

```
/* -----  
// string.h  
// ----- */  
  
#include <iostream>  
#include <string.h>  
  
void my_strncpy(char *obj1, const char *obj2, size_t n) {  
    // avoid stopping at '\0'  
    for ( size_t i=0; i<n; ++i )  
        obj1[i] = obj2[i];  
}  
  
class String {  
  
    friend std::ostream & operator << (std::ostream &os, const String  
&obj);  
    friend std::istream & operator >> (std::istream &is, String &obj);  
  
public:  
  
    // 1. default constructor  
    String() {  
        str_ = new char[capacity_];  
    }  
  
    // 2. copy constructor  
    String(const String &obj) {  
        str_ = new char[obj.size_];  
        size_ = obj.size_;  
        capacity_ = obj.size_;  
        my_strncpy(str_, obj.str_, size_);  
    }  
}
```

```
// 3. constructor with one parameter with type const char *
String(const char *obj) {
    str_ = new char[strlen(obj)];
    size_ = strlen(obj);
    capacity_ = strlen(obj);
    my_strncpy(str_, obj, size_);
}

// 4. destructor
~String() {
    if ( str_ ) delete[] str_;
}

// 5. size()
size_t size() const { return size_; }

// 6. c_str()
const char * c_str() const { return str_; }

// 7. operator []
char & operator [] (size_t i) const { return str_[i]; }

// 8. operator +=
String & operator += (const String &obj) {
    if ( capacity_ < size_ + obj.size_ ) {
        char *temp = new char[size_ + obj.size_];
        capacity_ = size_ + obj.size_;
        if ( str_ ) {
            my_strncpy(temp, str_, size_);
            delete[] str_;
        }
        str_ = temp;
    }
    for ( size_t i=size_; i<size_+obj.size_; ++i )
        str_[i] = obj.str_[i-size_];
    size_ += obj.size_;
    return *this;
}
```

```
// 9. clear()
void clear() {
    if ( !str_ ) return;
    str_[0] = '\0';
    size_ = 0;
}

// 10. operator =
String & operator = (String obj) {
    this->swap(obj);
    return *this;
}

// 11. swap()
void swap(String &obj) {
    using std::swap;
    swap(str_, obj.str_);
    swap(size_, obj.size_);
    swap(capacity_, obj.capacity_);
}

private:
    char *str_ = nullptr;
    size_t size_ = 0;
    size_t capacity_ = 0;
};
```

```
// A. relational operators (<, >, <=, >=, ==, !=)
bool operator < (const String &obj1, const String &obj2) {
    size_t minimum_length = std::min(obj1.size(), obj2.size());
    for ( size_t i=0; i<minimum_length; ++i )
        if ( obj1[i] != obj2[i] )
            return obj1[i] < obj2[i];

    // if they have the same prefix, the shorter, the smaller
    return obj1.size() < obj2.size();
}

bool operator > (const String &obj1, const String &obj2) {
    size_t minimum_length = std::min(obj1.size(), obj2.size());
    for ( size_t i=0; i<minimum_length; ++i )
        if ( obj1[i] != obj2[i] )
            return obj1[i] > obj2[i];

    // if they have the same prefix, the longer, the larger
    return obj1.size() > obj2.size();
}

bool operator <= (const String &obj1, const String &obj2) {
    size_t minimum_length = std::min(obj1.size(), obj2.size());
    for ( size_t i=0; i<minimum_length; ++i )
        if ( obj1[i] != obj2[i] )
            return obj1[i] <= obj2[i];

    // if they have the same prefix, compare the length of them
    return obj1.size() <= obj2.size();
}

bool operator >= (const String &obj1, const String &obj2) {
    size_t minimum_length = std::min(obj1.size(), obj2.size());
    for ( size_t i=0; i<minimum_length; ++i )
        if ( obj1[i] != obj2[i] )
            return obj1[i] >= obj2[i];

    // if they have the same prefix, compare the length of them
    return obj1.size() >= obj2.size();
}
```

```
bool operator == (const String &obj1, const String &obj2) {
    // different length implies they're different
    if ( obj1.size() != obj2.size() ) return false;
    for ( size_t i=0; i<obj1.size(); ++i )
        if ( obj1[i] != obj2[i] )
            return false;
    return true;
}

bool operator != (const String &obj1, const String &obj2) {
    // different length implies they're different
    if ( obj1.size() != obj2.size() ) return true;
    size_t minimum_length = std::min(obj1.size(), obj2.size());
    for ( size_t i=0; i<obj1.size(); ++i )
        if ( obj1[i] != obj2[i] )
            return true;
    return false;
}
```

```
// B. operator <<, >>
std::ostream & operator << (std::ostream &os, const String &obj) {
    for ( size_t i=0; i<obj.size_; ++i )
        os << obj.str_[i];
    return os;
}

// std::istream & operator >> (std::istream &is, String &obj) {

// }

// C. operator +
const String operator + (const String &obj1, const String &obj2) {
    String temp = obj1;
    temp += obj2;
    return temp;
}
```