# String Functions

## 1-strcpy()

The strcpy() function in C++ is used to copy the contents of one string (source) into another string (destination). It is a part of the C-style string handling functions in the <cstring> header.

**Example**

```
#include <iostream>
#include <cstring> // For strcpy()
void demonstrateStrcpy() {
    char source[] = "Hello, World!"; // Source string
    char destination[50];          // Ensure destination is large enough to hold the source string
    // Copy source to destination
    strcpy(destination, source);
    // Output the strings
cout << "Source: " << source << endl;
    cout << "Destination: " << destination << endl;
}
```

```
int main() {
    demonstrateStrcpy();
    return 0;
}
```

**Explanation:**

1. **Header File:** Include <cstring> to access strcpy().
2. **Parameters:**
   - destination: A character array where the string will be copied.
   - source: The character array or string to copy from.
3. **Usage:**
   - Make sure destination is large enough to hold source and the null-terminator (\0).
4. **Null-Termination:**
   - strcpy() automatically appends the null-terminator to the destination string.

**Key Notes:**

- **Buffer Size:** If the destination array is smaller than the source string, it will lead to **buffer overflow**, causing undefined behavior.

Use strcpy() with caution, especially when dealing with dynamic or unpredictable string sizes.

## 2-strcmp()

The strcmp() function in C++ is a standard library function used to compare two null-terminated C-style strings (const char*). It is defined in the <cstring> header. This function performs a lexicographical comparison(مقارنة معجمية) and returns:

- 0 if both strings are equal.
- A value less than 0 if the first string is lexicographically less than the second.
- A value greater than 0 if the first string is lexicographically greater than the second.

**Example:**

```
#include <iostream>
#include <cstring>

void compareStrings(const char* str1, const char*
str2) {
    int result = strcmp(str1, str2);

    if (result == 0) {
        cout << "The strings \"" << str1 << "\" and \""
<< str2 << "\" are equal.\n";
    } else if (result < 0) {
        cout << "The string \"" << str1 << "\" is
lexicographically less than \"" << str2 << "\".\n";
    } else {
        cout << "The string \"" << str1 << "\" is
lexicographically greater than \"" << str2 << "\".\n";
    }
}

int main() {
    const char* string1 = "Apple";
    const char* string2 = "Banana";

    // Call the function to compare strings
    compareStrings(string1, string2);
```

```
    const char* string3 = "Orange";
    const char* string4 = "Orange";

    // Call the function to compare another set of
strings
    compareStrings(string3, string4);

    return 0;
}
```

**Output:**

**The string "Apple" is lexicographically less than "Banana".**
**The strings "Orange" and "Orange" are equal.**

**Notes:**

- Always ensure the strings passed to strcmp() are null-terminated.

## 3-strlen()

The strlen() function is used to calculate the length of a null-terminated C-style string (not including the null character \0).

**Example:**

```
#include <iostream>
#include <cstring> // Required for strlen
```

```
void demonstrateStrlen() {

    const char* str = "Hello, World!";

    // Use strlen to get the length of the string
int length = strlen(str);

    // Print the result
    cout << "The length of the string \"" << str << "\"
is: " << length << endl;
}

int main() {
    demonstrateStrlen();
    return 0;
}
```

**Explanation:**

1. **Include <cstring>**: The strlen() function is part of the C standard library, so you need to include <cstring>.
2. **Define a C-style string**: This is typically a const char* or char array that is null-terminated.
3. **Call strlen()**: Pass the string to strlen(). It returns the length of the string as a integer value.
4. **Print the result**: Display the computed length.

**Notes:**

- The strlen() function does **not** count the null-terminator \0.
- Ensure the string passed to strlen() is null-terminated to avoid undefined behavior.

For example:

```
#include <iostream>
#include <string>
void demonstrateStringLength() {
    string str = "Hello, World!";
    cout << "The length of the string \"" << str << "\"
is: " << str.length() << endl;
}
```

## 4- strcat()

The strcat() function in C++ is used to concatenate (append) one C-style string (null-terminated character array) to another. It is part of the <cstring> library.

