CString

1.0

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 CString Struct Reference

Thread-safe dynamic string container.

```
#include <cstr.h>
```

Public Attributes

- char ∗ data

  Character buffer.
- size_t length

  Current string length.
- size_t capacity

  Allocated buffer size.
- CRITICAL_SECTION cs

  Thread synchronization primitive.

3.1.1 Detailed Description

Thread-safe dynamic string container.

3.1.2 Member Data Documentation

3.1.2.1 capacity

```
size_t CString::capacity
```

Allocated buffer size.

3.1.2.2 cs

```
CRITICAL_SECTION CString::cs
```

Thread synchronization primitive.

3.1.2.3 data

```
char* CString::data
```

Character buffer.

3.1.2.4 length

```
size_t CString::length
```

Current string length.

The documentation for this struct was generated from the following file:

- include/cstr.h

Chapter 4

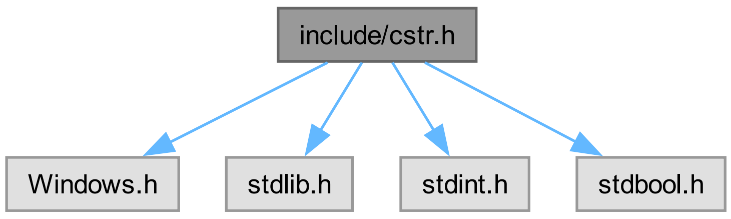
File Documentation

4.1 include/cstr.h File Reference

Thread-safe dynamic string implementation for C.

```
#include <Windows.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for cstr.h:



Classes

- struct CString

  Thread-safe dynamic string container.

Macros

- #define CSTR_H

Functions

- char ∗ cstr_strdup (_In_ const char ∗str)

  Duplicate null-terminated C string.
- wchar_t ∗ cstr_wcsdup (_In_ const wchar_t ∗str)

  Duplicate null-terminated wide string.
- bool cstr_create (_Inout_ CString ∗obj)

  Initialize a new empty CString.
- bool cstr_create_from_cstr (_Inout_ CString ∗obj, _In_ CString ∗obj2)

  Create CString copy from another CString.
- bool cstr_create_from_chars (_Inout_ CString ∗obj, _In_ const char ∗data)

  Create CString from null-terminated C string.
- bool cstr_create_from_wchars (_Inout_ CString ∗obj, _In_ const wchar_t ∗data)

  Create CString from wide character string.
- bool cstr_create_from_buffer (_Inout_ CString ∗obj, _In_ uint8_t ∗buffer, _In_ size_t size)

  Create CString from binary buffer.
- bool cstr_destroy (_In_ CString ∗obj)

  Destroy CString and release resources.
- void cstr_lock (_In_ CString ∗obj)

  Acquire exclusive access.
- void cstr_unlock (_In_ CString ∗obj)

  Release exclusive access.
- boolean cstr_at (_In_ CString ∗obj, _In_ size_t index, _Inout_ char ∗chr)

  Get character at specific index.
- char cstr_get (_In_ CString ∗obj, _In_ size_t index)

  Direct character access (unsynchronized)
- char cstr_front (_In_ CString ∗obj)

  Get first character.
- char cstr_back (_In_ CString ∗obj)

  Get last character.
- char ∗ cstr_data (_In_ CString ∗obj)

  Get raw character buffer.
- size_t cstr_length (_In_ CString ∗obj)

  Get current string length.
- size_t cstr_capacity (_In_ CString ∗obj)

  Get allocated buffer capacity.
- bool cstr_empty (_In_ CString ∗obj)

  Check if string is empty.
- bool cstr_resize (_In_ CString ∗obj, _In_ size_t size)

  Resize internal buffer.
- bool cstr_shrink_to_fit (_In_ CString ∗obj)

  Minimize buffer to fit current contents.
- bool cstr_clear (_In_ CString ∗obj)

  Clear string contents.
- bool cstr_push_back_char (_In_ CString ∗obj, _In_ char chr)

  Append single ASCII character.
- bool cstr_push_back_wchar (_In_ CString ∗obj, _In_ wchar_t chr)

  Append wide character.
- bool cstr_pop_back (_In_ CString ∗obj)

  Remove last character.
- bool cstr_append_cstr (_In_ CString ∗obj, _In_ CString ∗obj2)

Append CString contents.
- bool cstr_append_chars (_In_ CString ∗obj, _In_ const char ∗data)

  Append C string.
- bool cstr_append_wchars (_In_ CString ∗obj, _In_ const wchar_t ∗data)

  Append wide string.
- bool cstr_substring (_In_ CString ∗obj, _Inout_ CString ∗dest, _In_ size_t start, _In_ size_t length)

  Extract substring.
- bool cstr_erase (_In_ CString ∗obj, _In_ size_t index, _In_ size_t size)

  Remove characters.
- bool cstr_insert (_In_ CString ∗obj, _In_ size_t index, _In_ char chr)

  Insert character.
- bool cstr_swap (_In_ CString ∗obj, _In_ CString ∗obj2)

  Swap contents between two CStrings.
- size_t cstr_find_cstr (_In_ CString ∗obj, _In_ CString ∗obj2)

  Find substring (CString)
- size_t cstr_find_chars (_In_ CString ∗obj, _In_ const char ∗data)

  Find substring (C string)
- size_t cstr_find_wchars (_In_ CString ∗obj, _In_ const wchar_t ∗data)

  Find substring (wide string)
- bool cstr_to_upper (_In_ CString ∗obj)

  Convert to uppercase.
- bool cstr_to_lower (_In_ CString ∗obj)

  Convert to lowercase.
- bool cstr_trim (_In_ CString ∗obj)

  Trim whitespace from both ends.
- bool cstr_tokenize (_In_ CString ∗obj, _Inout_ CString ∗token, _In_ const char ∗delimiters, _Inout_ size_t ∗start_pos)

  Extract token using delimiters.
- bool cstr_tokenize_ex (_In_ CString ∗obj, _Inout_ CString ∗token, _In_ const char ∗delimiters, _In_ const char ∗zone_pairs, _In_ const char ∗escape_chars, _Inout_ size_t ∗start_pos)

  Advanced tokenization with zones/escaping.

Variables

- static const size_t invalid = (size_t)-1

4.1.1 Detailed Description

Thread-safe dynamic string implementation for C.

4.1.2 Macro Definition Documentation

4.1.2.1 CSTR_H

