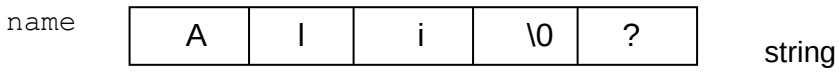
Strings

- In fact, a string is nothing but a character array. Each element of that array contains only one character. The C language uses **\0** (NULL character) to mark the end of a string. We can initialize a character array to a string during its declaration.

Example:

```
char name[5] = "Ali";
```

- As a result of the above statement, the C language will put the **\0** character in the 4th element (with index 3), therefore warning itself that the rest of the array is not used.

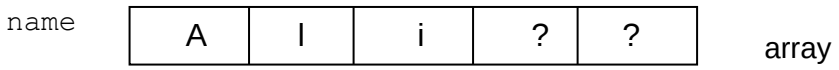


string

- As you know, We can also initialize the `name` array as

```
char name[5] = {'A', 'l', 'i'};
```

- What is the difference between them?



array of characters

- So, the longest string the `name` array above can represent is 4 characters plus the NULL character. But, you can store 5 characters in it using the second way.

```
char name[5] = "Aliye"; // COMPILATION ERROR!
char name[5] = {'A', 'l', 'i', 'y', 'e'}; // NO PROBLEM
```

- A string may also consist of a single character. For instance, the difference of writing `'a'` and `"a"` within a program is that the first one is a character and can be put into a one-byte character variable. The second one is a string and can not be put into a one-byte character variable. It can be put into a character array with the minimum size 2, because the computer stores the NULL character at the end of it.

```
char ch = 'a';
char ch = "a"; // COMPILATION ERROR
char ch[2] = "a";
```

Example: Write a function that finds the length of a string.

```
int strLength(char str[]) {
      int k = 0;
      // Go until the end of the string
      while (str[k] != '\0')
          k++;
      return (k);
}
```

Example: Write a function that finds the length of a string using pointers.

```
int strLength(char *str) {
      char *p = str;
      // Go until the end of the string
      while (*p != '\0')
          p++;
      return (p – str);
}
```

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

Example:

```
char week[7][10] = {"monday", "tuesday", "wednesday",
                    "thursday", "friday", "saturday" , "sunday"};
```

m	o	n	d	a	y	\0			
t	u	e	s	d	a	y	\0		
w	e	d	n	e	s	d	a	y	\0
t	h	u	r	s	d	a	y	\0	
f	r	i	d	a	y	\0			
s	a	t	u	r	d	a	y	\0	
s	u	n	d	a	y	\0			

```
printf("%c\n", week[2][0]); //What is the output?
```

Input/Output of Strings

- Upto now we used loops to input or output character arrays. The C programming language can input or output the whole string using the **%s** placeholder.

Example:

```
printf("%s\n", week[1]);
```

will output the string **tuesday**. What about

```
printf("%s\n", week);
```

It will output the string **monday**.

```
printf("%s\n", *week);
```

will also output the string **monday**.

- `printf` displays the characters in the string until NULL character is encountered. NULL character is automatically put in the constant strings.

- If no NULL character is found in the character array, `printf` would continue to print as many characters as it can find in the following memory locations until it finds a NULL character or until a run-time error is caused.

Example: What will be output for the following code segment?

```
char name1[5] = "Esin";
printf("%s\n", name1);
char name2[5] = {'D', 'e', 'n', 'i', 'z'};
printf("%s\n", name2);
```

- What about

```
printf("%s\n", name1+2);
```

- What about

```
printf("%c\n", *name1);
printf("%c\n", *(name1+2));
```

- We can format our string output using an integer between **%** and **s** in the **%s** placeholder. If that integer is positive, the string is right-justified (the left side of it stays blank). If it is negative, the string is left-justified (the right side of it stays blank). If it is smaller than the length of the string, the full string is displayed (with no blanks on the left or right side).

