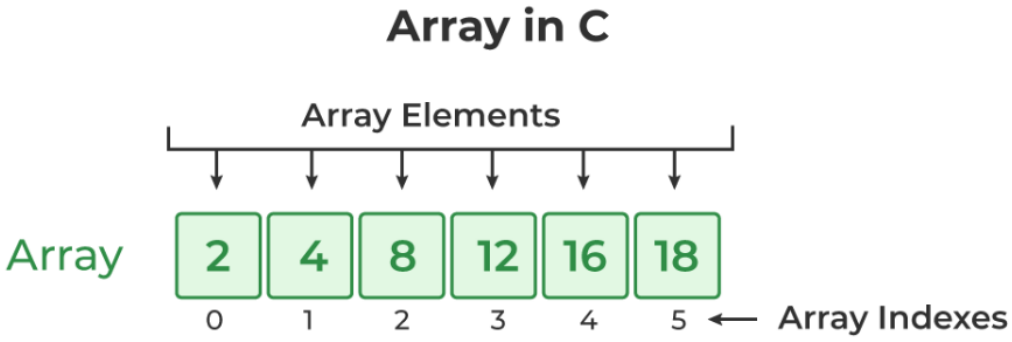
UNIT – III

Arrays: Arrays indexing, Accessing programs with array of integers, two dimensional arrays,
String: Introduction to Strings, string handling functions.
Sorting techniques: bubble sort, selection sort.
Searching Techniques: linear , Binary search.

Array:

An array in C is a fixed-size collection of similar data items stored in contiguous memory locations. It can be used to store the collection of primitive data types such as int, char, float, etc., and also derived and user-defined data types such as pointers, structures, etc.

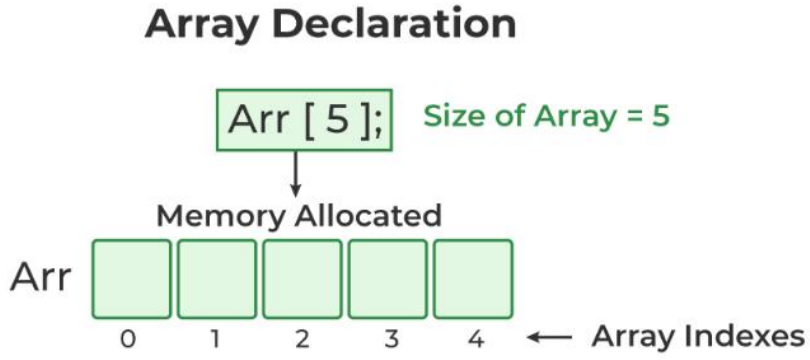


Array Declaration:

In C, we must declare the array like any other variable before using it. We can declare an array by specifying its name, the type of its elements, and the size of its dimensions. When we declare an array in C, the compiler allocates the memory block of the specified size to the array name.

Syntax:

```
data_type array_name [size];
                or
data_type array_name [size1] [size2]...[sizeN];
where N is the number of dimensions.
```



Array Initialization:

Initialization in C is the process to assign some initial value to the variable. When the array is declared or allocated memory, the elements of the array contain some garbage value. So, we need to initialize the array to some meaningful value. There are two types of initializations of an array.

1) Compile time initialization
2) Runtime initialization

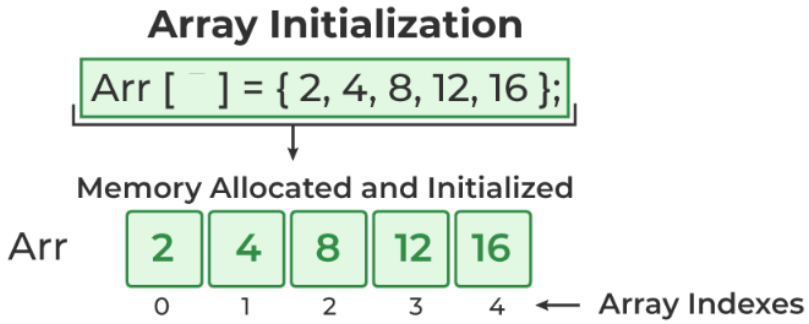
1. Compile time initialization:

In this method, we initialize the array along with its declaration. We use an initializer list to initialize multiple elements of the array. An initializer list is the list of values enclosed within braces **{ }** separated by a comma.

Syntax:

```
data_type array_name [] = {value1, value2, ... valueN};
```

Note: The size of the array in these cases is equal to the number of elements present in the initializer list as the compiler can automatically deduce the size of the array.



2. Runtime initialization:

We initialize the array after the declaration by assigning the initial value to each element individually. We can use for loop, while loop, or do-while loop to assign the value to each element of the array.

```
for (int i = 0; i < N; i++) {
        array_name[i] = value_i;
}
```

Example:

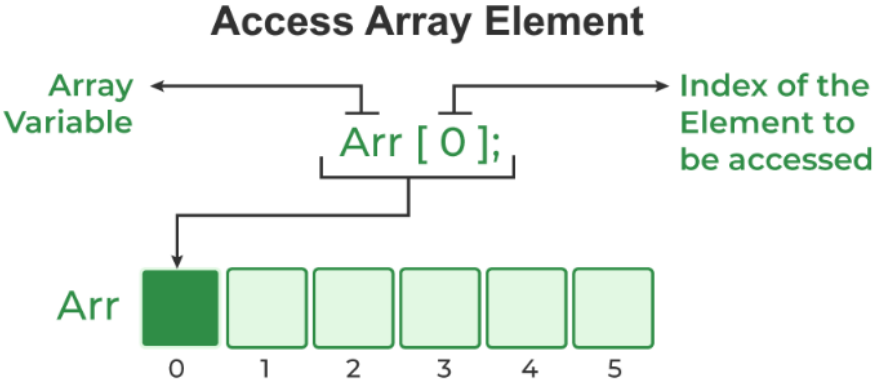
```
int arr[5];
for(int i=0;i<5;i++)
{
        scanf("%d",&arr[i]);
}
```

Accessing of Array Elements:

We can access any element of an array in C using the array subscript operator **[ ]** and the index value ***i*** of the element.

