Name: Om Singh Reg. No. - 2341019519

Department of Computer Science and Engineering
Institute of Technical Education & Research, SOA, Deemed to be University

MINOR ASSIGNMENT-03

Practical Programming with C (CSE 3544)

Publish on: 23-10-2025 **Submission on:** 27-10-2025
Course Outcome: CO_2 **Program Outcome:** PO_3 **Learning Level:** L_4

Problem Statement:

Experiment with arrays for storing and processing collections of values of the same types in different applications.

Assignment Objectives:

To learn how to declare and use arrays for storing collections of values of the same type as well as to learn how to process the elements of an array.

1. We initialize a 25-element array with the prime numbers less than 100.

```
int prime_lt_100[] = {2, 3, 5, 7, 11, 13, 17,
    19, 23, 29, 31, 37,41, 43, 47, 53, 59,
   61, 67, 71, 73, 79, 83, 89, 97};
```

Determine the array elements given in the following expressions;

```
(a) prime_lt_100[24];
(b) int i=10; prime_lt_100[i+4];
(c) prime_lt_100[prime_lt_100[2] +
                        prime_lt_100[0]];
(d) prime_lt_100[6]=prime_lt_100[6] +
                        prime_lt_100[16];
```

Output

a) 97
b) 47
c) 19
d) 76

2. Design a function with prototype;
`void sumarr(int a[], int b[], int r[], int size);` that takes 4 parameters, two int arrays as input arguments, 1 array as output arguments and their effective size respectively to produce a resultant array containing the sums of corresponding array elements **a** and **b**. For example, for the three-element input arrays 5 -1 7 and 2 4 -2 , the result would be an array containing 7 3 5 .

Space for Program ▼

```
void sumarr(int a[], int b[], int r[],
      int size){
 for(int i=0; i<size; i++)
   r[i] = a[i]+ b[i];
}
```

Output ▼

Output: 7 3 5

3. The **bubble sort** is another technique for sorting an array. A bubble sort compares adjacent array elements and exchanges their values if they are out of order. In this way, the smaller values "bubble" to the top of the array (toward element 0), while the larger values sink to the bottom of the array. After the first pass of a bubble sort, the last array element is in the correct position; after the second pass the last two elements are correct, and so on. Thus, after each pass, the unsorted portion of the array contains one less element. Write and test a function that implements this sorting method.

Space for Program ▼

```
void bubbleSort(int arr[], int 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 temp = arr[j];
     arr[j] = arr[j+1];
     arr[j+1] = temp;
  }
  }
}
```

Output ▼

for array = {5,2,9,1,5,6}

Output: 1 2 5 5 6 9

4. You have two independent sorted arrays of size m, and n respectively, where $m, n > 0$. You are required to merge the two arrays such that the merged array will be in sorted form and will contain exactly $m + n$ number of elements. You are not allowed to use any kind of sorting algorithm. Design your program to meet the above given requirement.

Example 1 :

First array: | 12 | 20 | 24 | 32 |

Second array: | 7 | 8 | 65 | 105 |

The merged sorted array: | 7 | 8 | 12 | 20 | 24 | 32 | 65 | 105 |

Example 2 :

First array: | 12 | 20 | 24 |

Second array: | 7 | 8 | 65 | 105 |

The merged sorted array: | 7 | 8 | 12 | 20 | 24 | 65 | 105 |

Example 3 :

First array : | 12 | 20 | 24 | 100 | 120 | 130 |

Second array : | 17 | 28 | 105 | 110 |

The merged sorted array: | 12 | 17 | 20 | 24 | 100 | 105 | 110 | 120 | 130 |

NOTE :

Assume the elements of the array are non-negative integers. The elements can be read from the keyboard or can be generated randomly.

Space for Program **Output ▼**

```
#include <stdio.h>
int main(){
int a[] = {12, 20, 24, 32};
int b[] = {7, 8, 65, 105};
int m=4, n=4;
int c[m+n];
int i=0, j=0, k=0;
```

Space for Program

```
while(i<m && j<n){
  if(a[i]<b[j])
    c[k++] = a[i++];
  else
    c[k++] = b[j++];
}

while (i<m)
  c[k++] = a[i++];
while(j<n)
  c[k++] = b[j++];

printf("Merged sorted array: ");
for(int i=0; i<m+n; i++)
  printf("%d ", c[i]);
return 0;
}
```

Output

```
Merged sorted array:
7 8 12 20 24 32 65 105
```

5. The *binary search* algorithm that follows may be used to search an array when the elements are in order. The algorithm for binary search given as;
   1. Let **bottom** be the subscript of the initial array element.
   2. Let **top** be the subscript of the last array element.
   3. Let **found** be false.
   4. Repeat as long as **bottom** isn't greater than **top** and the target has not been found
      5. Let **middle** be the subscript of the element halfway between **bottom** and **top**.
      6. if the element at **middle** is the target
         7. Set **found** to true and **index** to **middle**.

      else if the element at **middle** is larger than the target
         8. Let **top** be **middle - 1**.

      else
         9. Let **bottom** be **middle + 1**.

Write and test a function **binary_srch** that implements this algorithm for an array of integers. When there is a large number of array elements, which function do you think is faster: **binary_srch** or the linear search algorithm.

