3.6 Strings:

String is sequence of characters i.e letters, numbers and symbols and punctuation marks. The string in C programming language is actually a one-dimensional array of characters which is terminated by a null character '\0'. Since string is an array, the declaration of a string is the same as declaring a char array. Characters are enclosed in single quotes(‘ ‘) and strings are enclosed in double quotes(“ “).

String → **char [10];**

↓

One dimensional array

Syntax:

Data type variable_name[size];

↓

Char variable_name[size];

- ✓ You should not specify the datatype because string only char datatype.

char str1[20];
char str2[7] = “String”;

The following declaration creates string named *“str2”* and initialized with value *“String”*. To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word**.**

Declaration and Initialization of Strings

The following declaration and initialization create a string consisting of the word "Hello".

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

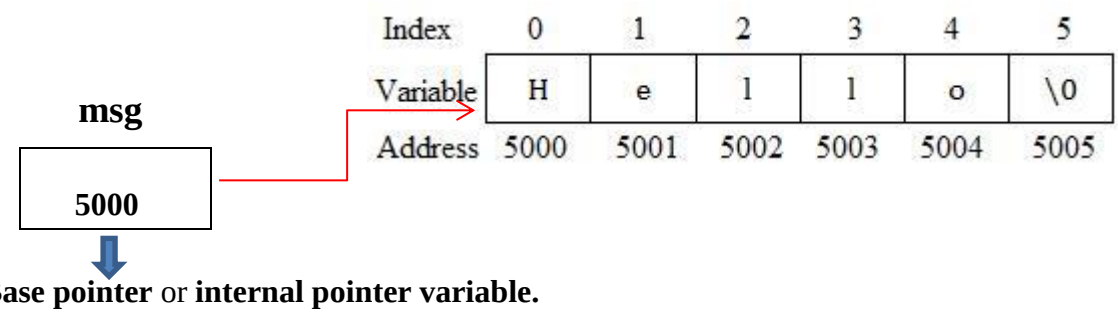
Another way of Initialization is (shortcut for initializing string)

char greeting[] = "Hello";

Note: The C compiler automatically places the '\0' at the end of the string when it initializes the array. The terminating null (‘\0’) is important, because it is the only way the functions that work with a string can know where the string ends.

Memory Representation of String

char msg[] = "Hello";



Base pointer or **internal pointer variable.**

/* Program to demonstrate printing of a string */

```
int main( )
{
char name[ ] = “Ramesh" ;
        int i = 0 ;
        while ( i <= 9 )
        {
                printf ( "%c", name[i] ) ;
                i++ ;
        }
    return 0;
}
```

OUTPUT :

Ramesh

Following program illustrates printing string using ‘\0’.

```
int main( )

        char name[ ] = “Ramesh" ;
        int i = 0 ;
        while ( name[i] != ‘\0’ )
        {
                printf ( "%c", name[i] ) ;
                i++ ;
        }
    return 0;
}
```

OUTPUT:

Ramesh

Note:

- ✓ To read and write string elements we need formatting specifier %s. here no need to use iterators (loops).
- ✓ Generally, to read input from the keyboard we provide & (address operator) but to read string we need not provide & operator because the string variable always holds the base address only, The compiler automatically collect the all characters one by one.

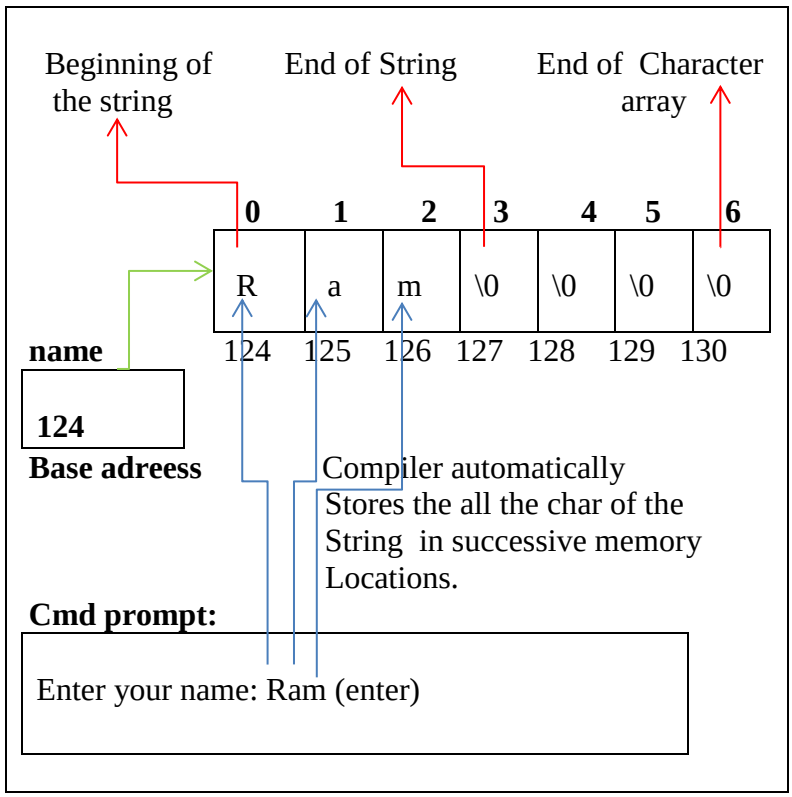
Example :

```
#include <stdio.h>
void main( )
{

 char name[7];
 printf(“enter your name);
scanf(“%s”,name);
printf(“\n Hi %s ,Welcome to
CSE”,name);

 }
```

OUTPUT:

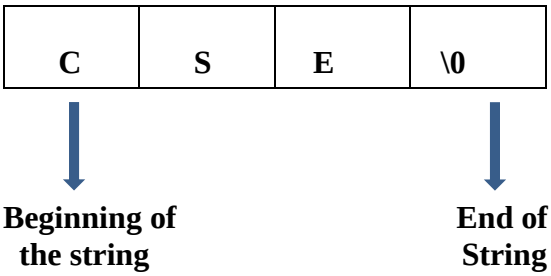
Enter your name: Ram
Hi Ram, Welcome to CSE

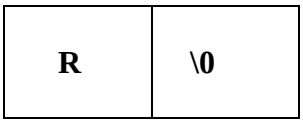
.

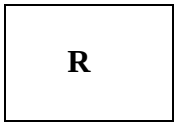


- ✓ When the compiler assigns a character string to a character array, it automatically appends null character to end of string .The size of string should be equal to maximum no of characters in the string plus one.
- ✓ In above program, the compiler creates a character array of size 7,stores the “Ram” in it and finally terminates the value with null character. Rest of the elements in the array are automatically initialized to NULL.

3.6.1 Difference between character storage and String Storage:

<table>
<tr><th>String</th><th>Character</th></tr>
<tr><td>
1.
char branch[]=”CSE”;


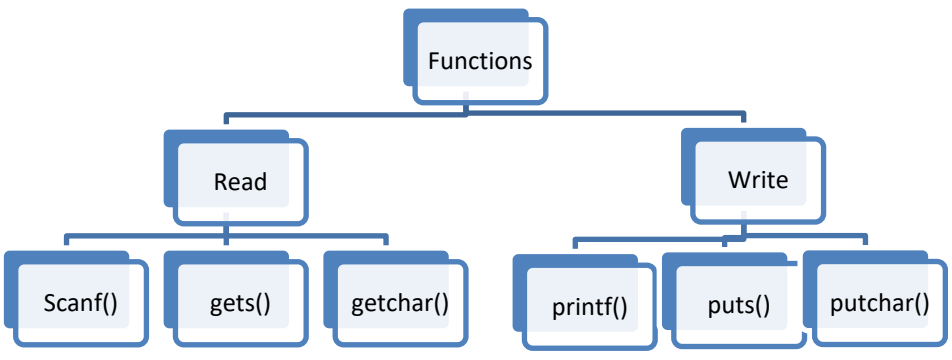
2.
Char str[]=”R”;


✓ Here R is a string not a character .
✓ The string H requires two memory locations one stores the character R and another to store the null character.
</td><td>
Char ch=’R’;


✓ Here R is a character not a string.
✓ The character R requires only one memory location.
</td></tr>
<tr><td colspan="2">
3. Char str[]=””;
Empty string ⟵ \0
✓ C permits empty string.
✓ it does not allow an empty character.
</td></tr>
</table>

3.7 Functions to Read and Write Strings / Characters:

- ✓ The string can be read by three ways, i.e scanf(),gets() and getchar().the getchar function reads the characters.these functions also called as string input functions.
- ✓ The string can be write by three ways, i.e printf(),puts() and putchar().the putchar function prints the characters. These functions also known as string output functions



Types of String input output functions

3.7.1 getchar() and putchar() functions

- ✓ The ***getchar()*** function reads a character from the terminal and returns it as an integer. This function reads only single character at a time. You can use this method in the loop in case you want to read more than one characters.
- ✓ The ***putchar()*** function prints the character passed to it on the screen and returns the same character. This function puts only single character at a time. In case you want to display more than one characters, use putchar() method in the loop.

```
#include <stdio.h>
int main( )
{
        int ch;
        printf("Enter a character:");
        ch=getchar();
        putchar(ch);
        return 0;
}
```

When you will compile the above code, it will ask you to enter a value. When you will enter the value, it will display the value you have entered.

