Fpt University

TRƯỜNG ĐẠI HỌC FPT

Strings

Objectives

After studying this section, you should be able to:

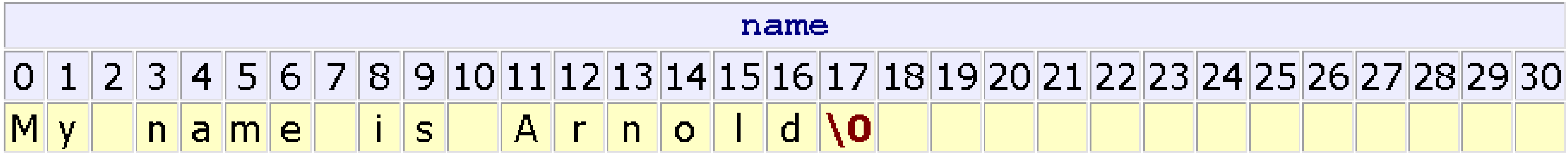
- The way is used to store a string of characters in C.
- How to declare/initialize a string in C?
- How to access a character in a string?
- What are operations on strings
  - Input/output (**stdio.h**)
  - Some common used functions in the library **string.h**
- How to manage an array of strings?

Contents

- Null-String/C-String
- Declare/Initialize a string
- Data stored in a string
- Output a String
- Input a string
- May Operators Applied to String?
- Other String Functions
- Array of Strings

1. Null-String/ C-String

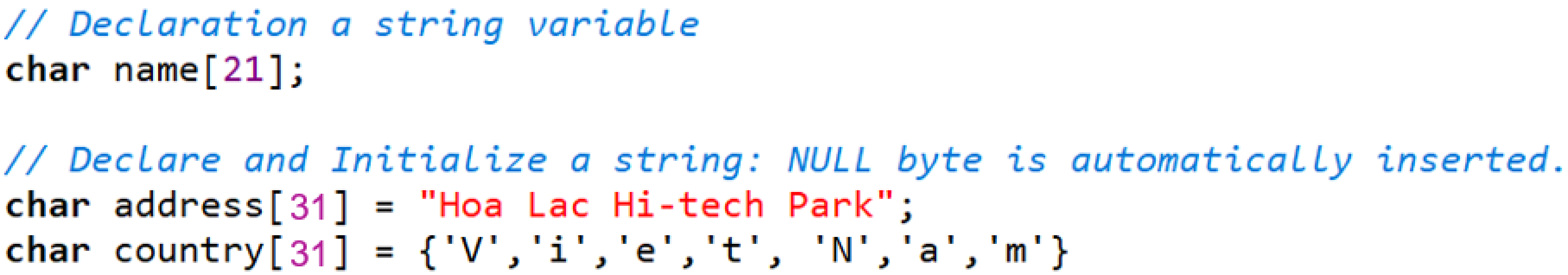
- A string is a group of characters. It is similar to an array of characters.
- A NULL byte (value of 0 - escape sequence **‘\0’**) is inserted to the end of a string. It is called **NULL-string** or **C-string**.
- A string is similar to an array of characters. The difference between them is at the end of a string, a NULL byte is inserted to locate the last meaningful element in a string.
- If a string with the length **n** is needed, declare it with the length **n+1**.



name																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
M	y		n	a	m	e		i	s		A	r	n	o	l	d	\0													

2. Declare/ Initialize a String

- **Static strings**: stored in data segment or stack segment. Compiler can determine the location for storing strings.
- Example:



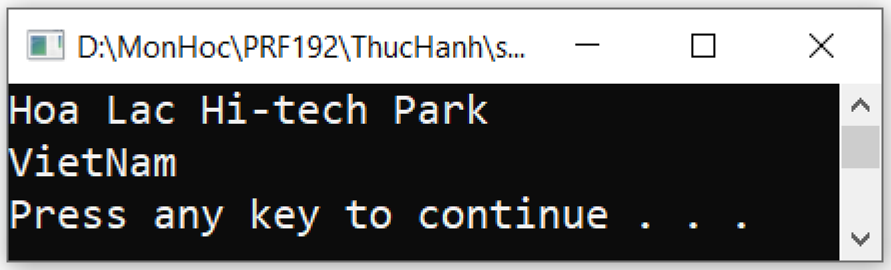
```
// Declaration a string variable
char name[21];

// Declare and Initialize a string: NULL byte is automatically inserted.
char address[31] = "Hoa Lac Hi-tech Park";
char country[31] = {'V','i','e','t', 'N','a','m'}
```

address																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
H	o	a		L	a	c		H	i	-	t	e	c	h		P	a	r	k	**\0**										

country																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
V	i	e	t	N	a	m	**\0**																							

Static Strings: Example

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

int main(){
    // Declaration a string variable
    char name[21];

    // Declare and Initialize a string: NULL byte is automatically inserted.
    char address[30] = "Hoa Lac Hi-tech Park";
    char country[30] = {'V','i','e','t', 'N','a','m', '\0'};

    int i;
    // Print out: address
    for(i=0; address[i]!='\0'; i++){
        printf("%c", address[i]);
    }
    printf("\n");
    // Print out: country
    for(i=0; country[i]!='\0'; i++){
        printf("%c", country[i]);
    }
    printf("\n");

    system("pause");
    return 0;
}
```



```
Hoa Lac Hi-tech Park
VietNam
Press any key to continue . . .
```

2. Declare/ Initialize a String (cont.)

- **Dynamic strings**: Stored in the heap
- Syntax:

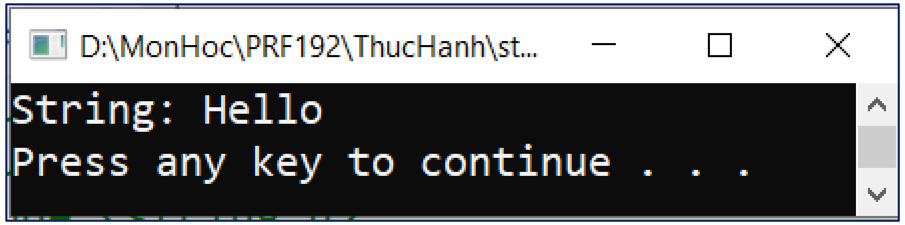
```
char *str = (char *)malloc(length * sizeof(char));
or:
char *str = (char *)calloc(length, sizeof(char));
```

- Note: Using malloc(…) and calloc(…) funtions in <**stdlib.h**>

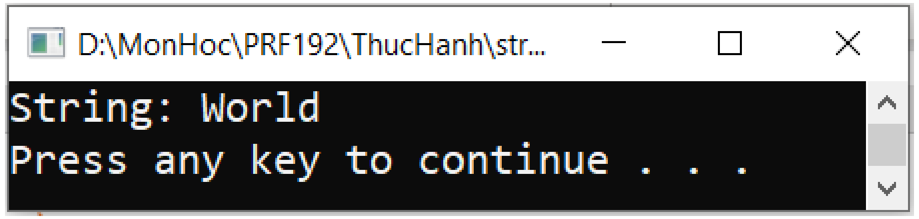
Dynamic Strings: Example

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

int main() {
    // Allocate memory for a string of 10 characters
    int length = 10;
    char *str = (char *)malloc(length * sizeof(char));

    // Directly set it as a string
    strcpy(str, "Hello");
    printf("String: %s\n", str);

    // Free the allocated memory
    free(str);

    system("pause");
    return 0;
}
```

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

int main() {
    // Allocate memory for a string of 10 characters
    int length = 10;
    char *str = (char *)calloc(length, sizeof(char));

    // Directly set it as a string
    strcpy(str, "World");
    printf("String: %s\n", str);

    // Free the allocated memory
    free(str);

    system("pause");
    return 0;
}
```



```
String: Hello
Press any key to continue . . .
```

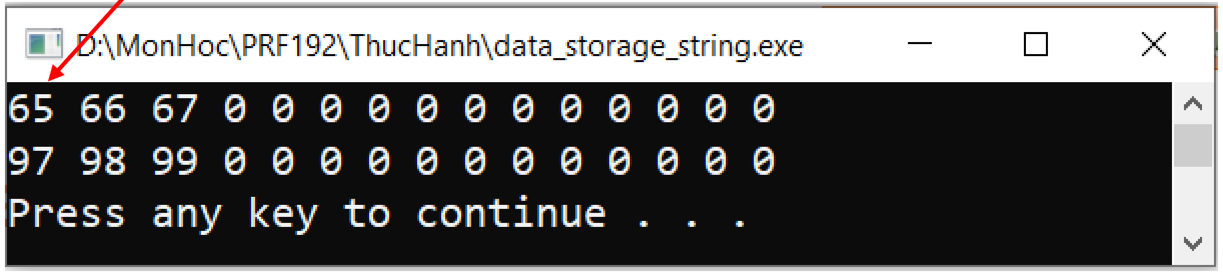


```
String: World
Press any key to continue . . .
```

3. Data Stored in a strings

- Each character in a string is stored as it's ASCII code.

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

int main(){
    // Declare and Initialize two Strings
    char s1[15] = "ABC";
    char s2[15] = {'a', 'b', 'c', '\0'};

    int i;
    // Print out data storage in s1
    for(i=0; i<15; i++){
        printf("%d ", s1[i]);
    }
    printf("\n");
    // Print out data storage in s2
    for(i=0; i<15; i++){
        printf("%d ", s2[i]);
    }
    printf("\n");

    system("pause");
    return 0;
}
```

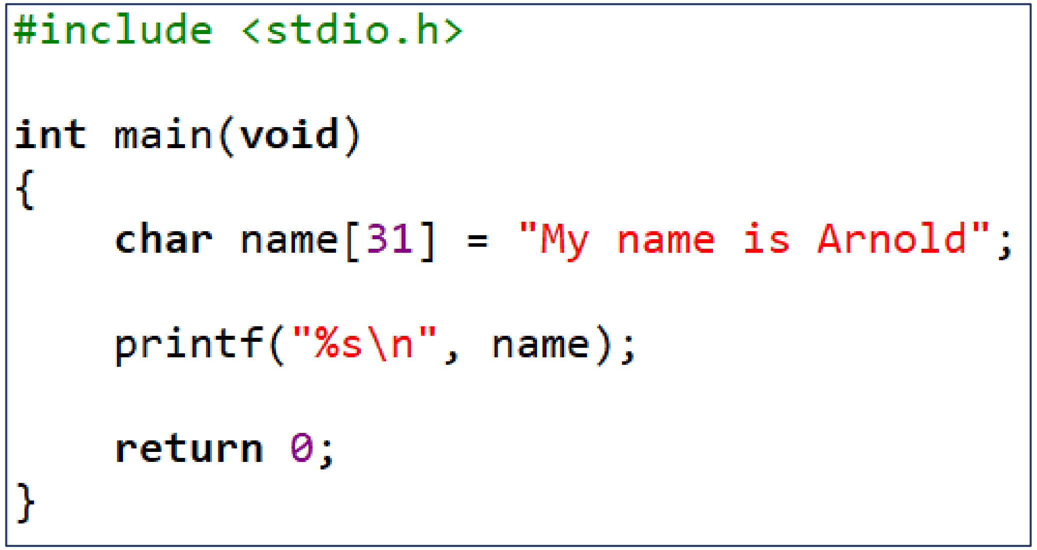


```
65 66 67 0 0 0 0 0 0 0 0 0 0 0 0
97 98 99 0 0 0 0 0 0 0 0 0 0 0 0
Press any key to continue . . .
```

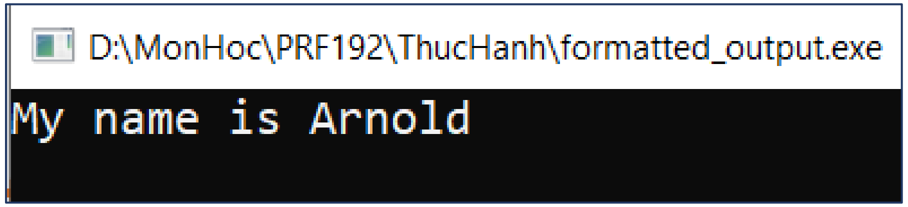
4. Output Strings

Formatted Output:

- **printf()** library functions support the **%s** conversion specifier for character string output
- printf() displays all of the characters from the address provided up to but excluding the null terminator byte. For example:



```
#include <stdio.h>

int main(void)
{
    char name[31] = "My name is Arnold";

    printf("%s\n", name);

    return 0;
}
```



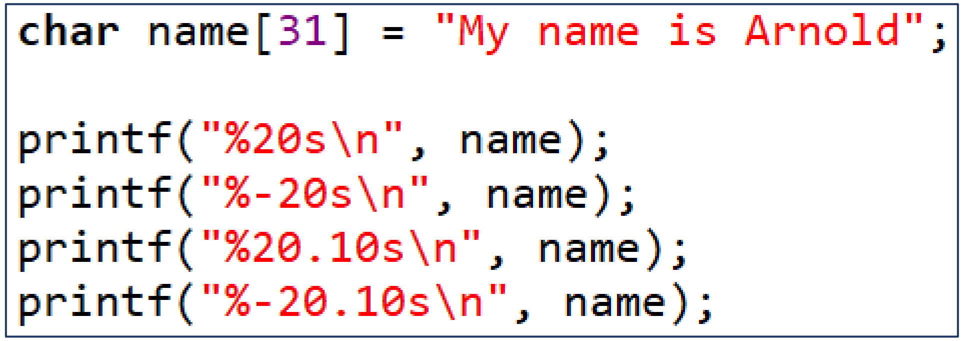
D:\MonHoc\PRF192\ThucHanh\formatted_output.exe

```
My name is Arnold
```

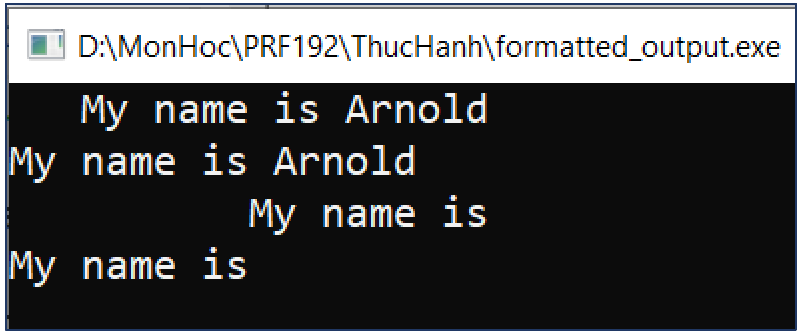
Formatted Output (cont.)

