

作业9

● **Ex1.** 编写程序，输入一系列字符，对其中的“->”进行计数。

```
int expCount(char str[])
{
    int count = 0;

    for(int i=1; i < strlen(str); ++i)
    {
        if (str[i-1] == '-' && str[i] == '>')
            ++count;
    }
    return count;
}
```

```
char str[81];
gets(str); //cin.getline(str, 81);

cout << expCount(str) << endl;
```

递推

```
int count = 0;
char ch1 = ' ', ch2;
cout << "Please input a string(terminated with #): \n";
while( (ch2=getchar()) != '#' )
{
    if (ch2 == '>' && ch1 == '-')
        ++count;
    ch1 = ch2;
}
cout << count << endl;
```

-
- 🌈 **Ex2.** 编写一个函数 `int Squeeze(char s1[], const char s2[])`, 它从字符串 **s1** 中删除所有在 **s2** 里出现过的字符, 并返回删除的字符个数。

不是删除子串

Hello **nju** ! I'm looking for **cs** building. \0

njucs \0

```
int squeeze(char s1[ ], const char s2[ ])
{
    int count = 0, i = 0;
    while (s1[i] != '\0')    // 遍历 s1
    {
        int j;
        for(j=0; s2[j] != '\0' && s1[i] != s2[j]; ++j) ;
        if (s2[j] == '\0')    // s2 中没有 s1[i]
            ++i;
        else                  // s2 中有 s1[i]
        {
            int k = i+1;
            for (; s1[k] != '\0'; ++k)
                s1[k-1] = s1[k];    // 往前搬1格
            s1[k-1] = '\0';
            ++count;
        }
    }
    return count;
}
```

漏删 (eg. 相邻两个n的第二个n不会被删除, 后面再出现的n不会被删除)

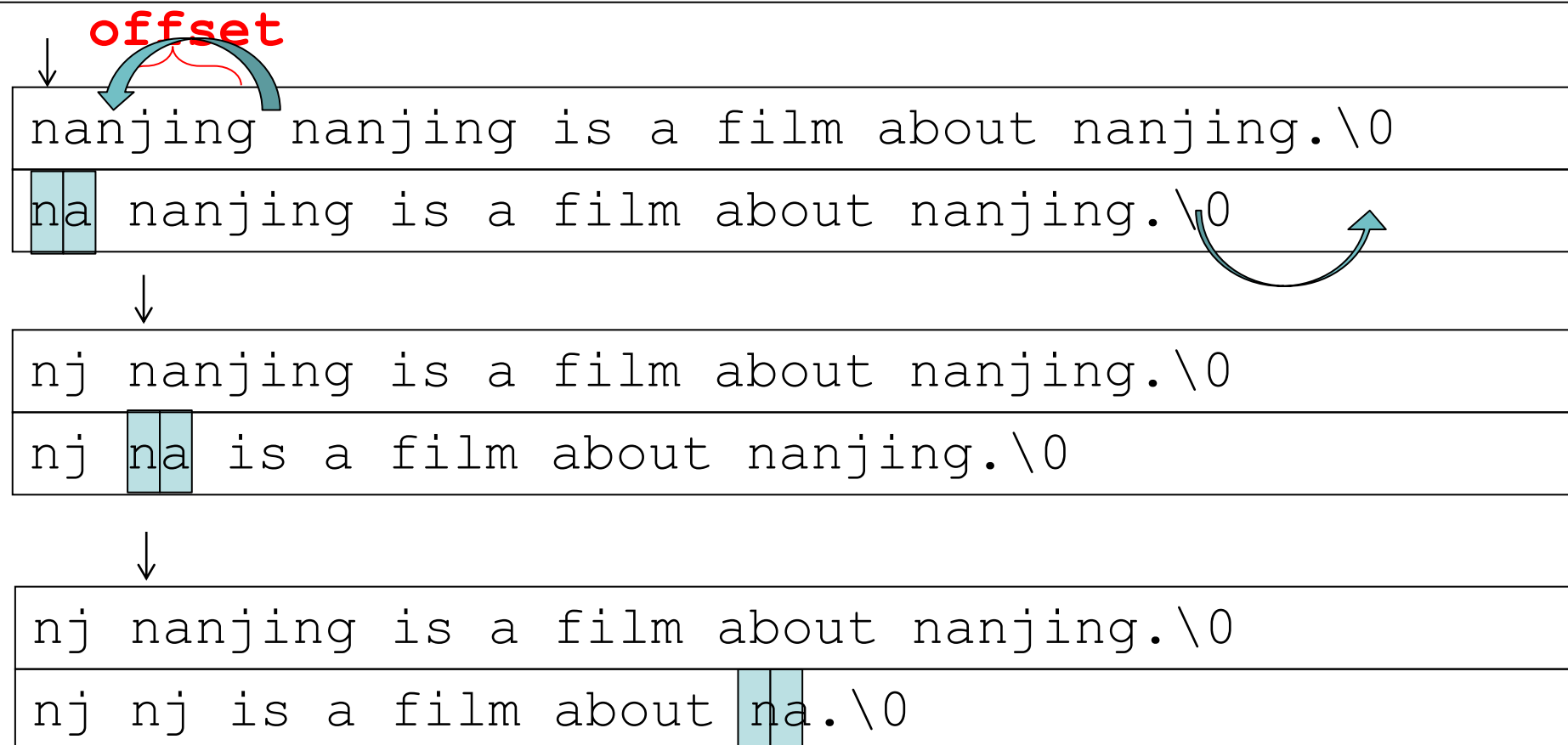
```
int squeeze(char s1[ ], const char s2[ ])
{
    int count = 0;
    for (int i = 0; s1[i] != '\0'; ++i)
        for (int j = 0; s2[j] != '\0'; ++j)
        {
            if (s1[i] == s2[j])
            {
                ++count;
                for (int k = i; s1[k] != '\0'; ++k)
                    s1[k] = s1[k + 1];
                --i; //或添加 j = -1;
            }
        }
    return count;
}
```

