Chapter Three
Strings

3. What is String?

- A string is simply series of characters.
- In all programs and concepts, we have seen so far, we have used only numerical variables, used to express numbers exclusively. But in addition to numerical variables there also exist strings of characters that allow us to represent successive characters, like words, sentences, names, texts, etc. Until now we have only used them as constants, but we have never considered variables able to contain them.
- In C++ there is no specific elementary variable type to store string of characters. In order to fulfill this feature, we can use arrays of type char, which are successions of char elements. Remember that this data type (char) is the one used to store a single character, for that reason arrays of them are generally used to make strings of single characters.
- For example, the following array (or string of characters) can store a string up to 20 characters long. You may imagine it thus:
  char name [20];
  name

- This maximum size of 20 characters is not required to be always fully used. For example, name could store at some moment in a program either the string of characters "Hello" or the string "studying C++”. Therefore, since the array of characters can store shorter strings than its total length, there has been reached a convention to end the valid content of a string with a null character, whose constant can be written as '\0’.
- We could represent name (an array of 20 elements of type char) storing the strings of characters "Hello" and "Studying C++" in the following way:

H	e	l	l	o	\0														

S	t	u	d	y	i	n	g		C	+	+	\0							

- Notice how after the valid content it is included a null character ('\0') in order to indicate the end of string. The empty cells (elements) represent indeterminate values.
- A string is an array of characters whose last character is \0. A typical string, such as ***Pacifique***, is graphically represented as follows:

P a c i f i q u e \0

P	a	c	i	f	q	u	e	\0	

- The last character \0 is called the null-terminating character. For this reason, a string is said to be null-terminated. The string library ships with a lot of functions used to perform almost any type of operation on almost any kind of string. Used under different circumstances, the string functions also have different syntaxes.
- The strings that you can use in your program may be defined in various libraries depending on your compiler but most of the time, they are available once you include the string library that is defined in the **std namespace**. For example, the functions we are about to review are part of the C language. The strings that are part of the (C++) Standard Template Library (STL) are defined in the string class of the **std namespace**. Based on this, most compilers make all these functions accessible once you include the string library and the std namespace in your program.
- Generally C++ provides following two types of string representations:
    - The C-style character string representation and.
        - For example
        - char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
        - char greeting[] = "Hello";
    - The string class type introduced with Standard C++ library representation.
        - String class library **string.h**
        - string str;

3.1 Initialization of Strings

- Because strings of characters are ordinary arrays they fulfill same rules as any array. For example, if we want to initialize a string of characters with predetermined values we can do it in a similar way to any other array:

  char mystring[] = { 'H', 'e', 'l', 'l', 'o', '\0' };

- In this case we would have declared a string of characters (array) of 6 elements of type char initialized with the characters that compose Hello plus a null character '\0'.
- Nevertheless, string of characters have an additional way to initialize its values: using constant strings.
- In the expressions we have used in examples of previous chapters there have already appeared several times constants that represented entire strings of characters. These are specified enclosed between double quotes (" "), for example:

Eg: "the result is:" is a constant string that we have probably used in some occasion.

- Unlike single quotes (‘) which allow to specify single character constants, double quotes (“) are constants that specify a succession of characters. These strings enclosed between double quotes have always a null character (‘\0’) automatically appended at the end.
- Therefore we could initialize the string mystring with values by any of these two ways:

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

- In both cases the Array or string of characters mystring is declared with a size of 6 characters (elements of type char): the 5 characters that compose Hello plus a final null character ('\0') which specifies the end of the string and that, in the second case, when using double quotes (") it is automatically appended.
- Before going further, you should note that the assignation of multiple constants like double-quoted constants (“) to arrays are only valid when ***initializing the array***, that is, at the moment when declared.

- The following expressions within a code are not valid for arrays

mystring="Hello"; mystring[] = "Hello";

- Neither would be: mystring = { 'H', 'e', 'l', 'l', 'o', '\0' };
- So remember: ***We can "assign" a multiple constant to an Array only at the moment of initializing it***. The reason will be more comprehensible when you know a bit more about pointers, since then it will be clarified that an array is simply a constant pointer pointing to an allocated block of memory. And because of this constant feature, the array itself cannot be assigned any value, but we can assign values to each of the elements of the array.
- At the moment of initializing an Array it is a special case, since it is not an assignation, although the same equal sign (=) is used. Anyway, have always present the rule previously underlined.