Qualifiers

- Qualifiers on the **%s** specifier add detail control:
  - **%20s** displays a string right-justified in a field of 20
  - **%-20s** displays a string left-justified in a field of 20
  - **%20.10s** displays the first 10 characters of a string right-justified in a field of 20
  - **%-20.10s** displays the first 10 characters of a string left-justified in a field of 20
- For Example:



```c
char name[31] = "My name is Arnold";

printf("%20s\n", name);
printf("%-20s\n", name);
printf("%20.10s\n", name);
printf("%-20.10s\n", name);
```



D:\MonHoc\PRF192\ThucHanh\formatted_output.exe

```
   My name is Arnold
My name is Arnold
          My name is
My name is
```

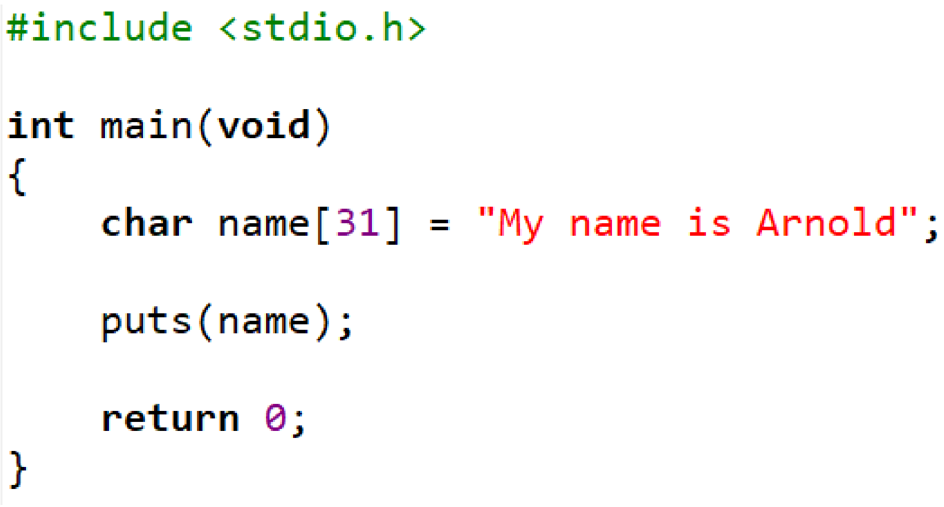
4. Output Strings (cont.)

Unformatted Output:

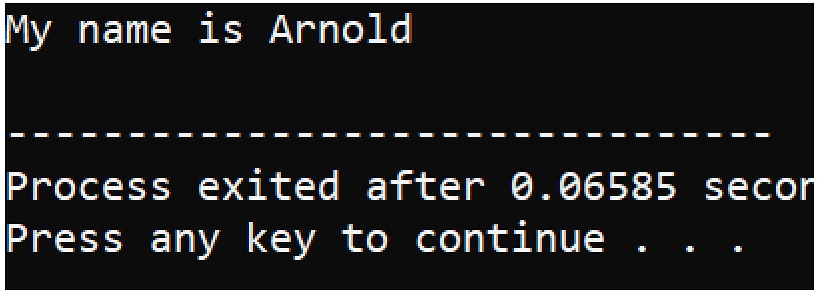
- **puts()** library function outputs a character string to the standard
- Prototype:

```
int puts(const char *);
```

- Example:



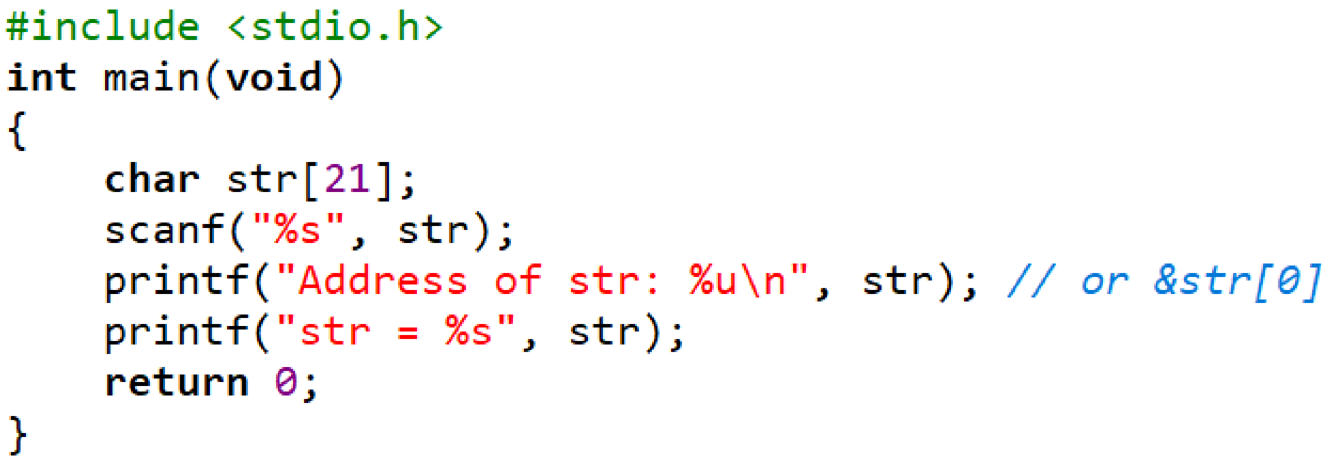
```
#include <stdio.h>

int main(void)
{
    char name[31] = "My name is Arnold";

    puts(name);

    return 0;
}
```



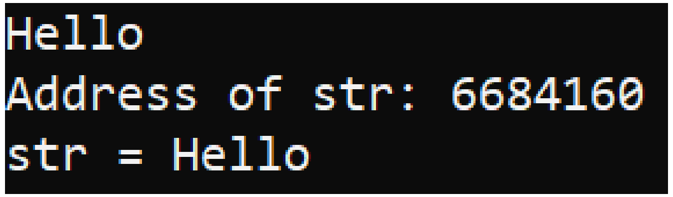
```
My name is Arnold

--------------------------------
Process exited after 0.06585 secor
Press any key to continue . . .
```

5. Input Strings: Using scanf(…) function

- The **scanf()** library function support conversion specifiers particularly designed for character string input. These specifiers are:
  - **%s** - whitespace delimited set.
  - **%[ ]** - rule delimited set
- The corresponding argument for these specifiers is the address of the string to be populated from the input stream. For example:



```
#include <stdio.h>
int main(void)
{
    char str[21];
    scanf("%s", str);
    printf("Address of str: %u\n", str); // or &str[0]
    printf("str = %s", str);
    return 0;
}
```



```
Hello
Address of str: 6684160
str = Hello
```

scanf(…): %s conversion specifier

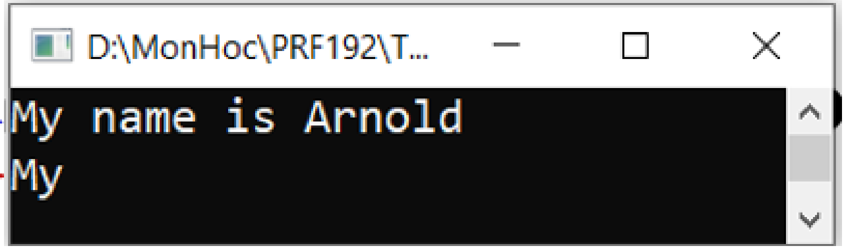
%s conversion specifier:

- Reads all characters until the first whitespace character
- Stores the characters read in the char array identified by the corresponding argument
- Stores the null terminator in the char array after accepting the last character
- Leaves the delimiting whitespace character and any subsequent characters in the input buffer

scanf(…): %s conversion specifier (cont.)

- **Example 1**:

```
#include <stdio.h>
int main(void)
{
        char name[32];
        scanf("%s", name);
        printf("%s", name);
        return 0;
}
```



```
My name is Arnold
My
```

- The **scanf()** function will stop accepting input after the character ‘**y**’ and stores:

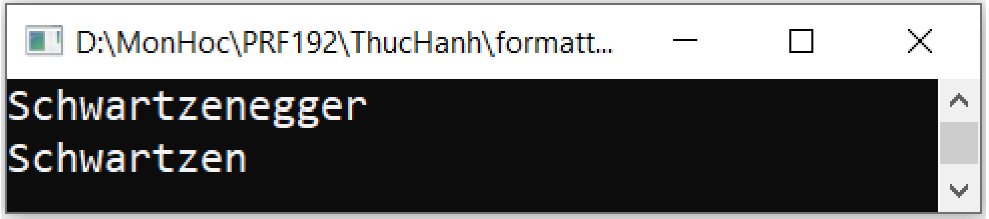
name																															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
M	y	**\0**																													

- The characters ' name is Arnold' remain in the input buffer.

scanf(…): %s conversion specifier (cont.)

- **Example 2**:

```c
#include <stdio.h>
int main(void)
{
    char name[31];
    scanf("%10s", name);
    printf("%s", name);
    return 0;
}
```



```
Schwartzenegger
Schwartzen
```

- The **scanf()** function will stop accepting input after the character ‘**n**’ and stores:

name																															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S	c	h	w	a	r	t	z	e	n	**/0**																					

- The characters ' name is Arnold' remain in the input buffer.

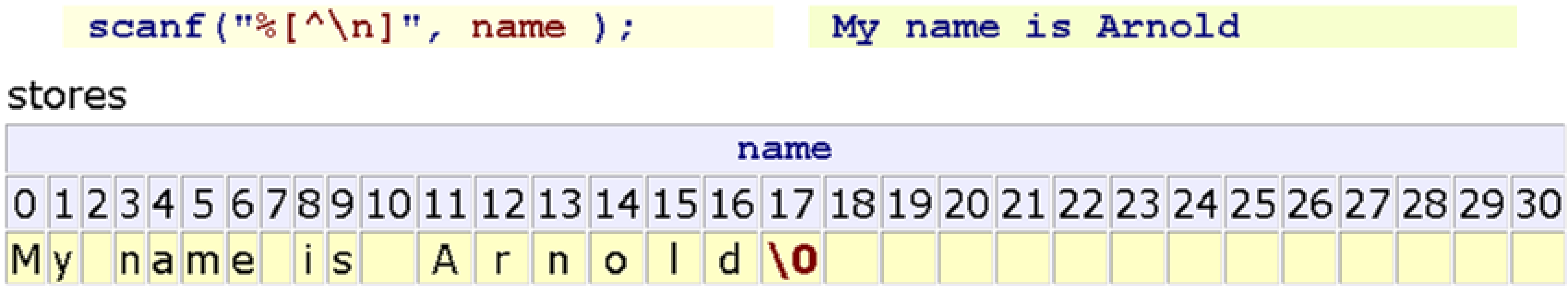
scanf(…): %[^\n] conversion specifier

How to accept blanks in a input string?

- **%[^\n]** conversion specifier:
  - Reads all characters until the **newline** (**'\n’**)
  - Stores the characters read in memory locations starting with the address passed to **scanf**
  - Stores the null byte in the byte following that where **scanf** stored the last character
  - Leaves the delimiting character (here, **'\n'**) in the input buffer

%[^\n] conversion specifier: Example

- Example 1:



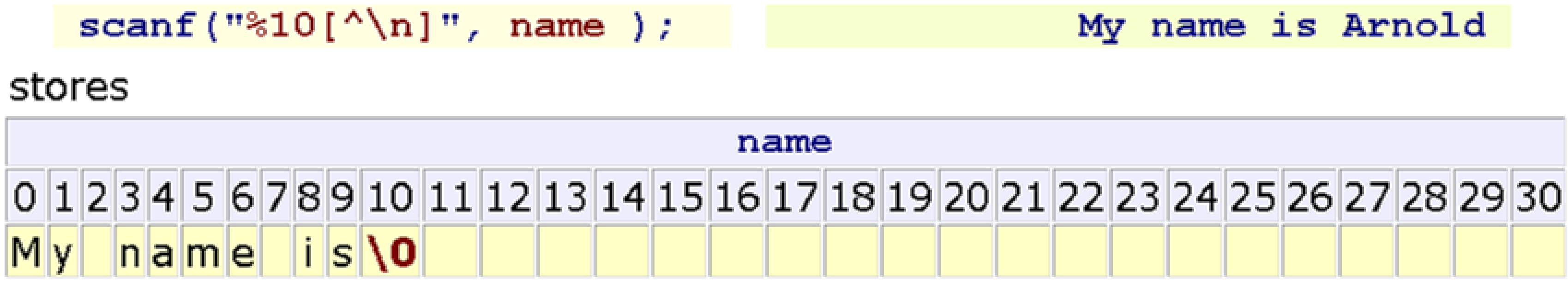
```
scanf("%[^\n]", name );
```

```
My name is Arnold
```

stores

name																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
M	y		n	a	m	e		i	s		A	r	n	o	l	d	\0													

- Example 2:



```
scanf("%10[^\n]", name );
```

```
            My name is Arnold
```