3.7.2 puts() and gets() functions

- ✓ The *gets()* function reads a line or multiword string from stdin into the buffer pointed to by *s* until either a terminating newline or EOF (end of file).

The *puts()* function writes the string *s* and a trailing newline to stdout.

```
#include<stdio.h>
int main()
{
        char str[100];
        printf("Enter a string:");
        gets( str );
        puts("Hello!");
        puts(str);
        return 0;
}
```

Output

Enter a string: Ramesh M
Hello! Ramesh M

3.7.3 Difference between scanf() and gets() and printf() and puts() :

S.No	scanf()	gets()
1	It reads single word strings	It reads multi-word strings
2	It stops reading characters when it encounters a whitespace	It stops reading characters when it encounters a newline or EOF
3	Example: Ramesh	Example: Ramesh m

S.No	printf()	puts()
1	It prints a variable with using formatting specifiers.	It prints a variable without using formatting specifier.
2	For new line we provide \n within printf.	For new line no need provide \n.
3	**Example:** printf(“%s \n”,sname);	**Example:** puts(sname);

3.8 Standard Library Character Functions:

With every C compiler a large set of useful character handling library functions are provided. For using these functions, we need to include the header file ***ctype.h .***

Functions	*Description*	*Example*
isalnum(int c)	Check whether character c is an alphanumeric or not.	isalnum('R');
isalpha(int c)	Check whether character c is an alphabetic or not	isalpha('s');
iscntrl(int c)	Check whether character c is control character or not	scanf("%d",&c); iscntrl(c);
isdigit(int c)	Check whether character c is a digit or not	isdigit(3);
islower(int c)	Check whether character c is lower case or not	islower('s');
isupper(int c)	Check whether character c is upper case or not	isupper('R');
ispunct(int c)	Check whether character c is punctuation mark or not	ispunct('?');
isspace(int c)	Check whether character c is a white space or not.	isspace(' ');
isxdigit(int c)	Check whether character c is a hexadecimal digit or not	isxdigit('F')
tolower(int c)	Converts the character to lower case	tolower('R') return r
toupper(int c)	Converts the character to Upper case	toupper('r') return R

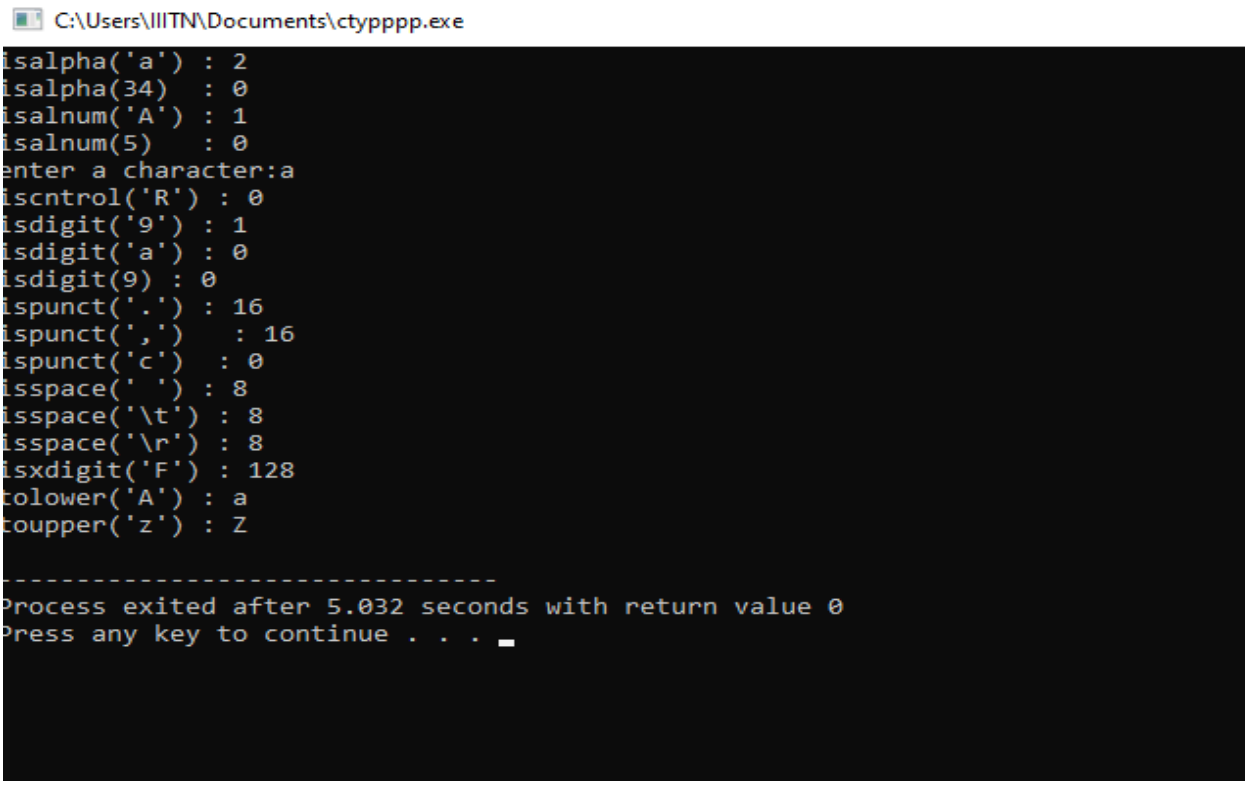
Note :

- ✓ The isalpha( ) function returns nonzero if ch is a letter of the alphabet; otherwise zero is returned.
- ✓ The isalnum( ) function returns nonzero if its argument is either a letter of the alphabet or a digit.If the character is not alphanumeric, zero is returned.
- ✓ The iscntrl( ) function returns nonzero if ch is a control character.
- ✓ The isdigit( ) function returns nonzero if ch is a digit, that is, 0 through 9. Otherwise zero is returned.
- ✓ The ispunct( ) function returns nonzero if ch is a punctuation character; otherwise zero is returned.
- ✓ The isspace( ) function returns nonzero if ch is a white-space character, including space, horizontaltab, vertical tab, formfeed, carriage return, or newline character; otherwise zero is returned.
- ✓ The isxdigit( ) function returns nonzero if ch is a hexadecimal digit; otherwise zero is returned

Example:

```
#include <stdio.h>
#include <ctype.h>
int main()
{
printf("isalpha('a') : %d\n" , isalpha('a'));
printf("isalpha(34)  : %d\n" , isalpha(34));
printf("isalnum('A') : %d\n" , isalnum('A') ) ;
printf("isalnum(5)   : %d\n" , isalnum(5) );
char ch='R';
printf("enter a character:");
scanf("%d"&ch);
printf("iscntrol('R') : %d\n" , iscntrl(ch)) ;
printf("isdigit('9') : %d\n" , isdigit('9'));
printf("isdigit('a') : %d\n" , isdigit('a')) ;
printf("isdigit(9) : %d\n" , isdigit(9)) ;
printf("ispunct('.') : %d\n" , ispunct('.')) ;
printf("ispunct(',')   : %d\n" , ispunct(',')) ;
printf("ispunct('c')  : %d\n" , ispunct('c')) ;
printf("isspace(' ') : %d\n" , isspace(' ')) ;
printf("isspace('\\t') : %d\n" , isspace('\t'));
printf("isspace('\\r') : %d\n" , isspace('\r')) ;
printf("isxdigit('F') : %d\n" , isxdigit('F')) ;
printf("tolower('A') : %c\n" , tolower('A')) ;
printf("toupper('z') : %c\n" , toupper('z')) ;
return 0;

}
```

OUTPUT :



3.9 Standard Library String Functions

With every C compiler a large set of useful string handling library functions are provided. For using these functions, we need to include the header file ***string.h***

Function	Description
strlen()	Finds length of a string
strlwr()	Converts a string to lowercase
strupr()	Converts a string to uppercase
strcat() strncat()	Appends one string at the end of another. And strncat() upto n characters.
strcpy() strncpy()	Copies a string into another and copies a string upto n characters
strcmp() strncmp()	Compares two strings and compare upto n numbers.
strchr()	Finds first occurrence of a given character in a string.
strrchr()	finds first occurrence of a given character in a string from end.
strstr()	Finds first occurrence of a given string in another string.
strrev()	Reversing of a string.
strspn()	Strspn() is used to returns the index of the first character in string that doesn't match character in str2
strcspn()	Strcspn() is used to returns the index of the first character in string that matches of character in str2
strtok()	Strtok() is used isolate sequential tokens in a terminated a null-terminated string. these tokens are separated in the string using delimiters.

3.9.1 strlen() function

This function counts the number of characters present in a string. Syntax for strlen() function is given below:

```
size_t strlen(const char *str);
```

The function takes a single argument, i.e, the string variable whose length is to be found, and returns the length of the string passed.

Note: While calculating the length it doesn't count '\0'.

