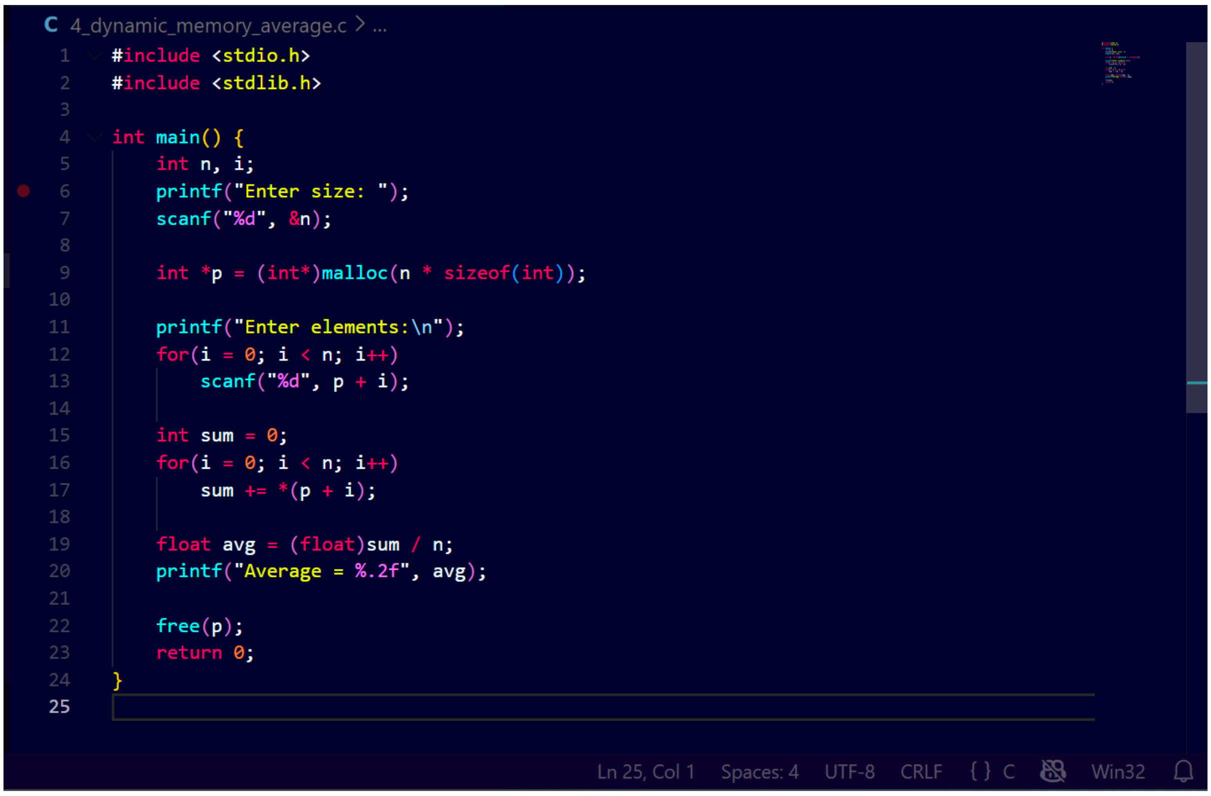
Aim

To write a C program to allocate memory dynamically for an array, read elements, and find the average using pointers.

Algorithm

1. Read required size
2. Allocate memory using malloc
3. Read values through pointer
4. Compute average
5. Free allocated memory

Program Code

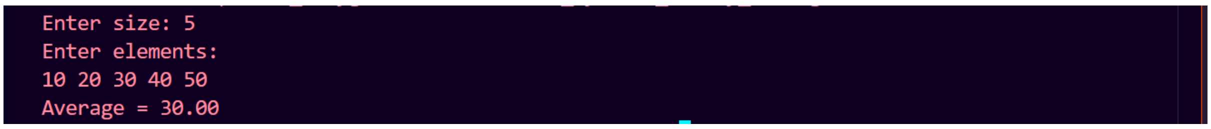


4_dynamic_memory_average.c > ...