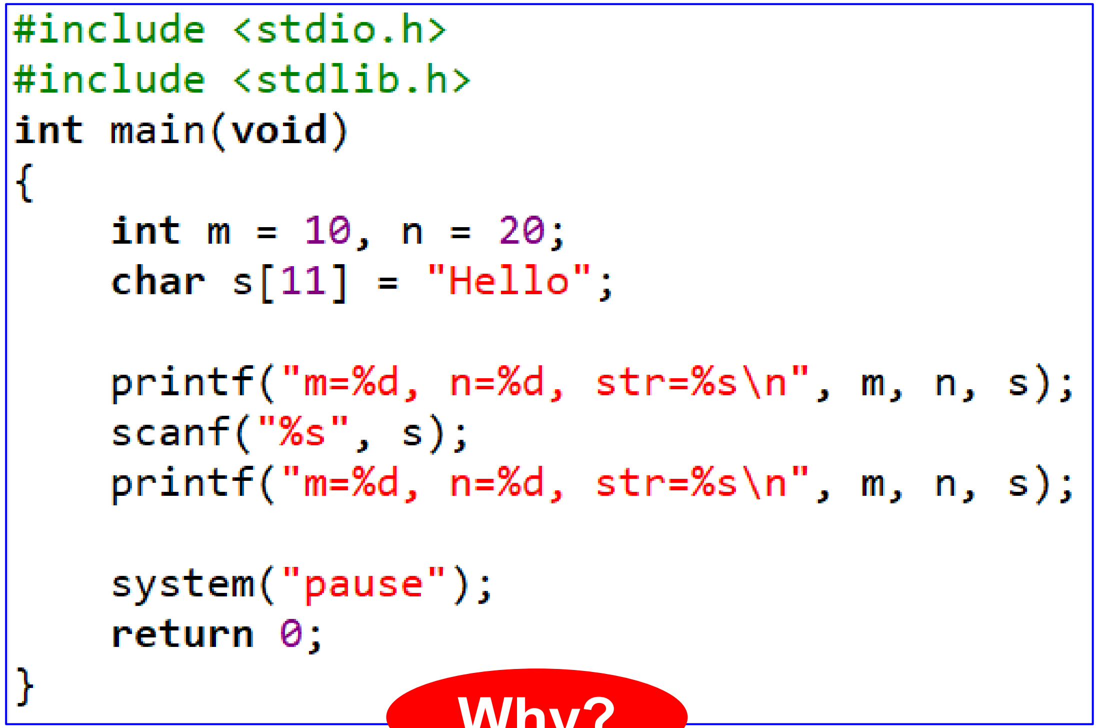
stores

name																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
M	y		n	a	m	e		i	s	\0																				

Exercise 1: Input Strings

- Compile & Run program
- Explain the results of two test cases
- Replace and Re-run:

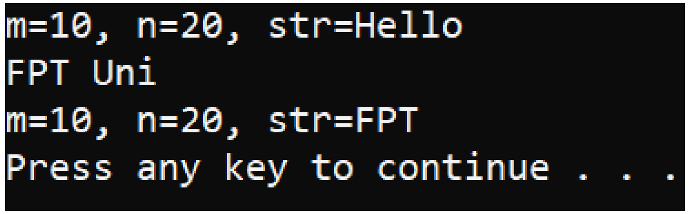
scanf(“%s”, S) → scanf(“%10[^\n]”, S)



```c
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int m = 10, n = 20;
    char s[11] = "Hello";

    printf("m=%d, n=%d, str=%s\n", m, n, s);
    scanf("%s", s);
    printf("m=%d, n=%d, str=%s\n", m, n, s);

    system("pause");
    return 0;
}
```

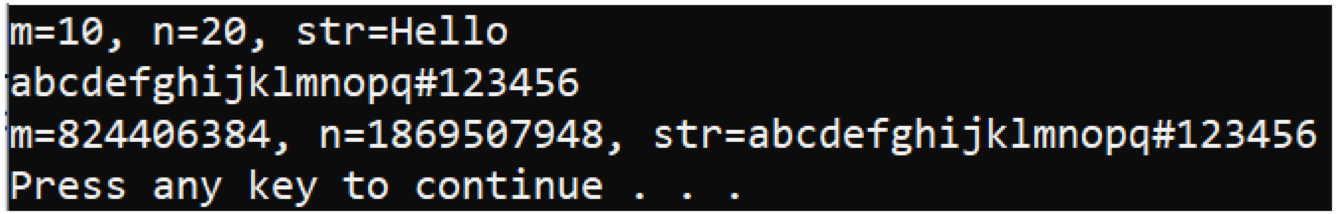
Why?

Test case 1



```
m=10, n=20, str=Hello
FPT Uni
m=10, n=20, str=FPT
Press any key to continue . . .
```

Test case 2



```
m=10, n=20, str=Hello
abcdefghijklmnopq#123456
m=824406384, n=1869507948, str=abcdefghijklmnopq#123456
Press any key to continue . . .
```

scanf(…) - cont.

- Some character specifiers used in the function **scanf()**: Set of character are or not accepted.

Specifier	Description
%[abcd]	Searches the input field for any of the characters a, b, c, and d
%[^abcd]	Searches the input field for any characters **except** a, b, c, and d
%[0-9]	To catch all decimal digits
%[A-Z]	Catches all uppercase letters
%[0-9A-Za-z]	Catches all decimal digits and all letters
%[A-FT-Z]	Catches all uppercase letters from A to F and from T to Z

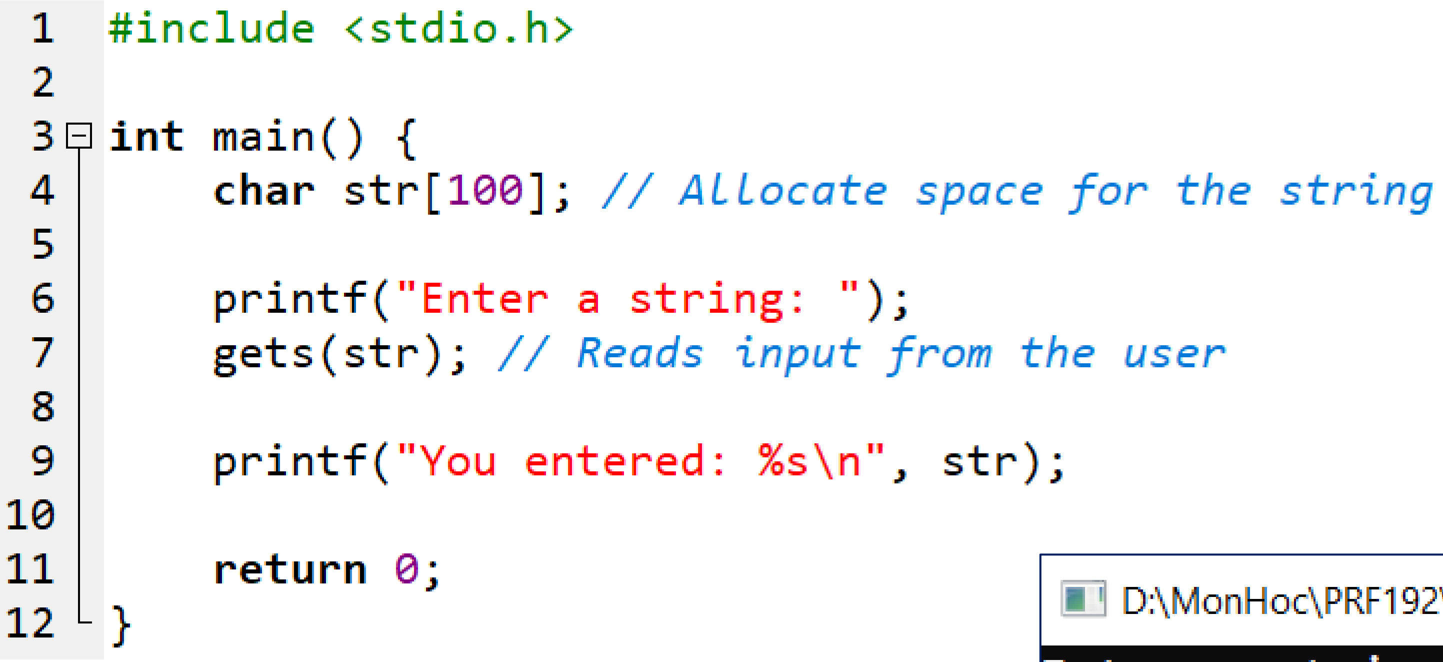
5. Input Strings: Using gets(…) function

- **gets** is a standard library function (stdio.h) that:
  - Accepts an empty string
  - Uses the **'\n'** as the delimiter
  - Throws away the delimiter after accepting the string
  - Automatically appends the null byte to the end of the set stored.
- The prototype for **gets** is:

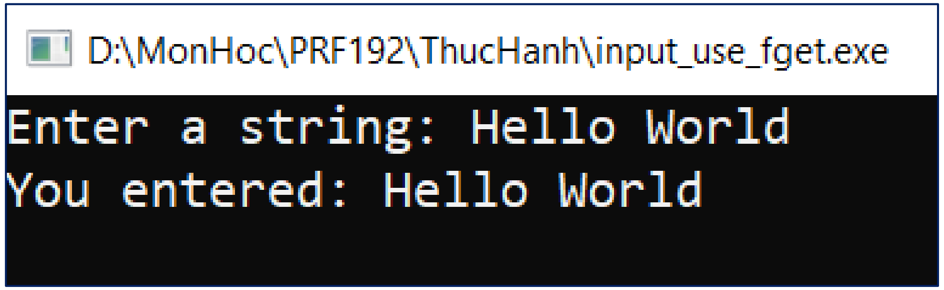
```
char* gets(char [ ]);
```

- **Warning**: **gets** is unsafe. Because it does not check the size of the buffer and can lead to buffer overflows.

gets(…) function: Example



```c
#include <stdio.h>

int main() {
    char str[100]; // Allocate space for the string

    printf("Enter a string: ");
    gets(str); // Reads input from the user

    printf("You entered: %s\n", str);

    return 0;
}
```



D:\MonHoc\PRF192\ThucHanh\input_use_fget.exe

```
Enter a string: Hello World
You entered: Hello World
```

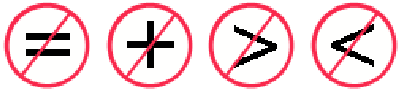
Exercise 2: Input Strings

- Using the hints below, write a program that takes in a string of characters and prints it out.
- *Hint*:

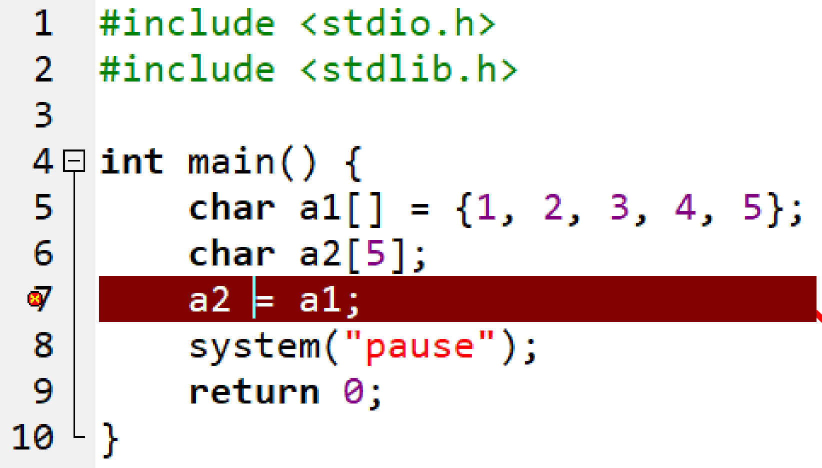
```
/* getstr accepts a newline terminated string s of up
*  to max characters, appends a null byte and throws
*  away the terminating character
*/
void getstr(char s[], int max) {
    int i, c;

    i = 0;
    while((c = getchar()) != '\n' && c != EOF)
        if (i < max)
            s[i++] = (char) c;
    s[i] = '\0';
}
```

6. May Operators Applied to String?

- C operators act on primitive data type only (char, int, float, …)
- Can not be applied to static arrays and static strings.

