

STRINGS: ONE-DIMENSIONAL CHARACTER ARRAYS

- Strings in C are represented by **arrays of characters**. **The end of the string is marked with a special character.**

In the *ASCII* character set, the null character value is 0. The null or string-terminating character is represented by another character escape sequence, `\0`.

- **Strings are actually one-dimensional array of characters terminated by a null character '`\0`'. Thus a null-terminated string contains the characters that comprise the string followed by a null.**

Declaration of A String

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

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

The C compiler automatically places the '\0' at the end of the string when it initializes the array

Declaration of A String

- Strings can be declared like one-dimensional arrays.

- For
char

- char text[80];

example,
str[30];

- An array formed by characters is a string in C.
- **The end of the string is marked with a the null character.**
- **When the character array size is explicitly specified** and the number of initializers completely fills the array size, the null character is not automatically appended to the array.

Initiation of a string

```
char s[]="Hello, World";
```

Printing Strings

- The conversion type 's' may be used for output of strings using printf().
- The following points should be noted.
 - When the field width is less than the length of the string, the entire string is printed.
 - The integer value on the right side of the decimal point specifies the number of characters to be printed.
 - When the number of characters to be printed is specified as zero, nothing is printed.
 - The minus sign in the specification causes the string to be printed as left justified.

Example

```
#include <stdio.h>
int main()
{
char s[]="Hello, World";
printf(">>%s<<\n",s);
printf(">>%20s<<\n",s);
printf(">>%-20s<<\n",s);
printf(">>%.4s<<\n",s);
printf(">>%-20.4s<<\n",s);
printf(">>%20.4s<<\n",s);
return 0;
}
```

output

```
>>Hello, World<<
>>    Hello, World<<
>>Hello, World    <<
>>Hell<<
>>Hell<<
>> Hell<<
```

String INPUT/OUTPUT

- One special case, where the null character is not automatically appended to the array, **is when the array size is explicitly specified and the number of initializers completely fills the array size.**
- `char nonterminated[5] = "12345";`
- `printf()` with the width and precision modifiers in the `%s` conversion specifier may be used to display a string.
- The `%s` format does not require the ampersand before the string name in `scanf()`.

String INPUT/OUTPUT

- If fewer input characters are provided, `scanf()` hangs until it gets enough input characters.
- `scanf()` only recognizes a sequence of characters delimited by white space characters as an external string.
- The library function `sprintf()` is similar to `printf()`.
- The C library function `sprintf ()` is used to store formatted data as a string.
- You can also say the `sprintf ()` function is used to create strings as output using formatted data.
- The only difference is that the formatted output is written to a memory area rather than directly to a standard output.

Enter your Name and Print

```
#include <stdio.h>
int main()
{
    char buffer[50];
    int a = 15, b = 25, res;
    res = a + b;
    sprintf(buffer, "The Sum of %d and %d is %d", a, b,
res);
    printf("%s", buffer);
    return 0;
}
```

Enter your Name and Print

```
#include <stdio.h>
int main()
{
    char str[50];
    printf("Enter a string : ");

    //Option 1 to read and print string
    scanf("%[^\\n]s",str);
    printf("You entered: %s", str);

    return(0);
}
```

Enter your Name and Print

```
#include<stdio.h>
int main()
{
    char string[20];
    printf("Enter the string: ");
    fgets(string,20,stdin);      #input from stdin stream
    printf("\nThe string is: %s",string);
    return 0;
}
```

isupper()

```
#include <stdio.h>
#include <ctype.h>
int main()
{ int var1 = 'M';
  int var2 = 'm';
  char ch = 'g'
  if( isupper(var1) )
  {
    printf("var1 = |%c| is uppercase character\n", var1 );
  }
  else
  {
    printf("var1 = |%c| is not uppercase character\n", var1 );
  }
  printf("var1 = |%c| is in uppercase character\n", toupper(ch)
    );
  return 0;
```

toupper()

```
#include <stdio.h>
#include <ctype.h>
int main()
{ char c; c = 'm';
  printf("%c -> %c", c, toupper(c));
  /*Displays the same argument passed if other characters
  than lowercase character is passed to toupper()*/.
  c = 'D';
  printf("\n%c -> %c", c, toupper(c));
  c = '9';
  printf("\n%c -> %c", c, toupper(c));
  return 0;
}
```

m -> M
D -> D
9 -> 9

fscanf & fprintf

fprintf(FILE *stream, const char *format, ...) sends formatted output to a stream.

