

# 60-141-02 LECTURE 5: CHARACTERS AND STRINGS

Edited by Dr. Mina Maleki

## Outline

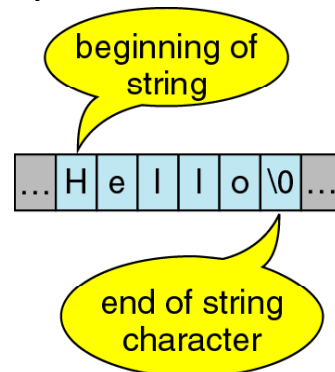
2

- Declaration
- Initialization Techniques
- Character array manipulation with pointers
- Arrays of Strings
- String Functions
- Standard Library Functions for character array manipulation
  - ▣ <ctype.h>
  - ▣ <string.h>
  - ▣ <stdlib.h>
  - ▣ <stdio.h>

## Character Strings

3

- A sequence of characters is often referred to as a character “string”.
- A string is stored in an array of type **char** ending with the null character `'\0'`.



## Declaration

4

- Declare a character (char) array the same way you declare arrays.

```
#define STRLEN 256
char strName [ STRLEN ] ;
char strFamily [200];
char * strPtr ;
```
- In C there is no native “String” data type. Strings are simply defined as a null terminated ‘array of characters’.
- Strings and string manipulations are therefore applications of arrays and of functions that work on these arrays.

## Initialization

5

- Arrays may be initialized at declaration time.

- Example

```
char myArray[5] = {'h','e','l','l','o'};
```

or

```
char Word[ ] = {'n','a','t','u','r','e'};
```

- Note the use of single quotes around each character.
  - Not using single quotes around characters is an error

## String Initialization

6

- To **store strings** (such words and sentences) in char arrays, we need to explicitly terminate the array with a special single character symbol :
  - The end-of-string terminator `'\0'` (escape-zero)
- `'\0'` should be inserted as the last character of the stored string to indicate to the system where the string ends.
- Do not confuse `'\0'` with NULL, they are different!
- Declare an char array to store a string of words:

```
char Weather[ 5 ] = { 'w', 'a', 'r', 'm', '\0' };
```

or

```
char Weather[ ] = { 'w', 'a', 'r', 'm', '\0' };
```

  - Array size is often omitted when declaring strings (char arrays) because it becomes too inconvenient to count the number of characters you are storing in a sentence.

## String Initialization ...

7

- Also, a string variable can be initialized at the same time it's declared as follow:

```
char date1[8] = "June 14";
```
- The compiler will automatically add a **null character** `\0` so that `date1` can be used as a string:

date1

J	u	n	e		1	4	\0
---	---	---	---	--	---	---	----

- If the initializer is too short to fill the string variable, the compiler adds extra null characters:

```
char date2[9] = "June 14";
```

Appearance of `date2`:

date2

J	u	n	e		1	4	\0	\0
---	---	---	---	--	---	---	----	----

## String Initialization ...

8

- An initializer for a string variable can't be longer than the variable, but it can be the same length:

```
char date3[7] = "June 14";
```
- There's no room for the null character, so the compiler makes no attempt to store one:

date3

J	u	n	e		1	4
---	---	---	---	--	---	---

- The declaration of a string variable may omit its length, in which case the compiler computes it:
  - `char date3[] = "June 14";`
  - The compiler sets aside **eight** characters for `date3`, enough to store the characters in "June 14" plus a null character.

## String Initialization ...

9

- The null string: A character string that contains no characters other than the null character.

```
char empty[] = ""; // char empty[] = {'\0'};  
char buf[100] = ""; // char buf[100] = {'\0'};
```

\0
----

## Example 1

10

- The string "abc" is stored as an array of four characters:

a	b	c	\0
---	---	---	----

- `char str1[] = {'H','e','l','l','o','\0'};`
- `char str2[] = "My name is Bob!";`
  - `//Array size = 16`
- `const char * str3 = "String that cannot be changed!";`

## Example 2

11

```
char strName [50] ;  
for(int k=0; k<49; k++ )  
    strName[k] = '#' ; // Fill with # symbols  
strName[49] = '\0' ;
```