```
#define CSTR_H
```

4.1.3 Function Documentation

4.1.3.1 cstr_append_chars()

```
bool cstr_append_chars (
            _In_ CString * obj,
            _In_ const char * data)
```

Append C string.

Parameters

obj	Destination CString
data	Null-terminated source string

Returns

true on success

4.1.3.2 cstr_append_cstr()

```
bool cstr_append_cstr (
            _In_ CString * obj,
            _In_ CString * obj2)
```

Append CString contents.

Parameters

obj	Destination CString
obj2	Source CString

Returns

true on success

4.1.3.3 cstr_append_wchars()

```
bool cstr_append_wchars (
            _In_ CString * obj,
            _In_ const wchar_t * data)
```

Append wide string.

Parameters

obj	Destination CString
data	Null-terminated wide string

Returns

true on success

Note

Converts using system code page

4.1.3.4 cstr_at()

```
boolean cstr_at (
            _In_ CString * obj,
            _In_ size_t index,
            _Inout_ char * chr)
```

Get character at specific index.

Parameters

obj	CString object
index	Character position (0-based)
chr	Output character

Returns

true if index valid, false otherwise

Note

Thread-safe version with bounds checking

4.1.3.5 cstr_back()

```
char cstr_back (
            _In_ CString * obj)
```

Get last character.

Parameters

obj	CString object

Returns

Last character or 0 if empty

4.1.3.6 cstr_capacity()

```
size_t cstr_capacity (
            _In_ CString * obj)
```

Get allocated buffer capacity.

Parameters

obj	CString object

Returns

Capacity in bytes or CSTR_INVALID

4.1.3.7 cstr_clear()

```
bool cstr_clear (
            _In_ CString * obj)
```

Clear string contents.

Parameters

obj	CString object

Returns

true on success

Note

Securely erases buffer and resets length

4.1.3.8 cstr_create()

```
bool cstr_create (
            _Inout_ CString * obj)
```

Initialize a new empty CString.

Parameters

obj	Pointer to CString object to initialize

Returns

true on success, false on allocation failure

Note

Creates empty string with capacity 1

4.1.3.9 cstr_create_from_buffer()

```
bool cstr_create_from_buffer (
            _Inout_ CString * obj,
            _In_ uint8_t * buffer,
            _In_ size_t size)
```

Create CString from binary buffer.

Parameters

obj	Destination CString
buffer	Source binary data
size	Number of bytes to copy

Returns

true on success, false on allocation failure

Note

Adds null-terminator after buffer contents

4.1.3.10 cstr_create_from_chars()

```
bool cstr_create_from_chars (
             _Inout_ CString * obj,
             _In_ const char * data)
```

Create CString from null-terminated C string.

Parameters

obj	Destination CString
data	Source C string

Returns

true on success, false on allocation failure

4.1.3.11 cstr_create_from_cstr()

```
bool cstr_create_from_cstr (
             _Inout_ CString * obj,
             _In_ CString * obj2)
```

Create CString copy from another CString.

Parameters

obj	Destination CString
obj2	Source CString

Returns

true on success, false on allocation failure

4.1.3.12 cstr_create_from_wchars()