Syntax: array_name [index];

One thing to note is that the indexing in the array always starts with 0, i.e., the first element is **at index 0** and the last element is **at N – 1** where **N** is the number of elements in the array.

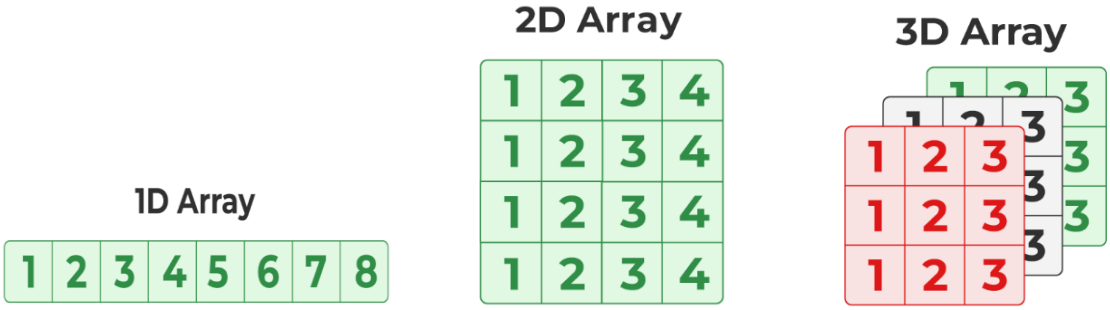


```
// C Program to demonstrate the use of array
#include <stdio.h>
int main()
{
        // array declaration and initialization
        int arr[] = { 10, 20, 30, 40, 50 };

        // modifying element at index 2
        arr[2] = 100;

        // traversing array using for loop
        printf("Elements in Array: ");
        for (int i = 0; i < 5; i++)
        {
                printf("%d ", arr[i]);
        }
  return 0;
}
```

Types of Arrays in C

There are two types of arrays based on the number of dimensions it has. They are as follows:

1. One Dimensional Arrays (1D Array)
2. Multidimensional Arrays



One Dimensional Arrays:

In C, a 1D array is a collection of elements of the same data type, stored in contiguous memory locations. Each element can be accessed using an index, starting from 0 up to n-1, where n is the array's length. Arrays are useful for storing lists of data, like numbers or characters, allowing efficient access and manipulation using loops and indices.

Practice Programs:

1) Program to read and display the elements of an array.

Source Code:

```
#include <stdio.h>
int main()
{
  int n, i;

  // Ask the user for the number of elements
  printf("Enter the number of elements: ");
  scanf("%d", &n);

  // Declare an array of size n
  int arr[n];
```

```
    // Read elements into the array
    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++)
   {
       scanf("%d", &arr[i]);
    }

    // Display the elements of the array
    printf("Array elements are:\n");
    for(i = 0; i < n; i++)
    {
       printf("%d ", arr[i]);
    }
    return 0;
}
```

2) Program to find the sum of all the elements of an array.

Source Code:

```
#include <stdio.h>
int main()
{
    int n, i, sum = 0;

    // Ask the user for the number of elements
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Declare an array of size n
    int arr[n];

    // Read elements into the array
    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++)
    {
       scanf("%d", &arr[i]);
    }

    // Calculate the sum of all elements
    for(i = 0; i < n; i++)
    {
       sum += arr[i];
    }

    // Display the sum
    printf("The sum of all elements is: %d\n", sum);
    return 0;
}
```

3) Program to find the given element is available in the array or not.

Source Code:

```
#include <stdio.h>
int main()
{
   int n, i, element, found = 0;
   // Ask the user for the number of elements
   printf("Enter the number of elements: ");
   scanf("%d", &n);

   // Declare an array of size n
   int arr[n];

   // Read elements into the array
   printf("Enter %d elements:\n", n);
   for(i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   // Ask the user for the element to search
   printf("Enter the element to search: ");
   scanf("%d", &element);

   // Check if the element is in the array
   for(i = 0; i < n; i++)
   {
      if(arr[i] == element)
     {
         found = 1;
         break;
      }
   }

   // Display result
   if(found == 1)
   {
          printf("Element %d is present in the array.\n", element);
   }
    else
   {
          printf("Element %d is not present in the array.\n", element);
   }
   return 0;
}
```

4) Program to find the biggest and smallest element in an array.

Source Code:

```
#include <stdio.h>
int main() {
   int n, i;
   // Ask the user for the number of elements
   printf("Enter the number of elements: ");
   scanf("%d", &n);
```

```
    // Declare an array of size n
    int arr[n];

    // Read elements into the array
    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++)
    {
       scanf("%d", &arr[i]);
    }

    // Initialize max and min with the first element
    int max = arr[0];
    int min = arr[0];

    // Find the maximum and minimum elements
    for(i = 1; i < n; i++)
    {
          if(arr[i] > max) {
                max = arr[i];
          }

          if(arr[i] < min) {
                min = arr[i];
          }
    }

    // Display the results
    printf("The largest element is: %d\n", max);
    printf("The smallest element is: %d\n", min);

    return 0;
}
```

5) Program to find the frequency of each element in the given array.