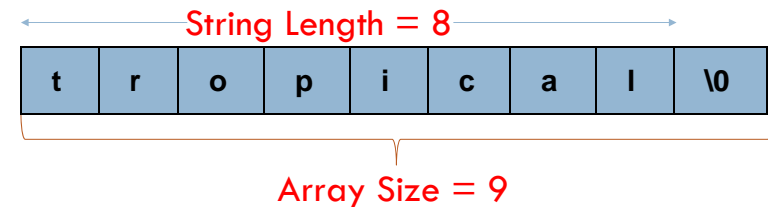
```
char strName [50] , *strPtr ;  
strPtr = strName;  
for(int k=0; k<49; k++, strPtr++ )  
    *strPtr = '#' ; // Fill with # symbols  
*strPtr = '\0' ;
```

## String Length

12

- String Length= the number of characters used in the initialization
- Array Size= String Length + 1 ('\0').
- Example: `char Weather[] = "tropical";`

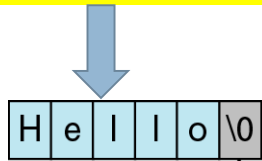
```
char Weather[]={ 't','r','o','p','i','c','a','l','\0' };
```



## Example 1

13

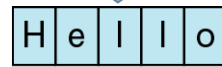
```
char str[] = {'H','e','l','l','o','\0'};
char str[] = "Hello";
```



end of string character

Array Size = 6  
String Length = 5

```
char str[] = {'H','e','l','l','o'};
```



an array —  
no end of string

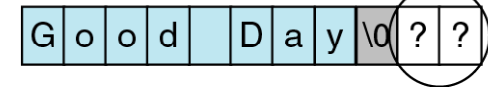
Array Size = 5  
String Length = -

## Example 2

14

- `char str[11];`
- Array size = 11
- String length = 8

Part of the array,  
but not part of the  
string



## Character vs. String

15

- A string constant is a sequence of characters enclosed in double quotes.

```
char s1[2] = "H";
```

s1: 

H	\0
---	----

 (two bytes)

```
char s2[] = "";
```

s2: 

\0
----

 (one byte)

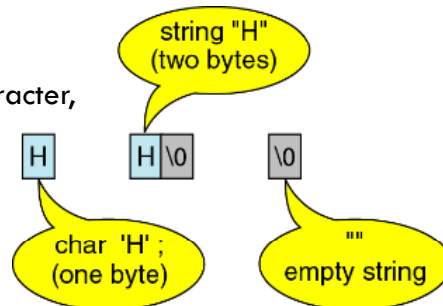
- On the other hand, the character, in single quotes:

```
char s3 = 'H';
```

s3: 

H
---

 (one byte)



## Accessing the Characters in a String

16

- Since strings are stored as arrays, we can use subscripting to access the characters in a string.

```
char str[] = "This is a test";
```

```
str[0] → 'T';
```

```
printf("%c", str[11]); → 'e'
```

- To process every character in a string `s`, we can set up a loop that increments a counter `i` and selects characters via the expression `s[i]`.

## Example

17

- A function that counts the number of spaces in a string:

```
int count_spaces(char s[])
{
    int count = 0, i;

    for (i = 0; s[i] != '\0'; i++)
        if (s[i] == ' ')
            count++;
    return count;
}
```

## Reading and Writing Strings

## Reading and Writing Strings

19

- Writing a string
  - ▣ printf or puts.
- Reading a string
  - ▣ To read a string in a single step, we can use either scanf or gets.
  - ▣ As an alternative, we can read strings one character at a time.
- printf and scanf work on char arrays slightly differently than integer or other data type arrays.

## Writing Strings

20

- The %s conversion specification allows printf to write a string:

```
char str[] = "Are we having fun yet?";
printf("%s\n", str);
```
- Given a pointer to a char array, the printf would print the char that the pointer is referring to as well as the rest of the char array.... until: the '\0' character is encountered.
- The '\0' plays a critical role in controlling or 'stopping' the printing task.
- If you omit the '\0' the printf will result in an error (or unreadable characters being printed)

## Writing Strings ...

21

- The C library also provides `puts` to write a string :

```
puts(str);
```

- After writing a string, `puts` always writes an additional new-line character.

```
#include <stdio.h>
int main()
{
    char str1[ ] = "Test";
    puts(str1);      //printf("%s \n", str1);
    return 0;
}
```

## Reading Strings

22

- The `%s` conversion specification allows `scanf` to read a string into a character array:

```
char str[40]; //can store up to 39 chars + implied '\0'
scanf("%s", str); //no ampersand &
```

- Reads characters from the keyboard and override the contents of `str`.
- Reads up to but not including a delimiter (white space character)
  - Delimiter examples: space, comma, tab, newline...
- No ampersand(&) when inputting strings into character arrays!
- `%ns` scans the next `n` characters or up to the next white space character, whichever comes first
- `scanf` always stores a null character at the end of the string.

