1. Introduction to strings

STRINGS IN C

- A string is a sequence/array of characters.
- C has no native string type; instead we use arrays of char.
- A special character, called a **"null"**, is important because it is the only way the functions that work with a string can know where the string ends.
- This may be written as **'\0'** (zero not capital 'o'). This is the only character whose ASCII value is zero.
- Depending on how arrays of characters are built, we may need to add the null by hand, or the compiler may add it for us.

The following operations performed on character strings,

- ✓ Reading and Writing strings.
- ✓ Combining Strings together.
- ✓ Copying one string to another.
- ✓ Comparing strings for equality.

1.1DECLARING AND INITIALIZING STRING VARIABLES

Declaring a String

A string variable is a valid C variable name and always declared as an array.

The general form of declaration of a string variable is,

```
char string_name [size];
```

The size determines the number of characters in the string_name. When the compiler assigns a character string to a character array ,it automatically supplies a null character('\0') at the end of the string. The size should be equal to the maximum number of characters in the string plus one.

Initializing String Variables

Character arrays may be initialized when they are declared. C permits a character array to be initialized in one of the following forms,

- ✓ Initializing locations character by character.
- ✓ Partial array initialization.
- ✓ Initializing without specifying the size.
- ✓ Array initialization with a string constant.

Initializing locations character by character

If you know all the characters at compile time, you can specify all your data within brackets:

Example,

char s[6]={'h','e','l','l','o'};

The compiler allocates **6** memory locations ranging from **0** to **5** and these locations are initialized with the characters in the order specified. The remaining locations are automatically initialized to null characters as shown in the below **figure 1.1.**

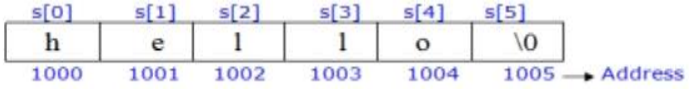

Figure 1.1: Initializing Location Character by character

Partial Array Initialization

If the number of characters values to be initialized is less than the size of the array, then the characters are initialized in the order from 0^{th} location. The remaining locations will be initialized to NULL automatically. Consider the following initialization,

char s[10]={'h','e','l','l','o'};

The above statement allocates **10** bytes for the variable **s** ranging from **0** to **9** and initializes first **5** locations with the characters. The remaining locations are automatically filled with NULL as shown in below **fig 1.2.**

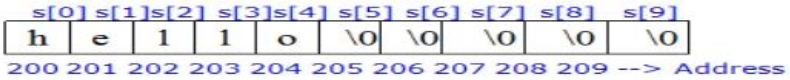

Figure 1.2 Partial Array Initialization

Initialization Without Size

If we omit the size of the array, but specify an initial set of characters, the compiler will automatically determine the size of the array. This way is referred as initialization without size.

char s[]={'h','e','l','l','o'};

In this declaration, even though we have not specified exact number of characters to be used in array s, the array size will be set of the total number of initial characters specified and appends the NULL character. Here, the compiler creates an array of 6 characters. The array s is initialized as shown in Figure 1.**3.**

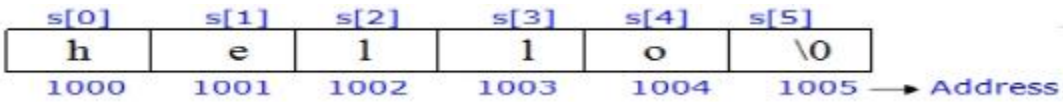

Figure 1.3: Initializing Without size

Array Initialization with a String Constant

It takes of the following form,

char s[]="hello";

Here the length of the string is **5 bytes**, but size is **6 bytes**. The compiler reserves **5+1** memory locations and these locations are initialized with the characters in the order specified. The string

is terminated by Null as shown in the **figure 1.4.**

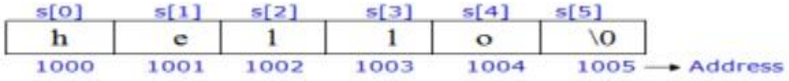

Figure 1.4: Array Initializing With a String

Here are some illegal statements representing initialization of strings, The following declaration creates character array only not a string

char s[5]={'h','e','l','l','o'}; *//no location for appending NULL*

The following declaration is illegal.

char str[3]="Good"; *//size is less than the total characters*

We cannot separate the initialization from declaration.

char str3[5];

str3="Good"; Is not allowed.

1.2.Handling strings as array of characters

1.2 ARRAY OF CHARACTERS/STRINGS

It is actually a two dimensional array of char type.

