C++中的字符串的使用和实现

Problems with C string

- In C
  - String is represented by an array of characters
  - Character arrays are treated differently than ordinary arrays
- Problems: user must know about some irrelevant details
  - Arrays have bounds
  - Meaningful characters must followed by a numerical zero

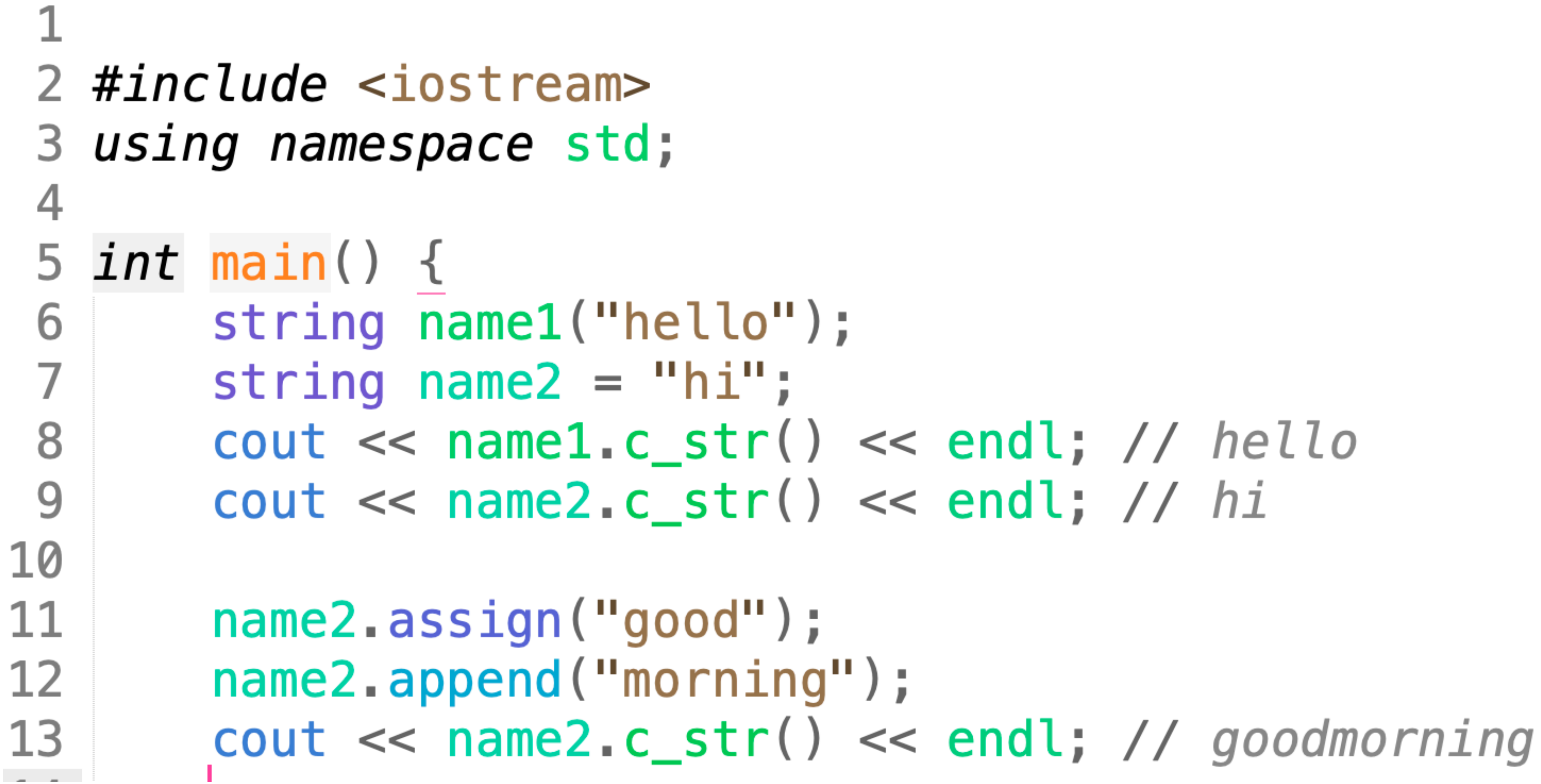
How C++ solves these problems

- Implement a class and a set of functions, such that the user
  - Using dynamically allocating character arrays to allow strings of any sizes
  - Encapsulating the character array and provide a set of comprehensive functions
- The results:
  - Need not knowing about array bounds and the zero at the end
  - Intuitive usage analogous to fundamental data types

The string class

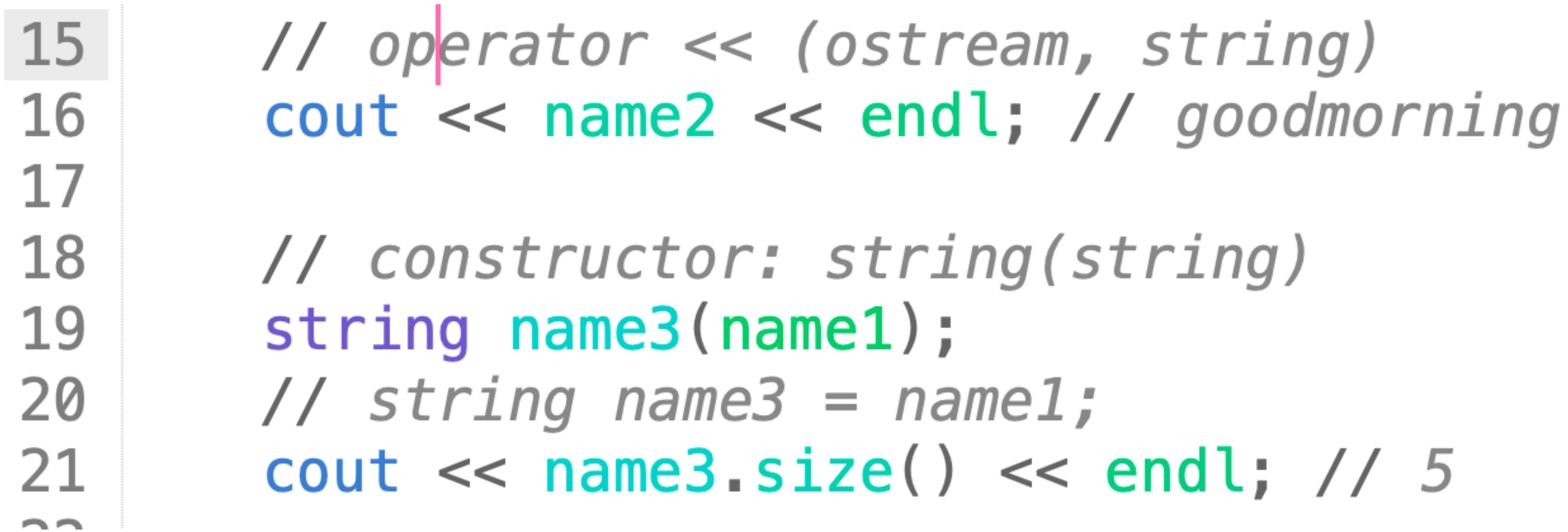
- Must include the C++ string library
  - #include <string>
  - using namespace std;