```
bool cstr_create_from_wchars (
             _Inout_ CString * obj,
             _In_ const wchar_t * data)
```

Create CString from wide character string.

Parameters

obj	Destination CString
data	Source wide string

Returns

true on success, false on conversion/allocation failure

Note

Uses WideCharToMultiByte with ANSI code page

4.1.3.13 cstr_data()

```
char * cstr_data (
             _In_ CString * obj)
```

Get raw character buffer.

Parameters

obj	CString object

Returns

Pointer to internal buffer

Warning

Buffer valid until next modifying operation

4.1.3.14 cstr_destroy()

```
bool cstr_destroy (
             _In_ CString * obj)
```

Destroy CString and release resources.

Parameters

obj	CString to destroy

Returns

true on success, false for invalid object

Note

Securely erases memory before freeing

4.1.3.15 cstr_empty()

```
bool cstr_empty (
             _In_ CString * obj)
```

Check if string is empty.

Parameters

obj	CString object

Returns

true if empty, false otherwise

4.1.3.16 cstr_erase()

```
bool cstr_erase (
            _In_ CString * obj,
            _In_ size_t index,
            _In_ size_t size)
```

Remove characters.

Parameters

obj	CString object
index	Starting position
size	Number of characters to remove

Returns

true on success

4.1.3.17 cstr_find_chars()

```
size_t cstr_find_chars (
            _In_ CString * obj,
            _In_ const char * data)
```

Find substring (C string)

Parameters

obj	CString to search
data	Null-terminated substring

Returns

Starting index or CSTR_INVALID

4.1.3.18 cstr_find_cstr()

```
size_t cstr_find_cstr (
            _In_ CString * obj,
            _In_ CString * obj2)
```

Find substring (CString)

Parameters

obj	CString to search
obj2	Substring to find

Returns

Starting index or CSTR_INVALID

4.1.3.19 cstr_find_wchars()

```
size_t cstr_find_wchars (
            _In_ CString * obj,
            _In_ const wchar_t * data)
```

Find substring (wide string)

Parameters

obj	CString to search
data	Null-terminated wide substring

Returns

Starting index or CSTR_INVALID

Note

Converts using system code page

4.1.3.20 cstr_front()

```
char cstr_front (
            _In_ CString * obj)
```

Get first character.

Parameters

obj	CString object

Returns

First character or 0 if empty

4.1.3.21 cstr_get()

```
char cstr_get (
            _In_ CString * obj,
            _In_ size_t index)
```

Direct character access (unsynchronized)

Parameters

obj	CString object
index	Character position

Returns

Character or 0 for invalid index

Warning

Not thread-safe - use between lock/unlock calls

4.1.3.22 cstr_insert()

```
bool cstr_insert (
             _In_ CString * obj,
             _In_ size_t index,
             _In_ char chr)
```

Insert character.

Parameters

obj	CString object
index	Insertion position
chr	Character to insert

Returns

true on success

4.1.3.23 cstr_length()

```
size_t cstr_length (
             _In_ CString * obj)
```

Get current string length.

Parameters

obj	CString object

Returns

Length in bytes or CSTR_INVALID

4.1.3.24 cstr_lock()

```
void cstr_lock (
             _In_ CString * obj)
```

Acquire exclusive access.

Parameters

obj	CString object

4.1.3.25 cstr_pop_back()

```
bool cstr_pop_back (
             _In_ CString * obj)
```

Remove last character.

Parameters

obj	CString object

Returns

true if character removed, false if empty

4.1.3.26 cstr_push_back_char()

```
bool cstr_push_back_char (
             _In_ CString * obj,
             _In_ char chr)
```

Append single ASCII character.

Parameters

obj	CString object
chr	Character to append

Returns

true on success

4.1.3.27 cstr_push_back_wchar()

```
bool cstr_push_back_wchar (
             _In_ CString * obj,
             _In_ wchar_t chr)
```

Append wide character.

Parameters

obj	CString object
chr	Wide character to append

Returns

true on success

Note

Converts to multibyte using system code page

4.1.3.28 cstr_resize()

```
bool cstr_resize (
             _In_ CString * obj,
             _In_ size_t size)
```

Resize internal buffer.

Parameters

obj	CString object
size	New buffer size

Returns

true on success, false on allocation failure

Note

Does not modify string contents

4.1.3.29 cstr_shrink_to_fit()

```
bool cstr_shrink_to_fit (
            _In_ CString * obj)
```

Minimize buffer to fit current contents.

Parameters

obj	CString object

Returns

true on success, false on allocation failure

4.1.3.30 cstr_strdup()

```
char * cstr_strdup (
            _In_ const char * str)
```

Duplicate null-terminated C string.

Parameters

str	Source string to copy

Returns

New allocated copy on success, NULL on failure

Note

Safe replacement for non-standard strdup()

Warning

Caller must free result with free()

4.1.3.31 cstr_substring()

