

• main.cpp

```

1 #include <iostream>
2 #include <sstream>
3 #include "String.h"
4
5 int main() {
6     //Power By NTNU_import_magic Discord!!
7     String s1, s2("abc"), s3(s2);
8     std::cout << "-----constructor-----\n";
9     std::cout << "s1(): " << s1 << '\n';
10    std::cout << "s2(\"abc\"): " << s2 << '\n';
11    std::cout << "s3(s2): " << s3 << '\n';
12    std::cout << "-----information----\n";
13    std::cout << "s3.size(): " << s3.size() << '\n';
14    std::cout << "s3.c_str(): " << s3.c_str() << '\n';
15    std::cout << "-----access-----\n";
16    std::cout << "s3[2]: " << s3[2] << '\n';
17    std::cout << "s3[2] = 'd';\n";
18    s3[2] = 'd';
19    std::cout << "s3: " << s3 << '\n';
20    std::cout << "-----operator+=-----\n";
21    std::cout << "s2 += s3;\n";
22    s2 += s3;
23    std::cout << "s2: " << s2 << '\n';
24    std::cout << "----copy assignment----\n";
25    std::cout << "s1 = s3;\n";
26    s1 = s3;
27    std::cout << "s1: " << s1 << '\n';
28    std::cout << "-----swap-----\n";
29    std::cout << "s1: " << s1 << '\n';
30    std::cout << "s1.size(): " << s1.size() << '\n';
31    std::cout << "s1.capacity(): " << s1.capacity() << '\n';
32    std::cout << '\n';
33    std::cout << "s2: " << s2 << '\n';
34    std::cout << "s2.size(): " << s2.size() << '\n';
35    std::cout << "s2.capacity(): " << s2.capacity() << '\n';
36    std::cout << '\n';
37    std::cout << "s1.swap(s2);\n";
38    s1.swap(s2);
39    std::cout << '\n';
40    std::cout << "s1: " << s1 << '\n';
41    std::cout << "s1.size(): " << s1.size() << '\n';
42    std::cout << "s1.capacity(): " << s1.capacity() << '\n';
43    std::cout << '\n';
44    std::cout << "s2: " << s2 << '\n';
45    std::cout << "s2.size(): " << s2.size() << '\n';
46    std::cout << "s2.capacity(): " << s2.capacity() << '\n';
47    std::cout << "-----plus-----\n";
48    std::cout << "s1: " << s1 << '\n';
49    std::cout << "s2: " << s2 << '\n';
50    std::cout << "s3: " << s3 << '\n';
51    std::cout << "s1 + ',' + ' ' + s2 + \"\\\", \" + s3: " << s1 + ',' + ' ' + s2 + "\", " + s3 << '\n';
52    std::cout << "-----relational-----\n";
53    std::cout << "s1: " << s1 << '\n' << "s2: " << s2 << '\n';
54    std::cout << "s1 == s2: " << ((s1 == s2) ? "True" : "False") << '\n';
55    std::cout << "s1 != s2: " << ((s1 != s2) ? "True" : "False") << '\n';
56    std::cout << "s1 < s2: " << ((s1 < s2) ? "True" : "False") << '\n';
57    std::cout << "s1 <= s2: " << ((s1 <= s2) ? "True" : "False") << '\n';
58    std::cout << "s1 > s2: " << ((s1 > s2) ? "True" : "False") << '\n';

```

```

59     std::cout << "s1 >= s2: " << ((s1 >= s2) ? "True" : "False") << '\n';
60     std::cout << "-----iostream-----\n";
61     std::cout << "stringstream ss(\"123456789 12345\\n6789\");\n";
62     std::stringstream ss("123456789 12345\n6789");
63     std::cout << "ss >> s1 >> s2 >> s3;\n";
64     ss >> s1 >> s2 >> s3;
65     std::cout << "s1: " << s1 << '\n';
66     std::cout << "s2: " << s2 << '\n';
67     std::cout << "s3: " << s3 << '\n';
68     std::cout << "-----clear-----\n";
69     std::cout << "s3.clear();\n";
70     s3.clear();
71     std::cout << "s3.size(): " << s3.size() << '\n';
72     std::cout << "s3.c_str(): " << s3.c_str() << '\n';
73     return 0;
74 }

