

39 | 串的顺序和链式存储结构：定长数组与动态数组

2023-05-12 王健伟 来自北京

《快速上手C++数据结构与算法》



你好，我是王健伟。

前面我带你一起学习了各种各样的排序算法。从这节课开始，我们就要进入到字符串的学习了。

字符串作为一种数据结构，在计算机科学领域也有着比较广泛的应用。比如在搜索引擎中搜索一个关键词、在文章或发言中过滤或屏蔽一些敏感词等。这些关键词、敏感词都属于字符串。

shikey.com 转载分享

提到这些，估计你就对它熟悉很多了。不过别着急，我们还是从一些基本的概念和术语学起。

串有哪些基本概念？

字符串简称**串**，是由零个或者多个字符组成的有限序列。计算机上非数值处理的对象通常指的就是串数据。在 C 语言中，针对串的处理函数常用的有这几个：strlen()、strcat()、

strcmp()、strcpy() 等。

看如下两行 C++ 风格的代码：

```
std::string mystr = "Hello World!";  
std::cout << mystr << std::endl;
```

上述代码行中用双引号括起来的 “Hello World! ”，就是一个串，在 C 或者 C++ 语言中，用双引号括起来的是串，用单引号括起来的是字符。其中 mystr 称为串名，串中字符的个数 n 称为串的长度。当 n=0 时的串称为空串。空串可以用希腊字母 Φ 表示。

这里有两个概念咱们看一下。一个是子串，也就是串中任意个连续的字符组成的子序列。当然，如果是零个字符，就叫做空串。还有一个叫主串，包含子串的串叫做主串。

字符在主串中的位置，也就是字符在主串中的序号。序号（位置编号）从 1 开始算起。比如字符 ‘e’ 在 “Hello World!” 这个字符串中的位置是 2。而空格字符 ‘ ’ 在 “Hello World!” 这个字符串中的位置是 6，注意，空格符也是一个正常的字符。

而子串在主串中的位置，也就是子串的第一个字符在主串中的序号。

可以看到，串是一种特殊的线性表，数据元素之间呈线性关系，或者说相邻字符之间有前趋和后继关系。串中的数据对象一般都会限定为字符集，比如中文字符、英文字符、数字字符、标点符号等等。

串的基本操作比如增加、删除、修改、查询等一般都是以子串为操作对象，因为人类语言通常针对多个字符组成的字符序列才有现实意义。比如在搜索引擎中输入一个串（子串）就可以查询到包含了给定子串的各种信息。

串的基本操作包括串赋值、串拷贝（串复制）、判断空串、求长度、串连接、串比较、获取串的一部分内容（求子串）、串插入、串删除、串清空、定位（求某个串在另一个串中第一次出现的位置）等等。不过，不管对串进行什么样的操作，首先要解决的是串的存储问题，那么这节课咱们先说说串是如何存储的。


串的顺序存储结构

串的存储结构主要分为两种——顺序存储结构和链式存储结构。其中串的顺序存储结构是用一块地址连续的内存保存串中的字符序列。

定长数组（静态数组）存储结构及基本操作的实现

众所周知，在 C 语言中，用 ‘\0’ 作为串结束标记。在后续的实现代码中，我还是沿用这个标记作为串结束标记。当然，如果你愿意也可以专门拿出位置空间保存串长度信息。

对于串的各种基本操作都是比较简单的，写代码时要注意细节，考虑周到。另外值得一提的是程序的写法有很多种，不必拘泥于某一种，关键是理解代码的实现意图。这里直接看实现代码。

