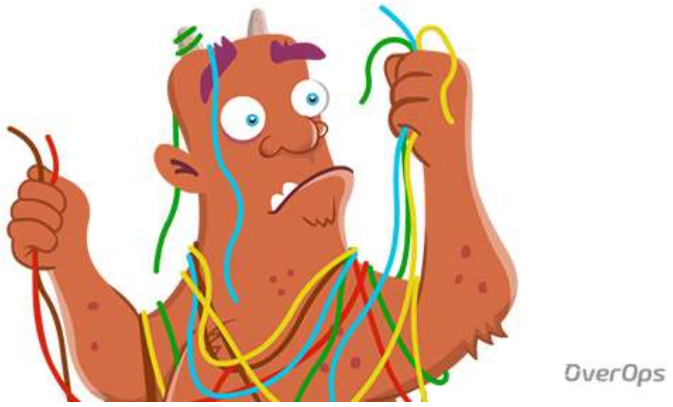


Strings

Mirza Mohammad Lutfe Elahi

Outline

- String Constants and Variables
- String Input and Output
- Character Related Functions
- String Library Functions
- Arrays of Strings and Arrays of Pointers

What is a String Constant?

- A sequence of characters enclosed in double quotes

  Example: `"Hello World"`

- Can be used in a **`printf`** statement:

```
printf("Average = %.2f\n", avg);
```

- Can also appear in **`#define`** directive, such as:

```
#define ERR_MSG "Error message: "
```

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	?	?	?	?

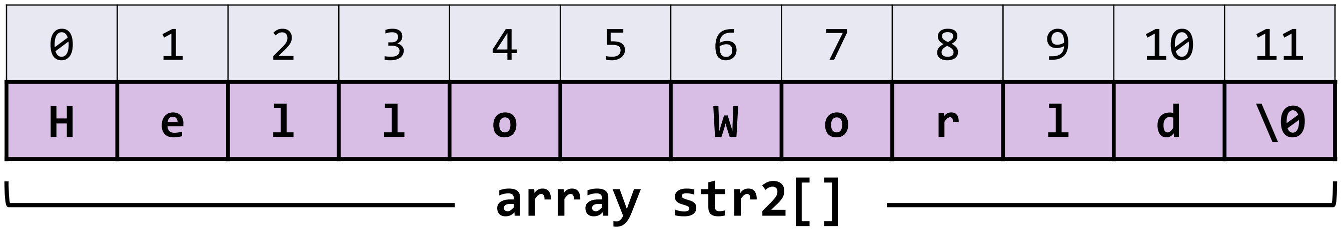
array str[16]

String Variables (Cont’d)

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

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

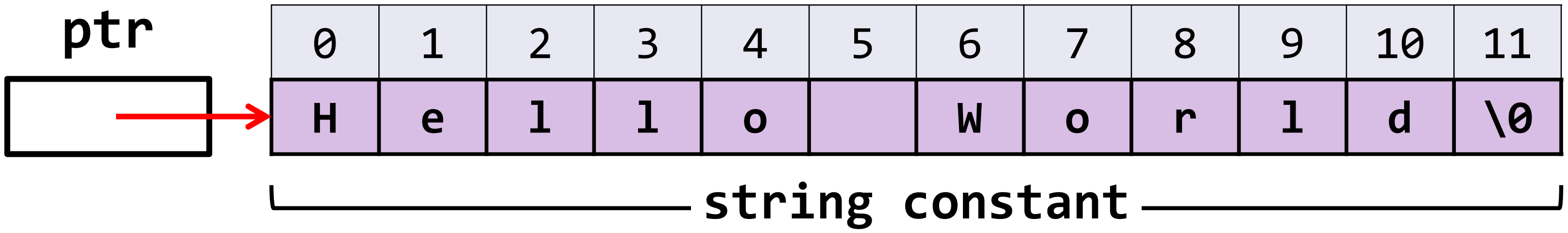
Only 12 characters are allocated (including `'\0'`)



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

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

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



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

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: Mirza
Hello Mirza
```

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

Example of String Input/Output

```c
#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: CSE 115 ST 940\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: CSE 115 ST 940

> CSE 215 MW 1120
CSE 215 meets MW at 1120
```

Placeholders Used with `printf`

Value	Placeholder	Output (□ is blank)

`'a'`	`%c`
	`%3c`
	`%-3c`
		`a`
		`□□a`
		`a□□`

`-10`	`%d`
	`%6d`
	`%-6d`
		`-10`
		`□□□-10`
		`-10□□□`

`49.76`	`%.3f`
	`%9.1f`
	`%9.2e`
		`49.760`
		`□□□□□49.8`
		`□4.98e+01`

`"fantastic"`	`%s`
	`%12s`
	`%-12s`
		`fantastic`
		`□□□fantastic`
		`fantastic□□□`

The `gets` and `puts` Functions

- A problem with **`scanf`** it 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, the **`gets`** function continues reading until the newline char (Enter key) is read.
- The **`'\n'`** character representing the Enter key 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.

Examples of gets and puts