Example:

`printf("***%10s***%-8s***%3s***\n", week[6], week[0], week[4]);` will output

*****_ _ _ _sunday***monday_ _***friday*****.

- When we use **%s** as a placeholder in `scanf`, it stops scanning whenever a white space character (blank, newline or tab) is encountered.

Example:

```
char std_name[20];
scanf("%s", std_name);
```

If the input is **Ekin Kara**, only **Ekin** will be stored into `std_name`.

E	k	i	n	\0	?	?	?	?	?	?	?	?	?	?	?	?	?	?

- Notice that, in `scanf` we did not put **&** in front of the variable name, because the name of an array represents the address of its starting position.

- A string can be input using a *scan set* or an *inverted scan set*. The *scan set* stops inputting characters when a character not contained in the scan set is encountered.

Scan Set:

```
%[a-z]        : input only lowercase letters
%[0-9]          : input only digits
%[a-zA-z]     : input both lowercase and uppercase letters
%[aeiouAEIOU] : input only vowels
```

- The *inverted scan set* causes the characters not appearing in the scan set to be stored until a character contained in the inverted scan set is encountered.

Inverted Scan Set:

```
%[^0-9]          : input anything until a digit
%[^aeiouAEIOU]   : input everything until a vowel
%[^\n]           : input everything until a newline
```

Example:

```
char str1[10], str2[10], str3[10], str4[10];
printf("Enter a string: ");
scanf("%[0-9]%[a-z]%[^a-z]%[^\n]", str1, str2, str3, str4);
printf("***%s***%s***%s***%s***\n", str1, str2, str3, str4);
```

Output:

```
Enter a string: 10years AGO here
```

- You can include any character in the inverted scan set to indicate the end point of the input of a string.

Example:

```
char str[50];
printf("Enter a string (finish with a question mark): ");
scanf("%[^?]", str);
```

➢ *READ from Deitel & Deitel.*

- Remember that we could input and output a single character with `getchar` and `putchar` functions. C language provides functions for also input and output of strings: `gets` and `puts`.

```
char *gets(char *s)
int puts(const char *s)
```

- The **`gets`** function gets a string from the keyboard (until a newline or end-of-file character is encountered) and stores it into a character array and returns a pointer to that array. A terminating NULL character is appended to the array.
- The **`puts`** function prints a string followed by a newline character.

Example:

```
char line[80];
char *line_ptr;
printf("Type in a line of data: ");
line_ptr = gets(line);  //  scanf("%[^\n]", line);
                            //    line_ptr = line;
puts(line);                 //    printf("%s\n", line);
puts(line_ptr);             //    printf("%s\n", line_ptr);
```

Output:

```
Type in a line of data: This is the sample sentence.
```

Character Handling Library

- The character handling library with the header file **<ctype.h>** includes several functions that perform useful tests and manipulations of character data. Each function receives a character represented as a one-byte integer.

- The following functions are boolean functions. Thus, they return a true (nonzero) value if the condition checked is true, otherwise return 0.

```
int islower(int c)  // Is c a lowercase letter?
int isupper(int c)  // Is c an uppercase letter?
int isalpha(int c)  // Is c a letter?
int isalnum(int c)  // Is c a digit or a letter?
int isdigit(int c)  // Is c a decimal digit (0 – 9)?
int isxdigit(int c)         // Is c a hexadecimal digit (0 – F)?
int isspace(int c)  // Is c a whitespace character (space, newline, tab, form feed)?
int iscntrl(int c)  // Is c a control character?
int isprint(int c)  // Is c a printing character (including space)?
int isgraph(int c)  // Is c a printing character (other than space)?
int ispunct(int c)  // Is c a printing character (other than a letter, digit and whitespace)?
```

- The following functions converts a character to lowercase or uppercase.

```
int tolower(int c)      // converts c to lowercase if appropriate
int toupper(int c)      // converts c to uppercase if appropriate
```

➢ *READ Sec. 8.3 from Deitel & Deitel.*