- main.cpp

```cpp
#include <iostream>
#include <sstream>
#include "String.h"

int main() {
    //Power By NTNU_import_magic Discord!!
    String s1, s2("abc"), s3(s2);
    std::cout << "------constructor-----\n";
    std::cout << "s1(): " << s1 << '\n';
    std::cout << "s2(\"abc\"): " << s2 << '\n';
    std::cout << "s3(s2): " << s3 << '\n';
    std::cout << "------information----\n";
    std::cout << "s3.size(): " << s3.size() << '\n';
    std::cout << "s3.c_ctr(): " << s3.c_str() << '\n';
    std::cout << "-------access--------\n";
    std::cout << "s3[2]: " << s3[2] << '\n';
    std::cout << "s3[2] = 'd';\n";
    s3[2] = 'd';
    std::cout << "s3: " << s3 << '\n';
    std::cout << "-----operator+=------\n";
    std::cout << "s2 += s3;\n";
    s2 += s3;
    std::cout << "s2: " << s2 << '\n';
    std::cout << "----copy assignment----\n";
    std::cout << "s1 = s3;\n";
    s1 = s3;
    std::cout << "s1: " << s1 << '\n';
    std::cout << "----------swap--------\n";
    std::cout << "s1: " << s1 << '\n';
    std::cout << "s1.size(): " << s1.size() << '\n';
    std::cout << "s1.capacity(): " << s1.capacity() << '\n';
    std::cout << '\n';
    std::cout << "s2: " << s2 << '\n';
    std::cout << "s2.size(): " << s2.size() << '\n';
    std::cout << "s2.capacity(): " << s2.capacity() << '\n';
    std::cout << '\n';
    std::cout << "s1.swap(s2);\n";
    s1.swap(s2);
    std::cout << '\n';
    std::cout << "s1: " << s1 << '\n';
    std::cout << "s1.size(): " << s1.size() << '\n';
    std::cout << "s1.capacity(): " << s1.capacity() << '\n';
    std::cout << '\n';
    std::cout << "s2: " << s2 << '\n';
    std::cout << "s2.size(): " << s2.size() << '\n';
    std::cout << "s2.capacity(): " << s2.capacity() << '\n';
    std::cout << "--------plus----------\n";
    std::cout << "s1: " << s1 << '\n';
    std::cout << "s2: " << s2 << '\n';
    std::cout << "s3: " << s3 << '\n';
    std::cout << "s1 + ',' + ' ' + s2 + \", \" + s3: " << s1 + ',' + ' ' + s2 + ", " + s3 << '\
        n';
    std::cout << "-----relational-------\n";
    std::cout << "s1: " << s1 << '\n' << "s2: " << s2 << '\n';
    std::cout << "s1 == s2: " << ((s1 == s2) ? "True" : "False") << '\n';
    std::cout << "s1 != s2: " << ((s1 != s2) ? "True" : "False") << '\n';
    std::cout << "s1 < s2: " << ((s1 < s2) ? "True" : "False") << '\n';
    std::cout << "s1 <= s2: " << ((s1 <= s2) ? "True" : "False") << '\n';
    std::cout << "s1 > s2: " << ((s1 > s2) ? "True" : "False") << '\n';
```

```
    std::cout << "s1 >= s2: " << ((s1 >= s2) ? "True" : "False") << '\n';
    std::cout << "-----iostream-------\n";
    std::cout << "stringstream ss(\"123456789 12345\\n6789\");\n";
    std::stringstream ss("123456789 12345\n6789");
    std::cout << "ss >> s1 >> s2 >> s3;\n";
    ss >> s1 >> s2 >> s3;
    std::cout << "s1: " << s1 << '\n';
    std::cout << "s2: " << s2 << '\n';
    std::cout << "s3: " << s3 << '\n';
    std::cout << "--------clear---------\n";
    std::cout << "s3.clear();\n";
    s3.clear();
    std::cout << "s3.size(): " << s3.size() << '\n';
    std::cout << "s3.c_ctr(): " << s3.c_str() << '\n';
    return 0;
}
```

