实验报告

东北林业大学

信息与计算机科学技术实验中心

|  |
| --- |
| 1. 实验目的   本课程设计的目标是创建一个 MyString 类，以实现基本的字符串操作功能。这包括构造函数、赋值操作符、字符串长度计算、字符串连接、字符串比较、子串提取、字符串交换、字符串查找、子串替换、子串删除、字符串插入以及流操作符重载。通过完成这些功能，学生能够理解并掌握 C++ 中类和对象的使用、动态内存管理、操作符重载以及异常处理。 |
| 二、实验环境  Codeblocks |
| 三、实验内容及结果  **1.设计思路**  作为一个c++类，我们可以将其分为十一个部分，分别是构造函数、对MyString对象赋值 、求字符串的长度 、MyString对象中字符串的连接 、MyString对象的比较 、求MyString对象的子串 、查找子串和字符 、替换子串 、 删除子串 、插入字符串 、输出流。  分别对应了构造，赋值，返回长度，连接，求子串，比较，查找，删除等。对于一个完备的类来说，每个部分我们需要不只一个函数，我们可以通过测试集来补充我们的函数，在有测试集的条件下，我们可以先写出类中共有函数的声明，方便我们后续阅读。  **2.函数声明**  过程中，我们将对每个部分进行声明  //构造函数  MyString();  MyString(const char\* str);  MyString(int n, char c);  MyString(const char\* str, int pos, int len);  MyString(const MyString& other);  MyString(MyString&& other) noexcept;  ~MyString(); //对MyString对象赋值  MyString& operator=(const MyString& other);  MyString& operator=(MyString&& other) noexcept;  MyString& operator=(const char\* str);  MyString& operator=(char c);  MyString& assign(const MyString& other);  MyString& assign(const MyString& other, int pos, int len);  MyString& assign(int n, char c);  MyString& assign(const char\* str, int pos, int len); //求字符串的长度  int length() const;  int size() const; //MyString对象中字符串的连接  MyString& append(const MyString& other);  MyString& append(const MyString& other, int pos, int len);  MyString& append(int n, char c);  MyString& append(const char\* str, int pos, int len); //MyString对象的比较  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; //求MyString对象的子串  MyString substr(int pos , int len) const;  MyString substr(int pos ) const;  //交换两个MyString对象的内容  void swap(MyString& other); //查找子串和字符  int find(char c, int pos = 0) const;  int find(const char\* str, int pos = 0) const;  int find(const MyString& other, int pos = 0) const;  int find\_first\_of(const char\* str, int pos = 0) const;  int find\_last\_of(char c) const;  int find\_first\_not\_of(const char\* str, int pos = 0) const; //替换子串  MyString& replace(size\_t pos, size\_t len, const char\* str, size\_t str\_pos, size\_t str\_len);  MyString& replace(size\_t pos, size\_t len, size\_t n, char c);  MyString& replace(size\_t pos, size\_t len, const char\* str) ; // 删除子串  MyString& erase(int pos , int len );  MyString& erase(int pos ); //插入字符串  MyString& insert(int pos, const char\* str);  MyString& insert(int pos, const MyString& other);  MyString& insert(int pos, int n, char c); //输出流  friend ostream& operator<<(ostream& os, const MyString& str);  接下来将对每个部分的函数进行补充  **3. 1. 1构造函数**  默认构造函数 MyString::MyString(): 构造一个空字符串对象。在函数体内，将 data 指针初始化为 nullptr，Length 初始化为 0。  带字符数组参数的构造函数 MyString::MyString(const char\* str): 根据传入的 C 风格字符串构造一个字符串对象。首先计算字符串长度，然后分配足够的内存，将字符串内容复制到 data 指向的内存中。  带长度和字符参数的构造函数 MyString::MyString(int n, char c): 构造一个由 n 个字符 c 组成的字符串对象。