Unit - 2

Chapter 9

Strings

Chapter Outline

- What is a String?
- Declaring Strings
- Initialization of String Variables
- Reading String from Terminal
  - Using scanf() function
  - Using getchar() function
  - Using gets() function
- Writing Strings to Screen
  - Using printf() function
  - Using putchar () function
  - Using puts() function
- Arithmetic Operations on Characters
- Operations on Strings
  - Finding Length of a String using strlen( ) function
  - Copying a String using strcpy( ) function
  - Concatenation of Strings using strcat( ) function
  - Comparing Strings using strcmp( ) function
- Character Handling Functions
- Arrays of Strings
- Review questions

9.1 What is a String?

Numeric data are not the only data types that are processed on a computer. Very often the data to be processed are in textual form such as words, names, addresses etc. This type of data care stored and processed using string type variables. C does not have an explicit string data type. However, the same can be simulated using character array.

What is a String?

A string is a sequence of characters. 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 word 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.

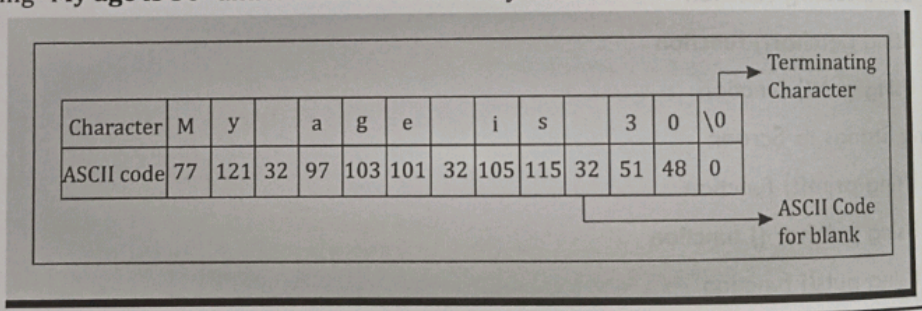
When we define a string as char s[10], the character s[10] is implicitly initialized with the null in the memory.

Examples: "My Life" "I Love C Programming"

Strings are stored in memory as ASCII codes of characters that make up the string appended with '\0' (ASCII value of null). Normally each character is stored in one byte, successive characters are stored in successive bytes.

Example

Consider a string "**My age is 30**" and it is stored in memory as shown below



Character	M	y		a	g	e		i	s		3	0	\0
ASCII code	77	121	32	97	103	101	32	105	115	32	51	48	0

Each character takes one byte and strings are stored in contiguous memory locations. The last character will be null character and it indicates the end of a string.

9.2 Declaring Strings

Variables of type **char** can hold only a single character, so they have limited usefulness. We also need a way to store *strings,* which are sequences of characters. A person's name and address are examples of strings. Although there is no special data type for strings, C handles this type of information with arrays of characters.

The general syntax for declaring a String is:

Syntax:

```
char string_name[size]
```

Where the ***size*** determines the number of characters in the string ***string_name.***

Example: `char name[10];  char city[20];`

Example

To hold a string of ten characters, we need to declare an array of type char with eleven elements. Arrays of type char are declared like arrays of other data types. For example, the statement

```
char str[5];
```

can hold a 5-character string, the fifth character (**str[4]**) holds Null character (the character written as: **\0**). Here, we have declared a array of 5 characters. The array is empty because we have just declared but not initialized.

str[0]	str[1]	str[2]	str[3]	str[4]

9.3 Initialization of String Variables

Like other C data types, character arrays can be initialized when they are declared. The character array or strings can be initialzed in so many ways.

1. **Character by Character Initialization:** Character arrays can be assigned values element by element, as shown here:

```
char str[10] = { 'S', 'R', 'I', 'K', 'A', 'N', 'T', 'H', '\0' };
```

If we assign character by character , we must specify the null character '\0' at the end of the string. The above code allocates 10 bytes of memory to store maximum of 10 characters. We have initialized an array with 9 characters including null character. It is stored in memory as shown below.

