**1.成员函数非引用方式：**

#include <iostream>

#include <cstring>

#pragma warning(disable:4996)

using namespace std;

class String

{

public:

String() { \_str = new char[1]; \*\_str = '\0'; cout << "Constructor called!" << endl; }

String(const char\* str) { \_str = new char [strlen(str) + 1]; strcpy(\_str, str); cout << "Constructor called!" << endl; }

String(const String& \_InitStr)

{

cout << "Copy constructor called!" << endl;

if (&\_InitStr == this) //防止String s = s;的出现

{

\_str = new char[1]; \*\_str = '\0';

}

else

{

\_str = new char[strlen(\_InitStr.\_str) + 1];

strcpy(\_str, \_InitStr.\_str);

}

}

void Print() const { cout << \_str; }

void Set(const char\* \_newstr);

void Set(const String& \_newStr);

bool operator==(String s) { return strcmp(\_str, s.\_str) == 0; }

bool operator<(String s) { return strcmp(\_str, s.\_str) < 0; }

bool operator>(String s) { return strcmp(\_str, s.\_str) > 0; }

bool operator!=(String s) { return strcmp(\_str, s.\_str); }

bool operator>=(String s) { return strcmp(\_str, s.\_str) >= 0; }

bool operator<=(String s) { return strcmp(\_str, s.\_str) <= 0; }

~String() { delete[] \_str; cout << "Destructor called!" << endl; }

private:

char\* \_str;

};

void String::Set(const char\* \_newstr)

{

delete[] \_str;

\_str = new char[strlen(\_newstr) + 1];

strcpy(\_str, \_newstr);

}

void String::Set(const String& \_newStr)

{

if (&\_newStr == this) return;

delete[] \_str;

\_str = new char[strlen(\_newStr.\_str) + 1];

strcpy(\_str, \_newStr.\_str);

}

int main()

{

String s1("abcd"), s2("abce");

String s3 = s1;

if (s1 > s2) cout << "s1 > s2!" << endl;

else if (s1 < s2) cout << "s1 < s2!" << endl;

else if (s1 == s2) cout << "s1 == s2!" << endl;

if (s2 < s3) cout << "s2 < s3!" << endl;

else if (s2 > s3) cout << "s2 > s3!" << endl;

else if (s2 == s3) cout << "s2 == s3!" << endl;

if (s1 < s3) cout << "s1 < s3!" << endl;

else if (s1 > s3) cout << "s1 > s3!" << endl;

else if (s1 == s3) cout << "s1 == s3!" << endl;

if (s1 >= s2) cout << "s1 >= s2!" << endl;

if (s1 <= s2) cout << "s1 <= s2!" << endl;

if (s2 <= s3) cout << "s2 <= s3!" << endl;

if (s2 >= s3) cout << "s2 >= s3!" << endl;

if (s1 <= s3) cout << "s1 <= s3!" << endl;

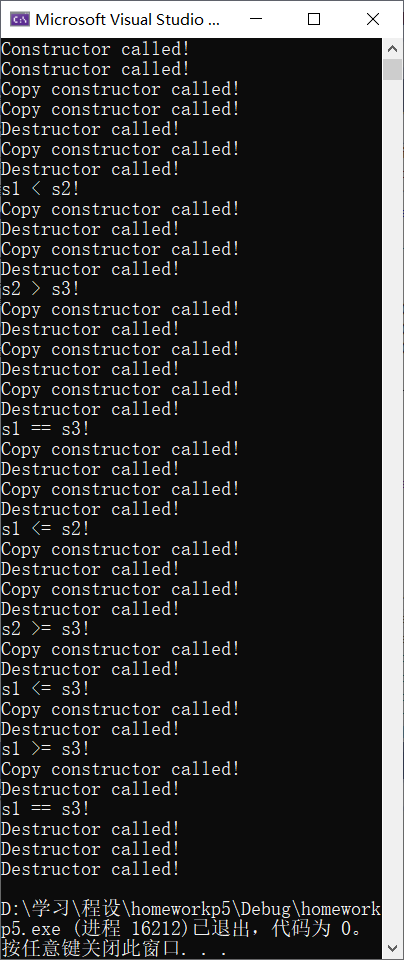
if (s1 >= s3) cout << "s1 >= s3!" << endl;

if (s1 == s3) cout << "s1 == s3!" << endl;

return 0;

