C语言风格字符串

目标：通过实战例题，彻底搞清楚C语言风格字符串。

1. C语言风格字符串的初始化

C语言风格字符串本质上是**字符数组**，但是其最大特征是数组的最后一个元素一定是`\0`！如果字符数组的最后一个元素并非`\0`，则一定不是字符串！

C字符串的初始化方式如下：

```
cppchar ch1[] = "hello world";
char ch2[] = {"hello world"};
char *ch3 = "hello world";
char ch4[] = "Hell" "o world";
char ch5[20] = "hello world";
char ch6[100];
```

- `ch1`、`ch3`、`ch5`的右值字符串`"hello world"`的数据类型是字符串常量`const char[12]`，包含了终止符`'\0'`。
- 如果没有事先声明字符串的大小，在初始化时，系统会自动给字符串分配与右值字符串常量相同大小的内存空间，并将该字符串拷贝进去。
- 如果已经声明字符串的大小，则会分配所声明大小的内存空间，未使用的空间会被初始化为`'\0'`。

例如，如果运行如下代码：

```
#include <iostream>
#include <cstring>
using namespace std;

int main() {
    char *ch3 = "hello world";
    char ch5[20] = "hello world";

    cout << (int)'\0' << endl;    // 输出 ASCII 码 0
    cout << (int)ch5[15] << endl; // 输出 0，ch5[15] 是 '\0'，即字符串的结束符
    // cout << (int)(ch3 + 15) << endl;  // 编译错误，因为 ch3+15 未初始化，无法访问
    return 0;
}
```

输出：

```
0
0
```

注意： `ASCII`码 0 对应的是`'\0'`。

字符串数组的初始化：

```
#include <iostream>
#include <cstring>
using namespace std;

#define MAX_STR_LEN 100
#define NUM_STRINGS 10

int main() {
    // 1. 定义一个二维字符数组，包含 NUM_STRINGS 个字符串，每个字符串最多 MAX_STR_LEN 个字
符
    char strings1[NUM_STRINGS][MAX_STR_LEN] = {
        "Hello, World!",
        "C Programming",
        "2D String Array Example",
        "Another string",
        "Example 5"
    };

    // 2. 定义一个指向字符的指针数组，最多包含 NUM_STRINGS 个字符串指针
    char *strings2[NUM_STRINGS] = {
        "Hello, World!",
        "C Programming",
        "2D String Array Example",
        "Another string",
        "Example 5"
    };

    return 0;
}
```

两种字符串数组的关系：

1. **二维字符数组** `strings1` 在内存中是连续的。`cout << sizeof(strings1) << endl;` 输出 `1000`，这是因为 `strings1` 作为二维字符数组存储了 1000 个字符变量（每个字符串最多 100 个字符，共 10 个字符串）。

   而 `strings2` 只是存储了 10 个常量字符的地址，每个地址占 8 字节（在64位系统上），因此 `cout << sizeof(strings2) << endl;` 输出 `80`。

2. `cout << *(strings1 + 2) + 3 << endl;` 与 `cout << *(strings2 + 2) + 3 << endl;` 均能输出 `"String Array"`，表示输出第 2 个（从 0 开始计算）字符串的第 3 个字符开始，直到遇到字符串结束符 `'\0'`。

3. `cout << strings1[2][3] << endl;` 与 `cout << strings2[2][3] << endl;` 均能输出 `S`，表示输出第 2 个字符串的第 3 个字符（从 0 开始计算）。

4. `cout << *(&strings1[0][0] + 100) << endl;` 可以输出 `C`，但 `cout << *(&strings2[0][0] + 100) << endl;` 会导致未定义行为（通常会崩溃）。这是因为 `strings1[0]` 的首地址与 `strings1[1]` 的首地址之间相差 100 字节，而 `strings2[0]` 的首地址与 `strings2[1]` 的首地址并没有这样的规律性关系。

小结：

- **二维字符数组**（例如 `strings1`）在内存中是连续存储的，每个字符串的字符紧密相连。
- **字符指针数组**（例如 `strings2`）存储的是指向各个字符串常量的指针，这些指针可能散布在内存的不同位置。

