Mini Project

Author: ThanhTh10

Date: 16/08/2024

Mylib.h

#pragma once

#include<iostream>

using namespace std;

class MyString {

    char\* str;

    int length;

public:

    int stringLength(const char\* s) const;

    /\* Constructors \*/

    MyString();

    MyString(const char\* s);

    MyString(const MyString& other);

    ~MyString();

    // Member functions for string operations

    MyString& operator=(const MyString& other);

    MyString operator+(const MyString& other);

    char& operator[](int index);

    friend std::ostream& operator<<(std::ostream& os, const MyString& s);

    friend std::istream& operator>>(std::istream& is, MyString& s);

    bool operator==(const MyString& other) const;

    bool operator!=(const MyString& other) const;

    bool operator<(const MyString& other) const;

    bool operator>(const MyString& other) const;

    bool operator<=(const MyString& other) const;

    bool operator>=(const MyString& other) const;

    // Friend functions for string operations

    friend MyString strcpy(MyString& dest, const MyString& src);

    friend MyString strncpy(MyString& dest, const MyString& src, int n);

    friend int strcmp(const MyString& s1, const MyString& s2);

    friend int strncmp(const MyString& s1, const MyString& s2, int n);

    friend MyString strcat(MyString& dest, const MyString& src);

    friend MyString strncat(MyString& dest, const MyString& src, int n);

    friend MyString strrev(MyString& s);

    friend MyString strupr(MyString& s);

    friend MyString strlwr(MyString& s);

    friend const char\* strchr(const MyString& s, char ch);

    friend const char\* strrchr(const MyString& s, char ch);

    friend const char\* strstr(const MyString& s1, const MyString& s2);

    friend int strlen(const MyString& s);

};

Constructors.cpp

#include"Mylib.h"

int MyString::stringLength(const char\* s) const

{

    int len = 0;

    while (s[len] != '\0')

    {

        ++len;

    }

    return len;

}

//Defaut constructor

MyString::MyString() : str(nullptr), length(0) {}

// Parameterized constructor

MyString::MyString(const char\* s)

{

    length = stringLength(s);

    str = new char[length + 1];

    for (int i = 0; i < length; ++i)

    {

        str[i] = s[i];

    }

    str[length] = '\0';

}

// Copy constructor

MyString::MyString(const MyString& other)

{

    length = other.length;

    str = new char[length + 1];

    for (int i = 0; i < length; ++i)

    {

        str[i] = other.str[i];

    }

    str[length] = '\0';

}

// Destructor

MyString::~MyString()

{

    delete[] str;

}

MemberFunctions.cpp

#include"Mylib.h"

// Member function

MyString& MyString::operator=(const MyString& other) {

    if (this == &other)

    {

        return \*this;

    }

    delete[]str;

    length = other.length;

    str = new char[length + 1];

    for (int i = 0; i < length; ++i)

    {

        str[i] = other.str[i];

    }

    str[length] = '\0';

    return \*this;

}

MyString MyString::operator+(const MyString& other) {

    MyString newStr;

    newStr.length = length + other.length;

    newStr.str = new char[newStr.length + 1];

    for (int i = 0; i < length; ++i)

    {

        newStr.str[i] = str[i];

    }

    for (int i = 0; i < other.length; ++i)

    {

        newStr.str[length + i] = other.str[i];

    }

    newStr.str[newStr.length] = '\0';

    return newStr;

}

char& MyString::operator[](int index)

{

    if (index >= 0 && index < length)

    {

        return str[index];

    }

    throw out\_of\_range("Index out of bounds");

}

std::ostream& operator<<(std::ostream& os, const MyString& s)

{

    return os << s.str;

}

std::istream& operator>>(std::istream& is, MyString& s)

{

    char buffer[1000];

    is >> buffer;

    s.length = s.stringLength(buffer);

    s.str = new char[s.length + 1];

    for (int i = 0; i < s.length; ++i)

    {

        s.str[i] = buffer[i];

    }

    s.str[s.length] = '\0';

    return is;

}

bool MyString::operator==(const MyString& other) const

{

    if (length != other.length)

    {

        return false;

    }for (int i = 0; i < length; ++i)

    {

        if (str[i] != other.str[i])

        {

            return false;

        }

    }

    return true;

}

bool MyString::operator!=(const MyString& other) const

{

    return !(\*this == other);

}

bool MyString::operator<(const MyString& other) const

{

    int minLength = (length < other.length) ? length : other.length;

    for (int i = 0; i < minLength; ++i)

    {

        if (str[i] < other.str[i])

        {

            return true;

        }if (str[i] > other.str[i])

        {

            return false;

        }

        return length < other.length;

    }

    return true;

}

bool MyString::operator>(const MyString& other) const

{

    return other < \*this;

}

bool MyString::operator<=(const MyString& other) const

{

    return !(\*this > other);

}

bool MyString::operator>=(const MyString& other) const

{

    return !(\*this < other);

}

FriendFunctions.cpp

#include"Mylib.h"

