جامعة بيرزيت
BIRZEIT UNIVERSITY

Strings

PROBLEM SOLVING AND PROGRAM DESIGN In C
7th EDITION
Jeri R. Hanly, Elliot B. Koffman

By: Mamoun Nawahdah (PhD)
2013/2014

Chapter Objectives

- ❖ To understand how a **string** constant is stored in an array of characters.
- ❖ To learn about the placeholder **%s** and how it is used in **printf** and **scanf** operations.
- ❖ To learn some of the operations that can be performed on strings such as **copying** strings, extracting **substrings**, and **joining** strings using functions from the library **string**.
- ❖ To understand how **C** **compares** two strings to determine their relative order.

Characters

- ❖ Characters are small integers (**0-255**).
- ❖ Character constants are integers that represent corresponding characters:
  - **‘0’** → 48
  - **‘A’** → 65
  - **‘\0’** (NULL) → 0
  - **‘\t’** (TAB) → 9
  - **‘\n’** (newline) → 10
  - SPACE → 32

Strings

- ❖ String → **a group of characters.**
- ❖ **C** implements the string data structure using **arrays of type char**.
- ❖ Two interpretations of String:
  - **Arrays** whose elements are characters.
  - **Pointer** pointing to characters.

Strings cont.

- ❖ Strings are always terminated with a **NULL** characters(‘**\0**’).
- ❖ **C** needs this to be present in every string so it knows where the string ends.

char a[] = “hello\n”;

char* b = “hello\n”;

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
*b	*(b+1)	*(b+2)	*(b+3)	*(b+4)	*(b+5)	*(b+6)
h	e	l	l	o	\n	null
104	101	108	108	111	10	0

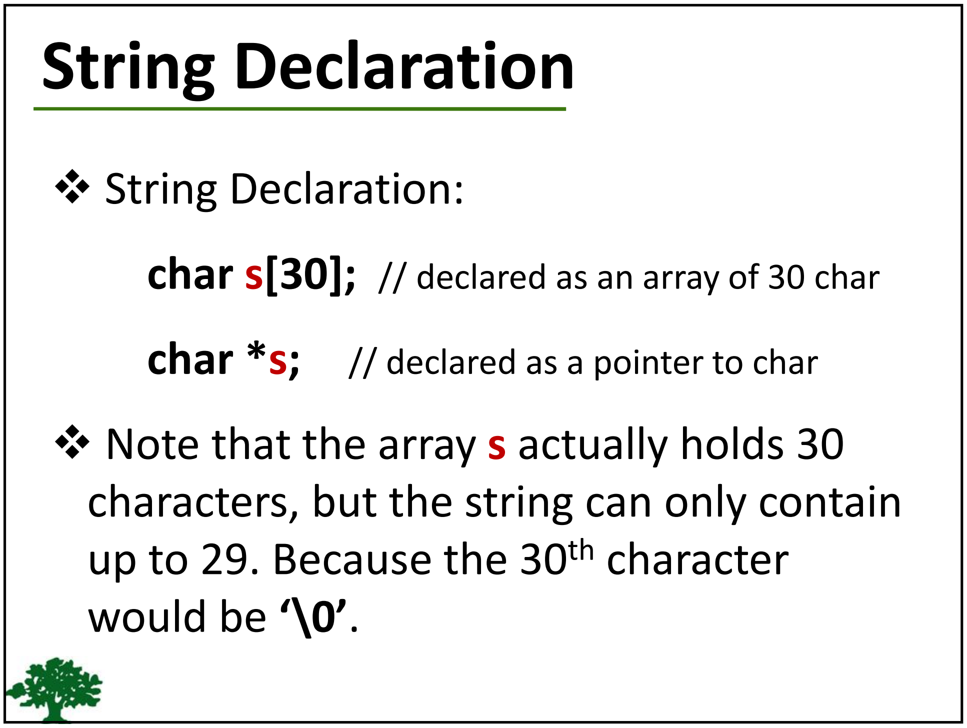
String Constants

- ❖ We have already used string constants extensively in our earlier work:

printf(“**Hello There**”);