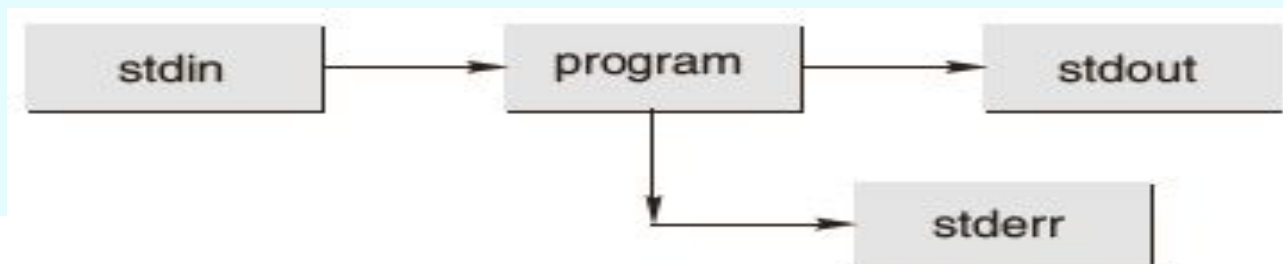
```
fprintf(fp, "%s %s %s %d", "We", "are", "in",  
2012);
```

fscanf(FILE *stream, const char *format, ...) reads formatted input from a stream.

```
fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
```

String input and output using fscanf() and fprintf()

- stdin, stdout, and stderr: Each C program has three I/O streams.
 - The input stream is called standard-input (stdin); the output stream is called standard-output (stdout); and the side stream of output characters for errors is called standard error (stderr).
 - Now one might think that calls to fprintf() and fscanf() differ significantly from calls to printf() and scanf().
 - fprintf() sends formatted output to a stream and fscanf() scans and formats input from a stream.



Standard input and output

Standard File	File Pointer	What is happening
Standard input	stdin	Standard input is stream data (often text) going into a program (data transfers by use of the <i>read</i> operation)
Standard output	stdout	Standard output is the stream where a program writes its output data(data transfer with the <i>write</i> operation)
Standard error	stderr	Another output stream typically used by programs to output error messages. It is a stream independent of standard output and can be redirected separately

See the following example

```
#include <stdio.h>
int main()
{
    int first, second;

    fprintf(stdout, "Enter two inputs in this line: ");
    fscanf(stdin, "%d %d", &first, &second);

    fprintf(stdout, "Their sum is: %d.\n", first + second);
    return 0;
}
```

String Manipulation

- C has the weakest character string capability of any general-purpose programming language.
- Strictly speaking, there are no character strings in C, just arrays of single characters that are really small integers.
- If `s1` and `s2` are such 'strings' a program cannot
 - Assign one to the other: `s1 = s2;`
 - Compare them for collating sequence: `s1 < s2;`
 - Concatenate them to form a single longer string: `s1 + s2;`
 - Return a string as the result of a function.

Table String Manipulation Functions available in string.h

Function	Description
<code>strcpy(s1,s2)</code>	Copies <code>s2</code> into <code>s1</code>
<code>strcat(s1,s2)</code>	Concatenates <code>s2</code> to <code>s1</code> . That is, it appends the string contained by <code>s2</code> to the end of the string pointed to by <code>s1</code> . The terminating null character of <code>s1</code> is overwritten. Copying stops once the terminating null character of <code>s2</code> is copied.
<code>strncat(s1,s2,n)</code>	Appends the string pointed to by <code>s2</code> to the end of the string pointed to by <code>s1</code> up to <code>n</code> characters long. The terminating null character of <code>s1</code> is overwritten. Copying stops once <code>n</code> characters are copied or the terminating null character of <code>s2</code> is copied. A terminating null character is always appended to <code>s1</code> .
<code>strlen(s1)</code>	Returns the length of <code>s1</code> . That is, it returns the number of characters in the string without the terminating null character.
<code>strcmp(s1,s2)</code>	Returns 0 if <code>s1</code> and <code>s2</code> are the same Returns less than 0 if <code>s1<s2</code> Returns greater than 0 if <code>s1>s2</code>
<code>strchr(s1,ch)</code>	Returns pointer to first occurrence <code>ch</code> in <code>s1</code>
<code>strstr(s1,s2)</code>	Returns pointer to first occurrence <code>s2</code> in <code>s1</code>

Copying a String into another

- Since C never lets entire arrays to be assigned, the `strcpy()` function can be used to copy one string to another.
 - `strcpy()` copies the string pointed to by the second parameter into the space pointed to by the first parameter.
 - The entire string, including the terminating NUL, is copied and there is no check that the space indicated by the first parameter is big enough.
 - The given code shows the use of the `strcpy(str1, str2)` function.

strcpy()

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

**Finally
Mid-sem is over!**

puts() and gets()

The C library function **puts(str)** writes a string to stdout up to but not including the null character. A newline character is appended to the output.

The C library function **gets (str)** reads a line from stdin and stores it into the string.

