第5章 字 符 串

字符串是程序设计中最复杂的编程内容之一。STL string 类提供了强大的功能，使得许多烦琐的编程内容用简单的语句就可完成。string 字符串类减少了 C 语言编程中三种最常见且最具破坏性的错误：超越数组边界；通过未被初始化或被赋以错误值的指针来访问数组元素；以及在释放了某一数组原先所分配的存储单元后仍保留的“悬挂”指针。

5.1 字符串创建及初始化

5.1.1 基本创建方式

【例 5.1】 字符串基本创建示例。

```
//文件名：e5_1.cpp
#include<string>
using namespace std;
int main(int argc, char * argv[])
{
    string s1;
    string s2("How are you?");
    string s3(s2);
    string s4(s2, 0, 3);
    string s5="Fine";
    string s6=s2+"Fine";
    return 0;
}
```

对象 s1～s4 均是通过构造函数的方式创建新字符串对象。主要理解点如下。

(1) s1 string 对象被创建，但不包含初始值。C 语言中的 char 型数组在初始化前都包含随机的无意义的位模式，而与此不同，s1 确实包含了有意义的信息。这个 string 对象被初始化成包含“没有字符”，通过类的成员函数能够正确地报告其长度为 0 并且没有数据元素。

(2) 创建 s4 对象,构造函数中有三个参数。第一个参数类型是 string 类型,第二、三个参数是整型,分别表示偏移量及计数量。该构造函数的含义是源串 s2,从偏移量 0 的字符开始连续取三个字符,构成新的字符串对象,因此 s4="How"。

(3) 对象 s5、s6 通过赋值的方式产生新的对象。s5 是单一赋值,可直接把 char 型数组赋给 s5;s6 表明可将不同的初始化数据源结合在一起,本例即是把一个 string 对象 s2 与一个 char 型数组结合,构成一个新的对象 s6。但要注意:写成 string s6=s2+"Fine"可编译通过,而写成 string s6="Fine"+s2 不能编译通过。从中可以看出:string 对象赋值,等号右边第一项必须是 string 类型,不能是 char 型数组。

5.1.2 迭代器创建方式

由于可将 string 看做字符的容器对象,因此可以给 string 类的构造函数传递两个迭代器,将它们之间的数据复制到新的 string 对象中。

【例 5.2】 字符串迭代器创建方式。

```
//文件名:e5_2.cpp
#include<string>
using namespace std;
int main(int argc, char * argv[])
{
    string s1="How are you?";
    string s2(s1.begin(), s1.end());
    string s3(s1.begin()+4,s1.begin()+7);
    return 0;
}
```

由于 s1.begin(),s1.end()迭代器覆盖了 s1 的全部内容,因此 s2="How are you?";s1.begin()+4,s1.begin()+7 仅代表了 s1 部分内容"are",因此 s3="are"。

5.2 字符串操作

5.2.1 插入操作

字符串一般包括首字符前、尾字符后、任意位置插入等几种情况。

【例 5.3】 字符串插入操作。

```
//文件名:e5_3.cpp
#include<string>
#include<iostream>
```

```
using namespace std;
int main(int argc, char * argv[])
{
    string s="do";
    cout<<"Inition size is:"<<s.size()<<endl;
    s.insert(0, "How");
    s.append("you");
    s=s+"do?";

    cout<<"final size is:"<<s.size()<<endl;
    cout<<s;
    return 0;
}
```

执行结果如下：

```
Inition size is:2
final size is:14
How do you do?
```

通过该示例，可得出主要知识点如下。

(1) insert 函数，第一个参数表明插入源串的位置，第二个参数表明要插入的字符串，因此利用该函数可实现串首、串尾及任意位置处的字符串插入功能。

(2) append 函数，仅有一个输入参数，在源字符串尾部追加该字符串。

(3) 利用＋实现字符串的连接，从而创建新的字符串。

(4) size 函数，无输入参数，通过例子可知：它表明字符串长度值，即有多少个字符，初始的时候值是 2，当完成增加后，值就变成 14 了。这说明字符串 string 类本身可根据需要自动调节串所在的内存空间大小，编程者无须参与，这一点是 C 语言中 char 型数组无法比拟的。

