Strings

Introduction

Strings in C++ are a sequence of characters, represented using the std::string class from the Standard Template Library (STL). They are used to store and manipulate text or character data. The std::string class provides various member functions and operators for string operations, such as concatenation, length, comparison, and substring extraction. C++ also supports string literals, which are sequences of characters enclosed in double quotes.

Strings	Character Array
Dynamic Size: Strings represented by string in C++ allow for dynamic resizing, meaning you can easily add, remove, or modify characters within the string without worrying about managing the memory manually. String Manipulation: String class provides a rich set of functions for string manipulation, including concatenation (+ operator), finding substrings (find()), and extracting substrings (substr()), among others. Enhanced Functionality: The string class provides additional functionality like string comparison (==, !=, <, >, etc.), string length (length()).	Fixed Size: Character arrays have a fixed size determined during declaration, and the size cannot be easily changed without manually managing memory allocation and deallocation. Manual Memory Management: Unlike strings, character arrays require manual memory management, meaning you need to allocate memory, free the memory. C-style String Functions: Character arrays are compatible with traditional C-style string functions such as strcpy(), strlen(), strcmp(), which provide basic string manipulation operations.

Initialization of Strings

We can declare strings with the use of the following syntax:

Syntax:

```
String variable_name
```

Example:

```
string hello;
string name = ”LearningStrings”;
string greeting = “GoodMorning”;
```

Input and Output in Strings

To read a single string input from the user, you can use the cin stream extraction operator (>>). Here's an example code snippet:

Syntax:

```
cin>>string_name;
```

Example:

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

int main() {
    string str;
    cout << "Enter a string: ";
    cin >> str;
    return 0;
}
```

Input:

```
HelloWorld
```

Output

```
HelloWorld
```

getline

The getline function allows you to read a line of text including spaces until a newline character is encountered. The getline function is used with cin as the input stream and string variable to store the input. The user can enter a line of text, and the entire line is stored in the variable, including spaces.

Syntax:

```
getline(cin,string_name);
```

Example:

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

int main() {
    string str;
    getline(cin, str);
    cout<<str<<endl;
    return 0;
}
```

Input:

```
Hello World
```

Output:

```
Hello World
```

cin.ignore():

The **cin.ignore()** function is used to discard or skip characters from the input stream. It is commonly used after reading data using cin to ignore any remaining characters, such as newline characters (‘\n’ - when you press enter), in the input stream. It takes the number of characters to be ignored. Here's an example code snippet:

Syntax:

```
cin.ignore(n);
```

Example:

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

int main() {
    int n;
    string str2;
    cin >> n;
    cin.ignore(1);
    getline(cin, str2);
    cout << "You entered: " << n << " and " << str2 << endl;
}
```

Input:

```
10
Hello World
```

Output

```
10 Hello World
```

Comparison Operators in Strings

Strings can be compared using various comparison operators, such as == (equal to), > (greater than), < (less than), >= (greater than or equal to), and <= (less than or equal to). These operators compare strings based on their lexicographical order, which means they are compared character by character.

'==' operator

The == operator is used to check if two strings are equal. It returns true if the strings have the same content and false otherwise. Here's an example code snippet:

Example:

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

int main() {
    string str1 = "Hello";
    string str2 = "World";
    string str3 = "World";

    if (str1 == str2) {
        cout << "str1 and str2 are equal." << endl;
    } else {
        cout << "str1 and str2 are not equal." << endl;
    }
    if (str3 == str2) {
        cout << "str3 and str2 are equal." << endl;
    } else {
        cout << "str3 and str2 are not equal." << endl;
    }

    return 0;
}
```

Output:

```
str1 and str2 are not equal.
str3 and str2 are equal.
```

The '>' operator:

The > operator is used to check if one string is lexicographically greater than another. It returns true if the left operand is greater than the right operand and false.

Example:

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

int main() {
    string str1 = "Apple";
    string str2 = "Banana";

    if (str1 > str2) {
        cout << "str1 is greater than str2." << endl;
    } else {
        cout << "str1 is not greater than str2." << endl;
    }

    return 0;
}
```

In this code, the > operator is used to compare str1 and str2. Since "Banana" comes after "Apple" in lexicographical order, the output will be "str1 is not greater than str2."

Output

```
str is not greater than str2.
```

The '<' operator:

The < operator is used to check if one string is lexicographically less than another. It returns true if the left operand is less than the right operand and false otherwise.

Example:

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

int main() {
    string str1 = "Apple";
    string str2 = "Banana";

    if (str1 < str2) {
        cout << "str1 is lesser than str2." << endl;
    } else {
        cout << "str1 is not lesser than str2." << endl;
    }

    return 0;
}
```

Output

```
str1 is lesser than str2.
```

Accessibility in Strings

Accessibility in strings refers to the ability to retrieve and modify individual elements or characters within a string. This includes methods such as operator[] and at(), as mentioned in the previous subtopics. These methods provide direct access to elements in a string, allowing you to read or modify specific characters.