C++ string usage



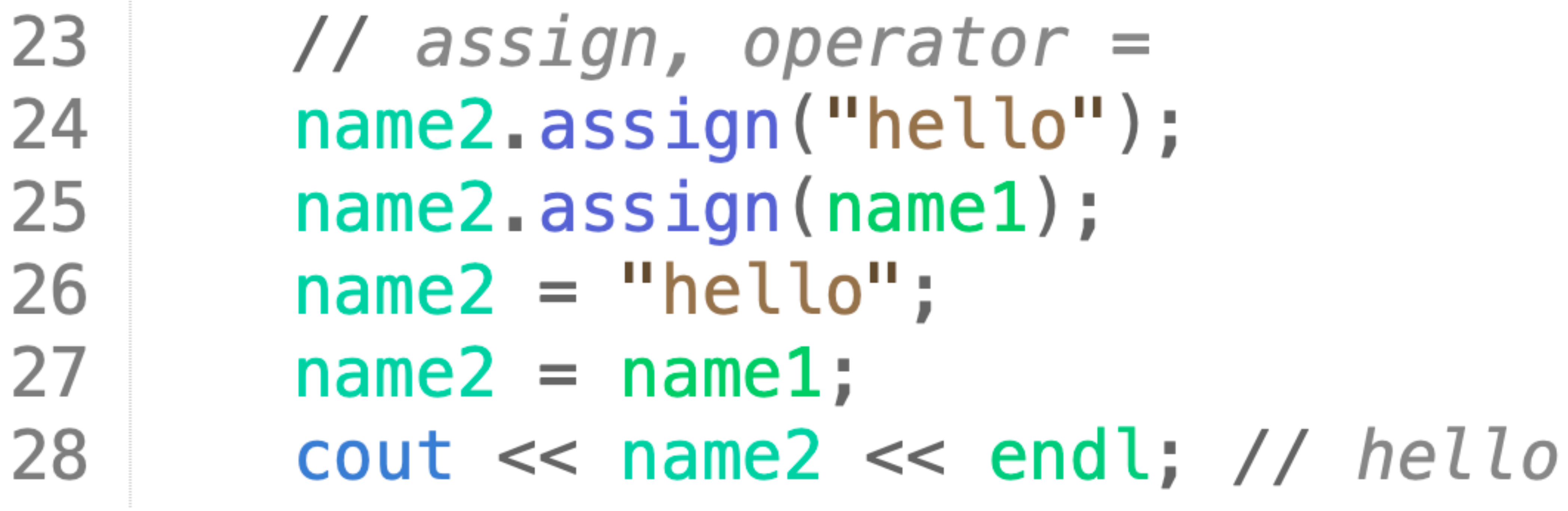
```cpp

#include <iostream>
using namespace std;

int main() {
	string name1("hello");
	string name2 = "hi";
	cout << name1.c_str() << endl; // hello
	cout << name2.c_str() << endl; // hi

	name2.assign("good");
	name2.append("morning");
	cout << name2.c_str() << endl; // goodmorning
```

C++ string usage



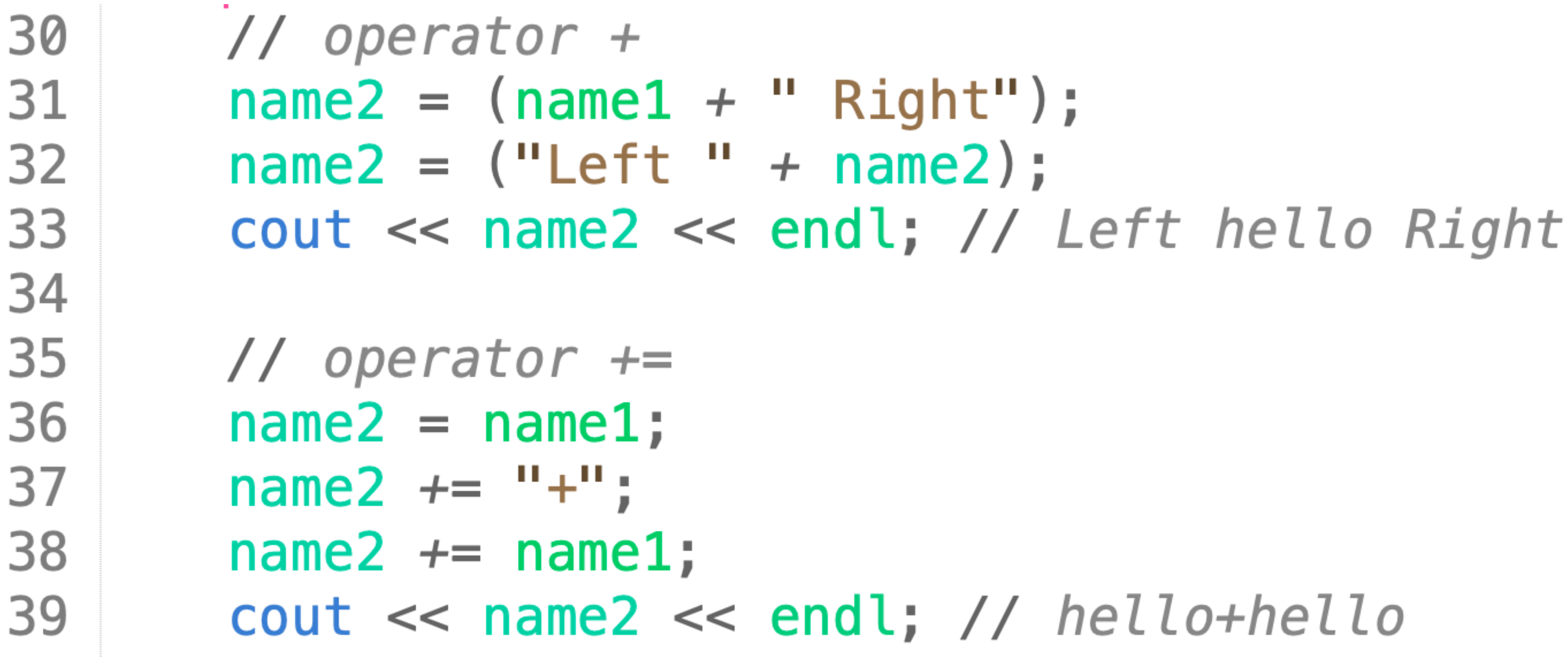
```
    // operator << (ostream, string)
    cout << name2 << endl; // goodmorning

    // constructor: string(string)
    string name3(name1);
    // string name3 = name1;
    cout << name3.size() << endl; // 5
```

C++ string usage



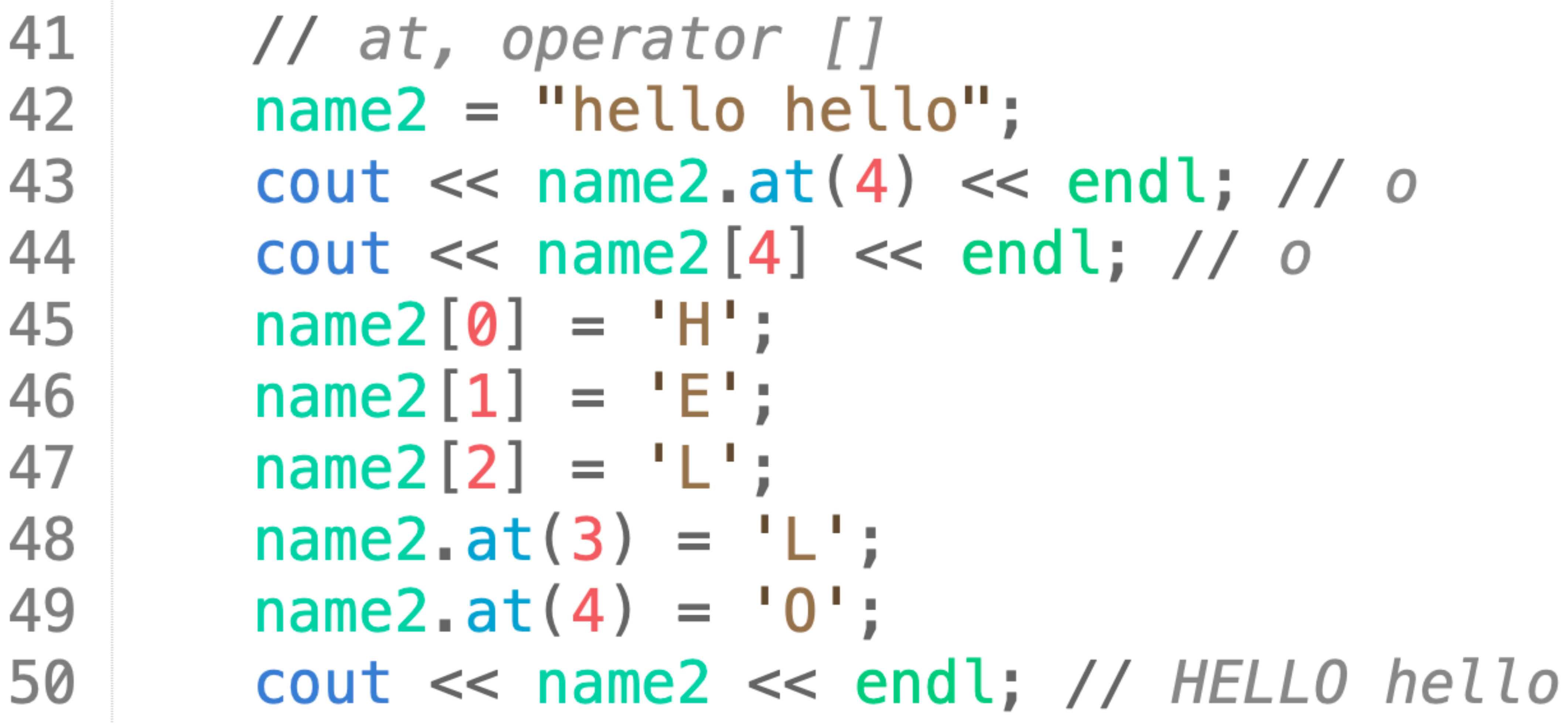
```
    // assign, operator =
    name2.assign("hello");
    name2.assign(name1);
    name2 = "hello";
    name2 = name1;
    cout << name2 << endl; // hello
```