  - **Hello There** is a string constant.
- ❖ In **C**, string constants are identified by surrounding **“**
  - Note that this is different from characters – they use single quotes – **‘**
  - Spaces are just treated like any other character.
  - A **null** character is automatically inserted after the constant string.
- ❖ You can also use **#define** directive to associate a symbolic name with a string constant:

#define ERR “*****ERROR*****”



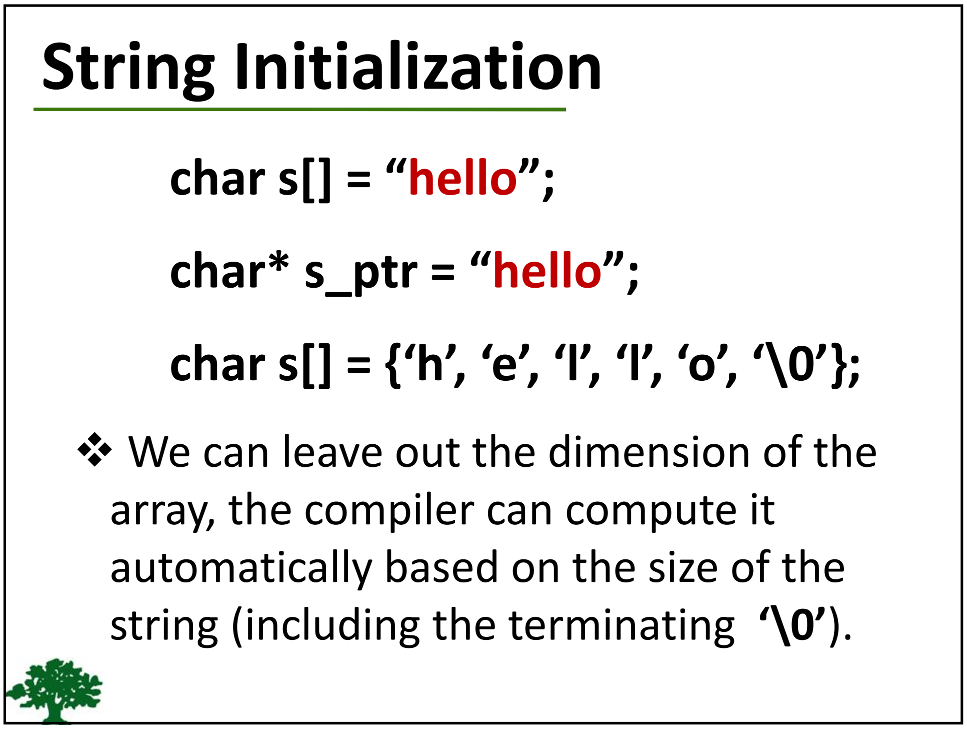
String Declaration

- ❖ String Declaration:

char s[30]; // declared as an array of 30 char

char *s; // declared as a pointer to char

- ❖ Note that the array **s** actually holds 30 characters, but the string can only contain up to 29. Because the 30th character would be **'\0'**.



String Initialization

char s[] = "hello";

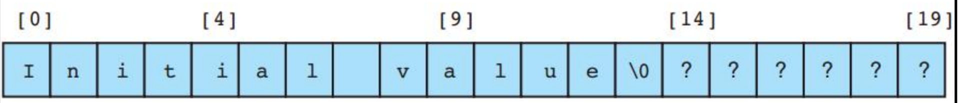
char* s_ptr = "hello";

char s[] = {'h', 'e', 'l', 'l', 'o', '\0'};

- ❖ We can leave out the dimension of the array, the compiler can compute it automatically based on the size of the string (including the terminating **'\0'**).

Example

char **str**[20] = "**Initial value**";



[0]				[4]					[9]					[14]					[19]
I	n	i	t	i	a	l		v	a	l	u	e	\0	?	?	?	?	?	?