 复制代码

```
1 #define MAX_LEN 250 //最大字符串长度（定长）
2 //采用定长数组存储结构
3 class MySString
4 {
5 public:
6     MySString()//构造函数
7     {
8         ch[0] = '\0'; //字符串结束标记，其实'\0'就是数字0，所以写成ch[0] = 0;也没问题
9         length = 0; //字符串长度
10    }
11    //串赋值
12    void StrAssign(const char* pcontent)
13    {
14        size_t iLen = strlen(pcontent);
15        if (iLen >= MAX_LEN) //内容太长，容纳不下，字符串存储中要给字符串结束标记'\0'留出位置
16            return;
17
18        for (int i = 0, j = 0; i < iLen; ++i)
19        {
20            ch[i] = pcontent[i];
21        } //end for
22        ch[iLen] = '\0'; //设置字符串结束标记，该标记不计入字符串长度中
23        length = iLen; //记录字符串长度
24    }
25
26    //串拷贝（串复制）
27    void StrCopy(const MySString &tmpstr)
```

```

28 {
29     for (int i = 0; i < tmpstr.length; ++i)
30     {
31         ch[i] = tmpstr.ch[i];
32     } //end for
33     length = tmpstr.length;
34     ch[length] = '\0';
35 }
36
37 //判断空串
38 bool IfStrEmpty()
39 {
40     if (ch[0] == '\0')
41         return true;
42     return false;
43 }
44
45 //串比较, 比较其实就是逐个比较两个字符串中每个字符的ASCII码
46 //结果大于返回1, 等于返回0, 小于返回-1
47 int StrCmp(const MySString& tmpstr)
48 {
49     if (length == 0 && tmpstr.length == 0) //两个字符串都是空的, 相等
50         return 0;
51
52     const char* p1 = ch;
53     const char* p2 = tmpstr.ch;
54
55     int result = 0;
56     while (*p1 != '\0' && *p2 != '\0')
57     {
58         result = (*p1) - (*p2);
59         if (result != 0)
60         {
61             if (result > 0)
62                 return 1;
63             else
64                 return -1;
65         }
66         p1++;
67         p2++;
68     } //end while
69
70     if (*p1 == '\0' && *p2 == '\0') //长度相同且内容相等
71         return 0;
72
73     //能走到下边流程的都是两个字符串一个长一个短, 但长的和短的字符串的前面内容相同, 比如字符串"a
74     if (*p1 == '\0') //p1小, 因为长度少
75         return -1;
76

```

shikekey.com转载分享

```

77     //else if (*p2 == '\0')
78     return 1;
79 }
80
81 //串连接
82 bool StrCat(const MySString& tmpstr)
83 {
84     if (length + tmpstr.length >= MAX_LEN) //空间不够保存不下, 这里直接返回false以通知
85         return false;
86
87     int idx = 0;
88     size_t i;
89     for (i = length; i < (length + tmpstr.length); ++i)
90     {
91         ch[i] = tmpstr.ch[idx++];
92     }
93     ch[i] = '\0'; //字符串结束标记
94     length += tmpstr.length;
95     return true;
96 }
97
98 //获取串的一部分内容 (求子串)
99 void SubString(MySString& resultstr, int pos, int len) //求得的子串给到resultstr。
100 {
101     //注意pos位置从0开始计算
102     if(pos < 0 || (pos + 1) > length || len <= 0) //pos位置要合法, len长度值要合法
103         return;
104
105     int icount = 0;
106     while(true)
107     {
108         resultstr.ch[icount] = ch[pos+ icount];
109         icount++;
110         if (icount == len) //截取够数量了
111             break;
112         if (ch[pos + icount] == '\0') //到主串末尾了, 不够截取, 截取多少算多少, 直接跳出循环
113             break;
114     } //end while
115     resultstr.length = icount;
116     resultstr.ch[resultstr.length] = '\0';
117     return;
118 }
119
120 //串插入
121 //在当前串的pos位置 (从0开始计算), 插入子串substr
122 void StrInsert(int pos, const MySString& substr)
123 {
124     if (pos < 0 || pos > length) //插入位置不合法
125         return;

```



```

126
127     if (length + substr.length >= MAX_LEN) //容纳不下插入的新内容，则直接返回
128         return;
129
130     //把原来的必须的内容向后挪动
131     int i = (int)(length - 1); //i为int类型，这样就可以为负数，保证下面这个for循环可以正确
132     for (; i >= pos; --i)
133     {
134         ch[i + substr.length] = ch[i];
135     }
136     //把子串插入进来
137     for (size_t i = 0; i < substr.length; ++i)
138     {
139         ch[pos + i] = substr.ch[i];
140     }
141     length += substr.length;
142     ch[length] = '\0';
143     return;
144 }
145
146 //串删除
147 //在当前串的pos位置（从0开始计算），删除len个字符
148 void StrDelete(int pos, int len)
149 {
150     //注意pos位置从0开始计算
151     if (pos < 0 || (pos + 1) > length || len <= 0) //pos位置要合法，len长度值要合法
152         return;
153
154     if (pos + len > length)
155     {
156         //要删除的字符太多，串中没那么多可删的字符
157         len = int(length - pos); //只能删除这么多
158     }
159
160     //把剩余的字符串搬位置（向左搬）
161     for (int i = pos; i < length; ++i)
162     {
163         ch[i] = ch[i + len];
164     } //end for
165
166     length = length - len;
167     ch[length] = '\0';
168     return;
169 }
170
171 //串清空
172 void StrClear()
173 {
174     ch[0] = '\0';


