Character array vs Strings

Character array in C++

In C++, a **character array** is a sequence of characters stored in contiguous memory locations. It is typically used to represent C-style strings, which are null-terminated character sequences (`'\0'` at the end). Character arrays are a fundamental concept in C and C++ for handling strings before the introduction of the `std::string` class in the Standard Library.

Declaring and Initializing a Character Array

char str[10]; // Uninitialized array of size 10

char str[10] = "Hello"; // Array size is 10, but only 5 characters used, rest are '\0'

char str[] = "Hello"; // Size automatically set to 6 (5 characters + null terminator)

char str[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; // Null terminator must be included manually

Example of Character Array Initialization

```
#include <iostream>

int main()
{
    char greeting1[6] = "Hello";  // Size includes space for null character
```

```
    char greeting2[] = "World";   // Size automatically inferred as 6

    std::cout << greeting1 << " " << greeting2 << std::endl;   }
```

Accessing Elements in a Character Array

```
#include <iostream>

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

    // Accessing elements by index
    std::cout << "First character: " << str[0] << std::endl;
    std::cout << "Last character: " << str[4] << std::endl;

    // Modifying a character
    str[0] = 'h';
    std::cout << "Modified string: " << str << std::endl;

    return 0;
}
```

Input and Output with Character Arrays

1. Output:

You can print a character array using `std::cout`.

```
#include <iostream>

int main() {
    char str[] = "Hello, World!";
    std::cout << str << std::endl;  // Outputs the entire string
    return 0;
}
```

2. Input:

You can use `std::cin` or `std::cin.getline()` to take input into a character array.

- **Using `std::cin`**: This stops reading input when a whitespace (space, tab, or newline) is encountered.

```
#include <iostream>

int main() {

    char str[50];

    std::cout << "Enter a string: ";

    std::cin >> str;  // Stops at first whitespace

    std::cout << "You entered: " << str << std::endl;

    return 0;

}
```

Using `std::cin.getline()`: This reads an entire line of text, including spaces, up to a specified size or until a newline (`'\n'`) is encountered.

```
#include <iostream>

int main()

 {

    char str[50];

    std::cout << "Enter a string: ";

   std::cin.getline(str, 50);  // Reads up to 49 characters, leaves space for '\0'

    std::cout << "You entered: " << str << std::endl;  }
```

In C++, character arrays are used to represent C-style strings (null-terminated character sequences). To work with character arrays, several functions from the C standard library (`<cstring>` or `<string.h>`) are available. These functions help manipulate and manage character arrays effectively.

Common C++ Character Array Functions (C-Style)

1.strlen(const char* str)

Returns the length of the string (excluding the null terminator).

#include <cstring>

const char* str = "Hello";

size_t len = strlen(str); // len = 5

2.strcpy(char* dest, const char* src)

Copies the string `src` to `dest` (including the null terminator).

char dest[20];

const char* src = "Hello";

strcpy(dest, src); // dest = "Hello"

3.strcat(char* dest, const char* src)

Concatenates `src` to the end of `dest`. The `dest` array must be large enough to hold the result.

char dest[20] = "Hello, ";

```
const char* src = "World!";

strcat(dest, src);  // dest = "Hello, World!"
```

4.strcmp(const char* str1, const char* str2)

Compares `str1` and `str2` lexicographically.

- Returns 0 if both strings are equal.
- Returns a negative value if `str1` is less than `str2`.
- Returns a positive value if `str1` is greater than `str2`

```
int result = strcmp("abc", "abc");  // result = 0 (equal)
```

Summary of Character Arrays in C++

- A **character array** is used to store a sequence of characters, often representing a C-style string.
- The array must be large enough to hold the string and the null terminator ('\0').
- You can manipulate character arrays using functions from <cstring>, such as strcpy(), strlen(), strcat(), and more.
- For input, std::cin can be used for single words, while std::cin.getline() can read an entire line of input.
- Be mindful of array bounds to avoid buffer overflows and undefined behavior.

Character arrays are useful for low-level string manipulation, but for more flexibility and convenience, the C++ Standard Library's std::string class is often preferred.

Strings in C++

In C++, the **`std::string`** class, provided by the Standard Template Library (STL), is the preferred way to handle and manipulate strings. It offers a higher-level abstraction than character arrays and provides several built-in functions to perform operations like concatenation, comparison, searching, and more.

The `std::string` class is part of the `<string>` header and handles memory management, making it easier and safer to work with strings than using C-style character arrays.

1. **Creating a String**

```
#include <iostream>

#include <string>


int main() {

    std::string str1 = "Hello";

    std::string str2("World");

    std::string str3;  // Empty string

    std::cout << str1 << " " << str2 << std::endl;  // Output: Hello World    }
```

2. Reading and Writing Strings

- **Output**: You can use `std::cout` to print a string.
- **Input**: You can use `std::cin` or `std::getline()` to read strings.

```
#include <iostream>

#include <string>

int main() {

    std::string name;

    std::cout << "Enter your name: ";

    std::getline(std::cin, name);  // Reads entire line including spaces

    std::cout << "Hello, " << name << "!" << std::endl;

    return 0;

}
```

Note: `std::getline()` is used when you need to capture an entire line, including spaces. If you use `std::cin >>` for input, it will only capture up to the first space.