}

**运行结果：**



**2.友元函数非引用方式：**

#include <iostream>

#include <cstring>

#pragma warning(disable:4996)

using namespace std;

class String

{

public:

String() { \_str = new char[1]; \*\_str = '\0'; cout << "Constructor called!" << endl; }

String(const char\* str) { \_str = new char[strlen(str) + 1]; strcpy(\_str, str); cout << "Constructor called!" << endl; }

String(const String& \_InitStr)

{

cout << "Copy constructor called!" << endl;

if (&\_InitStr == this) //防止String s = s;的出现

{

\_str = new char[1]; \*\_str = '\0';

}

else

{

\_str = new char[strlen(\_InitStr.\_str) + 1];

strcpy(\_str, \_InitStr.\_str);

}

}

void Print() const { cout << \_str; }

void Set(const char\* \_newstr);

void Set(const String& \_newStr);

friend bool operator==(String s1, String s2);

friend bool operator<(String s1, String s2);

friend bool operator>(String s1, String s2);

friend bool operator!=(String s1, String s2);

friend bool operator>=(String s1, String s2);

friend bool operator<=(String s1, String s2);

~String() { delete[] \_str; cout << "Destructor called!" << endl; }

private:

char\* \_str;

};

void String::Set(const char\* \_newstr)

{

delete[] \_str;

\_str = new char[strlen(\_newstr) + 1];

strcpy(\_str, \_newstr);

}

void String::Set(const String& \_newStr)

{

if (&\_newStr == this) return;

delete[] \_str;

\_str = new char[strlen(\_newStr.\_str) + 1];

strcpy(\_str, \_newStr.\_str);

}

bool operator==(String s1, String s2) { return strcmp(s1.\_str, s2.\_str) == 0; }

bool operator<(String s1, String s2) { return strcmp(s1.\_str, s2.\_str) < 0; }

bool operator>(String s1, String s2) { return strcmp(s1.\_str, s2.\_str) > 0; }

bool operator!=(String s1, String s2) { return strcmp(s1.\_str, s2.\_str); }

bool operator>=(String s1, String s2) { return strcmp(s1.\_str, s2.\_str) >= 0; }

bool operator<=(String s1, String s2) { return strcmp(s1.\_str, s2.\_str) <= 0; }

int main()

{

String s1("abcd"), s2("abce");

String s3 = s1;

if (s1 > s2) cout << "s1 > s2!" << endl;

else if (s1 < s2) cout << "s1 < s2!" << endl;

else if (s1 == s2) cout << "s1 == s2!" << endl;

if (s2 < s3) cout << "s2 < s3!" << endl;

else if (s2 > s3) cout << "s2 > s3!" << endl;

else if (s2 == s3) cout << "s2 == s3!" << endl;

if (s1 < s3) cout << "s1 < s3!" << endl;

else if (s1 > s3) cout << "s1 > s3!" << endl;

else if (s1 == s3) cout << "s1 == s3!" << endl;

if (s1 >= s2) cout << "s1 >= s2!" << endl;

if (s1 <= s2) cout << "s1 <= s2!" << endl;

if (s2 <= s3) cout << "s2 <= s3!" << endl;

if (s2 >= s3) cout << "s2 >= s3!" << endl;

if (s1 <= s3) cout << "s1 <= s3!" << endl;