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

int main() {
    int n, i;
    printf("Enter size: ");
    scanf("%d", &n);

    int *p = (int*)malloc(n * sizeof(int));

    printf("Enter elements:\n");
    for(i = 0; i < n; i++)
        scanf("%d", p + i);

    int sum = 0;
    for(i = 0; i < n; i++)
        sum += *(p + i);

    float avg = (float)sum / n;
    printf("Average = %.2f", avg);

    free(p);
    return 0;
}
```

Ln 25, Col 1   Spaces: 4   UTF-8   CRLF   { } C   Win32

Output



```
Enter size: 5
Enter elements:
10 20 30 40 50
Average = 30.00
```

Program 5: Linear Search using Pointer Traversal

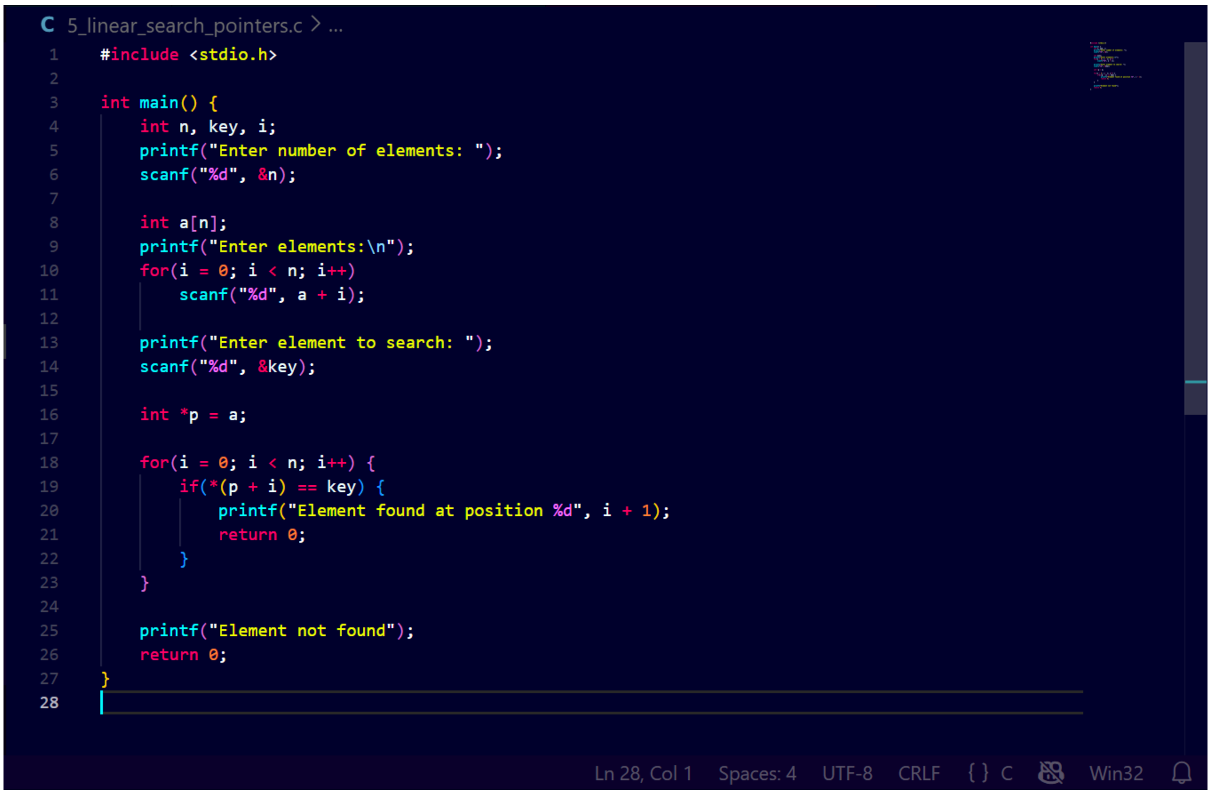
Aim

To write a C program to implement linear search using pointer traversal.

Algorithm

1. Read number of elements
2. Read array elements
3. Read key element
4. Traverse using pointer
5. Display position if found

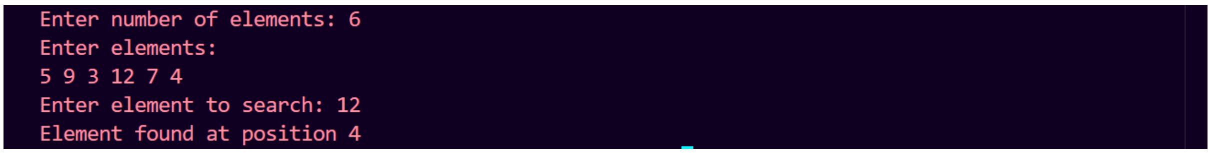
Program Code



C 5_linear_search_pointers.c > ...