**Example** :

```
#include <iostream>

#include <cstring> // For strcat

// Function to demonstrate strcat usage
```

```
void concatenateStrings(const char* str1, const char* str2) {
    char result[100]; // Ensure this is large enough to hold both strings and the null-terminator
    // Initialize result with str1
    strcpy(result, str1);
    // Concatenate str2 to result
    strcat(result, str2);
cout << "Concatenated String: " << result << endl;
}
int main() {
    const char* string1 = "Computer, ";
    const char* string2 = "Science!";
    concatenateStrings(string1, string2);
    return 0;
}
```

**Explanation**

1. **#include <cstring>**: Include the header file that contains strcat() and related functions.
2. **strcat(destination, source)**:

    - Appends the contents of source to destination.
    - destination must have enough space to hold the concatenated string.
3. **Steps in the Code**:
    - Copy str1 into a buffer result using strcpy().
    - Use strcat() to append str2 to result.
    - Print the concatenated string.

## Key Points to Remember

- Ensure the destination array is large enough to hold both strings and the null-terminator (\0).
- strcat() does not perform bounds checking. Using a buffer that is too small can lead to undefined behavior (buffer overflow).
- Consider using safer alternatives like strncat() to avoid overflow, especially when working with user input.

## 5-isalnum()

The isalnum() function in C++ is part of the <cctype> library. It checks whether a given character is alphanumeric (a letter or a digit).

### Syntax

```
#include <cctype>
int isalnum(int c);
```

- **Parameter**: c is the character to check, passed as an int (often cast from char).
- **Return Value**: Returns a non-zero value (true) if c is an alphanumeric character; otherwise, it returns 0 (false).

## Example

```
#include <iostream>
#include <cctype> // Required for isalnum()

// Function to check if a character is alphanumeric
void checkAlnum(char c) {
    if (isalnum(c)) {
        cout << c << " is alphanumeric.\n";
    } else {
        cout << c << " is not alphanumeric.\n";
    }
}

int main() {
    char testChar1 = 'A';
    char testChar2 = '1';
    char testChar3 = '#';

    checkAlnum(testChar1); // Output: A is
alphanumeric.
```

```
    checkAlnum(testChar2); // Output: 1 is
alphanumeric.
    checkAlnum(testChar3); // Output: # is not
alphanumeric.

    return 0;
}
```

**Key Points**

1. **Alphanumeric Characters**: Letters (a-z, A-Z) and digits (0-9).
2. Non-alphanumeric characters such as punctuation, symbols, or whitespace return false.
3. You can combine isalnum() with other <cctype> functions like isalpha() and isdigit() for more checks.

## 6-isalpha()

In C++, the isalpha() function is part of the <cctype> header and is used to check if a given character is an alphabetic letter (either uppercase or lowercase). Here's an example function that demonstrates how to use isalpha():

**Example**

```
#include <iostream>
#include <cctype> // For isalpha()
```

```
// Function to check if a character is alphabetic
void checkIfAlphabetic(char ch) {
    if (isalpha(ch)) {
        cout << ch << " is an alphabetic character." <<
endl;
    } else {
        cout << ch << " is NOT an alphabetic
character." << endl;
    }
}
int main() {
    char testChar1 = 'A';
    char testChar2 = '1';
    char testChar3 = 'z';

    // Test the function
    checkIfAlphabetic(testChar1);
    checkIfAlphabetic(testChar2);
    checkIfAlphabetic(testChar3);

    return 0;
}
```

**Output:**

**A is an alphabetic character.**
**1 is NOT an alphabetic character.**
**z is an alphabetic character.**

## Explanation

1. **Include <cctype>**: This header contains the declaration for isalpha() and other character-handling functions.
2. **Call isalpha(char)**: The function takes a single char as an argument and returns:
    - true (non-zero) if the character is alphabetic (A-Z or a-z).
    - false (0) otherwise.

## Key Notes

- isalpha() only works with individual characters.
- To check strings, you can iterate through each character and use isalpha() in a loop.

## 7- islower() and isupper

islower() and isupper() in C++ are used to check whether a character is lowercase or uppercase, respectively.

**Example:**

```
#include <iostream>
#include <cctype> // For islower() and isupper()

void checkCharacter(char ch) {
    if (islower(ch)) {
    cout << ch << " is a lowercase letter.\n";
    } else if (isupper(ch)) {
        cout << ch << " is an uppercase letter.\n";
    } else {
        cout << ch << " is not a letter.\n";
    }
}

int main() {
    char character;
    cout << "Enter a character: ";
    cin >> character;

    checkCharacter(character);

    return 0;}
```

**Explanation:**

1. **Header <cctype>**:
   - The islower() function checks if a character is a lowercase letter (a to z).
   - The isupper() function checks if a character is an uppercase letter (A to Z).
2. **Function checkCharacter**:
   - Takes a character as input.
   - Uses islower() and isupper() to determine the character type.
   - Prints the result.