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]	str[9]
S	R	I	K	A	N	T	H	\0	
4001	4002	4003	4004	4005	4006	4007	4008	4009	4010

2. **Assigning Direct String to a Chracter Array (array size is mentioned):**

```
char str[6] = "INDIA";
```

To store "INDIA", string size 5 is enough but we should give extra one (+1) size 6 to store null character. In general to store N character string, we should create character array with size N+1.

Note: *If we assign string directly with in double quotes, no need to bother about null character. Because compiler will automatically assign the null character at the end of string.*

It is stored in memory as shown below:

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]
I	N	D	I	A	\0
4001	4002	4003	4004	4005	4006

3. Assigning Direct String to a Character Array (array size is not mentioned)

If we don't specify the number of subscripts when we declare an array, the compiler calculates the size of the array for us. Thus, the following line creates and initializes a Nine-element array.

```
char str[] = "SRIKANTH";
```

The way the string is stored in memory can be understood from the following figure.

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]
S	R	I	K	A	N	T	H	\0
4001	4002	4003	4004	4005	4006	4007	4008	4009

Note

If we assign string directly with in double quotes, no need to bother about null character. Because compiler will automatically assign the null character at the end of string.

If we assign character by character , we must specify the null character '\0' at the end of the string.

Program 1 **To display the name of a person using strings by reading each character of a character array.**

```
#include<stdio.h>
int main()
{
      /* define & initializing a string of characters */
      char string[10]="SRIKANTH";
      int i =0;

     while(string[i] !='\0')
     {
        printf(" String name is %c\n", name[i]);
        i++;
     }
}
```

Output

```
String name is SRIKANTH
```

9.4 Reading String from Terminal

9.4.1 Using scanf() function

The scanf() function reads the sequence of characters until it encounters white-space (space, new tab, etc.). The function scanf with %s format specification is needed to read the character s from the terminal. The %s is used to read in text, but only until the first white space charac encountered. So a space or a tab or the Enter key terminates the string.

Example

```
char    city(10);
scanf("%s", City)
```

Format specifier — Name of the character array (& not required)

Suppose if we input "BANGALORE", then storage looks like this

B	A	N	G	A	L	O	R	E	\0

The scanf() function has a drawback it just terminates as soon as it finds a blank space, suppose if we type "NEW DELHI", then only the string "NEW" will be read and since there is a blank space after "NEW" it will terminate the string. The storage looks like this.

N	E	W	\0	?	?	?	?	?	?

Program 2 **Write a program to read a string using scanf ( )**

```
# include <stdio.h>
void main ( )
{
   char string1[20], string2[20], string3[20];
   printf ("\n Enter First String :");
   scanf ("%s", string1);
   printf ("\n Enter Second String:");
   scanf ("%s", string2);
   printf ("\n Enter Third String:");
   scanf ("%s", string3);
  printf("\n  First String is : %s", string1);
  printf ("\n Second String is : %s", string2);
  printf ("\n Third String is : %s", string3);
}
```

Output

```
Enter First String: Skyward
Enter Second String: C Book
Enter Third String: My World
First String is : Skyward
Second String is : C
Third String is : My
```

Explanation

In second scanf, even though C Books was entered, only "C" is assigned to string1. Similarly in third scanf, we have entered "My World", only 'My" is assigned is string3. It's because there was a space between the words. The problem with the scanf function is that it terminates its input on the first white space it finds. A white space includes blanks,tabs,carriage returns,form feeds and new lines.

So, %s specifier cannot be used to read a string with white spaces. But this can be done with help of %[] specification. The format specifier "%[^\n]" tells to the compiler that read the characters until "\n" is not found. [Refer Chapter 3 for more details]

9.4.2 Using getchar() function

We know that getchar( ) is used to read a single character from the terminal. We can use getchar( ) function repeatedly to read successive single characters from the terminal and place them into a character array. In this way, an entire line can be read and stored in an array. The reading characters from the terminal will be terminated by pressing the Enter Key (new line character \n) and null character should be inserted at the end of the string.