Example 1: C program that illustrates the usage of *strlen()* function.

```
#include<stdio.h>
#include<string.h>
int main( )
{
        char str[ ] = "Ramesh" ;
        int len1, len2 ;
```

```
        len1 = strlen ( str ) ;
        len2 = strlen ( "hello students" ) ;
        printf ( "\nThe string %s length is %d", str, len1 ) ;
        printf ( "\nThe string %s length is %d\n", "hello students", len2 ) ;
        return 0;
}
```

Output

The string *Ramesh* length is 6
The string *hello students* length is 14

3.9.2 strlwr( ) and strupr( ) :

These function converts to lower to upper and upper to lower. Syntax for strlwr() and strupr() functions given below.

Char strlwr(const char *str)
Char strlwr(const char *str)

Example 2 : C program that illustrates the usage of *strlwr()* and *strupr()*f unction.

```
#include<stdio.h>
#include<string.h>
int main()
{
        char str1[ ] = "RGUKT" ;
        printf ( "\n converting string to lower to upper case : %s", strlwr (str1)) ;
        printf ( "\n converting string to Upper to lower case : %s", strupr (str1)) ;
        return 0;
}
```

Output:

converting string to lower to upper case : rgukt
converting string to Upper to lower case : RGUKT.

3.9.3 strcpy( ) function

This function copies the contents of one string into another. Syntax for strcpy() function is given below.

char * strcpy (char * destination, const char * source);

Example

strcpy (str1, str2) – It copies contents of str2 into str1.
strcpy (str2, str1) – It copies contents of str1 into str2.

If destination string length is less than source string, entire source string value won't be copied into destination string.
For example, consider destination string length is 20 and source string length is 30. Then, only 20 characters from source string will be copied into destination string and remaining 10 characters won't be copied and will be truncated.

Example 3: C program that illustrates the usage of *strcpy()* function.

```
#include<stdio.h>
#include<string.h>
int main()
{
        char source[ ] = "Nuzivid" ;
        char destination[20]= "" ;
        strcpy (destination, source ) ;
        printf ( "\nSource string = %s", source ) ;
        printf ( "\nDestination string = %s", destination ) ;
        return 0;
}
```

Output
Source string = Nuzvid
Destinnation string = Nuzvid

3.9.4 strncpy( ) function

This function copies up to n characters from string.if n is zero or negative then nothing is copied. Syntax for strncpy( ) function is given below.

char * strncpy (char * destination, const char * source, size_t n);

Example 4: C program that illustrates the usage of *strcpy()* function.

```
#include<stdio.h>
#include<string.h>
int main()
{
        char source[ ] = "Nuzivid" ;
        char destination[20]= "" ;
        strcpy (destination, source,3 ) ;
        printf ( "\nSource string = %s", source ) ;
        printf ( "\nDestination string = %s", destination ) ;
        return 0;
}
```

Output
Source string = Nuzvid
Destinnation string = Nuz

3.9.5 strcat( ) function

It combines two strings. It concatenates the source string at the end of the destination string. Syntax for strcat( ) function is given below.

char * strcat (char * destination, const char * source);

For example, “Visakhapatanam” and “Vizag” on concatenation would result a new string “VisakhapatnamVizag”.

Example 5: C program that illustrates the usage of *strcat()* function.

```
#include<stdio.h>
#include<string.h>
int main( )
{
        char source[ ] ="Students!" ;
        char target[30] = "Hello" ;
        strcat ( target, source ) ;
        printf ( "\nSource string = %s", source ) ;
        printf ( "\nDestination string = %s", target ) ;
        return 0;
}
```

Output
Source string = Students!
Destination string = HelloStudents!

3.9.6 strcnat( ) function

It combines two strings. It concatenates the source string at the end of the destination string up to n character. Copying stops when n character s ae copied. Syntax for strncat( ) function is given below.

```
char * strncat ( char * destination, const char * source size_t n);
```

For example, “Vizag” and “Visakhapatam” on concatenation (upto n Characters) would result a new string “VizagVisakha”. Here n is 7.

```
#include<stdio.h>
#include<string.h>
int main( )
{
        char source[ ] =" Visakhapatnam " ;
        char target[30] = "Vizag" ;
        strncat ( target, source,7) ;
        printf ( "\nSource string = %s", source ) ;
        printf ( "\nDestination string = %s", target ) ;
        return 0;
}
```

Output

Source string = Visakhapatnam
Destination string = VizagVisakha

3.9.7 strcmp( ) and strncmp() functions

It compares two strings to find out whether they are same or different. The two strings are compared character by character until there is a mismatch or end of one of the strings is reached, whichever occurs first.s

- ✓ If the two strings are identical, **strcmp( )** returns a value zero. If they’re not identical, it returns the *numeric* difference between the ASCII values of the first non-matching pairs of characters.
- ✓ strncmp() function compares at most the first n characters of the string1 and string2.

Syntax for strcmp( ) function is given below.

```
int strcmp ( const char * str1, const char * str2)
int strncmp(const char *str1,const char *str2,size-t n)
```

Return Value from strcmp()

Return Value	Description
0	if both strings are identical (equal)
<0	if the ASCII value of first unmatched character is less than second.
>0	if the ASCII value of first unmatched character is greater than second.

Note: *strcmp( )* function is case sensitive. i.e, “A” and “a” are treated as different characters.

Example 6: C program that illustrates the usage of *strcmp()* function.

```
#include<stdio.h>
#include<string.h>
int main( )
{
        char string1[ ] = "Cse" ; char string2[ ] = "Ece";
         int i, j, k ;
        i = strcmp ( string1, "Cse" ) ;
         j = strcmp ( string1, string2 ) ;
        k = strcmp ( string1, "Cse branch" ); printf ( "\n%d %d %d", i, j, k ) ;
```

```
}
```
Output
```
0       4       -32
```

Example 7: C program that illustrates the usage of *strncmp()* function.
```
#include<stdio.h>
#include<string.h>
int main( )
{
        char string1[ ] = "Cse" ;
        char string2[ ] = "Ece";
         int i, j, k ;
        i = strncmp( string1, "Civil",1);
         j = strncmp(string2, “Eee” ,2);
        k = strcmp ( string1, "Air" );
        printf ( "\n%d %d %d", i, j, k ) ;
}
```
Output :
```
          0   -1   1
```

3.9.8 strrev( ) function

this function used to reversing of a string . Syntax for strrev( ) function is given below.

char *strrev(char *string)

Example 7:Write a program to implement "strrev()" function
```
#include <stdio.h>
 #include <string.h>
int main()
{
        char str[20];
        printf("Enter any String:");
        gets(str);
        printf("Reverse of String:");
        puts(strrev(str));
}
```
Output:
Enter any String: Ramesh
Reverse of String: hsemaR

Example 8: Program for checking string’s palindrome property.
```
#include<stdio.h>
#include<string.h>
int main()
{
   char str1[25],str2[25];
   int d=0;
   printf("\nEnter a string:");
   gets(str1);
   strcpy(str2,str1);
   strrev(str1);
   d= strcmp(str1,str2);
   if(d==0)
      printf("\n%s is pallindrome",str2);
   else
      printf("\n%s is not a pallindrome",str2);
    return 0;
}
```
Output:
Enter a string: madam
madam is palindrome

Some other palindrome strings are:
civic, dad, malayalam, mom,wow etc.

3.9.9 strchr( ) and strrchr() functions :

- ✓ It searches for the first occurrence of the character in the string. the function returns a pointer pointing to the first character, or null if no match is found.
- ✓ The strrchr forSyntax for strchr( ) function is given below.

```
char * strchr ( char * str, int c );
```

Example 9: Write a program to implement "strchr()" function

```
#include<stdio.h>
#include<string.h>
int main()
{
char str[20]="programming In c";
char *pos;
pos=strchr(str, 'n');
if(pos)
{
        printf("\n n is found in str at address posision %d",pos);
        printf("\n n is found in str at index posision %d",pos-str);
}
else
{
printf( "\n n is not found in string");
}
return 0;
}
```

Output:

```
n is found string at address position 661863
n is found string at address position 9
```

Syntax for strrchr( ) function is given below.

```
char *strrchr(const char *str,int c)
```

Example 10 : Write a program to implement "strrchr()" function

```
#include<stdio.h>
#include<string.h>
int main()
{
char str[20]="programming In c";
char *pos;
pos=strrchr(str, 'n');
if(pos)
{
        printf("\n n is found in str at posision %d",pos);
        printf("\n n is found in str at posision %d",pos-str);

}
else
{
printf( "\n n is not found in string");
}
return 0;
}
```

Output:

```
n is found string at address position 661837
n is found string at address position 13.
```

3.9.10 strstr( ) functions :

- ✓ It searches for the first occurrence of the sub string in the string. the function returns a pointer pointing to the first character, or null if no match is found.
- ✓ The strrchr for Syntax for strstr( ) function is given below.

```
char * strstr( char * str, int c );
```

Example11 : Write a program to implement "strstr()" function

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str1[]="HAPPY TEACHERSDAY TO ALL";
        char str2[]="DAY";
        char *res;
        res=strstr(str1,str2);
        printf("The result value:%d\n",res);
        if(res)
                printf("The sub string is found!!\n");
        else
                printf("The sub string is not found!!\n");
        return 0;
}
```

Output:

The substring address position at 1636788
The sub string is found !!

3.9.11 strspn( ) functions :

- ✓ The function returns the index of the first character in str1 that doesn't match any character any character in str2.
- ✓ The for Syntax for strspn( ) function is given below.

