



ICS-103

Computer Programming in C

Chapter 8

Strings

Dr. Tarek Ahmed Helmy El-Basuny

Outline of Ch. 08 Topics

- ❑ **String Constants and Variables**
- ❑ **String Input and Output Functions**
 - scanf, printf, gets, puts, fgets, fputs functions with string
- ❑ **Character Related Functions**
 - C provides functions that allows us to check, i.e. **is the character numeric or alphabetic, is upper case or lower case, etc.**
- ❑ **String Library Functions**
 - strlen, strcpy, strcmp, strcat, strstr, strtok,
- ❑ **Arrays of Strings and Arrays of Pointers**

What is a String Constant?

- ❑ A **String** means collection of characters (digits, letters, and special symbols) to form a particular word (i.e. name of the person, address, job, etc.)
- ❑ A **string constant** consists of zero or more character enclosed in double quotes " ", i.e.:
 - ⇒ "Hello World", is a string constant
 - ⇒ " " is an empty string constant,
 - ⇒ **Valid** String constants, i.e. "W" "100" "24, Doha Street"
 - ⇒ A **string constant** can be of any length.
- ❑ Every string **constant ends up with** a **NULL** (\0) character which is automatically assigned by the C compiler.
- ❑ In C, **there is no String Data Type**, instead of we use array of type character to create a String.

Declaring String Constant

- ❑ We have already used string constants extensively in our earlier work:
- ❑ `printf ("The result is: %d\n", result);`
The format string, `"The result is: %d\n"` is a string constant
- ❑ String constant can be declared as following:
 - `const char arr[] = "Error message:";`
 - It can also appear in `#define` directive, such as:
`#define ERR_MSG "Error message:"`
- ❑ Notice that:
 - **Character** constant `'A'` is a single character value (stored in 1 byte) as the ASCII value for A.
 - **String** constant `"A"` is an array contains character `'A'` and `'\0'` (NULL)
 - `'A'` is not equal to `"A"`

What is a String Variable?

❑ In C, a **string variable** is an **array** of type **char**

❑ We can declare a string variable as follows:

```
char string_var[20]; /* Array of char */
```

❑ We can initialize a string variable as follows:

```
/* A list of chars terminated by '\0' */
```

```
char str[16] = {'H','e','l','l','o',' ','  
               'W','o','r','l','d','\0'};
```

```
/* A string enclosed between double quotes */
```

```
char str[16] = "Hello World";
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
H	e	l	l	o		W	o	r	l	d	\0	?	?	?	?

array str[16]

String Variables (cont'd)

- We can **limit the string (array) size** as follows:

```
char str2[] = "Hello World";    /* 12 chars */
```

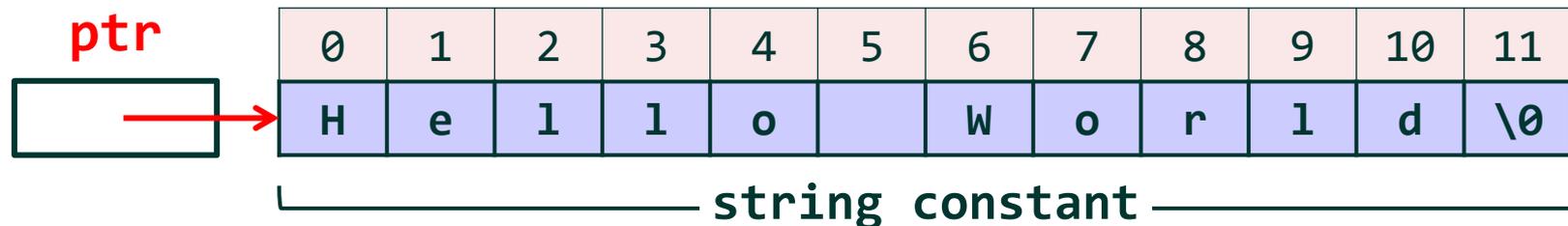
Only 12 characters are allocated (including '`\0`')

0	1	2	3	4	5	6	7	8	9	10	11
H	e	l	l	o		W	o	r	l	d	\0

array str2[]

- We can also declare a pointer to a string as follows:

```
char *ptr = "Hello World";
```



The **NULL** Character '\0'

- ❑ It is a byte that has the value zero
- ❑ Used to mark the end of a string in C
- ❑ A string constant is always ended with '\0'
- ❑ For example: "Hello World" has 12 chars (not 11)

0	1	2	3	4	5	6	7	8	9	10	11
H	e	l	l	o		W	o	r	l	d	\0

- ❑ C functions use '\0' to compute the string length
 - ➔ To avoid passing the size of a string to a function
 - ➔ A string variable must also terminate with a **NULL** char
 - ➔ The empty string "" stores the NULL char '\0'

Input a String with `scanf`

- ❑ To input a string, the **placeholder** must be `%s`