It stops when either the newline character is read or when the end-of-file is reached, whichever comes first.

```
#include <stdio.h>
int main()
{
    char str[50];
    printf("Enter a string : ");
    gets(str);
    \\fgets(str, sizeof(str), stdin);

    printf("You entered: %s",
    str);
    return(0);
}
```

Comparing strings

- strcmp() takes the start addresses of two strings as parameters and returns the value zero if the strings are equal.

declaration

int strcmp(char *str1, char *str2)

- if Return value < 0 then it indicates str1 is less than str2.
- if Return value > 0 then it indicates str2 is less than str1.
- if Return value = 0 then it indicates str1 is equal to str2.

- Each character is compared in turn and a decision is made as to whether the first or second string is greater, based on that character (ASCII value).
- Only if the characters are identical do you move to the next character and, if *all* the characters were identical, zero is returned.

strcmp()

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char str1[15];
    char str2[15];
    int ret;

    strcpy(str1, "abcdef");
    strcpy(str2, "ABCDEF");

    ret = strcmp(str1, str2);
    if(ret < 0)
    {
        printf("str1 is less than str2");
    }
    else if(ret > 0)
    {
        printf("str2 is less than str1");
    }
    else
    {
        printf("str1 is equal to str2");
    }
    return(0);
}
```

str2 is less than str1

//strcmp will give a positive number if the first string is greater

Comparing strings

- Since C never lets entire arrays to be assigned, the `strcpy()` function can be used to copy one string to another.
- Strings can be compared by the help of `strcmp()` function.
- The arithmetic addition cannot be applied for joining two or more strings; this can be done by using the standard library function, `strcat()`.

Putting strings together strcat()

char strcat(dest, src)

Parameters

dest -- This is (pointer to) the destination array, which should contain a C string, and should be large enough to contain the concatenated resulting string.

src -- This is the string to be appended. This should not overlap the destination.

This function returns (a pointer to the) resulting string dest.

String concatenation

```
#include <stdio.h>
int main()
{
    char string1[20];
    char string2[20];

    strcpy(string1, "Welcome");
    strcpy(string2, "ToPCclass");

    printf("Returned String : %s\n", strcat( string1, string2 ));
    printf("Concatenated String : %s\n", string1 );
    return 0;
}
```

Returned String : WelcomeToPCclass
Concatenated String : WelcomeToPCclass

Putting strings together strcat()

- The arithmetic addition cannot be applied for joining of two or more strings in the manner
 - `string1 = string2 + string3;` or
 - `string1 = string2 + "RAJA";`
 - For this, the standard library function, `strcat()`, that concatenates strings is needed. It does not concatenate two strings together and give a third, new string.
 - In this example, the first call to `printf` prints "Hello,", and the second one prints "Hello, world!", indicating that the contents of `str` have been appended to the end of `s`.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s[30] = "Hello,";
    char str[] = "world!";

    printf("%s\n", s);
    strcat(s, str);

    printf("%s\n", s);
    return 0;
}
```

Programs on strings

- WAP to find the reverse of a string by using library function for reverse operation.
- WAP to replace all occurrences of a character in a given string with a new character.

PROGRAM CODE

```
#include<stdio.h>
#include<string.h>
int main()
{
char s[100]; revs[100]
printf("\nEnter a string : ");
gets(s);
revs = strrev(s);
printf("\nThe reverse of the string is %s ", revs)
return 0 ;
}
```

Reverse a string

RUN-1

Enter a String : I am good.
The reverse of the string is .doog
ma I

RUN-1

Enter a String : How are you?
The reverse of the string is
?uoy era woH

PROGRAM CODE

```
#include<stdio.h>
#include<string.h>
int main()
{
char s[100];
int l, i;

printf("\nEnter a string : ");
gets(s);
l=strlen(s);

printf("\nThe reverse of the string is ");
for(i=l-1; i>=0; i--)
printf("%c", s[i]);
return 0;
}
```

**Reverse a string –
without library
function**

INPUT/ OUTPUT

RUN-1

Enter a String : I am good.
The reverse of the string is .doog
ma I

RUN-1

Enter a String : How are you?
The reverse of the string is
?uoy era woH

PROGRAM CODE

```
#include<stdio.h>
#include<string.h>
int main()
{
char s[100], och, nch;
int i, flag=0;
printf("\nEnter a string :");
gets(s);
printf("\nEnter a character :");
scanf("%c", &och);
printf("\nEnter the new character :");
scanf(" %c", &nch)//add a space
for(i=0; s[i]!='\0'; i++)
{
    if(s[i]==och)
    {
        s[i]=nch;
        flag=1;
    }
}
}
//for loop ends here
```

Replace a char

INPUT/OUTPUT

RUN-1

Enter a string: Bachelor of Engineering

Enter a character: e

Enter a new character: A

After the replacement by new character, the string is BachAlor of EnginAAring

```
if(flag==1)
printf("\nAfter the replacement by new character, the string is %s", s);
else
printf("\nThe given string does not contain the character %c", och);
return 0;
}
```