C++ string usage



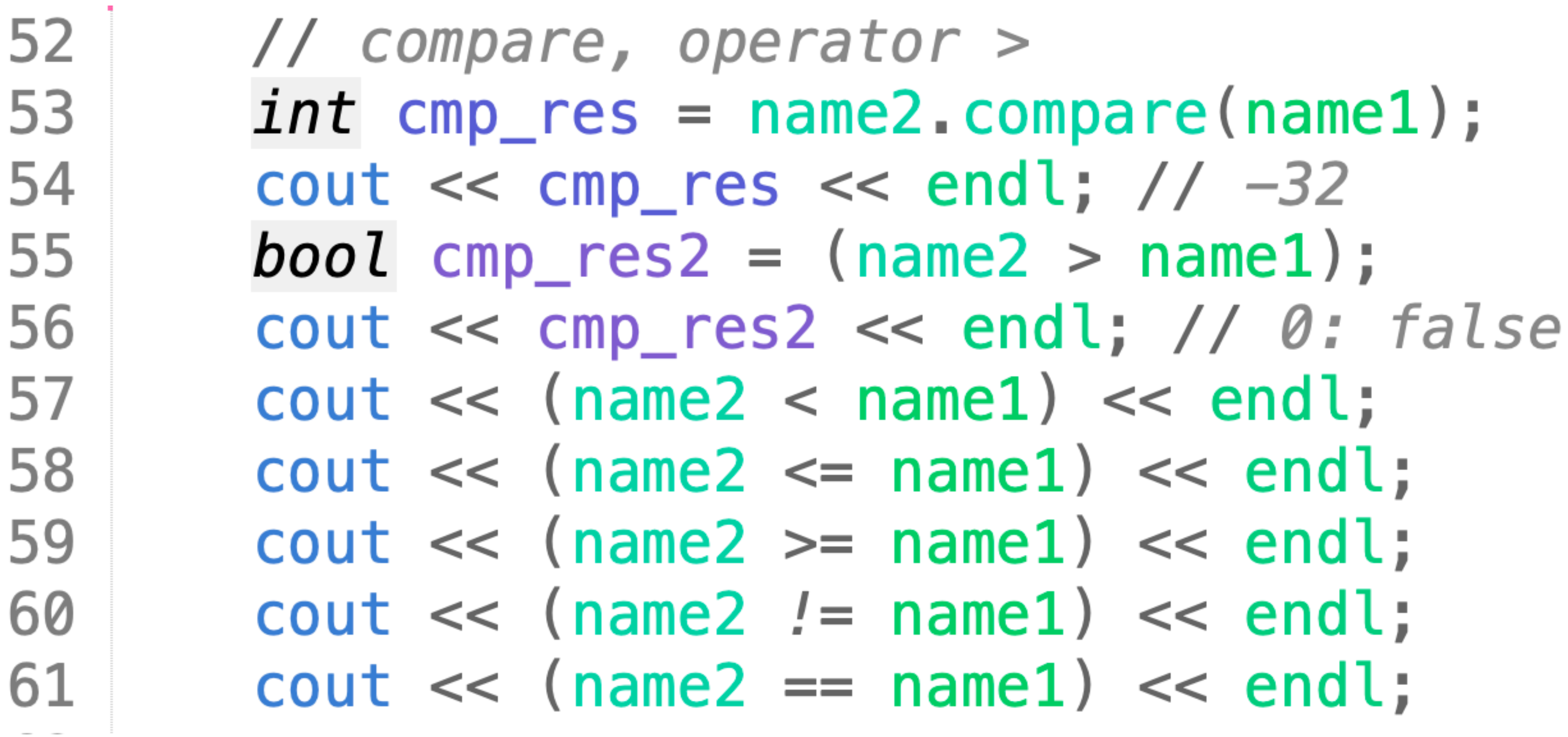
```
    // operator +
    name2 = (name1 + " Right");
    name2 = ("Left " + name2);
    cout << name2 << endl; // Left hello Right

    // operator +=
    name2 = name1;
    name2 += "+";
    name2 += name1;
    cout << name2 << endl; // hello+hello
```

C++ string usage



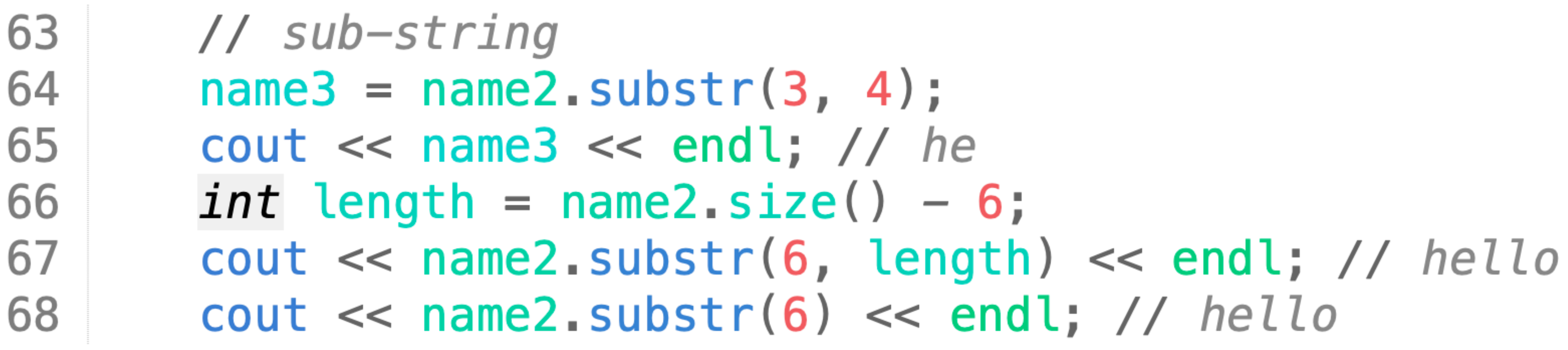
```
    // at, operator []
    name2 = "hello hello";
    cout << name2.at(4) << endl; // o
    cout << name2[4] << endl; // o
    name2[0] = 'H';
    name2[1] = 'E';
    name2[2] = 'L';
    name2.at(3) = 'L';
    name2.at(4) = 'O';
    cout << name2 << endl; // HELLO hello
```