```
char str[16];
```

```
/* str length must not exceed 15 chars */
```

```
scanf("%s", str);
```

```
/* when reading a string, scanf skips white */
```

```
/* space such as blanks, newlines, and tabs */
```

```
/* It stops reading at first white space */
```

```
/* It inserts '\0' at end of str */
```

```
scanf("%15s", str);
```

```
/* prevents reading more than 15 chars */
```

- ❑ Notice that there is **no** need for `&` before `str`

- ➞ Because `str` is an array, and it is passed by **address**.

Output a String with `printf`

- ❑ To print a string, the placeholder must also be `%s`
- ❑ Example of string input and output:

```
char str[16]; /* must not exceed 15 chars */
printf("Enter your first name: ");
scanf("%15s", str);
printf("Hello %s\n", str);
```

```
Enter your first name: Ahmed
Hello Ahmed
```

- ❑ If `printf` displays a string that does not end with `'\0'` then it causes a **run-time error**.

Example of String **Input/Output**

```
#include <stdio.h>

int main(void) {
    char dept[8], days[8];
    int course_num, time;

    printf("Enter course code, number, days, and time\n");
    printf("Similar to this: MATH 101 UTR 1100\n");
    printf("\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 course code, number, days, and time
Similar to this: MATH 101 UTR 1100

> ICS 103 MW 1100
ICS 103 meets MW at 1100
```

Placeholders Used with **printf**

Value	Placeholder	Output (is blank)
'a'	%c	a
	%3c	a
	%-3c	a
-10	%d	-10
	%6d	-10
	%-6d	-10
49.76	%.3f	49.760
	%9.1f	49.8
	%9.2e	4.98e+01
"fantastic"	%s	fantastic
	%12s	fantastic
	%-12s	fantastic

The `gets` and `puts` Functions

- ❑ A problem with `scanf` is that, it stops reading a string when it encounters a blank (or any whitespace).
- ❑ Blanks are natural separators between numeric data values, but it is a valid character in a string.
- ❑ To read a full line including (blanks/whitespaces), use the `gets` function that continues reading until the newline char (`Enter key`) is read.
- ❑ The `'\n'` character representing the Enter key and is **not stored** in the string. It is replaced with `'\0'`.
- ❑ The `puts` function is used to **print** a string.
 - ➔ `puts` automatically prints `'\n'` at end of the string.

Example of **gets** and **puts**

```
char line[80];  
printf("Type anything: ");  
gets(line);  
printf("You typed: ");  
puts(line);
```

```
Type anything: I enjoy programming in C  
You typed: I enjoy programming in C
```

File Input with `fgets`

- ❑ For data files, the `stdio` library provides the `fgets` function that works similar to `gets`.

```
char * fgets(char str[], int n, FILE *infile);
```



- ❑ `fgets` reads characters from `infile` into `str`, until it reads '\n' or n-1 chars, whichever comes first.
- ❑ `fgets` inserts '`\0`' at end of `str`.
- ❑ Unlike `gets`, `fgets` reads the '`\n`' char into `str`.
- ❑ `fgets` returns the address of `str` as its result value.
- ❑ If `fgets` cannot read from `infile` (End-Of-File or some error) then it returns the `NULL` pointer.

File Output with `fputs`

- In addition, the `stdio` library provides the `fputs` function that works similar to `puts`.

```
int fputs(char str[], FILE *outfile);
```



- `fputs` outputs `str` to `outfile`.
- Unlike `puts`, `fputs` does not output an extra newline character to `outfile`.
- `fputs` returns `0` if the file operation is successful.
- Otherwise, it returns `-1` if it cannot write to `outfile`.

Example of **fgets** and **fputs**

```
#include <stdio.h>
#define L_SIZE 100    /* line size */
#define N_SIZE 40     /* file name size */

int main() {
    char line[L_SIZE], inname[N_SIZE], outname[N_SIZE];

    printf("Enter the name of input file: ");
    scanf("%s", inname);
    FILE *infile = fopen(inname, "r");
    if (infile == NULL) {
        printf("Can't open %s", inname);
        return 1;      /* terminate program */
    }

    printf("Enter the name of output file: ");
    scanf("%s", outname);
```

Example of `fgets` and `fputs`

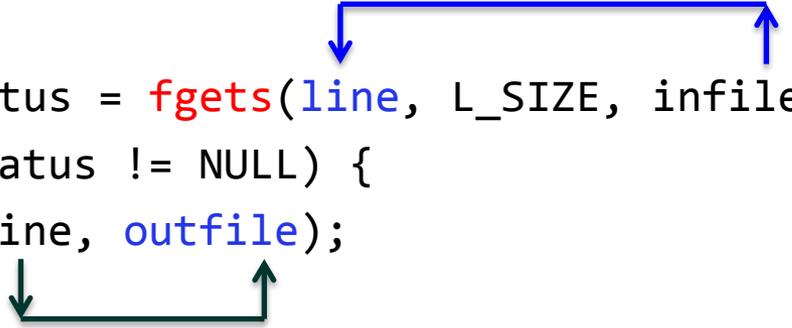
```
FILE *outfile = fopen(outname, "w");
if (outfile == NULL) {
    printf("Can't open %s", outname);
    return 1;          /* terminate program */
}

char *status = fgets(line, L_SIZE, infile);
while (status != NULL) {
    fputs(line, outfile);

    status = fgets(line, L_SIZE, infile);
}

fclose(infile);
fclose(outfile);

