1. 题目分析

说明程序设计的任务，强调的是程序要做什么，此外列出各成员分工

程序设计任务：设计一个解释魔王语言的程序将含有大写字母、小写字母、圆括号的字符串全部转化为人类语言（全部都为小写字母）。

成员分工： 陈欣欣 -> 栈类的设计、invert 函数；

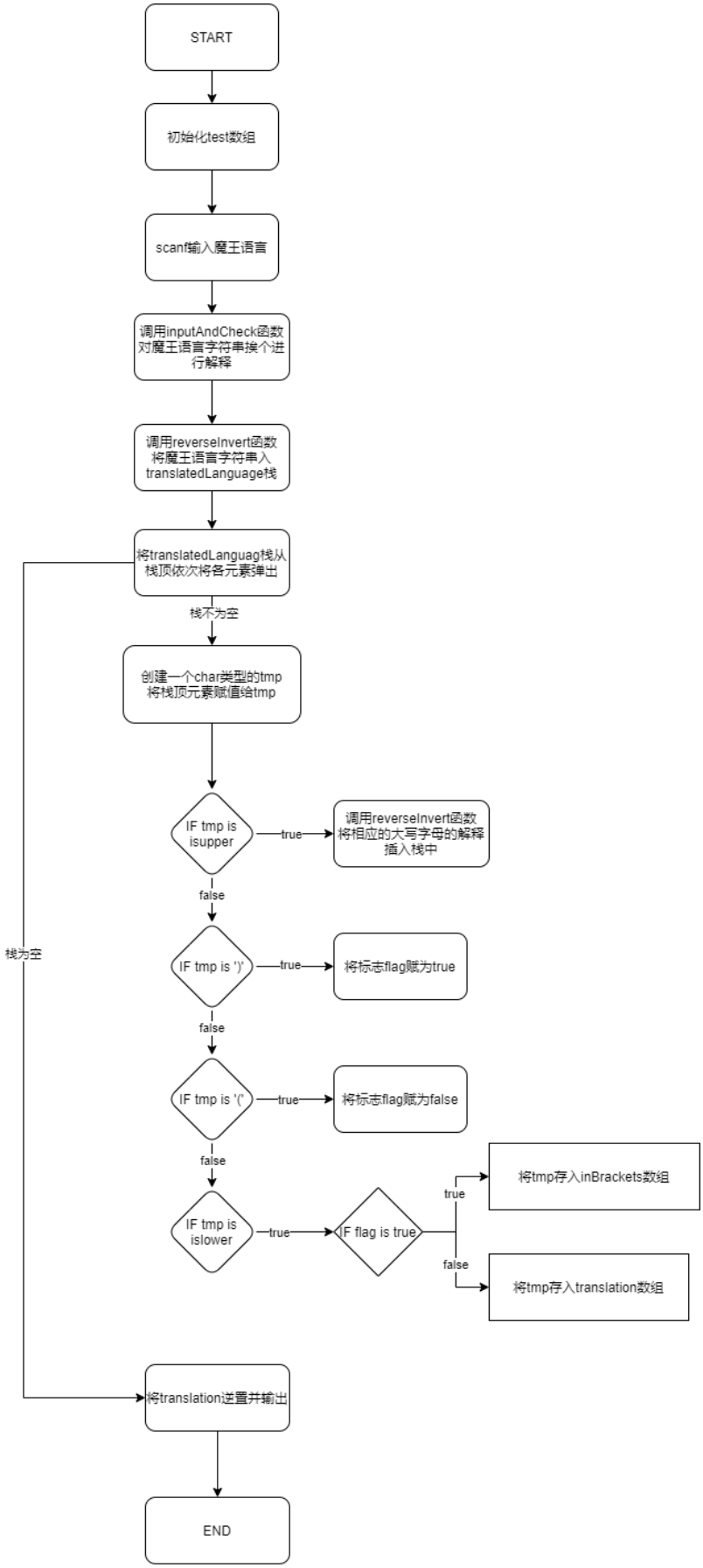
祝子贤 -> reverseStr 函数、reverseInvert 函数、inputAndCheck 函数、main 函数；