Source Code:

```
#include <stdio.h>
int main()
{
    int n, i, j;

    // Ask the user for the number of elements
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Declare an array of size n
    int arr[n];
    int freq[n];

    // Read elements into the array
    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++) {
       scanf("%d", &arr[i]);
```

```
    freq[i] = -1;                    // Initialize frequency array with -1
  }

  // Calculate frequency of each element
  for(i = 0; i < n; i++) {
    int count = 1;
    for(j = i + 1; j < n; j++) {
      if(arr[i] == arr[j]) {
        count++;
        freq[j] = 0;                 // Mark duplicate elements as 0
      }
    }
    if(freq[i] != 0) {
      freq[i] = count;
    }
  }

  // Display the frequency of each unique element
  printf("Element | Frequency\n");
  for(i = 0; i < n; i++)
  {
    if(freq[i] != 0) {
      printf("   %d    |    %d\n", arr[i], freq[i]);
    }
  }

  return 0;
}
```

6) Program to find the unique element in the given array.
7) Program to find the duplicate elements in the given array.

2 Dimensional Arrays:

In C, a 2-dimensional (2D) array is essentially an array of arrays, used to store data in a tabular form (rows and columns). It is declared by specifying two indices: the number of rows and columns.

For example, int arr[3][4]; declares a 2D array with 3 rows and 4 columns. Each element can be accessed using two indices, **arr[i][j],** where i is the row index and j is the column index. 2D arrays are widely used for matrices, grids, and other data structures where data is organized in rows and columns.

Practice Programs:

1. Program to read and display a matrix.

Source Code:

```
#include <stdio.h>
int main() {
  int rows, columns, i, j;
  // Ask the user for the number of rows and columns
  printf("Enter the number of rows: ");
  scanf("%d", &rows);
  printf("Enter the number of columns: ");
  scanf("%d", &columns);
```

```
    // Declare a 2D array with the specified dimensions
    int matrix[rows][columns];

    // Read elements into the matrix
    printf("Enter the elements of the matrix:\n");
    for(i = 0; i < rows; i++)
    {
        for(j = 0; j < columns; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    // Display the matrix
    printf("The matrix is:\n");
    for(i = 0; i < rows; i++)
    {
        for(j = 0; j < columns; j++)
       {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Output:

```
Enter the number of rows: 2
Enter the number of columns: 3
Enter the elements of the matrix:
1 2 3 4 5 6
```

The matrix is:

```
1   2   3
4   5   6
```

2. Program to find the sum of all the elements of a matrix.

Source Code:

```
#include <stdio.h>
int main()
{
    int rows, columns, i, j, sum = 0;

    // Ask the user for the number of rows and columns
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &columns);

    // Declare a 2D array with the specified dimensions
    int matrix[rows][columns];

    // Read elements into the matrix
    printf("Enter the elements of the matrix:\n");
```

```
    for(i = 0; i < rows; i++)
    {
        for(j = 0; j < columns; j++)
        {
            scanf("%d", &matrix[i][j]);
        }
    }

    // Calculate the sum of all elements in the matrix
    for(i = 0; i < rows; i++)
    {
        for(j = 0; j < columns; j++)
        {
            sum += matrix[i][j];
        }
    }

    // Display the sum
    printf("The sum of all elements in the matrix is: %d\n", sum);

    return 0;
}
```

3. Program to find maximum element in a matrix.

4. Program to find the addition of two matrices.

Source Code:

```
#include <stdio.h>
int main()
{
    int rows, columns, i, j;

    // Ask the user for the number of rows and columns
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &columns);

    // Declare two 2D arrays for the matrices and one for the result
    int matrix1[rows][columns], matrix2[rows][columns], sum[rows][columns];

    // Read elements into the first matrix
    printf("Enter elements of the first matrix:\n");
    for(i = 0; i < rows; i++) {
        for(j = 0; j < columns; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }

    // Read elements into the second matrix
    printf("Enter elements of the second matrix:\n");
    for(i = 0; i < rows; i++) {
```

```
        for(j = 0; j < columns; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }

    // Calculate the sum of the two matrices
    for(i = 0; i < rows; i++) {
        for(j = 0; j < columns; j++) {
            sum[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }

    // Display the resulting matrix
    printf("The sum of the two matrices is:\n");
    for(i = 0; i < rows; i++) {
        for(j = 0; j < columns; j++) {
            printf("%d ", sum[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Output:

```
Enter the number of rows: 2
Enter the number of columns: 3
Enter elements of the first matrix:
1   2   3
4   5   6
Enter elements of the second matrix:
7   8   9
10 11  12
```

