

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

```
1
2 #include <iostream>
3 using namespace std;
4
5 int main() {
6     string name1("hello");
7     string name2 = "hi";
8     cout << name1.c_str() << endl; // hello
9     cout << name2.c_str() << endl; // hi
10
11     name2.assign("good");
12     name2.append("morning");
13     cout << name2.c_str() << endl; // goodmorning
```

C++ string usage

```
15 // operator << (ostream, string)
16 cout << name2 << endl; // goodmorning
17
18 // constructor: string(string)
19 string name3(name1);
20 // string name3 = name1;
21 cout << name3.size() << endl; // 5
22
```

C++ string usage

```
23 // assign, operator =  
24 name2.assign("hello");  
25 name2.assign(name1);  
26 name2 = "hello";  
27 name2 = name1;  
28 cout << name2 << endl; // hello
```

C++ string usage

```
30 // operator +
31 name2 = (name1 + " Right");
32 name2 = ("Left " + name2);
33 cout << name2 << endl; // Left hello Right
34
35 // operator +=
36 name2 = name1;
37 name2 += "+";
38 name2 += name1;
39 cout << name2 << endl; // hello+hello
```


C++ string usage

```
41 // at, operator []
42 name2 = "hello hello";
43 cout << name2.at(4) << endl; // o
44 cout << name2[4] << endl; // o
45 name2[0] = 'H';
46 name2[1] = 'E';
47 name2[2] = 'L';
48 name2.at(3) = 'L';
49 name2.at(4) = 'O';
50 cout << name2 << endl; // HELLO hello
```

C++ string usage

```
52 // compare, operator >
53 int cmp_res = name2.compare(name1);
54 cout << cmp_res << endl; // -32
55 bool cmp_res2 = (name2 > name1);
56 cout << cmp_res2 << endl; // 0: false
57 cout << (name2 < name1) << endl;
58 cout << (name2 <= name1) << endl;
59 cout << (name2 >= name1) << endl;
60 cout << (name2 != name1) << endl;
61 cout << (name2 == name1) << endl;
```

C++ string usage

```
63 // sub-string
64 name3 = name2.substr(3, 4);
65 cout << name3 << endl; // he
66 int length = name2.size() - 6;
67 cout << name2.substr(6, length) << endl; // hello
68 cout << name2.substr(6) << endl; // hello
```

C++ string usage

```
70 // find
71 name2 = "hello hello";
72 int index = name2.find("ello");
73 cout << index << endl; // 1
74 name3 = "ll";
75 cout << name2.find(name3) << endl; // 2
76 // find all
77 int from = 0;
78 while (true) {
79     int index = name2.find(name3, from);
80     if (index == -1) break;
81     cout << index << endl; // 2, 8
82     from = index + 1;
83 }
```

C++ string usage

```
85 // erase, insert, replace
86 name2.erase(5, 1);
87 cout << name2 << endl; // hellohello
88 name2.insert(5, " +++ ");
89 cout << name2 << endl; // hello +++ hello
90 name2.replace(6, 3, " ----- ");
91 cout << name2 << endl; // hello ----- hello
92
93 }
```

Simple implementation of the C++ string class

```
1
2 #include <cstring> // string.h
3 #include <iostream> // will use ostream
4 using namespace std;
5
6 class String
7 {
8     private:
9         char * array;
10
11     public:
12         // The default constructor
13         String() {
14             array = new char[1];
15             array[0] = '\0';
16         }
```

Simple implementation of the C++ string class

```
18 private:
19     void __assign(const char text[]) {
20         if (array != 0) { // != NULL
21             delete [] array;
22         }
23         int length = strlen(text);
24         array = new char[length + 1];
25         strcpy(array, text);
26     }
27
28 public:
29     // The copy constructor
30     String(const String & str) {
31         array = 0;
32         __assign(str.array);
33     }
```

Simple implementation of the C++ string class

```
28 public:
29     // The copy constructor
30     String(const String & str) {
31         array = 0;
32         __assign(str.array);
33     }
34
35     // The assignment operator
36     String & operator = (const String & str) {
37         __assign(str.array);
38         return (*this);
39     }
40
41     // The destructor
42     ~String() {
43         delete [] array;
44     }
```


Simple implementation of the C++ string class

```
46      // The character array to string  
47      // type conversion constructor  
48      String(const char text[]) {  
49          array = 0; // NULL  
50          __assign(text);  
51      }  
52  
53      void assign(const String & str) {  
54          __assign(str.array);  
55      }  
56  
57      int size() const {  
58          return strlen(array);  
59      }
```

Simple implementation of the C++ string class

```
61 void append(const String & str) {  
62     int length = strlen(array);  
63     int length2 = strlen(str.array);  
64     char * array2 = new char[length + length2 + 1];  
65     strncpy(array2, array, length);  
66     strncpy(array2 + length, str.array, length2 + 1);  
67     delete [] array;  
68     array = array2;  
69 }  
70  
71 String & operator += (const String & str) {  
72     append(str.array);  
73     return (*this);  
74 }
```

Simple implementation of the C++ string class

```
76     const char * c_str() const {  
77         return array;  
78     }  
79  
80     char & at(int index) const {  
81         return array[index];  
82     }  
83  
84     char & operator [] (int index) const {  
85         return array[index];  
86     }  
87  
88     int compare(const String & str) const {  
89         return strcmp(array, str.array);  
90     }
```

Simple implementation of the C++ string class

```
92     bool operator > (const String & str) const {  
93         return compare(str) > 0;  
94     }  
95     bool operator < (const String & str) const {  
96         return compare(str) < 0;  
97     }  
98     bool operator >= (const String & str) const {  
99         return compare(str) >= 0;  
100    }  
101    bool operator <= (const String & str) const {  
102        return compare(str) <= 0;  
103    }  
104    bool operator == (const String & str) const {  
105        return compare(str) == 0;  
106    }  
107    bool operator != (const String & str) const {  
108        return compare(str) != 0;  
109    }
```

Simple implementation of the C++ string class

```
111     String substr(int start) const {  
112         String str(array + start);  
113         return str;  
114     }  
115  
116     String substr(int start, int length) const {  
117         String str(array + start);  
118         str.array[length] = '\\0';  
119         return str;  
120     }
```

Simple implementation of the C++ string class

```
122     int find(const String & str, int start) const {
123         int length = strlen(array);
124         int length2 = strlen(str.array);
125         int end = length - length2;
126         for (int i = start; i <= end; ++ i) {
127             if (strncmp(array + i, str.array, length2) == 0) {
128                 return i;
129             }
130         }
131         return -1;
132     }
133
134     int find(const String & str) const {
135         return find(str.array, 0);
136     }
```

Simple implementation of the C++ string class

```
138 void replace(int start, int length,  
139             const String & str) {  
140     String str1 = substr(0, start);  
141     String str2 = substr(start + length);  
142     assign(str1);  
143     append(str);  
144     append(str2);  
145 }  
146  
147 void erase(int start, int length) {  
148     replace(start, length, "");  
149 }  
150  
151 void insert(int index, const String & str) {  
152     replace(index, 0, str);  
153 }
```

Simple implementation of the C++ string class

155 };

156

157 ostream & operator << (ostream & out, const String & str) {

158 out << str.c_str();

159 return out;

160 }

161

162 String operator + (const String & str1, const String & str2) {

163 String str = str1;

164 str += str2;

165 return str;

166 }