首先设置字符串长度为 n，然后分配足够的内存，并使用 memset 将字符 c 填充到 data 指向的内存中。  带位置、长度和字符数组参数的构造函数 MyString::MyString(const char\* str, int pos, int len): 构造一个由指定位置和长度的字符数组组成的字符串对象。首先计算要复制的子串的长度，然后分配足够的内存，使用 strncpy 将指定位置开始的子串复制到 data 指向的内存中。  拷贝构造函数 MyString::MyString(const MyString& other): 根据另一个字符串对象构造一个新的字符串对象。首先获取另一个对象的长度，然后分配足够的内存，将另一个对象的内容复制到新对象中。  移动构造函数 MyString::MyString(MyString&& other) noexcept: 使用移动语义构造一个新的字符串对象。首先将其他对象的数据成员移动到新对象中，然后将其他对象的数据指针设为空指针，避免释放内存。  **3.1.2部分函数**  MyString::MyString(const char\* str, int pos, int len) {  Length = len;  data = new char[Length + 1];  strncpy(data, str + pos, len);  data[Length] = '\0'; }  **3.1.3测试集**  { //测试1  MyString s1; // si = ""  MyString s2("Hello"); // s2 = "Hello"  MyString s3(4, 'K'); // s3 = "KKKK"  MyString s4("12345", 1, 3); //s4 = "234"，即 "12345" 的从下标 1 开始，长度为 3 的子串  cout<<s1<<s2<<s3<<s4 ; }  **3.2.1析构函数**  析构函数 ~MyString()释放字符串对象占用的内存。首先释放 data 指向的内存，然后将 data 设为 nullptr。  **3.2.2部分代码**  MyString::~MyString() {  delete[] data; }  **3.3.1赋值运算符**  MyString& MyString::operator=(const MyString& other)实现对象之间的深拷贝。首先检查是否为自我赋值，然后释放原有内存，为新内容分配足够的内存，复制另一个对象的内容。MyString& MyString::operator=(MyString&& other) noexcept 使用移动语义实现对象之间的赋值。首先释放原有内存，然后将其他对象的数据成员移动到当前对象中，并将其他对象的数据指针设为空指针。MyString& MyString::operator=(const char\* str)将 C 风格字符串赋值给当前对象。首先释放原有内存，然后根据新字符串的长度分配足够的内存，将新字符串的内容复制到当前对象中。MyString& MyString::operator=(char c)将字符赋值给当前对象。首先释放原有内存，然后分配足够的内存以存储该字符，并将字符赋值给当前对象。  **3.3.2部分代码**  bool MyString::operator<(const MyString& other) const {  return strcmp(data, other.data) < 0; }  **3.3.3测试集**  {  MyString s1,s2;  s1 = "Hello"; // s1 = "Hello"  s2 = 'K'; // s2 = "K”  cout<<s1<<s2 ; }  **3.4.1字符串长度计算**  提供获取字符串长度的功能，包括 length() 和 size() 方法。length() 方法返回字符串的长度。MyString s1("12345");  int len = s1.length(); // len = 5; size() 方法返回字符串的长度（与 length() 方法相同）。测试条件：获取字符串长度。MyString s2("12345"); int size = s2.size(); // size = 5  **3.4.2部分函数**  int MyString::length() const {  return Length; }  int MyString::size() const {  return Length; }  **3.4.3数据集**  { //测试3  MyString s1("12345"), s2;  cout << s1.length() << s2.size() ; }  **3.5.1字符串连接**  首先计算新的长度，然后分配足够的内存来存储连接后的字符串。使用 strcpy 函数将当前字符串对象的内容复制到新内存中的合适位置。使用 strcpy 或 strncpy 将另一个字符串对象的内容复制到新内存中的合适位置。释放原有的内存，并将 data 指针指向新分配的内存，更新 Length 成员变量。  提供将两个字符串连接起来的功能，包括连接 MyString 对象、连接 C 风格字符串、连接重复字符。连接 MyString 对象 MyString& append(const MyString& other) 将另一个 MyString 对象连接到当前对象的末尾。s1.append(s2); // s1 = "123abc"  连接部分 MyString 对象 MyString& append(const MyString& other, int pos, int len)将另一个 MyString 对象的部分内容连接到当前对象的末尾。s1.append(s2, 1, 2); // s1 = "123abcbc"连接重复字符 MyString& append(int n, char c)在当前对象末尾添加 n 个字符 c。  连接部分 C 风格字符串 MyString& append(const char\* str, int pos, int len)将 C 风格字符串的部分内容连接到当前对象的末尾。  **3.5.2部分函数**  MyString& MyString::append(const MyString& other) {  return append(other.data, 0, other.Length); }  MyString& MyString::append(const MyString& other, int pos, int len) {  return append(other.data, pos, len); }  MyString& MyString::append(int n, char c) {  char\* new\_data = new char[Length + n + 1];  strcpy(new\_data, data);  memset(new\_data + Length, c, n);  new\_data[Length + n] = '\0';  delete[] data;  data = new\_data;  Length += n;  return \*this; }  MyString& MyString::append(const char\* str, int pos, int len) {  int str\_len = strlen(str);  if (pos > str\_len) return \*this;  int n = min(len, str\_len - pos);  char\* new\_data = new char[Length + n + 1];  strcpy(new\_data, data);  strncpy(new\_data + Length, str + pos, n);  new\_data[Length + n] = '\0';  delete[] data;  data = new\_data;  Length += n;  return \*this; }  **3.5.3数据集**  {//测试4  MyString s1("123"), s2("abc");  s1.append(s2); // s1 = "123abc"  cout<<s1;  s1.append(s2, 1, 2); // s1 = "123abcbc"  cout<<s1;  s1.append(3, 'K'); // s1 = "123abcbcKKK"  cout<<s1;  s1.append("ABCDE", 2, 3); // s1 = "123abcbcKKKCDE"，添加 "ABCDE" 的子串(2, 3)  cout<<s1 ; }  **3.6.1字符串比较**  实现比较运算符，用于比较两个 MyString 对象的内容，包括小于、小于等于、大于、大于等于、等于、不等于。operator<(const MyString& other) const比较两个字符串是否小于。  operator<=(const MyString& other) const比较两个字符串是否小于等于。bool result = s1 <= s2; // result = true  operator>(const MyString& other) const比较两个字符串是否大于。bool result = s1 > s2; // result = false  operator>=(const MyString& other) const比较两个字符串是否大于等于。bool result = s1 >= s2; // result = false  operator==(const MyString& other) const比较两个字符串是否相等。bool result = s1 == s2; // result = false  operator!=(const MyString& other) const比较两个字符串是否不相等。bool result = s1 != s2; // result = true  **3.6.2部分函数**  bool MyString::operator<(const MyString& other) const {  return strcmp(data, other.data) < 0; }  bool MyString::operator<=(const MyString& other) const {  return strcmp(data, other.data) <= 0; }  bool MyString::operator>(const MyString& other) const {  return strcmp(data, other.data) > 0; }  bool MyString::operator>=(const MyString& other) const {  return strcmp(data, other.data) >= 0; }  bool MyString::operator==(const MyString& other) const {  return strcmp(data, other.data) == 0; }  bool MyString::operator!=(const MyString& other) const {  return strcmp(data, other.data) != 0; }  **3.6.3数据集**  { //测试5  MyString s1("hello"), s2("hello, world");  cout<<(s1>s2)<<(s1<s2)<<(s1>=s2)<<(s1<=s2)<<(s1==s2)<<(s1!=s2) ; }  **3.7.1子串提取**  提供从字符串中提取子串的功能，包括截取指定位置的子串。根据参数指定的位置和长度，或者只指定位置，构造一个新的字符串对象，将子串内容复制到新对象中。判断截取位置是否在字符串范围内，如果在则进行截取，否则返回空字符串对象。  MyString substr(int pos, int len) const从字符串中提取从 pos 开始的长度为 len 的子串。