```

```

175     length = 0;
176     return;
177 }
178
179 public:
180     //显示字符串内容
181     void DispContent()
182     {
183         cout << ch << endl;
184     }
185
186 public:
187     char ch[MAX_LEN]; //串内容。每个位置保存一个字符
188     size_t length; //串实际长度，专门引入该变量保存，提高程序运行效率
189 }

```

在 main 主函数中，加入下面的代码。

 复制代码

```

1 //串赋值
2 MySString mys;
3 mys.StrAssign("我爱你中国! ");
4 mys.DispContent();
5
6 //串拷贝（串复制）、判断空串
7 MySString mys2;
8 cout <<"mys2为空吗? "<< mys2.IfStrEmpty() << endl;
9 mys2.StrCopy(mys);
10 mys2.DispContent();
11 cout <<"mys2为空吗? "<< mys2.IfStrEmpty() << endl;
12
13 //串比较
14 MySString mys3,mys4;
15 mys3.StrAssign("abc");
16 mys4.StrAssign("xyz");
17 cout <<"mys3和mys4字符串的比较结果为: "<< mys3.StrCmp(mys4) << endl;
18
19 //串连接
20 MySString mys5;
21 mys5.StrAssign("Hello China!");
22 MySString mys6;
23 mys6.StrAssign("Hello this World!");
24 mys6.StrCat(mys5);
25 cout <<"mys6和mys5连接的结果为: "<< mys6.ch << endl;
26
27 //获取串的一部分内容（求子串）

```

```

28 MySString mys7;
29 mys6.SubString(mys7, 0, 12); //子串放入mys7中
30 cout <<"子串mys7的内容是: "<< mys7.ch << endl;
31
32 //串插入 (在当前串的pos位置 (从0开始计算) , 插入子串substr)
33 MySString mys8;
34 mys8.StrAssign("我爱北京, 我爱中国! ");
35 mys5.StrInsert(12, mys8);
36 cout <<"插入新内容后的mys5串内容是: "<< mys5.ch << endl;
37
38 //串删除, 在当前串的pos位置 (从0开始计算) , 删除len个字符
39 MySString mys9;
40 mys9.StrAssign("Hello China!");
41 mys9.StrDelete(1, 10);
42 cout <<"删除部分内容后的mys9串内容是: "<< mys9.ch << endl;
43
44 //串清空
45 mys9.StrClear();
46 cout <<"清空内容后的mys9串内容是: "<< mys9.ch << endl;

```

执行结果如下:

我爱你中国!

mys2为空吗? 1

我爱你中国!

mys2为空吗? 0

mys3和mys4字符串的比较结果为: -1

mys6和mys5连接的结果为: Hello this World!Hello China!

子串mys7的内容是: Hello this W


插入新内容后的mys5串内容是: Hello China!我爱北京, 我爱中国!

删除部分内容后的mys9串内容是: H!

清空内容后的mys9串内容是:

动态数组（堆中分配内存）存储结构及基本操作的实现

采用定长数组存储结构存储串的方式虽然编码方便，但缺乏灵活性。所以引入了在堆中分配内存来保存串，实现不复杂，但要写好需要比较细心，下面是实现代码。

 复制代码

```

1 //采用堆中分配内存的存储结构
2 class MyHString//H表示Heap (堆)

