

# 程序设计实践实验报告

一种领域特定脚本语言的解释器的设计与实现

姓名：胡敏臻

学号：2019211424

班级：2019211307

日期：2021/12/20

## 1 概述

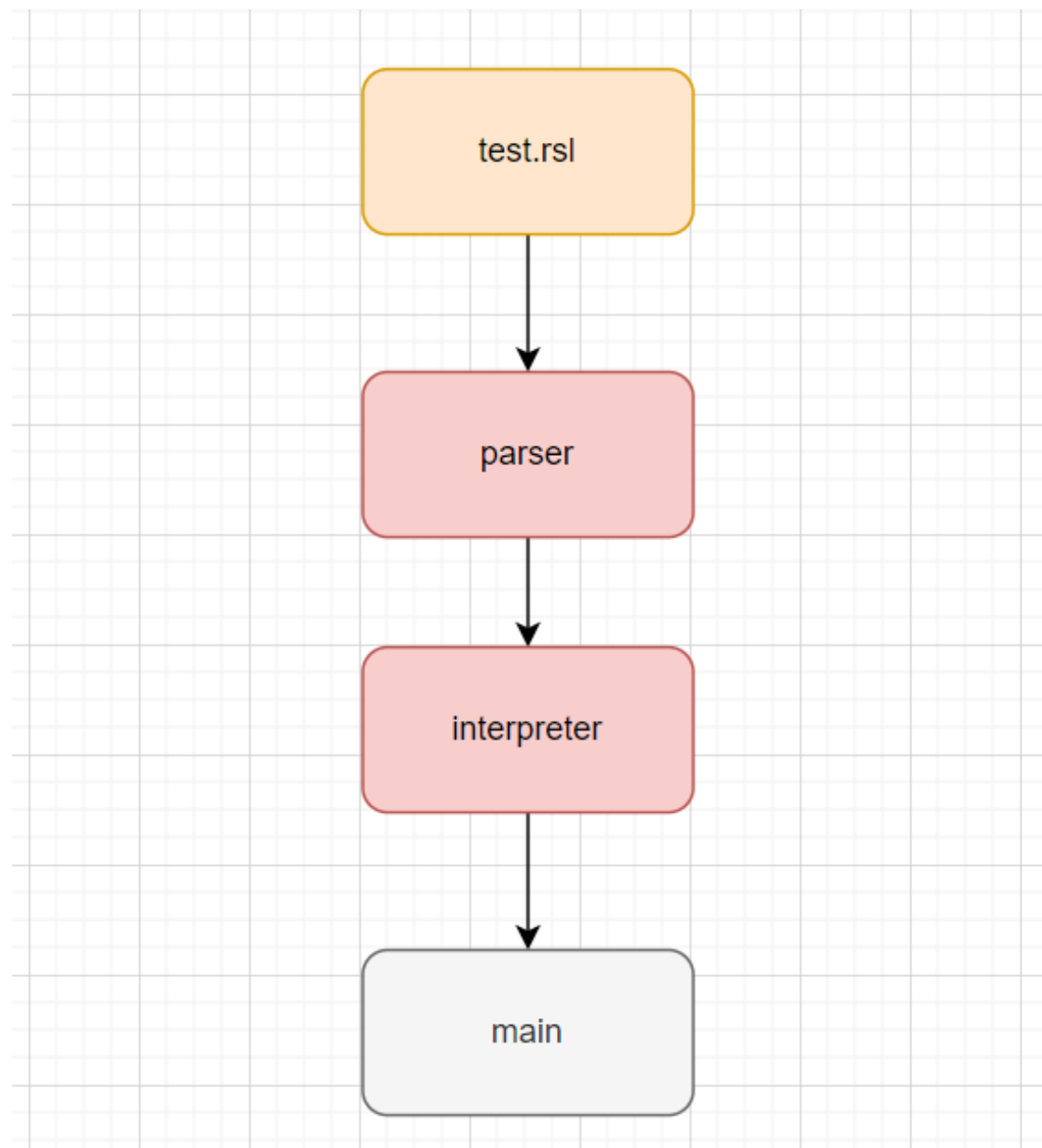
### 1.1 实验内容

定义一个领域特定脚本语言，这个语言能够描述在线客服机器人（机器人客服是目前提升客服效率的重要技术，在银行、通信和商务等领域的复杂信息系统中有着广泛的应用）的自动应答逻辑，并设计实现一个解释器解释执行这个脚本，可以根据用户的不同输入，根据脚本的逻辑设计给出相应的应答。

### 1.2 实验环境

Visual Studio 2019 ， Windows10 64 位

## 2 总体设计



分析 test.rsl 中自己设计的脚本语言。

parser 根据脚本语言生成语法树。

interpreter 根据语法树翻译函数过程，实现运行。

main 中通过命令行调用翻译实例运行。

## 2.1 parser 模块的设计

在 parser 模块中，我们建立了三个类，一个 step 类，用于记录每个 step 中的信息；一个 script 类记录每个 step；还有一个 parser 类用于解析脚本。