The sum of the two matrices is:

```
8       10      12
14      16      18
```

5. Program to find the product of two matrices.

Source Code:

```
#include <stdio.h>
int main()
{
    int rows1, cols1, rows2, cols2, i, j, k;

    // Ask the user for the dimensions of the first matrix
    printf("Enter the number of rows and columns for the first matrix: ");
    scanf("%d %d", &rows1, &cols1);

    // Ask the user for the dimensions of the second matrix
    printf("Enter the number of rows and columns for the second matrix: ");
    scanf("%d %d", &rows2, &cols2);
```

```
    // Check if the matrices can be multiplied
    if (cols1 != rows2)
    {
       printf("Matrix multiplication is not possible.\n");
       return 0;
    }

    // Declare the matrices
    int matrix1[rows1][cols1], matrix2[rows2][cols2], product[rows1][cols2];

    // Read elements into the first matrix
    printf("Enter elements of the first matrix:\n");
    for (i = 0; i < rows1; i++) {
       for (j = 0; j < cols1; j++) {
          scanf("%d", &matrix1[i][j]);
       }
    }

    // Read elements into the second matrix
    printf("Enter elements of the second matrix:\n");
    for (i = 0; i < rows2; i++) {
       for (j = 0; j < cols2; j++) {
          scanf("%d", &matrix2[i][j]);
       }
    }

    // Calculate the product of the two matrices
    for (i = 0; i < rows1; i++)
   {
       for (j = 0; j < cols2; j++)
      {
           product[i][j]=0;
          for (k = 0; k < cols1; k++)
           {
                       product[i][j] += matrix1[i][k] * matrix2[k][j];
           }
       }
    }

    // Display the resulting product matrix
    printf("The product of the two matrices is:\n");
    for (i = 0; i < rows1; i++) {
       for (j = 0; j < cols2; j++) {
          printf("%d ", product[i][j]);
       }
       printf("\n");
    }

    return 0;
 }
```

6. Program to find the transpose of a matrix.

Source Code:

```
#include <stdio.h>
int main()
{
   int rows, columns, i, j;

   // Ask the user for the number of rows and columns
   printf("Enter the number of rows: ");
   scanf("%d", &rows);
   printf("Enter the number of columns: ");
   scanf("%d", &columns);

   // Declare a 2D array for the matrix and another for the transpose
   int matrix[rows][columns], transpose[columns][rows];

   // Read elements into the matrix
   printf("Enter elements of the matrix:\n");
   for(i = 0; i < rows; i++) {
      for(j = 0; j < columns; j++) {
         scanf("%d", &matrix[i][j]);
      }
   }

   // Calculate the transpose of the matrix
   for(i = 0; i < rows; i++) {
      for(j = 0; j < columns; j++) {
         transpose[j][i] = matrix[i][j];
      }
   }

   // Display the transposed matrix
   printf("The transpose of the matrix is:\n");
   for(i = 0; i < columns; i++) {
      for(j = 0; j < rows; j++) {
         printf("%d ", transpose[i][j]);
      }
      printf("\n");
   }

   return 0;
}
```

String Handling

String:

The string can be defined as the one-dimensional array of characters terminated by a null ('\0'). The character array or the string is used to manipulate text such as words or sentences.

Each character in the array occupies one byte of memory, and the last character must always be '\0'.

The termination character ('\0') is important in a string since it is the only way to identify where the string ends.

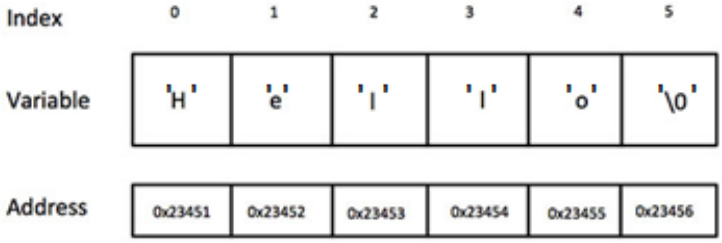
Declaration and Initialization:

As string is a character array, it is declared and initialized as 1-D array.

Syntax: char name[20];

Compile time initialization:

char msg[20]={'h','e','l','l','o','\0'}; or char msg[]={'h','e','l','l','o','\0'};



Ex:

```
#include <stdio.h>
int main()
{
        char greeting[5] = {'H', 'e', 'l', 'l', 'o', '\0'};
        printf("Greeting message: %s\n", greeting );
        return 0;
}
```

We can also define the string by the **string literal** in C language. For example:

```
char ch[]="hello";
```

Runtime initialization:

1. **To read a word**

scanf("%s",str);

Ex:

```
#include <stdio.h>
int main()
{
        char str[20];
        scanf("%s",str);
        printf("message: %s\n", str );
        return 0;
}
```

Input: Aditya Engineering College **Output:** message: Aditya

2. **To read a line of text**

scanf("%[^\n]s",str); or **gets(str);**

Ex:

```
#include <stdio.h>
int main()
{
        char str[20];
        scanf("%[^\n]s",str);
        printf("message: %s\n", str );
        return 0;
}
```

Input: Aditya Engineering College **Output:** message: Aditya Engineering College

Ex:

```
#include <stdio.h>
int main()
{
        char str[20];
        gets(str);              // it also reads a line of text
        printf("message: %s\n", str );
        return 0;
}
```

Input: Aditya Engineering College **Output:** message: Aditya Engineering College

3. To read multi line of text

```
scanf("%[a-zA-Z0-9 @\n]s",str);
```

Ex:

```
#include <stdio.h>
int main()
{
        char str[20];
        scanf("%[a-zA-Z0-9 @\n]s",str);
        printf("%s\n", str );
        return 0;
}
```

Input: Aditya
Engineering
College
#[reading of input ends here].

Output: Aditya
Engineering
College

gets(),puts(): These are the functions available in stdio.h header file to read and display strings.

gets():

The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array.

Syntax: **char**[] gets(**char**[]);

puts():

The puts() function is very similar to printf() function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. The puts() function returns an integer value representing the number of characters being printed on the console.