```
bool cstr_substring (
            _In_ CString * obj,
            _Inout_ CString * dest,
            _In_ size_t start,
            _In_ size_t length)
```

Extract substring.

Parameters

obj	Source CString
dest	Destination CString
start	Starting index
length	Number of characters to extract

Returns

true on success

Note

Automatically clamps to valid range

4.1.3.32 cstr_swap()

```
bool cstr_swap (
            _In_ CString * obj,
            _In_ CString * obj2)
```

Swap contents between two CStrings.

Parameters

obj	First CString
obj2	Second CString

Returns

true on success

4.1.3.33 cstr_to_lower()

```
bool cstr_to_lower (
            _In_ CString * obj)
```

Convert to lowercase.

Parameters

obj	CString object

Returns

true on success

4.1.3.34 cstr_to_upper()

```
bool cstr_to_upper (
            _In_ CString * obj)
```

Convert to uppercase.

Parameters

obj	CString object

Returns

true on success

4.1.3.35 cstr_tokenize()

```
bool cstr_tokenize (
            _In_ CString * obj,
            _Inout_ CString * token,
            _In_ const char * delimiters,
            _Inout_ size_t * start_pos)
```

Extract token using delimiters.

Parameters

obj	Source CString
token	Output token
delimiters	Separator characters
start_pos	Starting/ending position (updated)

Returns

true if token found

4.1.3.36 cstr_tokenize_ex()

```
bool cstr_tokenize_ex (
            _In_ CString * obj,
            _Inout_ CString * token,
            _In_ const char * delimiters,
            _In_ const char * zone_pairs,
            _In_ const char * escape_chars,
            _Inout_ size_t * start_pos)
```

Advanced tokenization with zones/escaping.

Parameters

obj	Source CString
token	Output token
delimiters	Separator characters
zone_pairs	Zone delimiter pairs (e.g., "\"\"")
escape_chars	Escape characters
start_pos	Starting/ending position (updated)

Returns

true if token found

```
size_t pos = 0;
CString str, token;
cstr_create_from_chars(&str, "Hello, \"my world\"!");
while (cstr_tokenize_ex(&str, &token, " ", "\"\"", "\\", &pos))
    printf("Token: %s\n", cstr_data(&token));
```

4.1.3.37 cstr_trim()

```
bool cstr_trim (
            _In_ CString * obj)
```

Trim whitespace from both ends.

Parameters

obj	CString object

Returns

true if modified, false otherwise

4.1.3.38 cstr_unlock()

```
void cstr_unlock (
            _In_ CString * obj)
```

Release exclusive access.

Parameters

obj	CString object

4.1.3.39 cstr_wcsdup()

```
wchar_t * cstr_wcsdup (
            _In_ const wchar_t * str)
```

Duplicate null-terminated wide string.

Parameters

str	Source wide string to copy

Returns

New allocated copy on success, NULL on failure

Note

Wide char version of cstr_strdup()

Warning

Caller must free result with free()

4.1.4 Variable Documentation

4.1.4.1 invalid

```
const size_t invalid = (size_t)-1  [static]
```

4.2 cstr.h

Go to the documentation of this file.

```
#pragma once


#ifndef CSTR_H
#define CSTR_H

#include <Windows.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>

#ifdef __cplusplus
extern "C"
{
#endif

    static const size_t invalid = (size_t)-1;

    typedef struct
    {
        char* data;
        size_t length;
        size_t capacity;
        CRITICAL_SECTION cs;
    }CString;

    char* cstr_strdup(_In_ const char* str)
    {
        size_t len = strlen(str) + 1;
        char* buf = (char*)malloc(len);
        if (buf)
            memcpy(buf, str, len);
        return buf;
    }

    wchar_t* cstr_wcsdup(_In_ const wchar_t* str)
    {
        size_t len = (wcslen(str) + 1) * sizeof(wchar_t);
        wchar_t* buf = (wchar_t*)malloc(len);
        if (buf)
            memcpy(buf, str, len);
        return buf;
    }

    bool cstr_create(_Inout_ CString* obj)
    {
        if (!obj)
            return false;

        char* data = (char*)malloc(1);
        if (data == NULL)
            return false;

        data[0] = '\0';

        obj->data = data;
        obj->length = 0;
        obj->capacity = 1;

        InitializeCriticalSection(&obj->cs);

        return true;
    }

    bool cstr_create_from_cstr(_Inout_ CString* obj, _In_ CString* obj2)
    {
        if (!obj || !obj2)
            return false;
```

```c

    obj->data = cstr_strdup(obj2->data);
    obj->length = obj2->length;
    obj->capacity = obj2->capacity;

