CS1002 – Programming Fundamentals

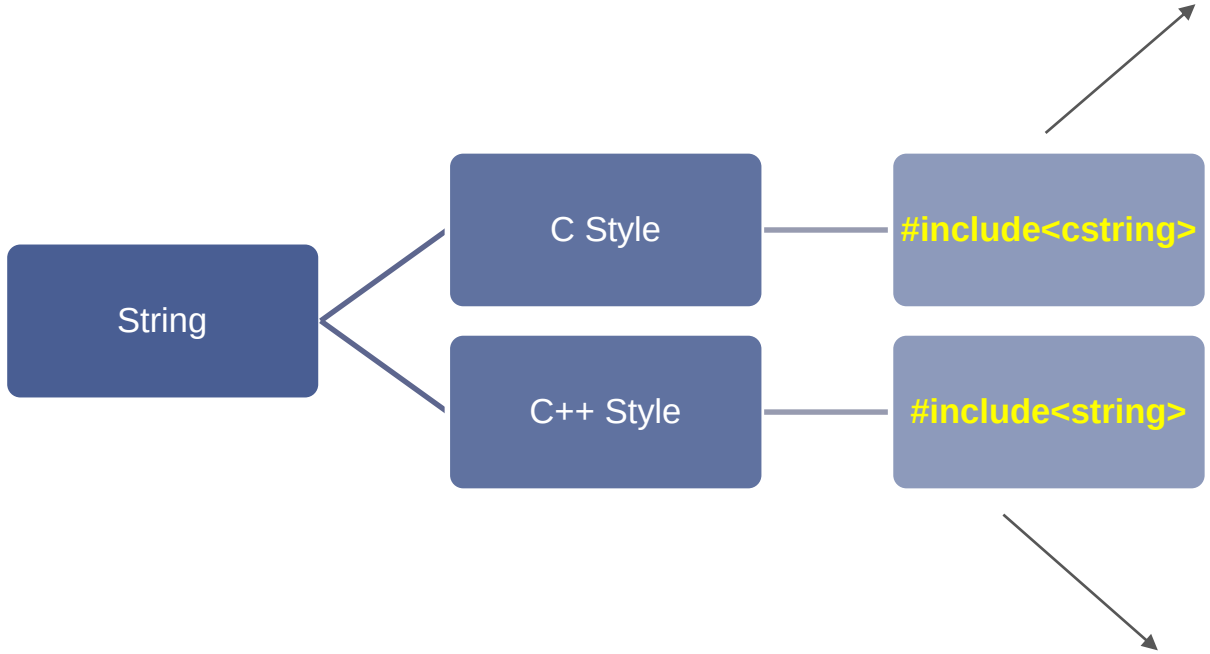
Lecture # 21
Monday, November 14, 2022
FALL 2022
FAST – NUCES, Faisalabad Campus

Rizwan Ul Haq

Outline

- C-Strings (C Language – Old Way)
- Strings (C++ Language – New Way)

C-String vs Strings




```cpp
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char s1[] = "ABCDE";
    char s2[5] = {'a', 'b','c','d','e'};
    cout<<s2<<endl;
    cout<<s1<<endl;
    cout<< strlen(s1) <<endl;
    cout<< strcmp(s1,s2) <<endl;

    system("pause");
}
```

```
C:\Users\muhammad.yousaf\documents\v
abcdeÍÍÍÍÍÍÍÍÍÍABCDE
ABCDE
5
-1
Press any key to continue . . .
```

```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string str1 = "ABCDEFG";
    string str2 = "abcdefg";
    cout<< str1.length() <<endl;
    cout<< str1.compare(str2) <<endl;

    system("pause");
}
```

```
C:\Users\muhammad.yousaf\documents\
7
-1
Press any key to continue . . .
```

Note: Check the library of both the programs!

C-Strings (**Character Arrays**)

- **Character array:** An array whose components are of type char
- C-strings are null-terminated ('\0‘) character arrays
- Example:
  - 'A' is the character A
  - "A" is the C-string A
  - "A" represents two characters, 'A' and '\0‘

C-Strings **(Character Arrays) (cont'd.)**

- Consider the statement

  char name[16];

- Since C-strings are null terminated and **name** has 16 components, the largest string that it can store has 15 characters
- If you store a string of length, say 10 in **name**
  - The first 11 components of **name** are used and the last five are left unused

C-Strings (Character Arrays) (cont'd.)

- The statement

```
char name[16] = {‘J’, ‘o’, ‘h’, ‘n’, ‘\0’} ;
```

- Equivalent

```
char name[16] = “John” ;
```

declares an array **name** of length 16 and stores the C-string **"John"** in it

- The statement

```
char name[] = “John” ;
```

declares an array **name** of length 5 and stores the C-string "John" in it

C-Strings (Character Arrays) (cont'd.)