动态字符数组

字符指针不一定是字符数组，更不一定是字符串，有可能是指向一个字符的指针；只有将字符串的数组名赋予字符指针后字符指针才体现字符串的性质；

- **字符指针 (`char*`)** 和 **字符数组 (`char[]`)** 是不同的。字符指针可以指向一个字符、一个字符数组，或指向一个字符串常量，但它本身并不等于字符数组。
- **字符数组** 是一块连续的内存区域，通常用于存储字符串数据。而字符指针仅仅是一个指向字符的指针，它可以指向一个字符、字符数组，或者是一个字符串常量。
- **字符串** 是由一系列字符组成的，以 `\0` 结尾的字符数组。在将一个字符串字面量（如 `"hello"`）赋值给字符指针时，字符指针才表现为指向一个字符串。

字符也可以声明**动态字符数组**，比如：

```
char *str = new char[100];
char *str1 = new char [100] {"hello world"};
```

`new char[100]` **不会初始化**分配的内存空间，所得到的内存区域中的内容是**未定义的**（即垃圾值，注意，不是 `\0`）。如果你希望内存被初始化为 `0`（即 `\0`），可以使用以下方式：

```
char *str = new char[100]();  // 使用 () 会将内存初始化为 0
```

`new char[100] {"hello world"}` 的行为是将 `"hello world"` 字符串的字符拷贝到分配的内存中。因此，在 `str1` 中的前 11 个字符是 `'h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'`，并且 `str1[11]` **会是** `'\0'`，即字符串结束符。剩余的内存（从 `str1[12]` 到 `str1[99]`）**不会被初始化**，仍然是未定义的。但是从 `str1[12]` 到 `str1[99]` 的值是 `\0`（就比如声明 `char a`，强行调用 `a` 的）。

此外，我们也可以开辟**二维动态字符数组**：

```
int n = 100;

char **strings = new char *[n];
for (int i = 0; i < n; i++) {
    strings[i] = new char [n];
}

for (int i = 0; i < n; i++) {
    delete []strings[i];
}
delete[]strings;
```

- 动态二维字符数组的内存结构实际上是由一个**字符指针数组**组成，其中每个指针指向一个单独分配的字符数组（即每行字符数组的内存是离散的）。因此，不同的字符数组（每行）在内存中并不连续。

  动态二维字符数组是一个指向指针的指针，每个指针指向一个字符数组。所以，二维数组的每一行的内存地址并不会是连续的，而是分散的。
- 如果你希望二维数组的内存是连续的，可以使用一个单独的大块内存来模拟二维数组，这样内存是连续的。例如：

```
int n = 100, m = 100;
char *strings = new char[n * m];  // 分配一个连续的内存块
```

然后使用索引计算每个字符位置：

```
for (int i = 0; i < n; i++) {
    // 访问第 i 行的第 j 列字符
    char ch = strings[i * m + j];  // 注意这里的索引计算
}
```

这种方法确保了内存的连续性。

2. 字符串的输入/输出

输入：

1. 逐个字符地输入/输出；
2. 使用`cin`和`cout`的`>>`、`<<`输入和输出。

   缺点：
   - `cin`以空格、换行符和制表位作为终止符；
   - 不检查**内存溢出**；
3. 使用`cin`的成员函数`get`和`getline`输入：

   ```
   cin.getline(字符数组, 数组长度, 结束标记);
   cin.get(字符数组, 数组长度, 结束标记);
   ```

两者都从键盘接收每一个字符，直到遇到终止符或者达到字符数组长度 - 1。默认回车`\n`为终止符；二者区别在于，遇到终止符时`get`将终止符保存在缓冲区，而`getline`会将终止符在缓冲区中删除；

例如：

```
//程序一：cin.getline
#include <iostream>
#include <cstring>
using namespace std;


int main() {
    char ch1[100], ch2[100];
    cin.getline(ch1, 100, '.');
    cin.getline(ch2, 100);

    cout << ch1 << endl << ch2 << endl;

    return 0;
}
```

```
//程序二：cin.get
#include <iostream>
#include <cstring>
using namespace std;


int main() {
    char ch2[100], ch2[100];
    cin.get(ch1, 100, '.');
    cin.get(ch2, 100);

    cout << ch1 << endl << ch2 << endl;

    return 0;
}
```