```
size-t strspn(const char *str1,const char *str2);
```

Example 12 : Write a program to implement "strspn()" function

```
int main()
{
        char str1[20];
        char str2[20];
        int count;
        printf("Enter String1:");
        gets(str1);
        printf("Enter String2:");
        gets(str2);
        count=strspn(str1,str2);
        printf("Similar characters occurred upto %d\n",count);
       return 0;
}
```

Output:

Enter String1: Welcome to CSE
Enter String1: Welcome to ECE
Similar characters occurred upto 12.

3.9.12 strcspn( ) functions :

- ✓ This function returns the index of the first character in str1 that matches any character any character in str2.
- ✓ The for Syntax for strspn( ) function is given below.

size-t strspn(const char *str1,const char *str2);

Example 13: Write a program to implement "strcspn()" function

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str1[20];
        char str2[20];
        int count;
        printf("Enter String1:");
        gets(str1);
        printf("Enter String2:");
        gets(str2);
        count=strcspn(str1,str2);//number of characters before first occurance
        printf("The first matched character:%d\n",count);
}
```

```
Enter String1: programming in c
Enter String1: in
The first matched character: 8
```

3.9.13 strtok( ) functions :

- ✓ This function is used isolate sequential tokens in a terminated a null-terminated string. these tokens are separated in the string using delimiters.
- ✓ The for Syntax for strtok( ) function is given below.

char strtok(char *str1,const char *delimiter);

Example 14: Write a program to implement "strtok()" function

```
#include<stdio.h>
 #include<string.h>
int main()
{

char str[100];
char deli[100];
char *token;
printf("Enter any string with delimiters :");
gets(str);
printf("enter delimiter :");
gets(deli);
token=strtok(str,deli);
while(res!=NULL)
{
        printf("\n %s",token);
        token=strtok(NULL,deli);
        }
```

Output:
Enter any string with delimiters: ramesh,cse,iiit nuzvid,programming in c
Enter delimiters: ,
ramesh
cse
iiit nuzvid
programming in c

3.10 String manipulation without using Standard library functions:

1. Finding the length of a string.
2. Converting characters of a string into **Lower to Upper** case and **Upper to Lower case.**
3. Concatenating two Strings.
4. Comparing two Strings.
5. Reverse of a String.
6. Extracting a substring from left, right and middle.

1. Write a program to find length of a string

```
#include <stdio.h>
 int main()
{
        char str[100];
        int i,length=0;
        printf("Enter a string:");
        gets(str);
        puts(str);
        for(i=0;str[i]!='\0';i++)
        {
                length++;
        }
        printf("The length of the %s is: %d\n",str,length);
}
```

Enter string : Ramesh
The length of Ramesh is : 6

2. Write a program to convert to upper to lower case.

```
#include <stdio.h>
int main()
{
        char str[100];
        int i,length;
        printf("Enter any string:");
        gets(str);
        for(i=0;str[i]!='\0';i++)
        {
                length++;
        }
        for(i=0;i<=length;i++)

        {
                if((str[i] >='a' && str[i]<='z')
                        str[i]=str[i]-32;
        }
```

Output:
Enter any string: ramesh
The string in upper case: RAMESH

```
        printf("The string in uppercase:");
        puts(str);
}
```

3. Write a program to convert to lower to upper case.

```
#include <stdio.h>
int main()
{
        char str[100];
        int i,length;
        printf("Enter any string:");
        gets(str);
        for(i=0;str[i]!='\0';i++)
        {
                length++;
        }
        for(i=0;i<=length;i++)
        {
                if((int)str[i] >=65 && (int)str[i]<=90)
                        str[i]=str[i]+32;
        }
        printf("The string in lower case:");
        puts(str);
}
```

Output:
Enter any string: CSE
The string in lower case: cse

4. Write a program to concatenating two and stores the result in new string.

```
#include<stdio.h>
int main()
{
        char str1[100],str2[100],str3[100];
        int i=0,j=0;
        printf("\n enter the first string:");
        gets(str1);
        printf("\n enter the second string: ");
        gets(str2);
        while(str1[i]!='\0')
        {
                str3[i]=str1[i];
                i++;
                j++;
        }
        i=0;
        while(str2[i]!='\0')
        {
                str3[j]=str2[i];
                i++;
                j++;
        }
        str3[j]='\0';
        printf("\n The Concatenated string is :");
        puts(str3);
        return 0;
}
```

Output:
enter the first string: Cse
enter the second string : IT
The Concatenated string is: CseIT

5. Write a program to concatenating (appending) a string to another string.

```
#include<stdio.h>
int main()
{
        char source[50],target[100];
        int i=0,j=0;
        printf("\n Enter the source string:");
        gets(source);
        printf("\n Enter the target string: ");
        gets(target);
        while(target[i]!='\0')
        {

                i++;
        }
        while(source[j]!='\0')
        {
                target[i]=source[j];
                i++;

                j++;
        }
        target[j]='\0';
        printf("\n The final target string is :");
        puts(target);
        return 0;

}
```

Output :
Enter the source string: Robotics
Enter the target string: AI &
The final target string is: AI&Robotics

6. Write a program to comparing two strings

```
#include<stdio.h>
#include<string.h>
int main()
{
        char str1[50],str2[50];
        int i=0,len1=0,len2=0,s=0;
        printf("\nEnter the first string:");
        gets(str1);
        printf("\n Enter second string:");
        gets(str2);
        len1=strlen(str1);
        len2=strlen(str2);
        if(len1==len2)
        {
                while(i<len1)
                {
                        if(str1[i]==str2[i])
                    i++;
                else break;
                }
                if(i==len1)
                {
                        printf("two strings are equal");
```

```
                        s=1;
                }
        }
        if(len1!=len2)
        {
                printf("\n two strings are not equal");

        }
        if(s==0)
        {
                if(str1[i]>str2[i])
                  printf("\n string 1 is greater than string 2 ");
                if(str1[i]<str2[i])
                  printf("\n string 1 is lessthan than string 2 ");
        }
}
```

Output :
Enter the first string: Cse
Enter the second string : Cse
Two strings are equal

7. Write a program to reverse of a strings.

```
#include<stdio.h>
#include<string.h>
int main()
{
        char str[100],temp;
        int i=0,j=0;
        printf("Enter a string ");
        gets(str);
        j=strlen(str)-1;
        while(i<j)
        {
                temp=str[j];
                str[j]=str[i];
                str[i]=temp;
                i++;
                j--;
        }
        printf("the reversed string is:");
        puts(str);
        return 0;
}
```

Output :
Enter a string : cse
the reversed string is: esc

8. Write a program to extract the first N characters (Sub String) of a string from left.

```
#include<stdio.h>
int main()
{
        char str[50];
        char sub_str[50];

        int i=0,n;
        printf("\n Enter a string: ");
        gets(str);
        printf("\nEnter no of characters to be copied: ");
        scanf("%d",&n);
        while(str[i]!='\0' && i<n)
        {
                sub_str[i]=str[i];
                i++;
        }
        sub_str[i]='\0';
        printf("\n The Substring is:");
        puts(sub_str);
        return 0;
}
```

Output:
Enter a string: Welcome to Cse
Enter no of characters to be copied :7
The Substring is: Welcome

9. Write a program to extract the first N characters (Sub String) of a string from Right.

```
#include<stdio.h>
#include<string.h>
int main()
{
        char str[50];
        char sub_str[50];

        int i=0,j=0,n;
        printf("\n Enter a string: ");
        gets(str);
        printf("\nEnter no of characters to be copied: ");
        scanf("%d",&n);
        j=strlen(str)-n;
        while(str[j]!='\0')
        {
                sub_str[i]=str[j];
                i++;
                j++;
        }
        sub_str[i]='\0';
        printf("\n The Substring is:");
        puts(sub_str);
        return 0;
}
```

Output:
Enter a string: Welcome to Cse
Enter no of characters to be copied :3
The Substring is:Cse

10. Write a program to extract the first N characters (Sub String) of a string from middle.

```
#include<stdio.h>
#include<string.h>
int main()
{
        char str[50];
        char sub_str[50];

        int start_pos ,i=0,m,n;
        printf("\n Enter a string: ");
        gets(str);
        printf("\n Enter the position from which to start the sub string :");
        scanf("%d"&start_pos);
        printf("\nEnter no of characters to be copied: ");
        scanf("%d",&n);

        while(str[start_pos]!='\0' && n>=0)
        {
                sub_str[i]=str[start_pos];
                i++;
                start_pos++;
                n--;
        }
        sub_str[i]='\0';
        printf("\n The Substring is:");
        puts(sub_str);
        return 0;
}
```

Output:
Enter a string: Welcome to Cse
Enter the position from to start of sub string: 8
Enter no of characters to be copied :2
The Substring is: to

3.11 Array of Strings

- ✓ String is nothing but array of characters.
- ✓ Array of strings is nothing but stores the multiple strings with single variable name. It is also known as two dimensional (2D) array.

Syntax :

```
char array_name [row-size][col_size];
```

Here row-size indicates no of strings, column-size indicates max size of a string.

Example:

char name[4][8]={“Ram”,”IIITN”,”Cse”,”Nuzvid”};

name[0]	R	a	m	\0				
name[1]	I	I	T	N	\0			
name[2]	C	s	e	\0				
name[3]	N	u	z	v	i	d	\0	

Memory representation of Array of string

Example: Write a program to read and print the students names.