## Reading Strings ...

23

- `scanf` won't usually read a full line of input.
- A new-line character will cause `scanf` to stop reading, but so will a space or tab character.
- To read an entire line of input, we can use `gets`.
- Properties of `gets`:
  - Doesn't skip white space before starting to read input.
  - Reads until it finds a new-line character.
  - Discards the new-line character instead of storing it; the null character takes its place.

## Example

24

- Suppose the user enters the line  
To C, or not to C: that is the question.
- `scanf("%s", sentence);`
  - `scanf` will store the string "To" in `sentence`.
- `gets(sentence);`
  - `gets` will store the string  
" To C, or not to C: that is the question."  
in `sentence`.

## String Functions

## String Functions

- Computing a String Length
- Copying one String to another
- Comparing String contents
- Appending one String to another
- Searching a String
- and many more...
  - ▣ Do we need to write a function for each?

## Compute String Length

- `strlen` accepts a terminated string (char array with `'\0'`)
- `strlen` returns the size of the string in the array. This is not the array size, this is the number of characters that are actually used to store the string, not including `'\0'`.

```
int strlen(char A[]){
    int len= 0;
    while (A[len]!='\0')
        len++;
    return len;
}
```

```
int strlen(char *A){
    int len = 0;
    while(*(A+len)!='\0')
        len++;
    return len;
}
```

## Example

- Display length of a string

```
#include <stdio.h>
int strlen (char word[]) ; // Function Prototype

int main(void) {
    char word[120];
    printf ("Please enter a string:\n");
    scanf ("%s", word);

    int len = strlen(word);
    printf ("Length of %s is %i.\n", word, len);

    char str[] = "My name is Bob!" ;    //str length = 15
    printf ("Length of %s is %i.\n", str,strlen(str));
    return 0;
}
```

## Assign a Value to a String

29

- Note: It is **illegal** to assign a value to a string variable (except at declaration).
- Example 1:  

```
char str1[10];  
str1 = "Hello"; // illegal assignment
```
- Example 2:  

```
char str1[] = "Hello" ;  
char str2[20];  
str2 = str1; // illegal assignment
```

## Copy one String to Another

30

```
// Function to copy a string to another  
void strcpy (char dest[], char source[])  
{  
    int i= 0;  
    while (source[i] != '\0')  
    {  
        dest[i] = source[i];  
        i++;  
    }  
    dest[i]= '\0';  
}
```

## Copy one String to Another ...

31

```
#include <stdio.h>  
void strcpy(char dest[], char source[]);  
int main (void) {  
    char str1[] = "Fall 2016", char str2[100];  
    strcpy (str2, str1);  
    printf ("str2 = %s\n", str2);  
    return 0;  
}  
void strcpy (char dest[], char source[]) {  
    int i;  
    for (i=0; source[i]!='\0';i++)  
        dest[i]=source[i];  
    dest[i]= '\0';  
}
```

## Compare Two Strings (array index)

32

The operator == doesn't test two strings for equality.  
if (string1 == string2) // illegal comparison

```
char buffer[40];  
char password[] = "secret";  
int p = 0;  
int match = 1;  
scanf("%s", buffer);  
while(password[p] != '\0')  
{  
    if (password[p] != buffer[p])  
        match = 0;  
    p++;  
}  
if (match == 1)  
    printf("success");  
else  
    printf("login incorrect");
```

buffer

0	1	2	3	4	5	6	...	39		
's'	'e'	'c'	'r'	'e'	't'	'\0'	'j'	'u'	'n'	'k'

password

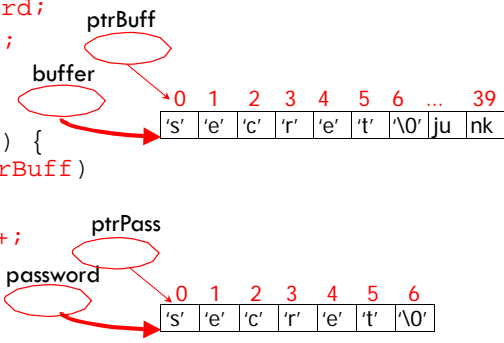
0	1	2	3	4	5	6
's'	'e'	'c'	'r'	'e'	't'	'\0'



## Compare Two Strings (array pointers)

33

```
char buffer[40];
char password[] = "secret";
char *ptrPass = password;
char *ptrBuff = buffer;
int match = 1;
scanf("%s", buffer);
while(*ptrPass != '\0') {
    if (*ptrPass != *ptrBuff)
        match = 0;
    ptrPass++; ptrBuff++;
}
if (match == 1)
    printf("success");
else
    printf("login incorrect");
```