/\* Friend function \*/

//copy str

MyString strcpy(MyString& dest, const MyString& src)

{

    delete[] dest.str;

    dest.length = src.length;

    dest.str = new char[dest.length + 1];

    for (int i = 0; i < dest.length; ++i)

    {

        dest.str[i] = src.str[i];

    }

    dest.str[dest.length] = '\0';

    return dest;

}

//copy n char

MyString strncpy(MyString& dest, const MyString& src, int n)

{

    delete[] dest.str;

    dest.length = src.length;

    dest.str = new char[dest.length + 1];

    for (int i = 0; i < n; ++i)

    {

        dest.str[i] = src.str[i];

    }

    dest.str[n] = '\0';

    return dest;

}

// compare 2 str

int strcmp(const MyString& s1, const MyString& s2)

{

    int strlen1 = s1.length, strlen2 = s2.length;

    int min\_length = strlen1 < strlen2 ? strlen1 : strlen2;

    for (int i = 0; i < min\_length; i++)

    {

        if (s1.str[i] != s2.str[i])

        {

            if (s1.str[i] > s2.str[i])

                return 1;

            else

                return -1;

        }

    }

    if (strlen1 < strlen2)

        return -1;

    else if (strlen1 > strlen2)

        return 1;

    else

        return 0;

}

// compare 2 str with n char

int strncmp(const MyString& s1, const MyString& s2, int n)

{

    int len1 = s1.length;

    int len2 = s2.length;

    int min\_length = n < len1 ? n : len1;

    min\_length = min\_length < len2 ? min\_length : len2;

    for (int i = 0; i < min\_length; i++)

    {

        if (s1.str[i] != s2.str[i])

        {

            return (s1.str[i] > s2.str[i]) ? 1 : -1;

        }

    }

    if (min\_length < n)

    {

        if (len1 < len2)

            return -1;

        else if (len1 > len2)

            return 1;

    }

    return 0;

}

// concat

MyString strcat(MyString& dest, const MyString& src)

{

    char\* newStr = new char[dest.length + src.length + 1];

    for (int i = 0; i < dest.length; ++i)

    {

        newStr[i] = dest.str[i];

    }

    for (int i = 0; i < src.length; ++i)

    {

        newStr[dest.length + i] = src.str[i];

    }

    newStr[dest.length + src.length] = '\0';

    delete[] dest.str;

    dest.str = newStr;

    dest.length += src.length;

    return dest;

}

//concat n char

MyString strncat(MyString& dest, const MyString& src, int n)

{

    char\* newStr = new char[dest.length + n + 1];

    for (int i = 0; i < dest.length; ++i)

    {

        newStr[i] = dest.str[i];

    }

    for (int i = 0; i < n; ++i)

    {

        newStr[dest.length + i] = src.str[i];

    }

    newStr[dest.length + n] = '\0';

    delete[] dest.str;

    dest.str = newStr;

    dest.length += n;

    return dest;

}

//reverse str

MyString strrev(MyString& s)

{

    for (int i = 0; i < s.length / 2; ++i)

    {

        char temp = s.str[i];

        s.str[i] = s.str[s.length - i - 1];

        s.str[s.length - i - 1] = temp;

    }

    return s;

}

// uppercase

MyString strupr(MyString& s)

{

    for (int i = 0; i < s.length; i++)

    {

        if (s.str[i] >= 'a' && s.str[i] <= 'z')

        {

            s.str[i] = s.str[i] - 'a' + 'A';

        }

    }

    return s;

}

//lowercase

MyString strlwr(MyString& s)

{

    for (int i = 0; i < s.length; i++)

    {

        if (s.str[i] >= 'A' && s.str[i] <= 'Z')

        {

            s.str[i] = s.str[i] - 'A' + 'a';

        }

    }

    return s;

}

//print str from the first char founded to end

const char\* strchr(const MyString& s, char ch)

{

    for (int i = 0; i < s.length; i++)

    {

        if (s.str[i] == ch)

        {

            return &s.str[i];

        }

    }

    return nullptr;

}

//same but from the last char founded to end

const char\* strrchr(const MyString& s, char ch)

{

    for (int i = s.length - 1; i >= 0; i--)

    {

        if (s.str[i] == ch)

        {

            return &s.str[i];

        }

    }

    return nullptr;

}

//same to strchr but accept as string

const char\* strstr(const MyString& s1, const MyString& s2)

{

    for (int i = 0; i < s1.length - s2.length; i++)

    {

        int j = 0;

        while (j < s2.length && s1.str[i+j] ==s2.str[j])

        {

            ++j;

        }

        if (j == s2.length) {

            return &s1.str[i];

        }

    }

    return nullptr;

}

//length of str

int strlen(const MyString& s)

{

    return s.length;

}

Main.cpp

#include"Mylib.h"

int main() {

    MyString s1("Hello, ");

    MyString s2("World.");

cout << strcat(s1, s2) << endl;

//… all functions available

    return 0;

}