Program 3 Write a program to read a string using getchar ( )

```
# include <stdio.h>
main ( )
{
      char string [40], ch;
      int i = 0;
      printf (" Enter a String. Press Enter key to terminate \n");
      do
      {
         ch = getchar( );
         string[i] = ch;
         i++;
      } while (ch! = '\n');

      string [i-1] = '\0';
      printf ("\n The String is % s", string);
}
```

Output

```
Enter a String. Press Enter key to terminate
MY AGE IS 30
The string is MY AGE IS 30
```

Explanation

getchar() function reads one character at a time. We can use getchar function inside a loop to read characters one by one till we don't read newline character (\n). Once we read newline character we break loop and add '\0' character at the end of string.

9.4.3 Using gets() function

The simplest and convenient method of reading a string is to use gets( ) function. This function is a library function and it is available in the <stdio.h> header file.

The gets() function receives a sequence of characters i.e., a string entered at the keyboard and store them in a variable (essentially as Array of type char) mentioned with it.

The gets() accepts single or multiple characters of string including spaces from the standard Input device.

The general syntax of gets() is shown below:

Syntax

```
gets (Character array variable);
```

Where character array varaible is a valid C variable declared as an array of character type.

The gets() function reads character from the standard input stream until a new line character ('\n') is read. The newline character is suppressed and a null ('\0') character is, then, appended in the end. This string is, then, stored in the memory address provided to by argument. That is why a string in C is also called as a character array terminated by a null character ('\0').

Example

```
char string[40];
get (string);
```

It reads characters from the keyboard until a new line character is encountered and then appends a null character to the string automatically.

Program 4 **Write a program to read a string using gets( ) and display using printf()**

```
# include <stdio.h>
main ( )
{
    char name[40], collegeName[40];
    printf("\n Enter your Name :");
    gets (name);
    printf("\n Enter your College Name : ");
    gets(collegeName);
    printf ("\n Name is : %s", name);
    printf ("\n College Name is  : %s", collegeName);
}
```

Output

```
Enter your Name: Indumathi
Enter your College Name: Jain College
Name is  :  Indumathi
College Name is :  Jain College
```

Note

- If we declare character array with size 40, then make sure that we read characters up to 40 only. Because C does not check array-bounds.
- C does not provide operators that work on strings directly.
- We cannot assign one string to another string directly.

Example

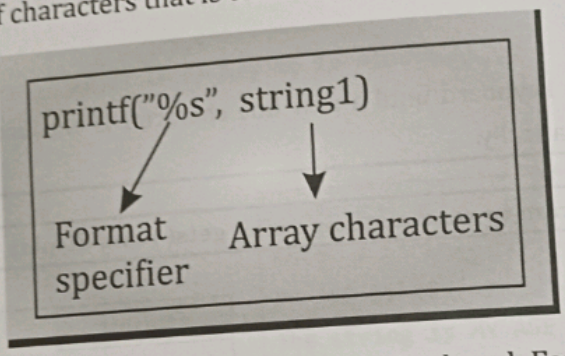
```
char string1[10]= "SNIGDHA";
char string2[10];
string2= string1;    // Error.
```

If we want to copy string1 to string2, then we should copy character by character only or use built-in str
functions like strcpy().

9.5 Writing Strings to Screen

9.5.1 Using printf() function

The **printf( )** function with **%s** format can be used to print strings to the screen. The format %s ca
be used to display an array of characters that is terminated by the null character.



We can also specify the precision with which the string is displayed, For example, **%15.5s** indicates
that first 5 characters are to be printed in a field width of **15** columns. Refer chapter3 for more details
on various formats of printing strings.

9.5.2 Using putchar () function

We already know that **putchar( )** function can be used to print only one character to the screen. We can use **putchar( )** function repeatedly to output a string of characters stored in an array using a loop.

Example

Consider the below code

```
char string[8]="SKYWARD";
int i=0;
for (i=0; i<7; i++)
   putchar (string[i]);
putchar ('\n');
```

In the above code, for loop executes 7 times and each time it will print one character using putchar().

9.3.3 Using puts() function

Like gets(), the most simplest and convenient way of printing array of characters is to use **puts()** function. This function is available in <stdio.h> header file. The puts() function, in contrast with gets(), writes a sequence of characters i.e., a string on the monitor .

The general syntax of puts() is shown below:

Syntax

```
puts (Character array variable);
```

Where character array variable is a valid C variable declared as an array of character type.