C++ string usage



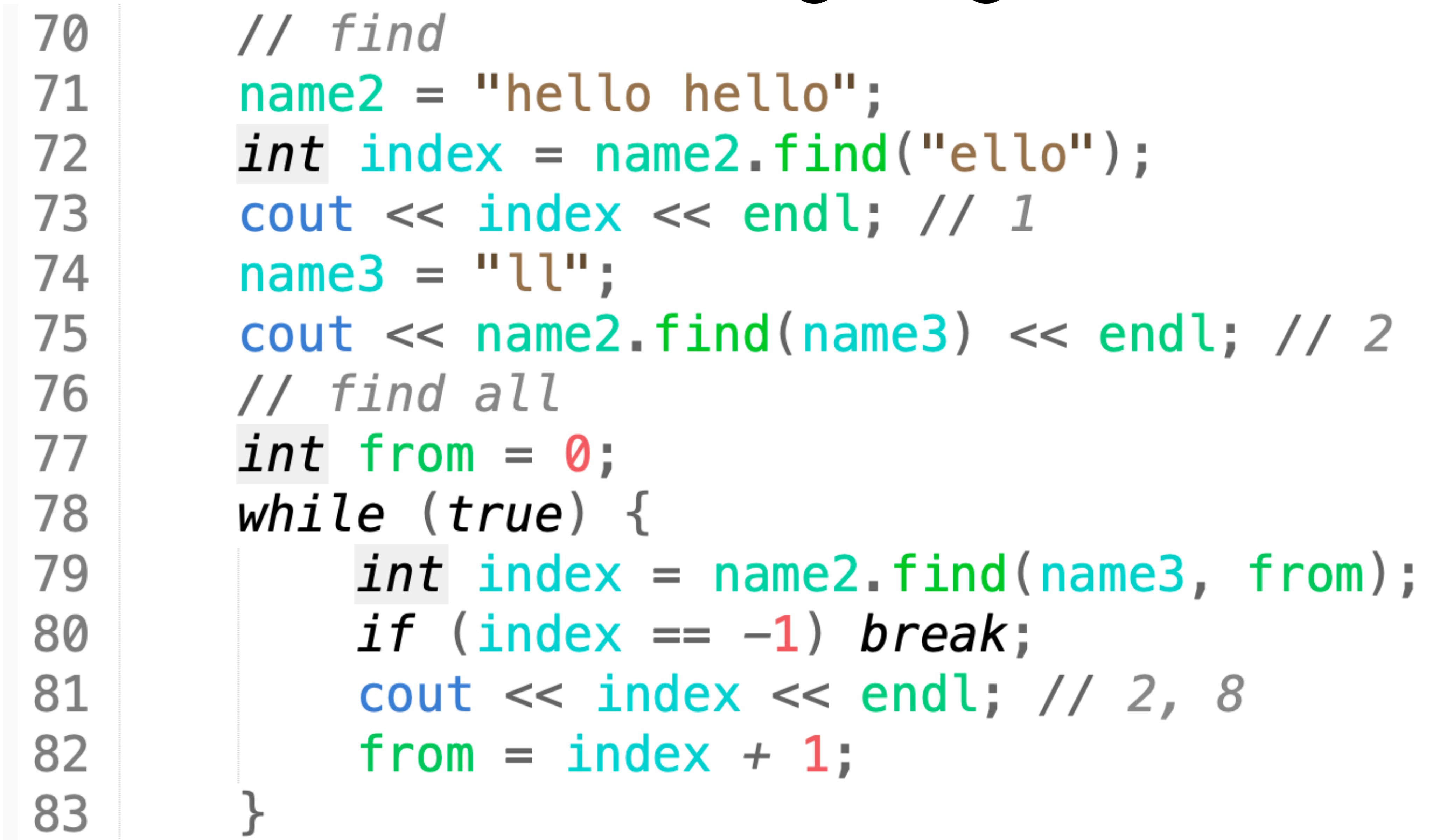
```
// compare, operator >
int cmp_res = name2.compare(name1);
cout << cmp_res << endl; // -32
bool cmp_res2 = (name2 > name1);
cout << cmp_res2 << endl; // 0: false
cout << (name2 < name1) << endl;
cout << (name2 <= name1) << endl;
cout << (name2 >= name1) << endl;
cout << (name2 != name1) << endl;
cout << (name2 == name1) << endl;
```

C++ string usage



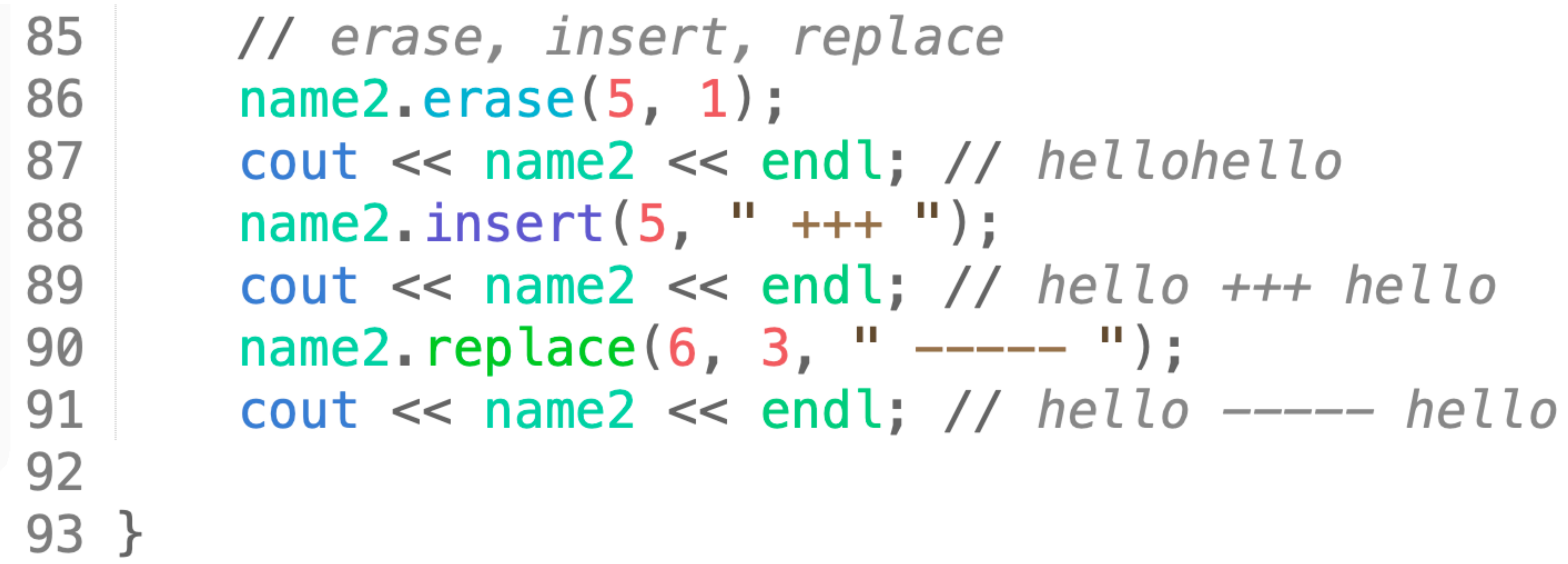
```cpp
// sub-string
name3 = name2.substr(3, 4);
cout << name3 << endl; // he
int length = name2.size() - 6;
cout << name2.substr(6, length) << endl; // hello
cout << name2.substr(6) << endl; // hello
```

C++ string usage



```cpp
// find
name2 = "hello hello";
int index = name2.find("ello");
cout << index << endl; // 1
name3 = "ll";
cout << name2.find(name3) << endl; // 2
// find all
int from = 0;
while (true) {
    int index = name2.find(name3, from);
    if (index == -1) break;
    cout << index << endl; // 2, 8
    from = index + 1;
}
```

C++ string usage

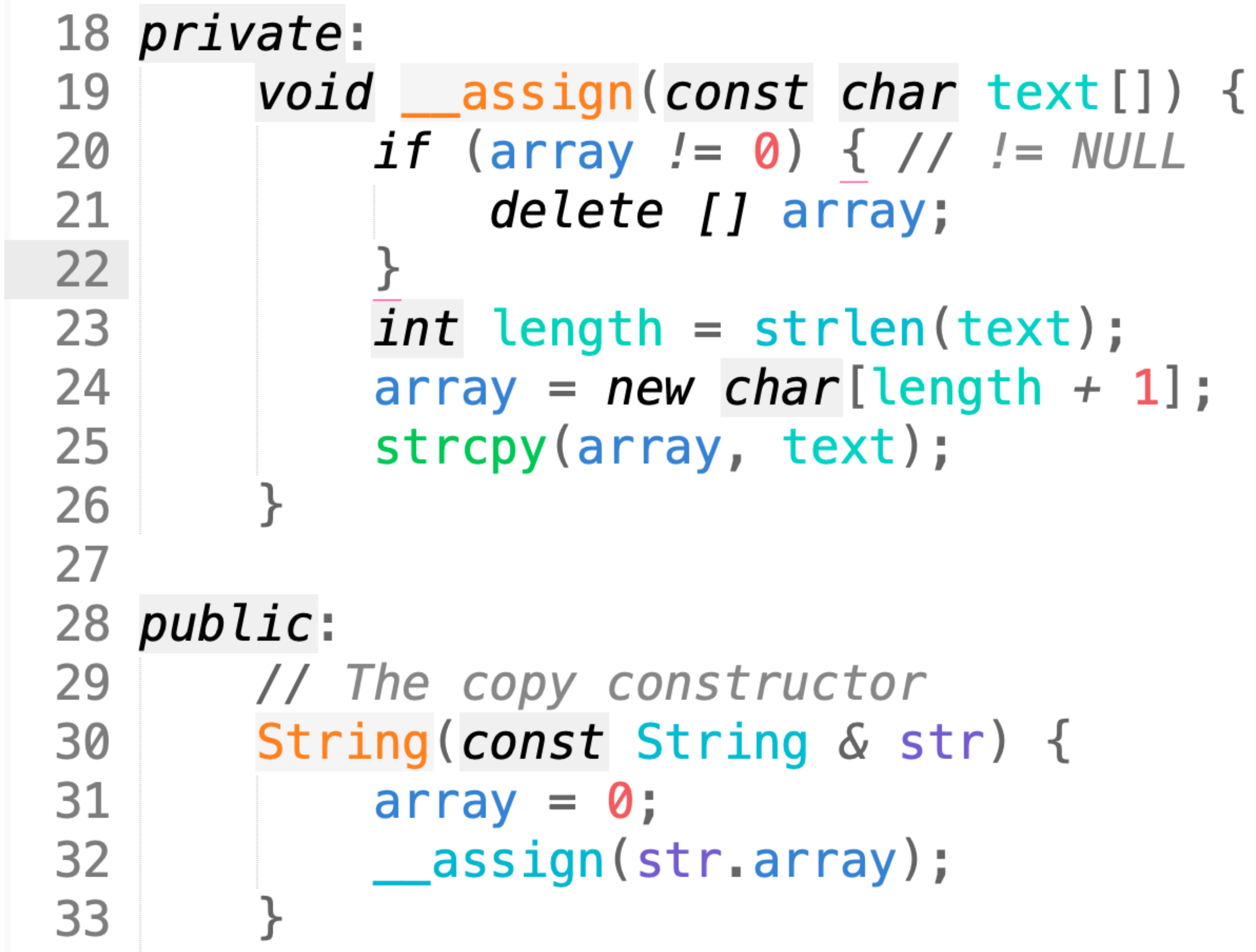


```cpp
    // erase, insert, replace
    name2.erase(5, 1);
    cout << name2 << endl; // hellohello
    name2.insert(5, " +++ ");
    cout << name2 << endl; // hello +++ hello
    name2.replace(6, 3, " ----- ");
    cout << name2 << endl; // hello ----- hello

}
```