```c
#include <stdio.h>

int main() {
    int n, key, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int a[n];
    printf("Enter elements:\n");
    for(i = 0; i < n; i++)
        scanf("%d", a + i);

    printf("Enter element to search: ");
    scanf("%d", &key);

    int *p = a;

    for(i = 0; i < n; i++) {
        if(*(p + i) == key) {
            printf("Element found at position %d", i + 1);
            return 0;
        }
    }

    printf("Element not found");
    return 0;
}

```

Output



```
Enter number of elements: 6
Enter elements:
5 9 3 12 7 4
Enter element to search: 12
Element found at position 4
```

Program 6: Binary Search using Pointers (No Indexing)

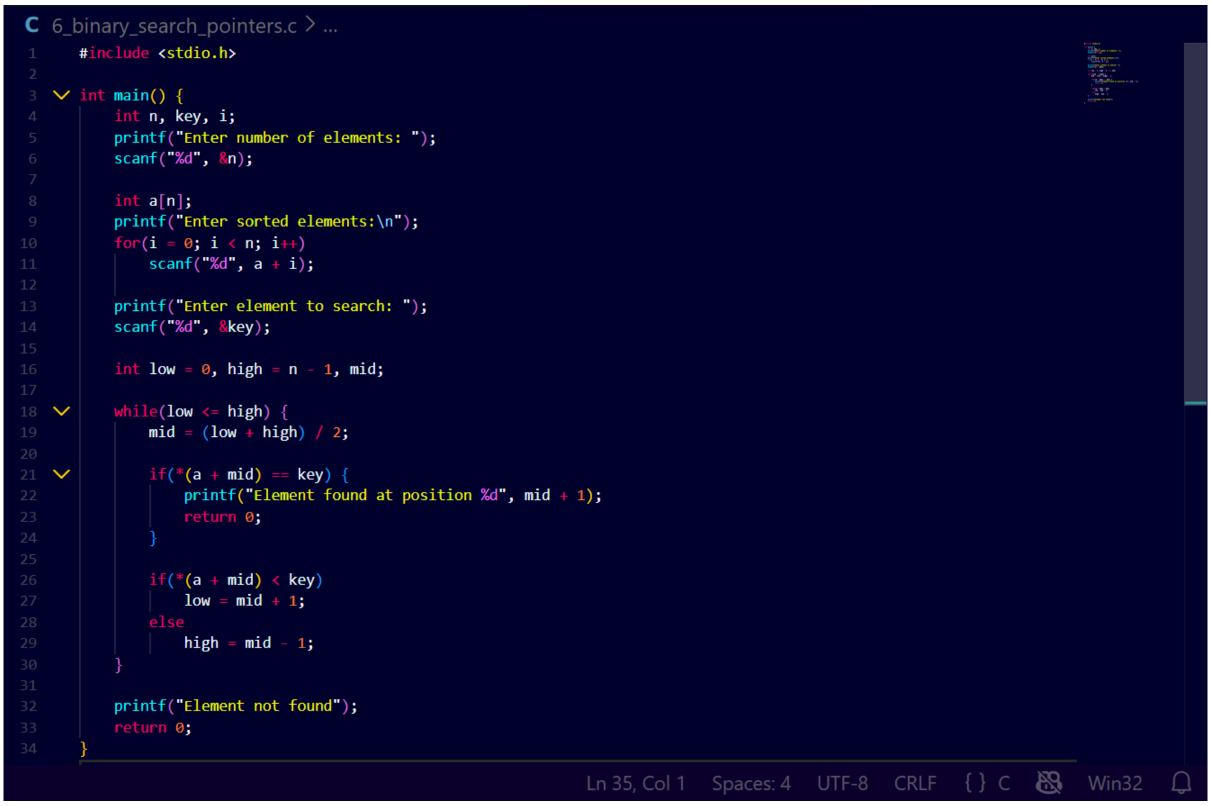
Aim

To write a C program to implement binary search using pointers without array indexing.

Algorithm

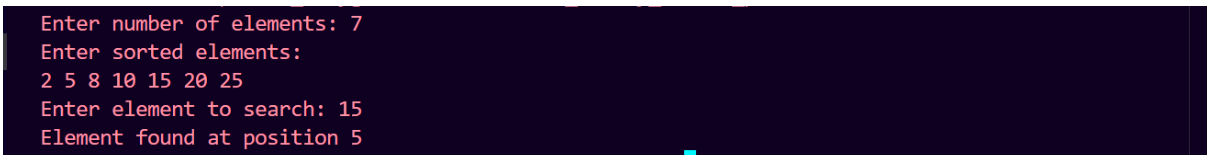
1. Read number of elements
2. Read sorted array elements
3. Use pointer arithmetic for mid, low, and high
4. Compare key with middle element
5. Display result

Program Code