Example

Consider the below code

```
char str[40];
printf("\n  Enter your Name: ");
gets(str);
printf("\n Your Name is: ");
puts(str);
```

In the above code, string is accepted through the keyboard using gets() and same is printed using puts() function. The above puts() statement will display a string on the monitor whatever is stored inside 'str' and will also causes the cursor to be moved in the next line.

Program 5 **Write a program to read a string using gets( ) and display using puts()**

```
# include <stdio.h>
main ( )
{
    char name[40], collegeName[40];
    printf("\n Enter your Name : ");
    gets (name);
    printf("\n Enter your College Name : ");
    gets(collegeName);
    printf ("\n Name is  : ");
    puts(name);
    printf ("\n College Name is  : ");
    puts(collegeName);
}
```

Output

```
Enter your Name : Indumathi
Enter your College Name : Jain College
Name is  : Indumathi
College Name is  : Jain College
```

9.6 Arithmetic Operations on Characters

We can also manipulate the characters as we manipulate numbers in C language. Whenever the system encounters the character data it is automatically converted into a integer value by the system. We can represent a character as a integer by using the following method.

```
char ch = 'A';
printf ("%c", ch);  ⟶  Prints A
printf ("%d", ch);  ⟶  Prints ASCII Code of A. i.e., it
prints 65.
```

By using %c format specifier, it prints the character and by using %d format specifier, it prints the ASCII value of character A (which is 65). The arithmetic operations on characters are shown below.

1. **Subtracting a number from a character:**

   Consider the below code.

```
char ch;
ch='b' - 1;
printf("%c",ch);  ⟶  prints 'a'
printf("%d",ch);  ⟶  prints 97
```

   We know that ASCII value of 'b' is 98 and subtracting one from 98 gives 97 and this will be assigned to character variable ch. By using %c, it prints 'a' and by using %d, it prints the ASCII value of 'a' and which is 97.

2. **Adding a number to character**

   Consider the below code.

```
int i=20;
i = i + 'A';
printf("%d",i);  ⟶  prints 85
```

   The above code adds the ASCII value of 'A' (which is 65) to the value of 'i' (which is 20) and result 85 is stored in integer variable 'i' , It prints 85 to the screen.

3. **Converting string to a number**

   To convert string to a number, there exits library function atoi ( ), using which we can convert string to a number. Consider the below code.

```
char string[5] = "12345";
int num;
num = atoi(string);
```

   In the above code, string is a character array which is assigned the string constant "12345". The function **atoi** converts the string "12345" to its numeric equivalent 12345 and assigns it to the integer variable **num**.

9.7 Operations on Strings

We can perform different types of string operations manually like: finding length of string, concatenating(joining) two strings etc. Some of the common string operations are shown below.

1. Finding the length of a string
2. Concatenating two strings
3. Copying a string to another string
4. Comparing two strings
5. Extracting portion of a string
6. Converting lowercase to uppercase
7. Converting uppercase to lowercase
8. Reversing a string

All string manipulation can be done manually by the programmer but, this makes programming complex and large. To solve this, the C supports a large number of string handling functions. There are numerous functions defined in **"string.h"** header file.

The C Standard Library commonly provides a number of very useful functions which handle strings. Here is a short list of some common ones. A good compiler will support a lot more than those listed below, but, again, it really depends upon the compiler.

strcat(s1,s2)	concatenates the second string to the first
int strcmpi(s1,s2)	compares two strings, without case sensitivity
strcpy(s1,s2)	copy a string to the target string
strstr(s1,s2)	scan a string for the first occurrence of a substring
int strlen(s1)	returns the length of a string
strncpy()	is like strcpy, but limits the copy to no more than n characters.
strncmp()	is like strcmp, but limits the comparison to no more than n characters.
strchr and strrchr	similar to strstr, but instead they search for characters
strdup()	similar to strcpy(), except that strdup() performs its own memory allocation for the destination string with a call to malloc()
strlwr()	converts all the letter characters in str from uppercase to lowercase
strupr()	converting all the characters in str to uppercase
strrev()	reverses the order of all the characters in a string

Few commonly used string handling functions are discussed below:

Function	Work of Function
strlen()	Calculates the length of string
strcpy()	Copies a string to another string
strcat()	Concatenates(joins) two strings
strcmp()	Compares two string

Note : When using functions from the string-handling library, include the <string.h> header

9.7.1 Finding Length of a String using strlen( ) function

Some times, we need to know the *length of a string* (the number of characters between the start and the end of the string). This length is obtained with the library function ***strlen()***. Its prototype will be in ***<string.h>***.