MyString s1("0123456789");MyString s2 = s1.substr(2, 3); // s2 = "234"  **3.7.2部分函数**  MyString MyString::substr(int pos, int len) const {  if (pos >= Length) return MyString();  int n = min(len, Length - pos);  MyString s1(data , pos, n);  return s1; } MyString MyString::substr(int pos) const {  if (pos >= Length) return MyString();  int n = Length - pos;  MyString s1(data , pos, n);  return s1; }  **3.7.3数据集**  { //测试6  MyString s1 = "this is ok";  MyString s2 = s1.substr(2, 4); // s2 = "is i"  cout<<s1<<s2;  s2 = s1.substr(2); // s2 = "is is ok"  cout<<s2 ; }  **3.8.1字符串交换**  提供交换两个 MyString 对象内容的功能。使用 std::swap 函数交换两个对象的 data 指针和 Length 成员变量。  void swap(MyString& other)交换两个字符串对象的内容。  MyString s1("123");MyString s2("abc");s1.swap(s2); // s1 = "abc", s2 = "123"  **3.8.2部分函数**  void MyString::swap(MyString& other) {  ::swap(data, other.data);  ::swap(Length, other.Length); }  **3.8.3数据集**  { //测试7  MyString s1("West"), s2("East");  cout<<s1<<s2;  s1.swap(s2); // s1 = "East"，s2 = "West"  cout<<s1<<s2 ; }  **3.9.1字符串查找**  提供查找子串或字符在字符串中出现位置的功能，包括从前往后查找和从后往前查找。采用标准库函数 strchr、strstr 等来实现查找操作，根据函数的不同，有的是从前往后查找，有的是从后往前查找。  int find(const MyString& other, int pos = 0) const从字符串的指定位置开始查找另一个字符串。  int rfind(const MyString& other, int pos = 0) const从字符串的指定位置开始反向查找另一个字符串。  int find(char c, int pos = 0) const从字符串的指定位置开始查找字符。  int rfind(char c, int pos = 0) const从字符串的指定位置开始反向查找字符。  **3.9.2部分函数**  int MyString::find(char c, int pos) const {  if (pos >= Length) return npos;  const char\* result = strchr(data + pos, c);  return result ? result - data : npos; }  int MyString::find(const char\* str, int pos) const {  if (pos >= Length) return npos;  const char\* result = strstr(data + pos, str);  return result ? result - data : npos; }  int MyString::find(const MyString& other, int pos) const {  return find(other.data, pos); }  int MyString::find\_first\_of(const char\* str, int pos) const {  if (pos >= Length) return npos;  for (int i = pos; i < Length; ++i) {  if (strchr(str, data[i])) {  return i;  }  }  return npos; }  **3.9.3数据集**  { //测试8  MyString s1("Source Code");  int n;  if ((n = s1.find('u')) != MyString::npos) //查找 u 出现的位置  cout << "1) " << n << "," << s1.substr(n) ;  //输出 l)2,urce Code  if ((n = s1.find("Source", 3)) == MyString::npos)//从下标3开始查找"Source"，找不到  cout << "2) " << "Not Found" ; //输出 2) Not Found  if ((n = s1.find("Co")) != MyString::npos) //查找子串"Co"。能找到，返回"Co"的位置  cout << "3) " << n << ", " << s1.substr(n) ;  //输出 3) 7, Code  if ((n = s1.find\_first\_of("ceo")) != MyString::npos)//查找第一次出现或 'c'、'e'或'o'的位置  cout << "4) " << n << ", " << s1.substr(n) ;  //输出 4) l, ource Code  if ((n = s1.find\_last\_of('e')) != MyString::npos)//查找最后一个 'e' 的位置  cout << "5) " << n << ", " << s1.substr(n) ;  //输出 5) 10, e  if ((n = s1.find\_first\_not\_of("eou", 1)) != MyString::npos) //从下标1开始查找第一次出现非 'e'、'o' 或 'u' 字符的位置  cout << "6) " << n << ", " << s1.substr(n) ;  //输出 6) 3, rce Code }  **3.10.1子串替换**  提供将子串替换为另一个字符串的功能。