- ❖ Notice that **str**[13] contains the character '**\0**', the null character that marks the end of a string.

String Input/Output

- ❖ Both **printf** and **scanf** can handle string arguments as long as the placeholder **%s** is used in the format string:

  char **st**[] = “**hello**”;

  printf(“**%s**\n”, **st**);

- ❖ The **printf** function, like other standard library functions that take string arguments, depends on finding a **null** character in the character array to mark the end of the string.

Input/Output Cont.

❖ The approach **scanf** takes to string input is very similar to its processing of numeric input.

- When it scans a string, **scanf** skips leading whitespace characters such as blanks, newlines, and tabs.
- Starting with the first non-whitespace character, **scanf** copies the characters it encounters into successive memory cells of its character array argument.
- When it comes across a whitespace character, scanning stops, and **scanf** places the **null** character at the end of the string in its array argument.

Input/Output Cont.

```
char st[10];

scanf(“%s”, st);    // bad practice

scanf(“%9s”, st);  // Good practice, prevents overflowing
```

❖ Notice that there is no **&**, this is because strings are arrays, and arrays are already pointers.

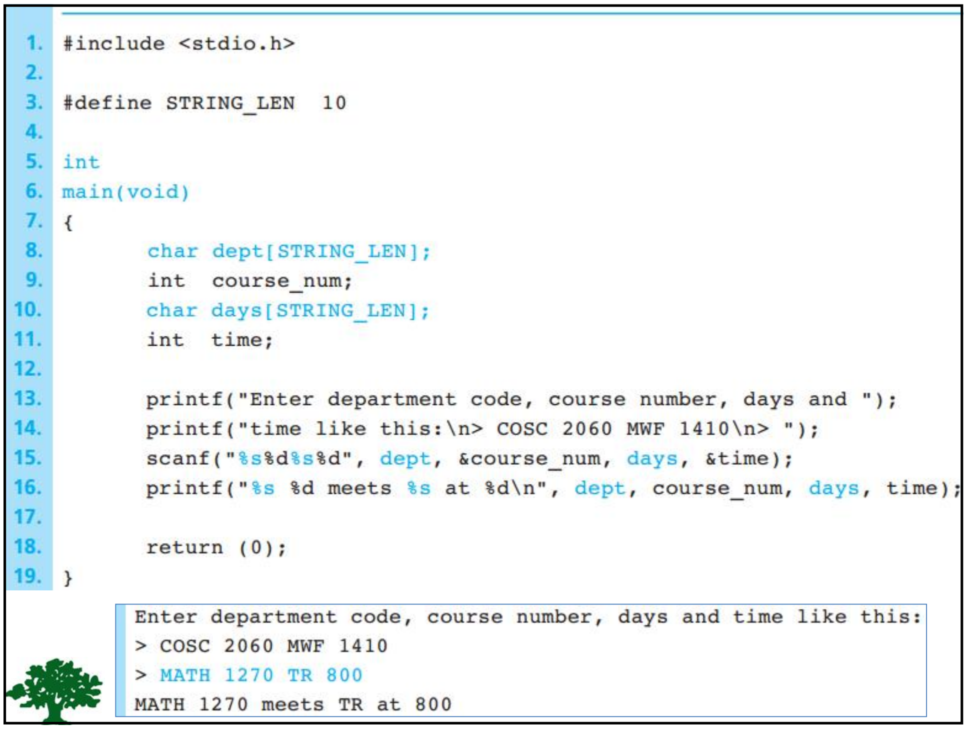
Example

printf (“**%8s%3s**\n", "Short", "Strings");

- The 1st string is displayed **right-justified** in a field of 8 columns.
- The 2nd string is longer than the specified field width, so the field is expanded to accommodate it exactly with no padding.

printf("**%-20s**\n", “president”);