3.2 Assigning Values to Strings

- Just like any other variables, array of character can store values using assignment operators. But the following is not allowed.
  mystring=”Hello”;
- This is allowed only during initialization. Therefore, since the ***lvalue*** of an assignation can only be an element of an array and not the entire array, what would be valid is to assign a string of characters to an array of char using a method like this:

mystring [0] = 'H'; mystring[1] = 'e'; mystring[2] = 'l'; mystring[3] = 'l'; mystring[4] = 'o'; mystring[5] = '\0';

- But as you may think, this does not seem to be a very practical method.
  - Generally, for assigning values to an array, and more specifically to a string of characters, a series of functions like strcpy are used. strcpy (string copy) is defined in the (string.h) library and can be called the following way:
  - strcpy (string1, string2);
  - This does copy the content of string2 into string1. string2 can be either an array, a pointer, or a constant string, so the following line would be a valid way to assign the constant string "Hello" to mystring:

    For example:

    ```
    #include <iostream.h>
    #include <string.h>
    int main ()
    {
            char szMyName [20];
            strcpy (szMyName,"Abebe");
            cout << szMyName;
            return 0;
    }
    ```

  - Look how we have needed to include <string.h> header in order to be able to use function strcpy.
- Another frequently used method to assign values to an array is by using directly the input stream (cin). In this case the value of the string is assigned by the user during program execution.
  - When cin is used with strings of characters it is usually used with its getline method, that can be called following this prototype:

    ```
            cin.getline (char buffer [], int length, char delimiter = ' \n');
    ```

    - Where
      - buffer is the address where to store the input (like an array, for example),
      - length is the maximum length of the buffer (the size of the array) and
      - delimiter is the character used to determine the end of the user input, which by default - if we do not include that parameter - will be the newline character (‘\n').
  - The following example repeats whatever you type on your keyboard. It is quite simple but serves as example on how you can use cin.getline with strings:

    ```
            // cin with strings
            #include <iostream.h>
            #include<conio.h>
            int main ()
            {
                    char mybuffer [100];
                    cout << "What's your name? ";
                    cin.getline (mybuffer,100);
                    cout << "Hello " << mybuffer << ".\n";
    ```

```
        cout << "Which is your favourite team? ";
        cin.getline (mybuffer,100);
        cout << "I like " << mybuffer << " too.\n";
        getch();
        return 0;
    }
```

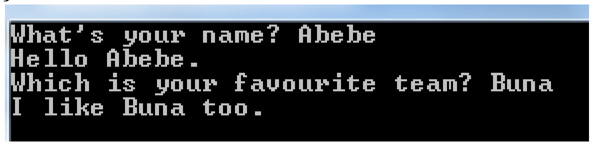
- Notice how in both calls to cin.getline we used the same string identifier (mybuffer). What the program does in the second call is simply step on the previous content of buffer by the new one that is introduced.

- If you remember the section about communication through console, you will remember that we used the extraction operator **(>>**) to receive data directly from the standard input. This method can also be used instead of cin.getline with strings of characters. For example, in our program, when we requested an input from the user we could have written:

  cin >> mybuffer;

For example

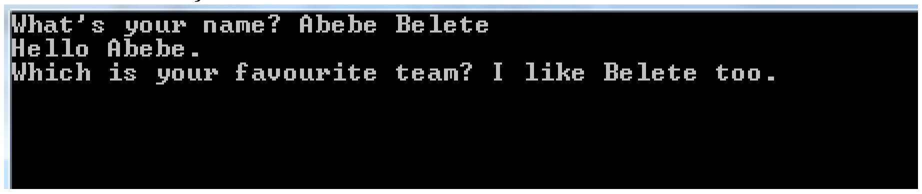
```
#include <iostream.h>
#include<conio.h>
int main ()
{
char mybuffer [100];
cout << "What's your name? ";
cin>>mybuffer;
cout << "Hello " << mybuffer << ".\n";
cout << "Which is your favourite team? ";
cin>>mybuffer;
cout << "I like " << mybuffer << " too.\n";
getch();
return 0;
}
```



```
What's your name? Abebe
Hello Abebe.
Which is your favourite team? Buna
I like Buna too.
```

- this would work, but this method has the following limitations that cin.getline has not:
  - It can only receive single words (no complete sentences) since this method uses as delimiter any occurrence of a blank character, including spaces, tabulators, newlines and carriage returns.
  - It is not allowed to specify a size for the buffer. What makes your program unstable in case that the user input is longer than the array that will host it.
  - For these reasons it is recommendable that whenever you require strings of characters coming from cin you use cin.getline instead of **cin** >> .

```
#include <iostream.h>
#include<conio.h>
int main ()
{
char mybuffer [100];
cout << "What's your name? ";
cin>>mybuffer;
cout << "Hello " << mybuffer << ".\n";
cout << "Which is your favourite team? ";
cin>>mybuffer;
cout << "I like " << mybuffer << " too.\n";
getch();
return 0;
}
```



```
What's your name? Abebe Belete
Hello Abebe.
Which is your favourite team? I like Belete too.
```