This function returns a type int value, which gives the length or number of characters in a string, not including the NULL character.

Example

```
int len;
char *str; (or) char str[20];
len = strlen(str);
```

So the function strlen() returns an *integer*. The argument passed to strlen is a pointer to the string of which we want to know the length or name of character array. The function strlen() returns the number of characters between str and the next null character, not counting the null character.

Program 6 Finding the string length using strlen( ) function

```
/* Using the strlen() function. */
#include <stdio.h>
#include <string.h>
void main()
{
     int length;
     char str[80];
     while(1)
     {
       puts("\nEnter a line of text; a blank line terminates.");
       gets(str);
       length = strlen(str);
       if (length != 0)
         printf("\nThe length of the string is : %d ", length);
       else
         break;
     }
}
```

Output

```
Enter a line of text; a blank line terminates.
Adarsh College
The length of the string is : 14
Enter a line of text; a blank line terminates
```

Note: strlen() calculates the length of a string up to, but not including, the terminat aracter.

9.7.2 Copying a String using strcpy( ) function

The function ***strcpy()*** copies the string (including the terminating null character \0) pointed to by source to the location pointed to by destination. The return value is a pointer to the new string, destination.

When using strcpy(), we must first allocate storage space for the destination string. The function has no way of knowing whether destination points to allocated space. If space hasn't been allocated, the function overwrites strlen(source) bytes of memory, starting at destination; this can cause unpredictable problems.

One of the uses for strcpy is *reassigning* values to string variables defined as char arrays.

Example

```
char source[20]="ADARSH";
char dest[20];
strcpy (dest,source);
```

Now, the dest array contains the string "ADARSH"

The use of strcpy() is illustrated in the following program using the strcpy() function.

Program 7 **To copy a string from the source to destination using strcpy() function**

```
/* Demonstrates strcpy(). */
#include <stdio.h>
#include <string.h>

void main()
{
    char source[20];
    char dest[20];
    printf(" Enter the string : \n");
    gets(source);
    strcpy(dest, source);
    printf("\nThe String in destination is: \n %s", dest);
}
```

Output

```
Enter the string:
Bhagyalaksmi
The String in destination is:
Bhagyalaksmi
```

9.7.3 Concatenation of Strings using strcat( ) function

strcat is the C library functions that perform string CONCATENATION. i.e. This operation is used for *appending* the strings.

Example

```
char str1[20]="Rama";
char str2[10]="Sita";                    /* Now str1 contains RamaSita */
strcat(str1,str2);
```

The function appends a copy of str2 onto the end of str1, moving the terminating null character to the end of the new string. We must allocate enough space for str1 to hold the resulting string. The return value of strcat() is a pointer to str1.

Program 8 To concatenate strings using *strcat()* function

```
#include<stdio.h>
#include<string.h>
void main()
{
    char str1[40];
    char str2[20];

    printf("Enter the first string\n");
    gets(str1);
    printf("Enter the second string\n");
    gets(str2);

    /*str2 is appended to str1 and the result is stored in str1*/
    strcat(str1,str2);

    printf("The concatenated string is : \n%s",str1);
}
```

Output

```
Enter the first string
Covid
Enter the second string
Virus
The concatenated string
is: CovidVirus.
```

9.7.4 Comparing Strings using strcmp( ) function

This function is used for comparing two strings and return a value which indicates how they compared

Example

```
int value;
char s1[20], s2[20];
value = strcmp(s1,s2);
```

This function indicates
the value is > 0, if s1 <
in the ASCII code sequ

Note

strcmp performs a case

Program 9 To illu

```
#include<stdio.h>
#include<string.h>
void main()
{
    char str1[20
    char str2[20
    int i;
    printf("Ente
    gets(str1);
    printf("Ente
    gets(str2);
    i=strcmp(str
    if(i == 0)
        printf("
    else
        printf("
}
```

9.8 Charac

As we know that
constant is repres
For example, a is
'\n' represents th

Return Value	Meaning
< 0	str1 is less than str2.
0	str1 is equal to str2.
> 0	str1 is greater than str2.

