Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

For "(()", the longest valid parentheses substring is "()", which has length = 2.

Another example is ")()())", where the longest valid parentheses substring is "()()", which has length = 4.

思路：要找到最长的匹配括号子串，需要保存已经匹配成功的子串“状态”，状态指的是已匹配好的子串的位置，比如例子：”**()**(((**()**((**()**”，当匹配到**橘黄色**的括号时，前面的**红色**和**绿色**的括号的状态也需要保留，如果橘黄色的括号后边在多加几个’)’的话，比如变成” **()**(((**()**((**()** ))))) ”后，所有的括号都变成匹配的了。所以，该问题的关键就是：如何保存已经匹配好的子串的状态！！

一般在匹配括号的时候，都会使用到栈这种数据结构，当遇到”(”的时候，就入栈，遇到”)”的时候，就将栈顶弹出，表示一个匹配的括号。

因此，该题也可以利用栈这种结构，然后，开辟一个跟原字符串同样长度的空间ressets，ressets中只记录那些匹配好的子串，而无法匹配的括号则不记录。扫描源字符串，当遇到”(”的时候，将该”(”的位置入栈，当弹出栈的时候，就可以知道匹配的括号的具体位置，这样，就可以在ressets中记录匹配好的子串了。扫描完原始字符串之后，ressets中就记录了所有已经匹配好的子串，再次扫描ressets，就可以得到最长匹配子串的长度了。代码如下：

**typedef** struct

**{**

int **\***sets**;**

int len**;**

int top**;**

**}**Stack**;**

int push**(**Stack **\***s**,** int key**)**

**{**

**if(**s**->**top **>=** s**->**len**)** **return** **-**1**;**

s**->**sets**[**s**->**top **++]** **=** key**;**

**return** 0**;**

**}**

int pop**(**Stack **\***s**)**

**{**

**if(**s**->**top **==** 0**)** **return** **-**1**;**

**return** s**->**sets**[--** s**->**top**];**

**}**

Stack **\***initstack**(**int stlen**)**

**{**

Stack **\***st **=** calloc**(**1**,** **sizeof(**Stack**));**

st**->**sets **=** calloc**(**stlen**,** **sizeof(**int**));**

st**->**len **=** stlen**;**

st**->**top **=** 0**;**

**return** st**;**

**}**

int longestValidParentheses**(**char**\*** s**)**

**{**

int slen **=** strlen**(**s**);**

char **\***ressets **=** calloc**(**slen**+**1 **,sizeof(**char**));**

Stack **\***st **=** initstack**(**slen**);**

int reslen **=** 0**;**

int i**,** j**;**

**for(**i **=** 0**;** i **<** slen**;** i**++)**

**{**

**switch(**s**[**i**])**

**{**

**case** '('**:**

push**(**st**,** i**);**

**break;**

**case** ')'**:**

**if(**st**->**top **==** 0**)** **continue;**

j **=** pop**(**st**);**

ressets**[**j**]** **=** '('**;**

ressets**[**i**]** **=** ')'**;**

**break;**

**}**

**}**

i **=** 0**;**

**while(**i **<** slen**)**

**{**

**if(**ressets**[**i**]** **==** '\0'**)**

**{**

i**++;**

**continue;**

**}**

int tmplen **=** strlen**(**ressets**+**i**);**

**if(**reslen **<** tmplen**)**

**{**

reslen **=** tmplen**;**

**}**

i **+=** tmplen**;**

**}**

**return** reslen**;**

**}**

这种算法，时间和空间复杂度都是O(n)，但是，其实ressets并不是必须的，考虑下面例子：”)))**(()())**”，该字符串，前三个字符都是无法匹配的字符，因此，从第四个字符开始，才是可能的最长子串的**起点**。扫描该字符串：将” **(**”和” **(**”入栈，扫描到” **)**”时，弹出栈顶元素，此时匹配的子串长度为当前栈顶” **(**”和” **)**”之间的距离（减1），然后，接着将” **(**”入栈，扫描到” **)**”时，弹出栈顶元素，此时匹配的子串长度为当前栈顶” **(**”和” **)**”之间的距离（减1），扫描到” **)**”时，弹出栈顶，栈变空，子串的长度，是**起点**到” **)**”之间的距离，所以，代码如下：

int longestValidParentheses2**(**char**\*** s**)**

**{**

int slen **=** strlen**(**s**);**

Stack **\***st **=** initstack**(**slen**);**

int reslen **=** 0**;**

int tmplen **=** 0**;**

int startindex **=** **-**1**;**

int i**;**

**for(**i **=** 0**;** i **<** slen**;** i**++)**

**{**

**switch(**s**[**i**])**

**{**

**case** '('**:**

push**(**st**,** i**);**

**break;**

**case** ')'**:**

**if(**st**->**top **==** 0**)**

**{**

startindex **=** i**;**

**}**

**else**

**{**

pop**(**st**);**

**if(**st**->**top **==** 0**)**

**{**

tmplen **=** i**-**startindex**;**

**}**

**else**

**{**

tmplen **=** i **-** st**->**sets**[**st**->**top**-**1**];**

**}**

**if(**reslen **<** tmplen**)** reslen **=** tmplen**;**

**}**

**break;**

**}**

**}**

**return** reslen**;**

**}**

参考：

https://github.com/haoel/leetcode/blob/master/algorithms/longestValidParentheses/longestValidParentheses.cpp