Operator[]

The operator[] is used to access individual characters in a string by their index. It provides direct access to the characters and allows modification. The indexing starts at 0, where the first character has an index of 0, the second character has an index of 1, and so on. Here's an example code snippet:

Syntax:

```
string_name[character_index]
```

Example:

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

int main() {
    string str = "Hello";
    char firstChar = str[0];
    char thirdChar = str[2];

    cout << "First character: " << firstChar << endl;
    cout << "Third character: " << thirdChar << endl;

    str[0] = 'J';
    cout << "Modified string: " << str << endl;
}
```

Output:

```
First character: H
Third character: l
Modified string: Jello
```

at() function

The at() function provides a way to access individual characters in a string by their index, similar to operator[]. However, at() performs bounds checking and throws an out_of_range exception if the index is out of range. Here's an example code snippet:

Syntax:

```
string_name.at(character_index)
```

Example:

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

int main() {
    string str = "Hello";

    char firstChar = str.at(0);
    char thirdChar = str.at(2);

    cout << "First character: " << firstChar << endl;
    cout << "Third character: " << thirdChar << endl;

    str.at(0) = 'J';
    cout << "Modified string: " << str << std::endl;

    return 0;
}
```

Output:

```
First character: H
Third character: l
Modified string: Jello
```

Traversal in Strings

Iteration allows you to traverse and process each character in a string sequentially. By leveraging the accessibility features and methods available for strings, we can employ loops to iterate over and perform manipulations on the elements of a string.

For loops

By using a normal for loop, you can iterate over a string using the index positions. The .size() function helps determine the length of the string, allowing you to iterate from index 0 to size() - 1. Each character at the current index can be accessed using the str[i] syntax.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string str = "Hello";
    for (int i = 0; i < str.size(); i++) {
        char ch = str[i];
        cout << "Character at index " << i << ": " << ch << endl;
    }
    return 0;
}
```

Output:

```
Character at index 0: H
Character at index 1: e
Character at index 2: l
Character at index 3: l
Character at index 4: o
```

Range-based For loop

Alternatively, you can use a range-based for loop to iterate over each character in a string directly. This loop simplifies the code by automatically handling the index and providing direct access to each character without explicitly using the .size() function.

Example:

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

int main() {
    string str = "Hello";
    for (char ch : str) {
        cout << "Character: " << ch << endl;
    }
    return 0;
}
```

Output:

```
Character at index 0: H
Character at index 1: e
Character at index 2: l
Character at index 3: l
Character at index 4: o
```

A range-based for loop is used to iterate over the characters in the string. The loop automatically iterates over each character in the string, assigning it to the variable ch in each iteration. The output displays each character one by one.

Operations on Strings

String manipulation is a common task in programming where we perform various operations on strings. By using iterations and accessibility, we can modify strings dynamically. These operations include adding or removing characters, inserting or erasing substrings, concatenating strings, and more. In C++, there are built-in functions and operators available to perform these operations efficiently.

push_back()

The push_back() function in C++ allows us to append a single character at the end of a string. It takes a character as a parameter and adds it to the string's existing characters.

Example:

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

int main() {
    string str = "Hello";
    str.push_back('!');
    cout << str << endl;
}
```

Output:

```
Hello!
```

insert()

The insert() function in C++ enables us to insert characters or substrings at specific positions within a string. It takes two parameters: the position at which insertion should occur and the string or character to be inserted.

Example:

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

int main() {
    string str = "Hello";
    str.insert(5, " World");
    cout << str << endl;
}
```

Output:

```
Hello World
```

pop_back()

The pop_back() function in C++ removes the last character from a string. It doesn't return the removed character but modifies the string directly.

Example:

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

int main() {
    string str = "Hello!";
    str.pop_back();
    cout << str << endl;
    return 0;
}
```

Output:

```
Hello
```

erase()

The erase() function in C++ removes a portion of a string specified by its starting position and length. It modifies the string directly and doesn't return anything.

Example:

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

int main() {
    string str = "Hello World";
    str.erase(6, 5);
    cout << str << endl;
}
```

Output:

```
Hello
```

'+' operator

In C++, the '+' operator can be used to concatenate two strings together. It creates a new string containing the combined characters of both strings.

Example:

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

int main() {
    string str1 = "Hello";
    string str2 = " World";
    string result = str1 + str2;
    cout << result << endl;
}
```

Output:

```
Hello World
```

append()

The append() function in C++ allows us to concatenate one string to the end of another string. It modifies the original string by adding the characters of the appended string.

Example:

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

int main() {
    string str1 = "Hello";
    string str2 = " World";
    str1.append(str2);
    cout << str1 << endl;
}
```

Output:

```
Hello World
```

Functions/Operators	Description
push_back()	The push_back() function in C++ allows us to append a single character at the end of a string.
insert()	The insert() function in C++ enables us to insert characters or strings at specific positions within a string.
pop_back()	The pop_back() function in C++ removes the last character from a string.
erase()	The erase() function in C++ removes a portion of a string specified by its starting position and length.

'+' operator	In C++, the '+' operator can be used to concatenate two strings together.
append()	The append() function in C++ allows us to concatenate one string to the end of another string.