Hello nn jn!

njucs \0

-
- **Ex3.** 编写函数**FindReplaceStr**，其功能是将字符串**str**中的所有子串**find_str**都替换成字符串**replace_str**（其长度与**find_str**的长度不一定相等），返回值为替换的次数，其原型为：
 - **int FindReplaceStr(char str[], const char find_str[], const char replace_str[]);**

```
str :      nanjing nanjing is a film about nanjing.  
find_str : nanjing  
replace_str : nj
```



```
int find_replace_str(char str[ ], const char find_str[ ],
                    const char replace_str[ ])
{
    int count = 0;
    int index = 0;                                //str中的当前处理位置
    int find_len = strlen(find_str);
    int replace_len = strlen(replace_str);
    int offset = find_len - replace_len;
    while (strlen(str+index) >= find_len)
    { ..... }
    return count;
}
```

```
.....  
while (strlen(str+index) >= find_len)  
{  
    if (strncmp(str+index, find_str, find_len) == 0)  
    {  
        ...  
        ++count;  
    }  
    else  
        ++index;  
}  
return count;  
}
```

...

```
if (strncmp(str+index, find_str, find_len) == 0)
{
    int n = strlen(str+index) - find_len + 1;
    //剩余部分的字符个数+1 ('\0')

    if (offset < 0) //替换字符串 比较长
    {
        //需把后部分字符后移 -offset 个位置
        for (int i=strlen(str); n>0;  --i, --n)
            str[i+(-offset)] = str[i];
    }
    else if (offset > 0) //被替换字符串 比较长
    {
        //需把后部分字符前移offset个位置
        for (int i=index+find_len; n>0;  ++i, --n)
            str[i-offset] = str[i];
    }
    ...
}
```


```
...
if (strncmp(str+index, find_str, find_len) == 0)
{
    ...
    for (int i=0; i < replace_len; ++i) //复制替换串到str
        str[index+i] = replace_str[i];
    index += replace_len;

    ++count;
}
else
    ++index;
.....
```

-
- **Ex4.** 编程实现对一个只包含字母的字符串进行压缩和解压缩。压缩规则是：假设某连续出现的同一字母的数量为 n ，则其在压缩字符串中为" n 字母"，若 $n=1$ 则 n 必须省略，例如，"**AAAABCCCCCDDDD**"压缩为"**A4BC5D4**"。解压缩规则是将压缩字符串还原，例如，"**A4BC5D4**"解压为"**AAAABCCCCCDDDD**"。

```
#include <stdio.h>

...
int main()
{
    char src[30] = "AAAAAAAAAAAAAAAAABCCCCCDDDD";
    char dst[10];
    char newSrc[30];
    printf("Before compress: %s\n", src);
    Compress(dst, src);
    printf("After compress: %s\n", dst);
    Decompress(newSrc, dst);
    printf("After decompress: %s\n", newSrc);
    return 0;
}
```



```
void Compress(char dst[], char src[])  
{
```

```
    枚举, i < strlen(src) 或 '\0'
```