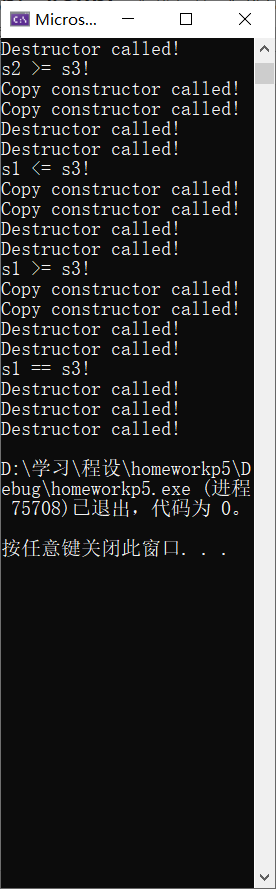
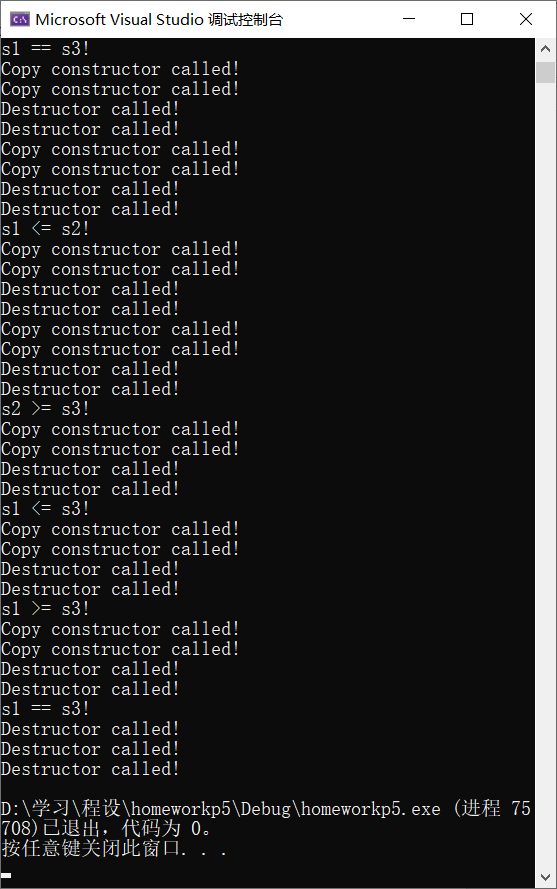
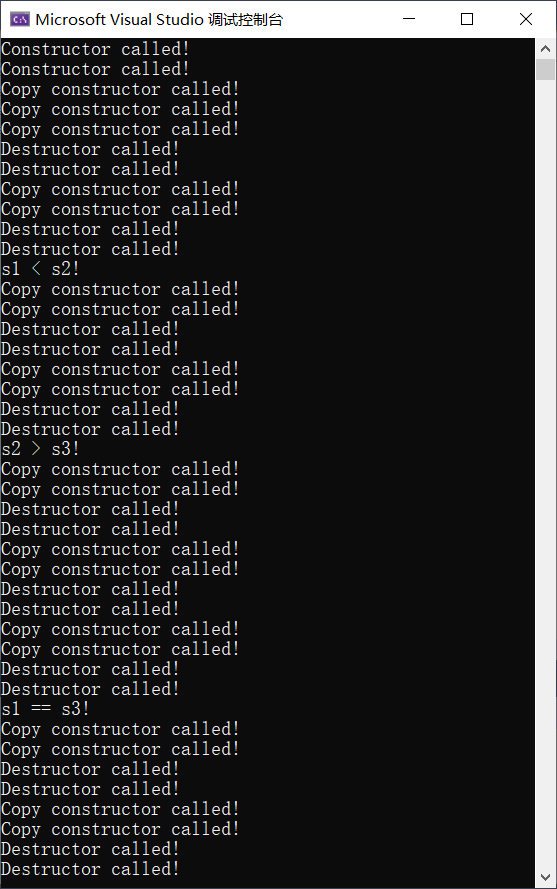
if (s1 >= s3) cout << "s1 >= s3!" << endl;

if (s1 == s3) cout << "s1 == s3!" << endl;

return 0;