- Most rules that apply to other arrays also apply to character arrays. Consider the following statement:

```
char studentName[26] ;
```

- Suppose you want to store "Lisa L.Johnson" in **studentName**
- Because aggregate operations, such as assignment and comparison, are not allowed on arrays, the following statement is not legal:

```
studentName = "Lisa L. Johnson"; //illegal
```

C-Strings (Character Arrays) (cont'd.)

TABLE 9-1 strcpy, strcmp and strlen Functions	
Function	**Effect**
strcpy(s1, s2)	Copies the string s2 into the string variable s1 The length of s1 should be at least as large as s2
strcmp(s1, s2)	Returns a value < 0 if s1 is less than s2 Returns 0 if s1 and s2 are the same Returns a value > 0 if s1 is greater than s2
strlen(s)	Returns the length of the string s; excluding the null character

To use these functions, the program must include the header file cstring via the `include` statement. That is, the following statement must be included in the program.

```
#include <cstring>
```

String Comparison

- C-strings are compared character by character using the collating sequence of the system
- If we are using the ASCII character set
  - "Air" < "Boat"
  - "Air" < "An"
  - "Bill" < "Billy"
  - "Hello" < "hello"

String Comparison (cont'd.)

Suppose you have the following statements:

```
char studentName[21];
char myname[16];
char yourname[16];
```

The following statements show how string functions work:

Statement	Effect
`strcpy(myname, "John Robinson");`	`myname = "John Robinson"`
`strlen("John Robinson");`	Returns 13, the length of the string `"John Robinson"`
`int len;` `len = strlen("Sunny Day");`	Stores 9 into `len`
`strcpy(yourname, "Lisa Miller");` `strcpy(studentName, yourname);`	`yourname = "Lisa Miller"` `studentName = "Lisa Miller"`
`strcmp("Bill", "Lisa");`	Returns a value < 0

`strcpy(yourname, "Kathy Brown");`
`strcpy(myname, "Mark G. Clark");`
`strcmp(myname, yourname);`
	`yourname = "Kathy Brown"`
	`myname = "Mark G. Clark"`
	Returns a value > 0

Reading and Writing Strings

- Most rules that apply to arrays apply to C-strings as well
- Aggregate operations, such as assignment and comparison, are not allowed on arrays
- Even the input/output of arrays is done component-wise
- The one place where **C++ allows aggregate operations** on arrays is the **input and output of C-strings** (that is, character arrays)

String Input

- **cin >> name;** stores the next input C-string into **name**
  - char name[31];
  - The length of the input C-string must be less than or equal to 30
  - stores the 30 characters that are input and the null character **'\0'**

String Input

- Recall that the extraction operator, >>, skips all leading whitespace characters and stops reading data into the current variable
  - As soon as it finds the first whitespace character or invalid data
- As a result, C-strings that contain blanks cannot be read using the extraction operator, >>
- For example, if a first name and last name are separated by blanks, they cannot be read into name

String Input

- To read strings with blanks, use get:

  cin.get(str, m+1);

- This statement stores the next **m** characters, or all characters until the newline character **'\n'** is found, into str.
- Stores the next **m** characters into **str** but the newline character is not stored in **str**
- If the input string has fewer than **m** characters, the reading stops at the newline character

Exam

Now, suppose that we have the statements:

```
char str1[26];
char str2[26];
char discard;
```

and the two lines of input:

```
Summer is warm.
Winter will be cold.
```

Further, suppose that we want to store the first C-string in **str1** and the second C-string in **str2**. Both **str1** and **str2** can store C-strings that are up to **25** characters in length. Because the number of characters in the first line is **15**, the reading stops at **'\n'**. You must read and discard the newline character at the end of the first line to store the second line into **str2**. The following sequence of statements stores the first line into **str1** and the second line into **str2**:

```
cin.get(str1, 26);
cin.get(discard);
cin.get(str2, 26);
```

To read and store a line of input, including whitespace characters, you can also use the stream function **getline**. Suppose that you have the following declaration:

```
char textLine[100];
```

The following statement will read and store the next 99 characters, or until the newline character, into **textLine**. The null character will be automatically appended as the last character of **textLine**.

```
cin.getline(textLine, 100);
```

String Output

- Aggregate operations are allowed on string output as well
- `cout << name;` outputs the content of name on the screen
  - << continues to write the contents of name until it finds the null character
  - If **name** does not contain the null character, then we will see strange output
    - << continues to output data from memory adjacent to name until '\0' is found

Questions