Syntax: **int** puts(**char**[]);

```
#include<stdio.h>
int main(){
        char name[50];
        printf("Enter your name: ");
        gets(name);             //reads string from user
        printf("Your name is: ");
        puts(name);             //displays string
        return 0;
 }
```

Input: Madava Rao **Ouput:** Madava Rao

String functions: string.h header file contains many predefined functions to manipulate strings. The following are the some important functions in string.h header file.

No.	Function	Description
1	strlen(string_name)	returns the length of string name.
2	strcpy(destination, source)	copies the contents of source string to destination string.
3	strcat(first_string, second_string)	concats or joins first string with second string. The result of the string is stored in first string.
4	strcmp(first_string, second_string)	compares the first string with second string. If both strings are same, it returns 0.
5	strrev(string)	returns reverse string.
6	strlwr(string)	returns string characters in lowercase.
7	strupr(string)	returns string characters in uppercase.
8	strchr(s1, ch)	Returns a pointer to the first occurrence of character ch in string s1.
9	strstr(s1, s2)	Returns a pointer to the first occurrence of string s2 in string s1.
10	strrchr(s1,ch)	Returns a pointer to the last occurrence of character ch in string s1.

1. strlen():

strlen() is the function used to find the length of the given string.

Syntax: int strlen(char str[])

With strlen() function

```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[20],len;
    scanf("%s",s1);
    len=strlen(s1);
    printf("Length = %d\n",len);
    return 0;
}
```

Input: Computer Output: Length = 8

Without strlen() function

```
#include<stdio.h>
int main()
{
    char s1[20],len=0,i;
    scanf("%s",s1);
    for(i=0;s1[i]!='\0';i++)
        len++;
    printf("Length = %d\n",len);
    return 0;
}
```

Input: Computer Output: Length = 8

2. strcpy():

Syntax: char * strcpy(char dest[], char src[])

With strcpy() function

```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[20],s2[20];
    scanf("%s%s",s1,s2);
    strcpy(s1,s2);
    printf("s1 = %s\n",s1);
    printf("s2 = %s\n",s2);
    return 0;
}
```

Without strcpy() function

```
#include<stdio.h>
int main()
{
    char s1[20],s2[20];
    int i;
    scanf("%s%s",s1,s2);
    for(i=0;s2[i]!='\0';i++){
       s1[i]=s2[i];}
    s1[i]='\0';
    printf("s1 = %s\n",s1);
    printf("s2 = %s\n",s2);
    return 0;
}
```

Input: Hello Welcome **Output:** s1 = Welcome s2 = Welcome

3. **strcat():** It is used to concatenate two strings.

Syntax: char* strcat(char dest[],char src[])

With strcat() function

```
#include<stdio.h>
#include<string.h>
int main()
{
     char s1[20],s2[20];
     scanf("%s%s",s1,s2);
     strcat(s1,s2);
     printf("s1 = %s\n",s1);
     printf("s2 = %s\n",s2);
     return 0;
}
```

Without strcat() function

```
#include<stdio.h>
#include<string.h>
int main()
{
     char s1[20],s2[20];
     int i,len;
     scanf("%s%s",s1,s2);
     len=strlen(s1);
     for(i=0;s2[i]!='\0';i++){
        s1[len]=s2[i];
        len++;
       }
     s1[len]='\0';
     printf("s1 = %s\n",s1);
     printf("s2 = %s\n",s2);
     return 0;
}
```

Input: Aditya Engineering **Output:** s1 = AdityaEngineering s2 = Engineering

4. **strcmp():** is used to compare two strings. It returns 0 if both the strings are equal. If s1 is greater than s2 then it returns 1 otherwise it returns -1.

With strcmp() function

```
#include<stdio.h>
#include<string.h>
int main()
{
     char s1[20],s2[20];
     int n;
     scanf("%s%s",s1,s2);
     n=strcmp(s1,s2);
     if(n==0)
       printf("Both are equal");
     else if(n>0)
       printf("s1 > s2");
     else
       printf("s1 < s2");
     return 0;
}
```

Without strcmp() function

```
#include<stdio.h>
#include<string.h>
int main()
{
     char s1[20],s2[20];
     int i=0,l1,l2,flag=0;
     scanf("%s%s",s1,s2);
     l1=strlen(s1);
     l2=strlen(s2);
     if(l1!=l2)
        printf("Strings are not Same\n");
     else{
       while(s1[i]!='\0' && s2[i]!='\0')
       {   if(s1[i]!=s2[i]) {
              flag=1;
              break;  }
          i++;
       }
       if(flag==0)
          printf("Strings are Same");
       else
          printf("Strings are not Same");
       }
      return 0;
}
```

Input: abc abcd **Output:** Strings are not Same

Input: ramesh ramesh **Output:** Strings are Same

5. **strrev():** it is used to get the reverse of the given string.

Syntax: char* strrev(char str[])

Note: A string is said to be palindrome if reverse of the given string is equal to original string.
Example: liril, madam are some examples of palindromes.

With strrev() function

```
#include<stdio.h>
#include<string.h>
int main()
{
     char s1[20];
     scanf("%s",s1);
     printf("S1 = %s\n",s1);
     strrev(s1);
     printf("S1 = %s\n",s1);
     return 0;
}
```

Without strrev() function

```
#include<stdio.h>
#include<string.h>
int main()
{
     char s1[20],temp;
     int i,n;
     scanf("%s",s1);
     n=strlen(s1);
     for(i=0;i<n/2;i++)
       {
         temp=s1[i];
         s1[i]=s1[n-i-1];
         s1[n-i-1]=temp;
       }
     printf("Reverse String = %s\n",s1);
     return 0;
}
```

Input: Welcome **Output:** Reverse String = emocleW

Note: Refer remaining functions like strlwr(), strupr(), strchr(), strstr() etc..

