

CHAR ARRAYS (C-STRING) - TEXT REPRESENTATION IN C/C++

Text constant is represented as an array of characters terminated by ASCII code: **0**

eg. string: **"Jestem tekstem"**

ASCII →	...	'J'	'e'	's'	't'	'e'	'm'	' '	't'	'e'	'k'	's'	't'	'e'	'm'	'\0'	...
values→	...	74	101	115	116	101	109	32	116	101	107	115	116	101	109	0	...
addresses →	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Definitions and initialization of strings - „text” variables:

```
char text [ ] = { 'J', 'e', 's', 't', 'e', 'm', ' ', 't', 'e', 'k', 's', 't', 'e', 'm', '\0' };  
char text2[ ] = { "Jestem tekstem" };  
char text3[ ] = "Jestem tekstem";  
char text4[100] = "Jestem tekstem";  
  
char * text5;           // pointer to char == pointer to the beginning of text  
text5 = "Jestem tekstem"; // assignment of the text pointer to variable text5  
  
text5 = text4 ;         // correct assignment pointers  
  
char text6[100];        // 100-element array of characters  
text6 = "Jestem tekstem"; // incorrect assignment !!!  
memcpy( text6, "Jestem tekstem", 15 ); // correct  
strcpy ( text6, "Jestem tekstem" );    // correct
```

Examples of the operations on single letters of the text:

```
text [1] = 'E';           // replace the second character  
text [2] = text [2] - 32; // conversion of the third character  
text [3] = toupper( text [3] ); // conversion to uppercase  
  
for( int i=4; i<10; i++)  
    text [ i ] = toupper( text [ i ] ); // conversion of character sequence  
  
text [5] = '\0';         // truncate the text to 5 letters  
  
for( int i=0; text [ i ] != '\0' ; i++)  
    printf( "%c" , text[ i ] ); // print the contents of the text (v.1)  
  
for( char* ptr=text; *ptr ; ptr++)  
    printf( "%c" , *ptr ); // print the contents of the text (v.2)
```

Example of word processing → adding the extension „**.txt**” at the end of the filename

```
#include <stdio.h>

int main(void)
{
    char name [100];
    printf( "Enter the filename: " );
    fgets(name, 100, stdin );    // or alternatively:  gets( name );

    // searching for the last „dot” in the sequence of characters
    int i, dot_position = -1;
    for(i=0; name [i] != '\0'; i++)
        if( name [i] == '.')
            dot_position = i;

    // checking the presence of the extension txt
    bool is_txt=false;
    if(dot_position!=-1)
        if( name [dot_position +1]=='t' && name [dot_position +2]=='x' &&
            name [dot_position +3]=='t' && name [dot_position +4]=='\0' )
            is_txt=true;

    // append the ".txt" extension (if it is not there)
    if( ! is_txt )
    {
        name [i+0] = '.';    // variable 'i' still indicates the end of name
        name [i+1] = 't';
        name [i+2] = 'x';
        name [i+3] = 't';
        name [i+4] = '\0';
    }

    // // same as above, but using the ready-to-use functions from <string.h>
    // char* dot_position =strchr(name, '.');
    // bool is_txt=false;
    // if(dot_position && strcmp(dot_position, ".txt" )==0 )
    //     is_txt=true;
    // if( !is_txt )
    //     strcat(name, ".txt");

    // display the final result – filename with txt extension at the end
    printf( " \n\n Name with extension \"txt\" = [ %s ]\n" , name );

    printf( "Press ENTER, to continue" );
    fflush( stdin );
    getchar( );
    return 0;
}
```

Copying the contents of one character array to another („string copy”).

Declaration:

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

```
// exemplary implementation (using square-bracket operator notation)
```

```
char * strcpy( char destination [ ], char source [ ] )  
{  
    int i = 0;  
    while( (destination [ i ] = source [ i ] ) != '\0' )  
        i++;  
    return(destination );  
}
```

```
// copy one string to another → version using the pointer notation (1)
```

```
char * strcpy( char * destination, char * source )  
{  
    char *ptr= destination;  
    while( ( *destination = *source ) != '\0' )  
    {  
        destination++;  
        source++;  
    }  
    return( ptr );  
}
```

```
// function that copies strings – version using the pointer notation (2)
```

```
char * strcpy( char *destination, char *source )  
{  
    char *ptr= destination;  
    while( *destination++ = *source++ );  
    return( ptr );  
}
```

```
// Copying, while limiting the length of the copied string
```

```
char * strncpy( char destination[ ], char source[ ], unsigned max_length )  
{  
    int i = 0;  
    while( (destination [ i ] = source[ i ] ) != '\0' && i < max_length )  
        i++;  
    return(destination);  
}
```