```c
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
- 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     /* name size */

int main(void) {
  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`

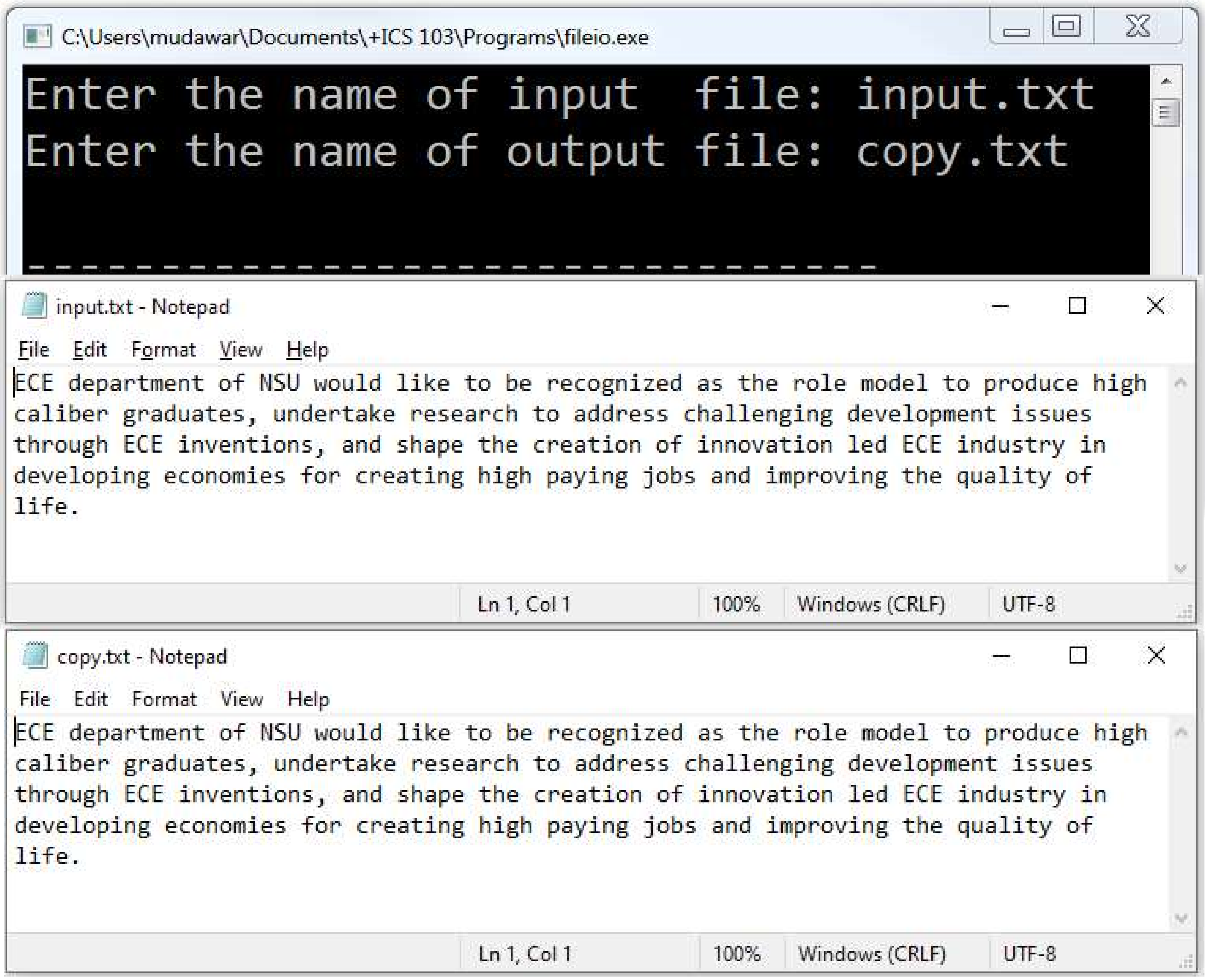
```
 FILE *outfile = fopen(outname, "w");
  if(outfile == NULL) {
    printf("Can't open %s", outname);
    return 1;          /* terminate program */
  }

  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…


C:\Users\mudawar\Documents\+ICS 103\Programs\fileio.exe

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

input.txt - Notepad
File Edit Format View Help