输入：

```
aaa bbb ccc.ddd eee fff
```

输出（程序一）：

```
aaa bbb ccc
ddd eee fff
```

输出（程序二）：

```
aaa bbb ccc
.ddd eee fff
```

3. C字符串处理函数

C字符串处理函数包含`cstring`库。注意，不是`string`库！二者截然不同！

1. `strcpy`

```
strcpy(dest, src);
```

将字符串`src`拷贝到`dest`。函数的返回值是`dest`的地址。其实现的**核心思想**是**逐个字符**地**复制**源字符串（**包括终止的`\0`字符**）到目标字符串中，直到遇到终止符`\0`。

简化版`strcpy`的实现：

```
char* strcpy(char* dest, const char* src) {
    char* d = dest;  // 保存目标指针的位置，用于返回
    while (*src) {    // 当源字符串的字符不为 '\0'
        *d = *src;    // 将当前字符复制到目标字符串
        ++d;          // 移动目标指针
        ++src;        // 移动源指针
    }
    *d = '\0';        // 最后添加字符串结束符
    return dest;     // 返回目标字符串的起始位置
}
```

注意：

- `dest`和`src`都是指针，这就意味着可以进行指针的运算操作，`strcpy`参数传递的过程中也可以不传递字符串名。
- `strcpy`逐个字符地复制源字符串（包括终止的`\0`字符）到目标字符串中，直到遇到终止符`\0`，如果一个字符数组中存在一位`\0`那么，`\0`之后的字符将不会被复制；
- `strcpy`不会检查目标数组是否足够大来存储源字符串。如果目标数组不够大，可能会发生缓冲区溢出，这会导致未定义行为。因此，使用`strcpy`时需要确保目标数组足够大，或者使用更安全的版本，如`strncpy`或`std::string`。

  ```
  strncpy(dest, n, src);
  ```

2. `strcat`

```
strcat(dest, src);
```

将`src`拼接到`dest`后，返回`dest`的地址

`strcat`会将源字符串（包括`\0`终止符）附加到目标字符串的末尾，并在结束时自动添加`\0`字符，具体地`strcat`的实现过程如下：

1. **找到目标字符串的末尾**：遍历目标字符串，直到找到 `\0` 终止符，表示字符串的结束。
2. **将源字符串复制到目标字符串的末尾**：将源字符串中的每个字符复制到目标字符串中的空白位置，直到遇到源字符串的 `\0` 终止符。

简化版的 `strcat` 实现：

```
char* strcat(char* dest, const char* src) {
    // 1. 移动到目标字符串的末尾
    char* d = dest;
    while (*d) {  // 当d指向的字符不为'\0'时，继续向后移动
        ++d;
    }

    // 2. 将源字符串复制到目标字符串的末尾
    while (*src) {  // 当源字符串的字符不为'\0'时，继续复制
        *d = *src;  // 将源字符串中的字符复制到目标字符串
        ++d;        // 移动目标指针
        ++src;      // 移动源指针
    }

    // 3. 添加字符串结束符 '\0'
    *d = '\0';

    // 4. 返回目标字符串的起始位置
    return dest;
}
```

注意：

- `dest` 与 `src` 也都是指针，这就意味着可以进行指针的运算操作，`strcat` 参数传递的过程中也可以不传递字符串名。
- `strcat` 先查找 `dest` 中最近的 `\0`，再将源字符串复制到 `dest` 的末尾，直到终止的 `\0` 字符，并且会在 `dest` 末尾添加 `\0`；
- `strcat` 不会检查目标数组是否足够大来容纳源字符串。如果目标数组没有足够的空间，可能会发生缓冲区溢出，导致未定义行为。因此，使用 `strcat` 时，需要确保目标数组有足够的空间来存储源字符串的追加内容。

  比如，如果 `dest` 的大小不够大，就会引发内存问题。为了避免这个问题，可以使用 `strncat`，它允许指定最多追加的字符数，从而避免溢出：

  ```
  strncat(dest, src, sizeof(dest) - strlen(dest) - 1);
  ```

- 如果目标字符串已经很长，每次调用 `strcat` 都需要遍历目标字符串的整个内容。因此，在多次连接字符串时，可能会导致性能问题。

3. `strlen`

```
strlen(s)
```

返回字符串 `s` 的长度，及 `s` 中字符的个数，从 `s` 的地址到 `\0` 为止，不包含 `\0`。返回值的类型是 `size_t`；

简化版的 `strlen` 实现：

```
size_t strlen(const char* str) {
    const char* s = str;  // 保存字符串起始位置
    while (*s) {          // 当指针指向的字符不是 '\0'
        ++s;              // 移动指针，继续检查下一个字符
    }
    return s - str;       // 返回字符串的长度，即指针间距
}
```

