StringTools

1.0.1

Generated by Doxygen 1.9.1

Chapter 1

StringTools

The strTools namespace provides a set of tools for manipulating C-style strings. These functions are designed to simplify common string operations, such as concatenation, substring extraction, insertion, deletion, searching, and replacement. The library ensures proper memory management using std::unique_ptr<char[]>.

1.1 Index

1.2 Overview

This project provides a string manipulation library to easily handle strings. It includes functions for calculating the length of a string, concatenating strings, searching for a substring, and generating substrings. The main.cpp file demonstrates examples of how to use this library through a simple console menu.

1.3 Namespace features

- **Length Calculation:** Calculate the length of a C-string.
- **Concatenation:** Concatenate two C-strings into a new dynamically allocated string.
- **Substring Extraction:** Extract a substring from a C-string.
- **Insertion:** Insert one C-string into another at a specified position.
- **Deletion:** Remove a substring from a C-string.
- **Searching:** Find the first occurrence of a substring within a C-string.
- **Replacement:** Replace the first occurrence of a substring with another substring.

1.4 Main function features

- **Calculate the Length of a String:** Enter a string and get its length.
- **Concatenate Strings:** Enter three strings and concatenate them.
- **Search for a Substring:** Enter a string and a substring to find its position within the string.
- **Generate a Substring:** Enter a string and generate a random substring from it.
- **Exit:** Exit the program.

1.5 Installation

To use the `strTools` library, include the `strtools.hh` header in your C++ project:

```
#include "src/strtools.hh"
// or
#include "strtools.hh"
```

Ensure that your project is set up to find the header file in its include path.

You can also try running the test program using:

```
g++ -std=c++20 -I src main.cpp -o main.exe && ./main.exe
```

1.6 Namespace Usage

1.6.1 Length Calculation

Calculate the length of a C-string.

```
const char* myString = "Hello, World!";
uint64_t length = strTools::len(myString);  // length will be 13
```

1.6.2 Concatenation

Concatenate two C-strings into a new unique_ptr<char[]>.

```
const char* str1 = "Hello, ";
const char* str2 = "World!";
auto result = strTools::concatStr(str1, str2);
// result will contain "Hello, World!"
```

1.6.3 Substring Extraction

Extract a substring from a C-string.

```
const char* myString = "Hello, World!";
auto sub = strTools::subStr(myString, 7, 5);
// sub will contain "World"
```

1.6.4 Insertion

Insert one C-string into another at a specified position.

```
const char* str1 = "Hello, World!";
const char* str2 = "Beautiful ";
auto result = strTools::insertStr(str1, str2, 8);
// result will contain "Hello, Beautiful World!"
```

1.6.5 Deletion

Remove a substring from a C-string.

```
const char* myString = "Hello, World!";
auto result = strTools::delSubStr(myString, 7, 6);
// result will contain "Hello, !"
```

1.6.6 Searching

Find the first occurrence of a substring within a C-string.

```
const char* myString = "Hello, World!";
int64_t index = strTools::findSubStr(myString, "World");
// index will be 7
```

1.6.7 Replacement

Replace the first occurrence of a substring with another substring.

```
const char* myString = "Hello, World!";
const char* sub1 = "World";
const char* sub2 = "Universe";
auto result = strTools::replaceStr(myString, sub1, sub2);
// result will contain "Hello, Universe!"
```

1.7 Main function Usage

NOTE: The `main` function requires C++20. If you are using C++17, this section will not compile.

1.7.1 Menu Options

1. **Calculate the Length of a String:**
   - Prompts the user to enter a string.
   - Calculates the length using `strTools::len`.
2. **Concatenate Three Strings:**
   - Prompts the user to enter three strings.
   - Concatenates them using `strTools::concatStr`.
   - Displays the concatenated result.
3. **Search for a Substring:**
   - Prompts the user to enter a string and a substring.
   - Searches for the substring using `strTools::findSubStr`.
   - Extracts the substring using `strTools::subStr`.
   - Displays the result or an error message if the substring is not found.
4. **Generate a Substring from a String:**
   - Prompts the user to enter a string.
   - Generates random start and end indices.
   - Extracts a substring using `strTools::subStr`.
   - Displays the extracted substring.
5. **Exit:**
   - Exits the program.

