

Lecture 5B: C String

Content

- I. Introduction: character array
- II. C Strings
- III. Array of Strings
- IV. C- string Manipulation

Reference: <https://www.cplusplus.com/reference/cstring/>

I. Introduction: character array

- Character array is an array with elements of the type `char`

Example:

- Number of minutes is ranging from 0 to 59. Thus, `char` is sufficient to store these small values

- `char minutes[5] = {10,15,40,47,55}; minutes[2] = 35;`

- `if(minutes[1] == 10)`

- `minutes[2] = 0;`

- Characters which represent academic grades

- `char grades[]={‘F’, ‘P’, ‘C’, ‘D’};`

- There is no fundamental difference between arrays of `char` type and other types besides that **you can initialize them with characters.**
- **Character arrays are commonly used to store textual information rather than numbers.**

II. C Strings

- ❖ C String is a **grouping of characters with a terminating NULL character '\0'**
- ❖ You can create **Strings** using a char array with a **terminating NULL character '\0'**.


The latter is used to mark the end of a character string. C C-String functions rely on the NULL character to confirm the end of the string.

- ❑ Initialisation during the declaration, e.g. `char firstName[4] = {'A', 'L', 'I', '\0'};`
 - make sure you allocate enough room for the NULL character '\0'
 - The size of the char array should be at least one element higher the length of the c-string
- ❑ You can also create strings with a **string literal**: a grouping of characters enclosed in quotation marks, e.g. `char myName[] = "ALI";`
 - Doing so this way **automatically appends** the **NULL character** and assigns the necessary number of elements

```
char firstName[3] = "ALI"; /* not enough */  
char firstName[4] = "ALI"; /*OK */
```

C-string Input/Output

- ❖ Use the format specifier `%s`
- ❖ Example: Consider `char name[31];`

`scanf("%s", name);`  `&` is not needed for c-strings

- The function will store the input string into `name`
- If the user's input includes a white space; the white space and what come after won't be stored in the variable `name`
- The input string must not be longer than `array_size - 1` (30 in this example, otherwise it **may cause you program crash**)
- To limit the input length, set the format specifier accordingly, a NULL will be added to your array :

`scanf("%30s", name); /* get no more than 30 characters */`

`printf("%s", name);` 

- will output the content on the screen from `name` on the screen until the NULL terminator (`name` must be a **NULL** terminated c-string!)

Example – String Input / Output

Write a program to ask the user to type in a string representing an academic department, an integer course code, a string abbreviations for the days of the course the meets, and an integer that gives the meeting time of the class.

```
1. #include <stdio.h>
2.
3. #define STRING_LEN 10
4.
5. int
6. main(void)
7. {
8.     char dept[STRING_LEN];
9.     int course_num;
10.    char days[STRING_LEN];
11.    int time;
12.
13.    printf("Enter department code, course number, days and ");
14.    printf("time like this:\n> COSC 2060 MWF 1410\n> ");
15.    scanf("%s%d%s%d", dept, &course_num, days, &time);
16.    printf("%s %d meets %s at %d\n", dept, course_num, days, time);
17.
18.    return (0);
19. }
```

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

dept
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
M A T H \0 ? ? ? ? ?
↑ ↑ ↑ ↑
data entered
skipped if present
triggers storage of '\\0' ...

Reading Strings with white space

```
char *fgets(char *str, int n, FILE *stream)
```

- The function **fgets()** reads a line from **stdin** (or a file stream) and stores it into the string (**str**) pointed to. It stops when either **(n-1)** characters are read, the **newline** character is read, or the **end-of-file** is reached, whichever comes first

```
# define SIZE 20
int main()
{
    char countryName[SIZE];
    printf("Enter a string\n");
    fgets(countryName, SIZE, stdin); // works with multiple words!
    printf("The string: %s\n", countryName);
    return 0;
}
```

III. Array of Strings

- An array of strings is a two-dimensional array of characters in which each row is one string.

Example: Declare an array to store up to 30 names, each of which is less than 25 characters long.

```
#define NUM_PEOPLE 30 /*number of people*/  
#define NAME_LEN 25  /*name length*/  
  
.....  
char names [NUM_PEOPLE][NAME_LEN] ={ '\0' };
```

OR

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

Example

```
int main()
{
    char colours [3][10] = {'\0'};

    printf("Enter 3 colours separated by spaces:");

    scanf("%s %s %s", colours[0], colours[1], colours[2]);

    printf("you entered..:");

    printf("\n%s \n%s \n%s", colours[0], colours[1], colours[2]);

}
```


IV. C-string Manipulation

- C provides a set of standard functions for C-string manipulation
- You need to include `<string.h>` header file to use these functions
- These functions **operate on C-strings**, a char array which ends with **NULL** character