Practice Programs:

```
// program to find the vowels and
// consonats in the given string
#include<stdio.h>
#include<string.h>
int main()
{
     char s1[20];
     int v=0,c=0,i;
     gets(s1);
     for(i=0;s1[i]!='\0';i++)
     {
      if(s1[i]=='A'||s1[i]=='E'||
      s1[i]=='I'||s1[i]=='E'||s1[i]=='U')
         v++;
      else
         c++;
     }
     printf("Vowels: %d Consonants: %d",v,c);
     return 0;
}
```

```
// program to find no of words
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
     char s1[20];
     int word=0,i;
     gets(s1);
     for(i=0;s1[i]!='\0';i++)
       if(s1[i]==' ')
         word++;
     printf("Word Count %d",word+1);
     getch();
     return 0;
}
```

Input: abc def ghi **Output:** 3

```
// program to check palindrome
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
     char s1[20],s2[20];
     gets(s1);
     strcpy(s2,s1);
     strrev(s1);
     int n=strcmp(s1,s2);
     if(n==0)
       printf("Palindrome");
     else
       printf("Not Palindrome");
     getch();
     return 0;
}
```

```
// program to count digits, alphabets
// symbols
#include<stdio.h>
#include<ctype.h>
int main()
{
     char s1[20];
     int d=0,a=0,s=0,i;
     gets(s1);
     for(i=0;s1[i]!='\0';i++){
       if(isdigit(s1[i]))  d++;
       else if(isalpha(s1[i])) a++;
       else s++;
      }
     printf("%d %d %d",d,a,s);
     return 0;
}
```

Input: Srinu@123 **Output:** 3 5 1

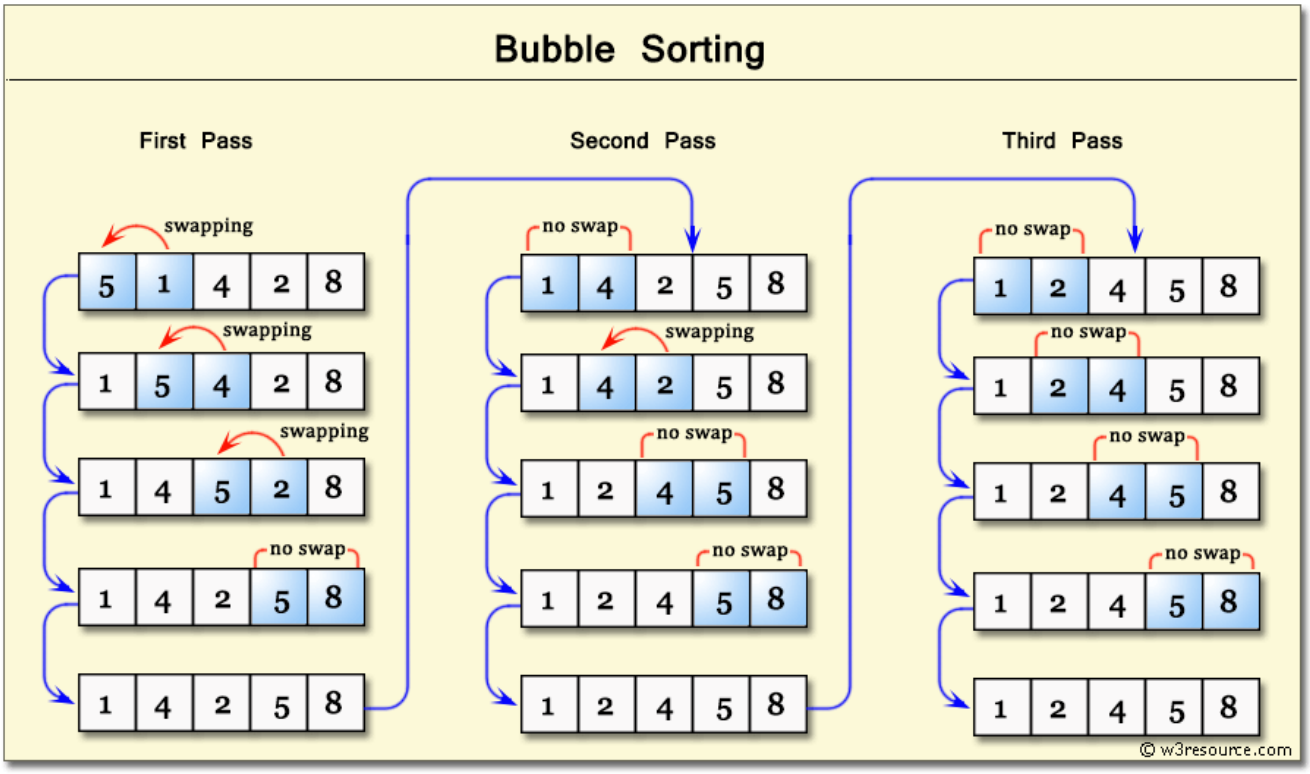
Sorting Techniques

Sorting is the process of arranging the elements of a collection (such as an array or list) in a specific order, typically in ascending or descending order.