The function indicates the (ASCII) alphabetical order of the two strings. If **s1 > s2** alphabetically, then the value is **> 0**. If **s1 < s2** alphabetically, then the value is **< 0**. Note that numbers come before letters in the ASCII code sequence and also that upper case comes before lower case.

Note

strcmp performs a case sensitive comparison,

Program 9 To illustrate strcmp( ) function

```
#include<stdio.h>
#include<string.h>
void main()
{
  char str1[20];
  char str2[20];
  int i;
  printf("Enter the first string\n");
  gets(str1);
  printf("Enter the second string\n");
  gets(str2);
  i=strcmp(str1,str2);/*str2 is compared with str1 and some value is returned based on that*/
  if(i == 0)
    printf("The two strings are equal",i);
  else
    printf("The strings are not equal",i);
}
```

Output

```
Enter the first string
Karnataka
Enter the second string
Karnataka
The two strings are equal
```

9.8 Character Handling Functions

As we know that characters are the fundamental building blocks of every program. A character constant is represented as a character in a single quote. The value of character constant is an integer. For example, a is represented as 'a' which is actually the integer value equal to 97 in ASCII. Similarly, '\n' represents the integer value of newline equal to 10 in ASCII.

There are various character handling functions in C which provide facility to handle character in different ways. C provides **<ctype.h>** header file for character handling. All character handling functions have defined in this header file. "ctype" means character type. We should include <ctype.h> in C program to use any of the character handling functions.

The character-handling library (<ctype.h>) includes several functions that perform useful tests and manipulations of character data. Each function receives a character represented as an int or EOF as an argument. As we discussed earlier characters are often manipulated as integers, because a character in C is usually a 1-byte integer. So the character-handling functions manipulate characters as integers.

Character Handling Functions

Function Prototype	Function Description	Example

int isdigit(int c);	This function is used to check whether a character is digit or not.
	It returns a true value if c is a digit and 0 (false) otherwise.
		printf("%d",isdigit('1')); //returns true
		printf("%d",isdigit('A')); //returns 0;

int isalpha(int c);	This function is used to check whether a character is alphabet or not.
	It returns a true value if c is a letter and 0 otherwise.
		printf("%d",isalpha('x')); // returns true
		printf("%d",isalpha('$')); // returns 0

int isalnum(int c);	This function is used to check whether a character is a letter or a digit i.e alphanumeric or not.
	It returns a true value if c is a digit or a letter and 0 otherwise.
		printf("%d",isalnum('8')); // returns true
		printf("%d",isalnum(';')); // returns 0

int islower(int c);	This function is used to check whether a character is lowercase alphabet or not.
	It returns a true value if c is a lowercase letter and 0 otherwise.
		printf("%d",islower('a')); // returns true
		printf("%d",islower('A')); // returns 0

int isupper(int c);	This function is used to check whether a character is uppercase or not.
	It returns a true value if c is an uppercase letter and 0 otherwise.
		printf("%d",isupper('B')); // returns true
		printf("%d",isupper('f')); // returns 0

int tolower(int c);	This function is used to convert uppercase character to lowercase character.
	If c is an uppercase letter, tolower() returns c as a lowercase letter. Otherwise, tolower() returns the argument unchanged.
		printf("%c",tolower('B')); // prints b
		printf("%c",tolower('n')); // prints n

int toupper(int c);	This fu lowerca charact If c is return Other argu
int isspace(int c);	This a ch not. It re spa (('\ tab
int ispunct(int c);	Th ch p