```



```
3 {
4 public:
5     MyHString()//构造函数
6     {
7         ch = nullptr;
8         length = 0;
9     }
10    ~MyHString()//析构函数
11    {
12        if (length > 0)
13            delete[] ch;
14    }
15
16    //串赋值
17    void StrAssign(const char* pcontent)
18    {
19        size_t iLen = strlen(pcontent);
20
21        if (length > 0)
22            delete[] ch;
23
24        ch = new char[iLen];
25
26        //拷贝字符串
27        for (int i = 0; i < iLen; ++i)
28        {
29            ch[i] = pcontent[i];
30        } //end for
31        length = iLen;
32    }
33
34    //串拷贝 (串复制)
35    void StrCopy(const MyHString& tmpstr)
36    {
37        if (length > 0)
38        {
39            delete[] ch;
40        }
41        ch = new char[tmpstr.length];
42        for (int i = 0; i < tmpstr.length; ++i)
43        {
44            ch[i] = tmpstr.ch[i];
45        }
46        length = tmpstr.length;
47        return;
48    }
49
50    //判断空串
51    bool IfStrEmpty()
```

```

52 {
53     if (length == 0)
54         return true;
55     return false;
56 }
57
58 //串比较, 比较其实就是逐个比较两个字符串中每个字符的ASCII码
59 //结果 大于返回1, 等于返回0, 小于返回-1
60 int StrCmp(const MyHString& tmpstr)
61 {
62     if (length == 0 && tmpstr.length == 0) //两个字符串都是空的, 相等
63         return 0;
64
65     const char* p1 = ch;
66     const char* p2 = tmpstr.ch;
67
68     int result = 0;
69     int i = 0;
70     int j = 0;
71     while (i < length && j < tmpstr.length)
72     {
73         result = ch[i] - tmpstr.ch[j];
74         if (result != 0)
75         {
76             if (result > 0)
77                 return 1;
78             else
79                 return -1;
80         }
81         i++;
82         j++;
83     } //end while
84
85     if(i >= length && j >= tmpstr.length)//长度相同且内容相等
86         return 0;
87
88     //能走到下边流程的都是两个字符串一个长一个短, 但长的和短的字符串的前面内容相同, 比如字符串"
89     if (i >= length) //p1小, 因为长度少
90         return -1;
91
92     return 1;
93 }
94
95 //串连接
96 void StrCat(const MyHString& tmpstr)
97 {
98     if (tmpstr.length <= 0) //目标是空串, 无须连接
99         return;
100

```

```

101     char* tmp = new char [length + tmpstr.length];
102     for (int i = 0; i < length; ++i)
103     {
104         tmp[i] = ch[i];
105     }
106     for (int i = 0; i < tmpstr.length; ++i)
107     {
108         tmp[i + length] = tmpstr.ch[i];
109     }
110     if (length > 0) //原来内存释放掉
111         delete[] ch;
112     ch = tmp;
113     length = length + tmpstr.length;
114     return;
115 }
116
117 //获取串的一部分内容 (求子串)
118 //求得的子串给到resultstr。 pos: 从该位置开始[注意位置从0开始计算], 截取len个字符
119 void SubString(MyHString& resultstr, int pos, int len)
120 {
121     //注意pos位置从0开始计算
122     if (pos < 0 || (pos + 1) > length || len <= 0) //pos位置要合法, len长度值要合法
123         return;
124
125     if (resultstr.length > 0)
126         delete[] resultstr.ch;
127
128     resultstr.ch = new char[len];
129     int icount = 0;
130     while (true)
131     {
132         resultstr.ch[icount] = ch[pos + icount];
133         icount++;
134         if (icount == len) //截取够数量了
135             break;
136         if (pos + icount >= length) //到主串末尾了, 不够截取, 截取多少算多少, 直接跳出循环
137             break;
138     } //end while
139     resultstr.length = icount;
140     return;
141 }
142
143 //串插入
144 //在当前串的pos位置 (从0开始计算), 插入子串substr
145 void StrInsert(int pos, const MyHString& substr)
146 {
147     if (pos < 0 || pos > length) //插入位置不合法
148         return;
149