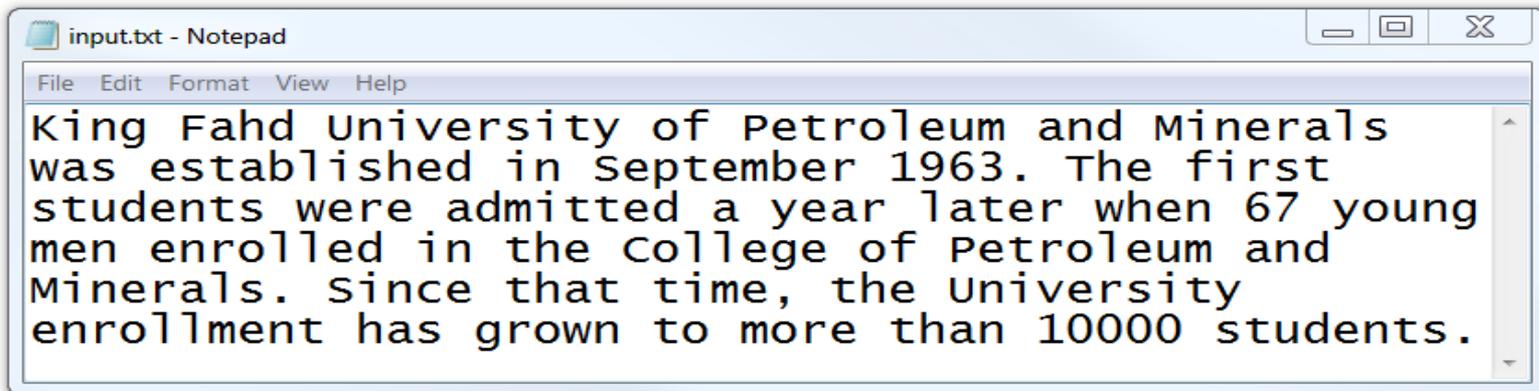
return 0;
}
```



Sample Run of the Previous Program

```
Enter the name of input file: input.txt
Enter the name of output file: copy.txt

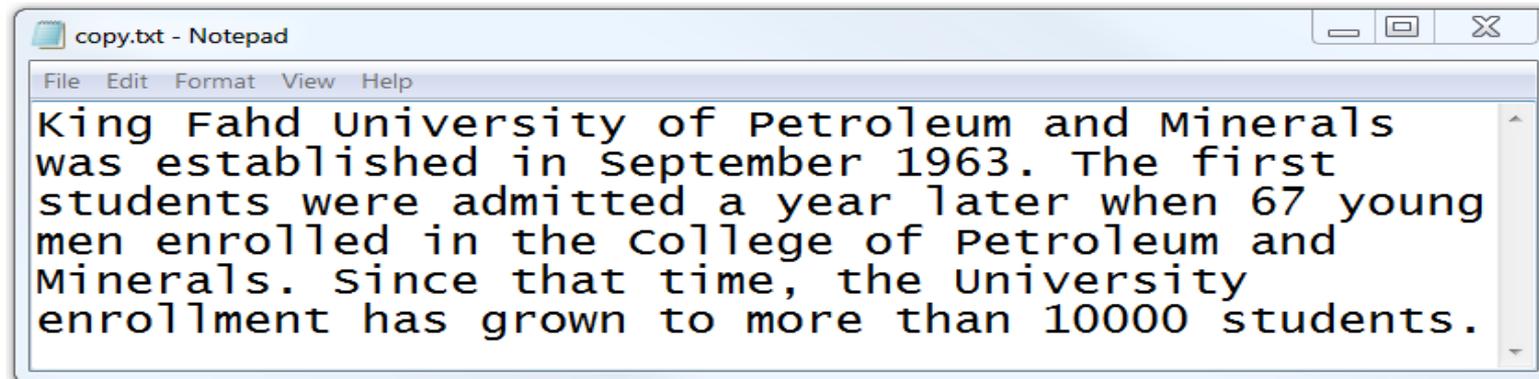
-----
Process exited with return value 0
Press any key to continue . . .
```



input.txt - Notepad

File Edit Format View Help

King Fahd University of Petroleum and Minerals was established in September 1963. The first students were admitted a year later when 67 young men enrolled in the college of Petroleum and Minerals. Since that time, the University enrollment has grown to more than 10000 students.



copy.txt - Notepad

File Edit Format View Help

King Fahd University of Petroleum and Minerals was established in September 1963. The first students were admitted a year later when 67 young men enrolled in the college of Petroleum and Minerals. Since that time, the University enrollment has grown to more than 10000 students.

Character Related Functions

- ❑ C provides functions that facilitate handling characters within a string.
- ❑ To use these functions `#include <ctype.h>`
- ❑ FYI, this part is not required for the exams.

Function	Description
<code>int isalnum(ch);</code>	True if <code>ch</code> is alphanumeric (letter or digit)
<code>int isalpha(ch);</code>	True if <code>ch</code> is alphabetic
<code>int isdigit(ch);</code>	True if <code>ch</code> is digit
<code>int isupper(ch);</code>	True if <code>ch</code> is uppercase letter
<code>int islower(ch);</code>	True if <code>ch</code> is lowercase letter
<code>int isspace(ch);</code>	True if <code>ch</code> is whitespace
<code>int iscntrl(ch);</code>	True if <code>ch</code> is a control character
<code>int ispunct(ch);</code>	True if <code>ch</code> is a punctuation character
<code>int toupper(ch);</code>	Convert <code>ch</code> to uppercase
<code>int tolower(ch);</code>	Convert <code>ch</code> to lowercase

Converting a String to Uppercase

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

int main() {
    char s[] = "ICS 103: Computer Programming in C";
    int i;

    for (i=0; s[i] != '\0'; i++)
        s[i] = toupper(s[i]); //converts the chs of s[] to upper chs.

    puts(s); //prints s[] to the screen

    printf("The digits in the string are: ");
    for (i=0; s[i] != '\0'; i++)
        if (isdigit(s[i])) printf("%c", s[i]); //prints only the digits

    printf("\n");
    return 0;
}
```

```
ICS 103: COMPUTER PROGRAMMING IN C
The digits in the string are: 103
```