## Concatenate Two Strings

34

```
#include <stdio.h>
int main (void) {
    char str1[] = "Test " ;
    char str2[] = "works.", str3[20];
    // copy str1 to result
    for (int i = 0; str1[i] != '\0'; ++i )
        str3 [i] = str1[i];
    // copy str2 to result
    for (int j = 0; str2[j] != '\0'; ++j )
        str3 [i + j] = str2[j];
    // Terminate the concatenated string with a null character
    str3[i + j] = '\0';
    printf ("%s\n", str3);
    return 0;
}
```

## Standard Library Functions

## Standard Library Functions

36

- The C standard libraries are rich with character and string handling functions
- When you include a standard string library function you will have available to you a large number functions that allow string manipulation with ease.
- First you need to write your own equivalent functions just so you appreciate their efficiency and understand how they work.
- Before using a function, carefully examine its prototype and read its documentation.
- We will discuss functions from four libraries:
  - #include <ctype.h>      and      #include <stdlib.h>
  - #include <stdio.h>      and      #include <string.h>

## Character Handling Library <ctype.h>

37

- The character handling library includes several functions that perform useful tests and manipulation of **character** data.
- Each function receives a character – represented as an `int` – or EOF as an argument.
  - Note that characters are often manipulated as integers, because a character in C is usually a one byte integer.
- EOF normally has the value of `-1` and some hardware architectures do not allow negative values to be stored in `char` variables.

## <ctype.h>

38

- **Query** functions provide information about the nature of the character data `int isValueRange(int c); // Returns 1 if a match, or 0`

Function Prototype	Function Description
<code>int isblank (int c);</code>	Returns a positive value if <code>c</code> is ' '; otherwise 0
<code>int isdigit (int c);</code>	Returns a positive value if <code>c</code> is a base-10 digit in the range '0' to '9' ; otherwise 0
<code>int isalpha (int c);</code>	Returns a positive value if <code>c</code> is an alphabetic character in the range 'a' to 'z', or 'A' to 'Z' ; otherwise 0
<code>int isalnum (int c);</code>	Returns a positive value if <code>c</code> is an alphabetic character, or a base-10 digit ; otherwise 0
<code>int isxdigit (int c);</code>	Returns a positive value if <code>c</code> is a base-16 (hexadecimal) digit in the range '0' to '9', or 'a' to 'f', or 'A' to 'F' ; otherwise 0
<code>int islower (int c);</code>	Returns a positive value if <code>c</code> is a lower case alphabetic character in the range 'a' to 'z' ; otherwise 0
<code>int isupper (int c);</code>	Returns a positive value if <code>c</code> is an upper case alphabetic character in the range 'A' to 'Z' ; otherwise 0

## <ctype.h> ...

39

- And still more *query* functions for non-alphanumeric character data

Function Prototype	Function Description
<code>int isspace (int c);</code>	Returns >0 if <code>c</code> is any valid <i>white space</i> character data (including blank, newline '\n', tab '\t', etc); otherwise 0
<code>int iscntrl (int c);</code>	Returns >0 if <code>c</code> is any valid <i>control</i> character data (including '\n', '\b', '\r', '\a' etc); otherwise 0
<code>int ispunct (int c);</code>	Returns >0 if <code>c</code> is any valid, printable punctuation character data (including ',', '.', '!', ':', ';' etc.); otherwise 0
<code>int isprint (int c);</code>	Returns >0 if <code>c</code> is any valid, printable character data; otherwise 0. A printable character is a character that is not a control character.
<code>int isgraph (int c);</code>	Returns >0 if <code>c</code> is any valid character data representing a graphical symbol (such as '<', '>', '#', '\$' etc, and including extensions to ASCII); otherwise 0

## <ctype.h> ...

40

- **Transformative** functions modify the character data

Function Prototype	Function Description
<code>int tolower (int c);</code>	Returns the value <code>c</code> if <code>c</code> is a lower case alphabetic character, or the upper case variant of the same alphabetic character (Ex. tolower( 'A' ) returns 'a')
<code>int toupper (int c);</code>	Returns the value <code>c</code> if <code>c</code> is an upper case alphabetic character, or the lower case variant of the same alphabetic character (Ex. toupper( 'e' ) returns 'E')

## String Conversion Functions <stdlib.h>

41

- String conversion functions comes from the general utilities library (stdlib).

```
double atof ( const char *nPtr);
int atoi ( const char *nPtr);
long atol ( const char *nPtr);
```

- Purpose of these functions is to convert a string of digits (or portion) to (1) an integer or (2) a floating point type

## <stdlib.h>

42

```
int atoi (char *str);
long atol (char *str);
```

- Takes a character string and converts it to an **integer** or **long**.
- White space and + or - are OK.
- Starts at beginning and continues until something non-convertible is encountered.