    InitializeCriticalSection(&obj->cs);

    return true;
}

bool cstr_create_from_chars(_Inout_ CString* obj, _In_ const char* data)
{
    if (!obj || !data)
        return false;

    obj->data = cstr_strdup(data);
    obj->length = strlen(data);
    obj->capacity = obj->length + 1;

    InitializeCriticalSection(&obj->cs);

    return true;
}

bool cstr_create_from_wchars(_Inout_ CString* obj, _In_ const wchar_t* data)
{
    if (!obj || !data)
        return false;

    int len = WideCharToMultiByte(CP_ACP, 0, data, -1, NULL, 0, NULL, NULL);
    if (len == 0)
        return false;

    char* mb_data = (char*)malloc(len);
    if (!mb_data)
        return false;

    if (!WideCharToMultiByte(CP_ACP, 0, data, -1, mb_data, len, NULL, NULL))
    {
        free(mb_data);
        return false;
    }

    obj->data = mb_data;
    obj->length = strlen(mb_data);
    obj->capacity = len;

    InitializeCriticalSection(&obj->cs);

    return true;
}

bool cstr_create_from_buffer(_Inout_ CString* obj, _In_ uint8_t* buffer, _In_ size_t size)
{
    if (!obj || !buffer)
        return false;

    char* data = (char*)malloc(size + 1);
    if (data == NULL)
        return false;

    memcpy(data, (void*)buffer, size);
    data[size] = '\0';

    obj->data = data;
    obj->length = size;
    obj->capacity = size + 1;

    InitializeCriticalSection(&obj->cs);

    return true;
}

bool cstr_destroy(_In_ CString* obj)
{
    if (!obj)
        return false;

    if (obj->data)
    {
        SecureZeroMemory(obj->data, obj->capacity);
        free(obj->data);
        obj->data = NULL;
    }

    DeleteCriticalSection(&obj->cs);

    obj->length = 0;
    obj->capacity = 0;

    return true;
}

void cstr_lock(_In_ CString* obj)
{
    if (obj)
        EnterCriticalSection(&obj->cs);
}

void cstr_unlock(_In_ CString* obj)
{
    if (obj)
        LeaveCriticalSection(&obj->cs);
}

boolean cstr_at(_In_ CString* obj, _In_ size_t index, _Inout_ char* chr)
{
    if (!obj)
        return false;

    cstr_lock(obj);

    if (index >= obj->length)
    {
        cstr_unlock(obj);
        return false;
    }

    *chr = obj->data[index];

    cstr_unlock(obj);

    return true;
}

char cstr_get(_In_ CString* obj, _In_ size_t index)
{
    if (!obj)
        return 0;

    cstr_lock(obj);

    char out = obj->data[index];

    cstr_unlock(obj);

    return out;
}

char cstr_front(_In_ CString* obj)
{
    if (!obj)
        return 0;

    cstr_lock(obj);

    char out = cstr_get(obj, 0);

    cstr_unlock(obj);

    return out;
}

char cstr_back(_In_ CString* obj)
{
    if (!obj)
        return 0;

    cstr_lock(obj);

    char out = cstr_get(obj, obj->length - 1);

    cstr_unlock(obj);

    return out;
}

char* cstr_data(_In_ CString* obj)
{
    if (!obj)
        return 0;

    cstr_lock(obj);

    char* out = obj->data;

    cstr_unlock(obj);

    return out;
}

size_t cstr_length(_In_ CString* obj)
{
    if (!obj)
        return invalid;

    cstr_lock(obj);

    size_t out = obj->length;

    cstr_unlock(obj);

    return out;
}

size_t cstr_capacity(_In_ CString* obj)
{
    if (!obj)
        return invalid;

    cstr_lock(obj);

    size_t out = obj->capacity;

    cstr_unlock(obj);

    return out;
}

bool cstr_empty(_In_ CString* obj)
{
    if (!obj)
        return false;

    cstr_lock(obj);

    bool out = obj->data == NULL || obj->length == 0;

    cstr_unlock(obj);

    return out;
}

bool cstr_resize(_In_ CString* obj, _In_ size_t size)
{
    if (!obj)
        return false;

    cstr_lock(obj);

    char* new_data = (char*)realloc(obj->data, size);
    if (new_data == NULL)
    {
        cstr_unlock(obj);
        return false;
    }

    obj->data = new_data;
    obj->capacity = size;

    cstr_unlock(obj);

    return true;
}

bool cstr_shrink_to_fit(_In_ CString* obj)
{
    if (!obj)
        return false;

    cstr_lock(obj);

    if (!cstr_resize(obj, obj->length + 1))
    {
        cstr_unlock(obj);
        return false;
    }

    cstr_unlock(obj);

    return true;
```

```
}

bool cstr_clear(_In_ CString* obj)
{
    if (!obj)
        return false;

    cstr_lock(obj);

