C6-D 题
题目：

Bellalabella 今天学习了回文串的内容，一个字符串是回文串当且仅当将其翻转后和原串相同。

Bellalabella 想知道对于一个字符串 S，它的最长回文子串长度是多少。

最快的算法是 Manacher 算法，时间复杂度为 O(n)。

由于不知道 s 的长度的奇偶性，寻找最长回文子串时会比较麻烦，所以首先来解决这个问题，对 s 进行一些改动。方法是在各个字符之间以及头尾添加一标记（不妨为'#'），再在字符串首添加一标记（不妨为'$'），再字符串末尾添加一标记（不妨为'^'）。假设输入的字符串 s 为：

abcbaade

则经过改动后得到的字符串 s_new 为：

$#a#b#c#b#a#a#d#e#^

注意作为标记的字符一定要是字符串 s 中不可能出现的字符，且开头和末尾的标记不能一样，如此可使得中心扩展判断字符是否相等时，扩展到字符串两端时字符一定不相等。改造后的字符串长度一定为奇数，不再需要担心奇偶性的问题。而字符串的末尾一定为'\0'，故可以直接以'\0'作为末尾的标记。

该部分的关键代码如下：

```
int len = strlen(s);
s_new[0] = '$';//开头标记
s_new[1] = '#';
int j = 2;
for (int i = 0; i < len; i++)
{
    s_new[j++] = s[i];
    s_new[j++] = '#';
}
s_new[j] = '\0';//末尾标记
```

接下来，开始寻找回文子串。需要一数组 p，p[i]表示从 i 位置的字符出发能向左右各扩展 P[i]个字符得到回文字符。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
s_new	$	#	a	#	b	#	c	#	b	#	a	#	a	#	d	#	e	#	\0
p	0	0	1	0	3	0	5	0	3	0	1	2	1	0	1	0	1	0	0

易知 p[i]即为以 i 位置为中心可得到的回文串长度。如在 s_new[6]左右各扩展 5 位可得到回文串#a#b#c#b#a#，去掉'#'可得 abcba，正好是 5 位。（同时还可以得到该回文子串开始的位置 int((i-p[i])/2)，如 int((6-5)/2)为 0，即以 s_new[6]为中心的回文子串在 s 的起始位

置为 s[0]，但是在题目中没有要求所以就算了）。

而求 p[i]的关键就在于充分利用回文字符串的对称性。

用 id 表示之前已经找到的某回文子串的中心位置，该回文串终点为 mx=id+p[id]。该回文子串是先前找到的所有回文子串里右端最靠右的，即 mx 尽量大。在求 p[i]的过程中，可以找到 i 关于 id 的对称点 im=2*id-i。因为该回文子串是对称的，i 和 im 又关于该回文子串的中心对称，故 p[i]=p[im]。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
s_new	$	#	a	#	b	#	c	#	b	#	a	#	a	#	d	#	e	#	\0
p	0	0	1	0	3	0	5	0	3										
					im		id		i			mx							

但是在三种情况下，这样直接赋值时不正确的：

1、超出 mx。如果 i+p[im]>mx，则不可直接利用对称性。但是以 i 为中心，至少可以向右扩展到 mx，向左也可以扩展 mx-i 位，保证是一个回文子串，即 p[i]>=mx-i。至于 p[i]究竟是否比 mx-i 大，则需要继续向左、向右扩展，直至无法扩展为止，记录下 p[i]。

2、im 遇到了 s_new 的边界。如果 im 为 2，则不可直接利用对称性。此时虽然 p[im]为 1，但是这是由于 s_new[im]的左边是'#'，再左边是表示左端边界的'$'，而不能保证 s_new[i]左右两端只能扩展到其左右相邻的两个'#'。故也需要继续向两边扩展，直至无法扩展为止，记录 p[i]。

3、i 与 mx 相等。此时无法再利用对称性，只能一步步向两边扩展。

为了确定 id 和 mx 以求 p[i]，需要及时更新 id 和 mx。当求出的 p[i]的右边界大于 mx 时，更新 id 为 i，mx 为 i+p[i]即可。

该部分的关键代码如下：

```
int id;
int mx = 0;
for (int i = 1; i < len; i++)
{
    int im=2*id-i;//求i关于id的对称点
    if (i < mx)
        p[i] = min(p[im], mx - i);
    else
        p[i] = 1;
    while (s_new[i - p[i]] == s_new[i + p[i]])//需要扩展的话，扩展；由于添加了标记，不用判断是否到达边界
        p[i]++;
    if (mx < i + p[i])//及时更新id和mx
    {
        id = i;
        mx = i + p[i];
    }

    max_len = max(max_len, p[i] - 1);
}
```

接着找出 p 数组的最大值，即可找到最大回文子串长度。