- Examples:

String	Value returned
"157"	157
"-1.6"	-1
"+50x"	50
"twelve"	0
"x506"	0

## <stdlib.h> ...

43

```
double atof (char * str);
```

- Handles digits 0-9.
- A decimal point.
- An exponent indicator (e or E).
- If no characters are convertible a 0 is returned.

- Examples:

String	Value returned
"12"	12.000000
"-0.123"	-0.123000
"123E+3"	123000.000000
"123.1e-5"	0.001231

## <stdlib.h>

44

- General prototype form:

```
resultType strtOuputType ( const char * nPtr,
                           char **endPtr
                           [, int base ] );
```

- nPtr points at the input string (protected as constant)
- resultType refers to one of double, long int, or unsigned long int
- OutputType refers to one of d, l, or ul
- base refers to the base of the input string (0, or 2..36)
- endPtr points at the position within the input string where a valid numeric representation terminates

## <stdlib.h> ...

45

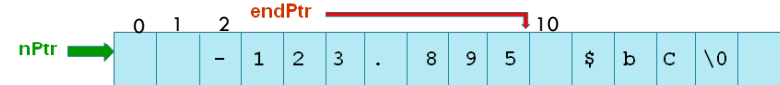
- `double strtod( const char * nPtr, char **endPtr );`
  - ▣ If `nPtr` points at a valid string representation of a signed real number (possibly followed by additional character data), return a *double* value;
  - ▣ Else return 0 if no part of the input string can be converted.
  - ▣ Return a pointer (through `*endPtr`) to the character following the last convertible character
  - ▣ if no part of the input string is convertible then `*endPtr` is set to `nPtr`.

## Example

46

```
double D ;
const char * S = " -123.895 $bC";
char * EP ;
D = strtod( S , &EP );
if( EP != S )
{
    printf("Value converted is %f\n", D );
    printf("EP is %ld pointed to '%c'.\n", EP-S, *EP);
}
else
    printf( "No value could be converted\n" );
```

Value converted is -123.895000  
EP is 10 pointed to '\$'.



## <stdlib.h> ...

Function Prototype	Function Description
<code>long strtol( const char * nPtr, char **endPtr, int base );</code>	If <code>nPtr</code> points at a valid string representation of a signed integer number (possibly followed by additional character data), return <b>a long int</b> value; Else return 0 if no part of the input string can be converted. Return a pointer (through <code>*endPtr</code> ) to the character <u>following</u> the last convertible character – if no part of the input string is convertible then <code>*endPtr</code> is set to <code>nPtr</code> . The input string may use any base digits in the range 0 to 32.
<code>unsigned long strtoul( const char * nPtr, char **endPtr, int base );</code>	Performs analogously to <code>strtol()</code> for string to <b>unsigned long int</b> conversion.

## Example

48

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char str[30] = "2030300 This is test";
    char *ptr;
    long ret;

    ret = strtoul(str, &ptr, 10);
    printf("The number(unsigned long integer) is %ld\n", ret);
    printf("String part is |%s|", ptr);
    return(0);
}
```

The number(unsigned long integer) is 2030300  
String part is | This is test |

## Standard I/O Library Functions <stdio.h>

49

- The standard input/output library (<stdio.h>) has a number of functions specifically for manipulating character and string data.
  - ▣ I/O of characters and strings
  - ▣ Conversion to and from character and internal data representations

## <stdio.h>

### Function Prototype and Description

**int getchar( void );**

Fetches and returns a single character from the input *stream* (stdin); if end of file is signalled then the return value is EOF

**int putchar( int C );**

Outputs a single character to the output *stream* (stdout). Returns the same character if successful; otherwise returns EOF on failure

**char \* gets( char \* S );**

Fetches all input characters from the standard input stream (stdin) up to either (a) a new line '\n', or (b) EOF, or (c) N-1 characters have been inputted, and then appends a delimiter '\0' to make a string. The pointer S points to the inputted string.

**int puts( const char \* S );**

Outputs the string of characters S, followed by a **newline** '\n'. Returns a non-zero integer result (typically the number of characters outputted), or EOF on failure.

**char \* fgets( char \* S, int N, FILE \* stream );**

Fetches all characters up to either (a) a new line '\n', or (b) EOF, or (c) N-1 characters have been inputted, and then appends a delimiter '\0' to make a string. The pointer S points to the inputted string. Input is from the input stream (typically stdin, but can be from a text file). Returns a pointer to the input string, or NULL if failure occurs (as with EOF).

## Example

51