注意：

- **返回值类型**： `strlen` 返回的是 `size_t` 类型，它是无符号整数类型，通常用于表示内存大小和字符串长度。
- **时间复杂度**： `strlen` 的时间复杂度是 `O(n)`，其中 `n` 是字符串的长度。因为它需要遍历整个字符串，直到找到 `\0`。
- **安全性**： `strlen` 仅仅是计算字符串长度，它假设传入的字符串是以 `\0` 结尾的。因此，使用 `strlen` 时必须确保传入的字符串是正确的 C 风格字符串，否则可能导致未定义行为。如果没有正确的 `\0` 终止符， `strlen` 可能会越界读取内存。

4. `strcmp`

```
strcmp(s1, s2)
```

比较 `s1` 与 `s2` 的字典序，如果$s1 > s2$返回正数，如果$s1 = s2$返回0，如果$s1 < s2$返回负数；

`strcmp` 的实现主要是逐个字符地比较两个字符串中的字符，直到遇到不相同的字符或遇到字符串的结束符 `\0` 为止。

1. **字符逐一比较**：从字符串的开始位置开始，逐个字符比较两个字符串对应位置的字符。
2. **返回比较结果**：如果遇到不同的字符，返回它们之间的差值。如果两个字符串完全相同，返回 0。
3. **处理不同长度的字符串**：如果一个字符串比另一个字符串长，但前面的字符都相同，那么较长的字符串会被认为是“大”字符串（因为它有更多的字符）。

简单的 `strcmp` 实现：

```
int strcmp(const char* str1, const char* str2) {
    while (*str1 && (*str1 == *str2)) {  // 逐字符比较，直到遇到不同的字符或字符串结束
        ++str1;
        ++str2;
    }
    return (unsigned char)*str1 - (unsigned char)*str2;  // 返回差值，处理字符差异
}
```

注意：

- 整形变量强制转化为布尔型，如果非零，则都转化为`true`，只有0才会被转化为`false`；
- `strcmp` 是逐个字符进行比较的，比较的是字符的 ASCII 值。大写字母的 ASCII 值小于小写字母，因此 `"A"` 会被认为小于 `"a"`。

- 如果其中一个字符串是空字符串（`""`），它会被认为是最小的字符串。在这种情况`strcmp("")`和任何非空字符串比较时，都会返回负值。
- `strcmp`的返回值是两字符串第一位存在差异的字符的`ASCII`码的差值，或者是0。
- `strncmp`至多比较`n`位；

5.`strchr`

```
strchr(s, ch)
```

其中，`s`是`char *`，`ch`是`const char`；

返回一个指向`s`中（从`s`的地址，到第一次出现`\0`的地址）第一次出现`ch`的地址；

`strnchr`最多检查`n`位；

5.`strstr`

```
strstr(s1, s2)
```

其中，`s1`是`char *`，`s2`是`const char*`；

返回一个指向`s1`中（从`s1`的地址，到第一次出现`\0`的地址）第一次出现`s2`的地址；

3. 关于C语言风格字符串的易错点

1. **库包含**：`#include <cstring>`而不是`#include <string>`
2. 字符指针只有指向字符数组或者new之后才具有字符数组的性质，否则字符指针只存储1位字符；

```
char strSrc[] = "hello world";
char *temp;
strcpy(temp, strSrc);//错误！temp是字符指针，只预留了存储一位字符的内存空间，更何况
temp没有内存空间用于存储\0；

temp = new char[100]();
strcpy(temp, strSrc);//正确，在堆区为temp开辟了存储100个字符的空间

temp = '\0';
```

3. 不能使用字符串给字符串初始化，也不能用字符串给字符串赋值，以下操作都是错误的：

```
char strSrc[100] = "hello world";
//不能使用字符串给字符串初始化
char str1[100] = strSrc;

//不能使用字符串给字符串赋值
char str2[100];
str2 = strSrc;
```

可以使用字符指针指向字符串，但是这属于**浅拷贝**，相当于两个指针指向了同一块内存；

若想要深拷贝，只能使用`strcpy`：

```
char strSrc[100] = "hello world";
//可以使用strcpy直接将源字符串深拷贝给目标字符串
char str1[100];
strcpy(str1, strSrc);

//可以让字符指针指向字符数组/字符串
char *str2 = strSrc;
```