}

**运行结果：**



**\*3.探究题：**

**（1）对于成员函数引用方式：**

#include <iostream>

#include <cstring>

#pragma warning(disable:4996)

using namespace std;

class String

{

public:

String() { \_str = new char[1]; \*\_str = '\0'; cout << "Constructor called!" << endl; }

String(const char\* str) { \_str = new char [strlen(str) + 1]; strcpy(\_str, str); cout << "Constructor called!" << endl; }

String(const String& \_InitStr)

{

cout << "Copy constructor called!" << endl;

if (&\_InitStr == this) //防止String s = s;的出现

{

\_str = new char[1]; \*\_str = '\0';

}

else

{

\_str = new char[strlen(\_InitStr.\_str) + 1];

strcpy(\_str, \_InitStr.\_str);

}

}

void Print() const { cout << \_str; }

void Set(const char\* \_newstr);

void Set(const String& \_newStr);

bool operator==(String& s) { return strcmp(\_str, s.\_str) == 0; }

bool operator<(String& s) { return strcmp(\_str, s.\_str) < 0; }

bool operator>(String& s) { return strcmp(\_str, s.\_str) > 0; }

bool operator!=(String& s) { return strcmp(\_str, s.\_str); }

bool operator>=(String& s) { return strcmp(\_str, s.\_str) >= 0; }

bool operator<=(String& s) { return strcmp(\_str, s.\_str) <= 0; }

~String() { delete[] \_str; cout << "Destructor called!" << endl; }

private:

char\* \_str;

};

void String::Set(const char\* \_newstr)

{

delete[] \_str;

\_str = new char[strlen(\_newstr) + 1];

strcpy(\_str, \_newstr);

}

void String::Set(const String& \_newStr)

{

if (&\_newStr == this) return;

delete[] \_str;

\_str = new char[strlen(\_newStr.\_str) + 1];

strcpy(\_str, \_newStr.\_str);

}

int main()

{

String s1("abcd"), s2("abce");

String s3 = s1;

if (s1 > s2) cout << "s1 > s2!" << endl;

else if (s1 < s2) cout << "s1 < s2!" << endl;

else if (s1 == s2) cout << "s1 == s2!" << endl;

if (s2 < s3) cout << "s2 < s3!" << endl;

else if (s2 > s3) cout << "s2 > s3!" << endl;

else if (s2 == s3) cout << "s2 == s3!" << endl;

if (s1 < s3) cout << "s1 < s3!" << endl;

else if (s1 > s3) cout << "s1 > s3!" << endl;

else if (s1 == s3) cout << "s1 == s3!" << endl;

if (s1 >= s2) cout << "s1 >= s2!" << endl;

if (s1 <= s2) cout << "s1 <= s2!" << endl;

if (s2 <= s3) cout << "s2 <= s3!" << endl;

if (s2 >= s3) cout << "s2 >= s3!" << endl;