Counting Letters, Digits, Spaces, Punctuations, Others

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

int main() {

    char line[100];
    int letters=0, digits=0, spaces=0, puncts=0, others=0;
    int i, total=0;

    printf("Type anything on the next line . . .\n");
    gets(line); // reads the input string into line

    for (i=0; line[i] != '\0'; i++) {
        total++; //count the # of ch in line (input string)
        if (isalpha(line[i])) letters++; //count letters in string
        else if (isdigit(line[i])) digits++; //count digits in string
```

Counting Letters, Digits, Spaces, Punctuations, Others

```
else if (isspace(line[i])) spaces++; //count spaces in string
else if (ispunct(line[i])) puncts++; //count puncts in string
else others++;
}

printf("\nYou typed %d chars\n", total);
printf("The count of letters = %d\n", letters);
printf("The count of digits = %d\n", digits);
printf("The count of spaces = %d\n", spaces);
printf("Punctuation chars = %d\n", puncts);
printf("Other characters = %d\n", others);
return 0;
}
```

Sample Run OF the Previous Program

```
Type anything on the next line . . .  
ICS 103 is interesting, but with ?!*&++ and :-(  
  
You typed 47 chars  
The count of letters = 26  
The count of digits  = 3  
The count of spaces  = 8  
Punctuation chars   = 10  
Other chars          = 0  
  
-----  
Process exited with return value 0  
Press any key to continue . . .
```

Counting **Vowels** in the input String

```
#include <stdio.h>

int isvowel(char ch); /* isvowel Function Prototype */

int main( )    {
    char line[100];
    int  i, vowels=0;

    printf("Type anything on the next line . . .\n");
    gets(line);

    for (i=0; line[i] != '\0'; i++)
        if (isvowel(line[i])) vowels++;

    printf("\nNumber of vowels = %d\n", vowels);
    return 0;
}
```

Function `isvowel`

```
/*isvowel function implementation, it returns true if character  
ch is a vowel */
```

```
int isvowel(char ch) {  
    return (ch == 'a' || ch == 'A' ||  
            ch == 'e' || ch == 'E' ||  
            ch == 'i' || ch == 'I' ||  
            ch == 'o' || ch == 'O' ||  
            ch == 'u' || ch == 'U') ;  
}
```

```
Type anything on the next line . . .  
This is a test line to count vowels "AEIOU"  
Number of vowels = 16
```

String Library Functions

- ❑ The standard C library contains useful string functions that allows us to manipulate strings.

- ❑ It can be used by including the following header file:

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

- ❑ Here, we look at few string library functions:

`strcpy`, `strlen`, `strcmp`, `strcat`, `strtok`, `strchr`, `strstr`

- ❑ The full list is available in appendix B OF THE TEXT BOOK

- ❑ The string library functions expects all strings to be **terminated with the null character '\0'**.

String Copy: `strcpy`

- We typically use `=` to assign a value into a variable.

```
char c, t[16], s[16] = "Example string";
```

```
c = 'a';      /* this is ok */
```

```
t = "Test string"; /* this does not work */
```

```
t = s;      /* this does not work */
```

- We can use `=` to initialize a string, but **not to assign**
- To assign a string, use the string copy function
- `strcpy` copies the `src` string into the `dest` string:

```
char *strcpy(char dest[], char src[]);
```

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



- `strcpy` copies all characters in the `src` string up to and including the **null char** into the `dest` string.
- `* src`: pointer to the source array where the content is to be taken.
- `* dest`: pointer to the destination array where the content is to be copied.

Examples: strcpy

```
char t[16], s[16] = "Example string";
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
E	x	a	m	p	l	e		s	t	r	i	n	g	\0	?

array s[16]

```
strcpy(t, "Test string");
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	e	s	t		s	t	r	i	n	g	\0	?	?	?	?

array t[16]

```
strcpy(t, s);
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
E	x	a	m	p	l	e		s	t	r	i	n	g	\0	?

array t[16]

Examples: strcpy

```
#include <stdio.h>
#include <string.h>
int main () {
char src[40];
char dest[100];
strcpy(src, "This is a tutorial for String copy");
strcpy(dest, src);
printf("Final copied string : %s\n", dest);
return(0);
}
```

Final copied string : This is a tutorial for String copy

String Length: `strlen`

❑ `strlen` counts the number of characters in a string that appear before the null character `'\0'`.