2. 数据结构设计

说明程序用到的数据结构的定义，主程序的流程及各模块之间的层次关系

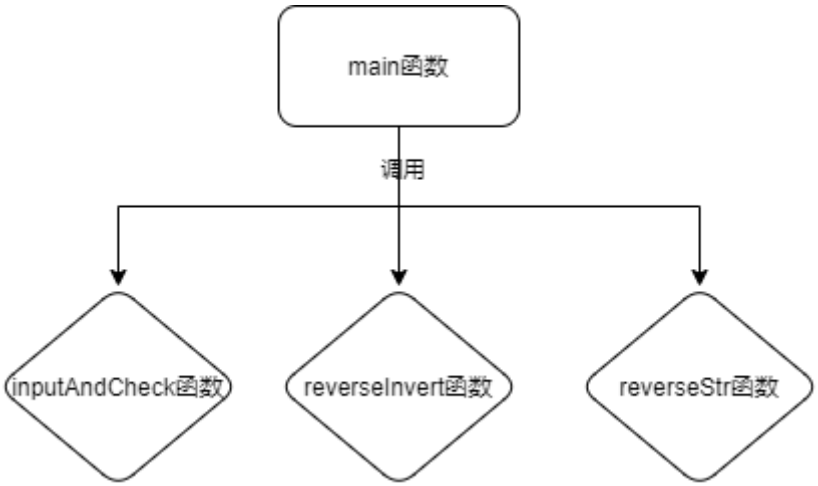
数据结构：栈（用链表实现的，栈顶指针指向最上面的一个元素）

主程序流程（若图片不清楚同 cpp 文件层级下有原图片）：

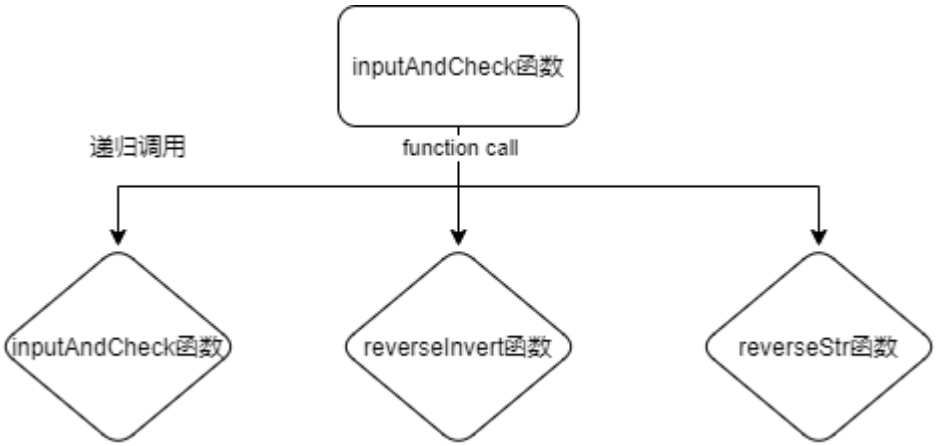


各模块的层级关系（在此只列出有调用其他函数的函数）：

Main 函数：



InputAndCheck 函数：



3. 程序设计

实现概要设计中的数据类型，对主程序、模块及主要操作写出伪代码，画出函数的调用关系

数据类型：

```
// test
    char test[26][MAX_SIZE];         // 存放26个大写英文字母的所对应的小写字母解释
    char language[MAX_SIZE];         // 一开始的魔王语言
    char inBrackets[MAX_SIZE];       // 括号内的字符
    char translatedInBra[MAX_SIZE]; // 转化括号内的字符
    char translation[MAX_SIZE];      // 最终解释的字符
    bool flag = false;               // 是否在括号内的标志
    Stack translatedLanguage;        // 经过大小写转换后的栈
    Stack finalInterp;               // 每个大写字母最终解释的栈
```