1) **Bubble Sort**: Compares adjacent elements and swaps them if they are in the wrong order. This process repeats until the array is sorted.
   - Time Complexity: $O(n^2)$
2) **Selection Sort**: Divides the array into a sorted and an unsorted part, repeatedly selecting the smallest (or largest) element from the unsorted part and moving it to the sorted part.
   - Time Complexity: $O(n^2)$
3) **Insertion Sort**: Builds the sorted array one element at a time by repeatedly taking an element from the unsorted part and inserting it into the correct position in the sorted part.
   - Time Complexity: $O(n^2)$ ($O(n)$ in the best case)
4) **Merge Sort**: A divide-and-conquer algorithm that splits the array into halves, recursively sorts each half, and then merges the sorted halves back together.
   - Time Complexity: $O(n \log n)$
5) **Quick Sort**: Another divide-and-conquer algorithm that selects a 'pivot' element and partitions the array into elements less than and greater than the pivot, then recursively sorts the partitions.
   - Time Complexity: $O(n \log n)$ ($O(n^2)$ in the worst case)

Bubble Sort:

Bubble Sort is a simple comparison-based sorting algorithm that repeatedly steps through a list, compares adjacent elements, and swaps them if they are in the wrong order. The process continues until no more swaps are needed, which means the list is sorted.



Example:

For the array [5, 2, 9, 1, 5], Bubble Sort would proceed as follows:

- First pass: Compare and swap: [2, 5, 1, 5, 9]
- Second pass: Compare and swap: [2, 1, 5, 5, 9]
- Third pass: Compare and swap: [1, 2, 5, 5, 9]
- No swaps in the next pass, so the algorithm stops.

Program:

```
#include <stdio.h>
void bubbleSort(int arr[], int n)
{
   int i, j, temp;
   // Outer loop for each pass
   for (i = 0; i < n - 1; i++)
    {
      // Inner loop for comparison and swapping
      for (j = 0; j < n - i - 1; j++)
      {
         if (arr[j] > arr[j + 1])
          {
            // Swap arr[j] and arr[j + 1]
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}
void printArray(int arr[], int n)
{
   int i;
   for (i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}

int main()
{
   int n, i;

   // Ask the user for the number of elements
   printf("Enter the number of elements: ");
   scanf("%d", &n);

   int arr[n];
   // Read elements into the array
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }
   // Sort the array using Bubble Sort
   bubbleSort(arr, n);

   // Display the sorted array
   printf("Sorted array:\n");
   printArray(arr, n);
   return 0;
}
```

Selection Sort:

Selection Sort is a straightforward comparison-based sorting algorithm that works by dividing the input array into two parts: a sorted section and an unsorted section. The algorithm repeatedly selects the smallest (or largest, depending on the desired order) element from the unsorted section and moves it to the end of the sorted section.

Example:

For the array [64, 25, 12, 22, 11], Selection Sort would proceed as follows:

- First pass: Find the minimum (11), swap with the first element: [11, 25, 12, 22, 64]
- Second pass: Find the minimum (12), swap with the second element: [11, 12, 25, 22, 64]
- Third pass: Find the minimum (22), swap with the third element: [11, 12, 22, 25, 64]
- Fourth pass: The remaining elements are already in order.

Program:

```
#include <stdio.h>
void selectionSort(int arr[], int n)
{
   int i, j, minIndex, temp;

   // Outer loop for each position in the array
   for (i = 0; i < n - 1; i++) {
      // Assume the minimum is the first element of the unsorted section
      minIndex = i;

      // Inner loop to find the minimum element in the unsorted section
      for (j = i + 1; j < n; j++) {
         if (arr[j] < arr[minIndex]) {
            minIndex = j;          // Update minIndex if a smaller element is found
         }
      }

      // Swap the found minimum element with the first element of the unsorted section
      if (minIndex != i)
      {
         temp = arr[i];
         arr[i] = arr[minIndex];
         arr[minIndex] = temp;
      }
   }
}

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

```
int main()
{
   int n, i;

   // Ask the user for the number of elements
   printf("Enter the number of elements: ");
   scanf("%d", &n);

   int arr[n];

   // Read elements into the array
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   // Sort the array using Selection Sort
   selectionSort(arr, n);

   // Display the sorted array
   printf("Sorted array:\n");
   printArray(arr, n);

   return 0;
}
```

Searching Techniques

Searching techniques are algorithms used to locate a specific value or element within a data structure (like an array or list).

Common Searching Techniques:

1. **Linear Search**:
   - **Description**: Scans each element in a list or array sequentially until the desired element is found or the end of the structure is reached.
   - **Time Complexity**: O(n)
   - **Use Case**: Simple to implement; best for small or unsorted data.
2. **Binary Search**:
   - **Description**: Works on sorted arrays. It repeatedly divides the search interval in half, comparing the target value to the middle element. If the target is less than the middle element, it searches the left half; otherwise, it searches the right half.
   - **Time Complexity**: O(log n)
   - **Use Case**: Efficient for large, sorted datasets.

Program to implement Linear Search:

```
#include <stdio.h>
int linearSearch(int arr[], int n, int target)
{
   for (int i = 0; i < n; i++) {
      if (arr[i] == target) {
         return i;                    // Return the index if the target is found
      }
   }
   return -1;                         // Return -1 if the target is not found
}
```

```
int main()
{
   int n, target, result;

   // Ask the user for the number of elements
   printf("Enter the number of elements: ");
   scanf("%d", &n);

   int arr[n];

   // Read elements into the array
   printf("Enter %d elements:\n", n);
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   // Ask the user for the target value to search
   printf("Enter the element to search for: ");
   scanf("%d", &target);