```c
#include <stdio.h>

int main() {
    int n, key, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int a[n];
    printf("Enter sorted elements:\n");
    for(i = 0; i < n; i++)
        scanf("%d", a + i);

    printf("Enter element to search: ");
    scanf("%d", &key);

    int low = 0, high = n - 1, mid;

    while(low <= high) {
        mid = (low + high) / 2;

        if(*(a + mid) == key) {
            printf("Element found at position %d", mid + 1);
            return 0;
        }

        if(*(a + mid) < key)
            low = mid + 1;
        else
            high = mid - 1;
    }

    printf("Element not found");
    return 0;
}
```

Output



```
Enter number of elements: 7
Enter sorted elements:
2 5 8 10 15 20 25
Enter element to search: 15
Element found at position 5
```

Program 7: Insertion Sort using Pointer Manipulation

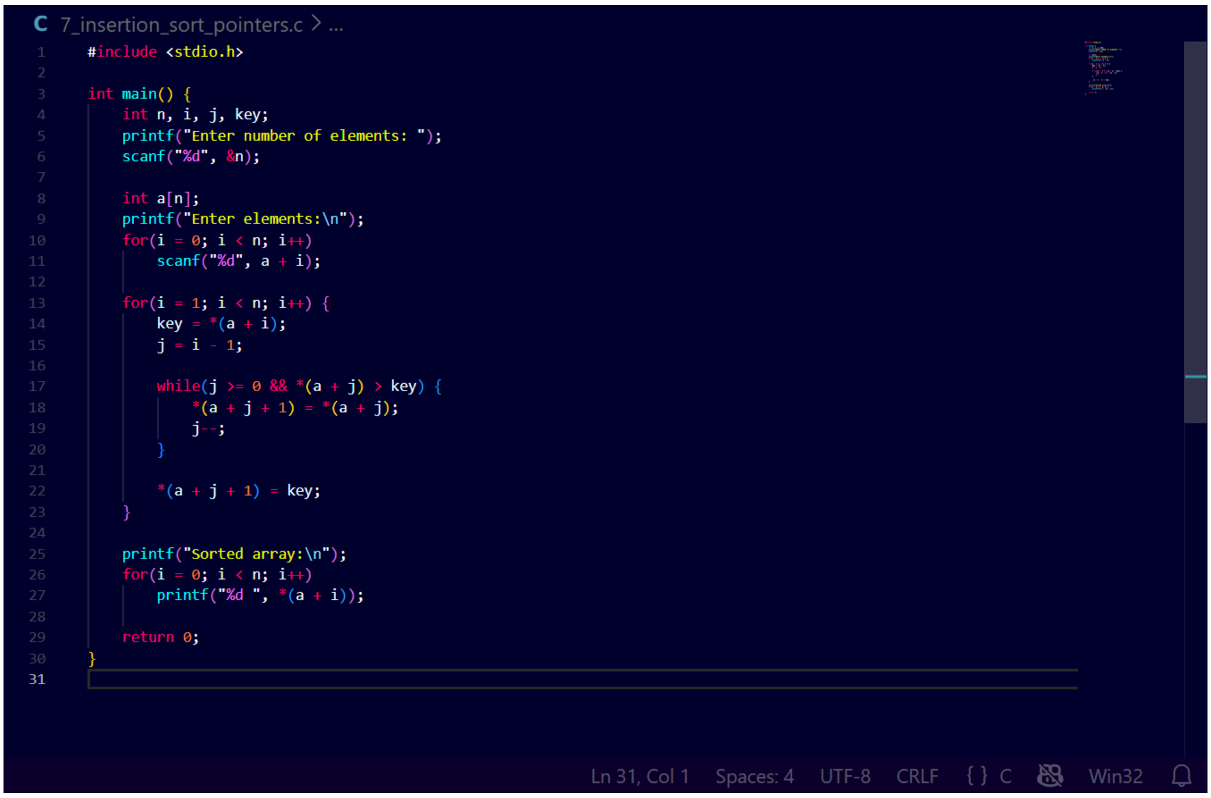
Aim

To write a C program to implement insertion sort using pointer manipulation instead of array indexing.

Algorithm

1. Read number of elements
2. Read array elements
3. Use pointer shifting for insertion
4. Arrange elements in ascending order
5. Display sorted array

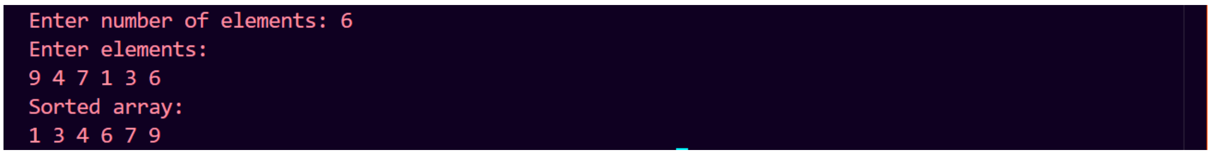
Program Code



C 7_insertion_sort_pointers.c > ...

```c
#include <stdio.h>

int main() {
    int n, i, j, key;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int a[n];
    printf("Enter elements:\n");
    for(i = 0; i < n; i++)
        scanf("%d", a + i);

    for(i = 1; i < n; i++) {
        key = *(a + i);
        j = i - 1;

        while(j >= 0 && *(a + j) > key) {
            *(a + j + 1) = *(a + j);
            j--;
        }