```
ECE department of NSU would like to be recognized as the role model to produce high
caliber graduates, undertake research to address challenging development issues
through ECE inventions, and shape the creation of innovation led ECE industry in
developing economies for creating high paying jobs and improving the quality of
life.
```

```
ECE department of NSU would like to be recognized as the role model to produce high
caliber graduates, undertake research to address challenging development issues
through ECE inventions, and shape the creation of innovation led ECE industry in
developing economies for creating high paying jobs and improving the quality of
life.
```

Character Related Functions

- In addition to the string library functions, C provides functions that facilitate character handling.
- To use these functions **`#include<ctype.h>`**

Function	Description
`int isalnum(char ch);`	true if ch is alphanumeric
`int isalpha(char ch);`	true if ch is alphabetic
`int isdigit(char ch);`	true if ch is digit
`int isupper(char ch);`	true if ch is uppercase letter
`int islower(char ch);`	true if ch is lowercase letter
`int isspace(char ch);`	true if ch is whitespace
`int iscntrl(char ch);`	true if ch is a control character
`int ispunct(char ch);`	true if ch is a punctuation character
`int toupper(char ch);`	convert ch to uppercase
`int tolower(char ch);`	convert ch to lowercase

Converting a String to Uppercase

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

int main(void) {
  char s[] = “CSE 115: Programming Language I";
  int i;

  for(i = 0; s[i] != '\0'; i++)
    s[i] = toupper(s[i]);

  puts(s);

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

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

```
CSE 115: PROGRAMMING LANGUAGE I
The digits in the string are: 115
```

Counting letters, Digits, Spaces, …

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

int main(void) {
  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);

  for(i = 0; line[i] !='\0'; i++) {
    total++;
    if(isalpha(line[i])) letters++;
    else if(isdigit(line[i])) digits++;
```

Counting letters, Digits, Spaces, …

```
        else if(isspace(line[i])) spaces++;
        else if(ispunct(line[i])) puncts++;
        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…

```
Type anything on the next line . . .
CSE 115 is interesting, but with ?!*&++ and :-(

You typed 47 characters
The count of the letters = 26
The count of the digits  = 3
The count of the spaces  = 8
Punctuation chars        = 10
Other chars              = 0
```

Counting Vowels

```
#include <stdio.h>

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

int main(void) {
  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`

```c
/* 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 the test line to count vowels “AEIOU”

Number of vowels = 16
```

String Library Functios

- The standard C library contains useful string functions
- 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
- The string library functions expects all strings to be terminated with the null character `'\0'`

String Copy: `strcpy`

- We typically use **=** to copy data 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[]);
```

`strcpy` copies all characters in the **`src`** string up to and including the null char into the **`dest`** string

Example: 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]

String Copy: `strlen`

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

```
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);
int len2 = strlen(s2);
```

String Copy: `strcmp`

- Characters are represented by numeric 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[]);
```

- Compares the two strings alphabetically (ASCII codes)
  - Returns `0` if `str1` is **equal** to `str2`
  - Returns `-1` if `str1` is **less than** `str2`
  - Returns `+1` if `str1` is **greater than** `str2`

Example: `strcmp`

```
char s1[16] = "Long string";
char s2[16] = "Short";
char s3[16] = "short";
char s4[16] = "";
printf("%d ", strcmp(s1, s2));
printf("%d ", strcmp(s2, s3));
printf("%d ", strcmp(s3, s4));
printf("%d ", strcmp(s4, s4));
```

```
-1 -1 1 0
```

String Copy: `strcat`

- Concatenation means appending a source string at the end of a destination string 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

Example: strcat

```c
#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;
}
```

```
Enter your first name: Mirza
Enter your first name: Elahi
Your full name is: Mirza Elahi
```