```

```

150     char* tmp = new char[length + substr.length];
151     for (int i = 0; i < length; ++i) //把原来的数据先拷贝到新位置去
152     {
153         tmp[i] = ch[i];
154     }
155
156     if (length > 0) //先把原来的内存释放了
157         delete[] ch;
158
159     ch = tmp;
160
161     //把原来的必须的内容向后挪动
162     int i = (int)(length - 1); //i为int类型，这样就可以为负数，保证下面这个for循环可以正确
163     for (; i >= pos; --i)
164     {
165         ch[i + substr.length] = ch[i];
166     }
167     //把子串插入进来
168     for (size_t i = 0; i < substr.length; ++i)
169     {
170         ch[pos + i] = substr.ch[i];
171     }
172     length += substr.length;
173     return;
174 }
175
176 //串删除
177 //在当前串的pos位置（从0开始计算），删除len个字符
178 void StrDelete(int pos, int len)
179 {
180     //注意pos位置从0开始计算
181     if (pos < 0 || (pos + 1) > length || len <= 0) //pos位置要合法，len长度值要合法
182         return;
183
184     if (pos + len > length)
185     {
186         //要删除的字符太多，串中没那么多可删的字符
187         len = (int)(length - pos); //只能删除这么多
188     }
189
190     //把剩余的字符串搬位置（向左搬）
191     for (int i = pos; i < length; ++i)
192     {
193         ch[i] = ch[i + len];
194     } //end for
195
196     length = length - len;
197     return;
198 }


```

```

199
200 //串清空
201 void StrClear()
202 {
203     if (length > 0)
204         delete[] ch;
205     length = 0;
206     return;
207 }
208
209 public:
210 //显示字符串内容
211 void DispContent()
212 {
213     for (int i = 0; i < length; ++i)
214     {
215         cout << ch[i];
216     }
217     cout << endl; //这里可以换一下行
218 }
219
220 public:
221 char *ch; //空间用new动态分配
222 size_t length; //串实际长度
223 };

```

在 main 主函数中，注释掉以往的代码，新增下面的测试代码。

 复制代码

```

1 //串赋值
2 MyHString mys;
3 mys.StrAssign("我爱你中国! ");
4 mys.DispContent();
5
6 //串拷贝（串复制）、判断空串
7 MyHString mys2;
8 cout <<"mys2为空吗? "<< mys2.IfStrEmpty() << endl;
9 mys2.StrCopy(mys);
10 mys2.DispContent();
11 cout <<"mys2为空吗? "<< mys2.IfStrEmpty() << endl;
12 mys.StrAssign("我爱你中国, 我爱你中国人! ");
13 mys2.StrCopy(mys);
14 mys2.DispContent();
15
16 //串比较
17 MyHString mys3, mys4;

```

```
18 mys3.StrAssign("abc");
19 mys4.StrAssign("xyz");
20 cout <<"mys3和mys4字符串的比较结果为: "<< mys3.StrCmp(mys4) << endl;
21
22 //串连接
23 MyHString mys5;
24 mys5.StrAssign("Hello China!");
25 MyHString mys6;
26 mys6.StrAssign("Hello this World!");
27 mys6.StrCat(mys5);
28 cout <<"mys6和mys5连接的结果为: ";
29 mys6.DispContent();
30
31 //获取串的一部分内容 (求子串)
32 MyHString mys7;
33 mys7.StrAssign("abcdefghijklmnopqrstuvwxyz!");
34 mys6.SubString(mys7, 2, 12); //子串放入mys7中
35 cout <<"子串mys7的内容是: ";
36 mys7.DispContent();
37
38 //串插入 (在当前串的pos位置 (从0开始计算) , 插入子串substr)
39 MyHString mys8, mys82;
40 mys8.StrAssign("我爱北京, 我爱中国! ");
41 mys82.StrInsert(0, mys8);
42 cout <<"插入新内容后的mys82串内容是: ";
43 mys82.DispContent();
44
45 //串删除, 在当前串的pos位置 (从0开始计算) , 删除len个字符
46 MyHString mys9;
47 mys9.StrAssign("Hello China!");
48 mys9.StrDelete(1, 11);
49 cout <<"删除部分内容后的mys9串内容是: ";
50 mys9.DispContent();
51
52 //串清空
53 mys9.StrClear();
54 cout <<"清空内容后的mys9串内容是: ";
55 mys9.DispContent();
```

shikey.com转载分享

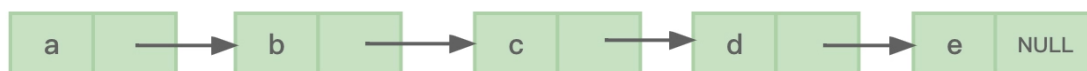
执行结果如下:


```
我爱你中国!
mys2为空吗? 1
我爱你中国!
mys2为空吗? 0
我爱你中国, 我爱你中国人!
mys3和mys4字符串的比较结果为: -1
mys6和mys5连接的结果为: Hello this World!Hello China!
子串mys7的内容是: llo this Wor
插入新内容后的mys8串内容是: 我爱北京, 我爱中国!
删除部分内容后的mys9串内容是: H
清空内容后的mys9串内容是:
```