- Placing a **minus sign** prefix on a placeholder’s field width causes **left justification** of the value displayed.



```
#include <stdio.h>

#define STRING_LEN  10

int
main(void)
{
        char dept[STRING_LEN];
        int  course_num;
        char days[STRING_LEN];
        int  time;

        printf("Enter department code, course number, days and ");
        printf("time like this:\n> COSC 2060 MWF 1410\n> ");
        scanf("%s%d%s%d", dept, &course_num, days, &time);
        printf("%s %d meets %s at %d\n", dept, course_num, days, time);

        return (0);
}
```

```
Enter department code, course number, days and time like this:
> COSC 2060 MWF 1410
> MATH 1270 TR 800
MATH 1270 meets TR at 800
```

String Assignment

```
char str[20];
str = “Test String”;    // Does not work
```

❖ Compiler error message such as ***Invalid target of assignment***.

❖ Exception: we can use the assignment symbol in a declaration of a string variable with initialization.

```
char str[] = “hello”;
```

❖ Instead use the **strcpy** function.

```
strcpy( string1, string2 );
```

String Library Functions

❖ **C** provides functions to work with strings, these are located in the **string.h** file.

❖ Put **#include <string.h>** at the top of the program to use these.

❖ Note that all of these function **expect** the strings to be **null** terminated.

strlen() function

Syntax: **len = strlen(ptr);**

- Where **len** is an integer and **ptr** is a pointer to char. **strlen()** returns the length of a string, **excluding** the **null**.
- The following code will result in **len** having the value **13**.

```
int   len;
char  str[15] = “Hello, world!”;
len = strlen(str);
```

strcpy() function

Syntax: **strcpy(ptr1, ptr2);**

- Where **ptr1** and **ptr2** are pointers to char.
- **strcpy()** is used to copy a null-terminated string into a variable.

strcpy() function

```
char  S[25];
char  D[25];
```

- Putting text into a string:

```
strcpy(S,  "This is String 1.");
```

- Copying a whole string from S to D:

```
strcpy(D, S);
```

- Copying the tail end of string S to D:

```
strcpy(D, &S[8]);
```

strncpy() function

Syntax: `strncpy(ptr1, ptr2, n);`

- Where **n** is an integer and **ptr1** and **ptr2** are pointers to char.
- **strncpy()** is used to **copy a portion** of a possibly null-terminated string into a variable.
- Function **strncpy** copies up to **n** characters of the source string to the destination.

strncpy() function

char **S**[25];

char **D**[25];

- Putting text into the source string:

  strcpy(**S** , "This is String 1.");

- Copying **4** characters from the beginning of **S** to **D** and placing a **null** at the end:

  strncpy(**D** , **S** , **4**);

  D[4] = '**\0**';

strncpy() function

- Copying two characters from the middle of string **S** to **D**:

  strncpy(D , &S[5], 2);

  D[2] = '\0';

- Copying the tail end of string **S** to **D**:

  strncpy(D , &S[8], 15);

  - which produces the same result as

  strcpy(D, &S[8]);

strcat() function

Syntax: strcat(ptr1, ptr2);

- Where **ptr1** and **ptr2** are pointers to strings.
- strcat() is used to **concatenate** a **null** terminated string to end of another string variable.
- This is equivalent to pasting one string onto the end of another, overwriting the **null** terminator.

strcat() function

- The strcat() function joins 2 strings together:

char **s1**[25] = "world!";

char **d1**[25] = "Hello, ";

- Concatenating the whole string **s1** onto **d1**:

strcat(d1, s1);