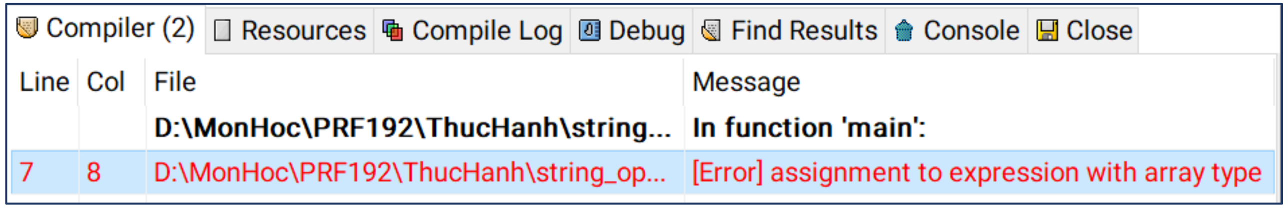


- Example:



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

int main() {
    char a1[] = {1, 2, 3, 4, 5};
    char a2[5];
    a2 = a1;
    system("pause");
    return 0;
}
```

We need functions for processing arrays and string



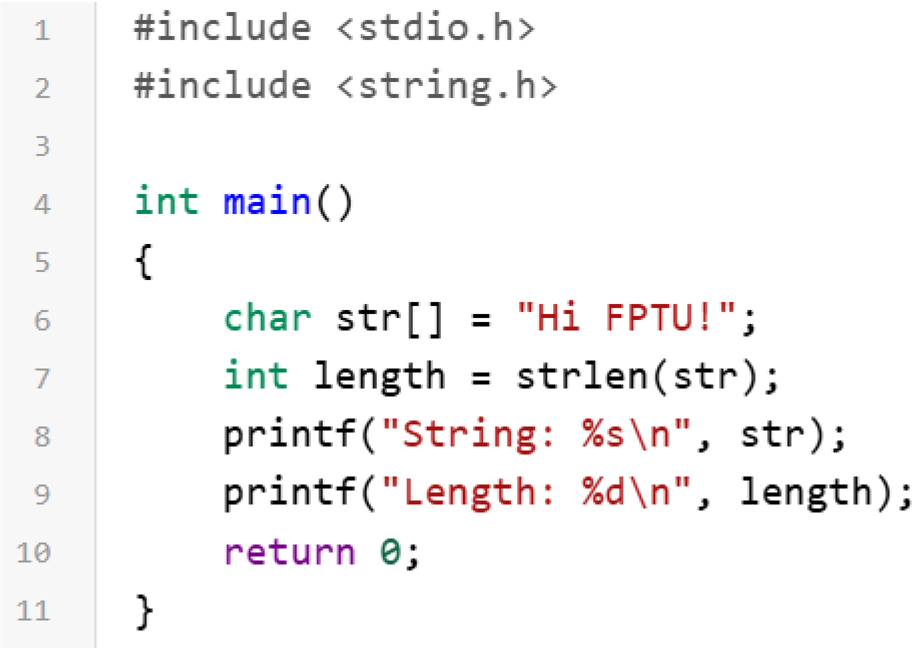
Line	Col	File	Message
		D:\MonHoc\PRF192\ThucHanh\string...	In function 'main':
7	8	D:\MonHoc\PRF192\ThucHanh\string_op...	[Error] assignment to expression with array type

String Functions: string.h

Common Funtions	Description
strlen()	Get the length of a string
strcpy()	Copy source string to destination string
strcmp()	Compare two strings
strcat()	Concatenate string src to the end of dest
strupr()	Convert a string to uppercase
strlwr()	Convert a string to lowercase
strstr()	Find the address of a substring
strtok()	Breaks string str into a series of tokens separated by delim

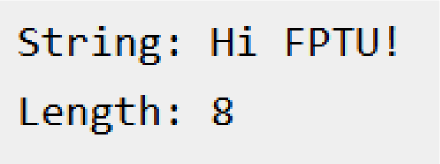
strlen() function

- The strlen() function calculates the length of a given string. It doesn't count the null character **'\0'.**
- **Syntax:** int **strlen**(const char *str);
- Example:



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

int main()
{
    char str[] = "Hi FPTU!";
    int length = strlen(str);
    printf("String: %s\n", str);
    printf("Length: %d\n", length);
    return 0;
}
```

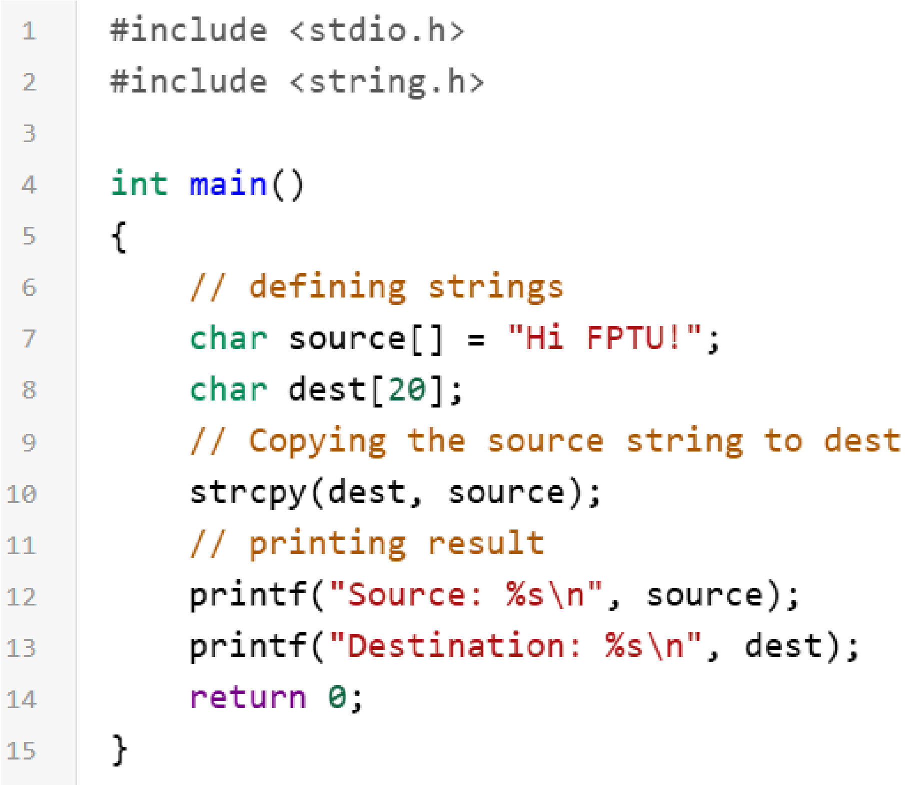
Output:



```
String: Hi FPTU!
Length: 8
```

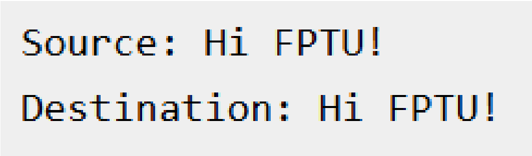
strcpy() function

- The strcpy() is a function used to copy one string to another
- **Syntax:** char* **strcpy**(char* dest, const char* src);
- Example:



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

int main()
{
    // defining strings
    char source[] = "Hi FPTU!";
    char dest[20];
    // Copying the source string to dest
    strcpy(dest, source);
    // printing result
    printf("Source: %s\n", source);
    printf("Destination: %s\n", dest);
    return 0;
}
```

Output:



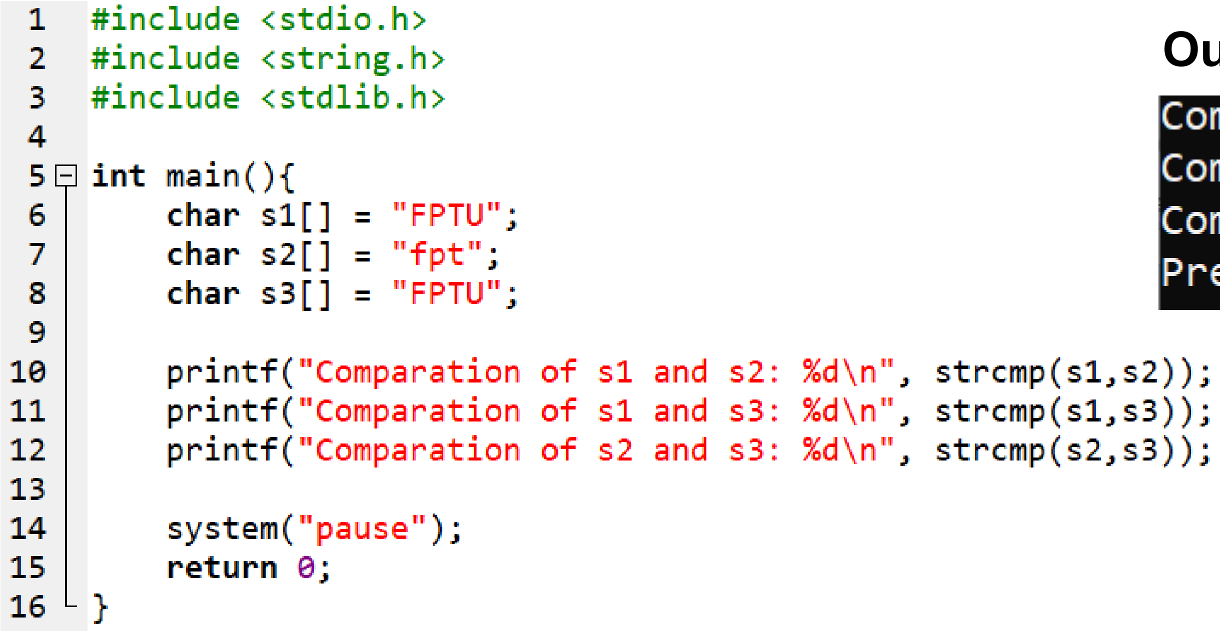
```
Source: Hi FPTU!
Destination: Hi FPTU!
```

strcmp() function

- This function lexicographically compares the first n characters from the two null-terminated strings and returns an integer based on the outcome.
- **Syntax:**

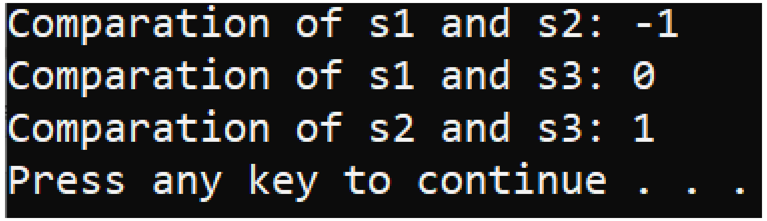
  int **strcmp**(const char *str1, const char *str2);

- Example:



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

int main(){
    char s1[] = "FPTU";
    char s2[] = "fpt";
    char s3[] = "FPTU";

    printf("Comparation of s1 and s2: %d\n", strcmp(s1,s2));
    printf("Comparation of s1 and s3: %d\n", strcmp(s1,s3));
    printf("Comparation of s2 and s3: %d\n", strcmp(s2,s3));

    system("pause");
    return 0;
}
```

Output:



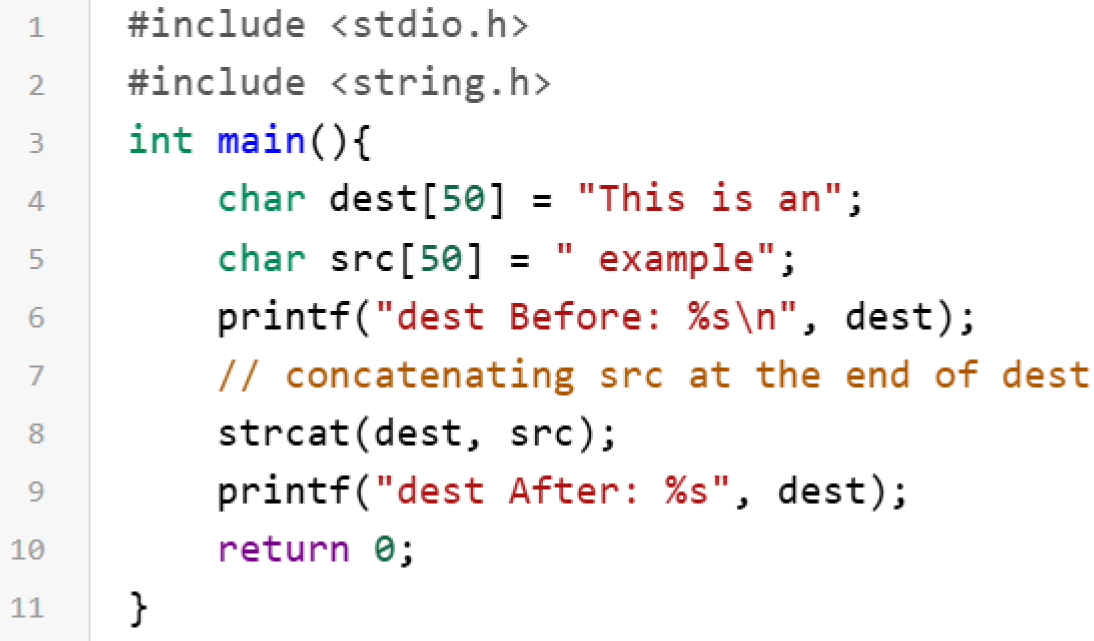
```
Comparation of s1 and s2: -1
Comparation of s1 and s3: 0
Comparation of s2 and s3: 1
Press any key to continue . . .
```

strcat() function

- The strcat() function is used for string concatenation. It will append a copy of the source string to the end of the destination string.
- **Syntax:**

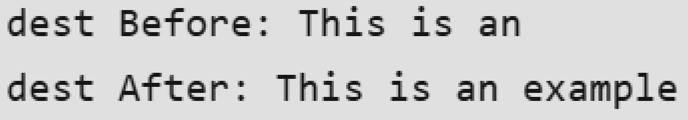
char* **strcat**(char* dest, const char* src);

- Example:



```c
#include <stdio.h>
#include <string.h>
int main(){
    char dest[50] = "This is an";
    char src[50] = " example";
    printf("dest Before: %s\n", dest);
    // concatenating src at the end of dest
    strcat(dest, src);
    printf("dest After: %s", dest);
    return 0;
}
```

Output:



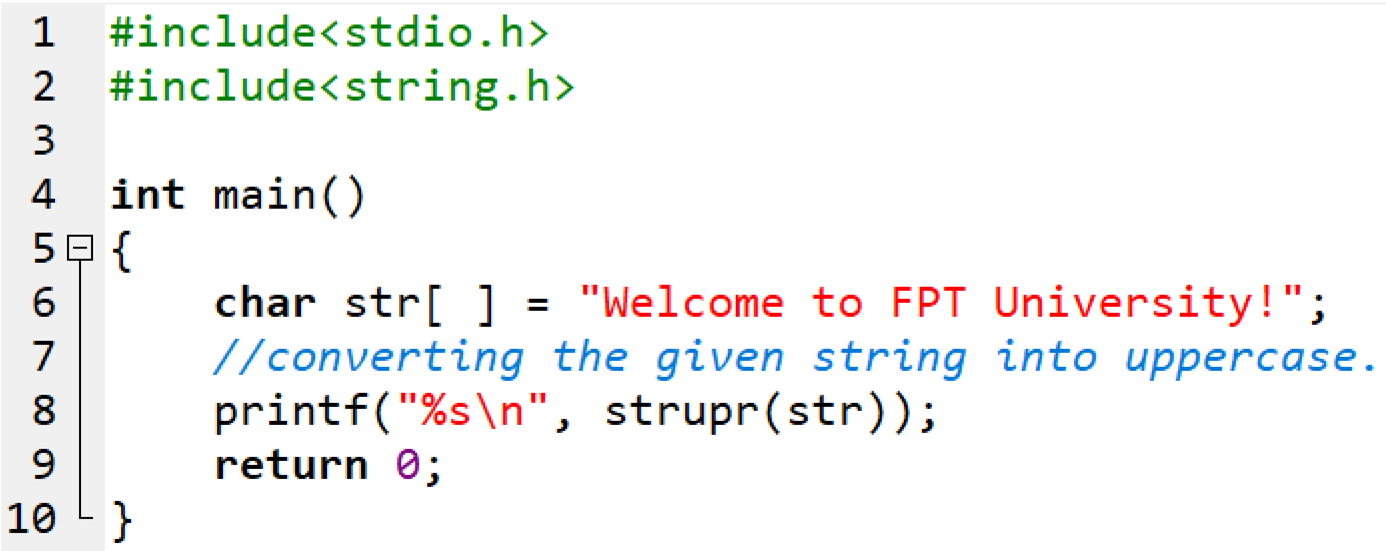
```
dest Before: This is an
dest After: This is an example
```

strupr() function

- The strupr() function is used to converts a given string to uppercase.
- **Syntax:**

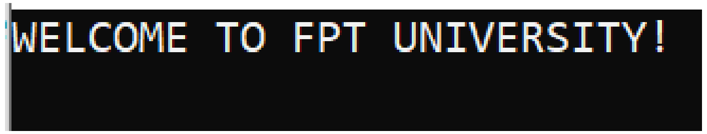
```
char *strupr(char *str);
```

- Example:



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

int main()
{
    char str[ ] = "Welcome to FPT University!";
    //converting the given string into uppercase.
    printf("%s\n", strupr(str));
    return 0;
}
```

Output:



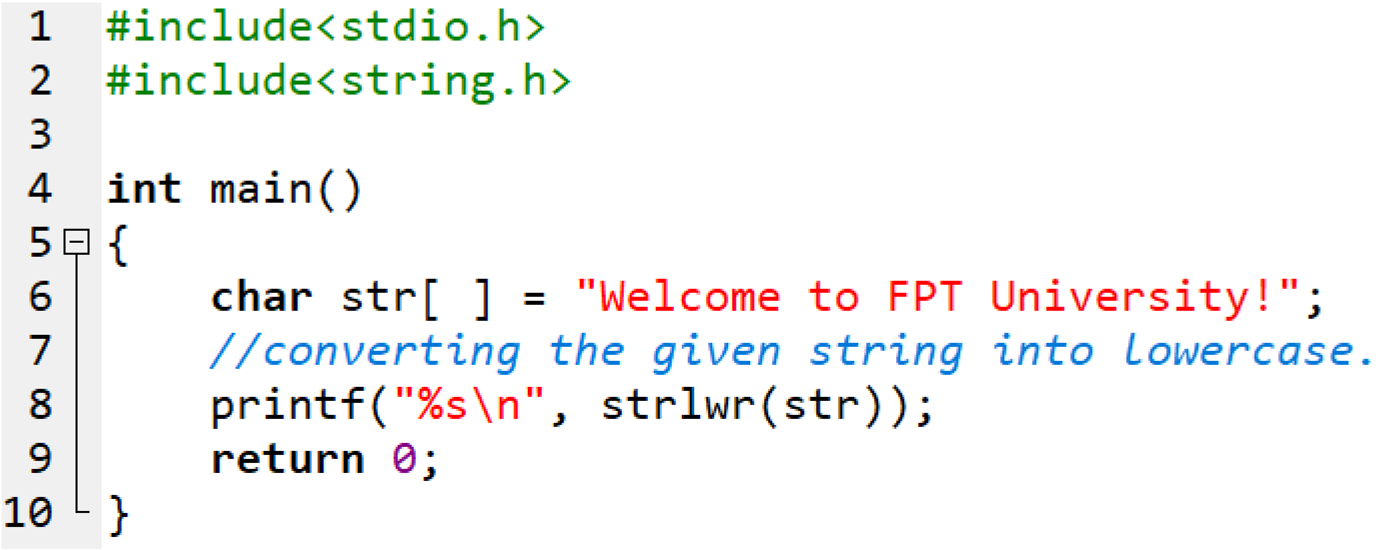
```
WELCOME TO FPT UNIVERSITY!
```

strlwr() function

- The strlwr() function is used to converts a given string to lowercase.
- **Syntax:**

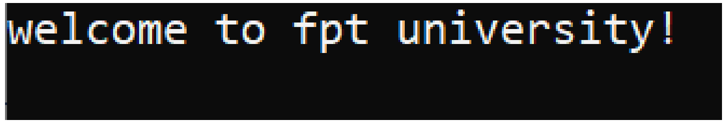
```
char *strlwr(char *str);
```

- Example:



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

int main()
{
    char str[ ] = "Welcome to FPT University!";
    //converting the given string into lowercase.
    printf("%s\n", strlwr(str));
    return 0;
}
```

Output:



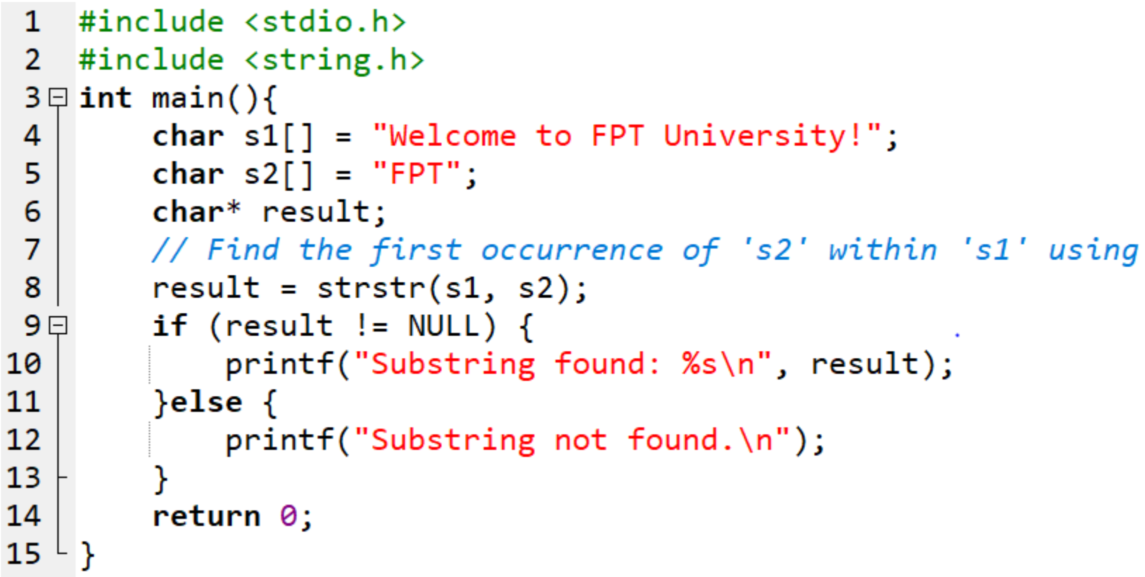
```
welcome to fpt university!
```

strstr() function

- The strstr() function is used to search the first occurrence of a substring in another string.
- **Syntax:**

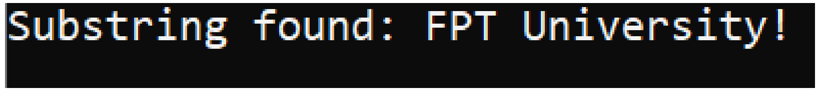
```
char *strstr (const char *s1, const char *s2);
```

- Example:



```c
#include <stdio.h>
#include <string.h>
int main(){
    char s1[] = "Welcome to FPT University!";
    char s2[] = "FPT";
    char* result;
    // Find the first occurrence of 's2' within 's1' using
    result = strstr(s1, s2);
    if (result != NULL) {
        printf("Substring found: %s\n", result);
    }else {
        printf("Substring not found.\n");
    }
    return 0;
}
```

Output:



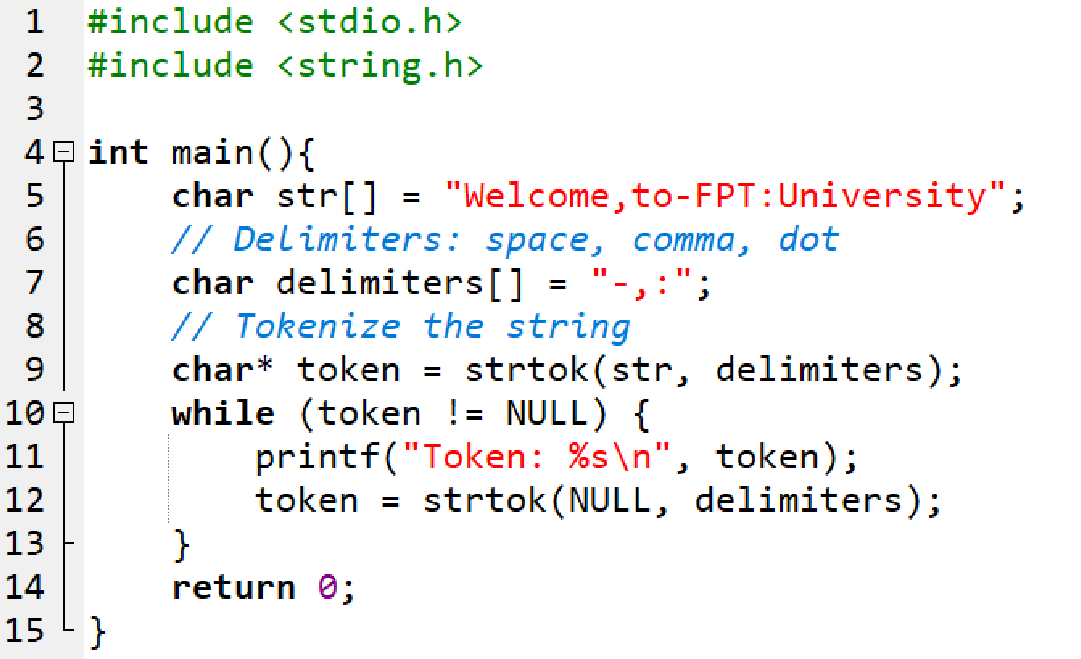
```
Substring found: FPT University!
```

strtok() function

- The strtok() function is used to split the string into small tokens based on a set of delimiter characters.
- **Syntax:**

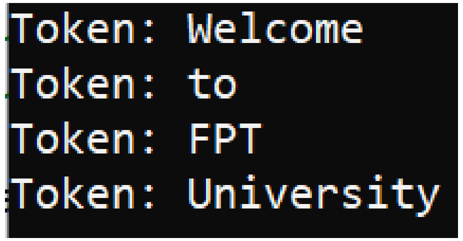
```
char * strtok(char* str, const char *delims);
```

- Example:



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

int main(){
    char str[] = "Welcome,to-FPT:University";
    // Delimiters: space, comma, dot
    char delimiters[] = "-,:";
    // Tokenize the string
    char* token = strtok(str, delimiters);
    while (token != NULL) {
        printf("Token: %s\n", token);
        token = strtok(NULL, delimiters);
    }
    return 0;
}
```

Output:



```
Token: Welcome
Token: to
Token: FPT
Token: University
```

Exercise 3: Using built-in function (string.h)

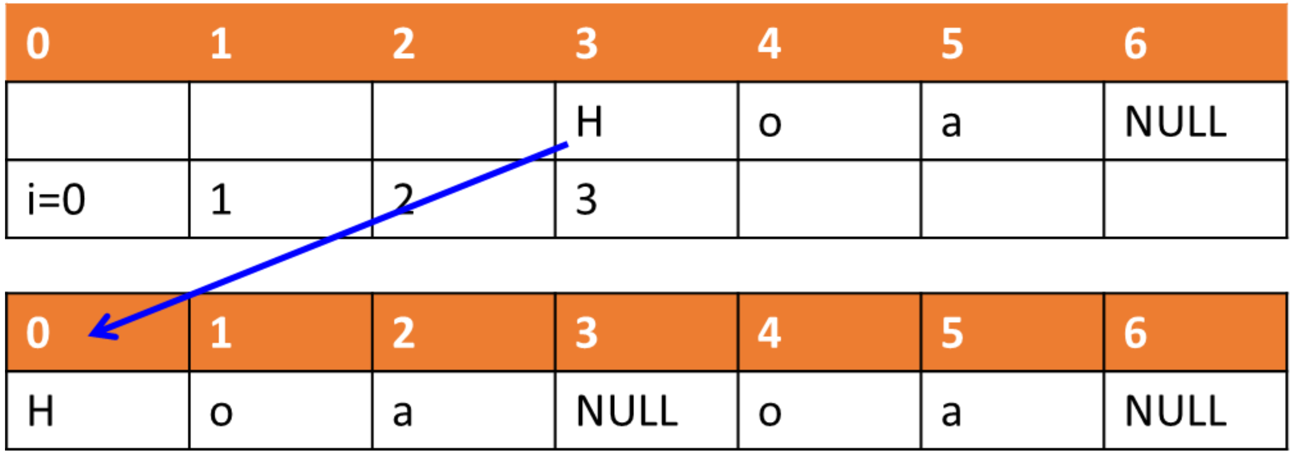
Write a program that:

- Prompts the user to input two strings.
- Compares the two strings and displays whether they are equal or which one is lexicographically greater.
- Concatenates the two strings and displays the result.
- Calculates and displays the length of the concatenated string.
- Searches for a specific character (input by the user) in the concatenated string and displays its position (or indicates if it’s not found).

7. Some User-Defined String Functions

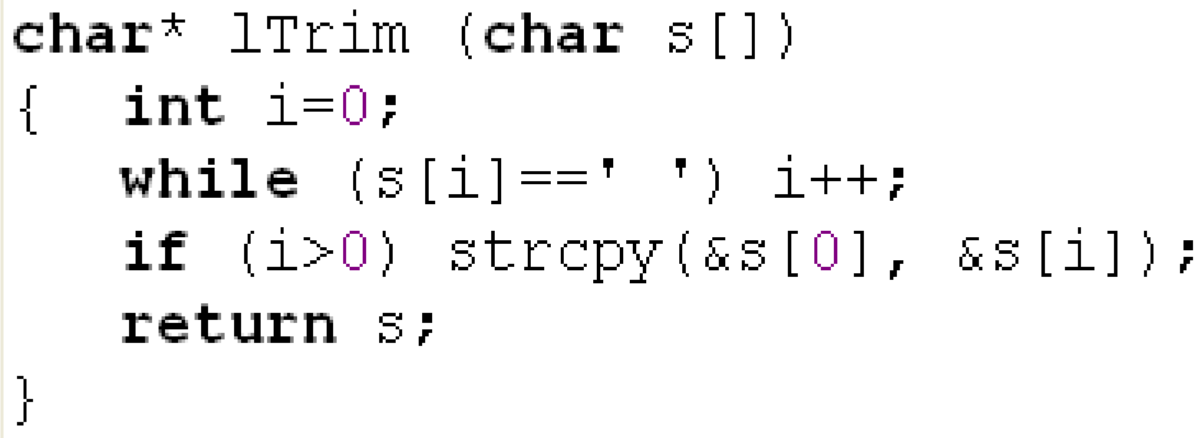
Purpose	Prototype
Trim blanks at the beginning of a string: “ Hello”→ “Hello”	char* **lTrim**(char s[])
Trim blanks at the end of a string: “Hello ” → “Hello”	char* **rTrim**(char s[])
Trim extra blanks ins a string: “ I am student ” → “I am a student”	char* **trim** (char s[])
Convert a string to a name: “ hoang thi hoa ” → “Hoang Thi Hoa”	char* **nameStr**(char s[])

lTrim() user-defined string function



0	1	2	3	4	5	6
			H	o	a	NULL
i=0	1	2	3			

0	1	2	3	4	5	6
H	o	a	NULL	o	a	NULL

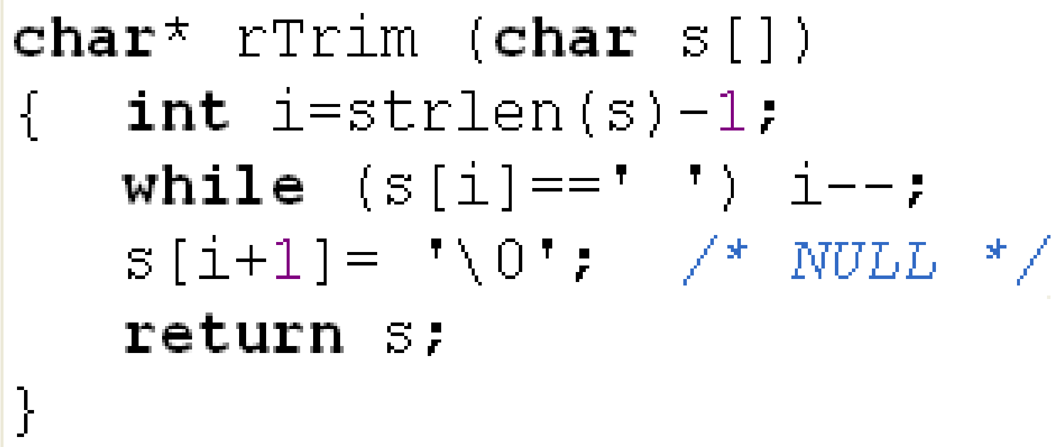


```
char* lTrim (char s[])
{   int i=0;
    while (s[i]==' ') i++;
    if (i>0) strcpy(&s[0], &s[i]);
    return s;
}
```

rTrim() user-defined string function

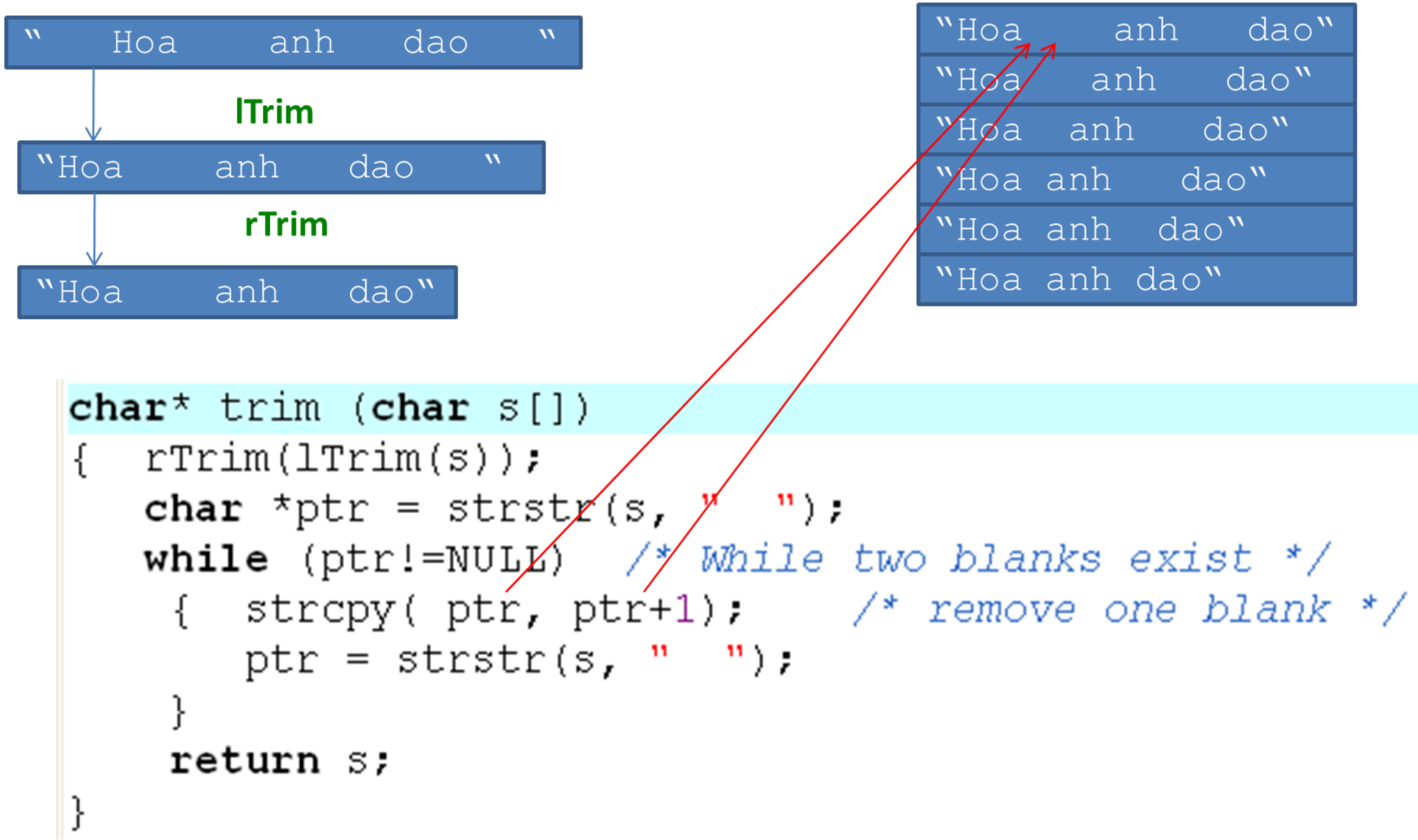
0	1	2	3	4	5	6
H	o	a				NULL
		2	3	4	i=5	

0	1	2	3	4	5	6
H	o	a	NULL			NULL



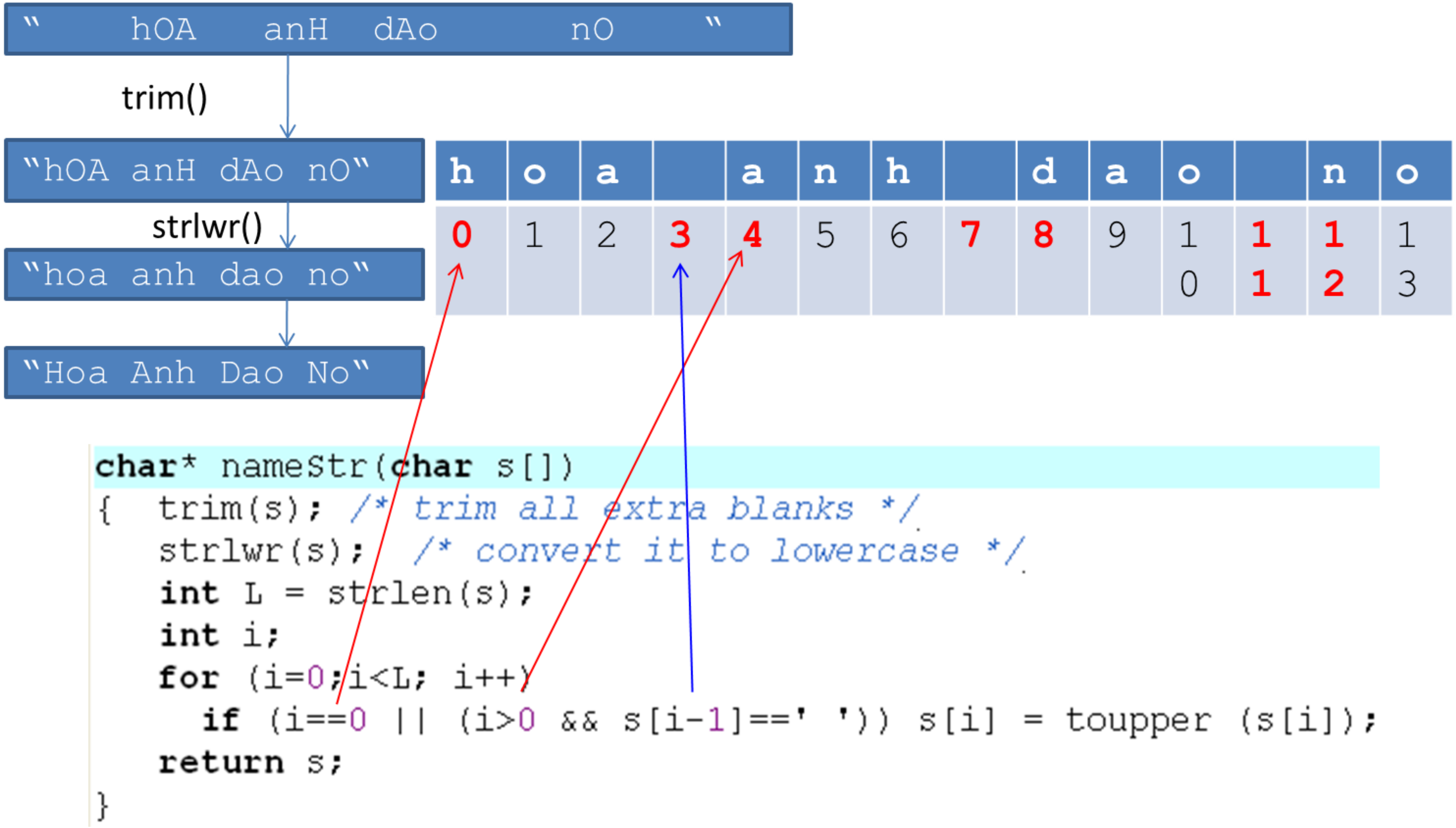
```
char* rTrim (char s[])
{   int i=strlen(s)-1;
    while (s[i]==' ') i--;
    s[i+1]= '\0';   /* NULL */
    return s;
}
```

trim() user-defined string function



```
char* trim (char s[])
{   rTrim(lTrim(s));
    char *ptr = strstr(s, "  ");
    while (ptr!=NULL)   /* While two blanks exist */
     {   strcpy( ptr, ptr+1);      /* remove one blank */
         ptr = strstr(s, "  ");
     }
     return s;
}
```

nameStr() user-defined string function

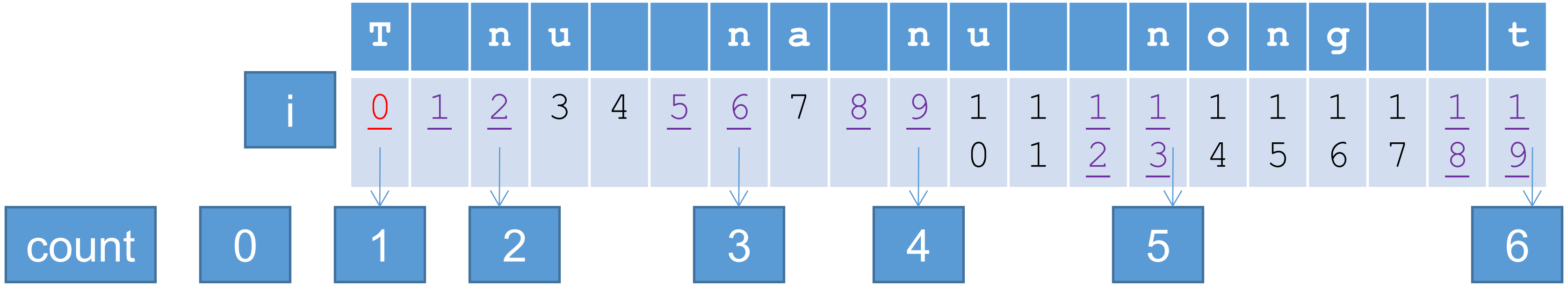


h	o	a		a	n	h		d	a	o		n	o
0	1	2	3	4	5	6	7	8	9	1 0	1 1	1 2	1 3