3.3 String Manipulation Functions

- The C++ and its parent the C languages do not have a string data type. In C and C++, strings are created from arrays. Therefore, the C++ language relies on operations performed on the arrays of characters or pointers to char. The functions used for this purpose are numerous and you should know what they are used for.
- As a reminder, thanks to the features of arrays of characters and the friendship between arrays of characters and strings, there are two main ways you can declare and initialize a string:

  char *Thing = “Telephone”;
  char Major[] = “Computer Sciences”;

1. The Length of a String (strlen)

- In many operations, you will want to know how many characters a string consists of. To find the number of characters of a string, use the strlen() function.
- Its syntax is: strlen(Major); where Major is the string variable
- The strlen ( ) function takes one argument, which is the string you are considering. The function returns the number of characters of the string.

```
char School[] = "Manchester United";
int Length = strlen(School);
cout << "The length of " << School<<”is”<<Length;
```

The length of Manchester United is 17 characters

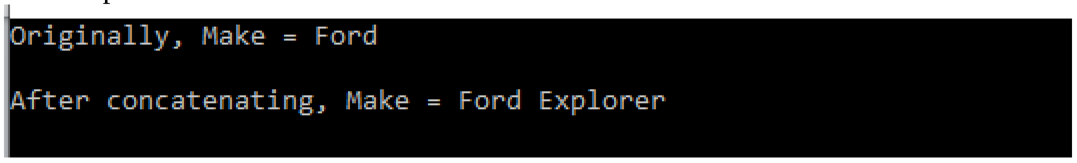
2. The strcat() Function

- If you have two strings, to append one to another, use the strcat () function.
- Its syntax is: char strcat(string1, string2); where string1 and string2 are the string variables

- The strcat ( ) function takes two arguments. The second argument, called the source string, is the string you want to add to the first string; this first string is referred to as the destination.
- Although the function takes two arguments. It really ends up changing the destination string by appending the second string at the end of the first string. This could be used to add two strings.
- Here is an example:

```
char Make[] = {"Ford "};
char Model[] = "Explorer";
cout << "Originally, Make = " << Make;
strcat(Make, Model);
cout << "\n\nAfter concatenating, Make = " << Make << endl;
```

- The output is



```
Originally, Make = Ford

After concatenating, Make = Ford Explorer
```

3. The strncat () Function

- Like the strcat( ) function, the strncat( ) function is used to append one string to another. The
  difference is that, while the strcat ( ) considers all characters of the source string, the strncat( ) function allows you to specify the number of characters from the source string that you want to append to the destination string. This means that, if the source string has 12 characters, you can decide to append only a set number of its characters.
- The syntax is:
  char strncat(string1, string2, number);
  Besides the same arguments as the strcat () function, the Number argument sets the number of characters considered from Source. To perform the concatenation, the compiler would count characters from left to right on the source string.
- Here is an example:

```
char Make[] = {"Ford "};
char Model[] = {"Explorer"};
cout << "Originally, Make = " << Make;
strcat(Make, Model,3);
cout << "\n\nAfter concatenating, Make = " << Make << endl;
```

Output

```
Originally, Make = Ford
After concatenating, Make = Ford Exp
```

4. Copying One String into Another- strcpy ( ) function

- The strcpy () function is used to copy one string into another string. In English, it is used to replace one string with another.
- The syntax of the strcpy () function is: **strcpy (string1, string2);**
- This function takes two arguments. The first argument is the string that you are trying to replace. The second argument is the new string that you want to replace. There are two main scenarios suitable for the strcpy ( ) function: To replace an existing string or to initialize a string.

```
char carName1[] = {"Ford Escort"};
char carName2[] = {"Toyota 4-Runner"};
cout<<strcpy(carName2, carName1);

Output
Ford Escort
```

5. The strncpy () Function

- The strncpy() function works like the strcpy() function. As a difference, the strncpy () function allows you to specify the number of characters that the compiler would copy from the source string.
- Here is the syntax: **char strncpy(string1, string2, number);**
  The Number argument specifies the number of characters that will be copied from the Source string.
- Here is an example:

```
char carName1[] = {"Ford Escort"};
char carName2[] = {"Toyota 4-Runner"};
cout<<strncpy(carName2, carName1,8);

Output is
Ford Esc-Runner
```

Comparing Strings

- Various routines are available for strings comparison. The C-String library is equipped with functions that perform the comparisons by characters. Alternatively, some compilers like Borland C++ Builder ships with additional multiple functions to perform these comparisons on null-terminated strings.