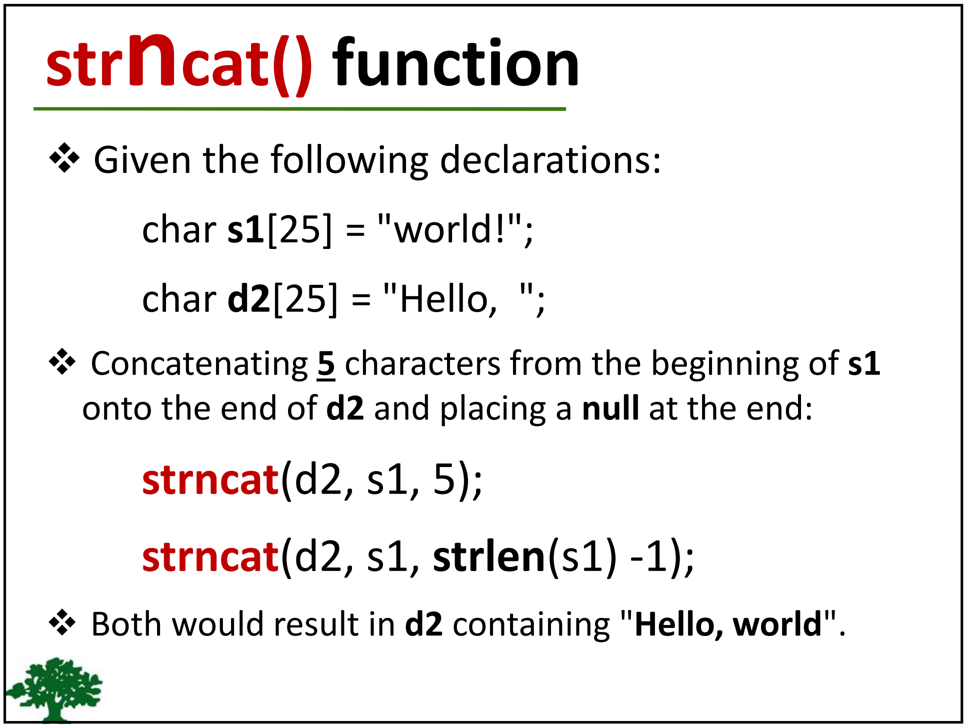
strcat() function

```
//another example of concatenating string

   char s1[20];

   char s2[20];

   strcpy(s1,“Crying   ");

   strcpy(s2,“Baby");

   printf("%s",   strcat(s1,s2)  );
```

strncat() function

Syntax: **strncat**(ptr1, ptr2, n);

❖ **strncat()** is used to **concatenate a portion** of a possibly null-terminated string onto the end of another string variable.



strncat() function

- ❖ Given the following declarations:

    char **s1**[25] = "world!";

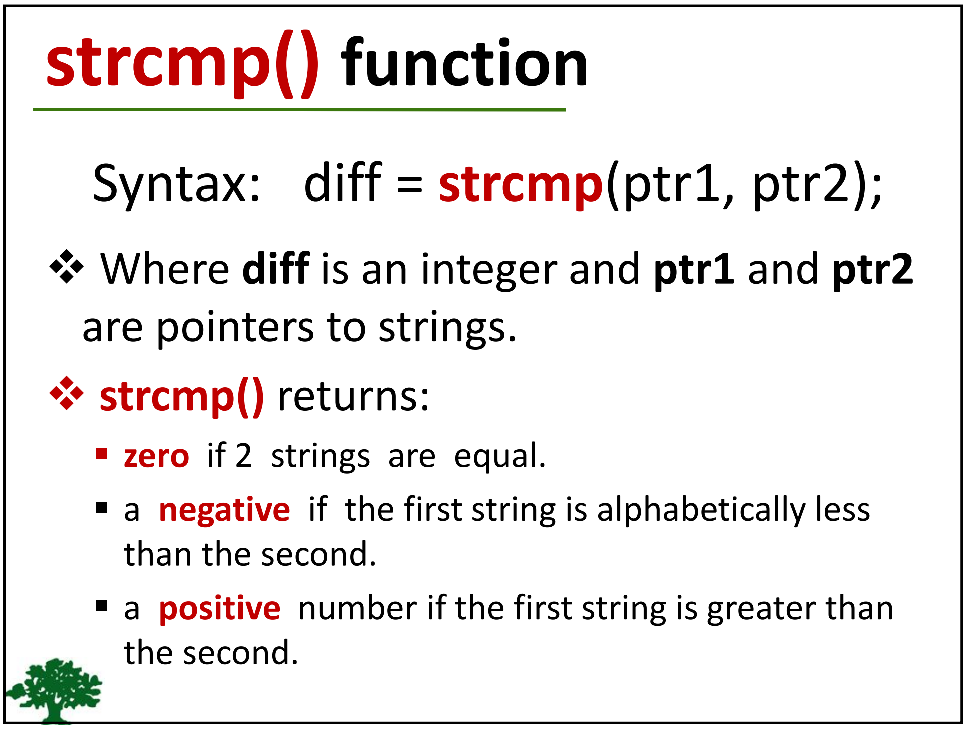
    char **d2**[25] = "Hello, ";