```

• String.h

```

1  #ifndef _String_H
2  #define _String_H
3  #include <iostream>
4  #include <cstring>
5  #include <cctype>
6
7  class String {
8  private:
9      size_t size_ = 0, capacity_ = 0;
10     char *str_ = nullptr;
11 public:
12     String();
13     String(const char *);
14     String(const String &);
15     String(std::nullptr_t) = delete;
16     ~String();
17     size_t size();
18     size_t capacity();
19     const char *c_str() const;
20     void reserve(size_t);
21     void clear();
22     void swap(String &);
23     char &operator[] (size_t);
24     const char &operator[] (size_t) const;
25     String & operator+= (const char *);
26     String & operator+= (const String &);
27     String & operator+= (char);
28     String & operator= (const String &);
29     String & operator= (const char *);
30     String & operator= (char);
31     // Non-member function
32     friend String operator+ (const String &, const String &);
33     friend String operator+ (const String &, const char *);
34     friend String operator+ (const char *, const String &);
35     friend String operator+ (const String &, char);
36     friend String operator+ (char, const String &);
37     friend bool operator== (const String &, const String &);
38     friend bool operator== (const char *, const String &);
39     friend bool operator== (const String &, const char *);
40     friend bool operator!= (const String &, const String &);
41     friend bool operator!= (const char *, const String &);
42     friend bool operator!= (const String &, const char *);

```

```

43     friend bool operator< (const String &, const String &);
44     friend bool operator< (const char *, const String &);
45     friend bool operator< (const String &, const char *);
46     friend bool operator<= (const String &, const String &);
47     friend bool operator<= (const char *, const String &);
48     friend bool operator<= (const String &, const char *);
49     friend bool operator> (const String &, const String &);
50     friend bool operator> (const char *, const String &);
51     friend bool operator> (const String &, const char *);
52     friend bool operator>= (const String &, const String &);
53     friend bool operator>= (const char *, const String &);
54     friend bool operator>= (const String &, const char *);
55     // I/O stream
56     friend std::istream & operator>> (std::istream &, String &);
57     friend std::ostream & operator<< (std::ostream &, const String &);
58 };
59
60 #endif

```

- String.cpp

```

1  #include "String.h"
2
3  String::String() : size_(0), capacity_(1), str_(new char[1]) {
4      str_[0] = '\0';
5  }
6
7  String::String(const char *t) : size_(strlen(t)), capacity_(size_ + 1), str_(new char[size_ +
8      1]) {
9      strncpy(str_, t, size_ + 1);
10 }
11 String::String(const String &t) : size_(t.size_), capacity_(t.capacity_), str_(new char[size_ +
12     1]) {
13     strncpy(str_, t.str_, size_ + 1);
14 }
15 String::~String() {
16     delete[] str_;
17 }
18
19 size_t String::size() {
20     return size_;
21 }
22
23 size_t String::capacity() {
24     return capacity_;
25 }
26
27 const char * String::c_str() const {
28     return str_;
29 }
30
31 void String::reserve(size_t n) {
32     if(n >= capacity_) {
33         while(capacity_ < n) capacity_ <= 1;
34         char *newStr = new char[capacity_];
35         strncpy(newStr, str_, size_ + 1);
36         delete[] str_;
37         str_ = newStr;
38     }