6. The strcmp() Function

- The strcmp () function compares two strings and returns an integer as a result of its comparison.
- Its syntax is: strcmp (char S1, char S2);
- This function takes two strings, S1 and S2 and compares them. It returns
  A negative value if S1 is less than S2
  0 if S1 and S2 are equal
  A positive value if S1 is greater than S2

- For example
  ```
  char carName1[] = {"Forda Escort"};
  char carName2[] = {"Fordb 4-Runner"};
  cout<<strcmp(carName2, carName1);
  ```
  The output is
  1

7. The strncmp () Function

- The strncmp () function compares two strings using a specified number of characters and returns an integer as a result of its findings. Its syntax is:
  strncmp (const char S1, char S2, int Number);
- This function takes three arguments. The first two arguments are the strings that need to be compared. The 3rd argument specifies the number of characters considered for the comparison.
- It returns
  A negative value if S1 is less than S2
  0 if S1 and S2 are equal
  A positive value if S1 is greater than S2
- For example
  ```
  char carName1[] = {"Forda Escort"};
  char carName2[] = {"Fordb 4-Runner"};
  cout<<strncmp(carName2, carName1,1);
  ```
  The output is
  0

8. The stricmp() Function

- The stricmp () function compares two strings without regard to their case. In other words, this function does not take into consideration if there is a mix of uppercase and lowercase characters in the strings.
- The syntax of the function is: stricmp(char S1, const char S2);
- This function takes two strings, S1 and S2 and compares them. It returns
  A negative value if S1 is less than S2
  0 if S1 and S2 are equal
  A positive value if S1 is greater than S2
- For example
  ```
  char carName1[] = {"Forda Escort"};
  char carName2[] = {"fordb 4-Runner"};
  cout<<strncmp(carName2, carName1,1);
  ```
  The output is
  ```
  32
  ```
- For example
  ```
  char carName1[] = {"Forda Escort"};
  char carName2[] = {"fordb 4-Runner"};
  cout<<stricmp(carName2, carName1);
  ```

The output is

```
1
```

9. The strnicmp() Function

- The strnicmp () function compares two strings without regard to their case but considers only a specified number of characters.
- The syntax of the function is: **int strnicmp (char S1, char S2, int n);**
- This function takes two strings, S1 and S2 and compares them. It returns
  A negative value if S1 is less than S2
  0 if S1 and S2 are equal
  A positive value if S1 is greater than S2
- For example

```
char carName1[] = {"Forda Escort"};
char carName2[] = {"fordb 4-Runner"};
cout<<strnicmp(carName2, carName1,1);
```

- The output is

```
0
```

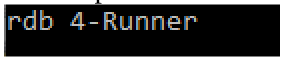
Working with Individual Characters

10. The strchr() Function

- The strchr () function looks for the first occurrence of a certain character in a string. Its syntax is:
  strchr(const char S, char c);
- This function takes two arguments. The second argument specifies what character to look for in the first argument which is a string. If the character c appears in the string S, the function would return a new string whose value starts at the first occurrence of c in S. If the character **c** does not appear in the string S, then the function would return NULL.
- For example

```
char carName2[] = {"fordb 4-Runner"};
char r='r';
cout<<strchr(carName2, r);
```

- The output is



```
rdb 4-Runner
```

Working with Sub-Strings

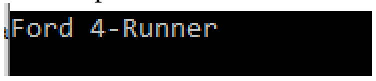
11.The strstr() Function

- The strstr () function looks for the first occurrence of a sub-string in another string and returns a new string as the remaining string. Its syntax is:
  char strstr(char Main, char Sub);

- The first argument of the function is the main string that would be examined. The function
  would look for the second argument, the Sub string appearance in the main string. If the
- Sub string is part of the Main string, then the function would return a string whose value starts at the first appearance of Sub and make it a new string. If Sub is not part of the Main string, the function would return a NULL value.
- For example

```
char carName1[] = {"Ford"};
char carName2[] = {"Ford 4-Runner"};
cout<<strstr(carName2, carName1);
```

- The output is



```
Ford 4-Runner
```

```
char carName1[] = {"EFord"};
char carName2[] = {"Ford 4-Runner"};
cout<<strstr(carName2, carName1);
```

- The output is null value as shown below

Working with Character Cases

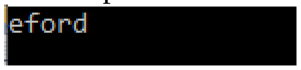
12.The strlwr() Function

- The strlwr() function is used to convert a string to lowercase.
- Its syntax is: **char strlwr(char S);**

- This function takes, as argument, the string that needs to be converted. During conversion, if a Latin character were in uppercase, it would be converted to lowercase. Otherwise, it would stay "as if". This means any symbol that is not a readable character would not be converted.
- For example

```
char carName1[] = {"EFord"};
cout<<strlwr(carName1);
```

- The output is



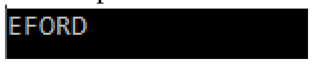
```
eford
```

13.The strupr( ) Function

- The strupr() function is used to convert a string to uppercase. Its syntax is:
  char strupr(char S);
  Each lowercase character in the function's argument, S, would be converted to uppercase. Any character or symbol that is not in lowercase would not be changed.
- For example

```
char carName1[] = {"EFord"};
cout<<strupr(carName1);
```

- The output is



```
EFORD
```