```
char* nameStr(char s[])
{   trim(s); /* trim all extra blanks */
    strlwr(s);  /* convert it to lowercase */
    int L = strlen(s);
    int i;
    for (i=0;i<L; i++)
      if (i==0 || (i>0 && s[i-1]==' ')) s[i] = toupper (s[i]);
    return s;
}
```

Exercise 4:

Suppose that only the blank character is used to separate words in a sentence. Implement a function for counting number of words in a sentence.



	T		n	u			n	a		n	u			n	o	n	g			t
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

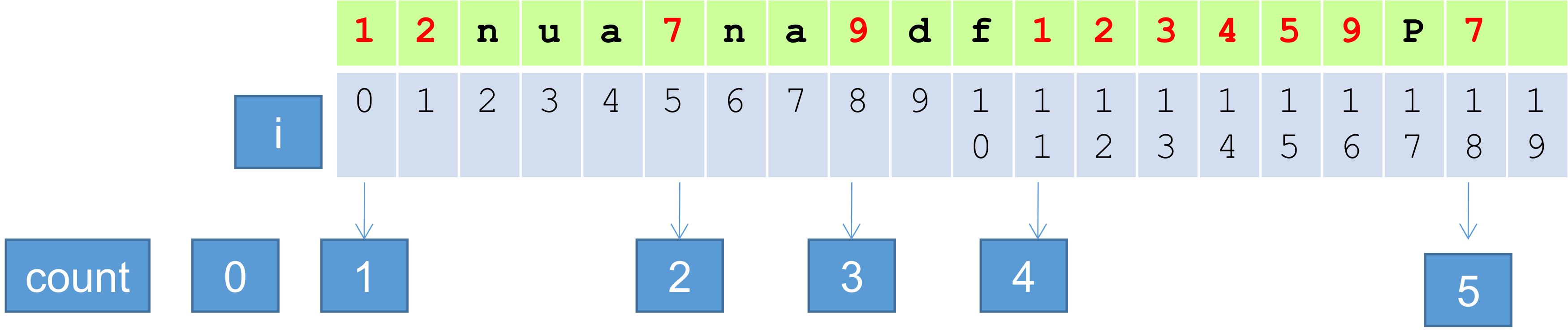
count 0 1 2 3 4 5 6

Counting words in a string
Do Yourself

Criteria for increasing count:
- s[i] is not a blank and (i==0 or s[i-1] is a blank)

Exercise 5:

Counting integers in a string



	1	2	n	u	a	7	n	a	9	d	f	1	2	3	4	5	9	P	7	
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Do Yourself

Criteria for increasing count:
- s[i] is a digit and (i==0 or s[i-1] is not a digit)

Exercise 6:

Do Yourself

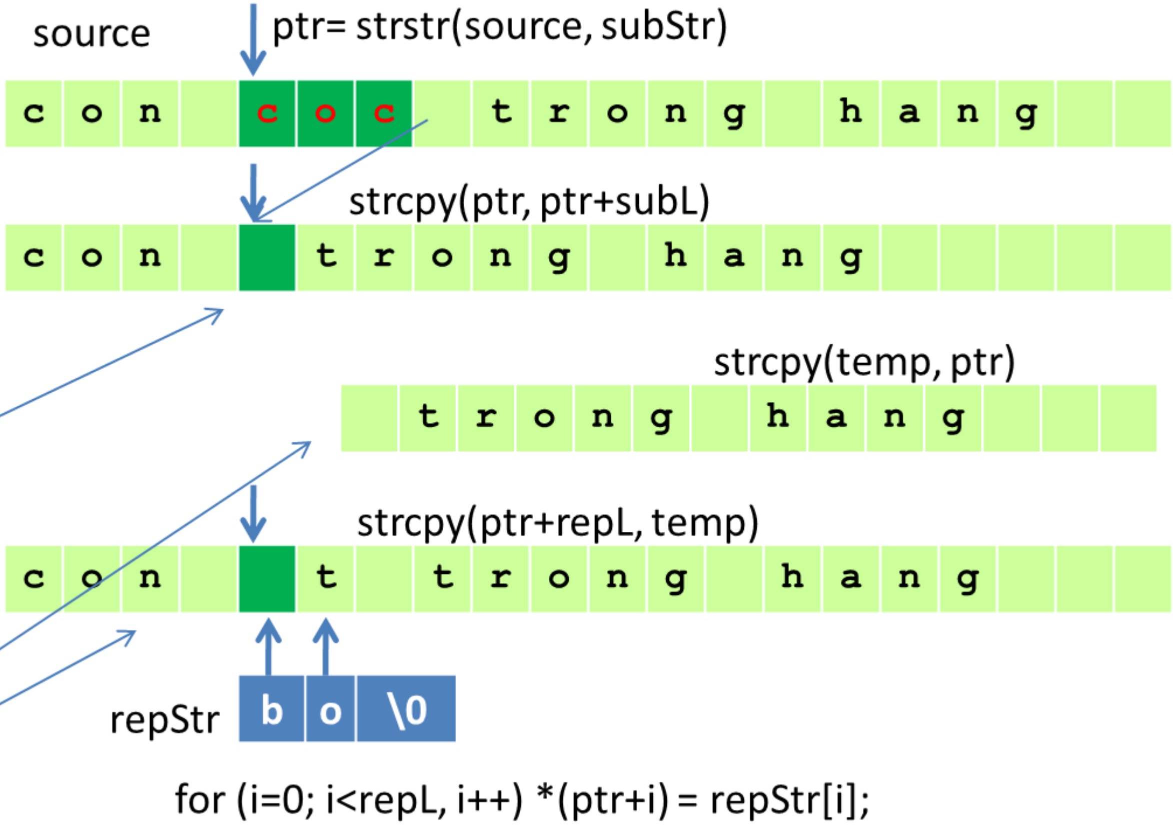
Replace all existences of a sub-string (subStr) in a string (source) by another (repStr)

subStr: "coc", subL=3

repStr: "bo", repL=2

The function **strcpy** will copy char-by-char from the left to the right of the source to the destination. So, it will work properly when a sub-string is shifted up only.

A temporary string is used when a sub-string is shifted down.



source

ptr= strstr(source, subStr)

c	o	n		c	o	c		t	r	o	n	g		h	a	n	g		

strcpy(ptr, ptr+subL)

c	o	n			t	r	o	n	g		h	a	n	g					

strcpy(temp, ptr)

	t	r	o	n	g		h	a	n	g			

strcpy(ptr+repL, temp)

c	o	n			t		t	r	o	n	g		h	a	n	g			

repStr

b	o	\0

for (i=0; i<repL, i++) *(ptr+i) = repStr[i];

8. Array of Strings

- A string array declaration takes the form

```
char  identifier [numberOfString][number_byte_per_string];
```

- For example, to declare an array of 5 names, where each name holds up to 30 characters, we write:

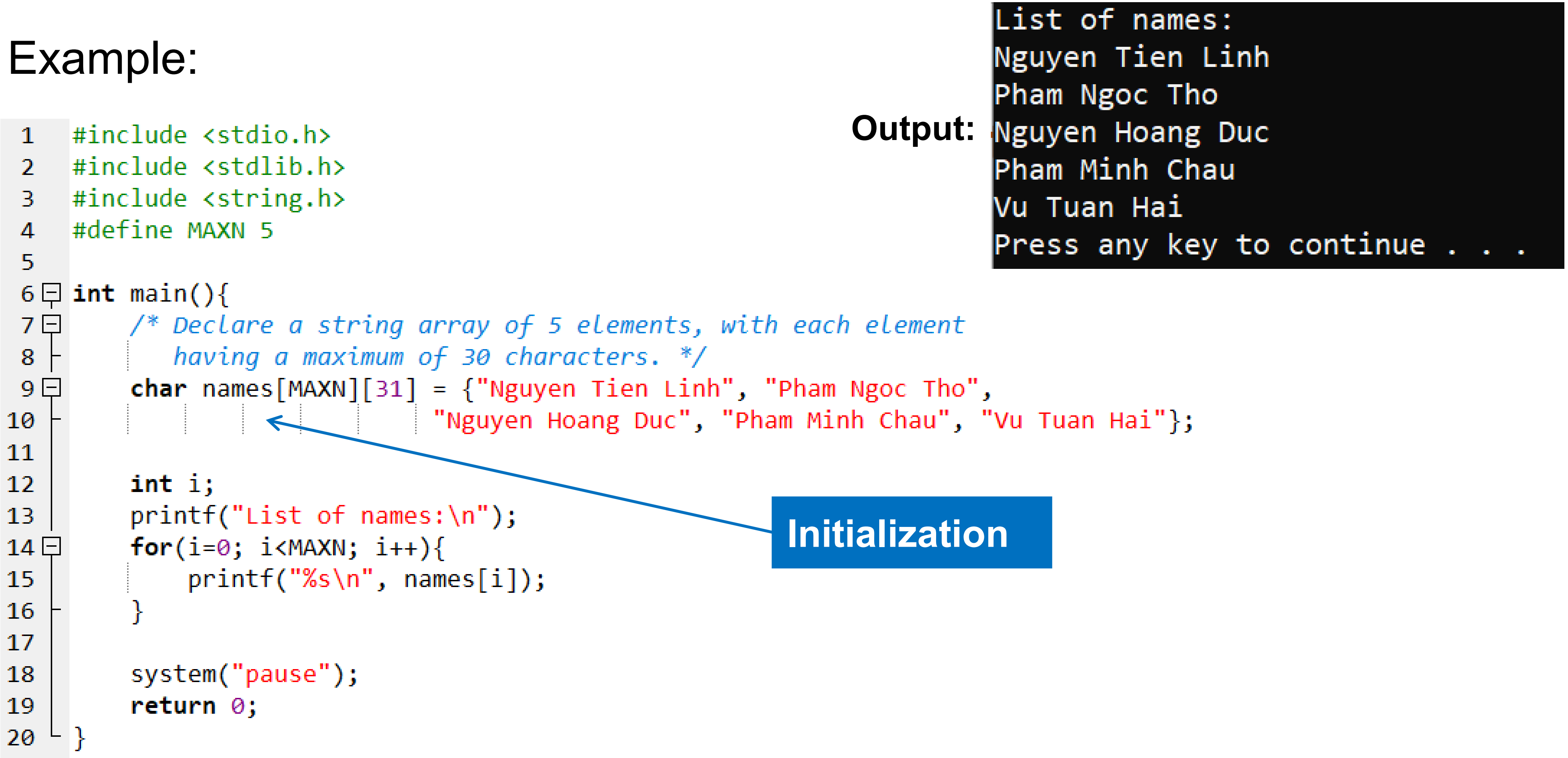
```
char names[5][31];
```

or:

```
char names[5][31] = { "Harry", "Jean", "Jessica", "Irene", "Jim" };
```

Array of Strings (cont.)

- Example:



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXN 5

int main(){
    /* Declare a string array of 5 elements, with each element
       having a maximum of 30 characters. */
    char names[MAXN][31] = {"Nguyen Tien Linh", "Pham Ngoc Tho",
                            "Nguyen Hoang Duc", "Pham Minh Chau", "Vu Tuan Hai"};

    int i;
    printf("List of names:\n");
    for(i=0; i<MAXN; i++){
        printf("%s\n", names[i]);
    }

    system("pause");
    return 0;
}
```