AAAABCCCCCDDDD

A4BC5D4

```
        计数, count=1
```

```
        枚举, src[i] == src[i+1]?  
            ++count, ++i
```

```
        count → ch(count+'0')
```

```
        dst[j] ← src[i], ++j
```

```
        dst[j] ← ch, ++j
```

```
        ++i
```

```
}
```

one more step

草稿

```
void Compress(char dst[], char src[])  
{
```

枚举, `i < strlen(src)` 或 `'\0'`

AAAAAAAAAAAAAAAAABCCCCCDDDD
A16BC5D4

计数, `count=1`

枚举, `src[i] == src[i+1]?`

`++count, ++i`

~~`count → ch(count+'0')`~~

`count → str`

`dst[j] ← src[i], ++j`

~~`dst[j] ← ch, ++j`~~

`dst[] ← str, j + strlen`

`++i`

`dst[j] = '\0';`

}



```
void Compress(char dst[], char src[])
```

```
{
```

```
    int i = 0;
```

```
    int j = 0;
```

```
    while(src[i] != '\0')
```

```
    {
```

```
        int count = 1;
```

```
        while(src[i] == src[i+1])
```

```
            ++count, ++i;
```

```
        dst[j++] = src[i++];
```

```
        if(count > 1)
```

```
            j += Int_to_str(dst + j, count);
```

```
//个数转为串
```

```
//个数是几位数
```

```
    }
```

```
    dst[j] = '\0';
```

```
}
```

```
AAAAAAAAAAAAAAAAABCCCCDDDD
```

```
A16BC5D4
```




```
int Int_to_str(char str[], int n)
```

```
{    int i = 0;
```

```
    while( n > 0)
```

```
    {        str[i++] = n % 10 + '0';
```

```
            n /= 10;
```

```
    }                // 逆序, 比如 16 个 A, 16 转换成 “61”
```

```
    str[i] = '\0';
```

```
    int length = i;
```

```
    --i;
```

```
    char t;
```

```
    for( int j = 0; j < i; ++j, --i )
```


```
    {        t = str[i];
```

```
            str[i] = str[j];
```

```
            str[j] = t;
```

```
    }                // 颠倒一下
```

```
    return length;    // 个数是几位数
```



```
void Decompress(char newSrc[], char dst[])
```

```
{    int n = strlen(dst);
```

```
    int i = 0, j = 0;
```

```
    char t;
```

```
    while (i < n)
```

```
    {    if (dst[i] > '0' && dst[i] <= '9')
```

```
        {    int count = 0;
```

```
            while (dst[i] >= '0' && dst[i] <= '9')
```

```
                count = dst[i++] - '0' + count*10;
```

```
            while(--count > 0)
```

```
                newSrc[j++] = t;
```

```
        }
```

```
    else
```

```
        t = dst[i++], newSrc[j++] = t;
```

```
}
```

```
newSrc[j] = '\0';
```

A16BC5D4

AAAAAAAAAAAAAAAAABCCCCDDDD

可处理多位 个数

else里已经处理了一个字母，
所以只要处理count - 1次

第13周自主训练任务

1. 自行设计小程序，验证：

(1) 程序中'A'与"A"的区别（提示：用sizeof操作符）；

(2) `char str[20] = "%%\t\n\x1a\092i\234s"`；中数组的长度和字符串常量真正的长度分别是多少？（提示：'\092'不是三位八进制ASCII码的转义符）

(3) 未初始化的字符数组元素默认值是什么？（提示：用%d格式符输出元素的ASCII码）

```
cout << sizeof(str) << endl;  
cout << strlen(str) << endl;
```

```
char str[20] = "%%\t\n\x1a\092i\234s";
```