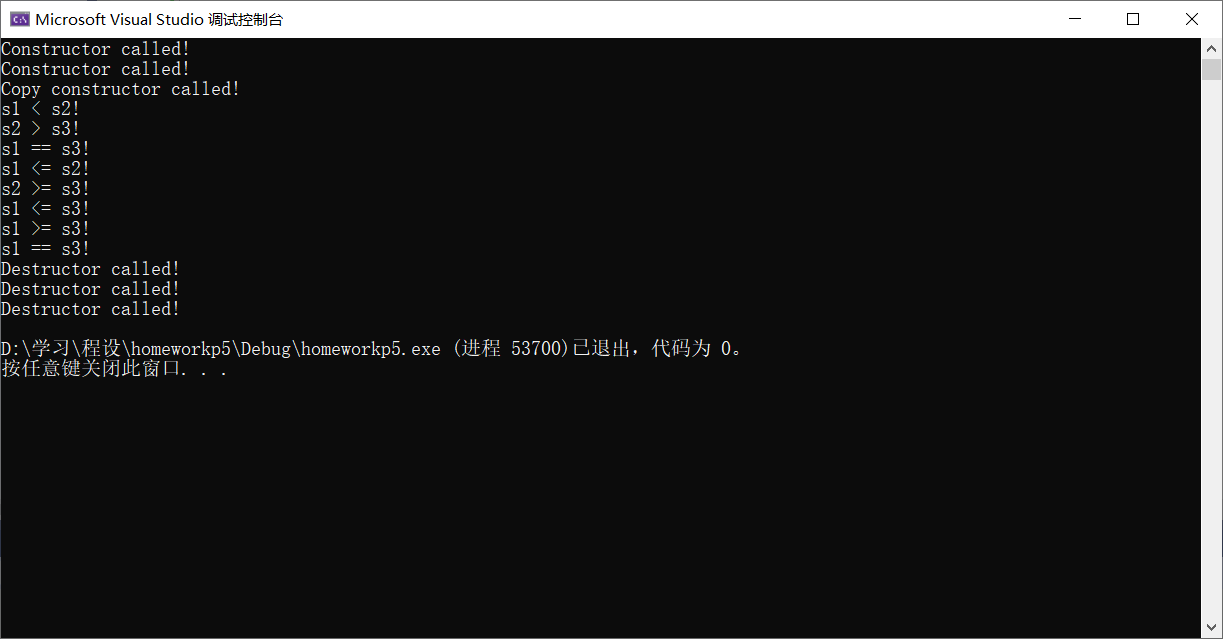
if (s1 <= s3) cout << "s1 <= s3!" << endl;

if (s1 >= s3) cout << "s1 >= s3!" << endl;

if (s1 == s3) cout << "s1 == s3!" << endl;

return 0;

}



**（2）对于友元函数引用方式：**

#include <iostream>

#include <cstring>

#pragma warning(disable:4996)

using namespace std;

class String

{

public:

String() { \_str = new char[1]; \*\_str = '\0'; cout << "Constructor called!" << endl; }

String(const char\* str) { \_str = new char[strlen(str) + 1]; strcpy(\_str, str); cout << "Constructor called!" << endl; }

String(const String& \_InitStr)

{

cout << "Copy constructor called!" << endl;

if (&\_InitStr == this) //防止String s = s;的出现

{

\_str = new char[1]; \*\_str = '\0';

}

else

{

\_str = new char[strlen(\_InitStr.\_str) + 1];

strcpy(\_str, \_InitStr.\_str);

}

}

void Print() const { cout << \_str; }

void Set(const char\* \_newstr);

void Set(const String& \_newStr);

friend bool operator==(String& s1, String& s2);

friend bool operator<(String& s1, String& s2);

friend bool operator>(String& s1, String& s2);

friend bool operator!=(String& s1, String& s2);

friend bool operator>=(String& s1, String& s2);

friend bool operator<=(String& s1, String& s2);

~String() { delete[] \_str; cout << "Destructor called!" << endl; }

private:

char\* \_str;

};

void String::Set(const char\* \_newstr)

{

delete[] \_str;

\_str = new char[strlen(\_newstr) + 1];

strcpy(\_str, \_newstr);

}

void String::Set(const String& \_newStr)

{

if (&\_newStr == this) return;

delete[] \_str;

\_str = new char[strlen(\_newStr.\_str) + 1];

strcpy(\_str, \_newStr.\_str);

}

bool operator==(String& s1, String& s2) { return strcmp(s1.\_str, s2.\_str) == 0; }

bool operator<(String& s1, String& s2) { return strcmp(s1.\_str, s2.\_str) < 0; }

bool operator>(String& s1, String& s2) { return strcmp(s1.\_str, s2.\_str) > 0; }

bool operator!=(String& s1, String& s2) { return strcmp(s1.\_str, s2.\_str); }

bool operator>=(String& s1, String& s2) { return strcmp(s1.\_str, s2.\_str) >= 0; }

bool operator<=(String& s1, String& s2) { return strcmp(s1.\_str, s2.\_str) <= 0; }

int main()