Simple implementation of the C++ string class

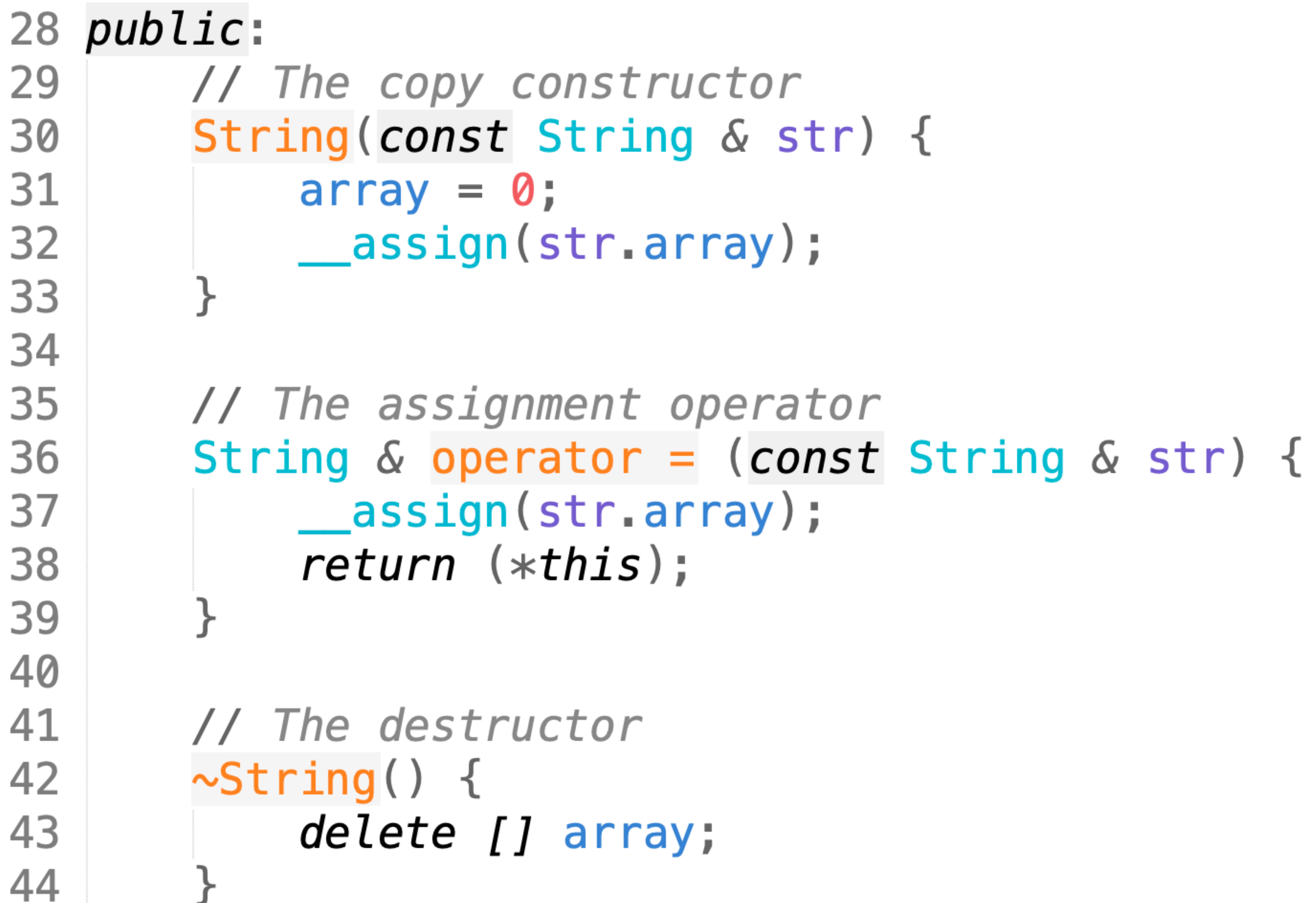
```cpp

#include <cstring> // string.h
#include <iostream> // will use ostream
using namespace std;

class String
{
private:
    char * array;

public:
    // The default constructor
    String() {
        array = new char[1];
        array[0] = '\0';
    }
```

Simple implementation of the C++ string class



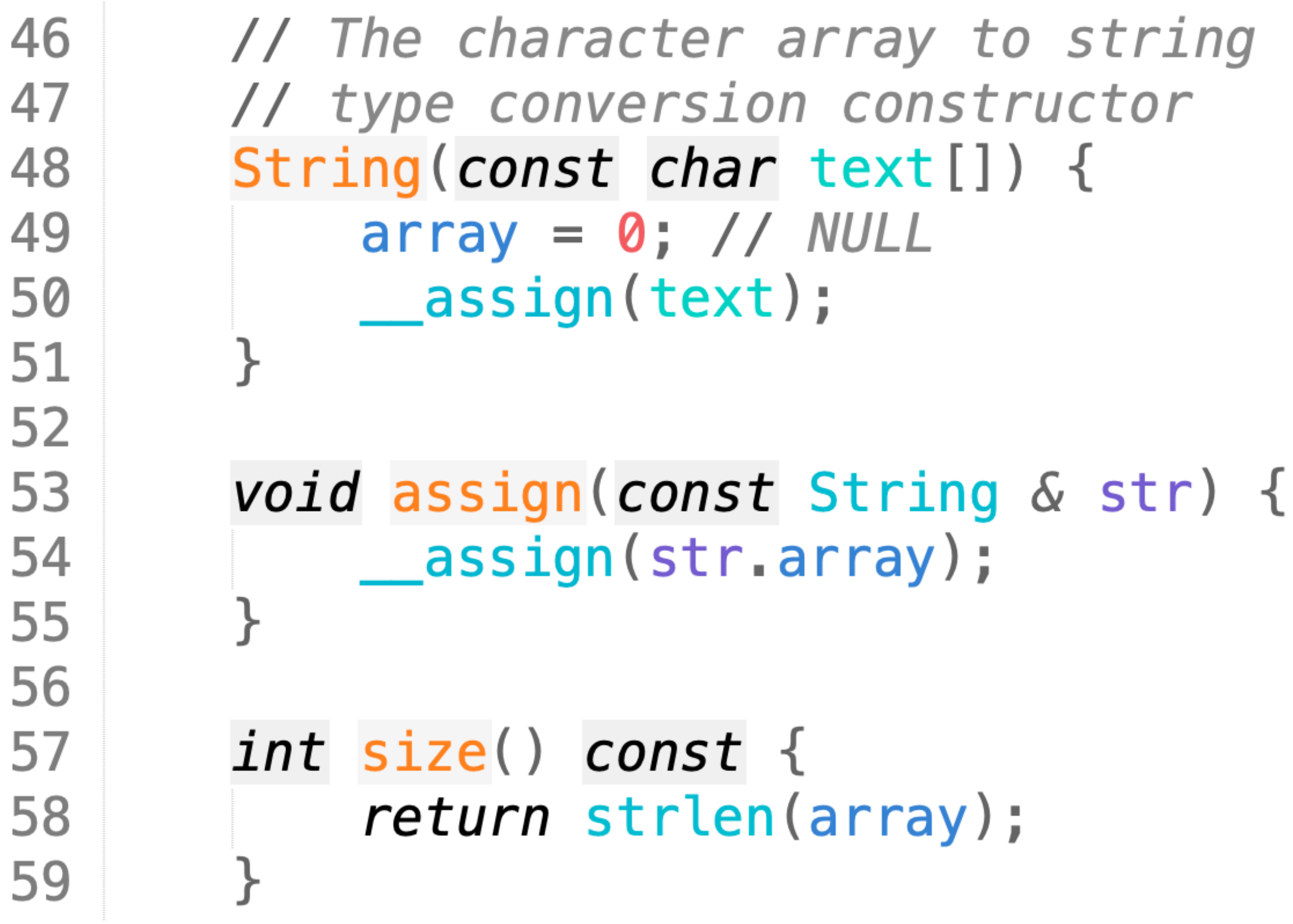
```
private:
    void __assign(const char text[]) {
        if (array != 0) { // != NULL
            delete [] array;
        }
        int length = strlen(text);
        array = new char[length + 1];
        strcpy(array, text);
    }

public:
    // The copy constructor
    String(const String & str) {
        array = 0;
        __assign(str.array);
    }
```