2. 设计程序，在输入行（以**#**结尾）中查找关键字**q**，找到则输出其第一次出现的位置，找不到则输出**-1**。

```
char ch = getchar();
int newPos = 1;
int pos = -1;
while(ch != '#')
{
    if(ch=='q' && pos<0)
        pos = newPos;
    ch = getchar();
    ++newPos;
}
cout << pos << endl;
```

设标志位

```
char ch = getchar();
int newPos = 1;
int pos = -1;
char flag = 'F';
while(ch != '#')
{
    if(ch == 'q')
        if(flag == 'T') ;
        else flag = 'T', pos = newPos;
    ch = getchar();
    ++newPos;
}
printf(" %d\n", pos);
```

3. 编写程序，对输入的一个算术表达式，检查圆括号配对情况，输出：配对、不配对（多左括号、多右括号或左右括号颠倒均算作不配对）。

```
bool expMatch(char str[])
{
    int count = 0;
    for(int i=0; i < strlen(str); ++i)
    {
        if (str[i] == '(' )
            ++count;
        else if (str[i] == ')' )
            --count;
        if(count < 0)           // 中途count<0, 说明先出现 ')'
            return false;
    }
    if (count == 0 )
        return true;
    else
        return false;
}
```

反映“配对/不配对”两种情况

```
int expMatch(char str[])
{
    int count = 0;
    for(int i=0; i < strlen(str); ++i)
    {
        if (str[i] == '(' )
            ++count;
        else if (str[i] == ')' )
            --count;
    }
    return count; //>0:多左括号, <0:多右括号, 0:一样多
}
```

只能反映“多左/多右/一样多”三种情况；用全局变量或指针参数可反映“多左/多右/相等但不配对/配对”四种情况

4. 改写课件例**7.1**中的程序，要求用一个独立的函数计算各种字符的个数，并被**main**函数调用（提示：利用指针型参数将多个计算结果通过传址调用的方式传递给主调函数）。

```
void Stat2(char string[], int *nDigit, int *nSpace, int *nOther)
{ int i=0;
  while(string[i]!='\0')
  { if((string[i])>='0' && (string[i])<='9') (*nDigit)++;
    else if((string[i])==' ') (*nSpace)++;
    else (*nOther)++;
    i++;
  }
}

int main()
{ char str[80];
  int dgt=0, spc=0, othr=0;
  gets(str);
  Stat2(str, &dgt, &spc, &othr);
  cout << dgt << ', ' << spc << ', ' << othr << endl;
  return 0;
}
```

```
graph LR
    A["str, &dgt, &spc, &othr"] --> B["char string[]"]
    A --> C["int *nDigit"]
    A --> D["int *nSpace"]
    A --> E["int *nOther"]
```

5. 已知一段英文密文的加密方法为：对原文中的每个字母，分别用字母表中该字母之后的第5个字母替换（例如，“I WOULD RATHER BE FIRST IN A LITTLE IBERIAN VILLAGE THAN SECOND IN ROME”的密文为“N BTZQI WFYMJW GJ KNWXY NS F QNYYQJ NGJWNFS ANQQFLJ YMFS XJHTSI NS WTRJ”）。编写解密函数。

```
char* decode(char *s)
```

```
{
```

```
    for (int i = 0; s[i] != '\0'; ++i)
```

```
    {
```

```
        if (s[i] >= 'a' && s[i] <= 'z')
```

```
            s[i] = (s[i] - 'a' - 5 + 26) % 26 + 'a';
```

```
        else if (s[i] >= 'A' && s[i] <= 'Z')
```

```
            s[i] = (s[i] - 'A' - 5 + 26) % 26 + 'A';
```

```
    }
```

```
    return s;
```

```
}
```

```
{ s[i] = s[i] - 5;  
  if (s[i] < 'a')  
      s[i] = s[i] + 26;  
}
```

```
{ s[i] = s[i] - 5;  
  if (s[i] < 'A')  
      s[i] = s[i] + 26;  
}
```