主程序伪代码：

```
int main()
{
    char language[10000];           // 一开始的魔王语言
    scanf("%s", language);          // 输入魔王语言
    inputAndCheck(language);        // 将魔王语言的大小写进行转换
    reverseInvert(language, Stack); // 将转换后的魔王语言入栈（现在全部都是小写字母和圆括号）
    while (!Stack.empty())          // 栈不为空就弹出栈顶元素
    {
        char tmp = Stack.top(); // 用char来接收栈顶元素
        Stack.pop();            // 将栈顶元素弹出
        // 判断栈顶元素是（、）、大写字母还是小写字母
        if (tmp == ')') // 为）
        {
            ...
        }
        if (tmp == '(') // 为（
        {
            ...
        }
        if (isupper(tmp)) // 为大写字母
        {
            ....
        }
        else // 为小写字母
        {
            if (fiag) // 弹出的元素是在括号内的
            {
                ...
            }
            if (!flag) // 弹出的元素不是括号内的
            {
                translation[j] = tmp; // 将tmp传给数组第j位
                j++;                  // j移动到下一位
            }
        }
    }
    reverseStr(translation); // 由于弹出的是反序，故需要逆置一下数组
    for (int k = 0; k < translation.length; k++)
    {
        printf("%c", translation[k]);
    }
    return 0;
}
```

模块伪代码：

InputAndCheck 模块：

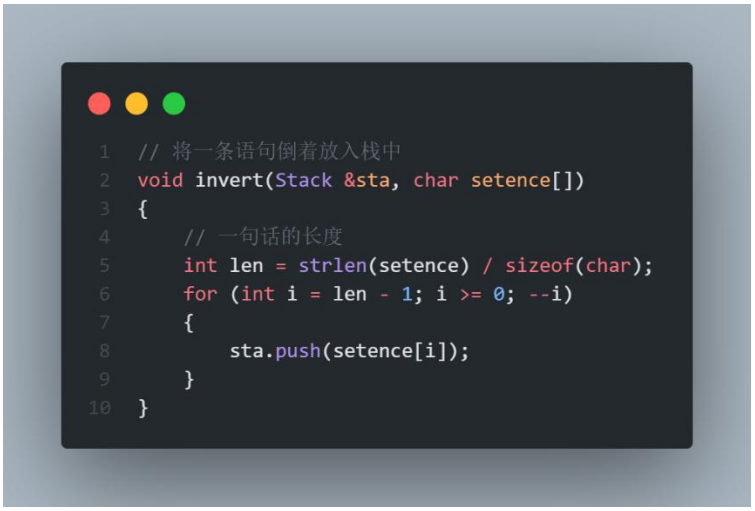
```
// 检查输入的字符是否全为小写字符 ch为输入的字符，index为其字符的ASCII码 - A的ASCII的值，text为预处理后的各大写字母的表示
void inputAndCheck(Stack &sta, char ch, int index, char test[26][MAX_SIZE])
{
    char supperVoca[MAX_SIZE]; // 存放用户输入该大写字母的表示
    int vocaVal;               // 输入解释字符的ASCII码
    char final[MAX_SIZE];      // 最终的解释
    scanf("%s", supperVoca);   // 输入对ch（输入字符）的解释
    // 挨个检查输入的解释是否为小写字母
    for (int i = 0; i < strlen(supperVoca) / sizeof(char); i++)
    {
        vocaVal = supperVoca[i];                                    // 获取当前字符的ascii码值
        if (isupper(vocaVal) && test[vocaVal - 'A'][0] == '\0') // 如果输入的是大写字母且该大写字母并无解释
        {
            ... inputAndCheck(newStrack, supperVoc[i], vocaVal - 'A', test); // 递归调用，需解释supperVoc[i]字符的含义
        }
        if (isupper(vocaVal) && test[vocaVal - 'A'][0] != '\0') // 如果输入的是大写字母且该大写字母有相应的解释
        {
            reverseInvert(); // 将对应大写字母的解释插入栈中
        }
        else
        {
            push(supperVoc[i]); // 为小写字母则将其直接插入栈中 }
        }
    }
    int i = 0;
    // 当栈不为空则一直弹出栈顶元素,i用于记录共有多个数
    while (!sta.isEmpty())
    {
        char tmp = sta.top();
        sta.pop();
        final[i] = tmp;
        i++;
    }
    reverseStr(final); // 由于栈是先进后出，故要将数组逆置
    int j = 0;
    while (j < i)
    {
        test[index][j] = final[j];//将逆置号的解释依次存入数组
        j++;
    }
    test[index][j] = '\0'; // 在每个大写字母的最终解释后加个\0
}
```