    SecureZeroMemory(obj->data, obj->capacity);
    obj->length = 0;

    cstr_unlock(obj);

    return true;
}

bool cstr_push_back_char(_In_ CString* obj, _In_ char chr)
{
    if (!obj)
        return false;

    cstr_lock(obj);

    if (obj->length + 1 >= obj->capacity)
    {
        if (!cstr_resize(obj, obj->length + 2))
        {
            cstr_unlock(obj);
            return false;
        }
    }

    obj->data[obj->length] = chr;
    obj->data[obj->length + 1] = '\0';
    obj->length++;

    cstr_unlock(obj);

    return true;
}

bool cstr_push_back_wchar(_In_ CString* obj, _In_ wchar_t chr)
{
    if (!obj)
        return false;

    cstr_lock(obj);

    wchar_t wstr[2] = { chr, L'\0' };
    int required_mb_len = WideCharToMultiByte(CP_ACP, 0, wstr, -1, NULL, 0, NULL, NULL);
    if (required_mb_len <= 0)
    {
        cstr_unlock(obj);
        return false;
    }

    char* mb_str = (char*)malloc(required_mb_len);
    if (!mb_str)
    {
        cstr_unlock(obj);
        return false;
    }

    if (WideCharToMultiByte(CP_ACP, 0, wstr, -1, mb_str, required_mb_len, NULL, NULL) == 0)
    {
        free(mb_str);
        cstr_unlock(obj);
        return false;
    }

    size_t data_len = required_mb_len - 1;

    size_t new_length = obj->length + data_len;
    size_t required_capacity = new_length + 1;

    if (required_capacity > obj->capacity)
    {
        size_t new_capacity = required_capacity;
        char* new_data = (char*)realloc(obj->data, new_capacity);
        if (!new_data)
        {
            free(mb_str);
            cstr_unlock(obj);
            return false;
        }
        obj->data = new_data;
```

```
            obj->capacity = new_capacity;
        }

        memcpy(obj->data + obj->length, mb_str, data_len);
        obj->length = new_length;
        obj->data[new_length] = '\0';

        free(mb_str);
        cstr_unlock(obj);

        return true;
    }

    bool cstr_pop_back(_In_ CString* obj)
    {
        if (!obj)
            return false;

        cstr_lock(obj);

        if (obj->length == 0)
        {
            cstr_unlock(obj);
            return false;
        }

        obj->data[obj->length - 1] = 0;
        obj->length--;

        cstr_unlock(obj);

        return true;
    }

    bool cstr_append_cstr(_In_ CString* obj, _In_ CString* obj2)
    {
        if (!obj || !obj2)
            return false;

        cstr_lock(obj);

        size_t new_length = obj->length + obj2->length;
        size_t required_capacity = new_length + 1;

        if (required_capacity > obj->capacity)
        {
            if (!cstr_resize(obj, required_capacity))
            {
                cstr_unlock(obj);
                return false;
            }
        }

        memcpy(obj->data + obj->length, obj2->data, obj2->length);
        obj->data[new_length] = '\0';
        obj->length = new_length;

        cstr_unlock(obj);

        return true;
    }

    bool cstr_append_chars(_In_ CString* obj, _In_ const char* data)
    {
        if (!obj || !data)
            return false;

        cstr_lock(obj);

        size_t data_len = strlen(data);
        size_t new_length = obj->length + data_len;
        size_t required_capacity = new_length + 1;

        if (required_capacity > obj->capacity)
        {
            if (!cstr_resize(obj, required_capacity))
            {
                cstr_unlock(obj);
                return false;
            }
        }

        memcpy(obj->data + obj->length, data, data_len);
        obj->data[new_length] = '\0';
        obj->length = new_length;

        cstr_unlock(obj);
        return true;
    }

    bool cstr_append_wchars(_In_ CString* obj, _In_ const wchar_t* data)
    {
        if (!obj || !data)
            return false;

        cstr_lock(obj);

        int len = WideCharToMultiByte(CP_ACP, 0, data, -1, NULL, 0, NULL, NULL);
        if (len == 0)
        {
            cstr_unlock(obj);
            return false;
        }

        char* mb_data = (char*)malloc(len);
        if (!mb_data)
        {
            cstr_unlock(obj);
            return false;
        }

        if (WideCharToMultiByte(CP_ACP, 0, data, -1, mb_data, len, NULL, NULL) == 0)
        {
            free(mb_data);
            cstr_unlock(obj);
            return false;
        }

        size_t data_len = strlen(mb_data);
        size_t new_length = obj->length + data_len;
        size_t required_capacity = new_length + 1;

        if (required_capacity > obj->capacity)
        {
            if (!cstr_resize(obj, required_capacity))
            {
                free(mb_data);
                cstr_unlock(obj);
                return false;
            }
        }