Initialization

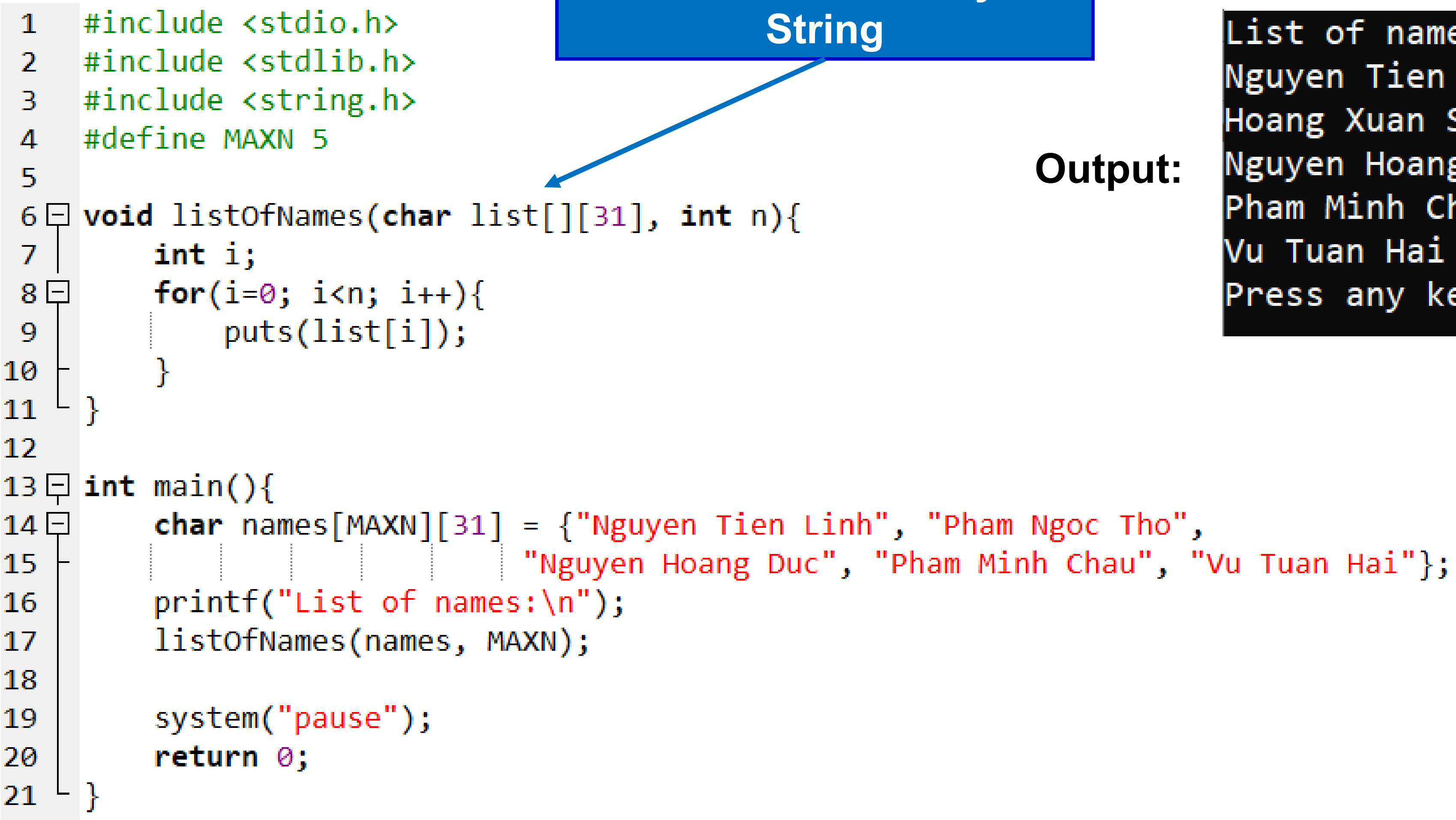
Output:

```
List of names:
Nguyen Tien Linh
Pham Ngoc Tho
Nguyen Hoang Duc
Pham Minh Chau
Vu Tuan Hai
Press any key to continue . . .
```

Array of Strings: Parameter in a function

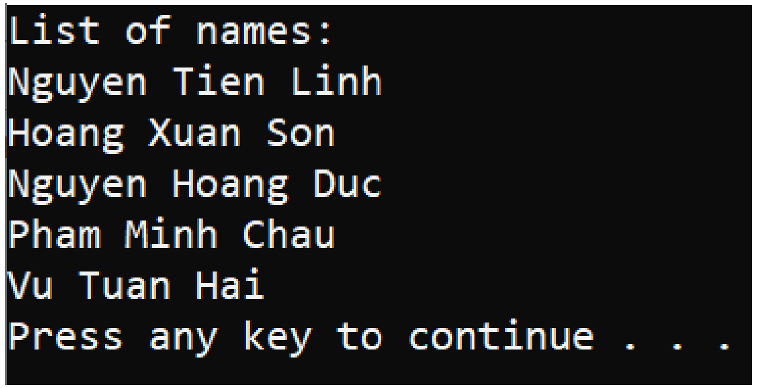
- Example:

Parameter is a Array of String



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXN 5

void listOfNames(char list[][31], int n){
    int i;
    for(i=0; i<n; i++){
        puts(list[i]);
    }
}

int main(){
    char names[MAXN][31] = {"Nguyen Tien Linh", "Pham Ngoc Tho",
                            "Nguyen Hoang Duc", "Pham Minh Chau", "Vu Tuan Hai"};
    printf("List of names:\n");
    listOfNames(names, MAXN);

    system("pause");
    return 0;
}
```

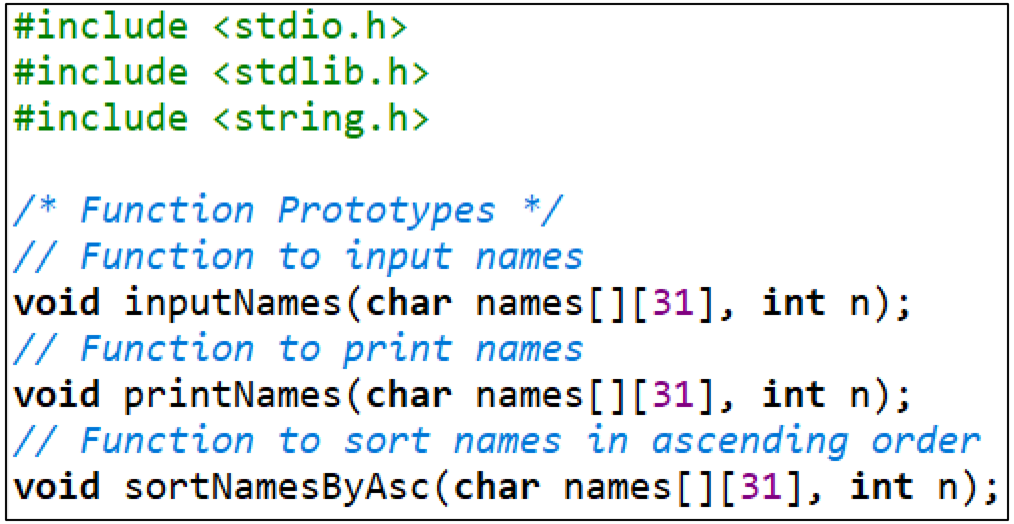
Output:



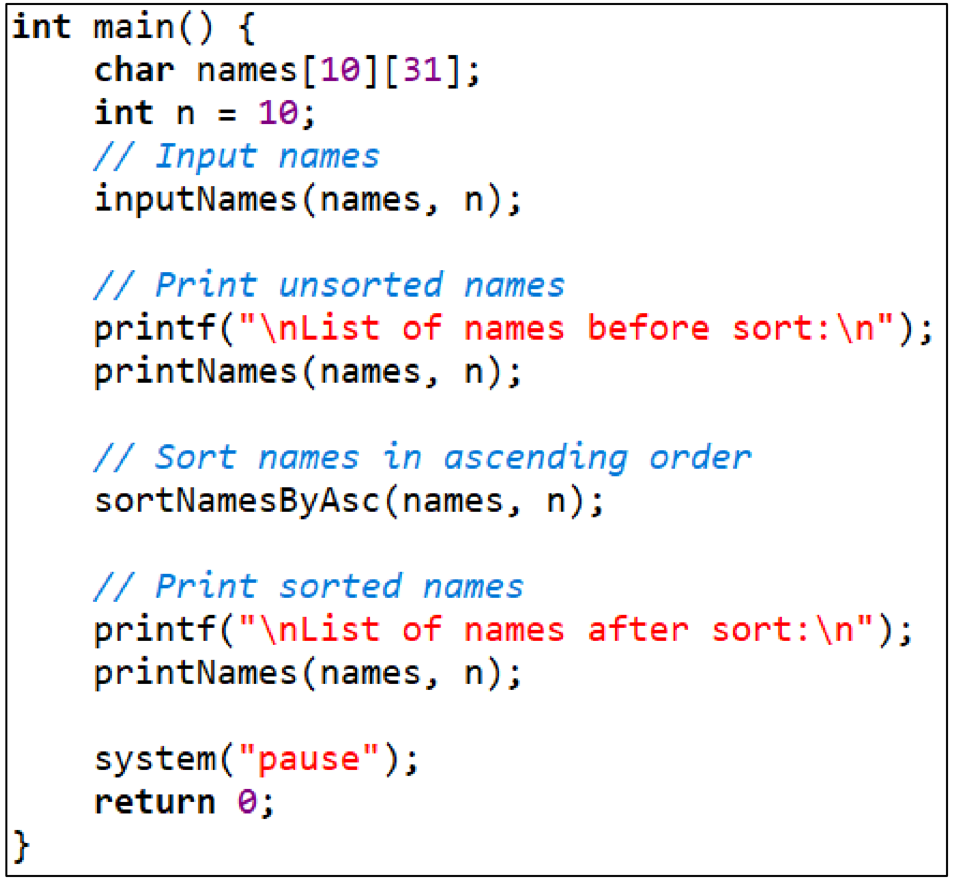
```
List of names:
Nguyen Tien Linh
Hoang Xuan Son
Nguyen Hoang Duc
Pham Minh Chau
Vu Tuan Hai
Press any key to continue . . .
```

Exercise 7:

- Write a C program that will accept 10 names, print out the list, sort the list using ascending order, print out the result.
- Students complete this exercise based on the code design prototype below.
- *Hint*:

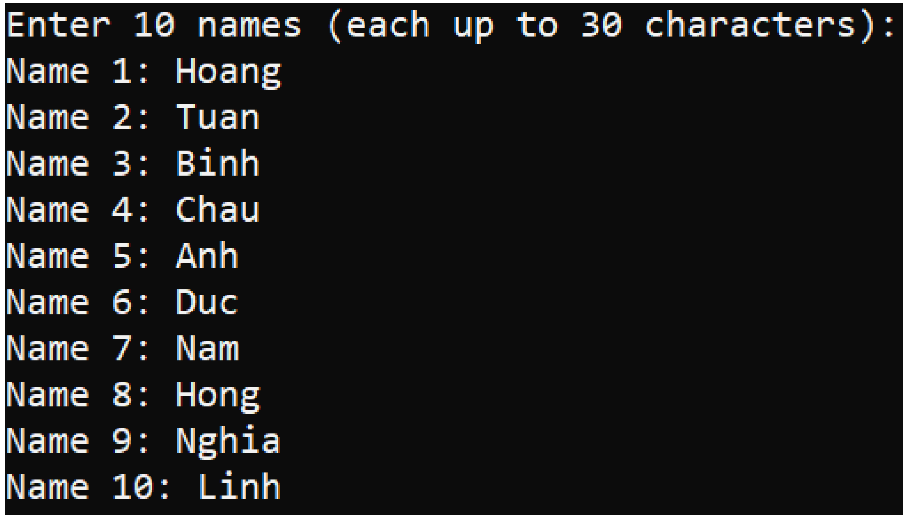


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Function Prototypes */
// Function to input names
void inputNames(char names[][31], int n);
// Function to print names
void printNames(char names[][31], int n);
// Function to sort names in ascending order
void sortNamesByAsc(char names[][31], int n);
```

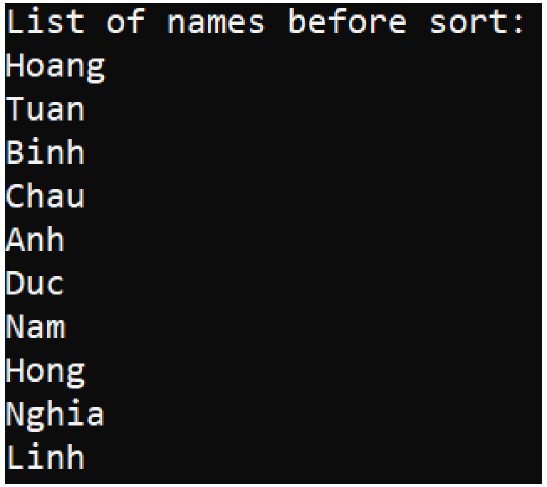


```c
int main() {
    char names[10][31];
    int n = 10;
    // Input names
    inputNames(names, n);

    // Print unsorted names
    printf("\nList of names before sort:\n");
    printNames(names, n);

    // Sort names in ascending order
    sortNamesByAsc(names, n);

    // Print sorted names
    printf("\nList of names after sort:\n");
    printNames(names, n);

    system("pause");
    return 0;
}
```

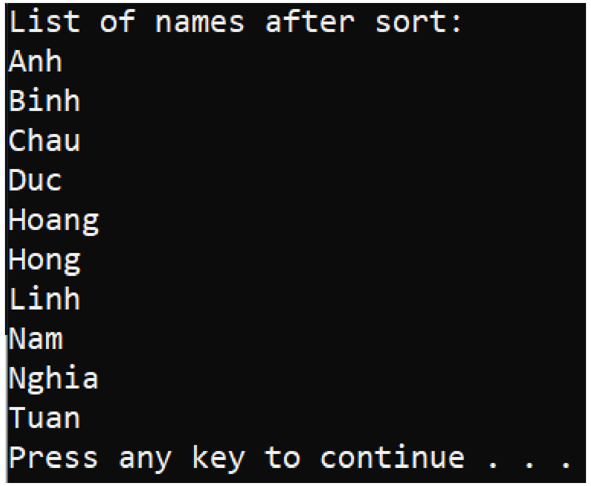
Exercise 7: Sample output



```
Enter 10 names (each up to 30 characters):
Name 1: Hoang
Name 2: Tuan
Name 3: Binh
Name 4: Chau
Name 5: Anh
Name 6: Duc
Name 7: Nam
Name 8: Hong
Name 9: Nghia
Name 10: Linh
```



```
List of names before sort:
Hoang
Tuan
Binh
Chau
Anh
Duc
Nam
Hong
Nghia
Linh
```



```
List of names after sort:
Anh
Binh
Chau
Duc
Hoang
Hong
Linh
Nam
Nghia
Tuan
Press any key to continue . . .
```

Summary

- String in C is terminated by the NULL character (‘\0’)
- A string is similar to an array of characters.
- All input functions for string will automatically add the NULL character after the content of the string.
- Using the functions on arrays, strings are implemented to operate on arrays and strings.
- If dynamic arrays or strings (using pointers), the assignment can be used on these pointers.

Summary

- **String Input**
  - scanf; gets
  - Do yourself using getchar()
- **String Functions and Arrays of Strings**
  - Functions: strlen(), strcpy(), strcmp(), strcat(), strupr(), strlwr(), strstr(), strtok()
- **Arrays of Strings**
  - Input and Output
  - Passing to Functions
  - Sorting an Array of Names