Simple implementation of the C++ string class



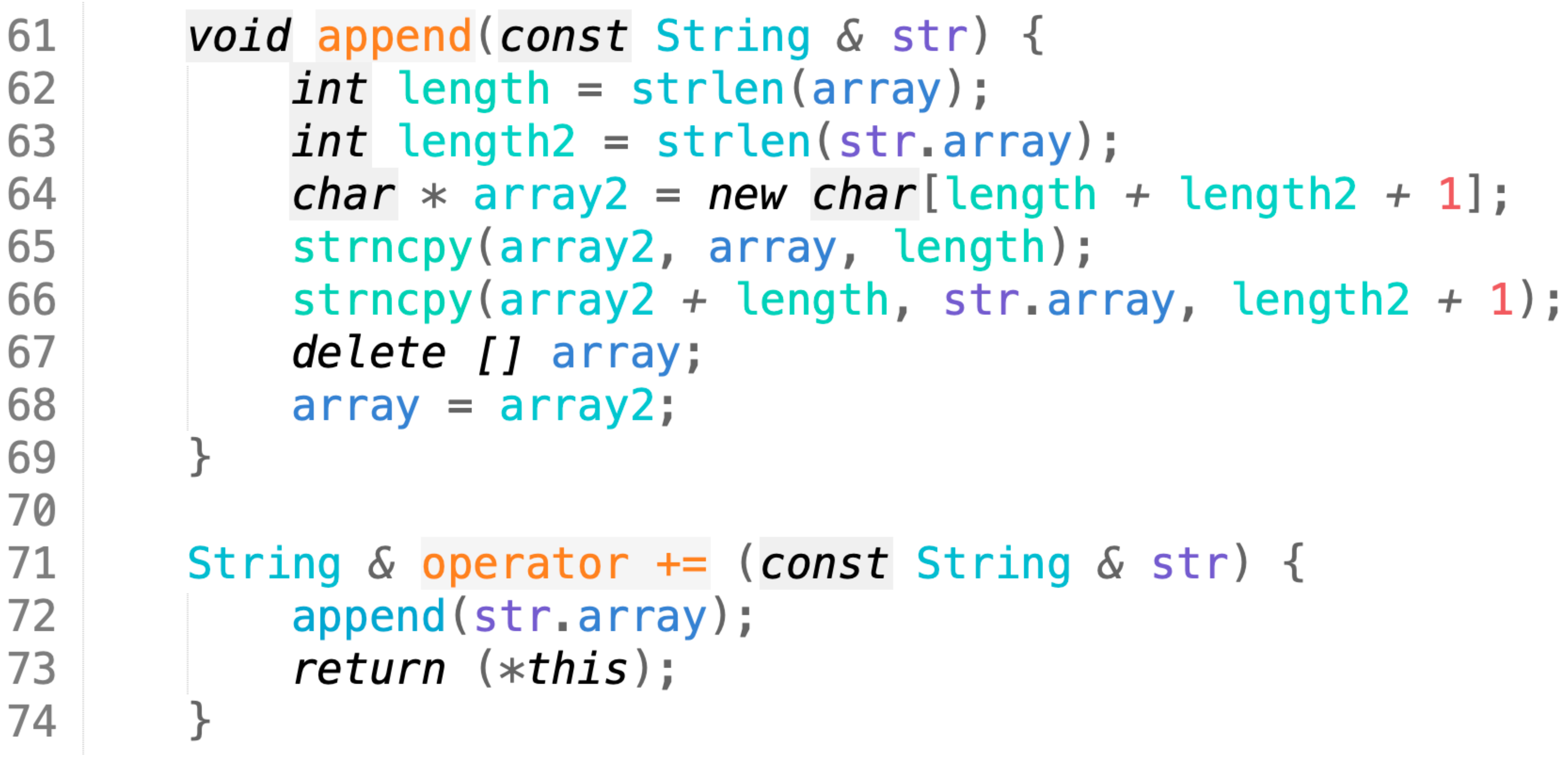
```
public:
    // The copy constructor
    String(const String & str) {
        array = 0;
        __assign(str.array);
    }

    // The assignment operator
    String & operator = (const String & str) {
        __assign(str.array);
        return (*this);
    }

    // The destructor
    ~String() {
        delete [] array;
    }
```

Simple implementation of the C++ string class



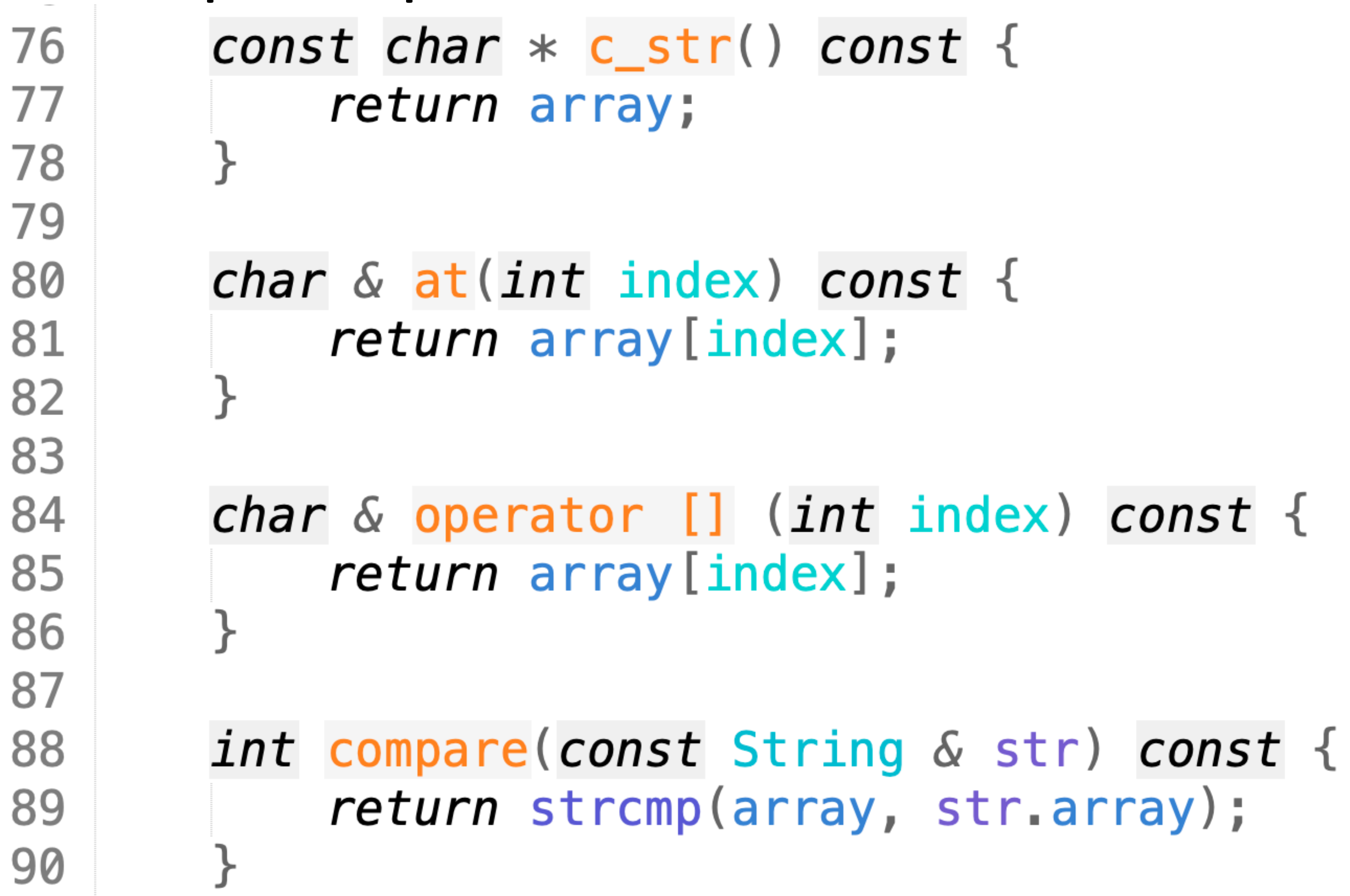
```cpp
    // The character array to string
    // type conversion constructor
    String(const char text[]) {
        array = 0; // NULL
        __assign(text);
    }

    void assign(const String & str) {
        __assign(str.array);
    }

    int size() const {
        return strlen(array);
    }
```

