String Functions

Definition of String

- String literal (or String Constant) is a sequence of characters enclosed within double quotes.
- “%s” is a placeholder for a string literal.

example:

```
printf(“India”);
```

or

```
char s[10] = “India”;
printf(“%s”,s);
```

Storing string literals

- Always stored in an array terminated by NULL character.
- In C, compiler treats a string literal as a pointer to the first character.

I	N	D	I	A	\0

- Initialized as;

  char *ptr = “Hello world”;

- Cannot be modified.

String Variable

- A string variable is a one dimensional array of characters that is capable of holding a string at a time.
- Make sure the array is one character longer than the size of the string.
- This character array is treated like any other type of array (Modification possible).
- Can be initialized in few ways:

```
char s1[6] = {‘I’,’N’,’D’,’I’,’A’,’\0’};
char s2[6] = “India”;
char s3[] = “India”;
```

Different initializers

- Short length initializer.
- Long length initializer.
- Equal length initializer.

Writing strings using printf()

(i) char str[] = “Hey there”;

printf(“%s”,str); Output: Hey there

(ii) char str[] = “Hey there”;

printf(“%.6s”,str); Output: Hey th

(iii) char str[] = “Hey there”;

printf(“%.6s \n”,str);

printf(“%7.6”,str); Output: Hey th

Hey th

Writing strings using puts()

- Prints strings.
- Automatically adds newline character.
- Doesn’t require defining the placeholder.

example:

```
char str[] = “Hello”;
puts(str);
```

Output: Hello.

Reading strings using scanf()

- Doesn’t store white space characters.

Example:

```
char str[50];
scanf(“%s”,str);
```

Output:

```
How are you doing?
How
```

Reading strings using gets()

- Reads the white spaces as well.

Example;

```
char s[50];
gets(s);
puts(s);
```

Output:

```
How are you doing?
How are you doing?
```

Different string functions

- Uses the header **string.h**
- strcpy() – String copy function.
- strncpy() – Specified number of strings can be copied.
- strlen() – String length function.
- strcat() – String concatenation function.
- strncat() - Specified number of strings can be appended.
- strcmp() – String comparison function.
- strcmpi() – String comparison function but ignores cases.
- strncmp() – Compares first n characters of two strings.
- strupr() – Converting string to uppercase.
- strlwr() - Converting string to lowercase.

- strrev() – Reverse the string.
- strchr() – Finds out first occurrence of a given character in a string.
- strrchr() – Finds out first occurrence in the reverse order.
- strstr() – Finds out first occurrence of a string in the given string.
- strset() – Set all the characters of the string into a given character.
- strnset() – Sets frist n characters of a given string into n characters.

strcpy()

- Synatx:

  char *strcpy(char *dest, const char *src)

- It is used to copy a string pointed by source including null character to the destination.

- Doesn’t check the length of the destination string array.

- Adds NULL character at the end.

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

int main()
{
        char str1[10] = “Hello”;
        char str2[10];
        printf("%s \n", strcpy(str2,str1));
        return 0;
}
```

Output:

Hello

strncpy():

- Copies the limited number of characters specified by the third argument passed to it.

- Synatx:

char *strncpy(char *dest, const char *src, size_t n)

- Leaves string in destination without terminating a null character, if the size of the destination string is equal to or greater than the source string.

str[sizeof(str) – 1] = ‘\0’;

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

int main()
{
        char s1[30];
        char s2[10];
        strcpy(s1,"This is an example");
        printf("%s \n",s1);
        strncpy(s2,s1,9);
        printf("%s \n", s2);
        return 0;
}
```

Output:

This is an example

This is a

strlen()

- Syntax:

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

- strlen() is used to determine the length of the given string.
- It does not count NULL character at the end of the string.
- Calculates the white space.
- Calculates the length of the string and not the length of the array.

str[] = “Hello”;

str[100] = “Hello”;

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

int main()
{
        char str[] = “Hello”;
        printf(“%d \n”, strlen(str));
        return 0;
}
```

Output:

5

strcat()

- Syntax:

char *strcat(char *dest, const char *src)

- strcat function appends the content of source string at the end of destination string.
- Resultant string will be stored in destination string.

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

int main()
{
        char str1[] = “Hello ”;
        char str2[] = “Everyone”;
        printf(“%s \n”, strcat(str1,str2));
        return 0;
}
```