Space for Program | **Output ▼**

```
#include <stdio.h>
int binary_srch(int arr[], int size,
 int target){
int bottom=0, top = size-1, middle;
while( bottom<=top){
  middle = (bottom+top)/2;
  if(arr[middle]==target)
     return middle;
  else if(arr[middle]>target)
     top = middle-1;
  else
     bottom = middle+1;
}
return -1;
}
```

Space for Program

```
int main(){
int arr[] = {5,8,12,20,33,40,42};
int size = sizeof(arr)/sizeof(arr[0]);
int target = 33;
int index = binary_srch(arr,size,target);
if(index != -1)
 printf("Element found at index %d\n",index);
else
 printf("Element not found\n");
return 0;
}
```

Output ▼

Output: Element found at index 4

6. Design a program to find the difference between two sets or arrays. *The difference between two sets or arrays*: **All the elements of the first array that don't appear in the second array**. If array **p** has the elements { 1, 2, 3, 4} and array **q** has the elements {2, 4, 5, 6 }, then the difference between the two arrays, **p−q** will be {1, 3 }.

Space for Program

```
#include <stdio.h>
int main(){
 int p[]={1,2,3,4};
 int q[]={2,4,5,6};
 int n1=4, n2=4;
 printf("Difference (p-q): ");
 for(int i=0; i<n1; i++){
    int found=0;
    for(int j=0; j<n2; j++){
      if(p[i]==q[j]){
         found=1;
         break;
      }
    }
    if(!found)
       printf("%d ",p[i]);
 }
 return 0;
}
```

Output ▼

```
Difference (p-q): 1 3
```

7. Write a program to copy the distinct elements of an int type array to another int type array. For example, if the input array is 4 7 7 3 2 5 5 then the output array will be 4 7 3 2 5.

Space for Program

```
#include <stdio.h>
int main(){
int arr[]={4,7,7,3,2,5,5};
int n=7, b[7], k=0;
for(int i=0; i<n; i++){
  int found=0;
  for(int j=0; j<k; j++){
    if(arr[j]==b[j]){
      found=1;
      break;
    }
  }
  if(!found)
    b[k++]=arr[i];
}
printf("Distinct array: ");
for(int i=0; i<k; i++)
  printf("%d ", b[i]);
return 0;
}
```

Output

```
Distinct array: 4 7 3 2 5
```

8. Construct a program to find the occurrence of the first repetitive character in a string. For example, let the string **racecar**, the program should give the output as **The first repititive character in the string `racecar` is c.**

Space for Program

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

int main(){
 char str[50];
 printf("Enter a string: ");
 scanf("%s", str);

for(int i=0; i<strlen(str); i++){
  for(int j=i+1; j<strlen(str); j++){
   if(str[i]==str[j]){
    printf("The first repetitive character in the string %s
       is %c\n", str, str[i]);
    return 0;
   }
  }
}

printf("No repetitive characters found.\n");
return 0;
}
```

Output ▼

```
Enter a string: racecar
The first repetitive
character in the string
racecar is c.
```

9. Design a program to display the count of each character in a string. For example: **input** string: **bintu**, **output:** The count of each character in the string **bintu** is b-1, i-1, n-1, t-i, u-1.

Space for Program

```
#include <stdio.h>
#include <string.h>
int main(){
  char str[100];
  int freq[256] = {0};
  printf("Enter a string: ");
  scanf("%s", str);
  for(int i=0; i<strlen(str); i++)
    freq[(unsigned char)str[i]]++;
  printf("The count of each character
in the string %s is:\n", str);
  for(int i=0; i<256; i++){
    if(freq[i]>0)
      printf("%c - %d\n", i, freq[i]);
  }
  return 0;
}
```

Output ▼

```
Enter a string: bintu
b - 1
i - 1
n - 1
t - 1
u - 1
```

10. The **Selection sort** is a **comparison-based** sorting algorithm that sorts a collection by repeatedly finding the **minimum** (or *maximum*) element and placing it in its correct position in the list. It is generally preferred when you have to manually implement the sorting algorithm for a small amount of dataset. Create a C program to sort an array of **n** elements using selection sort.

Space for Program

```
#include <stdio.h>
void selectionSort(int arr[], int n){
  for(int i=0; i<n-1; i++){
    int min = i;
    for(int j = i+1; j<n; j++){
      if(arr[j] < arr[min])
        min = j;
    }
int temp = arr[min];
 arr[min] = arr[i];
 arr[i] = temp;
 }
}
int main(){
 int arr[] = {64, 25, 12, 22, 11};
 int n = 5;
 selectionSort(arr, n);
 printf("Sorted array: ");
 for(int i=0; i<n; i++)
    printf("%d ", arr[i]);
 return 0;
}
```

Output ▼

```
Sorted array: 11 12 22 25 64
```

10. The **Insertion** sort is one of the simple and **comparison-based** sorting algorithms. The basic idea behind the algorithm is to virtually divide the given list into two parts: a sorted part and an unsorted part, then pick an element from the unsorted part and insert it in its place in the sorted part. It does this till all the elements are placed in the sorted part. Develop a C program to sort an array of **n** elements using insertion sort.

Space for Program

```
#include <stdio.h>
void insertionSort(int arr[], int n){
for(int i=1; i<n; i++){
  int key = arr[i];
  int j = i-1;
while (j>=0 && arr[j] > key){
   arr[j+1] = arr[j];
    j--;
  }
  arr[j+1] = key;
 }
}
int main(){
int arr[] = {9,5,1,4,3};
int n = 5;
insertionSort(arr, n);
printf("Sorted array: ");
for(int i=0; i<n; i++)
  printf("%d", arr[i]);
return 0;
}
```

Output ▼

```
Sorted array: 1 3 4 5 9
```