根据参数的不同，替换为另一个字符数组的一部分、多个相同字符，或者另一个字符数组。  分配新内存，根据参数替换原字符串中指定位置的内容，然后释放原有内存。  MyString& replace(int pos, int len, const MyString& other)  将从指定位置开始的子串替换为另一个字符串。  MyString s1("0123456789");  s1.replace(2, 3, "abc"); // s1 = "01abc56789"  **3.10.2部分函数**  MyString& MyString::replace(size\_t pos, size\_t len, const char\* str, size\_t str\_pos, size\_t str\_len) {  if (pos >= Length) return \*this; // 检查位置是否在字符串范围内  size\_t end\_pos = pos + len;  size\_t new\_len = Length - len + str\_len;  char\* new\_data = new char[new\_len + 1];  std::memcpy(new\_data, data, pos); // 复制替换位置之前的字符  std::memcpy(new\_data + pos, str + str\_pos, str\_len); // 复制新的子串  std::memcpy(new\_data + pos + str\_len, data + end\_pos, Length - end\_pos); // 复制替换位置之后的字符  new\_data[new\_len] = '\0';  delete[] data; // 释放原有内存  data = new\_data;  Length = new\_len;  return \*this; } MyString& MyString::replace(size\_t pos, size\_t len, size\_t n, char c) {  if (pos >= Length) return \*this; // 检查位置是否在字符串范围内  size\_t end\_pos = pos + len;  size\_t new\_len = Length - len + n;  char\* new\_data = new char[new\_len + 1];  std::memcpy(new\_data, data, pos); // 复制替换位置之前的字符  std::memset(new\_data + pos, c, n); // 填充新字符  std::memcpy(new\_data + pos + n, data + end\_pos, Length - end\_pos); // 复制替换位置之后的字符  new\_data[new\_len] = '\0';  delete[] data; // 释放原有内存  data = new\_data;  Length = new\_len;  return \*this; } MyString& MyString::replace(size\_t pos, size\_t len, const char\* str) {  if (pos >= Length) return \*this; // 检查位置是否在字符串范围内  size\_t str\_len = std::strlen(str);  size\_t end\_pos = pos + len;  if (end\_pos > Length) end\_pos = Length;  size\_t new\_len = Length - (end\_pos - pos) + str\_len;  char\* new\_data = new char[new\_len + 1];  std::memcpy(new\_data, data, pos); // 复制替换位置之前的字符  std::memcpy(new\_data + pos, str, str\_len); // 复制新的子串  std::memcpy(new\_data + pos + str\_len, data + end\_pos, Length - end\_pos); // 复制替换位置之后的字符  new\_data[new\_len] = '\0'; // 添加终止符  delete[] data; // 释放原有内存  data = new\_data;  Length = new\_len;  return \*this; }  **3.10.3数据集**  { //测试9  MyString s1("Real Steel");  cout<<s1;  s1.replace(1, 3, "123456", 2, 4); //用 "123456" 的子串(2,4) 替换 s1 的子串(1,3)  cout << s1 ; //输出 R3456 Steel  MyString s2("Harry Potter");  cout<<s2;  s2.replace(2, 3, 5, '0'); //用 5 个 '0' 替换子串(2,3)  cout << s2 ; //输出 HaOOOOO Potter  int n = s2.find("00000"); //查找子串 "00000" 的位置，n=2  cout<<n; //输出2  s2.replace(n, 5, "XXX"); //将子串(n,5)替换为"XXX"  cout << s2 ; //输出 HaXXX Potter }  **3.11.1子串删除**  提供删除指定位置和长度的子串的功能删除函数用于删除字符串对象中指定位置和长度的子串，或者从指定位置开始到字符串末尾的所有字符。分配新内存，根据参数删除原字符串中指定位置的内容，然后释放原有内存。