- String.h

```
#ifndef _String_H
#define _String_H
#include <iostream>
#include <cstring>
#include <cctype>

class String {
  private:
    size_t size_ = 0, capacity_ = 0;
    char *str_ = nullptr;
  public:
    String();
    String(const char *);
    String(const String &);
    String(std::nullptr_t) = delete;
    ~String();
    size_t size();
    size_t capacity();
    const char *c_str() const;
    void reserve(size_t);
    void clear();
    void swap(String &);
    char &operator[] (size_t);
    const char &operator[] (size_t) const;
    String & operator+= (const char *);
    String & operator+= (const String &);
    String & operator+= (char);
    String & operator= (const String &);
    String & operator= (const char *);
    String & operator= (char);
    // Non-member function
    friend String operator+ (const String &, const String &);
    friend String operator+ (const String &, const char *);
    friend String operator+ (const char *, const String &);
    friend String operator+ (const String &, char);
    friend String operator+ (char, const String &);
    friend bool operator== (const String &, const String &);
    friend bool operator== (const char *, const String &);
    friend bool operator== (const String &, const char *);
    friend bool operator!= (const String &, const String &);
    friend bool operator!= (const char *, const String &);
    friend bool operator!= (const String &, const char *);
```

```
    friend bool operator< (const String &, const String &);
    friend bool operator< (const char *, const String &);
    friend bool operator< (const String &, const char *);
    friend bool operator<= (const String &, const String &);
    friend bool operator<= (const char *, const String &);
    friend bool operator<= (const String &, const char *);
    friend bool operator> (const String &, const String &);
    friend bool operator> (const char *, const String &);
    friend bool operator> (const String &, const char *);
    friend bool operator>= (const String &, const String &);
    friend bool operator>= (const char *, const String &);
    friend bool operator>= (const String &, const char *);
    // I/O stream
    friend std::istream & operator>> (std::istream &, String &);
    friend std::ostream & operator<< (std::ostream &, const String &);
};

#endif
```

- String.cpp

```
#include "String.h"

String::String() : size_(0), capacity_(1), str_(new char[1]) {
    str_[0] = '\0';
}

String::String(const char *t) : size_(strlen(t)), capacity_(size_ + 1), str_(new char[size_ + 1]) {
    strncpy(str_, t, size_ + 1);
}

String::String(const String &t) : size_(t.size_), capacity_(t.capacity_), str_(new char[size_ + 1]) {
    strncpy(str_, t.str_, size_ + 1);
}

String::~String() {
    delete[] str_;
}

size_t String::size() {
    return size_;
}

size_t String::capacity() {
    return capacity_;
}

const char * String::c_str() const {
    return str_;
}

void String::reserve(size_t n) {
    if(n >= capacity_) {
        while(capacity_ < n) capacity_ <<= 1;
        char *newStr = new char[capacity_];
        strncpy(newStr, str_, size_ + 1);
        delete[] str_;
        str_ = newStr;
    }
```

```
}

void String::clear() {
    size_ = 0;
    str_[0] = '\0';
}

void String::swap(String &t) {
    size_ ^= t.size_;
    t.size_ ^= size_;
    size_ ^= t.size_;
    capacity_ ^= t.capacity_;
    t.capacity_ ^= capacity_;
    capacity_ ^= t.capacity_;
    char *tmp = str_;
    str_ = t.str_;
    t.str_ = tmp;
}

char & String::operator[] (size_t pos) {
    return str_[pos];
}

const char & String::operator[] (size_t pos) const {
    return str_[pos];
}

String & String::operator+= (const char *t) {
    size_t tLen = strlen(t);
    size_t newSize = size_ + tLen;
    if(newSize + 1 > capacity_) reserve(newSize);
    strncpy(str_ + size_, t, tLen + 1);
    size_ = newSize;
    return *this;
}

String & String::operator+= (const String &t) {
    return operator+= (t.str_);
}

String & String::operator+= (char c) {
    char str[2] = {c, '\0'};
    return operator+= (str);
}

String & String::operator= (const char *t) {
    size_t tLen = strlen(t);
    if(tLen + 1 > capacity_) reserve(tLen);
    strncpy(str_, t, tLen + 1);
    size_ = tLen;
    return *this;
}

String & String::operator= (const String &t) {
    return operator= (t.str_);
}

String & String::operator= (char c) {
    char str[2] = {c, '\0'};
    return operator= (str);
}
```

```
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
```

```cpp
bool operator<= (const String &lhs, const String &rhs) {
    return !operator>(lhs, rhs);
}

bool operator<= (const char *lhs, const String &rhs) {
    return !operator>(lhs, rhs);
}

bool operator<= (const String &lhs, const char *rhs) {
    return !operator>(lhs, rhs);
}

bool operator> (const String &lhs, const char *rhs) {
    return strncmp(lhs.str_, rhs, std::max(lhs.size_, strlen(rhs))) > 0;
}

bool operator> (const String &lhs, const String &rhs) {
    return strncmp(lhs.str_, rhs.str_, std::max(lhs.size_, rhs.size_)) > 0;
}

bool operator> (const char *lhs, const String &rhs) {
    return operator>(rhs, lhs);
}

bool operator>= (const String &lhs, const String &rhs) {
    return !operator<(lhs, rhs);
}

bool operator>= (const char *lhs, const String &rhs) {
    return !operator<(lhs, rhs);
}

bool operator>= (const String &lhs, const char *rhs) {
    return !operator<(lhs, rhs);
}

std::istream & operator>> (std::istream &is, String &t) {
    t.clear();
    char c;
    while(is.get(c) && isspace(c));
    is.putback(c);
    while(is.get(c) && !isspace(c))
        t += c;
    is.putback(c);
    return is;
}

std::ostream & operator<< (std::ostream &os, const String &t) {
    return os << t.str_;
}
```

- makefile

```
all:
  g++ main.cpp String.cpp -o main

clean:
  rm -rf main
```