Function name	Description
<code>strcat</code> (s1, s2)	Appends s2 to s1 (also <code>strncat</code>)
<code>strcmp</code> (s1, s2)	Return 0 if (s1 is same as s2), -1 if (s1 < s2), 1 if (s1 > s2)
<code>strcpy</code> (s1, s2)	copy s2 to s1
<code>strncpy</code> (s1, s2, n)	copy n characters from s2 to s1 (substrings)
<code>strlen</code> (s1)	return the length of s1 up to (but not including) the null character
... •

Strings Concatenation: strcat

```
char * strcat ( char * destination, const char * source );
```

- Concatenate or append a copy of the *source* string to the *destination* string.
 - The size of *destination* should be long enough to hold the copy of *source*. If not, we will get the **segmentation fault error**.
- The terminating NULL character in ***destination*** is **overwritten** by the first character of *source*, and a **null-character** is included at the end of the new string formed by the concatenation of both in *destination*.
- The function returns the concatenated string

Strings concatenation: `strncat`

```
char * strncat ( char * destination, const char * source, size_t num );
```

- Appends the first ***num*** characters of *source* to *destination*, **plus a terminating null-character**.
- If the length of the C string in *source* is less than *num*, only the content up to the terminating null-character is copied.

C-strings Comparison

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

- This function starts comparing the first character of each string using their corresponding ASCII codes. If they are equal to each other, it continues with the following pairs until the characters differ or until a terminating null-character is reached.

Examples:

The string "Air" is smaller than the string "Boat".

ASCII codes: **65** 105 114 0 **66** 111 97 116 0



The string "Air" is smaller than the string "An".

ASCII codes: 65 **105** 114 0 65 **110** 0



C-string comparison

Relationship	Value Returned	Example
<code>str1</code> is less than <code>str2</code>	negative integer	<code>str1</code> is "marigold" <code>str2</code> is "tulip"
<code>str1</code> equals <code>str2</code>	zero	<code>str1</code> and <code>str2</code> are both "end"
<code>str1</code> is greater than <code>str2</code>	positive integer	<code>str1</code> is "shrimp" <code>str2</code> is "crab"

```
if( strcmp(Str1, Str2) == 0 )  
    /* strings are equal */  
else  
    /* strings are not equal */
```

Strings copy: strcpy

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

- Copies the string source *src* into another string *dest*, **including the terminating NULL character and stopping at that point.**
- The function returns the value of the first argument

```
char str1[15]= "Ali Al-Baz"  
char str2[] = "Eva Rodriguez";  
printf("String 1 says %s", strcpy(str1,str2));
```

- To avoid overflows, the size of the array pointed by *dest* shall **be long enough** to contain the same C string as *source* (including the terminating NULL character)

Strings copy: strncpy

```
char * strncpy ( char * destination, const char * source, size_t num );
```

- Copies the first num characters of `source` to `destination`.
 - If the end of the `source` C string (which is signaled by a null-character) is found before `num` characters have been copied, destination is padded with zeros until a total of `num` characters have been written to it.
- No null-character is implicitly appended at the end of `destination` if `source` is longer than `num`. Thus, in this case, `destination` shall not be considered a **NULL terminated C string** (reading it as such would overflow).
 - You can however add the NULL terminator to `destination`

Example

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

int main () {
char str1[]= "To be or not to be"; // 18 chars+NULL
char str2[40];
char str3[40];
strncpy ( str2, str1, sizeof(str2) );
strncpy ( str3, str2, 5 ); /* partial copy (only 5 chars): */
str3[5] = '\0'; /* null character manually added */
puts (str2);
puts (str3);
return 0;
}
```

Output: To be or not to be
To be

Strings search: strstr

```
char * strstr ( char * str1, const char * str2 );
```

- Locate substring *str2* in *str1*, useful for word searches
- Returns a pointer to the first occurrence of *str2* in *str1*, or a **NULL** pointer if *str2* is not part of *str1*.
- The matching process does not include the terminating null-characters, but it stops there.

```
char str1[] = "Analysing strings for certain words and things";  
char str2[] = "ing";  
char str3[] = "xyz";  
if (strstr(str1, str2) != NULL)  
    printf("str2 was found in str1");  
else  
    printf("was not found");
```

String length: strlen()

- It is part of the string handling library <string.h>.
- It returns the numeric length of the string up to the terminating **NULL** character (but does not include it)