5.2.2 替换操作

常用的是 replace 函数，有三个输入参数：第一个用于指示从字符串的什么位置开始改写；第二个用于指示从原字符串中删除多少个字符，第三个是替换字符串的值。

【例 5.4】 字符串替换操作。

```
//文件名：e5_4.cpp
#include<string>
#include<iostream>
using namespace std;
int main(int argc, char * argv[])
{
```

```
    string s="what's your name?";
    cout <<"替换前:"<<s<<endl;
    s.replace(7, 4, "her");
    cout <<"替换后:"<<s<<endl;
    return 0;
}
```

执行结果为:

```
替换前:what's your name?
替换后:what's her name?
```

5.3 字符串查询

主要掌握的知识点如下。

- string::npos:这是 string 类中的一个成员变量,一般应用在判断系统查询函数的返回值上,若等于该值,表明没有符合查询条件的结果值。
- find 函数:在一个字符串中查找指定的单个字符或字符组。如果找到,就返回首次匹配的开始位置;如果没有找到匹配的内容,则返回 string::npos。一般有两个输入参数,一个是待查询的字符串,一个是查询的起始位置,默认起始位置为 0。
- find_first_of 函数:在一个字符串中进行查找,返回值是第一个与指定字符串中任何字符匹配的字符位置;如果没有找到匹配的内容,则返回 string::npos。一般有两个输入参数,一个是待查询的字符串,一个是查询的起始位置,默认起始位置为 0。
- find_last_of 函数:在一个字符串中进行查找,返回最后一个与指定字符串中任何字符匹配的字符位置;如果没有找到匹配的内容,则返回 string::npos。一般有两个输入参数,一个是待查询的字符串,一个是查询的起始位置,默认起始位置为 0。
- find_first_not_of 函数:在一个字符串中进行查找,返回第一个与指定字符串中任何字符都不匹配的元素位置;如果没有找到匹配的内容,则返回 string::npos。一般有两个输入参数,一个是待查询的字符串,一个是查询的起始位置,默认起始位置为 0。
- find_last_not_of 函数:在一个字符串中进行查找,返回下标值最大的与指定字符串中任何字符都不匹配的元素位置;如果没有找到匹配的内容,则返回 string::npos。一般有两个输入参数,一个是待查询的字符串,一个是查询的起始位置,默认起始位置为 0。
- rfind 函数:对一个串从尾至头查找指定的单个字符或字符组,如果找到,就返回首次匹配的开始位置;如果没有查找到匹配的内容,则返回 string::npos。一般有两个输入参数,一个是待查询的字符串,一个是查询的起始位置,默认起始位置为串尾部。

【例 5.5】 字符串查询函数基本用法。

```
//文件名：e5_5.cpp
#include<string>
#include<iostream>
using namespace std;

int main(int argc, char * argv[])
{
    string s="what't your name?my name is TOM. How do you do?Fine, thanks. ";
    int n=s.find("your");
    cout<<"the first your pos:"<<n<<endl;
    n=s.find("you", 15);
    cout<<"the first your pos begin from 15:"<<n<<endl;
    n=s.find_first_of("abcde");
    cout<<"find pos when character within abcde:"<<n<<endl;
    n=s.find_first_of("abcde", 3);
    cout<<"find pos begin from 2 when character within abcde:"<<n<<endl;
    return 0;
}
```

执行结果如下：

```
the first your pos: 7
the first your pos begin from 15:41
find pos when character within abcde:2
find pos begin from 2 when character within abcde:13
```

(1) find("your")：从源串起始位置为 0(默认值)处查找有"your"字符串位置，所以结果为 7。

(2) find("you", 15)：从源串起始位置为 15 处查找有"you"字符串位置，所以结果为 41。

(3) find_first_of("abcde")：从源串起始位置为 0(默认值)处依次查找每个字符，如果它在输入的字符串参数“abcde”中，则返回该字符的位置。由于源串头两个字符“w”，“h”不在查找的串中，而第三个字符“a”在查找的目的串中，又由于字符串是以 0 为基点的，所以结果值为 2。