6. 设计两个函数，分别实现将一个十进制正整数变换为十进制和二进制字符串。

```
char *IntToStr(int n)
{
    char *str = new char[len+1];

    str[len] = '\0';

    while(len > 0)
    {
        --len;
        str[len] = n%10 + '0';
        n = n/10;
    } //执行转换

    return str;
}
```



```
int len = 0;
int temp = n;
while(temp > 0)
{
    ++len;
    temp = temp/10;
} //求n的位数len
```

```
#define N 32
char * BiToDgt(int num, char str[])
{
    int i=0;
    for(; i < N; ++i)
        str[i] = '0';
    str[i] = '\\0';
    while(num > 0)
    {
        str[--i] = num%2 + '0';
        num = num/2;
    }
    return str;
}
```

```
int main()
{
    int n;
    cin >> n;

    char str[N+1];
    BiToDgt(n, str);

    cout << str << endl;

    return 0;
}
```

7. 将课件例7.4中的“`for(; (*pt) != '\0'; ++pt);`”改为：“`for(int i = 0; i < 10; ++i, ++pt);`”，观察程序的执行结果，并分析原因。（略）

8. 分析下面Hash函数的功能，并应用该函数实现某班同学的分组。

Hash函数的代码如下：

```
const int HASHSIZE = 10;
unsigned Hash(char *s)           //学号、姓名、电脑 IP 地址.....
{
    unsigned hashValue;
    for(hashValue = 0; *s != '\0'; s++)
        hashValue = *s + 31* hashValue;
    return hashValue%HASHSIZE;
}    // 修改 for 循环的算法，可以影响分组后的数据分布
```


9. 不使用库函数，编写StrLen、StrCat、StrNcat、StrCmp及StrNcmp函数。

```
int strLen(const char s[ ])
{
    int len = 0;

    for(int i=0; s[i] != '\0'; ++i)
        ++len;

    return len;
}
```

```
int strLen(const char *s)
{
    int len = 0;
    while (s[len++] != '\0');
    return --len;
}
```

```
char *strCat(char *dst, const char *src)
{
    char *p = dst;
    for(; *p != '\0'; ++p);    //循环结束p指向dst尾
    while(*src != '\0')
        *p++ = *src++;
    *p = '\0';
    return dst;
}
```

?

```
char *StrCat(char *dst, const char *src)
{

    for(; *dst != '\0'; ++dst);
    while(*src != '\0')
        *dst++ = *src++;
    *dst = '\0';
    return dst;
}
```

问题：得到空串
原因：没注意指针的移动！
对策：细心、多上机

```
int strcmp(const char *src1, const char *src2)
{
    while(*src1 == *src2)
    {
        if (*src1 == '\0')
            return 0;
        ++src1, ++src2;
    }
    return *src1 - *src2;
}
```

?

```
int strcmp(const char *str1, const char *str2)
{
    int m = 0, i = 0;
    while(str1[i] != '\0' && str2[i] != '\0')
    {
        if(str1[i] == str2[i])
        {
            ++i;
            continue;
        }
        else
        {
            m = str1[i] - str2[i];
            return m;
        }
    }
    return m;
}
```