        *(a + j + 1) = key;
    }

    printf("Sorted array:\n");
    for(i = 0; i < n; i++)
        printf("%d ", *(a + i));

    return 0;
}
```

Output



```
Enter number of elements: 6
Enter elements:
9 4 7 1 3 6
Sorted array:
1 3 4 6 7 9
```

Program 8: Stack Implementation using Arrays and Pointers (PUSH, POP, PEEK)

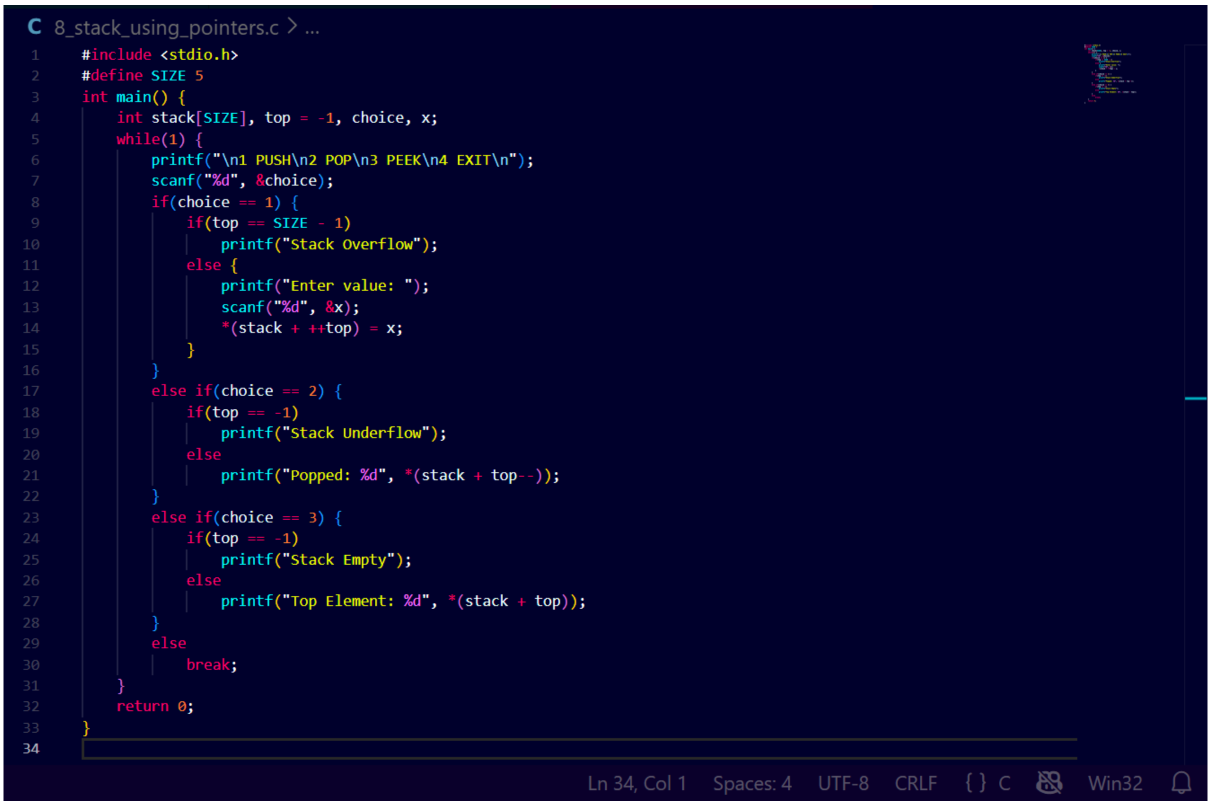
