Classes and Dynamic Memory Allocation

【核心问题】，对于类中有动态内存分配的情况，默认情况下，初始化，赋值，函数按值传递，等使用的是**浅拷贝**，没有内存分配，而是直接复制内存地址（指针）

【解决思路】搞清楚各种机制，如果需要复制带有动态内存的对象，需要进行**深拷贝**，即，申请内存，再内存复制

```
#include <iostream>
class StringBad
{
private:
    char * str;                 // pointer to string
    unsigned int len;           // length of string
public:
    StringBad(const char * s); // constructor
    StringBad();                // default constructor
    ~StringBad();               // destructor
};
#endif

StringBad::StringBad(const char * s)
{
    len = std::strlen(s);               // set size
    str = new char[len + 1];            // allot storage
    std::strcpy(str, s);                // initialize
pointer
}

StringBad::StringBad()                  // default
constructor
{
    len = 4;
    str = new char[4];
```

```
    std::strcpy(str, "C++");          // default string
}

StringBad::~StringBad()               // necessary
destructor
{
    delete [] str;                    // required
}

//////////////////////////////////////////////////

void callme1(StringBad & rsb)
{
    cout << "String passed by reference:\n";
    cout << "    \"" << rsb << "\"\n";
}

void callme2(StringBad sb)
{
    cout << "String passed by value:\n";
    cout << "    \"" << sb << "\"\n";
}

int main()
{
    using std::endl;
    StringBad headline1("Celery Stalks at Midnight");
    StringBad headline2("Lettuce Prey");
    StringBad sports("Spinach Leaves Bowl for
Dollars");

    callme1(headline1);
    callme2(headline2);

    StringBad sailor = sports;

    StringBad knot;
    knot = headline1;

    return 0;
}
```

```
64
65
66  int *p1 = new int;
67  int *p2 = p1;
68
69  if(p1)
70  {
71      delet p1;
72      p1 = nullptr;
73  }
74  if(p2)
75  {
76      delet p2;
77      p2 = nullptr;
78  }
```

解决方案，补齐要素

析构函数，拷贝构造函数，赋值操作重载，深拷贝

```
1   #include <iostream>
2   using std::ostream;
3   using std::istream;
4
5   class String
6   {
7   private:
8       char * str;             // pointer to string
9       unsigned int len;                 // length of
    string
10      static int num_strings; // number of objects
11      static const int CINLIM = 80;  // cin input limit
12  public:
13  // constructors and other methods
14      String(const char * s); // constructor
15      String();               // default constructor
16      String(const String &); // copy constructor
17      ~String();              // destructor
18      int length () const { return len; }
19  // overloaded operator methods
20      String & operator=(const String &);
21      String & operator=(const char *);
```

```
22      char & operator[](int i);
23      const char & operator[](int i) const;
24  // overloaded operator friends
25      friend bool operator<(const String &st, const
    String &st2);
26      friend bool operator>(const String &st1, const
    String &st2);
27      friend bool operator==(const String &st, const
    String &st2);
28      friend ostream & operator<<(ostream & os, const
    String & st);
29      friend istream & operator>>(istream & is, String &
    st);
30  // static function
31      static int HowMany();
32  };
33
34
35  #include <cstring>                  // string.h for
    some
36  #include "04-string1.h"               // includes
    <iostream>
37  using std::cin;
38  using std::cout;
39
40  // initializing static class member
41
42  int String::num_strings = 0;
43
44  // static method
45  int String::HowMany()
46  {
47      return num_strings;
48  }
49
50  // class methods
51  String::String(const char * s)     // construct String
    from C string
52  {
53      len = std::strlen(s);          // set size
54      str = new char[len + 1];       // allot storage
```

```
55      std::strcpy(str, s);           // initialize
pointer
56      num_strings++;                 // set object count
57  }
58
59  String::String()                   // default
constructor
60  {
61      len = 4;
62      str = new char[1];
63      str[0] = '\0';                 // default string
64      num_strings++;
65  }
66
67  String::String(const String & st)
68  {
69      num_strings++;             // handle static member
update
70      len = st.len;              // same length
71      str = new char [len + 1];  // allot space
72      std::strcpy(str, st.str);  // copy string to new
location
73  }
74
75  String::~String()                  // necessary
destructor
76  {
77      --num_strings;                 // required
78      delete [] str;                 // required
79  }
80
81  // overloaded operator methods
82
83      // assign a String to a String
84  String & String::operator=(const String & st)
85  {
86      if (this == &st)
87          return *this;
88      delete [] str;
89      len = st.len;
90      str = new char[len + 1];
```

```
91      std::strcpy(str, st.str);
92      return *this;
93  }
94
95      // assign a C string to a String
96  String & String::operator=(const char * s)
97  {
98      delete [] str;
99      len = std::strlen(s);
100     str = new char[len + 1];
101     std::strcpy(str, s);
102     return *this;
103 }
104
105     // read-write char access for non-const String
106 char & String::operator[](int i)
107 {
108     return str[i];
109 }
110
111     // read-only char access for const String
112 const char & String::operator[](int i) const
113 {
114     return str[i];
115 }
116
117 // overloaded operator friends
118
119 bool operator<(const String &st1, const String &st2)
120 {
121     return (std::strcmp(st1.str, st2.str) < 0);
122 }
123
124 bool operator>(const String &st1, const String &st2)
125 {
126     return st2 < st1;
127 }
128
129 bool operator==(const String &st1, const String &st2)
130 {
131     return (std::strcmp(st1.str, st2.str) == 0);
```

```
}

    // simple String output
ostream & operator<<(ostream & os, const String & st)
{
    os << st.str;
    return os;
}

    // quick and dirty String input
istream & operator>>(istream & is, String & st)
{
    char temp[String::CINLIM];
    is.get(temp, String::CINLIM);
    if (is)
        st = temp;
    while (is && is.get() != '\n')
        continue;
    return is;
}
```