MyString& erase(int pos, int len)删除从指定位置开始的子串。  **3.11.2部分函数**  MyString& MyString::erase(int pos, int len) {  if (pos > Length) return \*this;  int new\_len = Length - len;  char\* new\_data = new char[new\_len + 1];  strncpy(new\_data, data, pos);  strcpy(new\_data + pos, data + pos + len);  new\_data[new\_len] = '\0';  delete[] data;  data = new\_data;  Length = new\_len;  return \*this; }  MyString& MyString::erase(int pos) {  char\* new\_data = new char[pos + 1];  strncpy(new\_data, data, pos);  new\_data[pos] = '\0';  delete[] data;  data = new\_data;  Length = pos+1;  return \*this; }  **3.11.3数据集**  { //测试10  MyString s1("Real Steel");  cout<<s1;  s1.erase(1, 3); //删除子串(1, 3)，此后 s1 = "R Steel"  cout<<s1;  s1.erase(5); //删除下标5及其后面的所有字符，此后 s1 = "R Ste"  cout<<s1 ; }  **3.12.1字符串插入**  设计思路  提供在字符串的指定位置插入另一个字符串或字符的功能。插入函数用于在字符串对象的指定位置插入字符数组、另一个字符串对象或者多个相同字符。分配新内存，根据参数在指定位置插入内容，然后释放原有内存。  MyString& insert(int pos, const MyString& other)在指定位置插入另一个字符串。  MyString& insert(int pos, const char\* str)在指定位置插入 C 风格字符串。  MyString& insert(int pos, char c)在指定位置插入字符。  **3.12.2部分函数**  MyString& MyString::insert(int pos, const char\* str) {  size\_t str\_len = std::strlen(str);  if (pos > Length) pos = Length;  size\_t new\_len = Length + str\_len;  char\* new\_data = new char[new\_len + 1];  std::memcpy(new\_data, data, pos); // 复制插入位置之前的字符  std::memcpy(new\_data + pos, str, str\_len); // 插入新的子串  std::memcpy(new\_data + pos + str\_len, data + pos, Length - pos); // 复制插入位置之后的字符  new\_data[new\_len] = '\0'; // 添加终止符  delete[] data;  data = new\_data;  Length = new\_len;  return \*this; }  MyString& MyString::insert(int pos, const MyString& str) {  return insert(pos, str.data); }  MyString& MyString::insert(int pos, int n, char c) {  if (pos > Length) pos = Length;  size\_t new\_len = Length + n;  char\* new\_data = new char[new\_len + 1];  std::memcpy(new\_data, data, pos); // 复制插入位置之前的字符  std::memset(new\_data + pos, c, n); // 插入 n 个字符 c  std::memcpy(new\_data + pos + n, data + pos, Length - pos); // 复制插入位置之后的字符  new\_data[new\_len] = '\0'; // 添加终止符  delete[] data;  data = new\_data;  Length = new\_len;  return \*this;  **3.12.3数据集**  { //测试11  MyString s1("Limitless"), s2("00");  cout<<s1<<s2;  s1.insert(2, "123"); //在下标 2 处插入字符串"123"，s1 = "Li123mitless"  cout<<s1;  s1.insert(3, s2); //在下标 2 处插入 s2 , s1 = "Li10023mitless"  cout<<s1;  s1.insert(3, 5, 'X'); //在下标 3 处插入 5 个 'X'，s1 = "Li1XXXXX0023mitless"  cout<<s1; }  **3.13.1流操作符重载**  提供输入输出流操作符的重载，用于方便地读取和打印 MyString 对象的内容。  friend std::ostream& operator<<(std::ostream& os, const MyString& str)输出 MyString 对象的内容到输出流。  friend std::istream& operator>>(std::istream& is, MyString& str)从输入流中读取内容并存储到 MyString 对象。 |

|  |
| --- |
|  |
| 五、指导教师意见  指导教师签字：  年 月 日 |