Program 10 Progr

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

void main()
{
  char ch;
  printf("\n En
  ch=getchar()

  if(isalpha(c
    printf("
  if(isdigit(
    printf("
  if(isupper
    printf(
  if(islowe
    printf
```

int toupper(int c);	This function is used to convert lowercase character to uppercase character.
	If c is a lowercase letter, toupper() returns c as an uppercase letter. Otherwise, toupper() returns the argument unchanged.
		printf("%c",toupper('m')); // prints M
		printf("%c",toupper('A')); // prints A

int isspace(int c);	This function is used to check whether a character is white space (blank) or not.
	It returns a true value if c is a white-space character—newline ('\n'), space (' '), form feed ('\f'), carriage return ('\r'), horizontal tab ('\t') or vertical tab ('\v') and 0 otherwise.
		printf("%d",isspace('\n')); // returns true
		printf("%d",isspace('X')); // returns 0

int ispunct(int c);	This function checks whether the character is a punctuator or not. Punctuators are comma, semicolon etc.
	It returns a true value if c is a punctuator character other than a space, a digit, or a letter and returns 0 otherwise.
		printf("%d",ispunct(';')); // returns true
		printf("%d",ispunct('9')); // returns 0

Program 10 **Program to demonstrate the use of character handling functions.**

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

void main()

{
 char ch;
 printf("\n Enter a character = ");
 ch=getchar();

 if(isalpha(ch))
  printf("\n You typed an alphabet");
 if(isdigit(ch))
  printf("\n You typed a digit");
 if(isupper(ch))
  printf("\n You typed uppercase letter");
 if(islower(ch))
  printf("\n You typed lowercase letter");
```

Output

```
Enter a character = 9
You typed a digit

Enter a character = A
You typed an alphabet
You typed uppercase letter

Enter a character = x
You typed an alphabet
You typed lowercase letter

Enter a character = ;
You typed punctuator

Enter a character =
You typed a white space
```

int toupper(int c);	This function is used to convert lowercase character to uppercase character.
	If c is a lowercase letter, toupper() returns c as an uppercase letter. Otherwise, toupper() returns the argument unchanged.
		printf("%c",toupper('m')); // prints M
		printf("%c",toupper('A')); // prints A

int isspace(int c);	This function is used to check whether a character is white space (blank) or not.
	It returns a true value if c is a white-space character—newline ('\n'), space (' '), form feed ('\f'), carriage return ('\r'), horizontal tab ('\t') or vertical tab ('\v') and 0 otherwise.
		printf("%d",isspace('\n')); // returns true
		printf("%d",isspace('X')); // returns 0

int ispunct(int c);	This function checks whether the character is a punctuator or not. Punctuators are comma, semicolon etc.
	It returns a true value if c is a punctuator character other than a space, a digit, or a letter and returns 0 otherwise.
		printf("%d",ispunct(';')); // returns true
		printf("%d",ispunct('9')); // returns 0

Program 10 **Program to demonstrate the use of character handling functions.**

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

void main()
{
  char ch;
  printf("\n Enter a character = ");
  ch=getchar();

  if(isalpha(ch))
    printf("\n You typed an alphabet");
  if(isdigit(ch))
    printf("\n You typed a digit");
  if(isupper(ch))
    printf("\n You typed uppercase letter");
  if(islower(ch))
    printf("\n You typed lowercase letter");
```

Output

```
Enter a character = 9
You typed a digit

Enter a character = A
You typed an alphabet
You typed uppercase letter

Enter a character = x
You typed an alphabet
You typed lowercase letter

Enter a character = ;
You typed punctuator

Enter a character =
You typed a white space
```

```
    if(isspace(ch))
      printf("\n You typed a white space");
    if(ispunct(ch))
      printf("\n You typed punctuator");
}
```

Program 11 Program to read a character and convert into uppercase if it is lowercase otherwise convert into lowercase.

```
#include<ctype.h>
#include<stdio.h>
void main()
{
    char ch;
    printf("\n Enter an Alphabet = ");
    ch=getchar();
    if(islower(ch))
      ch=toupper(ch);
    else
      ch=tolower(ch);
    printf("\n Now Alphabet is = %c",ch);
}
```

Output

```
Enter an Alphabet = x
Now Alphabet is = X

Enter an Alphabet = A
Now Alphabet is = a
```

Program 12 Program to read a string and find out the number of alphabets, digits and spaces in a string.