1.7.2 Example Usage

1. Run the program.
2. Select an option by entering a number (0-4).
3. Follow the prompts to perform the desired operation.
4. View the result or error message.
5. Repeat until you choose to exit (option 0).

1.7.3 Input Handling

The program uses the `helpers` namespace to manage invalid inputs, out-of-bounds values, and user input for different operations:

- **Invalid Input:** If the input is invalid (non-numeric), an error message is shown.
- **Out-of-Bounds Input:** If the input is not within the range [0, 4], an error message is shown.
- **User Input Handling:** Manages input from the user, including handling exit commands and input overflow.

1.7.4 String Operations

The `strtools` namespace provides the following functions for string manipulation:

- **Length Calculation (`strTools::len`):** Calculates the length of a string.
- **Concatenation (`strTools::concatStr`):** Concatenates multiple strings.
- **Substring Search (`strTools::findSubStr`):** Finds the position of a substring within a string.
- **Substring Extraction (`strTools::subStr`):** Extracts a substring from a string based on start and end indices.

1.8 Full Documentation

For more detailed documentation on the code, including function descriptions and usage, refer to the Doxygen documentation available here.

1.9 License

The `strTools` library is licensed under the GNU General Public License v3.0. For more details, see the GNU General Public License v3.0 file.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 __strToolsUtil Namespace Reference

Utility tools for the strTools namespace.

4.1.1 Detailed Description

Utility tools for the strTools namespace.

This namespace contains helper functions that are designed to be used internally. If you find yourself needing to use this namespace directly, you are likely doing something wrong. These functions provide basic utilities for string manipulation and error checking.

4.2 helpers Namespace Reference

Utility functions for input handling and console management.

Functions

- bool isCapturedValueInvalid ()

  Checks if the captured value from standard input is invalid.
- void clearScr ()

  "Clears" the console screen.
- bool userInputHandler (char ∗input, int32_t size)

  Handles user input, checks for exit command, and handles overflow.

4.2.1 Detailed Description

Utility functions for input handling and console management.

This namespace provides a set of utility functions for handling standard input errors, checking bounds of values, clearing the console screen, and managing user input with overflow and exit command handling.

4.2.2 Function Documentation

4.2.2.1 clearScr()

```
void helpers::clearScr ( )
```

"Clears" the console screen.

This function sends escape sequences to the console to clear the screen and move the cursor to the top-left corner. This functionality is platform-specific and might not work on all terminals.

Note

Example usage:

```
helpers::clearScr();
std::cout « "Screen cleared.\n";
```

4.2.2.2 isCapturedValueInvalid()

```
bool helpers::isCapturedValueInvalid ( )
```

Checks if the captured value from standard input is invalid.

This function checks if the last input operation on `cin` failed (e.g., due to non-numeric input). If it fails, it performs the following:

- Clears the error flags from `cin` to allow further input.
- Ignores the remaining invalid input in the stream up to the next newline.

Returns

`true` if the captured value was invalid, `false` otherwise.

Note

Example usage:

```
int value;
std::cin » value;
if (helpers::isCapturedValueInvalid()) {
    std::cout « "Invalid input. Please enter a numeric value.\n";
}
```

4.2.2.3 userInputHandler()

```
bool helpers::userInputHandler (
            char * input,
            int32_t size )
```

Handles user input, checks for exit command, and handles overflow.

This function reads user input from the standard input stream until a newline character is encountered or the buffer is full. It checks if the user input is the exit command "/exit". If the input exceeds the buffer size, it truncates the input and clears the remaining characters in the input stream.

Parameters

input	Pointer to the character array where the input will be stored.
size	The size of the input buffer.

Returns

`true` if the input is the exit command "/exit".

`false` otherwise.

Note

Example usage:

```
const int32_t bufferSize = 100;
char input[bufferSize];
bool exit = helpers::userInputHandler(input, bufferSize);
if (exit) {
    std::cout « "Exit command received.\n";
} else {
    std::cout « "You entered: " « input « "\n";
}
```

4.3 strTools Namespace Reference

String manipulation tools.