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

        char name[4][8];
        while(i<4)
        {
                printf("\nEnter the student %d name: ",i+1);
                gets(name[i]);
                i++;
        }
        printf("\n Names of the students :");
        for(i=0;i<4;i++)
        {
                puts(name[i]);

        }
        return 0;
}
```

Output:

```
Enter the student 1 name : Ram
Enter the student 2 name : Krish
Enter the student 1 name : Sree
Enter the student 2 name : Sai
Names of the students:
Ram
Krish
Sree
Sai
```

3.0 Introduction

The fundamental data types, namely *char, int, float,* and *double* are used to store only one value at any given time. Hence these fundamental data types can handle limited amounts of data.

In some cases we need to handle large volume of data in terms of reading, processing and printing. To process such large amounts of data, we need a powerful data type that would facilitate efficient storing, accessing and manipulation of data items. C supports a derived data type known as ***array*** that can be used for such applications.

Arrays

- ✓ An array is a fixed size sequenced collection of elements of the same data type.
- ✓ It is simply grouping of like type data such as list of numbers, list of names etc.

Types of Arrays

We can use arrays to represent not only simple lists of values but also tables of data in two or three or more dimensions.

1. One – dimensional arrays(1-D)
2. Two – dimensional arrays (2-D)
3. Multidimensional arrays (n-D)

Some examples where arrays can be used are

1. List of temperatures recorded every hour in a day, or a month, or a year.
2. List of employees in an organization.
3. List of products and their cost sold by a store.
4. Test scores of a class of students
5. List of customers and their telephone numbers.

3.1 Array

An array is collection of homogeneous elements that are represented under a single variable name.

(Or)

An array is collection of same data type elements in a single entity.

- ✓ It allocates sequential memory locations.
- ✓ Individual values are called as elements.

Declaration of One-Dimensional Array

Like any other variables, arrays must be declared before they are used. The general form of array declaration is

Syntax

datatype array_name[size];

- ✓ The *data type* specifies the type of element that will be contained in the array, such as *int*, *float*, or *char* or any valid *data type*.
- ✓ The *array name* specifies the name of the array.
- ✓ The *size* indicates the maximum number of elements that can be stored inside the array.
- ✓ The *size* of array should be a *constant* value.

For example

```
int marks[10];                    //integer array
```

The above statement declares a *marks* variable to be an array containing 10 elements. In C, the array *index* (also known as *subscript*) start from *zero.* i.e. The first element will be stored

in *marks[0]*, second element in *marks[1]*, and so on. Therefore, the last element, that is the 10th element, will be stored in *marks[9]*.

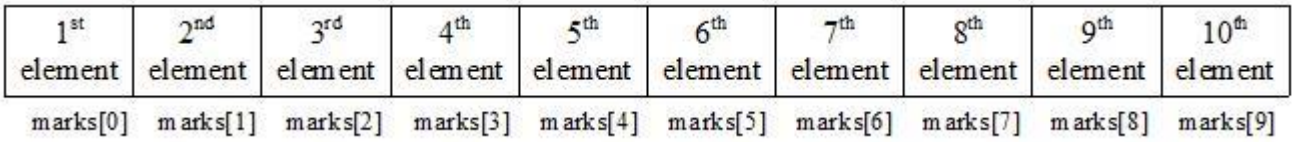


Fig. memory representation of an array of elements

Examples

float temp[24]; //floating-point array

Declare the group as an array to contain a maximum of 24 real constants.

char name[10]; //character array

Declare the name as a character array (string) variable that can hold a maximum of 10 characters.

C array indices start from 0. So for an array with N elements, the index that last element is N-1

Valid Statements

- ✓ a = marks[0] + 10;
- ✓ marks[4] = marks[0] + marks[2];
- ✓ marks[2] = x[5] + y[10];
- ✓ value[6] = marks[i] * 3;

Example 1: A C program that prints bytes reserved for various types of data and space required for storing them in memory using array.

```
#include<stdio.h>
int main()
{
   char c[10];
   int i[10];
   float f[10];
   double d[10];
   printf("\nThe type char requires %d byte",sizeof(char));
   printf("\nThe type int requires %d bytes",sizeof(int));
   printf("\nThe type float requires %d bytes",sizeof(float));
   printf("\nThe type double requires %d bytes",sizeof(double));
   printf("\n%d memory locations are reserved for 10 character elements", sizeof(c));
   printf("\n%d memory locations are reserved for 10 integer elements", sizeof(i));
   printf("\n%d memory locations are reserved for 10 float elements", sizeof(f));
   printf("\n%d memory locations are reserved for 10 double elements", sizeof(d));
    return 0;
}
```

Output

The type *char* requires 1 byte
The type *int* requires 2 bytes
The type *float* requires 4 bytes
The type *double* requires 8 bytes
10 memory locations are reserved for 10 character elements
20 memory locations are reserved for 10 integer elements
40 memory locations are reserved for 10 float elements
80 memory locations are reserved for 10 double elements

3.1.1 Accessing elements the array

To access all the elements from an array, we must use a loop. That is, we can access all the elements of an array by varying the value of the subscript into the array. But note that the subscript must be an integral value or an expression that evaluates to an integral value.

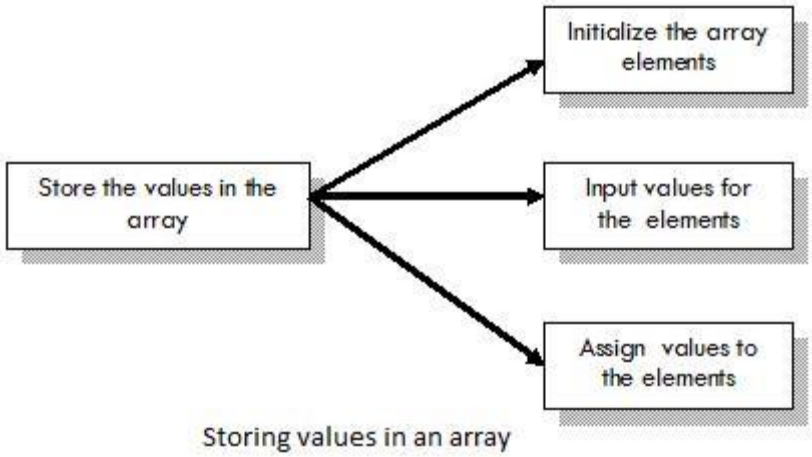
For example

```
int i ,marks[10];
for (i=0; i<10; i++)
        printf("%d",marks[i]);
```

3.1.2 Storing Values in Array

When we declare an array, we are just allocating space for its elements; no values are stored in the array. There are three ways to store values in an array.

1. Initialize the array elements during declaration.
2. Input values for individual elements from the keyboard.
3. Assign values to individual elements using assignment (=) operator.



Storing values in an array

Initializing Arrays during Declaration

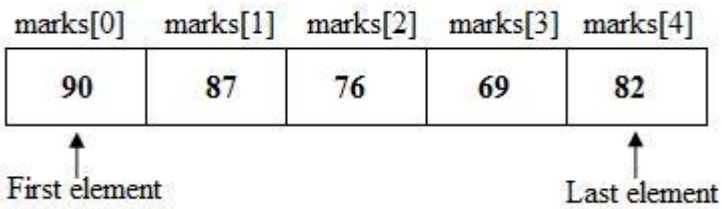
When an array is initialized, we need to provide a value for every element in the array. Arrays are initialized by writing:

type array_name[size]={list_of_values};

The *values* are written within curly brackets and every value is separated by a *comma*.

For example

```
int marks[5]={90,87,76,69,82};
```



While initializing the array at the time of declaration, the programmer may omit the size of the array. For example,

```
int marks[]= {98, 97, 90};
```

Input values from keyboard

An array can be filled by inputting values from the keyboard. In this method, a *while* or *for* loop is executed to input the value for each element of the array. For example:

```
int i ,marks[10];
for (i=0; i<10; i++)
        scanf("%d",&marks[i]);
```

Assigning Values to Individual Elements

The third way is to assign values to individual elements of the array by using the *assignment operator*. Any value that evaluates to the data type as that of the array can be assigned to the individual array element. A simple assignment statement can be written as

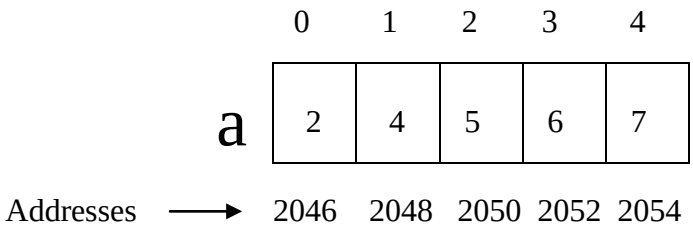
```
marks[3] = 100;
```

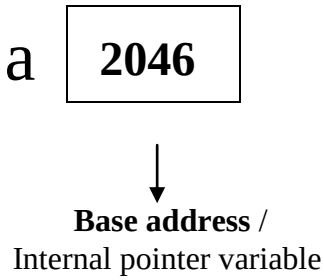
Here, 100 is assigned to the fourth element of the array which is specified as *marks[3]*.

Note:

1. Array variable always holds the base address only . It is also called internal pointer variable.
   Example:

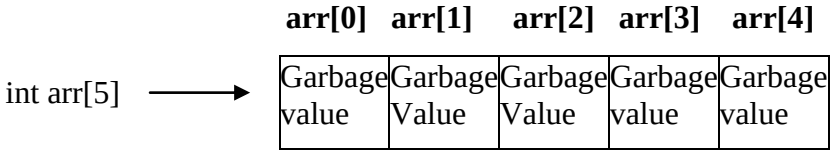
   int a[5]={2,4,5,6,7}

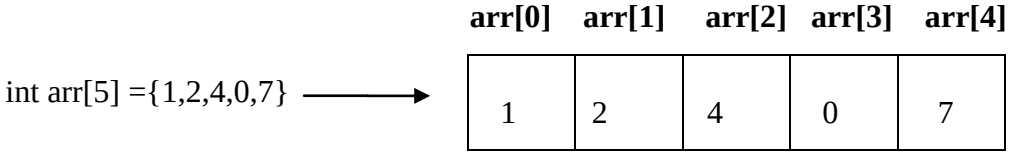




2. **Arrays behavior in Local and Global declaration:**

   - ✓ Declare a variable inside a function is know as **Local variable**.
   - ✓ If we declare array variable with in the function and we are not assign the values into array then stores the garbage values in the array.
     Example:





   - ✓ If we declare an array with size of five elements and we assign only two values into array then remaining elements stores with zero.

int arr[5] ={2,7} ⟶

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
2	7	0	0	0

- ✓ Declare a variable outside the function is know as **Global variable**.
- ✓ If we declare an array variable outside the function and we are not assign the values into array then the values stores with '0'. In global variable the default integer value is zero.

Example:

int arr[5] ⟶

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
0	0	0	0	0

Example 2 : Write a program to print local and global and initialized arrays values .

```
#include<stdio.h>
int g[5]; ⟶ // global variable declaration.
int main()
{
        int  L[5]; ⟶ // Local variable declaration
        int b[5]={10.20}; ⟶ // values assigned less than size of array.
        printf("\n Global values \t\t Local values \t\t  Initialized array variable values : ");
        for(int i=0;i<5;i++)
        printf("\t index[%d]=%d\t\t index[%d]=%d\t\t index[%d]=%d ", i, g[i],i,L[i],i,b[i]);
        printf("\n");
        return 0;
}
```

Output

Global values	Local values	Initialized array variable values
index[0] = 0	index[0] = garbage value	index[0] = 10
index[1] = 0	index[1] = garbage value	index[1] = 20
index[2] = 0	index[2] = garbage value	index[2] = 0
index[3] = 0	index[3] = garbage value	index[3] = 0
index[4] = 0	index[4] = garbage value	index[4] = 0

Example 3: Write a C Program to read and display *N* numbers using an array.

```
#include <stdio.h>
int main()
{
        int i, n, arr[20];
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n arr[%d] = ", i);
                scanf("%d",&arr[i]);
        }
        printf("\n The array elements are ");
        for(i=0;i<n;i++)
        printf("\t %d", arr[i]);
        return 0;
}
```

Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
The array elements are 1 2 3 4 5
```

Example 4: Write a C Program to print the maximum and minimum element in the array.

```
#include <stdio.h>
int main()
{
	int i, n, a[20], max, min;
	printf("Enter the number of elements in the array :");
	scanf("%d", &n);
	printf("Enter the elements\n");
	for(i=0;i<n;i++)
	{
		printf("a[%d] = ",i);
		scanf("%d",&a[i]);
	}
	max = min = a[0];
	for(i=1;i<n;i++)
	{
		if(a[i]<min)
			min = a[i];
		if(a[i]>max)
			max = a[i];
	}
	printf("\n The smallest element is : %d\n", min);
	printf("\n The largest element is : %d\n", max);
	return 0;
}
```

Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
The smallest element is : 1
The largest element is : 5
```

3.2 Searching the Array elements

Searching means to find whether a particular value is present in an array or not. If the value is present in the array, then searching is said to be *successful* and the searching process gives the location of that value in the array. if the value is not present in the array, the searching is said to be *unsuccessful*. There are two popular methods for searching the array elements:

1. Linear search
2. Binary search.

Linear Search

Linear search, also called as *sequential search*, is a very simple method used for searching an array for a particular value. It works by comparing the value to be searched with every element of the array one by one in a sequence until a match is found. It is mostly used to search an unordered list of elements. For example, if an array *a[]* is declared and initialized as,

int a[] = {10, 8, 2, 7, 3, 4, 9, 1, 6, 5};

and the value to be searched is *VAL = 7*, then searching means to find whether the value '7' is present in the array or not. If yes, then it returns the position of its occurrence. Here, POS = 3 (index starting from 0).

Example 5: Write a C Program to search an element in an array using the linear search technique.

```
#include <stdio.h>
int main()
{
   int a[100],n,i,key,flag=0;
   printf("Enter number of elements:");
   scanf("%d", &n);
   printf("Enter elements\n");
   /* Read array elements */
   for(i=0;i<n;i++)
   {
         printf("Enter a[%d]=",i);
         scanf("%d", &a[i]);
   }
   printf("Enter an element to be searched:");
   scanf("%d", &key);
   /* linear search starts here */
   for(i=0;i<n;i++)
   {
      if(key==a[i])
      {
printf("%d is found at position %d\n", key, i);
            flag=1;
        break;
      }
   }
   if(flag==0)
         printf("%d is not found\n",key);
   return 0;
}
```

Output

```
Enter number of elements: 5
Enter elements
Enter a[0] = 14
Enter a[1] = 5
Enter a[2] = 23
Enter a[3] = 9
Enter a[4] = 15
Enter an element to be searched: 9
9 is found at position 3
```

Binary Search

Binary search is a searching algorithm that works efficiently with a sorted list.

Example 6: Write a C Program to search an element in an array using binary search.

```
#include <stdio.h>
int main()
{
        int a[25],i,n,key,high,low,mid,flag=0;
        printf("\n Enter the number of elements in the array: ");
        scanf("%d", &n);
        printf("\n Enter the elements\n");
        for(i=0;i<n;i++)
        {
                printf("Enter a[%d]=",i);
                scanf("%d", &a[i]);
        }
        printf("Enter the element to be searched: ");
        scanf("%d", &key);
        low = 0, high = n-1;
        while(high>=low)
        {
                mid = (low + high)/2;
                if (a[mid] == key)
                {
                        printf("\n %d is found at position %d", key, mid);
                        flag = 1;
                        break;
                }
                 else if (a[mid]>key)
                         high = mid-1;
                else
                        low = mid+1;
        }
        if (flag == 0)
                printf("\n %d does not found in the array", key);
        return 0;
}
```

Output

```
Enter the number of elements in the array: 5
Enter the elements
Enter a[0] = 11
Enter a[1] = 26
Enter a[2] = 32
Enter a[3] = 49
Enter a[4] = 68
Enter the element to be searched: 49
49 is found at position 3
```

3.3 Sorting the Array elements

Sorting means arranging the elements of an array in specific order may be either ascending or descending. There are different types of sorting techniques are available:

1. Bubble sort
2. Selection sort
3. Insert sort
4. Merge sort
5. Quick sort etc.

Bubble sort, sometimes referred to as *sinking sort*, is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order.

Example 7: C Program to implement Bubble Sort Technique

```
#include <stdio.h>
int main()
{
   int a[100],n,i,j,temp;
   printf("Enter number of elements:");
   scanf("%d", &n);
   printf("Enter elements\n");
   /* Read array elements */
   for(i=0;i<n;i++)
   {
         printf("Enter a[%d]=",i);
         scanf("%d", &a[i]);
   }
   /* bubble sort logic starts from here */
   for(i=0;i<n-1;i++)
   {
    for(j=i+1;j<n;j++)
           if(a[i]>a[j])            //> for ascending order, < for descending order
           {
            temp=a[i];
            a[i] = a[j];
            a[j] = temp;
           }
   }
   printf("Sorted elements ....\n");
   for(i=0;i<n;i++)
     printf("%3d",a[i]);
   printf("\n");
   return 0;
}
```

Output

```
Enter number of elements: 5
Enter elements
Enter a[0] = 14
Enter a[1] = 5
Enter a[2] = 23
Enter a[3] = 9
Enter a[4] = 15
Sorted elements ....
5        9        14       15      23
```

Selection sort is a simple sorting algorithm. This sorting algorithm is in-place comparison based algorithm in which the list is divided into two parts, sorted part at left end and unsorted part at right end. Initially sorted part is empty and unsorted part is entire list.

Example 8: C program to implement *Selection sort* technique

```
#include <stdio.h>
int main()
{
   int a[100],n,i,j,temp;
   printf("Enter number of elements:");
   scanf("%d", &n);
   printf("Enter elements\n");
   /* Read array elements */
   for(i=0;i<n;i++) {
          printf("Enter a[%d]=",i);
          scanf("%d", &a[i]);
   }
   /* selection sort */
   for(i=0;i<n-1;i++)
   {
          min=i;
          for(j=i+1;j<n;j++)
          {
             if(a[j]<a[min])
                min=j;
          }
          temp=a[i];
           a[i] = a[min];
           a[min] = temp;
    }
    printf("Sorted elements ....\n");
    for(i=0;i<n;i++)
      printf("%d\t",a[i]);
    printf("\n");
    return 0;
}
```

Output