```
#include <stdio.h>
#define MAX 256
int main () {
    char S [MAX], * sPtr ;
    while((sPtr = fgets(S,MAX,stdin))!= NULL)
        puts( S ) ;
    return 0 ;
}
```

```
#include <stdio.h>
int main () {
    int C ;    // can also use char
    while((C = getchar()) != EOF && C != '\n' )
        putchar( C ) ;
    return 0 ;
}
```

## <stdio.h> ...

52

- `int sprintf( char *s, const char *format,... );`
  - ▣ Equivalent to `printf`, except the output is stored in the array instead of printing it on the screen
- `int sscanf( char *s, const char *format, ... );`
  - ▣ Equivalent to `scanf`, except the input is read from the array instead of reading it from the keyboard

```
#include <stdio.h>
int main() {
    char str[80];
    int x = 12345;

    sprintf(str, "Value of x = %d", x);
    puts(str);

    return(0);
}
```

Value of x = 12345

## Example

53

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

int main() {
    int day, year;
    char weekday[20], month[20];
    char dtm[]="Saturday March 25 1989";

    sscanf( dtm, "%s %s %d %d",weekday, month, &day, &year);

    printf("%s %d, %d = %s\n", month, day, year, weekday);

    return(0);
}
```

March 25, 1989 = Saturday

## String Manipulation Functions <string.h>

54

- The string handling library provides many useful functions for :
  - ▣ manipulating string data
  - ▣ comparing strings
  - ▣ searching strings for characters and other strings
  - ▣ tokenizing strings (separating strings into logical pieces)
  - ▣ determining the length of strings.
- Every function, except for strncpy, appends the null character to its result.

## Copy Functions - <string.h>

55

- Two functions are provided to perform copying of one string into another string.
- **char \* strcpy( char \* Dest, const char \* Src);**
  - ▣ Copies the source string Src to the destination Dest.
  - ▣ If Src is shorter, or equal in length, to Dest, the entire string is copied.
  - ▣ If Src is longer than Dest, only those characters that will fit are copied
    - ▣ note that this may leave Dest without a delimiter '\0' (which fails to define a proper string).
- **char \* strncpy( char \* Dest, const char \* Src, size\_t N);**
  - ▣ Copies the first N characters of the source string Src to the destination Dest.
  - ▣ Remember that strncpy() does not append the delimiter automatically!

## Example 1

56

```
void strcpy(char dest[], const char src[]); //copies string src into string dest
example:
```

```
char name1[16], name2[16];
```

```
strcpy (name1, "Chan Tai Man");
```

name1: 

C	h	a	n		T	a	i		M	a	n	\0	?	?	?
---	---	---	---	--	---	---	---	--	---	---	---	----	---	---	---

```
strcpy(name2, "9999999999999999");
```

name2: 

9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	\0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----

```
strcpy(name2, name1);
```

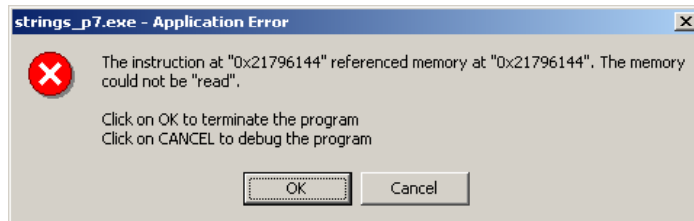
name2: 

C	h	a	n		T	a	i		M	a	n	\0	9	9	\0
---	---	---	---	--	---	---	---	--	---	---	---	----	---	---	----

## Example 2

57

```
#include <stdio.h>
#include <string.h>
int main () {
    char string_1[6] = "Short";
    char string_2[17] = "Have a Nice Day";
    char string_3[6] = "Other";
    strcpy(string_1, string_2);
    return 0;
}
```



## Concatenation Functions <string.h>

58

- Joining together of two strings is called string *catenation* (also called *concatenation*).
- **char \* strcat( char \* S1, const char \* S2);**
  - Copy string S2 to a position in S1, following the string already in S1.
  - Note that the original '\0' delimiter in S1 is overwritten by the first character in the S2 string, so that only one delimiter occurs at the end of the modified S1 string.
  - If the total number of characters is greater than the capacity of S1 then a logical error will likely ensue.
- **char \* strncat( char \* S1, const char \* S2, size\_t N);**
  - Copy the first N characters of the string S2 to a position in S1, following the string already in S1.