Functions

- uint64_t len (const char *n) noexcept

  Returns the length of the C-string.
- std::unique_ptr< char[]> concatStr (const char *s1, const char *s2) noexcept

  Concatenates two C-strings into a new unique_ptr<char[]>.
- std::unique_ptr< char[]> subStr (const char *s, const uint64_t i, uint64_t j)

  Extracts a substring from a string.
- std::unique_ptr< char[]> insertStr (const char *s1, const char *s2, const uint64_t i)

  Inserts one string into another at the specified position.
- std::unique_ptr< char[]> delSubStr (const char *s, const uint64_t i, const uint64_t j)

  Removes a substring from a string.
- int64_t findSubStr (const char *s, const char *find)

  Finds the first occurrence of a substring within a string.
- std::unique_ptr< char[]> replaceStr (const char *s, const char *sub1, const char *sub2)

  Replaces the first occurrence of a substring with another substring.

4.3.1 Detailed Description

String manipulation tools.

This namespace provides a set of functions for various string operations, including length calculation, concatenation, substring extraction, insertion, deletion, finding substrings, and replacement of substrings. These functions use C-style strings and return results in `std::unique_ptr<char[]>` to ensure proper memory management.

4.3.2 Function Documentation

4.3.2.1 concatStr()

```
std::unique_ptr<char[]> strTools::concatStr (
            const char * s1,
            const char * s2 )  [noexcept]
```

Concatenates two C-strings into a new unique_ptr<char[]>.

This function takes two C-strings and concatenates them into a new dynamically allocated string managed by `std::unique_ptr<char[]>`.

Parameters

s1	The first source C-string.
s2	The second source C-string.

Returns

A unique_ptr<char[]> containing the concatenated string.

Note

Example usage:

```
const char* str1 = "Hello, ";
const char* str2 = "World!";
auto result = strTools::concatStr(str1, str2);
// result will contain "Hello, World!"
```

4.3.2.2 delSubStr()

```
std::unique_ptr<char[]> strTools::delSubStr (
            const char * s,
            const uint64_t i,
            const uint64_t j )
```

Removes a substring from a string.

This function removes a substring from the source string `s` starting at position `i` and having length `j`. The resulting string is returned as a `std::unique_ptr<char[]>`.

Parameters

s	The source C-string.
i	The starting position of the substring to be removed.
j	The length of the substring to be removed.

Returns

A unique_ptr<char[]> containing the resulting string.

Exceptions

std::out_of_range	if indices are out of bounds.

Note

Example usage:

```
const char* myString = "Hello, World!";
auto result = strTools::delSubStr(myString, 7, 6);
// result will contain "Hello, !"
```

4.3.2.3 findSubStr()

```
int64_t strTools::findSubStr (
            const char * s,
            const char * find )
```

Finds the first occurrence of a substring within a string.

This function searches for the first occurrence of the substring `find` within the source string `s`. It returns the index of the first occurrence, or `INT64_MAX` if the substring is not found.

Parameters

s	The source C-string.
find	The substring to find.

Returns

The index of the first occurrence of the substring, or INT64_MAX if not found.

Note

Example usage:

```
const char* myString = "Hello, World!";
int64_t index = strTools::findSubStr(myString, "World");
// index will be 7
```

4.3.2.4 insertStr()

```
std::unique_ptr<char[]> strTools::insertStr (
            const char * s1,
```

```
const char * s2,
const uint64_t i )
```

Inserts one string into another at the specified position.

This function inserts the source string `s2` into the destination string `s1` at the specified position `i`. The resulting string is returned as a `std::unique_ptr<char[]>`.

Parameters

s1	The destination C-string.
s2	The source C-string to be inserted.
i	The position at which to insert s2 into s1.

Returns

A unique_ptr<char[]> containing the resulting string.

Exceptions

std::out_of_range	if the position is out of bounds.