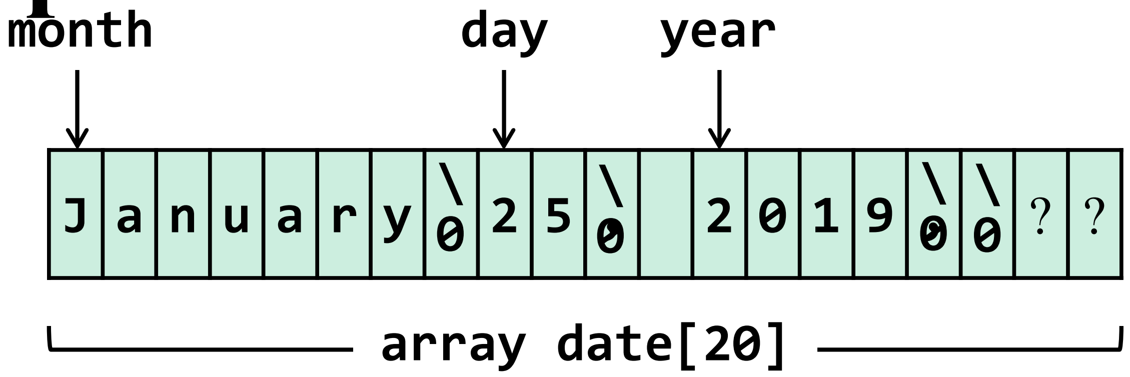
String Copy: `strtok`

- Tokenization means splitting a string into parts called tokens based on a specified set of delimiters.

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

- 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

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: March 30, 2020
> January 25, 2019
> January
> 25
> 2019
```

Searching a String

- Two functions for searching a string:

```
char * strchr(char str[], char target);

char * strstr(char str[], char target[]);
```

- **`strchr`** returns a pointer to the first occurrence of **`target`** char in **`str`**, or NULL if **`target`** is not found
- **`strstr`** returns a pointer to the first occurrence of **`target`** string in **`str`**, or NULL if no match is found

Example: strstr

```c
#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: test
test not found
```

Arrays of Strings

- An array of strings is a **2D array** of characters
- The first dimension represents the number of strings
- The second dimension represents the string itself
- 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:

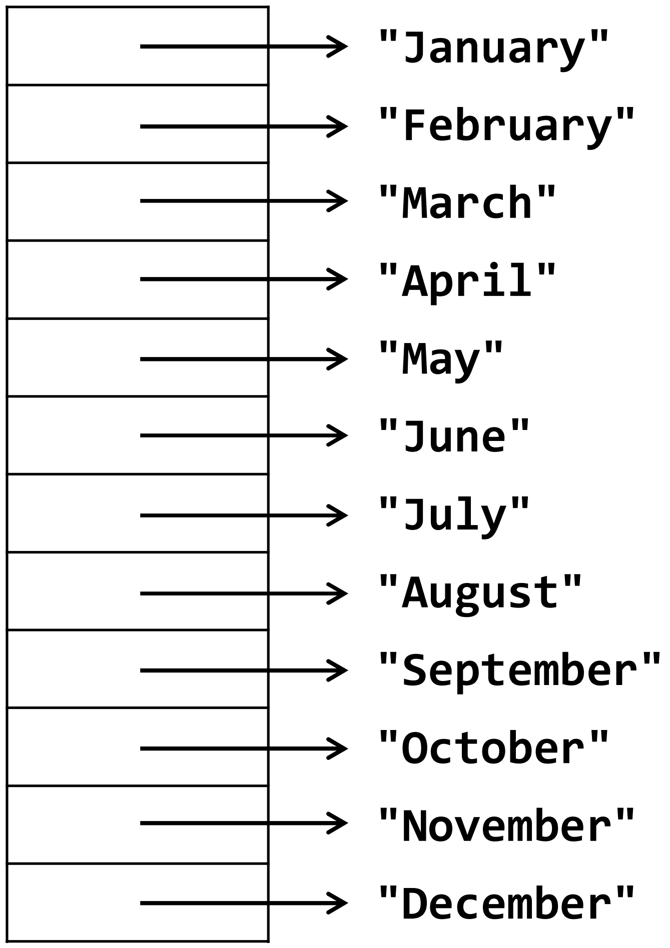
```
char *month[12] = { "January", "February",
  "March", "April", "May", "June", "July",
  "August", "September", "October",
  "November", "December" };
```

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 4)

```
/* 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);

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

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

Sorting an Array of Names (2 of 4)

```
/* main function */

int main(void) {
  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;
}
```

Sorting an Array of Names (3 of 4)

```
/* 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]);
  }
}

/* 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]);
}
```

Sorting an Array of Names (4 of 4)

```
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);
  }
}
```

Read an Array from a File

```
#include <stdio.h>
#define  SIZE 50    /* maximum array size */

int  read_file(const char filename[], double list[]);
void print_array(const double list[], int n);

int main(void) {
  double array[SIZE];
  int count = read_file("scores.txt", array);
  printf("Count of array elements = %d\n", count);
  print_array(array, count);

  return 0;
}
```

Sample Run

```
Enter total number of names: 5
Enter name[0]: January
Enter name[1]: February
Enter name[2]: March
Enter name[3]: April
Enter name[4]: May

Alphabetical sorting of names

April
February
January
March
May
```