- ❖ Concatenating **5** characters from the beginning of **s1** onto the end of **d2** and placing a **null** at the end:

    strncat(d2, s1, 5);

    strncat(d2, s1, **strlen**(s1) -1);

- ❖ Both would result in **d2** containing "**Hello, world**".



strcmp() function

Syntax: diff = **strcmp**(ptr1, ptr2);

- ❖ Where **diff** is an integer and **ptr1** and **ptr2** are pointers to strings.
- ❖ **strcmp()** returns:
  - **zero** if 2 strings are equal.
  - a **negative** if the first string is alphabetically less than the second.
  - a **positive** number if the first string is greater than the second.

strncmp() function

- ❖ Compares the first **n** characters of first string and second string returning positive, zero, and negative values as does **strcmp.**

```
if (strncmp(n1, n2, 12) == 0) {
        :
}
```

strcmp() and strncmp() function

/*diff will have a **negative** value after the following statement is executed.*/

```
diff = strcmp(str1, str2);  //strcmp(“cat”,”cut”)

printf("%d", diff);        // -1

diff = strncmp(str1, str2, 2);

printf("%d",diff);         // -20
```

strcmp() and strncmp() function

```
/*diff will have a positive value after the
   following statement is executed.*/

diff = strcmp(str2, str1);  //strcmp(“cut”,”cat”)

printf("%d",diff);                    // 1

diff = strncmp(str2, str1, 3);

printf("%d",diff);                    // 20
```

strcmp() and strncmp() function

```
/*diff will have a value of zero (0) after the
   execution of the following statement */

diff = strcmp(str1,  str1);

printf("%d", diff);                  //   0

diff = strncmp(str1, str1, 2);

printf("%d",diff);                   //   0
```

strcmpi() function

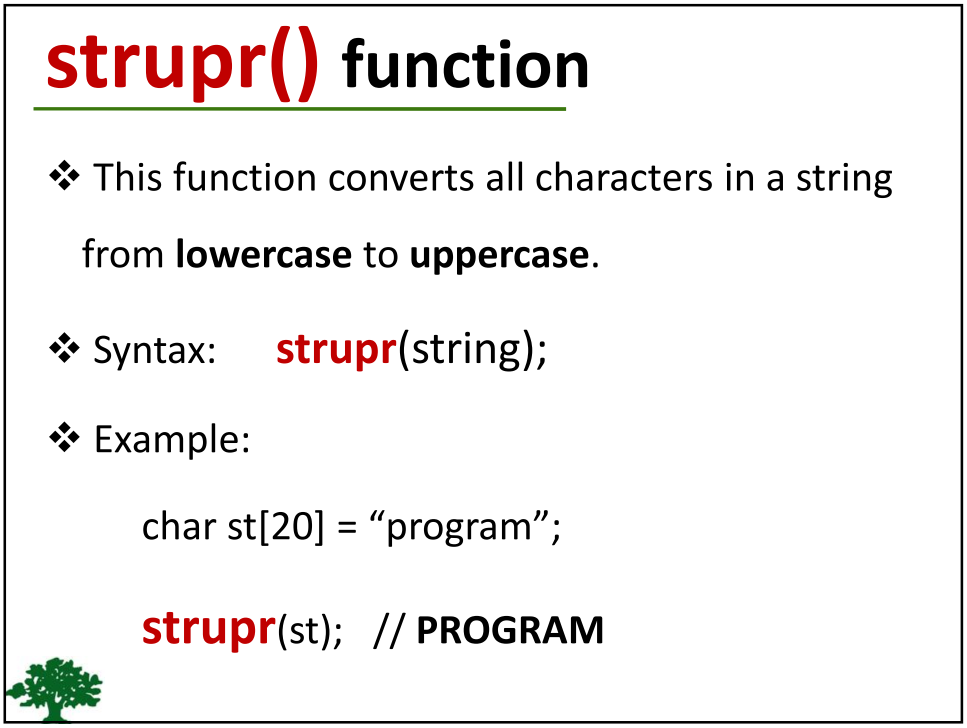
- This function is same as **strcmp()** which compares **2** strings but **not case sensitive**.
- Example:

```
strcmpi("THE","the");   //will return 0.
```