## Example

59

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

int main () {
    char str[80];
    strcpy (str, "these ");
    strcat (str, "strings ");
    strcat (str, "are ");
    strcat (str, "concatenated.");
    puts (str);
    return 0;
}
```

these strings are concatenated .

## Comparison Functions <string.h>

60

- Comparison of two strings is based on the notion of *lexical ordering*.
- **int strcmp( const char \* S1, const char \* S2);**
  - Compares strings S1 and S2.
  - Returns 0 if S1 == S2
  - Return a positive number if S1 is lexically greater than S2
  - Return a negative number is S1 is lexically less than S2.
- **int strncmp( const char \* S1, const char \* S2, size\_t N);**
  - Compares up to the first N characters of the strings S1 and S2.
  - Note that if the length of either S1 or S2 is less than N, the comparison is done only for the characters present in each string.

## Example 1

61

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

int main () {
    char str1[5], str2[5];
    int ret;

    strcpy(str1, "abc");
    strcpy(str2, "adc");

    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);
}
```

str1 is less than str2

## Example 2

62

```
#include <stdio.h>

int main() {
    char str1[20];
    char str2[20];

    strcpy(str1, "Hello");
    strcpy(str2, "Hellooo");
    printf("Return Value is : %d\n", strcmp( str1, str2, 4));

    strcpy(str1, "Helloooo");
    strcpy(str2, "Hellooo");
    printf("Return Value is : %d\n", strcmp( str1, str2, 10));

    strcpy(str1, "Helloooo");
    strcpy(str2, "Hellooo");
    printf("Return Value is : %d\n", strcmp( str1, str2, 20));

    return 0;
}
```

Return Value is : 0

Return Value is : 111

Return Value is : 0

## Search Functions - <string.h>

63

- C provides functions for searching for various characters and substrings within a string
  - ▣ This is a huge advantage in text processing

### Function Prototype and Description

**char \* strchr( const char \* S, int C);**

Locates the position in S of the first occurrence of C. Returns the pointer value to where C is first located; otherwise returns NULL.

**char \* strrchr( const char \* S1, int C );**

Locates the position in S of the last occurrence of C., and returns a pointer to that position in S1. Otherwise a NULL value is returned.

**char \* strstr( const char \* S1, const char \* S2 );**

Locates the first occurrence in S1 of the entire string S2. Otherwise a NULL value is returned.

## Example

64

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

int main () {
    char str[] = "This is a sample";
    char * pch;

    printf ("Looking for the 's' character:\n");
    pch=strchr(str,'s');
    while (pch!=NULL) {
        printf ("found at %d\n",pch-str);
        pch=strchr(pch+1,'s');
    }

    return 0;
}
```

Looking for the 's' character:  
found at 3  
found at 6  
found at 10

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	h	i	s		i	s		a		s	a	m	p	l	e	\0



## Split Function - <string.h>

65

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

- Split a string of text S1 that contains various words (substrings) separated by specially designated characters used as delimiters (and contained in a string S2).
- The objective is to extract the words from the text.
- A sequence of calls to strtok breaks string s1 into "tokens" – logical pieces such as words in a line of text- separated by characters contained in string s2.
- The first call contains s1 as the first argument
- Each subsequent call to strtok() uses **NULL** as the first argument (instead of the string S1), and the function remembers where it left off from the last time it was called.
- A pointer to the current token is returned by each call.
- If there are no more tokens when the function is called, NULL is returned.

## Example 1

66

- Each identified substring in S1, delimited by a character in S2, is called a *token*. Thus, strtok() is called the string tokenizer function

```
char string[] = "this is a sentence, with 7 tokens.";
char *tokenPtr;
tokenPtr = strtok( string, " " );
while( tokenPtr != NULL)
{
    printf( "%s\n", tokenPtr );
    tokenPtr = strtok( NULL, " " );
}
```

this  
is  
a  
sentence,  
with  
7  
tokens.

## Example2

67