❑ Note, the length does not include the terminating null character.

```
int strlen(char s[]);
```

❑ The null character is **NOT** counted.

❑ The empty string `""` that starts with a null character has a `strlen` equal to `0`.

❑ Examples:

```
char s1[20] = "", s2[20] = "KFUPM, Dhahran"
```

```
int len1 = strlen(s1);    /* returns 0 */
```

```
int len2 = strlen(s2);    /* returns 14 */
```

```
#include <stdio.h>
#include <string.h>
int main () {
    char str[50];
    int len;
    strcpy(str, "This String is");
    len = strlen(str);
    printf("Length of |%s| is |%d|\n", str, len);
    return(0);
}
```

Length of |This is String is |14|

String Comparison: `strcmp`

- ❑ Characters are represented by numeric codes (ASCII codes).
- ❑ We can compare characters using relational operators.
- ❑ For example: if `(ch1 < ch2) { . . . }`
- ❑ However, if `str1` and `str2` are arrays of characters
- ❑ We cannot compare strings like this: `(str1 < str2)`
- ❑ To compare two strings, we use the `strcmp` function.

```
int strcmp(char str1[], char str2[]);
```

```
int strcmp(const char *str1, const char *str2);
```

- ❑ Compares the string pointed to, by `str1` to the string pointed to by `str2`.
- ❑ Compares the two strings alphabetically (ASCII codes)
 - Returns `0` if `str1` is equal to `str2`
 - Returns `<0` if `str1` is less than `str2`
 - Returns `>0` if `str1` is greater than `str2`

Examples: 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

Examples: strcmp

- Compares `str1` to `str2`, returns a value based on the first character they differ at:
 - **Less than 0**: if ASCII value of the character they differ at is smaller for `str1` or if `str1` starts the same as `str2` (and `str2` is longer).
 - `strcmp("Hello","hello")` -- returns value < 0
 - **Greater than 0**: if ASCII value of the character they differ at is larger for `str1` or if `str2` starts the same as `str1` (and `str1` is longer):
 - `strcmp("yello","hello")` -- returns value > 0
 - **0** if the two strings do not differ:
 - `strcmp("hello","hello")` -- returns 0
- ❑ `char s1[16] = "Long string";`
- ❑ `char s2[16] = "Short";`
- ❑ `char s3[16] = "short";`
- ❑ `char s4[16] = "";`
- `printf("%d ", strcmp(s1, s2));` //Returns -1 if s1 is less than s2
- `printf("%d ", strcmp(s2, s3));` //Returns -1 if s2 is less than s3
- `printf("%d ", strcmp(s3, s4));` //Returns 1 if s3 is greater than s4
- `printf("%d ", strcmp(s4, s4));` //Returns 0 if s4 is equal s4

String Comparison (**Continue**)

- If we want to ignoring case while comparing two strings:

```
int strcasecmp(char *str1, char *str2)
```

- Similar to **strcmp** except that upper and lower case characters (e.g., 'a' and 'A') are considered to be equal.

- Sometimes we only want to compare first **n** chars of two strings:

```
int strncmp(char *s1, char *s2, int n)
```

- ⇒ Works the same as **strcmp** except that it stops at the **n**th character.

- ⇒ If we want to ignoring case while comparing the **n**th character of two strings:

- ⇒ The version of **strncmp** that ignores case:

```
⇒ int strncasecmp(char *str1, char *str2, int n)
```

String Concatenation: `strcat`

- ❑ **Concatenation** means appending a source string (`src`) at the end of a destination string (`dest`) to make it longer.

```
char * strcat(char dest[], char src[]);
```

- ❑ The `src` string is copied at the end of the `dest` string.
- ❑ The **position of the null char** in the `dest` string is set after the appended copy of the `src` string.
- ❑ **Overflow** is possible if the `dest` string does not have sufficient space to append the `src` string.
- ❑ If **overflow** happens, **other variables can be overwritten, which might cause a runtime error.**
- ❑ Concatenate the first `n`th character of the source to the destination.
- ❑ `char *strncat(char *dstS, char *addS, int n)`
 - Appends the first `n` characters of `addS` to `dstS`.
 - If less than `n` characters in `addS`, only the characters in `addS` appended.
 - Always appends a `\0` character.