```
char myName[] = "Ali";  
printf("Your name has %d letters in it..", strlen(myName));
```

Output: Your name has 3 letters in it..

Character Analysis and Conversion

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

Facility	Checks	Example
<code>isalpha</code>	if argument is a letter of the alphabet	<pre>if (isalpha(ch)) printf("%c is a letter\n", ch);</pre>
<code>isdigit</code>	if argument is one of the ten decimal digits	<pre>dec_digit = isdigit(ch);</pre>
<code>islower</code> (<code>isupper</code>)	if argument is a lowercase (or uppercase) letter of the alphabet	<pre>if (islower(fst_let)) { printf("\nError: sentence "); printf("should begin with a "); printf("capital letter.\n"); }</pre>
<code>ispunct</code>	if argument is a punctuation character, that is, a noncontrol character that is not a space, a letter of the alphabet, or a digit	<pre>if (ispunct(ch)) printf("Punctuation mark: %c\n", ch);</pre>
<code>isspace</code>	if argument is a whitespace character such as a space, a newline, or a tab	<pre>c = getchar(); while (isspace(c) && c != EOF) c = getchar();</pre>
Facility	Converts	Example
<code>tolower</code> (<code>toupper</code>)	its lowercase (or uppercase) letter argument to the uppercase (or lowercase) equivalent and returns this equivalent as the value of the call	<pre>if (islower(ch)) printf("Capital %c = %c\n", ch, toupper(ch));</pre>

Character case: tolower(), toupper()

- The two functions **tolower()** and **toupper()** apply on a single character, To use them on strings, use loop.

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

int main(){
    char firstname[] = "ali";
    char lastname[] = "BENZ";
    convertL(lastname);
}

void convertL(char str[])
{
    for (int x=0; x<strlen(str);x++)
        str[x] = tolower(str[x]);
    printf("last name in lowercase is %s \n", str);
}
```

Strings to Numbers Conversion

- ASCII language uses letters, digits and special characters.
- We use integral and float data type to represent values in the calculations
- C standard library `stdlib.h` provides several functions that convert strings, as char digits, to numbers

- Two common functions:

- **atof** :converts string to a floating point number
- **atoi**: converts string to an integer

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

int main(){
    char str1[]="123.79";
    char str2[]="55";
    float x;
    int y;
    x=atof(str1);
    y=atoi(str2);
    printf("\nstr1 is %s which corresponds to the value %.2f", str1,x);
    printf("\nstr2 is %s which corresponds to the value %d", str2,y);
}
```

Example1: separate components into elemental components

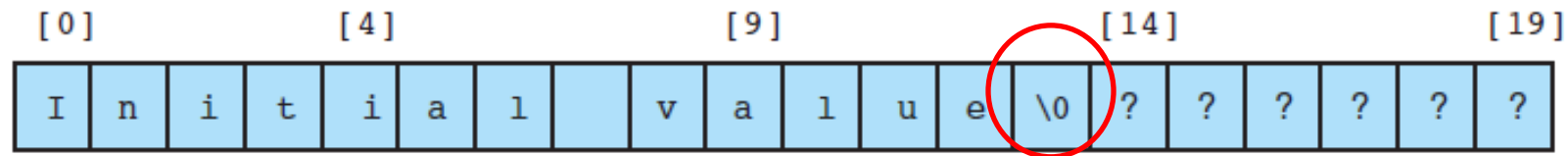
A program to break compounds into their elemental components, assuming that each element name begins with a capital letter

```
1. /*
2.  * Displays each elemental component of a compound
3.  */
4.
5. #include <stdio.h>
6. #include <string.h>
7.
8. #define CMP_LEN 30 /* size of string to hold a compound */
9. #define ELEM_LEN 10 /* size of string to hold a component */
10.
11. int
12. main(void)
13. {
14.     char compound[CMP_LEN]; /* string representing a compound */
15.     char elem[ELEM_LEN];    /* one elemental component */
16.     int first, next;
17.
18.     /* Gets data string representing compound */
19.     printf("Enter a compound> ");
20.     scanf("%s", compound);
21.
22.     /* Displays each elemental component. These are identified
23.      by an initial capital letter. */
24.     first = 0;
25.     for (next = 1; next < strlen(compound); ++next)
26.         if (compound[next] >= 'A' && compound[next] <= 'Z') {
27.             strncpy(elem, &compound[first], next - first);
28.             elem[next - first] = '\0';
29.             printf("%s\n", elem);
30.             first = next;
31.         }
32.
33.     /* Displays the last component */
34.     printf("%s\n", strcpy(elem, &compound[first]));
35.
36.     return (0);
37. }
```

```
Enter a compound> H2SO4
H2
S
O4
```

Example 2: C String Initialisation

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



- Str[13] contains the character `'\0'`, the **NULL character** that marks the end of a string.
- A blank (empty space) in a string is a valid character.
- A string constant can be associated with a symbolic name using the `#define` directive.

```
#define ERR_PREFIX "*****Error - "  
#define INSUFF_DATA "Insufficient Data"
```