C++ String Handling

What is String Handling? String handling refers to the process of creating, manipulating, and analyzing strings (text data) in C++. The `string` class in C++ provides a wide range of built-in functions to handle common operations such as concatenation, searching, replacing, comparing, and extracting substrings.

C-style Strings (Character Arrays)

- Represented as arrays of characters ending with a null character (`\0`).
- Header required: `<cstring>`
- More manual and error-prone.

Example:

```cpp
#include <iostream>
#include <cstring>
using namespace std;

int main() {
    char str1[20] = "Hello";
    char str2[] = "World";
    strcat(str1, str2); // Concatenates str2 to str1
    cout << str1;       // Output: HelloWorld
    return 0;
}
```

□ C++-style Strings (std::string)

- Uses the `string` class from the Standard Template Library (STL).
- Header required: `<string>`
- Easier and safer to use.

Example:

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

int main() {
    string name = "Parth";
    cout << "Hello, " << name << "!" << endl;
    return 0;
}
```

Why is it Important?

- Strings are essential for user input, output, data storage, and communication.
- Efficient string handling allows easy manipulation of textual data.
- Reduces complexity using in-built functions.

Header Required:

```
#include <string>
```

List of all Major Functons

- **`length()`**
- **`size()`**
- **`at()`**
- **`front()`**
- **`back()`**
- **`append()`**
- **`clear()`**
- **`compare()`**
- **`empty()`**
- **`erase()`**
- **`find()`**
- **`insert()`**
- **`replace()`**
- **`substr()`**
- **`swap()`**

length()

- Returns the number of characters in the string.
- Same as `size()`.
- Counts even spaces and symbols.
- Used for loops, validations, etc.
- Useful in user input length checks.

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

int main() {
    string str = "Hello World";
    cout << "Length: " << str.length();
    return 0;
}
```

Output:

```
Length: 11
```

---

size()

- Same as `length()`.
- Returns total characters.
- No difference from `length()`.
- Available for compatibility.
- Often used in STL operations.

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

int main() {
    string str = "OpenAI";
    cout << "Size: " << str.size();
    return 0;
}
```

Output:

```
Size: 6
```

---

at()

- Returns character at given index.
- Indexing starts from 0.
- Performs bounds checking.
- Throws out_of_range error if invalid.
- Safer than using `[]` operator.

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

int main() {
    string str = "Hello";
    cout << "Character at index 1: " << str.at(1);
    return 0;
}
```

Output:

```
Character at index 1: e
```

---

front()

- Returns the first character.
- Equivalent to `str[0]`.
- Useful in validations.
- No argument required.
- Throws error if string is empty.

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

int main() {
    string str = "C++";
    cout << "Front: " << str.front();
    return 0;
}
```

Output:

```
Front: C
```

---

back()

- Returns the last character.
- Equivalent to `str[str.length()-1]`.
- Easy way to check suffix.
- Throws error if string is empty.
- No arguments required.

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

int main() {
    string str = "Java";
    cout << "Back: " << str.back();
    return 0;
}
```

Output:

```
Back: a
```

---

append()

- Adds string to the end.
- Supports both `append(str)` and `+=`.
- Doesn't modify the original string unless assigned.
- Can chain multiple appends.
- Efficient way to build strings.

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

int main() {
    string str1 = "Hello ";
    string str2 = "World!";
    str1.append(str2);
    cout << "Appended: " << str1;
    return 0;
}
```

Output:

```
Appended: Hello World!
```

---

clear()

- Removes all characters.
- Sets length to 0.
- Doesn't deallocate memory.
- After clear, string becomes empty.
- Good for reusing strings.

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

int main() {
    string str = "Temporary";
    str.clear();
    cout << "Length after clear: " << str.length();
    return 0;
}
```

Output:

```
Length after clear: 0
```

---

compare()

- Compares two strings.
- Returns 0 if equal, <0 if first is smaller, >0 if greater.
- Case-sensitive.
- Lexicographical comparison.
- Safer than relational operators.

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

int main() {
    string a = "apple";
    string b = "banana";
    cout << "Compare: " << a.compare(b);
    return 0;
}
```

Output:

```
Compare: -1
```

---

empty()

- Checks if string is empty.
- Returns true/false.
- Equivalent to `str.length() == 0`.
- Safer for validations.
- Good before accessing front/back.

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

int main() {
    string s = "";
    if (s.empty())
        cout << "String is empty.";
    else
        cout << "Not empty.";
    return 0;
}
```

Output:

```
String is empty.
```

---

erase()

- Removes characters.
- Specify position and count.
- Modifies original string.
- Useful to delete parts.
- Throws error if range invalid.

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

int main() {
    string str = "Hello World!";
    str.erase(5, 6);
    cout << str;
    return 0;
}
```

Output:

```
Hello
```

---

find()

- Finds position of first occurrence.
- Returns index.
- Returns `string::npos` if not found.
- Case-sensitive.
- Can specify start position.

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

int main() {
    string s = "programming";
    cout << "Index of 'gram': " << s.find("gram");
    return 0;
}
```

Output:

```
Index of 'gram': 3
```

---

insert()

- Inserts a string at position.
- Shifts existing chars.
- Useful for formatting.
- Can insert substring too.
- Returns modified string.

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

int main() {
    string s = "Hello!";
    s.insert(5, " World");
    cout << s;
    return 0;
}
```

Output:

```
Hello World!
```

---

replace()

- Replaces part of string.

- Specify start index and count.
- Replaces with given string.
- Useful in search-replace tasks.
- Can be chained with other methods.

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

int main() {
    string s = "I like C++";
    s.replace(7, 3, "Java");
    cout << s;
    return 0;
}
```

Output:

```
I like Java
```

---

substr()

- Extracts substring.
- Takes start index and length.
- Doesn’t modify original.
- Can get suffix or prefix.
- Throws error if out of range.

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

int main() {
    string str = "HelloWorld";
    string sub = str.substr(0, 5);
    cout << sub;
    return 0;
}
```

Output:

```
Hello
```

---

swap()

- Swaps contents of two strings.
- Faster than copying.
- Built-in memory management.
- Useful in sorting.
- No return value.

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

int main() {
    string a = "First", b = "Second";
    a.swap(b);
    cout << "a: " << a << "\nb: " << b;
    return 0;
}
```

Output:

```
a: Second
b: First
```