Example: `strcat`

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

int main(void) {
    char first[20], last[20], full[40];

    printf("Enter your first name: ");
    gets(first);
    printf("Enter your last name: ");
    gets(last);

    strcpy(full, first);
    strcat(full, " ");
    strcat(full, last);

    printf("Your full name is: ");
    puts(full);

    return 0;
}
```

String Tokenization: `strtok`

- ❑ **Tokenization** means splitting a string into parts called **tokens** based on a specified set of **delimiters** (`\n\t.;;,!?"`)

```
char * strtok(char str[], char delim[]);
```

```
char *strtok(char *str, const char *delim)
```

- ❑ Breaks string `str` into a series of tokens using the delimiter `delim`.
- ❑ The first call to `strtok` should have `str` point to the string to be tokenized.
- ❑ Subsequent calls to `strtok` must use `NULL` as `str`.
- ❑ The `strtok` function returns a pointer to the next token in `str` that ends with a delimiter in `delims`.
- ❑ It modifies `str` by replacing delimiters with `'\0'`
- ❑ It returns `NULL` when tokens are exhausted.

String Tokenization: `strtok`

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

int main() {
    char str[80] = "This is - www.kfupm.edu.sa - website";
    const char s[2] = "-";
    char *token; /* get the first token */
    token = strtok(str, s); /* walk through other tokens */
    while( token != NULL ) {
        printf( " %s\n", token );
        token = strtok(NULL, s);
    }
    return(0);
}
```

This is
www.kfupm.edu.sa.com
website

String Tokenization: `strtok`

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

int main () {
    char str[] = "- This, a sample string.";
    char * pch;
    printf ("Splitting string \"%s\" into tokens:\n",str);
    pch = strtok (str, " ,.-");
    while (pch != NULL) {
        printf ("%s\n",pch);
        pch = strtok (NULL, " ,.-");
    }
    return 0;
}
```

Splitting string "- This, a sample string." into tokens:
This
a
sample
string

Example: `strtok`

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

```
int main(void) {
    char date[20];
```

```
    printf("Enter a date like this: May 5, 2014\n> ");
    gets(date);
```

```
    char *month = strtok(date, " ,"); /* first call */
```

```
    char *day = strtok(NULL, " ,"); /* subsequent call */
```

```
    char *year = strtok(NULL, " ,"); /* subsequent call */
```

```
    puts(month);
```

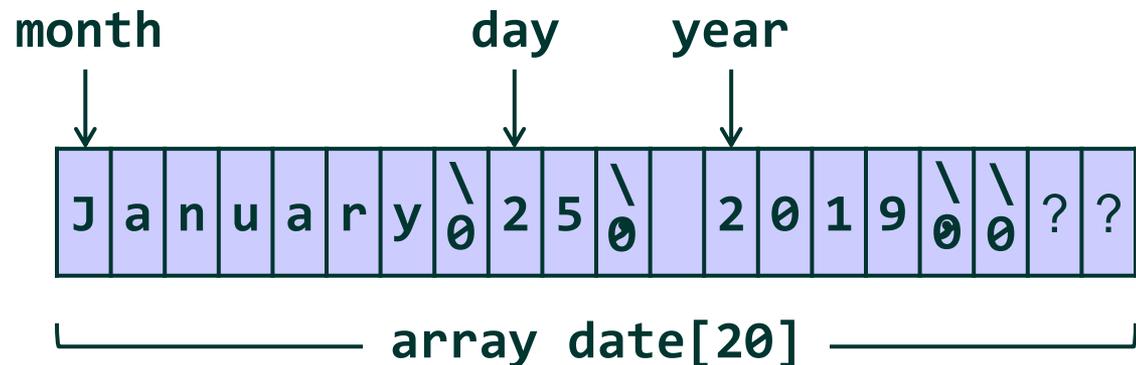
```
    puts(day);
```

```
    puts(year);
```

```
    return 0;
```

```
}
```

```
Enter a date like this: May 5, 2014
> January 25, 2019,
January
25
2019
```



We discussed

- ⇒ **strlen**: to determine the length of a string,
- ⇒ **strcpy**: to copy one string into the other
- ⇒ **strcmp**: to compare two strings
- ⇒ **strncmp**: compare first **n** chars of two strings
- ⇒ **strcat**: add one string to the end of another one.
- ⇒ **strncat**: Appends the first **n** characters of **addS** to **dstS**.
- ⇒ **strtok**: to tokenize/split a long string into tokens/words
- **In today's class: We will discuss:**
 - ⇒ **strstr**: to search for a string in a long string
 - ⇒ **strchr**: to search for a character in a string
- Code examples of how to use the above functions, i.e.
 - ⇒ How to **count the # of words in a file or the # of lower case characters in a file, etc.**
 - ⇒ Declaring Arrays of Strings and Arrays of Pointers
 - ⇒ Code examples of how to use the Arrays of Strings and Pointers
 - ⇒ **Open Discussion**

Searching a String

- ❑ Two functions for searching in a string:
- ❑ To search for a char in string, you can use:
 - `char * strchr(char str[], char target[]);`
 - `strchr` returns a pointer to the first occurrence of `target` char in `str`, or `NULL` if `target` is not found.
- ❑ To search for a string in a string, you can use:
 - `char * strstr(char str[], char target[]);`
 - `strstr` returns a pointer to the first occurrence of `target` string in `str`, or `NULL` if no match is found.

Example of `strstr`

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

int main(void) {

    char sentence[100], word[40], *result;

    printf("Enter a sentence: ");
    gets(sentence);
    printf("Enter a word to search: ");
    gets(word);

    result = strstr(sentence, word);
    if (result != NULL) printf("%s was found\n", word);
    else printf("%s was not found\n", word);

    return 0;
}
```

```
Enter a sentence: Searching a string
Enter a word to search: ching
ching was found
```