原因：考虑不周，对循环流程理解不到位
对策：分情况考虑周全，复习

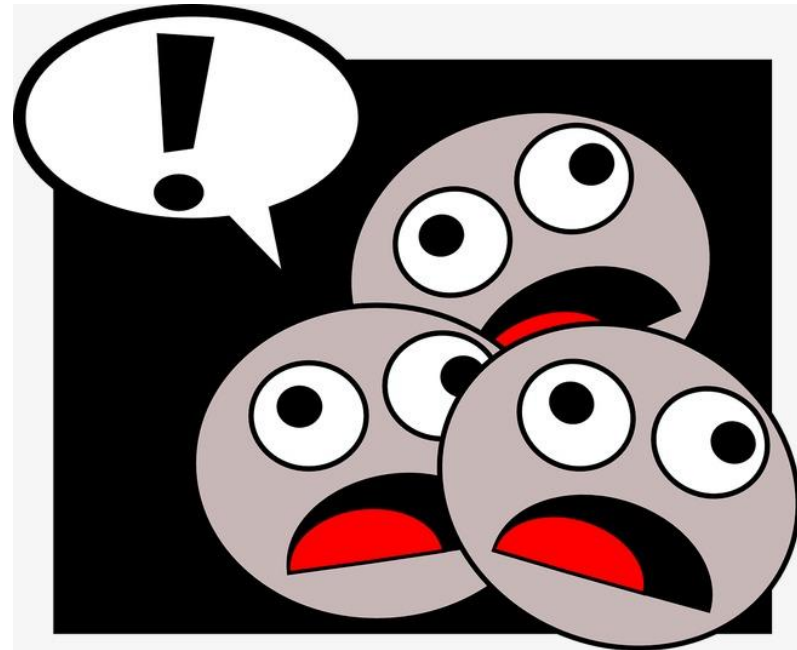
问题：Stu
Student
结果为0

?

```
int strcmp(const char *str1, const char *str2)
{
    if ((*str1 == '\0') && (*str2 == '\0'))
    {
        return 0;
    }
    while (*str1 == *str2)
    {
        ++str1;
        ++str2;
    }
    if (*str1 < *str2)
    {
        return -1;
    }
    else if (*str1 > *str2)
```

问题：相等会越界
原因：对 \0 理解不透
对策：多上机、思考

```
#define strCpy strcpy  
#define strCmp strcmp  
#define strCat strcat
```



为什么返回值类型是char *

- `char *strcpy(char * s1, const char * s2);`
- `char str[20];`
- `strcpy(str, "Hello ");`
- `cout << str << endl;`
- `cout << strcpy(str, "Hello ") << endl;`
- `cout << strcat(strcpy(str, "Hello "), "World!") << endl;`

链式操作

10. 编写函数，将字符数组（对应第一个参数）中的所有子串（对应第二个参数）去除。

先找

后删

```
#include <iostream>
#include <cstring>
using namespace std;
const int M = 20;
void DelSubstring(char a[], const char sub[]);
int main( )
{
    char a[M] = {0};
    cin >> a;
    const char sub[] = "bug";
    DelSubstring(a, sub);
    cout << a << endl;
    return 0;
}
```

`*(p + strlen(sub) - 1) != '\0'`

```
char *FindSubstring(char s[ ], const char sub[ ])
{
    char *p = s;
    while((strncmp(p, sub, strlen(sub)) != 0) && (*p != '\0'))
        ++p;
    if(*p != '\0')
        return p;
    else
        return NULL;
}
```

`*(p + strlen(sub) - 1) != '\0'`

?

```
void DelSubstring(char s[ ], const char sub[ ])
{

    char *position = FindSubstring(s, sub);
    if(position)
    {
        char *p = position + strlen(sub);
        while(*p != '\0')    //将后面的位置前移
            *position++ = *p++;
        *position = '\0';    //加上字符串结束符
    }
}
```

“dearbugbug” 中的第一个子串 “bug” 去除，结果为 “dearbug”

}

改正

```
void DelSubstring(char s[], const char sub[])
{
    while(*s)
    {
        char *position = FindSubstring(s, sub);
        if(position)
        {
            char *temp = position;
            char *p = position + strlen(sub);
            while(*p != '\0') //将后面的位置前移
                *position++ = *p++;
            *position = '\0'; //加上字符串结束符
            s = temp;
        }
    }
}
```

“dearbugbug” 中的子串 “bug” 去除，结果为 “dear”

```
void DelSubstring(char s[], const char sub[])
{
    while(*s)
    {
        char *position = FindSubstring(s, sub);
        if(position)
        {
            char *p = position + strlen(sub);
            while(*p != '\0')    //将后面的位置前移
                *position++ = *p++;
            *position = '\0';    //加上字符串结束符
        }
        else
            break;
    }
}
```

此代码可将删除子串之后又产生的新子串也删除

不过 不是原串的子串，并不符合本题要求

“dear**bugbugbug**” 中的子串 “**bug**” 去除，结果为 “dear”

?

```
void DelSubstring(char a[], const char sub[])
{
    int n = strlen(a), nSub = strlen(sub);
    int i;
    for(i=0; a[i+nSub-1] != '\0'; ++i)
        if(strncmp(a+i, sub, nSub) == 0)
        {
            int j;
            for(j=i; j < n-nSub+1; ++j) //含字符串结束符
                a[j] = a[j+nSub];

            } //将后面的字符往前移
    }

// "dearbugbugbug"
// "dearbugbug"
```