```
Enter number of elements: 5
Enter elements
Enter a[0] = 5
Enter a[1] = 3
Enter a[2] = 9
Enter a[3] = 7
Enter a[4] = 2
Sorted elements....
2       3       5       7       9
```

5.4 Two Dimensional (2D) Arrays

There could be situations where a table of values will have to be stored. Consider a student table with marks in 3 subjects.

Student	Maths	Physics	Chemistry
Student #1	89	77	84
Student #2	98	89	80
Student #3	75	70	82

- ✓ The above table contains a total of 9 values.
- ✓ We can think this table as a matrix consisting of 3 rows and 3 columns.
- ✓ Each row represents marks of student # 1 in all (different) subjects.
- ✓ Each column represents the subject wise marks of all students.
- ✓ In mathematics, we represent a particular value in a matrix by using two subscripts such as *Vij*.
- ✓ Here *V* denotes the entire matrix, *Vij* refers to the value in i^{th} row and j^{th} column.

In the above table *V23* refers to the value "80". C allows us to define such tables of items by using two-dimensional arrays.

Definition

A list of items can be represented with one variable name using two subscripts and such a variable is called a two – subscripted variable or a two – dimensional *(2D)* array.

5.4.1 Declaring 2-D Array

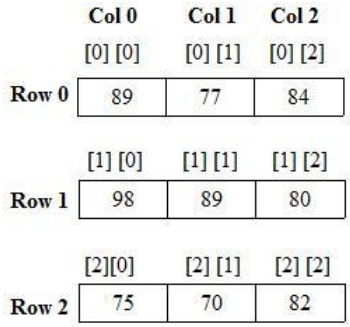
A two-dimensional array is declared as:

data_type array_name[row_size][column_size];

For example, if we want to store the marks obtained by three students in three different subjects, we can declare a two dimensional array as:

int marks[3][3];

The pictorial form of a two-dimensional array in memory is shown in Figure.



3.4.2 Initializing Two- Dimensional Arrays

Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in curly braces.

int table[2] [3] = {0,0,0,1,1,1};

This initializes the elements of first row to zero and the second row to one.

- ✓ This initialization can also be done row by row. The above statement can be equivalently written as

int table[2][3] = { {0,0,0}, {1,1,1} };

Commas are required after each curly brace that closes of a **row**, except in case of last **row**.

- ✓ If the values are missing in an initializer, they are automatically set to zero.

Ex: *int table [2] [3] = {*

{1,1}, \\ 1 1 0

{2} \\ 2 0 0

};

- ✓ When all the elements are to be initialized to zero, the following short-cut method may be used.

  int m[3][5] = { {0}, {0}, {0}};
- ✓ The first element of each row is explicitly initialized to zero while the other elements are automatically initialized to zero.
- ✓ The following statement will also achieve the same result

  int m[3][5] = { 0 };

3.4.2.1 Memory representation of 2D Array:

The two-dimensional array shows a rectangular picture, these elements will be actually stored sequentially in memory. Since computer memory basically one-dimensional, a multi-dimensional cannot be stored in memory as a grid.

There are two ways of storing a two-dimensional arrays in memory.

1. Row major order (RMO).
2. Column major order (CMO).

Row major order:

- ✓ The elements of the array are stored in row by row where n elements of the first row will occupy the first n locations.
- ✓ The elements of the first row are stored in before the elements of second and third row.

	Col 0	Col 1	Col 2
	[0] [0]	[0] [1]	[0] [2]
Row 0	89	77	84
	[1] [0]	[1] [1]	[1] [2]
Row 1	98	89	80
	[2][0]	[2] [1]	[2] [2]
Row 2	75	70	82

[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]	[2][0]	[2][1]	[2][2]
89	77	84	98	89	80	75	70	82

Fig: Elements of 3x3 2D array in RMO

Column major order:

- ✓ The elements of the array are stored in column by column where n elements of the first column will occupy the first n locations.
- ✓ The elements of the first column are stored in before the elements of second and third column.

[0][0]	[1][0]	[2][0]	[0][1]	[1][1]	[2][1]	[0][2]	[1][2]	[2][2]
89	98	75	77	89	70	84	80	82

Fig: Elements of 3x3 2D array in CMO

3.4.3 Accessing the Elements of Two-dimensional Arrays

Since the *2D* array contains two subscripts, we will use two for loops to scan the elements. The first *for* loop will scan each row in the 2D array and the second *for* loop will scan individual columns for every row in the array.

Example 9: Write a program to print the elements of a 2D array.

```
#include <stdio.h>
int main()
{
        int a[3][3] = {12, 34, 56,32,89,23,44,67,99};
        int i, j;
        for(i=0;i<3;i++)
        {

                for(j=0;j<3;j++)
                        printf("%d\t", a[i][j]);
                printf("\n");
        }
        return 0;
}
```

Output

```
12      34      56
32      89      23

44      67      99
```

3.4.4 Operations on 2D Arrays

Example 10: Write a C program to print the *transpose* of a given matrix

```
#inlude<stdio.h>
int main( )
{
int a[10][10].trans[10][10];
int rows,cols,i,j;
// read the size of matrix
Printf("enter no of rows & cols:");
scanf("%d%d", &rows,&cols);
// read the elements of the array
printf("enter the elements of matrix:");
   for(i=0; i<rows;i++)
     for(j=0; j<cols;j++)
     {
          printf("Enter a[%d][%d]=",i,j);
          scanf("%d",&a[i][j]);
     }
     /* Display the matrix A */
     printf("The Matrix A\n");
     for(i=0; i<rows;i++) {

     for(j=0; j<cols; j++)
           printf("%3d",a[i][j]);
     printf("\n");
   }
   /* Change rows into columns and columns into rows */
   for(i=0;i<cols;i++)
   {
         for(j=0;j<rows;j++)
            trans[i][j] = a[j][i];
            printf("%3d",a[j][i]);
```

```
        printf("\n");
   }
/* Display the transpose matrix */
   printf("Transpose of a given matrix\n");
   for(i=0;i<cols;i++)
   {
         for(j=0;j<rows;j++)
            printf("%3d",trans[i][j]);
         printf("\n");
   }
  return 0;
 }
```

Output

```
Enter number of rows and columns: 3 3 Enter elements of matrix Enter a[0][0]=4
Enter a[0][1]=7
Enter a[0][2]=3
Enter a[1][0]=9
Enter a[1][1]=5
Enter a[1][2]=2
Enter a[2][0]=6
Enter a[2][1]=1
Enter a[2][2]=8
The Matrix A
 4  7  3
 9  5  2
 6  1  8
Transpose of a given matrix
 4   9  6
 7   5  1
 3   2  8
```

Example 11: Write a C program to perform addition or subtraction of two matrices

```
#include <stdio.h>
int main()
{
   int a[10][10],b[10][10],c[10][10];
   int rows,cols,i,j;
   printf("Enter number of rows and columns (between 1 and 10):");
   scanf("%d%d",&rows,&cols);
   /* read the elements of first matrix */
   printf("Enter elements of first matrix\n");
   for(i=0;i<rows;i++)
   {
      for(j=0;j<cols;j++)
           scanf("%d",&a[i][j]);
   }
/* read the elements of second matrix */
   printf("Enter elements of second matrix\n");
   for(i=0;i<rows;i++)
   {
         for(j=0;j<cols;j++)
            scanf("%d",&b[i][j]);
     }
```

```
/* Display A and B matrices */
   printf("The Matrix A\n");
   for(i=0;i<rows;i++)
   {
     for(j=0;j<cols;j++)
         printf("%3d",a[i][j]);
     printf("\n");
   }
   printf("The Matrix B\n");
   for(i=0;i<rows;i++)
   {
     for(j=0;j<cols;j++)
         printf("%3d",b[i][j]);
     printf("\n");
   }
          /*Add two matrices and print resultant matrix */
  printf("The resultant matrix is\n");
  for(i=0;i<rows;i++)
  {
     for(j=0;j<cols;j++)
     {
            c[i][j]=a[i][j]+b[i][j];        // for subtract, use minus (-) instead of plus (+) sign
            printf("%3d",c[i][j]);
     }
     printf("\n");
  }

  return 0;
}
```

Output

```
Enter number of rows and columns (between 1 and 10): 3 3
Enter elements of first matrix: 1 2 3 4 5 6 7 8 9
Enter elements of second matrix: 1 2 3 4 5 6 7 8 9
The matrix A
1 2 3
4 5 6
7 8 9
The matrix B
1 2 3
4 5 6
7 8 9
The resultant matrix is
2    4    6
8    10 12
14   16  18
```

Example 12: Write a C program to perform multiplication of two matrices