(4) find_first_of("abcde",3)：从源串起始位置为 3 处依次查找每个字符，如果它在输入的字符串参数“abcde”中，则返回该字符的位置。分析同 find_first_of("abcde")，结果为 13，对应的源串字符为“a”。

本例是以 find、find_first_of 函数为例来写的，同理可写出近似的 find_last_of()、find_first_not_of()、find_last_not_of()、rfind()代码，同学们可自行完成。

5.4 在字符串中删除字符

在字符串中删除字符主要用 erase 函数,有两个迭代器输入参数,之间表示的字符将被删除掉。

【例 5.6】 字符串删除函数示例。

```
//文件名: e5_6.cpp
#include<string>
#include<iostream>
using namespace std;
int main(int argc, char * argv[])
{
    string s1="How are you?";
    s1.erase(s1.begin(), s1.begin()+3);
    cout<<"after erase to s1 is:"<<s1<<endl;

    string s2="Fine, thanks";
    s2.erase(s2.begin(), s2.end());
    cout<<"after erase to s2 is:"<<s2<<endl;
    return 0;
}
```

执行结果为:

```
after erase to s1 is: are you?
after erase to s2 is:
```

s1.erase(s1.begin(), s1.begin()+3)表明删除 s1 串的前三个字符,所以结果是 s1 变为"are you?"; s2.erase(s2.begin(), s2.end())表明删除整个 s2 字符串,故没有显示结果。

5.5 字符串比较

主要是依据 ASCII 值来比较大小。若字符串 s1"大于"s2,表明两者相比较时遇到了第一对不同的字符,字符串 s1 中第一个不同的字符比字符串 s2 中同样位置的字符在 ASCII 表中的位置更靠后。

C++ STL 提供了多种字符串比较方法,它们各具特色。其中最简单的就是使用非成员的重载运算符函数 operator==、operator!=、operator>、operator<、operator>=和

operator<=。

【例 5.7】 字符串比较函数示例。

```
//文件名：e5_7.cpp
#include <string>
#include <iostream>
using namespace std;
int main(int argc, char * argv[])
{
    string s1="this";
    string s2="that";
    if(s1>s2) cout<<"s1>s2"<<endl;
    if(s1==s2)cout<<"s1=s2"<<endl;
    if(s1<s2) cout<<"s1<s2"<<endl;
    return 0;
}
```

可以看出，直接运用重载运算符就可以了，非常方便。s1、s2 前两个字符是相同的，第三个字符比较时由于 i 的 ASCII 大于 a 的 ASCII，因此 s1>s2。

5.6 综合示例

下面列举一些常用功能的简单示例，对一些前文未讲到的知识点做了必要的说明。

【例 5.8】 整型数据与字符串互相转化。

```
//文件名：e5_8.cpp
#include<string>
#include<iostream>
#include<sstream>
using namespace std;

int main(int argc, char * argv[])
{
    //将整型 10 转化成字符串"10"
    int n1=10;
    string s1;
    stringstream os1;
    os1<<n1;
    os1>>s1;
    cout<<"整型 10 转化成字符串 10："<<s1<<endl;

    //将字符串"123"转化成整型 123
```

```
    int n2=0;
    string s2="123";
    stringstream os2;
    os2<<s2;
    os2 >>n2;
    cout<<"字符串 123 转化成整型 123: "<<n2<<endl;
    return 0;
}
```

执行结果为：

```
整型 10 转化成字符串 10: 10
字符串 123 转化成整型 123: 123
```

可知在 STL 中，实现基本数据类型与字符串的相互转化，用流类如 stringstream 做中间媒介是一个很好的思路。stringstream 是输入输出流，利用输出流功能先填充输出缓冲区，再利用输入流功能依次读缓冲区，赋值给相应的变量。

