6.5 串

Strings

Strings in C——not encapsulated

- Every C-string has type **char** *. Hence, a C-string references an address in memory, the first of a contiguous set of bytes that store the characters making up the string.
- The storage occupied by the string must terminate with the special character value '\0'.
- The standard header file <cstring> (or <string.h>) contains a library of functions that manipulate C-strings.
- some important functions:

```
char* strcpy(char* to, char* from);
int strcmp(char* one, char* two);
char* strcat(char* to, char* from);
int strlen(char* str);
```

Strings

Strings in C++——encapsulated

- In C++, the output operator << is overloaded to apply to cstrings, so that a simple instruction cout << s prints the string s.
- In C++, it is easy to use encapsulation to embed C-strings into safer class-based implementations of strings.
- The standard template library includes a safe string implementation in the header file <string>. This library implements a class called std :: string that is convenient, safe, and efficient.

```
#include <string>
using namespace std;
```

Strings implementation

- class specification:

```
class String{
  public:
      (1)String();  //构造函数
      (2)~String();//析构函数
      (3)String(const String &copy); //拷贝构造函数
      (4)String(const char* copy);  //将C字符串转成C++中的串
      (5)String(List<char> &copy);//将List转成C++串
      (6)void operator=(const String &copy);          //赋值符号重载
      (7)const char*  c_str() const;//转成C中的字符串

  protected:
      char* entries;
      int length;
};
```

String s("some_string");

4

String s;
S="some_string";

1 4 6

String s="some_stri...

const char *new_s=s.c_str();

4 7

Strings

- Strings in C++
  - 全局重载操作符

```
bool operator==(const String& first, const String& second);
bool operator>(const String& first, const String& second);
bool operator<(const String& first, const String& second);
bool operator>=(const String& first, const String& second);
bool operator<=(const String& first, const String& second);
bool operator!=(const String& first, const String& second);
```

运算符	规则
所有的一元运算符	建议重载为成员函数
= () [] ->	只能重载为成员函数
+= -= /= *= &= |= ~= %= >>= <<=	建议重载为成员函数
所有其它运算符	建议重载为全局函数

Strings

- realization of some important methods

```
//利用C的字符串构造
String::String(const char* in_string){
        length=strlen(in_string);
        entries=new char[length+1];
        strcpy(entries,in_string);
}
//利用List进行构造
String::String(List<char>& in_list){
        length=in_list.size();
        entries=new char[length+1];
        for (int i=0;i<length;i++)
                in_list.retrieve(i,entries[i]);
        entries[length]='\0';
}
```

Strings

- realization of some important methods

```
//转成C中的字符串
const char* String::c_str() const
{
    return (const char*) entries;//提供到内部String数据的访问
}
//有什么问题？是否有更好的办法？
```

```
String s="abc";
const char *new_string=s.c_str();
s="def";//调用赋值重载，要将s的原空间回收！
cout<<new_string;
```

可选的实现：

为string数据的副本分配动态内存。

```
const char* String::c_str() const
{int len;
char * temp;
len=strlen(entries;)
temp=new char[len+1];
strcpy(temp,entries);
return temp;
}
```

```
String s="some very_long string";
cout<<s.c_str();
//效率低，特别是字符串很长时
//客户程序必须记住使用之后要删除它，否则会因为临时对象没有删除而产生了垃圾！
```

Strings

```
//等于等于符号的重载
bool operator= =(const String& first, const String& second)
{
   return (strcmp(first.c_str(),second.c_str())==0);
}
```

Further String Operations

- **void** strcat(String &add_to, **const** String &add_on)
- **void** strcpy(String &copy, **const** String &original);
- **void** strncpy(String &copy, **const** String &original, **int** n);
- **int** strstr(**const** String &text, **const** String &target);
- String read_in(istream &input)
- String read_in(istream &input, **int** terminator);
- **void** write(String &s)

Samples of Further String Operations

```
void strcat(String &add_to, const String &add_on)

/* Post: The function concatenates String add on onto the end of
String add to .*/

{

const char *cfirst = add_to.c_str( );

const char *csecond = add_on.c_str( );

char *copy = new char[strlen(cfirst) + strlen(csecond) +1];

strcpy(copy, cfirst);

strcat(copy, csecond);

add_to = copy;

delete []copy;

}
```

```
int strstr(const String &text, const String &target);
/*postcondition: If String target is a substring of String text, the
function returns the array index of the first occurrence of the
string stored in target in the string stored in text.
else: The function returns a code of -1.*/
{
int answer;
const char * content_s = text.c_str( );
char *p = strstr((char *) content_s, targer.c_str( ));
if (p == NULL)
        answer = -1;
else
        answer = p- content_s;
return answer;
}
```

```
String read_in(istream &input)

/* Post: Return a String read (as characters terminated by a newline or
an end-of-file character) from an istream parameter. */

{

List<char> temp;

int size = 0;

char c;

while ((c = input.peek( )) != EOF && (c = input.get( )) != '\n')

        temp.insert(size++, c);

String answer(temp);

return answer;

}
```

We shall also find it useful to apply the following String output function as an alternative to the operator << .

```
void write(String &s)
/* Post: The String parameter s is written to cout. */
{
cout << s.c_str( ) << endl;
}
```