{

String s1("abcd"), s2("abce");

String s3 = s1;

if (s1 > s2) cout << "s1 > s2!" << endl;

else if (s1 < s2) cout << "s1 < s2!" << endl;

else if (s1 == s2) cout << "s1 == s2!" << endl;

if (s2 < s3) cout << "s2 < s3!" << endl;

else if (s2 > s3) cout << "s2 > s3!" << endl;

else if (s2 == s3) cout << "s2 == s3!" << endl;

if (s1 < s3) cout << "s1 < s3!" << endl;

else if (s1 > s3) cout << "s1 > s3!" << endl;

else if (s1 == s3) cout << "s1 == s3!" << endl;

if (s1 >= s2) cout << "s1 >= s2!" << endl;

if (s1 <= s2) cout << "s1 <= s2!" << endl;

if (s2 <= s3) cout << "s2 <= s3!" << endl;

if (s2 >= s3) cout << "s2 >= s3!" << endl;

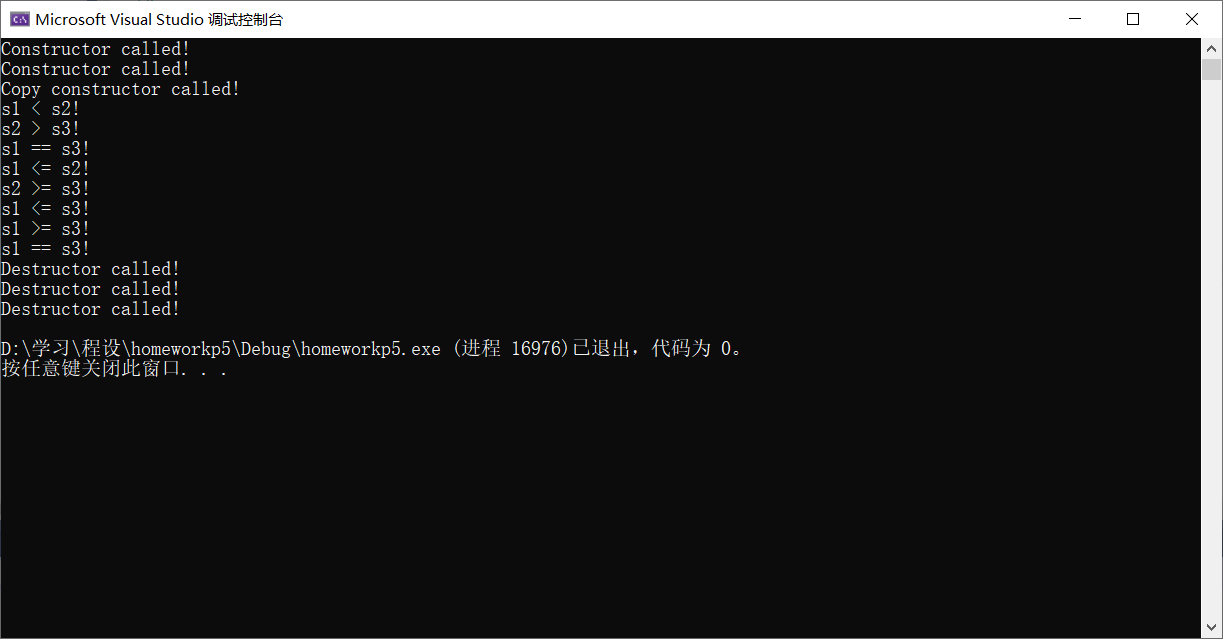
if (s1 <= s3) cout << "s1 <= s3!" << endl;

if (s1 >= s3) cout << "s1 >= s3!" << endl;

if (s1 == s3) cout << "s1 == s3!" << endl;

return 0;

}



**可见，采用引用方式传递参数可以减少复制构造函数的调用，进而提高效率。**