1.

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include<stdio.h>
#include<string.h>
int is_left_move(char* str1, char* str2)
{
 int len = strlen(str1);
 int len2 = strlen(str2);
 if (len != len2)
 return 0;
 //1.在str1中追加一个str1字符串
 //2.判断str2指向的字符串是否是str1指向的子串
 //strcat strncat
 //strcat(str1, str1); 自己给自己追加的时候不能使用strcat函数
 strncat(str1, str1, len);
 //strstr - 找字串函数
 char *ret = strstr(str1, str2); //在str1中找是否有str2 返回结果的首元素地址
 if (ret == NULL)
 {
 return 0;
 }
 else {
 return 1;
 }
}

int main()
{
 char arr1[30] = "abcdef";
 char arr2[] = "cdefab";
 int ret = is_left_move(arr1, arr2);
 if (ret == 1)
 {
 printf("yes\n");
 }
 else {
 printf("no\n");
 }
 return 0;
```

```
}
```

2.题目要求：

杨氏矩阵：有一个数字矩阵，矩阵的每行从左到右是递增的，矩阵从上到下是递增的，请编写出程序在这样的矩阵中查找某个数字是否存在：

要求：时间复杂度小于o(N);

```

int FindNum(int arr[3][3], int k, int *px, int *py)
{
 //最右上角开始
 int x = 0;
 int y = *py - 1;
 while (x <= *px - 1 && y >= 0)
 {
 if (arr[x][y] > k)
 {
 //排除本列
 y--;
 }
 else if (arr[x][y] < k) {
 x++; //排除本行
 }
 else {
 *px = x;
 *py = y;
 return 1;
 }
 }
 //此时没有找到
 return 0;
}

int main()
{
 //时间复杂度小于o(N)也就是不能遍历
 //通过比较最右上角元素和目标元素的大小 比目标元素大那么本列排除 小的话排除本行 最左下角也行
 int arr[3][3] = { {1,2,3},{4,5,6},{7,8,9} };

```

```
  int key = 7;
  int x = 3;
  int y = 3;
  //返回型参数
  int ret = FindNum(arr, key, &x, &y);
  if (ret == 1)
  {
  printf("找到了\n");
  printf("下标是: %d,%d\n", x, y);
  }
  else {
  printf("没找到\n");
  }
  return 0;
}
```

3.

3.1 求字符串长度

uint strlen() 求常量字符串长度 遇 到 \0 结束

3.2 长度不首限制的字符串长度

strcpy 常量字符串

```
int main()
{
 char arr1[] = "abcdefghi";
 char arr2[] = "bit";
 strcpy(arr1, arr2);
}

int my_strcpy(char *dest,char *src)
{
  assert(dest != NULL);
  assert(src != NULL);

  while(*src != '\0'){
  *dest++ = *src++;

  }
  *dest = *src;
```

```
18
19
20 }
21
22 int main()
23 {
24  char arr1[] = "abcdefghi";
25  char arr2[] = "bit";
26  my_strcpy(arr1, arr2);
27  printf("%s\n", arr1);
28  return 0;
29 }
```

strcat---

将arr2追加到arr1后面
源字符串要足够大能够放得下第二个字符串
源字符串后面要包含\0

```
1
```

int strcmp 比较两个字符串是否相等 比较的是对应字符ascll码的大小

VS编译器下：
0---- 1=2
1---- 1>2
-1---- 1<2
linux-gcc下：
0---1=2
<0 -- 1<2
>0 -- 1>2

```
1 int my_strcmp(const char* str1, const char* str2)
2 {
3  assert(str1 && str2);
4
5  while(*str1 == *str2)
6  {
7  if(*str1 == '\0')
8  {
9  return 0;//相等
10  }
11  str1++;
12  str2++;
13  }
14  if(*str1 > *str2)
15  return 1; //大于
16  else
17  return -1; //小于
```

```
}

int main()
{
 char* str1 = "abcdef";
 char* str2 = "asdfg";
 int ret = my_strcmp(str1, str2);
 if(ret == 0)
 {
 printf("相等");
 }else if(ret > 0){
 printf("str1 > str2");
 }else{
 printf("str1 < str2");
 }
 return 0;
}
```

3.3 长度受限制的字符串函数介绍

strncpy 将源串copy到目的地字符串

strncat 追加 都要有\0

strncmp 字符串比较 n-字节的个数

3.4 字符串查找

strstr -- 查找字符串 多次出现 则返回第一次出现的地址

NULL -- 空指针
NUL/Null -- '\0'

```
char* my_strstr(const char* p1, const char* p2)
{
 assert(p1 != NULL);
 assert(p2 != NULL);
 char* s1 = NULL;
 char* s2 = NULL;
 char* cur = p1;

 if(*p2 == '\0'){
 return p1;
 }
```

```
 while(*cur)
 {
 s1 = cur;
 s2 = (char*)p2;
 while((*s1!= '\0') && (*s2 != '\0') && (*s1== *s2))
 {
 s1++;
 s2++;
 }
 if(*p2 == '\0')
 {
 return cur;
 }
 cur++;
 }
 return NULL; //找不到字符串
}

int main()
{
 char* str1 = "asdfghj";
 char* str2 = "dfg"

 char* ret = my_strstr(str1, str2);

 return 0;
}
```

strtok --

在源字符串中找到目标字符串 并将其改为 \0 返回一个指向这个标记的指针

```
int main()
{
 char arr[] = "192.168.43.125";
 char* p = ".";

 char buf[1024] = {0};
 strcpy(buf, arr);

 //切割buff中的字符串
 char* ret = NULL;
 for(ret = strtok(arr, p); ret != NULL; ret = strtok(NULL,p))
 {
```

```
  printf("%s\n" ret);
  }
 }
```

3.5 错误信息报告

strerror

返回错误信

```
#include<errno.h>
//当库函数在执行过程中，发生了错误，就会把对应的错误码 赋值到errno中
strerror(errno);
```

3.6 字符操作

3.7 内存操作函数

memcpy 处理不重叠内存拷贝就可以

vs2013环境下可以处理重叠拷贝

```
#include<string.h>
#include<assert.h>
void* my_memcpy(void* dest, const void* src, size_t count)
{
 char* ret = dest;
 assert(dest != NULL);
 assert(src != NULL);


 while(count--)
 {
 *(char*)dest = *(char*)src;
 ++(char*)dest;
 ++(char*)src;
 }
 return ret;
}

int main()
{
 int arr1[] = {1,2,3,4,5};
 int arr2[] = {0};

 my_memcpy(arr2, arr1, sizeof(arr1));
 return 0;
}
```

memmove 处理重叠内存的拷贝

memset

memcmp