   // Perform linear search
   result = linearSearch(arr, n, target);

   // Display the result
   if (result != -1)
   {
      printf("Element %d found at index %d.\n", target, result);
   }
   else
   {
      printf("Element %d not found in the array.\n", target);
   }

   return 0;
}
```

Program to implement Binary Search:

```
#include <stdio.h>
int binarySearch(int arr[], int n, int target)
{
   int left = 0, right = n - 1;

   while (left <= right)
   {
      int mid = left + (right - left) / 2;          // To prevent overflow

      // Check if the target is present at mid
      if (arr[mid] == target)
      {
         return mid;                                // Return the index if the target is found
      }
```

```
        // If the target is greater, ignore the left half
        if (arr[mid] < target)
        {
            left = mid + 1;
        }
        else
        {   // If the target is smaller, ignore the right half
            right = mid - 1;
        }
    }

    return -1;                                  // Return -1 if the target is not found
}

int main()
{
    int n, target, result;

    // Ask the user for the number of elements
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];

    // Read elements into the array
    printf("Enter %d sorted elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Ask the user for the target value to search
    printf("Enter the element to search for: ");
    scanf("%d", &target);

    // Perform binary search
    result = binarySearch(arr, n, target);

    // Display the result
    if (result != -1)
    {
        printf("Element %d found at index %d.\n", target, result);
    }
    else
    {
        printf("Element %d not found in the array.\n", target);
    }

    return 0;
}
```

MCQ Questions with Answers

What is the size of an array declared as int arr[10];?
A) 10 B) 20 C) Depends on the system D) 40

Which of the following is the correct way to access the first element of an array arr?
A) arr[1] B) arr[0] C) &arr[0] **D) Both B and C**

What will be the output of the following code?
int arr[5] = {1, 2, 3, 4, 5};
printf("%d", arr[2]);
A) 1 B) 2 **C) 3** D) 4

Which of the following declares an array of 10 integers?
A) int arr[10]; B) int arr(10); C) int arr{10}; D) int arr = new int[10];

What is the output of the following code?
int arr[] = {1, 2, 3, 4}; printf("%d", arr[4]);
A) 1 B) 4 **C) Compilation error** D) Undefined behavior

Which of the following statements correctly initializes an array of 5 floats?
A) float arr[5]; B) float arr = {1.0, 2.0, 3.0, 4.0, 5.0};
C) float arr[5] = {1.0, 2.0, 3.0}; **D) Both A and C**

Which of the following can be used to store a string in C?
A) char str[10]; B) char *str; **C) Both A and B** D) None of the above

Which of the following is the correct way to declare a 2D array of integers?
A) int arr[5][5]; B) int arr(5)(5); C) int arr[5, 5]; D) int arr{5}{5};

Which of the following statements about arrays is true?
A) Arrays can hold different data types. **B) The size of an array must be constant.**
C) Arrays can be resized after declaration. D) None of the above.

What does the sizeof operator return when applied to an array?
A) The size of the first element **B) The total size in bytes of the array**
C) The number of elements in the array D) A pointer to the array

In C, what happens when you assign an array to another array?
A) The entire array is copied. B) Only the first element is copied.
C) You cannot assign arrays directly. D) A reference to the array is created.

What is the output of the following code?
int arr[3] = {1, 2, 3};
printf("%d", arr[arr[1]]);
A) 2 **B) 3** C) 1 D) Undefined behavior

Which of the following will NOT compile?
A) int arr[5] = {1, 2, 3}; **B) int arr[5] = {1, 2, 3, 4, 5, 6};**
C) int arr[5] = {}; D) int arr[] = {1, 2, 3};

What function is used to calculate the length of a string in C?
A) strlength() **B) strlen()** C) string_length() D) length()

Which of the following is the correct way to declare a string in C?
A) char str[10]; B) char str[] = "Hello"; **C) Both A and B** D) None of the above

What will be the output of the following code?
```
char str[] = "Hello, World!";
printf("%c", str[7]);
```
A) H B) e **C) W** D) !

Which function is used to copy one string to another in C?
A) strcpy() B) copystr() C) string_copy() D) strcopy()

What does the function strcat() do?
A) Compares two strings B) Copies one string to another
C) Concatenates two strings D) Finds the length of a string

Which of the following functions is used to compare two strings in C?
A) string_compare() **B) strcmp()** C) str_compare() D) compare()

What is the purpose of the function strchr()?
A) To find a character in a string B) To find the length of a string
C) To convert a string to lowercase D) To concatenate two strings

Which function can be used to convert a string to an integer?
A) atoi() B) itoa() C) str_to_int() D) convert()

Which function would you use to find the first occurrence of a substring in a string?
A) strlocate() B) strfind() **C) strstr()** D) strchr()

What is the maximum length of a string that can be defined in C?
A) 255 characters B) 1024 characters
C) There is no fixed limit, depends on memory D) 512 characters

What is the worst-case time complexity of bubble sort?
A) O(n) B) O(n log n) **C) O(n^2)** D) O(log n)

Which sorting algorithm is generally considered the fastest for small datasets?
A) Bubble Sort B) Selection Sort **C) Insertion Sort** D) Quick Sort

What is the time complexity of linear search in the worst case?
A) O(1) B) O(log n) **C) O(n)** D) O(n log n)

In a binary search algorithm, what is the main advantage over linear search?
A) It works on unsorted arrays. B) It is easier to implement.
C) It is faster with a time complexity of O(log n). D) It requires less memory.