Function to compare strings: **int strcmp (char *text_1, char *text_2)**
(„string compare”)

the function returns: < 0 *when* text_1 < text_2
 = 0 *when* text_1 == text_2
 > 0 *when* text_1 > text_2

```
int strcmp( char text_1[ ], char text_2[ ] )           // array notation
{
    int i = 0;
    while( text_1[ i ] == text_2[ i ] )
        if( text_1[ i++ ] == '\0' )
            return( 0 );
    return( text_1[ i ] - text_2[ i ] );
}
```

```
int strcmp( char *text_1, char *text_2 )             // pointer notation (1)
{
    while( *text_1 == *text_2 )
    {
        if( *text_1 == '\0' )
            return( 0 );
        text_1 = text_1 + 1;
        text_2 = text_2 + 1;
    }
    return( *text_1 - *text_2 );
}
```

```
int strcmp( char *text_1, char *text_2 )             // pointer notation (2)
{
    for( ; *text_1 == *text_2 ; text_2++ )
        if( !*text_1++ )
            return( 0 );
    return( *text_1 - *text_2 );
}
```

```
...                                     // exemplary application of strcmp
char text[100];
scanf( "%s", text );
if( strcmp( text, "Kowalski" ) == 0 )
    printf( "You have just entered the text: \"Kowalski\" " );
...
```

Selected functions from library <string.h>

```
size_t strlen( const char *s )
```

„**string length**”

This function calculates and returns the length / number of characters (without '\0')

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

„**string concatenate**”

This function appends the string src (*source*) to the end of dest (*destination*)

Returns the pointer to the concatenated string (dest)

```
char *strchr( const char *s, int c )
```

„**string char**”

Function searches for the FIRST occurrence of the character c in the given string s

Returns the pointer to the found occurrence or **NULL** (if not found)

```
char *strrchr( char *s, int c )
```

„**string right char**”

Function searches for the LAST occurrence of the character c in the given string s

Returns the pointer to the found occurrence or **NULL**.

```
char *strstr( char *s, const char *sub )
```

„**scans string for substring**”

Function searches for the FIRST occurrence of the substring sub in the given s

Returns a pointer to the found entry of address or **NULL**.

```
char*strupr( char *s )
```

„**string upper**”

The function converts the contents of the string s to uppercase

```
char*strlwr( char *s )
```

„**string lower**”

The function converts the contents of the string s to lowercase

Examples of C-string processing

```
1) #include <stdio.h>                                // example of converting ALL letters to upercase
    #include <ctype.h>

    // standard functions converting the text to upercase / lowercase
    // #include <string.h> → char *strupr (char *s);   char *strlwr (char *s);

char *Convert_To_Upper ( char* text )
{
    char *ptr = text;
    do
        *ptr = toupper(*ptr );           // conversion of a single letter to upercase
    while(*ptr++ );
    return( text );
} //----- Convert_To_Upper

int main( void )
{
    char text_array [100] = "abcdefghijklmnopqrstuvwxyz";
    printf( "%s\n" , Convert_To_Upper ( text_array ) );
    return 0;
}
```

```
2) #include <stdio.h>                                // example of converting FIRST letters to upercase
    #include <ctype.h>

char *Words_To_Upper ( char* text )
{
    char *ptr = text;
    if( !* ptr )                                     // exit, if the text is empty
        return(text);
    * ptr = toupper( * ptr );                         // replace the first letter
    while( *++ ptr )
        if( *( ptr -1) == ' ' )                     // if the preceding character is a space
            * ptr = toupper( * ptr );               // convert the character to upper
    return( text );
} //----- Words_To_Upper

int main( void )
{
    char text_array[100] = "this is the example of text ";
    printf( "%s\n" , Words_To_Upper ( text_array ) );
    return 0;
}
```

```

3) #include <stdio.h>           // function that finds and replaces the sections of text
    #include <string.h>        // C-string

void Replace_Section ( char* text,
                      char* old_pattern,
                      char* new_text )
{
    char* ptr = text;
    int length_of_old = strlen( old_pattern );
    int length_of_new = strlen( new_text );
    do {
        ptr = strstr( text, old_pattern );
        if( ptr )                // if( ptr != null )
        {
            // ewentualne zsunięcie lub rozsuniecie tekstu
            memmove( ptr + length_of_new ,
                    ptr + length_of_old ,
                    strlen(ptr + length_of_old ) +1 );
            // wpisanie nowego wzorca w przygotowane miejsce
            memcpy(ptr, new_text, length_of_new);
        }
    } while(ptr );
} //----- Replace_Section

int main( void )
{
    char text[200] = "Ala ma kota a Ola ma Asa";
    printf( "Initial contents of the text: %s\n" , text );
    Replace_Section ( text, "ma", "miala" );
    printf( " After replacement: %s\n" , text );    // "Ala miala kota a Ola miala Asa"
    return 0;
}

```

ATTENTION !

- Application of standard function strcpy instead of memmove (in above example) will generate errors (when new_text will be longer than old_pattern)
eg. `strcpy(ptr+length_of_new, ptr+length_of_old);`
will create text: " Ala ma ko ko ko ko ko ko ko k"
- Definition: `char* tekst = "Ala ma kota a Ola ma Asa";`
is equivalent to: `char tekst[24+1] = "Ala ma kota a Ola ma Asa";`
During the conversion, the text can be extended (by putting longer fragments), so the text array variable should be longer than the length of initiating text.