Output:

Hello Everyone.

strncat()

- Appends the limited number of characters specified by the third argument passed to it.
- Automatically adds NULL character at the end of the resulting string.

```
char *strncat(char *dest, const char *src, size_t n)
```

```
#include <stdio.h>
#include <string.h>
int main()
{
        char s1[10] = “Hello”;
        char s2[20] = “Everyone”;
        strncat(s1,s2,3);
        printf("%s \n",s1);
        return 0;
}
```

Output:

Hello Eve

strcmp()

- Synatx:

```
int strcmp(const char *leftStr, const char *rightStr );
```

(str1) (str2)

- Returns:

if str1<str2, less than zero is returned.

if str1==str2, zero is returned.

if str1>str2, greater than zero is returned

str1>str2

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[] = "Hello all";
    char str2[] = "Hello";
    int res = strcmp(str1,str2);
    printf ("Comparison is: %d", res);
    return 0;
}
```

Output:

Comparison is: 1

str1<str2

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[] = "Hello";
    char str2[] = "Hello all";
    int res = strcmp(str1,str2);
    printf ("Comparison is: %d",
res);
    return 0;
}
```

Output:

Comparison is: -1

str1==str2

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[] = "Hello";
    char str2[] = "Hello";
    int res = strcmp(str1,str2);
    printf ("Comparison is: %d", res);
    return 0;
}
```

Output:

Comparison is: 0

str1==str2 (different case)

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[] = "Hello";
    char str2[] = "hello";
    int res = strcmp(str1,str2);
    printf ("Comparison is: %d", res);
    return 0;
}
```

Output:

Comparison is: -1

strcmpi() – Ignores cases.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[] = "Hello";
    char str2[] = "hello";
    int res = strcmpi(str1,str2);
    printf ("Comparison is: %d", res);
    return 0;
}
```

Output:

Comparison is: 0

strncmp():

- Synatx:

```
int strncmp(const char *str1,
const char *str2, size_t n)
```

- Compares only specified length of string.

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[20] = "Hellboy";
        char str2[10] = "Hello";
        int res = strncmp(str,str2,3);
        printf("%d",res);
    return 0;
}
```

Output:

0

strupr()

- Syntax:

```
char *strupr(char *str);
```

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[] = “hey”;
        printf(“%s”,strupr(str));
        return 0;
}
```

Output:

HEY

strlwr()

- Synatx:

```
char *strlwr(char *str);
```

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[] = “HEY”;
        printf(“%s”,strlwr(str));
        return 0;
}
```

Output:

hey

strrev()

- Synatx:

```
char *strrev(char *str);
```

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[] = “Strings”;
        printf(“%s”,strrev(str));
        return 0;
}
```

Output:

sgnirtS

strchr()

- Syntax:

  `char *strchr(char *str, int ch)`

- Searches the given string for a specific character.

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[] = “Strings are fun”;
        printf(“%s”,strchr(str,’i’));
        return 0;
}
```

Output:

ings are fun

strrchr()

- Syntax:

```
char *strrchr(char *str, int ch)
```

- Searches the given string for a specific character but in reverse order.

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[] = “Strings are fun”;
        printf(“%s”,strrchr(str,’s’));
        return 0;
}
```

Output:

s are fun

strstr()

- Syntax:

  `char *strstr(char *str, char *search_string)`

- Searches the given string for a specific string.

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[] = “Strings are fun”;
        printf(“%s”,strrchr(str,”are”));
        return 0;
}
```

Output:

are fun

strset()

- Synatx:

char *strset(char *string, int c);

- Changes every character of a string into the specified character.

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[] = "Strings are fun";
        printf("%s \n",str);
        printf("%s",strset(str,'*'));
        return 0;
}
```

Output:

Strings are fun

****

strnset()

- Synatx:

  char *strnset(char *string, int c, size_t n);

- Changes every character of a string into the specified character.

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[] = "Strings are fun";
        printf("%s \n",str);
        printf("%s",strnset(str,'*', 7));
        return 0;
}
```

Output:

Strings are fun

******* are fun