Aim

To write a C program to implement a stack using arrays and pointers with push, pop, and peek operations.

Algorithm

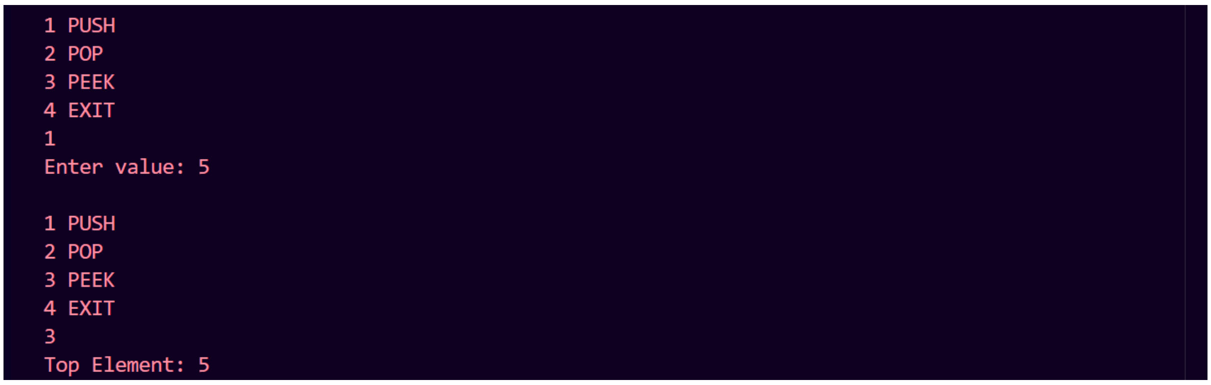
1. Initialize stack and top pointer
2. Implement PUSH operation
3. Implement POP operation
4. Implement PEEK operation
5. Display results

Program Code

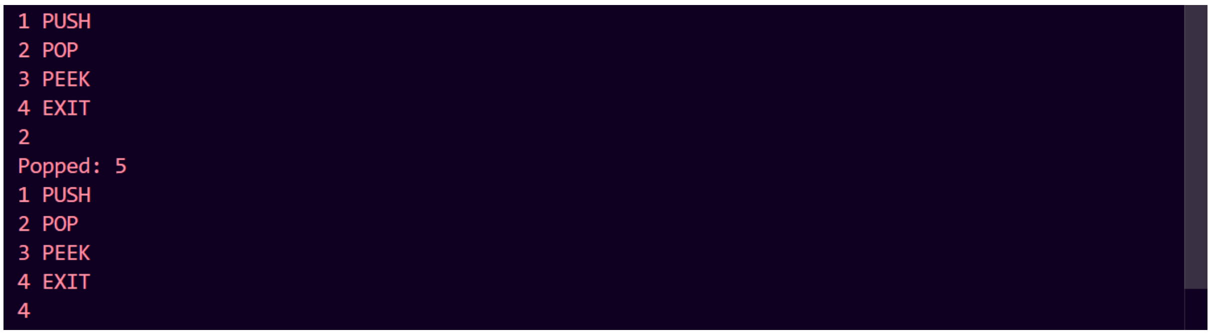