```

```
39 }
40
41 void String::clear() {
42     size_ = 0;
43     str_[0] = '\0';
44 }
45
46 void String::swap(String &t) {
47     size_ ^= t.size_;
48     t.size_ ^= size_;
49     size_ ^= t.size_;
50     capacity_ ^= t.capacity_;
51     t.capacity_ ^= capacity_;
52     capacity_ ^= t.capacity_;
53     char *tmp = str_;
54     str_ = t.str_;
55     t.str_ = tmp;
56 }
57
58 char &String::operator[] (size_t pos) {
59     return str_[pos];
60 }
61
62 const char &String::operator[] (size_t pos) const {
63     return str_[pos];
64 }
65
66 String &String::operator+= (const char *t) {
67     size_t tLen = strlen(t);
68     size_t newSize = size_ + tLen;
69     if(newSize + 1 > capacity_) reserve(newSize);
70     strncpy(str_ + size_, t, tLen + 1);
71     size_ = newSize;
72     return *this;
73 }
74
75 String &String::operator+= (const String &t) {
76     return operator+= (t.str_);
77 }
78
79 String &String::operator+= (char c) {
80     char str[2] = {c, '\0'};
81     return operator+= (str);
82 }
83
84 String &String::operator= (const char *t) {
85     size_t tLen = strlen(t);
86     if(tLen + 1 > capacity_) reserve(tLen);
87     strncpy(str_, t, tLen + 1);
88     size_ = tLen;
89     return *this;
90 }
91
92 String &String::operator= (const String &t) {
93     return operator= (t.str_);
94 }
95
96 String &String::operator= (char c) {
97     char str[2] = {c, '\0'};
98     return operator= (str);
99 }
```

```
100
101 String operator+ (const String &lhs, const char *rhs) {
102     String tmp(lhs);
103     tmp.operator += (rhs);
104     return tmp;
105 }
106
107 String operator+ (const String &lhs, const String &rhs) {
108     return operator+ (lhs, rhs.str_);
109 }
110
111 String operator+ (const char *lhs, const String &rhs) {
112     return operator+ (rhs, lhs);
113 }
114
115 String operator+ (const String &lhs, char rhs) {
116     char str[2] = {rhs, '\0'};
117     return operator+ (lhs, str);
118 }
119
120 String operator+ (char lhs, const String &rhs) {
121     char str[2] = {lhs, '\0'};
122     return operator+ (rhs, str);
123 }
124
125 bool operator== (const String &lhs, const char *rhs) {
126     return strncmp(lhs.str_, rhs, std::max(lhs.size_, strlen(rhs))) == 0;
127 }
128
129 bool operator== (const String &lhs, const String &rhs) {
130     return strncmp(lhs.str_, rhs.str_, std::max(lhs.size_, rhs.size_)) == 0;
131 }
132
133 bool operator== (const char *lhs, const String &rhs) {
134     return operator==(rhs, lhs);
135 }
136
137 bool operator!= (const String &lhs, const String &rhs) {
138     return !operator== (lhs, rhs);
139 }
140
141 bool operator!= (const char *lhs, const String &rhs) {
142     return !operator== (lhs, rhs);
143 }
144
145 bool operator!= (const String &lhs, const char *rhs) {
146     return !operator== (lhs, rhs);
147 }
148
149 bool operator< (const String &lhs, const char *rhs) {
150     return strncmp(lhs.str_, rhs, std::max(lhs.size_, strlen(rhs))) < 0;
151 }
152
153 bool operator< (const String &lhs, const String &rhs) {
154     return strncmp(lhs.str_, rhs.str_, std::max(lhs.size_, rhs.size_)) < 0;
155 }
156
157 bool operator< (const char *lhs, const String &rhs) {
158     return operator<(rhs, lhs);
159 }
160
```

```

161 bool operator<= (const String &lhs, const String &rhs) {
162     return !operator>(lhs, rhs);
163 }
164
165 bool operator<= (const char *lhs, const String &rhs) {
166     return !operator>(lhs, rhs);
167 }
168
169 bool operator<= (const String &lhs, const char *rhs) {
170     return !operator>(lhs, rhs);
171 }
172
173 bool operator> (const String &lhs, const char *rhs) {
174     return strcmp(lhs.str_, rhs, std::max(lhs.size_, strlen(rhs))) > 0;
175 }
176
177 bool operator> (const String &lhs, const String &rhs) {
178     return strcmp(lhs.str_, rhs.str_, std::max(lhs.size_, rhs.size_)) > 0;
179 }
180
181 bool operator> (const char *lhs, const String &rhs) {
182     return operator>(rhs, lhs);
183 }
184
185 bool operator>= (const String &lhs, const String &rhs) {
186     return !operator<(lhs, rhs);
187 }
188
189 bool operator>= (const char *lhs, const String &rhs) {
190     return !operator<(lhs, rhs);
191 }
192
193 bool operator>= (const String &lhs, const char *rhs) {
194     return !operator<(lhs, rhs);
195 }
196
197 std::istream & operator>> (std::istream &is, String &t) {
198     t.clear();
199     char c;
200     while(is.get(c) && isspace(c));
201     is.putback(c);
202     while(is.get(c) && !isspace(c))
203         t += c;
204     is.putback(c);
205     return is;
206 }
207
208 std::ostream & operator<< (std::ostream &os, const String &t) {
209     return os << t.str_;
210 }

```

- makefile

```

1 all:
2     g++ main.cpp String.cpp -o main
3
4 clean:
5     rm -rf main

```