Example of `strchr`

```
#include <stdio.h>

#include <string.h>

int main () {

const char str[] = "This is just a String";

const char ch = 'u';

char *p; p = strchr(str, ch);

printf("String starting from %c is: %s", ch, p);

return 0;

}
```

String starting from u is: ust a String

Example: Remove non-Alphabet Characters in String

- ❑ This program takes a string from the user and stores it in the variable line.
- ❑ In the for loop, each character in the string is checked if it's an alphabet or not.
- ❑ If any character **inside a string is not a alphabet**, all characters after it including the null character is shifted by 1 position to the left.

```
#include<stdio.h>
#include<string.h>
int main() {
char line[150];
int i, j;
printf("Enter a string: ");
gets(line);
for(i = 0; line[i] != '\0'; ++i) {
    while (!( (line[i] >= 'a' && line[i] <= 'z') || (line[i] >= 'A'
&& line[i] <= 'Z') || line[i] == '\0') )
        {for(j = i; line[j] != '\0'; ++j) {
            line[j] = line[j+1]; }
        line[j] = '\0'; } }
printf("Output String: ");
puts(line);
return 0;
}
```

Enter a string: p2'r-o@gram84./
Output String: program

Example: Counts the # of words in a text file

```
/* Counts the number of Words in a text file if the delimiters are white space only*/
#include <stdio.h>
#include <stdlib.h>
//stdlib.h defines four variable types, macros, and various functions for performing general
    functions, i.e. NULL, EXIT_FAILURE, etc.
int main(void){
    FILE* infile;
    infile = fopen("input2.txt", "r");
    if(infile == NULL){
        printf("Error in opening input.txt\n");
        exit(1);
    }
    int wordCount = 0;
    char word[40];
    while(fscanf(infile, "%s", word)!= EOF)
    {
        wordCount++;
    }
    printf("Number of English words in the file = %d\n", wordCount);
    return 0;
}
```

Example: Counts the # lowercase letters in a text file

```
/* Counts the number of lowercase letters in a text file */
#include <stdio.h>
#include <stdlib.h>
int main(void){
    FILE* infile;
    infile = fopen("input2.txt", "r");
    if(infile == NULL){
        printf("Error in opening input.txt\n");
        exit(1);
    }
    int count = 0;
    char ch;
    while(fscanf(infile, "%c", &ch) != EOF){
        if(ch >= 'a' && ch <= 'z')
            count++;
    }
    printf("Number of lowercase letters in the file = %d\n", count);
    return 0;
}
```

Example: Counts the # lowercase letters in a String

```
#include <stdio.h>
int countLower(char string[]) { // can also use char *string instead of char string[ ]
    int k = 0, count = 0;
    while(string[k] != '\0'){
        if(string[k] >= 'a' && string[k] <= 'z'){
            count++;
        }
        k++;
    }
    return count;
}
int main(void){
    char str[ ] = "This Is the ICS Department";
    int count;
    count = countLower(str);
    printf("The count of lowercase letters is %d\n", count);
    char* ptr;
    ptr = str;
    count = countLower(ptr); // using pointer to char
    printf("The count of lowercase letters is %d\n", count);
    return 0;
}
```

More String Functions

<u>strdup</u> ()	Duplicates the string
<u>strlwr</u> ()	Converts string to lowercase
<u>strupr</u> ()	Converts string to uppercase
<u>strrev</u> ()	Reverses the given string
<u>strset</u> ()	Sets all character in a string to given character
<u>strnset</u> ()	It sets the portion of characters in a string to given character

Arrays of Strings

- ❑ A string is an array of characters.
- ❑ An array of strings is a **2D array** of characters.
- ❑ The **first dimension** represents the **number of strings**.
- ❑ The **second dimension** represents the **string length**.
- ❑ Example: **declare an array to store up to 30 names**, each of size **20 chars** (including null character).

```
#define MAX_NAMES 30
```

```
#define NAME_SIZE 20
```

```
...
```

```
char names[MAX_NAMES][NAME_SIZE];
```

Arrays of Pointers

- An array of pointers is a **1D Array** of **addresses**

```
char *ptr[30]; /* array of 30 pointers */
```

- **Initializing an array of strings:**

```
char month[12][10] = {"January", "February",  
    "March", "April", "May", "June", "July",  
    "August", "September", "October",  
    "November", "December" };
```