上述代码其实是有改进空间的。比如在分配 ch（串）所占内存空间时可以记录 new 出的内存大小，当进行字符串赋值、拷贝、连接等操作时，如果空间足够则不需要 delete 原有内存而是直接用这块内存实现相应功能，有兴趣可以自行改进代码。

串的链式存储结构

串的链式存储结构会使用到链表。比如字符串“abcde”，采用链式存储结构可能会如图 1 所示：



极客时间

图1 串的链式存储结构示意图

可以看到，每个链的节点中保存着一个字符。那么节点可以像下面这样定义：

shikey.com转载分享

复制代码

```
1 //字符节点（串的链式存储）
2 struct StringNode
3 {
4     char ch;
5     StringNode* next; //指针域，指向下一个同类型（和本节点类型相同）节点
6 };
```

图 1 的问题是每个节点保存一个字符，而一个字符只占 1 个字节，但一个 next 指针域在 32 位平台却占了 4 个字节，这种存储方式的存储密度非常低，因为实际有用的信息占的内存比例太小，造成了内存空间的巨大浪费。改进方式是让每个链的节点中保存多个字符比如保存 4 个字符，下面是改进后的字符节点。

复制代码

```
1 //字符节点（串的链式存储）
2 struct StringNode
3 {
4     char ch[4];
5     StringNode* next; //指针域，指向下一个同类型（和本节点类型相同）节点
6 };
```

此时，每个节点中实际有用的信息占的内存比例会提高，从而提高了存储密度。如图 2 所示：



极客时间

图2 串的链式存储结构示意图【改进版】

从图 2 可以看到，第二个节点只有 e 和 f 两个字符，并没有填满，可以在字符 f 后面用字符串结束标记 '\0' 标记字符串的结束。

串的链式存储不需要大块连续的内存空间。插入、删除等操作通过修改指针即可实现，不需要大量的移动字符数据。但串的链式存储不具备串的顺序存储中的随机存取特性，所以其基本操作实现代码可能会更加复杂，因而也不如串的顺序存储常用。

shikey.com 转载分享

这里我就不提供串的链式存储相应代码了，如果你有兴趣可以自行实现。

小结

这节课我带你认识了串，给出了串、串的长度、空串的定义，也给出了串的常用处理函数。在编写代码时，注意串需要用双引号引起来。

接下来，为了引出与串相关的算法，又给出了子串、主串、字符在主串中位置概念。我希望你把串理解成一种数据元素之间呈线性关系的特殊线性表。接着我还向你介绍了串的基本操作——串赋值、串拷贝、判断空串、求长度、串连接、串比较、获取串的一部分内容、串插入、串删除、串清空、定位。

串的存储结构主要分为顺序存储结构和链式存储结构。顺序存储结构是用一块地址连续的内存保存串中的字符序列，而串的链式存储结构会使用链表来保存串中的各个字符序列。

在串的顺序存储结构介绍中，我们首先展示了用定长数组作为串的存储结构及在该结构之上实现的各种针对串的各种基本操作。然后为了进一步增加对串操作的灵活性，又展示了在堆中分配内存来保存串以及针对串的各种基本操作。

串的链式存储结构在本节中只做了简单介绍，这种存储结构有两个好处。

不需要大块连续的内存空间。

插入、删除等操作通过修改指针即可实现，不需要大量的移动字符数据。

但是，我们要知道，串的链式存储不具备串的顺序存储中的随机存取特性，所以它的实用性其实是不如串的顺序存储的。

思考题

请你实现串的链式存储代码，并使用串的链式存储实现：输入一个字符串，将其中的大写字母转换为小写字母并输出转换后的字符串。

欢迎你在留言区和我分享成果。如果觉得有所收获，也可以把课程分享给更多的朋友一起学习。我们下节课见！

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (1)



Se7en

2023-05-18 来自北京

细致，重温一遍技术基础



shikey.com转载分享