4. 输入句子时使用`cin.getline(字符串名字, 最大长度, 终止符);`，而不是使用`cin`；
5. `strcmp`会返回正整数、负整数和0，但是正整数和负整数强制转化为`bool`会变为`true`，所以`if (strcmp(s1, s2))`只要`s1`和`s2`不相等就会进行；
6. 开辟字符数组的语法：`char *p = new char [100];`注意是方括号，不是圆括号！
7. 字符串的终止符是`\0`不是`/0`！
8. `char *p = new char [100];`之后，`p`所申请的空间存储的都是垃圾值，不是`\0`。所以，动态字符串操作结束后务必注意要在最后加上`\0`，或者一开始就`char *p = new char [100]();`
9. `toupper(c)` 是将字符转换为大写字母，并不会改变 `c` 的值，但 `toupper` 返回的是转换后的字符。应该将 `c = toupper(c)` 来确保 `c` 被更新为大写字母。
10. 为字符串申请内存空间一定至少要多申请一位字符，用于存储`\0`；
11. 如果通过例如`char *p = new char [100];`的方式开辟了动态字符串，如果要把另一个字符串的内容拷贝给`p`应该使用`strcpy`，因为`p`中的值都是垃圾值，而不包含`\0`，`strcat`需要从`\0`开始拼接；

接下来鉴赏上财历年关于字符串的程序设计题：

1. 数组与字符串

本题目的程序可以对输入的 3 个句子（包含空格的字符串）进行升序排序。并输出排好序的句子。程序正常运行示例如下：

```
Please input three sentences:
The Chinese people are hardworking people!
The Chinese people are selfless and enterprising people!
The Chinese people are brave to innovate!

Sorted sentences:
The Chinese people are brave to innovate!
The Chinese people are hardworking people!
The Chinese people are selfless and enterprising!
```

下面是实现上述功能的程序，请仔细阅读程序，找出其中错误并修改。
注意：
1 不能改变`main`函数中的数据定义：`char sentences[MAXNUM][MAXLENGHT];`
2）不能改变`stringSort`函数中的选择排序算法。

```
#include <iostream>
#include <string>
using namespace std;
const int MAXNUM = 3;
const int MAXLENGHT = 80;
void stringSort (char str[MAXNUM][], int n) {
    char *temp;
    for (int i = 0; i < MAXNUM - 1; i++) {
        k = i;
        for (int j = i + 1; j < MAXNUM; j++)
            if (strcmp (str[k], str[j]) ) k = j;
        if (k != i) {
            temp = str[i];
            str[i] = str[k];
            str[k] = temp;
        }
    }
}
int main() {
    char sentences[MAXNUM][MAXLENGHT];
    cout << "Please input three sentences:" << endl;
    for (int i = 0; i < MAXNUM; i++)
        cin >> sentences[i] << endl;
    cout << endl;
    stringSort (sentences, MAXNUM);
    cout << "Sorted sentences: " << endl;
    for (int i = 0; i < MAXNUM; i++)
        cout << sentences[i] << endl;
    return 0;
}
```

修改之后：

```
#include <iostream>
#include <cstring>                                  //是cstring而不是string
```

```
using namespace std;
const int MAXNUM = 3;
const int MAXLENGHT = 80;

//可以是char str[][MAXLENGTH]也可以是char str[MAXNUM][MAXLENGTH]
void stringSort (char str[][MAXLENGHT], int n) {
    //temp作为字符指针只存储了一位字符;
    char temp[MAXLENGHT];
    for (int i = 0; i < MAXNUM - 1; i++) {
        int k = i; //变量k违背初始化
        for (int j = i + 1; j < MAXNUM; j++)
            if (strcmp (str[k], str[j])>0) k = j;    //只有str[k]的字典序大于str[j]才
交换
        if (k != i) {
            strcpy(temp, str[i]);//temp = str[i];
            strcpy(str[i], str[k]);//str[i] = str[k];
            strcpy(str[k], temp);//str[k] = temp;
        }
    }
}
int main() {
    char sentences[MAXNUM][MAXLENGHT];
    cout << "Please input three sentences:" << endl;
    for (int i = 0; i < MAXNUM; i++)
        cin.getline(sentences[i], MAXLENGHT);
    cout << endl;
    stringSort (sentences, MAXNUM);
    cout << "Sorted sentences: " << endl;
    for (int i = 0; i < MAXNUM; i++)
        cout << sentences[i] << endl;
    return 0;
}
```