```
#include <stdio.h>
#include <string.h>
int main () {
    int N = 0 ;
    char S[] = "This, is a test sentences." ;
    char * tokenPtr, * DelimList = " .,:;$" ;
    printf( "The following tokens were found in S.\n" ) ;

    tokenPtr = strtok( S, DelimList ) ; //First time use S;
    while( tokenPtr != NULL ) {
        N++ ;
        printf( "%s\n", tokenPtr ) ;
        // Use NULL in successive calls of strtok
        tokenPtr = strtok( NULL, DelimList ) ;
    }
    printf( "Number of tokens found = %d\n", N ) ;
    return 0 ;
}
```

The following tokens were found in S.  
This  
is  
a  
test  
sentences  
Number of tokens found = 5

## String Length- <string.h>

68

`size_t strlen( const char *s);`

- Determines the length of string s
- Returns the number of characters in S, not including the '\0' delimiter.
- **size\_t**: A type defined in string.h that is equivalent to an unsigned int

```
char A[] = "hello" ;
printf("Length of A =%d\n", strlen(A));
```

Length of A =5

## Memory Functions - <string.h>

69

- Functions for dealing with blocks of data in RAM
  - ▣ The blocks may be characters, or other data types, hence the functions typically return a **void \*** pointer value.
    - A void \* pointer value can be assigned to any other pointer type, and vice versa.
    - However, void \* pointers cannot be dereferenced, thus the size of the block must be specified as an argument.
    - None of the functions discussed perform checks for terminating null characters (delimiters).

## Memory Functions in <string.h>

70

### Function Prototype and Description

**void \* memcpy( void \* S1, const void \* S2, size\_t N );**

Copies N characters (bytes) from the object S2 into the object S1. A pointer to the resulting object (S1) is returned, otherwise NULL is returned on failure.

**int memcmp( const void \* S1, const void \* S2, size\_t N );**

Compares the first N characters (bytes) of S1 and S2. Returns 0 if S1==S2, >0 if S1>S2, and <0 if S1<S2.

**void \* memchr( const void \* S1, int C, size\_t N );**

Locates the first occurrence of the character C in the first N characters (bytes) of S1. If C is found, a pointer to C in S1 is returned. Otherwise, NULL is returned.

**void \* memset( void \* S1, int C, size\_t N );**

Copies the character (byte) C to the first N positions of S1. A pointer to S1 is returned, or NULL on failure.

Note: the type of C is modified to *unsigned char* to enable copying to blocks of arbitrary data type.

## Example

71

```
char str[] = "almost every programmer should know memset!";
memset (str, '-', 6);
puts (str);
```

----- every programmer should know memset!

```
#include <stdio.h>
#include <string.h>
int main () {
    char * pch;
    char str[] = "Example string";
    pch = (char*) memchr (str, 'p', strlen(str));
    if (pch!=NULL)
        printf (" 'p' found at position %d.\n", pch-str+1);
    else
        printf (" 'p' not found.\n");
    return 0;
}
```

'p' found at position 5

## Some Common Errors

72

- It is illegal to assign a value to a string variable (except at declaration).

```
char A_string[10];
A_string = "Hello"; // illegal assignment
```

Should use instead

```
strcpy (A_string, "Hello");
```
- The operator == doesn't test two strings for equality.

```
if (string1 == string2) // illegal comparison
```

Should use instead

```
if (!strcmp(string1, string2))
```

  - ▣ Note that strcmp returns 0 (false) if the two strings are the same.

## Question 1 - Output?

73

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

void main() {
    char a[]="abc";
    char b[]={'a','b','c','\0'};
    char d[]={'a','b','c'};
    printf("%d \n", strlen(a));
    printf("%d\n",strlen(strcpy(d,"\0")));
    printf("%s\n",d);
}
```

3  
0

## Question 2 - Output?

74

```
char s1[50] = "jack", s2[50] = "jill", s3[50];
printf( "%c%s", toupper(s1[0]), &s1[1]);
printf( "%s", strcpy(s3,s2));
printf( "%s", strcat(strcat(strcpy(s3,s1)," and "), s2));
printf( "%u", strlen(s1) + strlen(s2));
printf( "%u", strlen(s3));
```

Jack  
jill  
jack and jill  
8  
13

## Lecture 5: Summary

75

- Characters and Strings in the C language
  - Multiple library sources
  - Query functions
  - Transformative functions
  - Conversion functions
  - Memory functions
- Assigned Reading:
  - Chapter 8 – Characters and Strings in the C language
  - Chapter 9 – Formatted Input and Output
    - This chapter is straightforward and is assigned for self-directed independent study and learning – it will be tested!
- Assignment
  - Complete the two-stage survey