- **Initializing an array of pointers:**

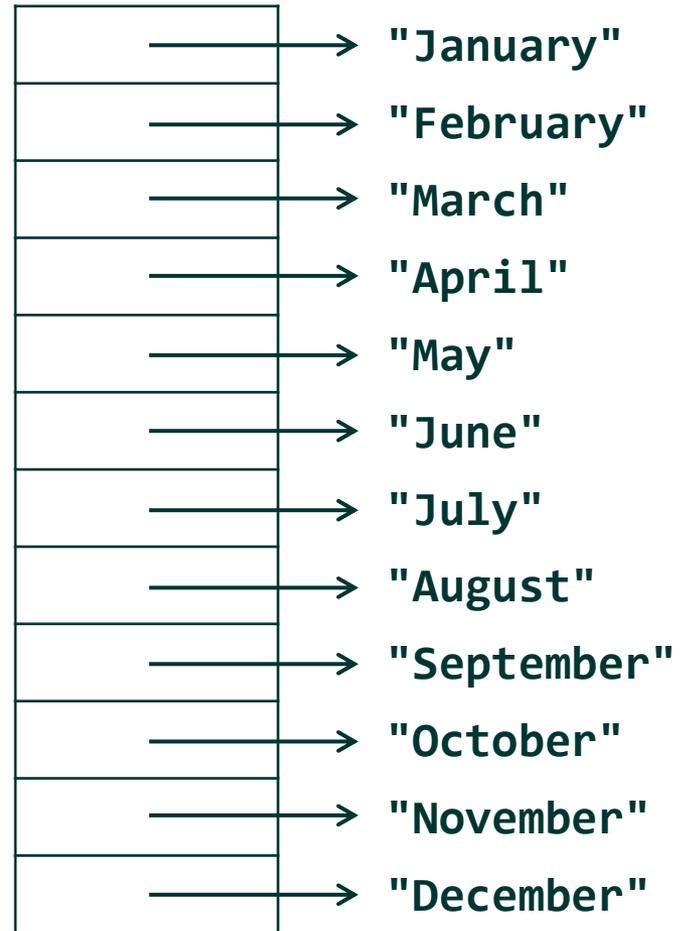
```
char *month[12] = { "January", "February",  
    "March", "April", "May", "June", "July",  
    "August", "September", "October",  
    "November", "December" };
```

Array of Strings **versus** Pointers

`char month[12][10]`

J	a	n	u	a	r	y	\0		
F	e	b	r	u	a	r	y	\0	
M	a	r	c	h	\0				
A	p	r	i	l	\0				
M	a	y	\0						
J	u	n	e	\0					
J	u	l	y	\0					
A	u	g	u	s	t	\0			
S	e	p	t	e	m	b	e	r	\0
O	c	t	o	b	e	r	\0		
N	o	v	e	m	b	e	r	\0	
D	e	c	e	m	b	e	r	\0	

`char *month[12]`



Sorting an Array of Names (1 of 5)

```
/* Sort an array of names alphabetically */
#include <stdio.h>
#include <string.h>

#define MAX_NAMES 30 /* maximum number of names */
#define NAME_SIZE 20 /* maximum name size */

/* read n names into array of strings */
void read_names(char array[][NAME_SIZE], int n);

/* sort an array of n names alphabetically */
void sort_names(char array[][NAME_SIZE], int n);

/* print an array of n names */
void print_names(char array[][NAME_SIZE], int n);
```

Main: Sorting an Array of Names (2 of 5)

```
/* main function */
int main() {
    int total;
    char name[MAX_NAMES][NAME_SIZE];

    printf("Enter total number of names: ");
    scanf("%d", &total);

    read_names(name, total);
    sort_names(name, total);
    printf("\nAlphabetical sorting of names\n\n");
    print_names(name, total);

    return 0;
}
```

read_names: Sorting an Array of Names (3 of 5)

```
/* read n names into array of strings */
void read_names(char array[][NAME_SIZE], int n) {
    int i;
    for (i=0; i<n; i++) {
        printf("Enter name[%d]: ", i);
        scanf("%s", array[i]);
    }
}
```

sort_names: Sorting an Array of Names (4 of 5)

```
void sort_names(char array[][NAME_SIZE], int n) {
    int fill, index_min, j;
    char temp_name[NAME_SIZE]; /* temporary name */
    for (fill=0; fill < n-1; fill++) {
        index_min = fill;
        for (j=fill+1; j<n; j++) {
            if (strcmp(array[j], array[index_min]) < 0)
                index_min = j; /* found a new min */
        }
        strcpy(temp_name, array[fill]);
        strcpy(array[fill], array[index_min]);
        strcpy(array[index_min], temp_name);
    }
}
```

`print_names`: Sorting an Array of Names (5 of 5)

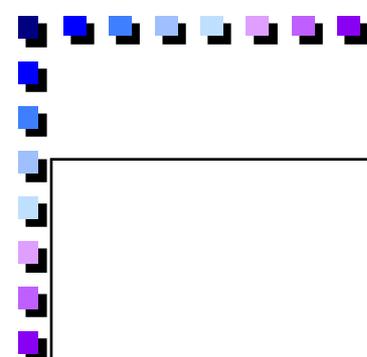
```
/* print an array of n names */  
void print_names(char array[][NAME_SIZE], int  
n) {  
    int i;  
    for (i=0; i<n; i++) puts(array[i]);  
}
```

Sample Run of the Previous Program

```
Enter total number of names: 5
Enter name[0]: Mohamed
Enter name[1]: Anwar
Enter name[2]: Ahmed
Enter name[3]: Maher
Enter name[4]: Abdulrahman

Alphabetical sorting of names

Abdulrahman
Ahmed
Anwar
Maher
Mohamed
```



The End!!

Thank you

Any Questions?