2.指针与动态数组

下面程序功能包括：

1. 将3个字符串，每个字符串去除其中非数字和非英文字母的字符，并将小写字母转化为大写字母，然后进行连接合并；
2. 统计3个字符串中数字字符和英文字母的频次；
3. 输出连接合并后的字符串、以及3个字符串中数字字符和英文字母的频次。

例如：

```
const char *str[3] = { "I love VC++ 2010!","Do you like Python 3.7?","Java SE18
is good!"};
```

去除非英文字母和非数字的字符，小写字母转化为大写字母，进行连接合并；

`mergestring`：`ILOVEVC2010DOYOULIKEPYTHON37JAVASE18ISGOOD`

输出连接合并串，英文字母和数字字符的频次；

```
Merged strings: ILOVEVC2010DOYOULIKEPYTHON37JAVASE18ISGOOD
数字字符的频次：0:2 1:2 2:1 3:1 7:1 8:1
英文字母的频次：A:2 C:1 D:2 E:3 G:1 H:1 I:3 J:1 K:1 L:2 N:1 O:6 P:1 S:2 T:1 U:1 V:3
Y:2
```

仔细阅读实现上述功能的`MergeStatistics`函数的实现，找出其中的错误，并（在不改变程序结构基础上）进行修改。
提示：注意用于计算频次的数组下标、大小写转换 、内存空间的动态分配等。

```
#include <iostream>
#include <cstring>
#include <cctype>
using namespace std;
const int MAXLENGTH = 3;
//去除非英文字母和非数字的字符，小写字母转化为大写字母，返回连接合并的字符串指针
char *MergeStatistics (const char *str[], int MaxLen, int digital[], int
alphabet[]) {
    int mergedlen = 0;
    for (int i = 0; i < MaxLen; i++)
        mergedlen += strlen (str[i]);
    char *mergestring = new char [mergedlen + 1]();      //圆括号改为方括号，在尾部加上
()，使所有元素初始化为\0;
    int counter = 0;
    for (int i = 0; i < MaxLen; i++) {
        const char *ptr = str[i];
        char c;
        while ( (c = *ptr) != '\0') {                    //是\0不是/0
            if (c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z') {
                if (c >= 'a' && c <= 'z') c -= 32;       //只有小写字母才要转换为大写
                ++alphabet[c - 'A'];                     //c对应的是ASCII码，而不是数
字，否则会导致数组越界
                mergestring[counter++] = c;              //改为counter++，否则跳过了
nergestring[0]
            }
            if (c >= '0' && c <= '9') {
                ++digital[c - '0'];
                mergestring[counter++] = c;
            }
            ptr++;
        }
    }
    return mergestring;
}
// 以下主程序无须修改
int main() {
    const char *str[MAXLENGTH] = { "I love VC++ 2010!",
                                   "Do you like Python 3.7?",
                                   "Java SE18 is good!"
                                 };
    int digital[10] = {0}; // 存放数字字符频次的数组，
    int alphabet[26] = {0}; // 存放英文字母频次的数组
//去除非英文字母和非数字的字符，小写字母转化为大写字母，返回连接合并的字符串指针
    char *mergedStr = MergeStatistics (str, MAXLENGTH, digital, alphabet);
```

```
    cout << "Merged strings: " << mergedStr << endl; // 输出连 接合并的字符串
    for (int i = 0; i < 10; i++) // 输出数字字符的频次
        if (digital[i]) cout << i << ":" << digital[i] << " ";
    cout << endl;
    for (int i = 0; i < 26; i++) // 输出英文字母的频次
        if (alphabet[i]) cout << char (i + 'A') << ":" << alphabet[i] << " ";
    delete []mergedStr; // 释放 new申请的空间
    return 0;
}
```

上一板块中，关于C语言风格字符串的易错点主要参考了这两道往年题。

非常稚嫩的一篇文章，有许多潜在的问题。希望大家帮忙勘误，找出文章中的错误或者表述不严谨的地方。

也希望阅读这篇文章的同学收益！

祝大家程序设计基础考试顺利！

作者：arctan37