```c
#include <stdio.h>
#define SIZE 5
int main() {
    int stack[SIZE], top = -1, choice, x;
    while(1) {
        printf("\n1 PUSH\n2 POP\n3 PEEK\n4 EXIT\n");
        scanf("%d", &choice);
        if(choice == 1) {
            if(top == SIZE - 1)
                printf("Stack Overflow");
            else {
                printf("Enter value: ");
                scanf("%d", &x);
                *(stack + ++top) = x;
            }
        }
        else if(choice == 2) {
            if(top == -1)
                printf("Stack Underflow");
            else
                printf("Popped: %d", *(stack + top--));
        }
        else if(choice == 3) {
            if(top == -1)
                printf("Stack Empty");
            else
                printf("Top Element: %d", *(stack + top));
        }
        else
            break;
    }
    return 0;
}
```

Output



```
1 PUSH
2 POP
3 PEEK
4 EXIT
1
Enter value: 5

1 PUSH
2 POP
3 PEEK
4 EXIT
3
Top Element: 5
```



```
1 PUSH
2 POP
3 PEEK
4 EXIT
2
Popped: 5
1 PUSH
2 POP
3 PEEK
4 EXIT
4
```

Result

All pointer-based programs and stack operations for Experiment 3 were successfully implemented, executed, and verified.