        memcpy(obj->data + obj->length, mb_data, data_len);
        obj->data[new_length] = '\0';
        obj->length = new_length;

        free(mb_data);

        cstr_unlock(obj);

        return true;
    }

    bool cstr_substring(_In_ CString* obj, _Inout_ CString* dest, _In_ size_t start, _In_ size_t
length)
    {
        if (!obj || !dest)
            return false;

        cstr_lock(obj);

        if (start >= obj->length)
        {
            cstr_unlock(obj);
            return false;
        }

        size_t max_length = obj->length - start;
        if (length > max_length)
            length = max_length;

        char* buffer = (char*)malloc(length + 1);
        if (!buffer)
        {
            cstr_unlock(obj);
            return false;
        }

        memcpy(buffer, obj->data + start, length);
        buffer[length] = '\0';

        if (!cstr_create_from_chars(dest, buffer))
        {
            free(buffer);
            cstr_unlock(obj);
            return false;
        }

        free(buffer);

        cstr_unlock(obj);

        return true;
    }

    bool cstr_erase(_In_ CString* obj, _In_ size_t index, _In_ size_t size)
    {
        if (!obj)
            return false;

        cstr_lock(obj);

        if (index >= obj->length || size == 0)
        {
            cstr_unlock(obj);
            return false;
        }

        if (size > obj->length - index)
            size = obj->length - index;

        size_t new_length = obj->length - size;
        size_t move_size = (obj->length - (index + size)) + 1;

        memmove(obj->data + index, obj->data + index + size, move_size);
        obj->length = new_length;

        cstr_unlock(obj);

        return true;
    }

    bool cstr_insert(_In_ CString* obj, _In_ size_t index, _In_ char chr)
    {
        if (!obj)
            return false;

        cstr_lock(obj);

        if (index > obj->length)
        {
            cstr_unlock(obj);
            return false;
        }

        size_t new_length = obj->length + 1;
        size_t required_capacity = new_length + 1;

        if (required_capacity > obj->capacity)
        {
            if (!cstr_resize(obj, required_capacity))
            {
                cstr_unlock(obj);
                return false;
            }
        }


        memmove(obj->data + index + 1, obj->data + index, (obj->length - index) + 1);
        obj->data[index] = chr;
        obj->length = new_length;

        cstr_unlock(obj);

        return true;
    }

    bool cstr_swap(_In_ CString* obj, _In_ CString* obj2)
    {
        if (!obj || !obj2)
            return false;

        cstr_lock(obj);
        cstr_lock(obj2);

        char* temp_data = obj->data;
        obj->data = obj2->data;
        obj2->data = temp_data;

        size_t temp_length = obj->length;
        obj->length = obj2->length;
        obj2->length = temp_length;

        size_t temp_capacity = obj->capacity;
        obj->capacity = obj2->capacity;
        obj2->capacity = temp_capacity;

        cstr_unlock(obj2);
        cstr_unlock(obj);

        return true;
    }

    size_t cstr_find_cstr(_In_ CString* obj, _In_ CString* obj2)
    {
        if (!obj || !obj2)
            return invalid;

        cstr_lock(obj);
        cstr_lock(obj2);

        char* pos = strstr(obj->data, obj2->data);
        size_t out = (pos != NULL) ? (size_t)(pos - obj->data) : invalid;

        cstr_unlock(obj2);
        cstr_unlock(obj);

        return out;
    }

    size_t cstr_find_chars(_In_ CString* obj, _In_ const char* data)
    {
        if (!obj || !data)
            return invalid;

        cstr_lock(obj);

        char* pos = strstr(obj->data, data);
        size_t out = (pos != NULL) ? (size_t)(pos - obj->data) : invalid;

        cstr_unlock(obj);

        return out;
    }

    size_t cstr_find_wchars(_In_ CString* obj, _In_ const wchar_t* data)
    {
        if (!obj || !data)
            return invalid;

        int len = WideCharToMultiByte(CP_ACP, 0, data, -1, NULL, 0, NULL, NULL);
        if (len == 0)
            return invalid;

        char* mb_data = (char*)malloc(len);
        if (!mb_data)
            return invalid;

        if (WideCharToMultiByte(CP_ACP, 0, data, -1, mb_data, len, NULL, NULL) == 0)
        {
            free(mb_data);
            return invalid;
        }

        cstr_lock(obj);

        char* pos = strstr(obj->data, mb_data);
        size_t result = (pos != NULL) ? (size_t)(pos - obj->data) : invalid;

        cstr_unlock(obj);

        free(mb_data);

        return result;
    }

    bool cstr_to_upper(_In_ CString* obj)
    {
        if (!obj)
            return false;

        cstr_lock(obj);

