```
/*
有效括号字符串为空 ("")、"(" + A + ")" 或 A + B，其中 A 和 B 都是有效的括号字符串，+ 代表字符串的连接。
例如，""，"()"，"(())()" 和 "(()(()))" 都是有效的括号字符串。
如果有效字符串 S 非空，且不存在将其拆分为 S = A+B 的方法，我们称其为原语（primitive），其中 A 和 B 都是非空有效括号字符串。
给出一个非空有效字符串 S，考虑将其进行原语化分解，使得：S = P_1 + P_2 + ... + P_k，其中 P_i 是有效括号字符串原语。
对 S 进行原语化分解，删除分解中每个原语字符串的最外层括号，返回 S 。
示例 1：
输入："(()())(())"
输出："()()()"
解释：
输入字符串为 "(()())(())"，原语化分解得到 "(()())" + "(())"，
删除每个部分中的最外层括号后得到 "()()" + "()" = "()()()"。
示例 2：
输入："(()())(())(()(()))"
输出："()()()()(())"
解释：
输入字符串为 "(()())(())(()(()))"，原语化分解得到 "(()())" + "(())" + "(()(()))"，
删除每隔部分中的最外层括号后得到 "()()" + "()" + "()(())" = "()()()()(())"。
示例 3：
输入："()()"
输出：""
解释：
输入字符串为 "()()"，原语化分解得到 "()" + "()"，
删除每个部分中的最外层括号后得到 "" + "" = ""。
提示：
    S.length <= 10000
    S[i] 为 "(" 或 ")"
    S 是一个有效括号字符串
来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/remove-outermost-parentheses
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 按照题设分析,我们可以对括号进行层次编号.然后拷贝编号指代非最外层括号的元素到新的空间.
- 在编号过程中,考虑到括号匹配问题，需要用到栈结构，优先使用C++语言。（方法一）
- 优化逻辑：编号过程和拷贝过程可以同时进行.(方法二)
- C语言模拟用数组模拟栈。(方法三)

---

方法一:C++_Solution

```
class Solution
{
    private:
        void mark(string& S, int* arr,int length)
        {
```

```
            int level   =   0;
            int i       =   0;
            char temp   =   0;

            stack<char> sc;
            for(i = 0;i<length;i++)
            {
                if(sc.empty()==true)
                {
                    level = 1;
                    sc.push(S.at(i));
                    arr[i] = level;
                }
                else
                {
                    if(S.at(i)=='(')
                    {
                        level++;
                        sc.push(S.at(i));
                        arr[i] = level;
                    }
                    else
                    {
                        sc.pop();
                        arr[i]=level;
                        level--;
                    }
                }
            }
        }
public:
    string removeOuterParentheses(string S)
    {
        string  ret_val                                         ;
        int     original_length =   S.length()                 ;
        int*    mark_level      =   new int[original_length]    ;
        int     i               =   0                           ;

        /*标注括号层级*/
        mark(S,mark_level,original_length);
        for(i = 0; i < original_length ; i++)
        {
            if(mark_level[i]>1)
            {
                ret_val += S[i];
            }
        }
        delete[] mark_level;
        return ret_val;
    }
};
```

```
/*
执行结果：
通过
显示详情
执行用时 :4 ms, 在所有 C++ 提交中击败了96.67% 的用户
内存消耗 :9.5 MB, 在所有 C++ 提交中击败了19.21%
*/
```

---

方法二:C++_Solution

```
class Solution
{
    public:
        string removeOuterParentheses(string S)
        {
            string ret_val ;
            int original_length = S.length();
            int i               = 0;
            int level           = 0;
            stack<char> sc;

            for(i = 0; i < original_length ; i++)
            {
                if(sc.empty()==true)
                {
                    level = 1;
                    sc.push(S.at(i));
                }
                else
                {
                    if(S.at(i)=='(')
                    {
                        level++;
                        sc.push(S.at(i));
                        if(level>1)
                        {
                            ret_val += S.at(i);
                        }
                    }
                    else
                    {
                        sc.pop();
                        if(level>1)
                        {
                            ret_val += S.at(i);
                        }
                        level--;
                    }
                }
            }
```

```
            return ret_val;

        }
};

/*
执行结果：
通过
显示详情
执行用时 :4 ms, 在所有 C++ 提交中击败了96.67% 的用户
内存消耗 :9 MB, 在所有 C++ 提交中击败了58.04%的用户
*/
```

---

方法3:C_Solution

```
char * removeOuterParentheses(char * S)
{
    int     str_len         = strlen(S);
    char*   ret_val         = (char*)malloc(sizeof(char)*str_len);
    int     valid_len       = 0;
    char*   stack_s         = (char*)malloc(sizeof(char)*str_len);
    int     stack_val_count = 0;
    int     i               = 0;
    int     level           = 0;

    //memset(ret_val,0,sizeof(char)*str_len);
    for(i = 0; i < str_len ; i++)
    {
        if(stack_val_count==0)
        {
            level = 1;
            stack_s[stack_val_count] = S[i]; /*sc.push(S.at(i));*/
            stack_val_count++;
        }
        else
        {
            if(S[i]=='(')
            {
                level ++;
                stack_s[stack_val_count++]= S[i]; /*sc.push(S.at(i));*/
                if(level > 1)
                {
                    ret_val[valid_len++] = S[i];
                }

            }
            else
            {
```

```
                //sc.pop();
                stack_val_count--;
                if(level-- > 1)
                {
                    ret_val[valid_len++] = S[i];
                }

            }
        }
    }

    ret_val[valid_len] = 0;
    free(stack_s);
    stack_s = NULL;
    return ret_val;
}

/*
执行结果：
通过
显示详情
执行用时 :0 ms, 在所有 C 提交中击败了100.00% 的用户
内存消耗 :7.3 MB, 在所有 C 提交中击败了14.29%的用户
*/
```

---

AlimyBreak
2019.07.18