3. Accessing Characters

You can access individual characters in a `std::string` using array indexing (`[]`) or the `.at()` method:

```
#include <iostream>

#include <string>

int main()

{

    std::string str = "Hello";
```

```
    std::cout << "First character: " << str[0] << std::endl;   // Using []
    std::cout << "Last character: " << str.at(4) << std::endl; // Using .at()
    // Modifying a character
    str[0] = 'h';
    std::cout << "Modified string: " << str << std::endl;      // Output: hello
    return 0;
}
```

String functions in C++

C++ provides the **`std::string`** class as part of the Standard Library, which offers a wide range of functions to manipulate and work with strings. Here's a breakdown of the most commonly used string functions:

1. `size()` or `length()`

- Returns the number of characters in the string.
- Both `size()` and `length()` do the same thing

```
std::string str = "Hello";
std::cout << str.size() << std::endl;   // Output: 5
std::cout << str.length() << std::endl; // Output: 5
```

2. `empty()`

- Returns `true` if the string is empty (i.e., has a length of 0); otherwise, it returns `false`.

```
std::string str = "";
if (str.empty()) {
    std::cout << "String is empty" << std::endl;
}
```

3. `append()` or `+=`

- Appends another string or character to the end of the current string.

```
std::string str = "Hello";
str.append(" World!");  // str becomes "Hello World!"
// Or
str += " World!";       // Equivalent
```

4. `insert()`

- Inserts a string or a character at a specific position.

```
std::string str = "Hello!";
str.insert(5, ", World"); // Inserts ", World" at index 5
std::cout << str << std::endl;  // Output: Hello, World!
```

5. `erase()`

- Erases a portion of the string starting from a specific position. You can either erase a number of characters or the entire string.

```
std::string str = "Hello, World!";

str.erase(5, 7);  // Erases 7 characters starting from index 5

std::cout << str << std::endl;  // Output: Hello!
```

6. **`replace()`**

- Replaces a part of the string with another string.

```
std::string str = "Hello, World!";

str.replace(7, 5, "C++");  // Replaces "World" with "C++"

std::cout << str << std::endl;  // Output: Hello, C++!
```

7. **`substr()`**

- Returns a substring starting at a specific position, with an optional length.

```
std::string str = "Hello, World!";

std::string sub = str.substr(7, 5);  // Extracts "World"

std::cout << sub << std::endl;       // Output: World
```

8. **`find()`**

- Searches for the first occurrence of a substring or character within the string.

- Returns the index of the first match, or `std::string::npos` if not found.

```
std::string str = "Hello, World!";

size_t found = str.find("World");

if (found != std::string::npos) {

    std::cout << "'World' found at: " << found << std::endl;

}
```

9. `compare()`

- Compare two strings lexicographically.
    - Returns 0 if both strings are equal.
    - Returns a positive value if the first string is greater.
    - Returns a negative value if the first string is smaller.

```
std::string str1 = "apple";

std::string str2 = "banana";

int result = str1.compare(str2);

if (result == 0) {

    std::cout << "Strings are equal" << std::endl;

} else if (result < 0) {

    std::cout << "str1 is less than str2" << std::endl;

} else {

    std::cout << "str1 is greater than str2" << std::endl;

}
```

10. `swap()`

- Swaps the contents of two strings.

```
std::string str1 = "Hello";

std::string str2 = "World";

str1.swap(str2);  // Swaps the contents

std::cout << str1 << std::endl;  // Output: World

std::cout << str2 << std::endl;  // Output: Hello
```

11.`push_back()` **and** `pop_back()`

- `push_back()` appends a character to the end of the string.
- `pop_back()` removes the last character of the string.

```
std::string str = "Hello";

str.push_back('!');

std::cout << str << std::endl;  // Output: Hello!

str.pop_back();

std::cout << str << std::endl;  // Output: Hello
```

12.`to_string()`

- Converts a numerical value to a string.

```
int num = 123;
```

```
std::string str = std::to_string(num);

std::cout << "Number as string: " << str << std::endl;  // Output: 123
```

Conclusion

The `std::string` class in C++ provides many useful functions for manipulating strings, making it versatile and efficient for a wide variety of tasks. By using these functions, you can efficiently handle string operations like concatenation, comparison, searching, and modification. This allows for safer and more flexible string handling than using C-style character arrays.

Difference between String and Character array in C++

The main difference between a string and a character array is that strings are immutable, while character arrays are not.

String	Character Array
Strings define objects that can be represented as string streams.	The null character terminates a character array of characters.
A string class provides numerous functions for manipulating strings.	Character arrays do not offer inbuilt functions to manipulate strings.

Memory is allocated dynamically.	The size of the character array has to be allocated statically.