Simple implementation of the C++ string class



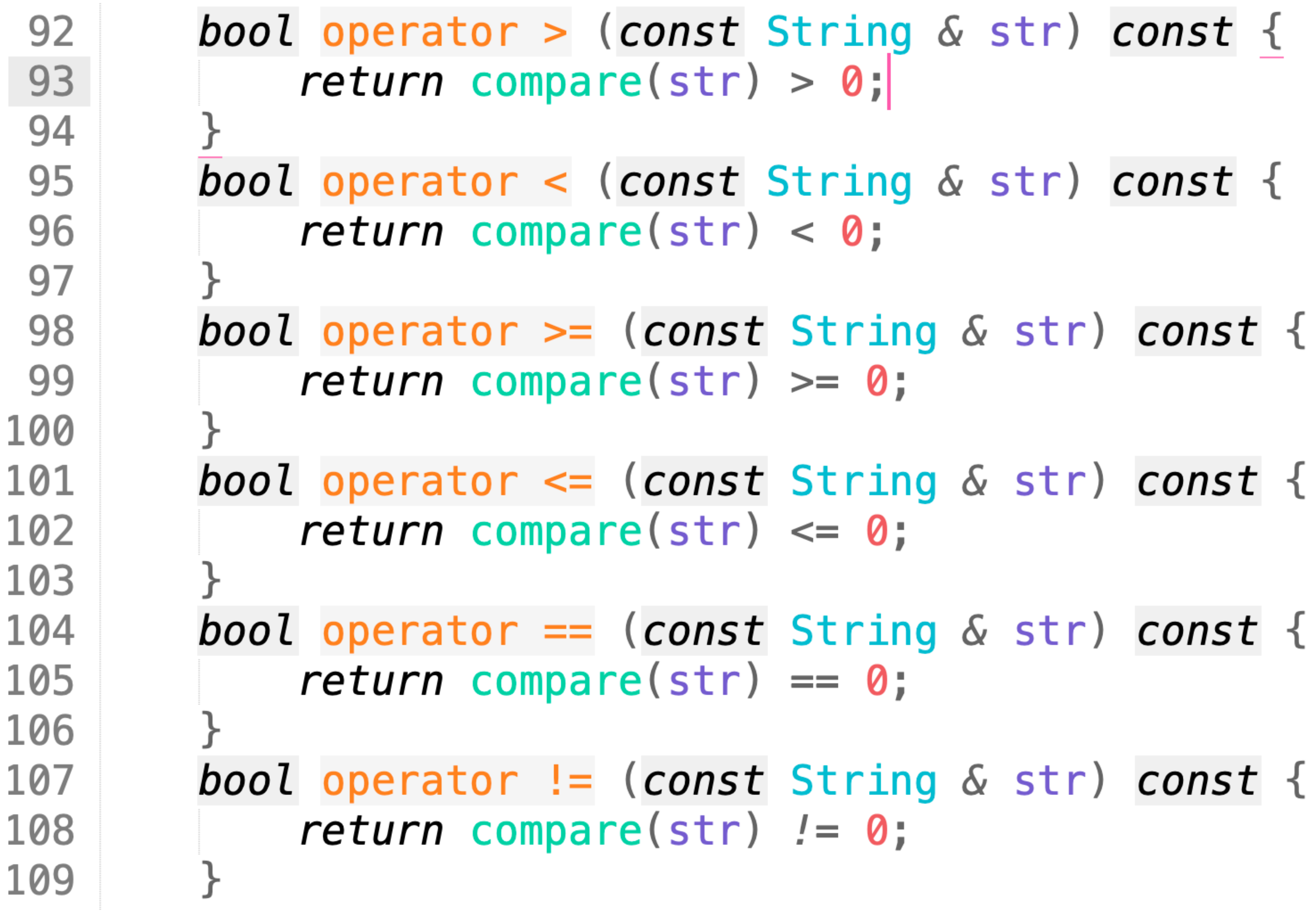
```
    void append(const String & str) {
        int length = strlen(array);
        int length2 = strlen(str.array);
        char * array2 = new char[length + length2 + 1];
        strncpy(array2, array, length);
        strncpy(array2 + length, str.array, length2 + 1);
        delete [] array;
        array = array2;
    }

    String & operator += (const String & str) {
        append(str.array);
        return (*this);
    }
```

Simple implementation of the C++ string class



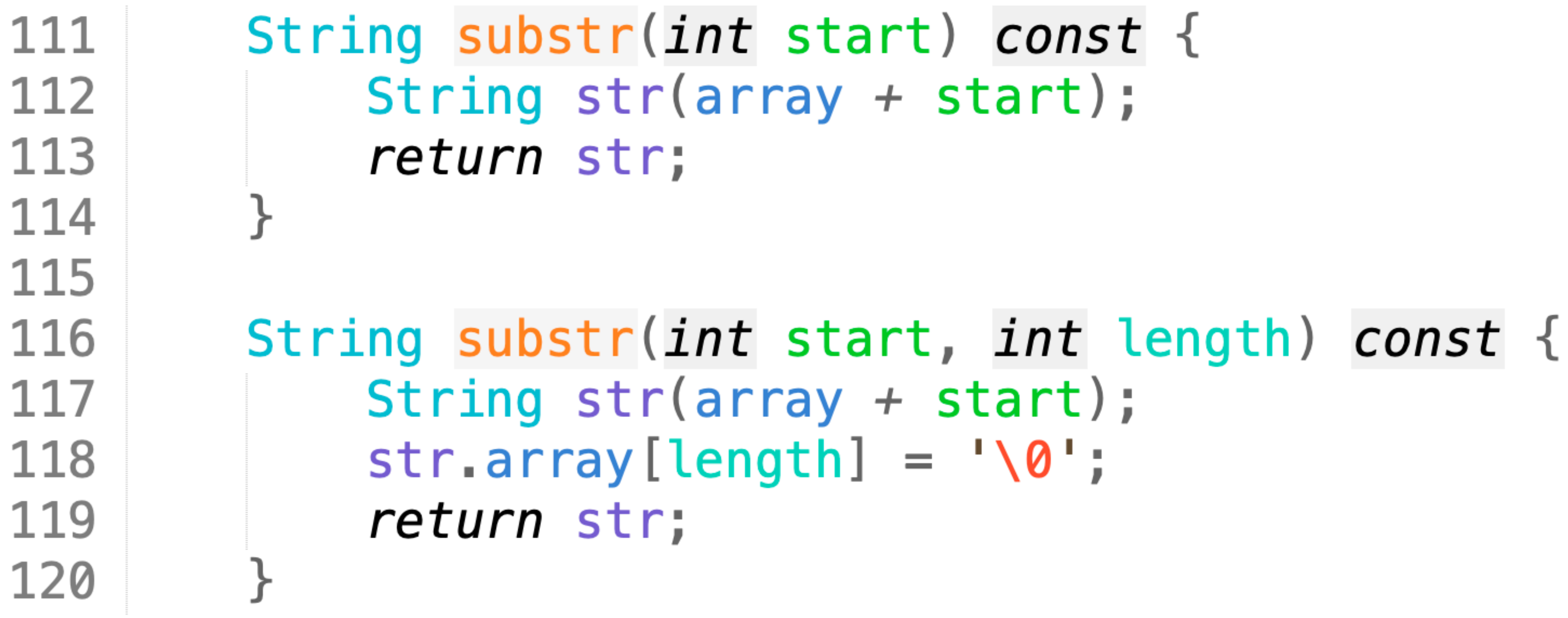
```cpp
    const char * c_str() const {
        return array;
    }

    char & at(int index) const {
        return array[index];
    }

    char & operator [] (int index) const {
        return array[index];
    }

    int compare(const String & str) const {
        return strcmp(array, str.array);
    }
```