```
#include<ctype.h>
#include<stdio.h>
void main()
{
    int i, count_alpha = 0, count_digit = 0, count_spaces=0;
    char ch[40];
    printf("Enter a String : \n");
    gets(ch);
    for (i=0; ch[i]!='\0'; i++)
    {
      // To check the character is alphabet
      if (isalpha(ch[i]))
        count_alpha++;
      // To check the character is a digit
      if (isdigit(ch[i]))
        count_digit++;
      // To check the character is a space
      if (isspace(ch[i]))
        count_spaces++;
    }
    printf("\n The number of alphabets are : %d",count_alpha);
    printf("\n The number of digits are : %d",count_digit);
    printf("\n The number of spaces are : %d",count_spaces);
}
```

Output

```
Enter a String :
Skyward Publishers Bangalore 560018
The number of alphabets are : 26
The number of digits are : 6
The number of spaces are : 3
```

9.9 Arrays of Strings

A string is an array of characters; so, an array of strings is an array of arrays of characters. Of course, the maximum size is the same for all the strings stored in a two dimensional array.

We can declare a two dimensional character array of MAX strings of size SIZE as follows:

```
char names[MAX][SIZE];
```

Since names is an array of character arrays, names[i] is the first character array, i.e. it points to the first character array or string, and may be used as a string of maximum size SIZE - 1.

As usual with strings, a NULL character must terminate each character string in the array.

Example

```
char names[5][10];
```

Here, we can store 5 names of max length 9 characters (last one is NULL character).

We can think of an array of strings as a table of strings, where each row of the table is a string .

Program 13 To illustrate the array of strings

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

void main()
{
    char names[5][100];
    int i;
    /* Input the names from the keyboard */
    for(i=0;i<5;i++)
    {
       printf("\n Enter the name %d :",i+1);
       scanf("%s",names[i]);
    }
    /* Print the names */
    for(i=0;i<5;i++)
        printf("\n The name %d is %s",i+1,names[i]);
}
```

Output

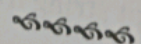
```
Enter the name 1: Rakshith
Enter the name 2: Snigdha
Enter the name 3: Smayan
Enter the name 4: Saatvik
Enter the name 5: Bhagya

The name 1 is Rakshith
The name 2 is Snigdha
The name 3 is Smayan
The name 4 is Saatvik
The name 5 is Bhagya
```

9.10 Review Questions

1. What is a string?
2. Which character is terminating character of strings?
3. Write the general syntax of declaring a string?
4. Give an example of declaring a string.
5. Give an example of initializing a string.
6. What are the different ways of reading strings from terminal?
7. Which format specifier is used to accept string from terminal?

8. What is the drawback of reading string using scanf( ) ?
9. Which method is convenient method of reading string from terminal?
10. What are the different ways of writing strings to screen?
11. What are the arithmetic operations possible on characters?
12. Which function is used to convert string to number?
13. What is concatenation?
14. Mention any four standard string library functions?
15. What is the use of strlen ( )?
16. What is the use of strcat( )?
17. What is the use of strcmp( ) ?
18. Which headerfile should be included for standard string handling functions?
19. Which headerfile should be included for standard character handling functions?
20. Explain isdigit( ) function with an example.
21. Explain isalpha( ) function with an example.
22. Explain isspace( ) function with an example.
23. Explain islower( ) function with an example.
24. Explain isupper( ) function with an example.
25. Explain isalnum( ) function with an example.
26. Explain ispunct( ) function with an example.
27. What is the use of tolower( )?
28. What is the use of toupper ( )?
29. How strings are stored in memory? Explain with an example.
30. What are the different ways of initializing a string variable?
31. Write a program to read a string from terminal using scanf ( ) function?
32. Write a program to read a string from terminal using getchar( )?
33. How strings are displayed in screen?
34. Write a program to print a string using printf( ).
35. Write a program to print a string using putchar( ).
36. Explain any four standard string library functions with an example.
37. Write a program to find the length of a string using strlen( ).
38. Write a program to concatenate two strings using strcat( ).
39. Write a program to copy one string to another string using strcpy( ).
40. Write a program to compare two strings using strcmp( ).
41. Explain operations on strings.
42. Explain arrays of strings with an example.
43. Explain any 4 character handling functions with an example.
44. Write a program to count the number of alphabets, digits and spaces in a string.



chapter 10