strlwr() function

- This function converts all characters in a string from **uppercase** to **lowercase**.
- Syntax: **strlwr**(string);
- For example:

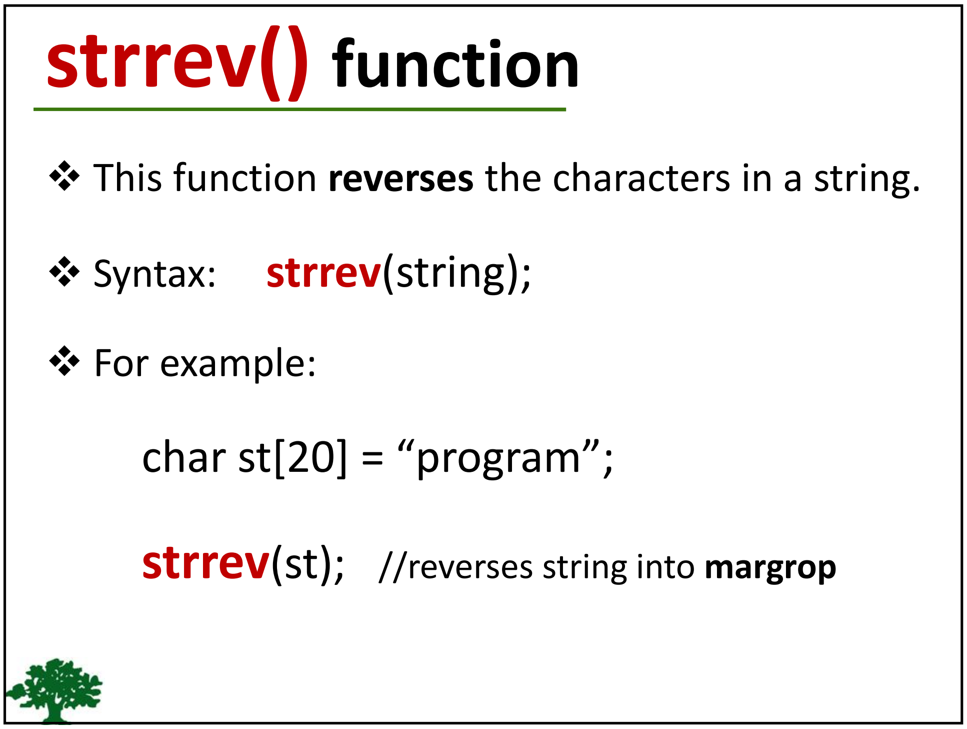
```
char st[20] = "SYSTEMS";

strlwr(st) ;  //converts to systems
```



strupr() function

- This function converts all characters in a string from **lowercase** to **uppercase**.
- Syntax: **strupr**(string);
- Example:

```
char st[20] = “program”;

strupr(st);   // PROGRAM
```



strrev() function

- This function **reverses** the characters in a string.
- Syntax: **strrev**(string);
- For example:

```
char st[20] = “program”;

strrev(st);   //reverses string into margrop
```