## Input/Output:

*Example 1:*

Input:

Enter a character: a

Output:

a is a lowercase letter.

*Example 2:*

Input:

Enter a character: Z

Output:

Z is an uppercase letter.

*Example 3:*

Input:

Enter a character: 1

Output:

1 is not a letter.

## 9- struper() & strlwr()

In C++, the strupr() and strlwr() functions are typically not part of the C++ standard library, but they are available in some compilers (like in the Turbo C++ library). These functions convert a string to uppercase (strupr()) or lowercase (strlwr()).

If you're using a compiler that doesn't have these functions by default, you can implement similar functionality using the standard C++ library.

Example

```
#include <iostream>
#include <cstring>

int main() {
    char str[] = "Hello World!";
```

```
// Convert to uppercase using strupr()
strupr(str);
 cout << "Uppercase: " << str << endl;

// Convert to lowercase using strlwr()
strlwr(str);
 cout << "Lowercase: " << str << endl;

return 0;}
```

## 10-strchr()

The strchr() function in C++ is part of the C Standard Library, defined in the <cstring> header, and is used to find the first occurrence of a character in a C-style string (null-terminated character array).

Example

**Syntax:**

**char* strchr(const char* str, int character);**

- str: The C-string in which to search.
- character: The character to search for. This is passed as an int, but it is typically a char value.

**Example:**

```
#include <iostream>
#include <cstring>

int main() {
    const char* str = "Hello, world!";
    char ch = 'o';
     char* nullptr =Null;
    // Using strchr to find the first occurrence of 'o'
    char* result = strchr(str, ch);

    if (result != nullptr) {
        cout << "Character '" << ch << "' found at
position: "
                  << (result - str) << endl;
    } else {
        cout << "Character '" << ch << "' not found!" <<
endl;
    }    return 0;
}
```

**Explanation:**

- strchr(str, ch) searches for the first occurrence of the character ch in the string str.
- If the character is found, strchr() returns a pointer to the first occurrence of the character.
- If the character is not found, strchr() returns null.

- In the example, we use result - str to calculate the position of the character in the string (i.e., the index).

**Output:**

**Character 'o' found at position: 4**

**Note:**

When you use strchr() to find a character in a string, it returns a pointer to the first occurrence of the character. To determine the position (index) of the character within the string, we subtract the base pointer of the string (str) from the pointer returned by strchr() (result).

**Explanation of Pointer Arithmetic:**

- str is a pointer to the first character of the string.
- result is a pointer to the first occurrence of the character you are searching for in the string.

In C++, when you subtract two pointers that point to elements of the same array (or string in this case), the result is the difference in terms of the number of elements between those pointers. This difference corresponds to the index of the element in the array or string.

## Let's break down the subtraction result - str:

1. str points to the start of the string. If str = "Hello, world!", then str points to the character 'H'.
2. result is the pointer returned by strchr(), which points to the character where the character 'o' is found. In the string "Hello, world!", the first 'o' is at position index 4.

   So, result points to the 'o' at index 4.

3. Now, when you subtract result - str, you are calculating how far the result pointer is from the str pointer. This distance is the index of the character in the string.

   In this case:

   - result points to the 5th character in the string (index 4).
   - str points to the first character (index 0).
   - So, result - str equals 4, which is the index of the character 'o' in the string.

## 11-strstr()

In C++, the function strstr() is used to find the first occurrence of a substring in a string. If the substring is found, strstr() returns a pointer to the beginning of the found substring; otherwise, it returns nullptr.

### Example

```
#include <iostream>
#include <cstring>  // For strstr()

int main() {
    const char* str = "Hello, welcome to the C++ world!";
    const char* substring = "welcome";

    // Use strstr to find the substring
    const char* result = strstr(str, substring);

    if (result != nullptr) {
        cout << "Substring found at position: " << (result - str) << endl;
    } else {
        cout << "Substring not found!" << endl;
    }

    return 0;
}
```

**Explanation:**

1. strstr(str, substring):
    - str: The main string in which you want to search.
    - substring: The substring you're looking for.
    - It returns a pointer to the first occurrence of the substring, or nullptr if the substring is not found.
2. result - str: This calculates the position of the substring within the main string by subtracting the base address of str from the pointer result.

**Output:**

Substring found at position: 7

This program checks if the substring "welcome" is found in the main string "Hello, welcome to the C++ world!" and outputs the position where the substring starts. If it's not found, it will output "Substring not found!".