Note

Example usage:

```
const char* str1 = "Hello, World!";
const char* str2 = "Beautiful ";
auto result = strTools::insertStr(str1, str2, 8);
// result will contain "Hello, Beautiful World!"
```

4.3.2.5 len()

```
uint64_t strTools::len (
            const char * n )  [noexcept]
```

Returns the length of the C-string.

This function calculates the length of a given C-string by using the `strlen` function from the C standard library.

Parameters

n	The source C-string.

Returns

The length of the string.

Note

Example usage:

```
const char* myString = "Hello, World!";
uint64_t length = strTools::len(myString);  // length will be 13
```

4.3.2.6 replaceStr()

```
std::unique_ptr<char[]> strTools::replaceStr (
            const char * s,
            const char * sub1,
            const char * sub2 )
```

Replaces the first occurrence of a substring with another substring.

This function replaces the first occurrence of the substring `sub1` in the source string `s` with the substring `sub2`. The resulting string is returned as a `std::unique_ptr<char[]>`.

Parameters

s	The source C-string.
sub1	The substring to be replaced.
sub2	The substring to replace with.

Returns

A unique_ptr<char[]> containing the resulting string.

Note

Example usage:

```
const char* myString = "Hello, World!";
const char* sub1 = "World";
const char* sub2 = "Universe";
auto result = strTools::replaceStr(myString, sub1, sub2);
// result will contain "Hello, Universe!"
```

4.3.2.7 subStr()

```
std::unique_ptr<char[]> strTools::subStr (
            const char * s,
            const uint64_t i,
            uint64_t j )
```

Extracts a substring from a string.

This function extracts a substring from a given C-string starting at position `i` and having `j` characters. The extracted substring is returned as a `std::unique_ptr<char[]>`.

Parameters

s	The source C-string.
i	Position of the first character to include (index 0 = first character).
j	Number of characters to extract from i.

Returns

A unique_ptr<char[]> containing the extracted substring.

Exceptions

std::out_of_range	if indices are out of bounds.

Note

Example usage:

```
const char* myString = "Hello, World!";
auto sub = strTools::subStr(myString, 7, 5);
// sub will contain "World"
```

Chapter 5

File Documentation

5.1 helpers.hh File Reference

Utilities for input handling and console management.

```
#include <cstdint>
#include <iosfwd>
#include <iostream>
#include <limits>
#include <string.h>
```

Include dependency graph for helpers.hh:

5.2 main.cpp File Reference

Examples on how to use the strtools.hh header.

```
#include "src/helpers.hh"
#include "src/strtools.hh"
#include <array>
#include <cstdint>
#include <iostream>
#include <istream>
#include <memory>
#include <new>
#include <random>
#include <string>
#include <string.h>
```

Include dependency graph for main.cpp:

Macros

- #define STRING_MAX_SIZE 256

  String max input size.
- #define EMPTY_CHAR (char∗) ""

  Array of characters with an empty array.

Functions

- int main ()

  Main function demonstrating examples on how to use the strtools.hh header.

5.2.1 Detailed Description

Examples on how to use the strtools.hh header.

Author

Zperk

Version

1.0.1

Date

2024-07-28

Copyright

Copyright (c) zperk 2024

5.2.2 Function Documentation

5.2.2.1 main()

```
int main ( )
```

Main function demonstrating examples on how to use the strtools.hh header.

The main function provides a console menu to the user with various string manipulation options. The user can choose to calculate the length of a string, concatenate strings, search for a substring, or generate a substring from a string. The main loop continues until the user chooses to exit.

The main function works as follows:

- Initialization:
  - `mainLoop`: A boolean variable to control the main loop.
  - `selector`: An integer to capture the user's menu choice.
  - `extraMsg`: A character pointer initialized with a smiley face.
- Main Loop: The loop will continue to display the menu, handle user input, and perform the chosen operation until the user decides to exit.

```
do {
    helpers::clearScr();
    cout «
        "1. Calculate the length of a string.\n"
        "2. Concatenate three strings requested.\n"
        "3. Search for a character in a string.\n"
        "4. Generate a substring from a string.\n"
        "0. Exit.\n"
        « extraMsg « "\n\n> " « flush;
    cin » selector;
```