Simple implementation of the C++ string class



```cpp
bool operator > (const String & str) const {
    return compare(str) > 0;
}
bool operator < (const String & str) const {
    return compare(str) < 0;
}
bool operator >= (const String & str) const {
    return compare(str) >= 0;
}
bool operator <= (const String & str) const {
    return compare(str) <= 0;
}
bool operator == (const String & str) const {
    return compare(str) == 0;
}
bool operator != (const String & str) const {
    return compare(str) != 0;
}
```

Simple implementation of the C++ string class

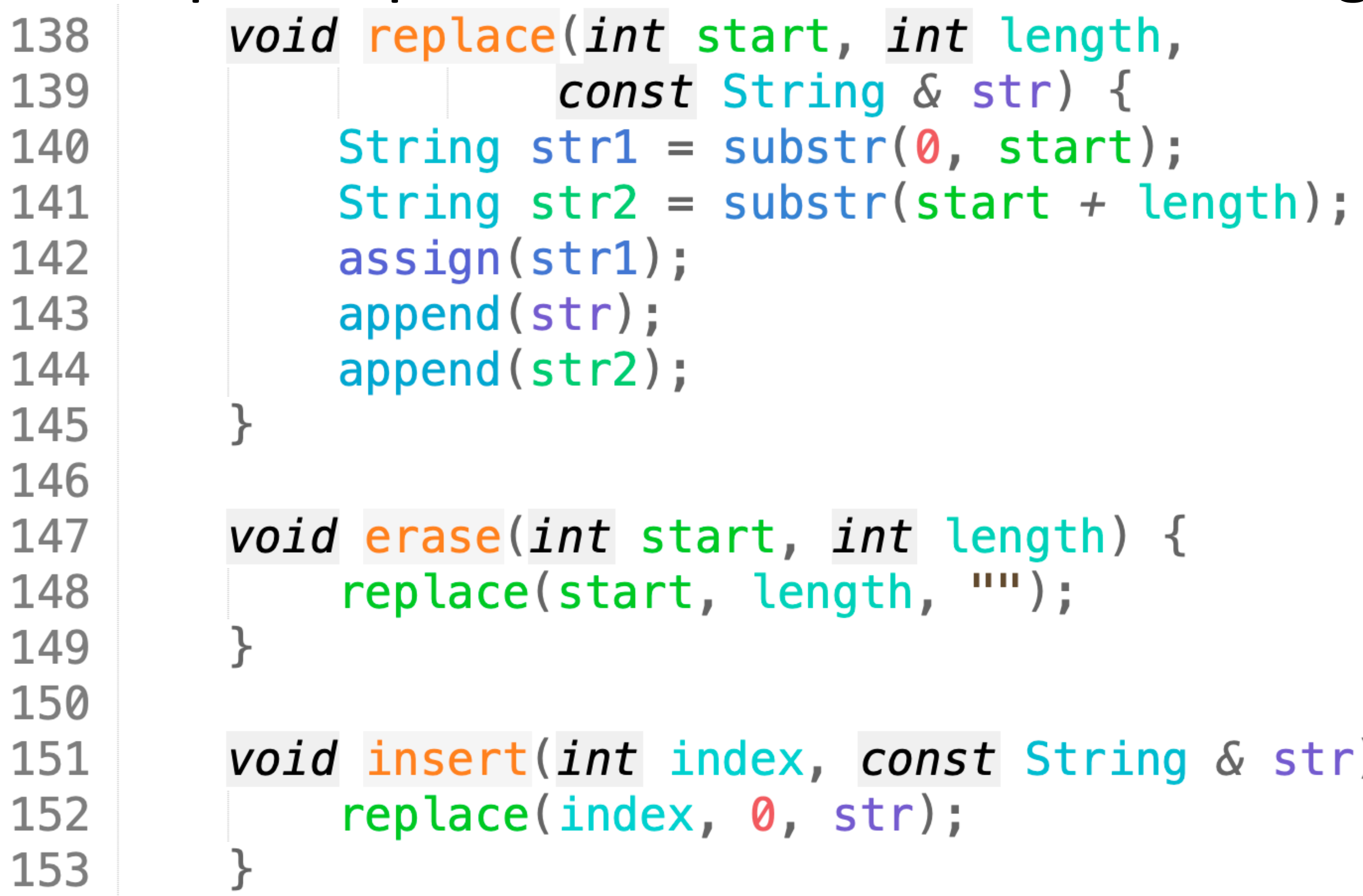


```
    String substr(int start) const {
        String str(array + start);
        return str;
    }

    String substr(int start, int length) const {
        String str(array + start);
        str.array[length] = '\0';
        return str;
    }
```

Simple implementation of the C++ string class

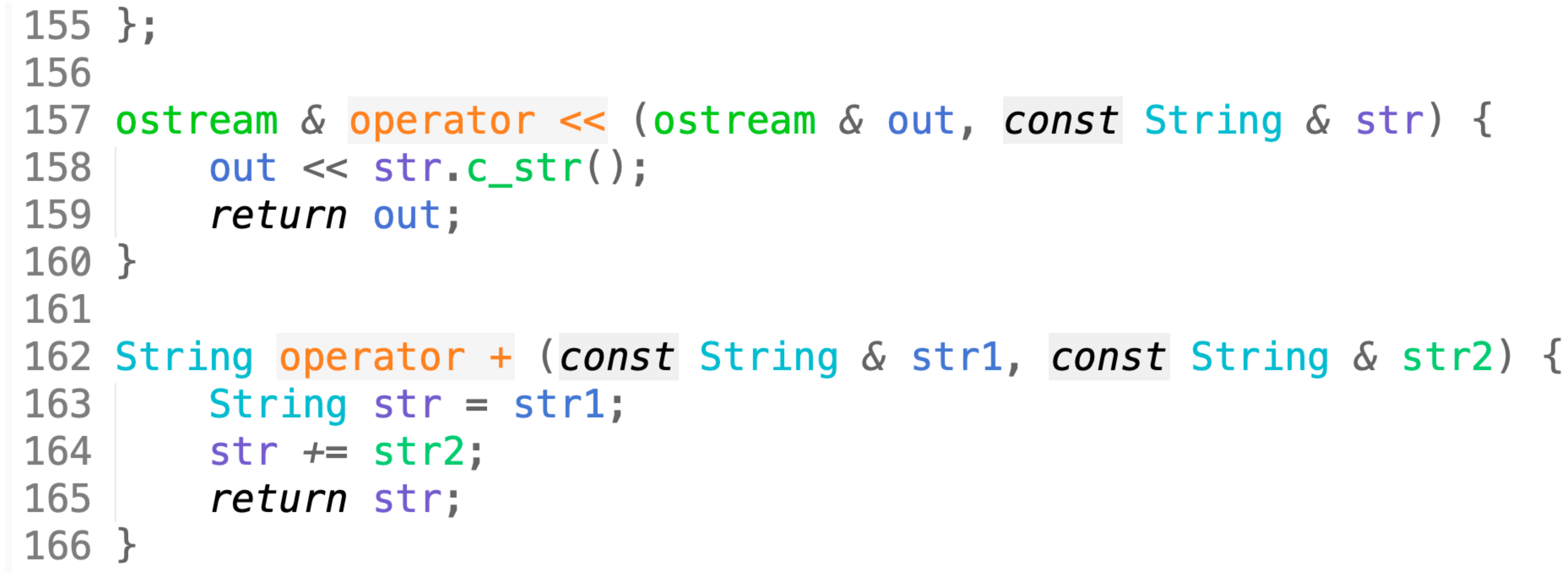
```cpp
    int find(const String & str, int start) const {
        int length = strlen(array);
        int length2 = strlen(str.array);
        int end = length - length2;
        for (int i = start; i <= end; ++ i) {
            if (strncmp(array + i, str.array, length2) == 0) {
                return i;
            }
        }
        return -1;
    }

    int find(const String & str) const {
        return find(str.array, 0);
    }
```

Simple implementation of the C++ string class



```
    void replace(int start, int length,
                 const String & str) {
        String str1 = substr(0, start);
        String str2 = substr(start + length);
        assign(str1);
        append(str);
        append(str2);
    }

    void erase(int start, int length) {
        replace(start, length, "");
    }

    void insert(int index, const String & str) {
        replace(index, 0, str);
    }
```

Simple implementation of the C++ string class



```
};

ostream & operator << (ostream & out, const String & str) {
    out << str.c_str();
    return out;
}

String operator + (const String & str1, const String & str2) {
    String str = str1;
    str += str2;
    return str;
}
```