Lecture #9 c_string

String vs c_string & <cctype> Function

#include <cctype> **-** Seven C++ standard library functions

The return type for all of these functions is int. **If true, a non-zero value is returned**.
char character;

`isalpha(character)` // ('A' – 'Z', 'a' – 'z')

`isalnum(character)` // ('A' – 'Z', 'a' – 'z', '0' – '9')

`isdigit(character)` // ('0' – '9')

`islower(character)` // ('a' – 'z')

`isupper(character)` // ('A' – 'Z')

The following two functions convert lowercase to upper and vice-versa.

`toupper(character)` // (lowercase is converted to uppercase)

`tolower(character)` // (uppercase is converted to lowercase)

Ex 1
```
#include <iostream>
#include <cctype>   // (char contents type)

using namespace std;

int main()
{
    char character;

    cout << "Enter a character";
    cin >> character;
```

```
    if (isalpha(character))      // Function returns non-zero if true – 0 if false
      cout << "The character is an alphabetic character.  \n\n";

    else
      cout << "The character is not an alphabetic character.\n\n";

    return 0;
}
```

Ex_9-1 - Working with <cctype> functions

Character Array (Also called a **cString**)

- A sequence of characters stored in an array.
- A c_string is a null-terminated character array.
  - The computer places a null character at the end of the string of characters.

Ex 2: **Read character data into a cString.**

```
#include <iostream>
#include <string>
using namespace std;

void displayName(char name[]);

const int SIZE = 30;      // global variable is visible in main() and displayArray()

int main()
{
     char name[SIZE];

     cout << "Enter your name:  ";

     // The cin.getline() function reads characters and stops at the '\n' character.
     // In this case, 29 characters are read in using the  cin.getline() function.
     // The last element is automatically assigned the NULL character.

     cin.getline(name, SIZE);

     // Pass an array to a function.

     displayName(name);          // Pass the address of name[0] to
                                 //    the displayName function
     return 0;
```

```
}     // End main
// ================================================================
```

```
// ==== displayName function ======================================
void displayName(char name[ ])
{
     cout << endl << endl;
     cout << "Your name is:  " << name << endl;
}
// ================================================================
```

string vs. **c_string**

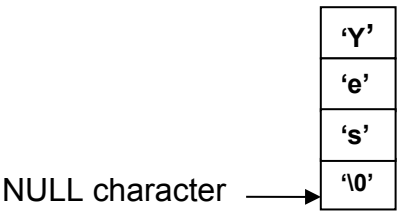
string fullName; ← **string type** requires: **#include<string>**
getline(cin, fullName); ← **getline( )** requires: **#include<string>**
* Does not require declaring the size of fullName.

char fullName[SIZE];
cin.getline(fullName, SIZE); ← **cin.getline( ) Does <u>not</u> require #include<string>**
* The array size must be specified before using.

c_string – A character array always has a terminating **NULL** character.

NULL is the `'\0'` character and is allocated automatically.

- c_strings always have a **terminating NULL character**.
- In Programming, NULL means zero.



Any of the following 3 ways can be used to assign the null character to a variable.

<u>Ex_3</u>: `char character;`

```
character = NULL;    // NULL is a predefined constant
character = '\0';    // '\0' is ASCII Code character for the null character
character = 0;       // 0 is ASCII Code numerical value for null character
```

- **To initialize a c_string to an empty string:**

  - `char fullName[30] = {'\0'};` // Initializes first element with a null character.

`char fullName[30] = {NULL};`

String literal - A sequence of characters that are typed directly into a program are called string literals.

- String literals are null-terminated c_strings.

  Ex_4: `cout << "Hello";`

- When the compiler encounters a string literal, it places it in an array in the data section of the program.
- In the case above, the compiler places "Hello" in an array of size 6, (one extra space for the null character).

H	e	l	l	o	'\0'

- **To put values into a c_string** – Three ways:
  1. To initialize a character array: `char fullName[20] = "Bob";`

     `fullName = "Tom";` ← no

     `strcpy_s(fullName, "Tom");`

     - The array is assigned a string of characters (within double quotes)
     - No braces.
     - The 4th element ([3]) is automatically assigned the NULL character.
  2. To assign a value, use a standard string function – **strcpy** // #include <string>
     - `strcpy_s(fullName, "Tom Lee");`

     Note: This does not work → `fullName = "Tom Lee";`
  3. To read input, use **cin.getline** function
     - `cin.getline(fullName, 30);` // Reads up to 29 characters.
     - **cin.getline function** – Reads an entire line of text up to a newline (`'\n'`), or to

the length of the character array, whichever comes first.

- **cin.getline function** – Replaces the newline (`'\n'`) character with a NULL

c_string functions

1. `strcpy_s(str1, str2)` **-** Copies a literal string from one to another.