- Input Validation:
  - Invalid Input: If the input is invalid (non-numeric), an error message is shown.
  - Out-of-Bounds Input: If the input is not within the range [0, 4], an error message is shown.

```
if( helpers::isCapturedValueInvalid() ) {
    extraMsg = "Value is invalid!";
    continue;
}
if( helpers::isOutOfBounds(selector, 0, 4) ) {
    extraMsg = "Value is out of bounds!";
    continue;
}
```

- Menu Options: Depending on the user's choice, different cases in the switch statement are executed:
  - Case 0: Exit the program by setting `mainLoop` to `false`.
  - Case 1: Calculate the length of a string.
  - Case 2: Concatenate three user-input strings.
  - Case 3: Search for a substring within a string.
  - Case 4: Generate a random substring from a user-input string.

```
switch( selector ) {
case 0:
    mainLoop = false;
    break;
case 1:
    // Code for calculating the length of a string
    break;
case 2:
    // Code for concatenating three strings
    break;
case 3:
    // Code for searching for a substring in a string
    break;
case 4:
    // Code for generating a substring from a string
    break;
}
```

- Case Details:
  - Case 1: Calculate the Length of a String: Prompts the user to enter a string and calculates its length using `strTools::len`.
  - Case 2: Concatenate Three Strings: Prompts the user to enter three strings, concatenates them using `strTools::concatStr`, and displays the result.
  - Case 3: Search for a Character in a String: Prompts the user to enter a string and a substring, then searches for the substring using `strTools::findSubStr` and extracts it using `strTools::subStr`.
  - Case 4: Generate a Substring from a String: Prompts the user to enter a string, generates random start and end indices, and extracts a substring using `strTools::subStr`.
- Ending the Program: After the loop exits, a farewell message is displayed, and the program terminates.

```
cout « "Bye bye!" « endl;
return 0;
```

Example Use:

1. The user starts the program, and the main menu is displayed.
2. The user selects an option by entering a number (0-4).
3. The program performs the corresponding operation and displays the result or an error message if applicable.
4. The menu is displayed again, allowing the user to make another selection.
5. The loop continues until the user selects the exit option (0).

Key Points:

- Input Handling: The program uses the `helpers` namespace to manage invalid inputs, out-of-bounds values, and user input for different operations.
- String Operations: The `strtools` namespace provides functions for string length calculation, concatenation, substring search, and substring extraction.
- Menu Loop: The main loop ensures the user can repeatedly perform operations until they choose to exit.

5.3 strtools.hh File Reference

String manipulation tools.

```
#include <algorithm>
#include <cctype>
#include <cstdint>
#include <cstring>
#include <memory>
#include <stdexcept>
#include <string>
#include <string.h>
```

Include dependency graph for strtools.hh: This graph shows which files directly or indirectly include this file:

Namespaces

- __strToolsUtil
  Utility tools for the strTools namespace.
- strTools
  String manipulation tools.

Functions

- uint64_t strTools::len (const char *n) noexcept
  Returns the length of the C-string.
- std::unique_ptr< char[]> strTools::concatStr (const char *s1, const char *s2) noexcept
  Concatenates two C-strings into a new unique_ptr<char[]>.
- std::unique_ptr< char[]> strTools::subStr (const char *s, const uint64_t i, uint64_t j)
  Extracts a substring from a string.
- std::unique_ptr< char[]> strTools::insertStr (const char *s1, const char *s2, const uint64_t i)
  Inserts one string into another at the specified position.
- std::unique_ptr< char[]> strTools::delSubStr (const char *s, const uint64_t i, const uint64_t j)
  Removes a substring from a string.
- int64_t strTools::findSubStr (const char *s, const char *find)
  Finds the first occurrence of a substring within a string.
- std::unique_ptr< char[]> strTools::replaceStr (const char *s, const char *sub1, const char *sub2)
  Replaces the first occurrence of a substring with another substring.

5.3.1 Detailed Description

String manipulation tools.

Author

Ian Hylton

Version

1.0.0

Date

2024-06-16

Copyright