        for (size_t i = 0; i < obj->length; ++i)
            obj->data[i] = (char)toupper((unsigned char)obj->data[i]);

        cstr_unlock(obj);
        return true;
    }

    bool cstr_to_lower(_In_ CString* obj)
    {
        if (!obj)
            return false;

        cstr_lock(obj);

        for (size_t i = 0; i < obj->length; ++i)
            obj->data[i] = (char)tolower((unsigned char)obj->data[i]);

        cstr_unlock(obj);

        return true;
    }

    bool cstr_trim(_In_ CString* obj)
    {
        if (!obj)
            return false;

        cstr_lock(obj);

        if (obj->length == 0)
        {
            cstr_unlock(obj);
            return false;
        }

        size_t start = 0;
        size_t end = obj->length - 1;

        while (start <= end && isspace((unsigned char)obj->data[start]))
            start++;

        while (end >= start && isspace((unsigned char)obj->data[end]))
            end--;

        size_t new_length = (start <= end) ? (end - start + 1) : 0;

        if (start > 0)
            memmove(obj->data, obj->data + start, new_length);

        obj->data[new_length] = '\0';
        obj->length = new_length;

        cstr_unlock(obj);

        return true;
    }

    bool cstr_tokenize(_In_ CString* obj, _Inout_ CString* token, _In_ const char* delimiters, _Inout_
size_t* start_pos)
    {
        if (!obj || !delimiters || !start_pos || *start_pos >= obj->length)
            return false;

        cstr_lock(obj);

        size_t len = obj->length;
        size_t pos = *start_pos;

        while (pos < len && strchr(delimiters, obj->data[pos]) != NULL)
            pos++;

        if (pos >= len)
        {
            *start_pos = pos;
            cstr_unlock(obj);
            return false;
        }

        size_t token_start = pos;

        while (pos < len && strchr(delimiters, obj->data[pos]) == NULL)
            pos++;

        size_t token_end = pos;

        size_t token_len = token_end - token_start;
        char* temp = (char*)malloc(token_len + 1);
        if (!temp)
        {
            cstr_unlock(obj);
            return false;
        }

        memcpy(temp, obj->data + token_start, token_len);
        temp[token_len] = '\0';

        if (!cstr_create_from_chars(token, temp))
        {
            free(temp);
            cstr_unlock(obj);
            return false;
        }

        free(temp);

        *start_pos = (token_end < len) ? token_end + 1 : len;

        cstr_unlock(obj);

        return true;
    }

    bool cstr_tokenize_ex(_In_ CString* obj, _Inout_ CString* token, _In_ const char* delimiters, _In_
const char* zone_pairs, _In_ const char* escape_chars, _Inout_ size_t* start_pos)
    {
        if (!obj || !delimiters || !start_pos || *start_pos >= obj->length)
            return false;

        cstr_lock(obj);

        size_t len = obj->length;
        size_t pos = *start_pos;

        while (pos < len && strchr(delimiters, obj->data[pos]) != NULL)
            pos++;

        if (pos >= len)
        {
            *start_pos = pos;
            cstr_unlock(obj);
            return false;
        }

        size_t token_start = pos;
        size_t token_end = invalid;
        bool in_zone = false;
        char zone_end = '\0';
        bool escape = false;

        for (; pos < len; pos++)
        {
            char c = obj->data[pos];

            if (escape)
            {
                escape = false;
                continue;
            }

            if (in_zone)
            {
                if (c == zone_end)
                {
                    in_zone = false;
                    zone_end = '\0';
                }
            }
            else
            {
                if (strchr(delimiters, c) != NULL)
                {
                    token_end = pos;
                    break;
                }

                if (zone_pairs)
                {
                    for (int z = 0; zone_pairs[z] != '\0'; z += 2)
                    {
                        if (zone_pairs[z + 1] == '\0')
                            break;
                        if (c == zone_pairs[z])
                        {
                            in_zone = true;
                            zone_end = zone_pairs[z + 1];
                            break;
                        }
                    }
                }

                if (escape_chars && strchr(escape_chars, c) != NULL)
                    escape = true;
            }
        }

        token_end = (pos == len) ? len : token_end;

        size_t token_len = token_end - token_start;
        char* temp = (char*)malloc(token_len + 1);
        if (!temp)
        {
            cstr_unlock(obj);
            return false;
        }

        memcpy(temp, obj->data + token_start, token_len);
        temp[token_len] = '\0';

        if (!cstr_create_from_chars(token, temp))
        {
            free(temp);
            cstr_unlock(obj);
            return false;
        }

        free(temp);
        *start_pos = (token_end < len) ? token_end + 1 : len;

        cstr_unlock(obj);

        return true;
    }

#ifdef __cplusplus
}
#endif

#endif // CSTR_H
```

Index