问题：删除子串之后紧接着的子串没有删除
原因：对循环的执行过程理解不透彻
（第一次循环之后 i 对应第二个子串的第二个字符，而不是第一个字符）
对策：多上机

改正

```
void DelSubstring(char a[], const char sub[])
{
    int n = strlen(a), nSub = strlen(sub);
    int i;
    for(i=0; a[i+nSub-1] != '\0'; ++i)
        if(strncmp(a+i, sub, nSub) == 0)
        {
            int j;
            for(j=i; j < n-nSub+1; ++j) //含字符串结束符
                a[j] = a[j+nSub];
            --i;
        } //将后面的字符往前移
}
```

```
// "dearbugbugbug"
// "dearbug"
```



```
void DelSubstring(char a[], const char sub[])
{
    int n = strlen(a), nSub = strlen(sub);
    int i;
    for(i=0; a[i+nSub-1] != '\0'; ++i)
        if(strncmp(a+i, sub, nSub) == 0)
        {
            int j;
            for(j=i; j < n-nSub+1; ++j) //含字符串结束符
                a[j] = a[j+nSub];

            i=0;
        } //将后面的字符往前移
}
```

此代码可将删除子串之后又产生的新子串也删除

不过 不是原串的子串，并不符合本题要求

```
// "dearbubugggbug"
// "dearbubuggbug"
```



?

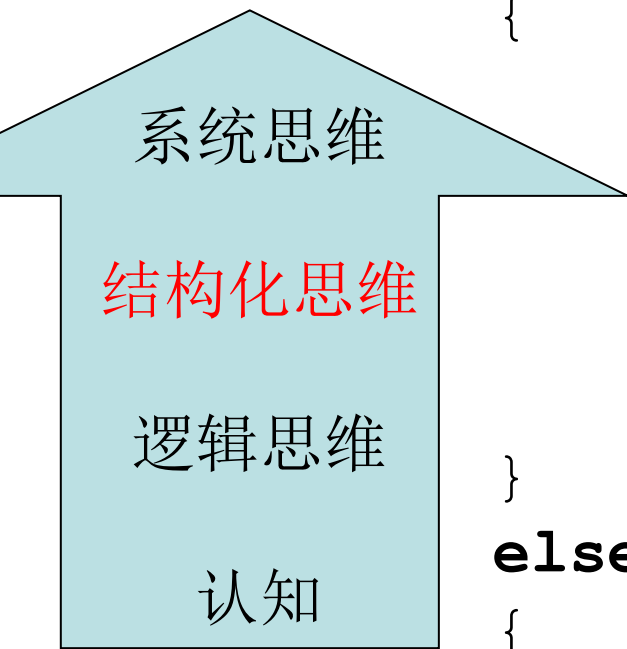
void DeleteSubstr(char * str1, const char * str2, char * str)

```
{    int a = 0;
    int b = 0;
    int c = 0;
    while(str1[a] != '\0')
    {    while(str2[b] != '\0')
        {    if(str2[b] == str1[a])
            {    ++a;
                ++b;
                if(str2[b] == '\0')
                {    b = 0;
                    break;
                }
            }
        }
        else
        {    b = 0;
            str[c]=str1[a];
            ++c;
```

即使调试正确，也是在强化 非结构化思维

对策：看例子规范代码，做每道题时 有意识地训练思维能力，不只是记忆

循环流程的执行过程没完全理解，内循环结束后，第二次外循环，a在哪里？ str1一定比str2长吗？



只搬迁一个字符。分支、顺序流程的执行过程也没完全理解

11. (选做) 编程，用动态数组存储用户输入的n个字符串，并按字典顺序重新排序。

```
void stringSort(char *p[],int n);
int main( )
{
    int n, m = 20;
    cin >> n;
    char **p = new char *[n];
    int i;
    for(i=0; i < n; ++i)
    {
        p[i] = new char[m];
        cin >> p[i];
    }
    stringSort(p,n);

    return 0;
}
```

```
for(i = 0; i < n; ++i)
    printf("%s \n", p[i]);

for(int i=0; i < n; ++i)
    delete []p[i];
delete []p;
```

```
void stringSort(char *name[], int n)
{
    for(int i = 0; i < n - 1; ++i)
    {
        int min = i;
        for(int j = i + 1; j < n; ++j)
            if(strcmp(name[min], name[j]) > 0) min = j;
        if(min != i)
        {
            char *temp = name[i];
            name[i] = name[min];
            name[min] = temp;
        }
    }
}
```

12.（选做） 用命令行方式（参见附录2）运行例7.8中的**echomyEcho**程序。
（课堂已演示）

Thanks!