操作伪代码：

字符串逆置

```
// 用双指针将字符串逆置
void reverseStr(char str[], int length)
{
    int start = 0;
    int end = length - 1;
    while (start < end) // 当开始指针与结束指针相遇（等于或大于）则停止循环
    {
        swap(str[start], str[end]); //交换两者位置
        start++;
        end--;
    }
}
```

将一句话倒着插入栈中



```
// 将一条语句倒着放入栈中
void invert(Stack &sta, char setence[])
{
    // 一句话的长度
    int len = strlen(setence) / sizeof(char);
    for (int i = len - 1; i >= 0; --i)
    {
        sta.push(setence[i]);
    }
}
```

将一句话正着插入栈中

```
// 将一句话正着放入栈中
void reverseInvert(Stack &sta, char setence[])
{
    // 一句话的长度
    int len = strlen(setence) / sizeof(char);
    // 与invert唯一不同就是循环的条件不同
    for (int i = 0; i <= len - 1; ++i)
    {
        ...
    }
}
```

4. 调试分析

设计实现过程中遇到的问题及如何解决的；算法的复杂度分析；经验体会

遇到的问题：

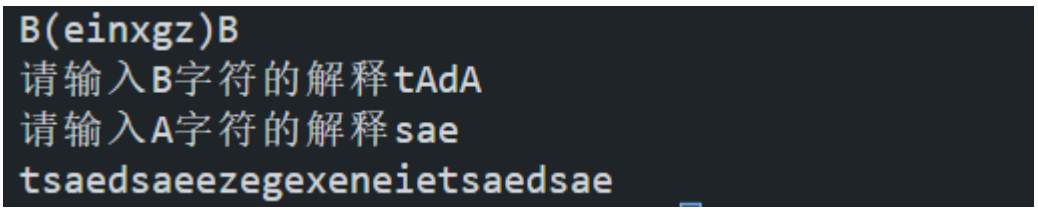
1. 在调用 inputAndCheck 函数时若输入大写字母的解释仍有大写字母将进行递归，递归传入的参数中有栈，需要注意的是不能传入一开始的栈，而是需要新开一个栈并传入也就是代码中的 Stack newSta。如下：用户输入 B(exingz)B -> 用户需要输入 B 的解释 -> 输入 tAdA -> 需要用户输入 A 的解释 -> 输入 sae。如果递归时候不传入一个新的栈，那么对 A 的解释将会变成 tsae。
2. 在调用 inputAndCheck 函数的最后会将对该大写字母的解释存入 test 数组中，若不在最后加上 `\0` 则输出最终字符串时会产生输入多余的乱码的情况如（会输出 tsaed 潪 saeezegexeneiets 笈 aedsae）虽然字符没错，但是会输出多余的乱码。

算法复杂度分析：$T(n) = O(n)$;

5. 测试结果

列出测试结果，包括输入和输出

- INPUT1:
- B(einxgz)B;
- OUTPUT1;
- 请输入 B 字符的解释;
- INPUT2:
- tAdA
- OUTPUT2:
- 请输入 A 字符的解释
- INPUT3:
- sae;
- FINAL OUTPUT:
- tsaedsaeezegexeneietsaedsae



6. 用户使用说明

给出主界面及主要功能界面

7. 附录

给出源程序文件清单，如：

Mowangyuyan.cpp //主程序