```
#include <stdio.h>
int main()
{
  int a[10][10],b[10][10],c[10][10];

  int m,n,p,q,i,j,k;
  printf("Enter number of rows and columns of first matrix(between 1 and 10):");
  scanf("%d%d",&m,&n);
  printf("Enter number of rows and columns of second matrix(between 1 and 10):");
  scanf("%d%d",&p,&q);
  if(n==p)
  {
        /* Read the elements of first matrix */
        printf("Enter elements of first matrix\n");
        for(i=0;i<m;i++)
        {
             for(j=0;j<n;j++)
                  scanf("%d",&a[i][j]);
         }
        /* Read the elements of second matrix */
        Print f("Enter elements of second matrix\n");
        for(i=0;i<p;i++)
        {
            for(j=0;j<q;j++)
                  scanf("%d",&b[i][j]);
         }
    /* Display A and B matrices */
    printf("The Matrix A\n");
    for(i=0;i<m;i++)
    {
          for(j=0;j<n;j++)
             printf("%3d",a[i][j]);
          printf("\n");
    }
    printf("The Matrix B\n");
    for(i=0;i<p;i++)
    {
          for(j=0;j<q;j++)
            printf("%3d",b[i][j]);
          printf("\n");
    }
    /*multiply two matrices and print resultant matrix */
    printf("The resultant matrix is\n");
    for(i=0;i<m;i++)
    {
          for(j=0;j<q;j++)
          {
            c[i][j]=0;              /* Initializing resultant matrix elements to zero */
            for(k=0;k<p;k++)
```

```
                c[i][j] += a[i][k]*b[k][j];
            printf("%3d",c[i][j]);
           }

           printf("\n");

           }
  }
  else
     printf("Multiplication is not possible\n");
  return 0;
}
```

Output

```
Enter number of rows and columns of first matrix (between 1 and 10):2 3
Enter number of rows and columns of second matrix (between 1 and 10):3 2
Enter elements of first matrix: 1 2 3 4 5 6
Enter elements of second matrix: 1 2 3 4 5 6
The Matrix A
1  2  3
4  5  6
The Matrix B
1    2
3    4
5    6
The resultant matrix is
22    28
49    64
```

3.5 Multi – Dimensional Array

A list of items can be represented with one variable name using more than two subscripts and such a variable is called Multi – dimensional array.

Three Dimensional (3D) Array

A list of items can be represented with one variable name using three subscripts and such a variable is called Three – dimensional (3D) array.

Syntax

data type array_name [size_of_2d_matrix][row_ size][column_ size];

No of pages / No of tables

Initializing 3D Array

Like the one-dimensional arrays, three-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces.

int table[2][2][3] = {0,0,0,1,1,1,4,4,4,7,7,7};

This initializes the elements of first two dimensional (matrix) first row to zero's and the second row to one's and second matrix elements are first row to four's and the second row to seven's.

- ✓ This initialization is done row by row.
- ✓ The above statement can be equivalently written as

int table[2][2][3] = {{{0,0,0},{1,1,1}},{{4,4,4,},{7,7,7}}}

we can also initialize a two – dimensional array in the form of a matrix as shown.

```
int table[2][2][3] = {
                        {
                            {0,0,0},
                            {1,1,1}
                        },
                        {
                            {4,4,4},
                            {7,7,7}
                        }
};
```

Example:

The memory representation of 3D array in row major order (RMO) and column major order (CMO).

- ✓ A three dimensional array consist of pages or tables . each page in turn contains m rows and n columns.
- ✓ Lets int a[t][r][c];

  Here

  t ⟶ no of pages or table in array

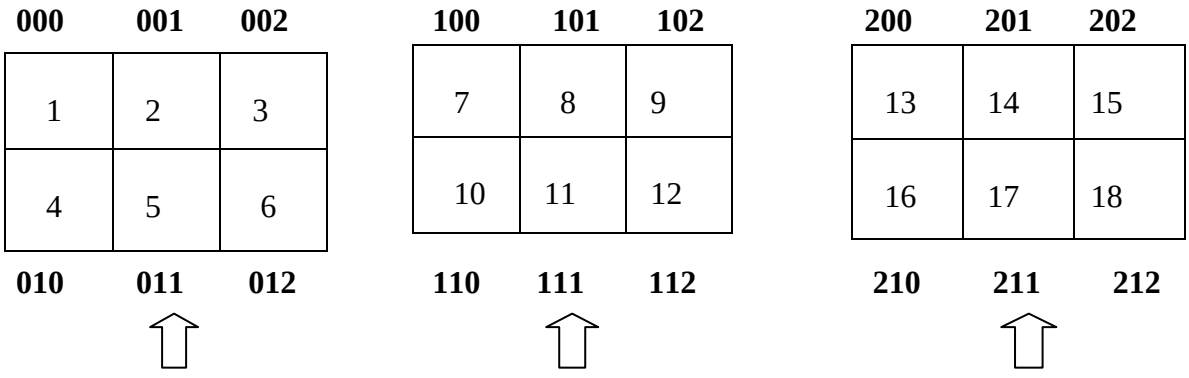
  r ⟶ no of rows in page/table

  c ⟶ no of columns in page/table.

```
int a[3][2][3]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18};
```

- ✓ The 3D array will contain 3x2x3=18 elements.
- ✓ Every page starts with index '0'.

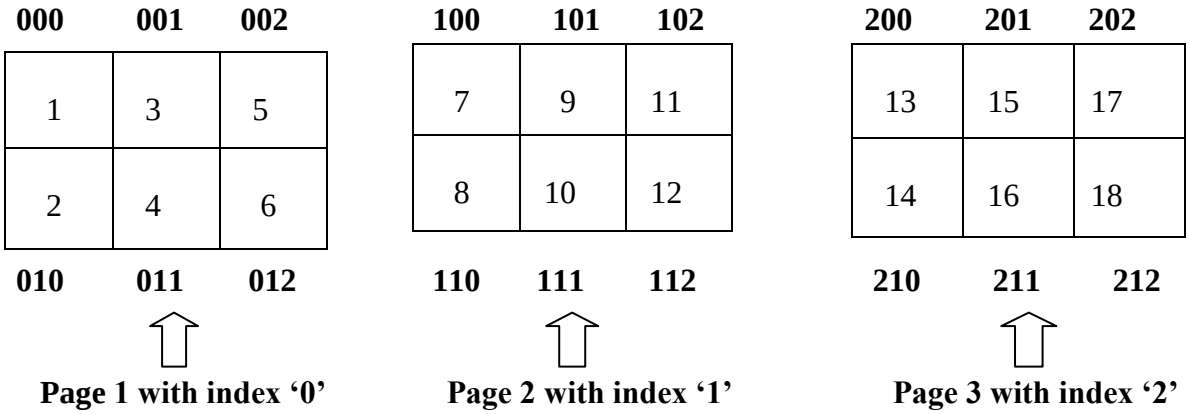
3D array elements stored in row major order.



000	001	002	010	011	012	101	101	102	110	111	112	200	201	202	210	211	212
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

3D array memory representation in Row Major Order

3D array elements stored in column major order.



000	010	001	011	002	012	100	110	101	111	102	112	200	210	201	211	202	212
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

3D array memory representation in Column Major Order

Example 13: Write a C program to read and display 3D array elements.

```
#include<stdio.h>
int main()
{
	int i,j,k,table,rows,cols;
	printf("Enter the no of tables:");
	scanf("%d",&table);
	printf("\nEnter the no of rows And columns:");
	scanf("%d%d",&rows,&cols);
	int arr[table][rows][cols];
	for(i=0;i<table;i++)
	{
		for(j=0;j<rows;j++)
		{
			for(k=0;k<cols;k++)
			{
				printf("\n arr[%d][%d][%d]=",i,j,k);
				scanf("%d",&arr[i][j][k]);
			}
		}
	}
	for(i=0;i<table;i++)
	{
		printf("Table %d  elements\n ",i+1);
		printf("------------------------\n");
		for(j=0;j<rows;j++)
		{
			for(k=0;k<cols;k++)
			{

				printf("%d \t",arr[i][j][k]);
```

```
                        }
                        printf("\n");
                }
        }
}
```

Output:

```
Enter no of  tables: 3
Enter  the no of rows and cols : 3   4
arr[0][0][0]=1
arr[0][0][1]=3
arr[0][0][2]=5
arr[0][0][3]=7
arr[0][1][0]=9
arr[0][1][1]=11
arr[0][1][2]=13
arr[0][1][3]=15
arr[0][2][0]=17
arr[0][2][1]= 19
arr[0][2][2]=21
arr[0][2][3]=23
arr[1][0][0]=2
arr[1][0][1]=4
arr[1][0][2]=6
arr[1][0][3]=8
arr[1][1][0]=10
arr[1][1][1]=12
arr[1][1][2]=14
arr[1][1][3]=16
arr[1][2][0]=18
arr[1][2][1]=20
arr[1][2][2]=22
arr[1][2][3]=24
arr[2][0][0]=2
arr[2][0][1]=3
arr[2][0][2]=5
arr[2][0][3]=7
arr[2][1][0]=11
arr[2][1][1]=13
arr[2][1][2]=17
arr[2][1][3]=19
arr[2][2][0]=23
arr[2][2][1]=29
arr[2][2][2]=31
arr[2][2][3]=37

Table 1 elements
---------------------------
1    3    5    7
9   11  13  15
17 19  21  23
```

Table 2 elements

```
------------------------
2     4     6     8
10  12   14   16
18  20   22   24
```

Table 3 elements

```
--------------------------
2     3     5     7
11   13   17    19
23   29   31    37
```