Example: char names[6][30];

In above example, **names** is an array of strings which can contain 6 string values. Each of the string value can contain maximum 30 characters.

1. Initialization of array of strings

We can initialize an array of strings just as we initialize a normal array The way to initialize an array of strings is

char var_name[R][C]={List of String values};

char specifies that we are declaring an array of strings.

Var_name refers to the array of strings defined by the programmer. The string array name should follow all the rules of a valid identifier of C Language.

[ ] [ ] Pair of square brackets specifies that it is an array of strings.

R specifies the maximum number of string values which can be stored in string array.

C specifies the maximum number of characters which can be stored in each string value stored in string array.

List of String values specifies various string values which are to be stored into string array. The list of string values must be enclosed between braces.

Example 1:

char names[3][20]={"Lovejot","Ajay","Amit"};

L	o	v	e	j	o	t	\0												
A	j	a	y	\0															
A	m	i	t	\0															

In the above example, string array names has been initialized with three values "Lovejot", "Ajay" and "Amit".

Program to initialize an array of strings

```
#include<stdio.h>
void main()
{
char names[3][20]={“Lovejot”,”Ajay”,”Amit”};
int i;
printf(”\nElements of string array are”);
for(i=0;i<3;i++)
printf(“\n%s”,names[i]);
}
```

---

2. Reading and displaying array of strings

We can read and display an array of strings just as we read and display an array of other data types.

Program to read an array of strings.

```
#include<stdio.h>
void main()
{
char names[3][20];
int i;
printf(”\nEnter three string values\n”);
for(i=0;i<3;i++)
gets(names[i]);

printf(”\nElements of string array are”);
for(i=0;i<3;i++)
printf(“\n%s”,names[i]);
}
```

It is also referred as table of strings.

Example 2: char list[6][10]={ “akshay”, “parag”,
“raman”, “srinivas”,
“gopal”};

The names would be store in the memory as shown below.

j=0	1	2	3	4	5	6	7	8	9
A	k	s	h	a	y	\0			
p	a	r	a	g	\0				
r	a	m	a	n	\0				
s	r	i	n	i	v	a	s	\0	
g	o	p	a	l	\0				

1.3 STRING INPUT/OUTPUT FUNCTIONS

Strings can be read from the keyword and can be displayed onto the monitor using the following I/O functions.

Formatted Input Function-scanf ()

The string can be read using the scanf function also. The format specifier associated with the string is **%s.**

Syntax for reading string using scanf function is

scanf ("%s", string_name);

Disadvantages

The termination of reading of data through scanf function occurs, after finding first white space through keyboard. White space may be new line (\n), blank character, tab(\t). For example if the input string through keyword is "hello world" then only "hello" is stored in the specified string.

Formatted Output function-printf ()

The various options associated with printf ():

a) Field width specification

b) Precision specifier

c) Left Justification

a) Field Width Specification

Syntax: ***%ws***

W is the field with specified width.

S indicates that the string is being used.

NOTE:

- If the string to be printed is larger than the field width w, the entire string will be printed.
- If the string to be printed is smaller than the field width w, then appropriate numbers of blank spaces are padded at the beginning of the string so as to ensure that field width w is reached.

Example:

```
#include<stdio.h>
void main ()
{
char s[]="RAMANANDA";
printf ("%4s\n", s);
printf ("%15s",s);
}
```

OUTPUT:

```
RAMANANDA
      RAMANANDA
```

b) Precision Specifier

Syntax: **%w.ns**

W is the field specified width

N indicates that first n characters have to be displayed. This gives precision.

S indicates that the string is being used.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char s []={'R','A','M','A','N','A','N','D','A'};
clrscr();
printf("%0.2s\n",s);
printf("%9.4s\n",s);
printf("%9.3s\n",s);
printf("%3.5s",s);
getch(); }
```

OUTPUT:

```
RA
     RAMA
      RAM
RAMAN
```

NOTE:

- The string is printed right justification by default.

- If **w > n**, **w** columns are used to print first n characters .example 2nd and 3rd printf statements.
- If **w < n**, minimum n columns are used to print first n characters. Example, 1st and 4th printf statements.

c) Left justification

*Syntax:***%-w.ns**

- just before **w** indicates that string is printed using left justification.
- **W** is the field with specified width.
- **S** indicates that the string is being printed.

Example:

```
#include<stdio.h>
#include<conio.h>
void main ()
{
char s []={'R','A','M','A','N','A','N','D','A'};
clrscr();
printf ("%-0.2s\n", s);
printf("%-9.4s\n",s);
printf("%-9.3s\n",s);
printf("%-3.5s",s);
getch(); }
```

OUTPUT:

```
RA
RAMA
RAM
RAMAN
```

Character I/O from Keyboard

To read characters from the keyboard and write to screen it takes the following form:

```
c = getchar( );  //reads one character from the keyboard
putchar(c); // display the character on the monitor
```

The **int getchar(void)** function reads the next available character from the screen 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 character from the screen.

The **int putchar(int c)** function puts the passed character on the screen and returns the same character. This function puts only single character at a time. You can use this method in the loop in case you want to display more than one character on the screen. Check the following example −

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

int c;

   printf("Enter a value :");
   c = getchar();

   printf("\nYou entered: ");
   putchar( c );

return0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads only a single character and displays it as follows −

```
$./a.out
Enter a value : this is test
You entered: t
```

Un-Formatted Input/Output Function: The gets() and puts() Functions

The **char *gets(char *s)** function reads a line from **stdin** into the buffer pointed to by **s** until either a terminating newline or EOF (End of File).

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

NOTE: Though it has been deprecated to use gets() function, Instead of using gets, you want to use fgets().

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

char str[100];

   printf("Enter a value :");
   gets( str );

   printf("\nYou entered: ");
   puts( str );

return0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads the complete line till end, and displays it as follows −

```
$./a.out
Enter a value : this is test
You entered: this is test
```

1.4. STRING HANDLING FUNCTIONS/ MANIPULATION FUNCTIONS

The C Library provides a rich set of string handling functions that are placed under the header file <string.h>.

1.***strcat () function:***

The strcat function joins two strings together. It takes of the following form:

strcat(string1,string2);

string1 and **string2** are character arrays. When the function **strcat** is executed, **string2** is appended to **string1**.It does so by removing the null character at the end of **string1** and placing **string2** from there.

strcat function may also append a string constant to a string variable. The following is valid.

strcat(part1,"Good");

C permits nesting of strcat functions. Example:

strcat(strcat(string1,string2),string3);

Example

```
#include <stdio.h>
#include <string.h>
 main ()
{
char opening[255] = "And, the Oscar goes to... ";
char winner[255] = "American Beauty";
strcat (opening, winner);
printf ("%s", opening); getchar();
}
```

OUTPUT:

```
And, the Oscar goes to... American Beauty
```

The strcmp function compares two strings identified by the arguments and has the value 0 if they are equal. If they are not, it has the numeric difference between the first non matching characters in the strings. It takes the following form:

strcmp(str1,str2);

return value less than **0** means **"str1"** is less than **"str2"‘** return value **0** means **"str1"** is equal to **"str2"‘**

return value greater than **0** means **"str1"** is greater than **"str2"**

String1 and string2 may be string variables or string constants.

Example: strcmp(name1,name2);

strcmp(name1,"John");

strcmp("their" ,"there");

3.***strcpy () function:*** The syntax or general form is as follows:

strcpy(string1,string2);

and assign the contents of string2 to string1. String2 may be a character array variable or a string constant.

Example: strcpy(city ,"Delhi");

strcpy(city1,city2);

program:
```
#include <stdio.h>
#include <string.h>
main ()
{
char word1[] = "And, the winner is....";
char word2[255];
strcpy (word2, word1);
 printf ("%s", word2);
getch ();
}
```

4. strrev () function

Reverses the contents of the string. It takes of the form

strrev(string);

```
Example:
#include<stdio.h>
#include<string.h>
void main()
{
char s[]="hello";
strrev(s);
puts(s);
```

OUTPUT:

olleh

5. strstr () function:

It is a two-parameter function that can be used to locate a sub-string in a string. It takes the form:

strstr (s1, s2);

Example: strstr (s1,"ABC");

The function **strstr** searches the string s1 to see whether the **string s2** is contained in **s1**.If yes,

the function returns the position of the first occurrence of the sub-string. Otherwise, it returns a NULL pointer.

Example: if (strstr (s1, s2) ==NULL)

printf ("substring not found");

else

printf ("s2 is a substring of s1");

6. **strlen():**
This function is used to find the length of the string.It counts the no. of characters in the string.

Example:

```
#include<stdio.h>
#include<string.h>
void main()
{
char name[100]="Gore";
printf ("%d", strlen (name));
getch();
}
```

output: length of string is 4.