(str1 is the destination and str2 is the source)

Note: Some compilers prefer: `strcpy(str1, str2).`

2. `strcat_s(str1, str2)` **-** Concatenates cstrings (combines the strings).

Note: Some compilers prefer: `strcat(str1, str2).`

3. `strlen(str1)` **-** Counts the number of array characters (excluding the NULL character).

4. `strcmp(str1, str2)` **-** Compares cstrings to see if they are equal. Returns 0 if equal.
(Case sensitive - 'A' != 'a')

5. `_strcmpi(str1, str2)` **-** Compares cstrings, but not case sensitive.

Using c_string functions:

To compare alphabetPart1 to alphabetPart2, use strcmp(alphabetPart1, alphabetPart2).

- The result is: -1. - A non-zero value means the strings are not equal.

```
char firstHalf[14] = "abcdefghijklm";
char secondHalf[14] = "nopqrstuvwxyz";
char alphabet[27];
strcpy(alphabet, firstHalf);
strcat(alphabet, secondHalf);

cout << alphabet;         // abcdefghijklmnopqrstuvwxyz
```

The result of strlen(firstHalf) is: 13

The result of strlen(alphabet) is: 26

To output a c_string or string - use **`cout <<`**

```
#include <iostream>
#include <string>              // Required in order to use string data type and getline()
using namespace std;

int main()
{
  string message;                  //   Declare a variable of string data type

  cout << "Enter a message:  ";
  getline(cin, message);           //   cin.getline() does not work with string type.

  cout << "The message you entered is:  " << message << endl;
```

string Operations

Ex 9:
```
int main( )
{
   string  str1;
   string  str2  =  "Day";
   string message;

   str1  =  "Sunny";     //  Assignment operator ( = ) works with string
                         //       but not with c_string.
```

Ex 10:
```
message = str1 + " " + str2;  //   Concatenation operator ( + )

cout << message;              //   Output:  Sunny Day
```

NOTE: The + operator works only if at least one operand is a string variable.

Ex 11:
```
str1  =  str1  +  "Day";          //  OK
```

Ex 12:
```
str1  =  "Sunny"  +  " " + "Day";      //  Wrong
```

string Functions (string class member functions)

`length` Function - Returns the number of characters in a string.

Ex 13:
```
string firstName  =  "Tom";
string fullName  =  firstName  +  " " + "Smith";
```

```
cout << firstName << " has " << firstName.length( )
     << "characters\n";          // OUTPUT→ Tom has 3 characters
cout << fullName  << " has " << fullName.length( )
     << "characters\n";          // OUTPUT → Tom Smith has 9 characters
```

`substr` Function - Returns a string of characters (a substring) in a string.
- Two parameters: substr(index, number of characters)

Ex:
```
string message = "To be or not to be, this is the question.";
cout << message.substr(3, 15);    //output:  not to be
```

`find` Function - Searches for a substring within a string and returns the index number where the substring begins.

Ex:
```
string message = "To be or not to be, this is the question.";
cout << message.find("not to be")     //output:  9
```

Ex:
```
int main()
{
     string message = "To be or not to be, this is the question.";

     cout << "Message length equals " << message.length() << ".\n\n";

     cout << "Substring at index 3:  " << message.substr(3, 15)
          << ".\n\n";

     cout << "The substring, \"not to be\" begins at index["
          << message.find("not to be") << "].\n\n";

/* OUTPUT
Message length equals 41.

Substring at index 3:  be or not to be.

The substring, "not to be" begins at index [9].  */
```

SUMMARY - string vs. c_string

c_string - c_string is just a name for a null-terminated character array.
- c_string is not a reserved word. (You will get a compile error if you use it in code)

Ex_15: `char name[30];` // Correct way to declare a c_string

Ex_16: `c_string name[30];` // No - Error (no such thing as c_string)

string - string is a data type, and is a reserved word.

Ex_17: `string name;` // Correct way to declare a variable of string type

	string	**c_string**
Initialize a variable	string name = “Tom Lee”;	char name[30] = “Tom Lee”;

Note: The c_string initialization puts “Tom Lee” in the array, followed by the Null character. Therefore, therefore, there are 8 characters in the array.

Note: The assignment operator (=) only works with c_strings in an initialization statement.

Assignment statement	name = “Bob Jones”;	strcpy_s(name, “Bob Jones”);
Concatenate strings	name = “Bob” + “ ” + “Jones”;	strcpy(name, “Bob”); strcat(name, “ ”); strcat(name, “Jones ”):

Note: Both strcpy() and strcat() require #include<string>

Read from keyboard	getline(cin, name);	cin.getline(name, 8);
Read from a file	getline(inFile, name);	inFile.getline(name, 8);