【例 5.9】 利用 STL 拆分字符串。

例如，有字符串"How are you?"按空格拆分，应得到三个子串："How","are","you"。

方法 1：

```
//文件名: e5_9_1.cpp
#include<iostream>
#include<string>
using namespace std;
int main(int argc, char * argv[])
{
    string strText="How are you? ";                          //源串
    string strSeparator=" ";                                 //按空格拆分
    string strResult;                                        //拆分结果串
    int size_pos=0;                                          //拆分子串结束位置
    int size_prev_pos=0;                                     //拆分子串起始位置

    while((size_pos=strText.find_first_of(strSeparator ,size_pos))!=string::npos)
                                                             //找到子串
    {
        strResult=strText.substr(size_prev_pos , size_pos-size_prev_pos);
                                                             //取子串

        cout<<"string="<<strResult<<endl;
        size_prev_pos=++size_pos;    //下一子串起始位置、结束位置=当前子串结束位置+1
    }

    if(size_prev_pos!=strText.size())                        //判断有无最后一个子串
```

```
    {
        strResult=strText.substr(size_prev_pos , size_pos-size_prev_pos);
        cout<<"string="<<strResult<<endl;
    }
    return 0;
}
```

执行结果为：

```
string=how
string=are
string=you?
```

主要思路分析如下。

(1) 在搜索每一个符合条件的子串前，先使子串起始位置等于结束位置，然后按照find_firsr_of 函数查找相应子串，若有，则子串结束位置不等于起始位置，再利用 substr 函数取之间的字符，即得子串值。

(2) 程序中的 if 语句是判断最后一个子串的边界条件的，若起始位置等于该字符串的长度，表明已无字符串可取，否则还有一个子串。例如本例中的"How are you?"，当前两个串("How","are")取完后，size_prev_pos ＝size_pos＝8，但最后一个串"you?"已搜索不到空格字符，因此跳出 while 循环，if 语句条件成立，可取到子串"you?"。当然，若源串是"How are you?"，结尾多一空格，则所有的子串均可在 while 条件语句中搜索到，if 语句条件不成立，条件体也就不执行了。

方法 2：

```
//文件名：e5_9_2.cpp
#include<iostream>
#include<sstream>
#include<string>
using namespace std;
int main(int argc, char * argv[])
{
    string strResult="";
    string strText="How are you";
    istringstream istr(strText);                    //用字符串输入流封装字符串

    while(! istr.eof())                             //当非字符串输入流末尾
    {
        istr>>strResult;                            //读输入流并给变量赋值
        cout<<"string="<<strResult<<endl;
    }
    return 0;
}
```

巧妙地把静态字符串进一步封装成 istringstream 对象,变为动态,再通过提取符动态拆分字符串。这种“静态-动态”思维值得借鉴。但是可能有同学会问:这种方法只能按空格拆分,若按其他字符比如“,”拆分该怎么办呢?其实,只要用 getline 函数代替提取符>>就可以了,因为该函数最后一个参数可以设置拆分字符,代码如下:

```
#include<iostream>
#include<sstream>
#include<string>
using namespace std;
int main(int argc, char * argv[])
{
    string strResult="";
    string strText="How,are,you";
    istringstream istr(strText);                    //用字符串输入流封装字符串

    while(! istr.eof())                             //当非字符串输入流末尾
    {
        getline(istr, strResult, ',');              //读输入流并给变量赋值
        cout<<"string="<<strResult<<endl;
    }
    return 0;
}
```

【例 5.10】 C++ STL 实现的 string trim()功能。

即去掉字符串两端的空格,STL 中没有单独的 trim 函数可用,主要是应用 erase、find_first_not_of、find_last_not_of 函数。

```
//文件名:e5_10.cpp
#include<string>
#include<iostream>
using namespace std;

int main(int argc, char * argv[])
{
    string s=" hello ";
    s.erase(0, s.find_first_not_of(" "));           //删除左空格
    s.erase(s.find_last_not_of(" ")+1);             //删除右空格
    cout<<s;
    return 0;
}
```

【例 5.11】 对 string 类的进一步封装。

string 类中提供了丰富的函数,可视为抽象出的字符串功能的最小集。在实际的工程