**step 类:**

```
class step {
public:
    std::vector<std::string> expression;           //存储speak的变量
    std::vector<int> listen;                       //存储listen的开始计时器
和结束计时器
    std::unordered_map<std::string, std::string> ans_step; //前者为回答，后者为步骤
名
    std::string silence_to;                        //silence情况的步
骤名
    std::string default_to;                       //default情况的步骤
名
    int exit = 0;                                  //为0表示此时不存在退出
程序，为1则表示此时需要退出
};
```

在 step 类中，我们用 expression 记录了 speak 的变量，listen 数组记录 listen 的开始计时器和结束计时器。ans\_step 记录了在此 step 对应输入需要跳转的 stepid。silence\_to 和 default\_to 分别记录了在静音情况和没有情况匹配之下需要跳转的 stepid。exit 记录了此 step 需不需要退出，0 表示没有退出，1 表示需要退出。

**script 类:**

```
class script {
public:
    std::unordered_map<std::string, step> steps; //存储所有step，前者为
step名，后者为step
    std::string entry;                          //存放入口stepid
    std::vector<std::string> vars;              //存放所有变量
};
```

在 script 类中的 steps 记录了所有 step 对应的 stepid 和 step 实例。entry 记录了入口步骤的 stepid。vars 记录了该脚本的所有变量。

**parser 类:**

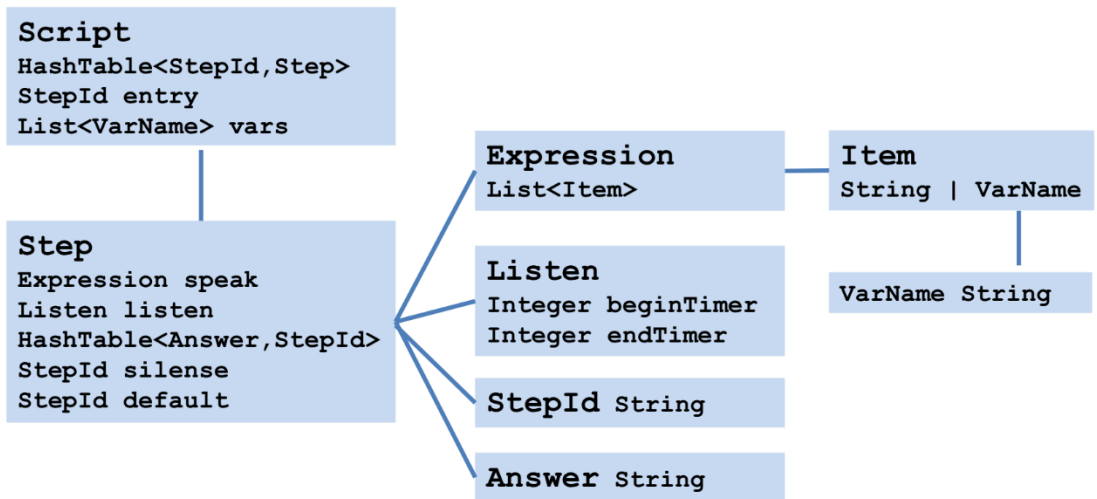
```
class parser
{
public:
    void parse_file(std::string filename); //读取文件
    script Script;                        //Script实例
private:
```

```
std::string old_stepid=""; //上一个step_id名

step tempstep; //当前step
void trim(std::string& s); //删除首尾空白字符
void parse_line(std::string line); //处理一行
void process_tokens(std::vector<std::string>& tokens); //处理一行中的token情况
void process_step(std::string stepid); //step创建过程
void process_speak(std::vector<std::string>& tokens); //语音输出过程
void process_expression(std::vector<std::string>& tokens); //语音输出语句
void process_listen(int start_timer, int stop_timer); //听取语音过程
void process_branch(std::string answer, std::string next_stepid); //分支过程
void process_silence(std::string next_stepid); //安静过程
void process_default(std::string next_stepid); //默认过程
void process_exit(); //退出过程

int string_to_int(std::string str); //字符串转整型
void test_parser(); //测试parser
};
```

parser 类主要用于分析脚本语言，构建语法树。通过 parser\_file() 函数读取脚本文件，将最后分析的语法树建立在 Script 实例上。process\_xxx() 函数是用来解析不同语句的，如创建 step，语音输入输出，听取语音，分支情况等。trim() 函数用来截断前后的空格。string\_to\_int 表示将字符串转为整型。test\_parser() 是检查此时的语法树建立的是否正确。起到测试桩的作用。最后设计出的数据结构如下：



## 2.2 interpreter 模块的设计

interpreter 用于翻译此时的程序。

```
class interpreter
{
    std::unordered_map<std::string, std::unordered_map<std::string, std::string>>
var_table;    //变量表
    parser p;    //parser实例，传递语法树
    step tstep;    //当前step
public:
    std::string key_words;    //此时输入的识别主键
    void init();    //初始化，从database中读数据
    void run();    //翻译过程
    void inter_expression(std::vector<std::string>& exps);    //翻译表达式内容
    void inter_listen(std::vector<int> listen);    //读取输入操作
};
```

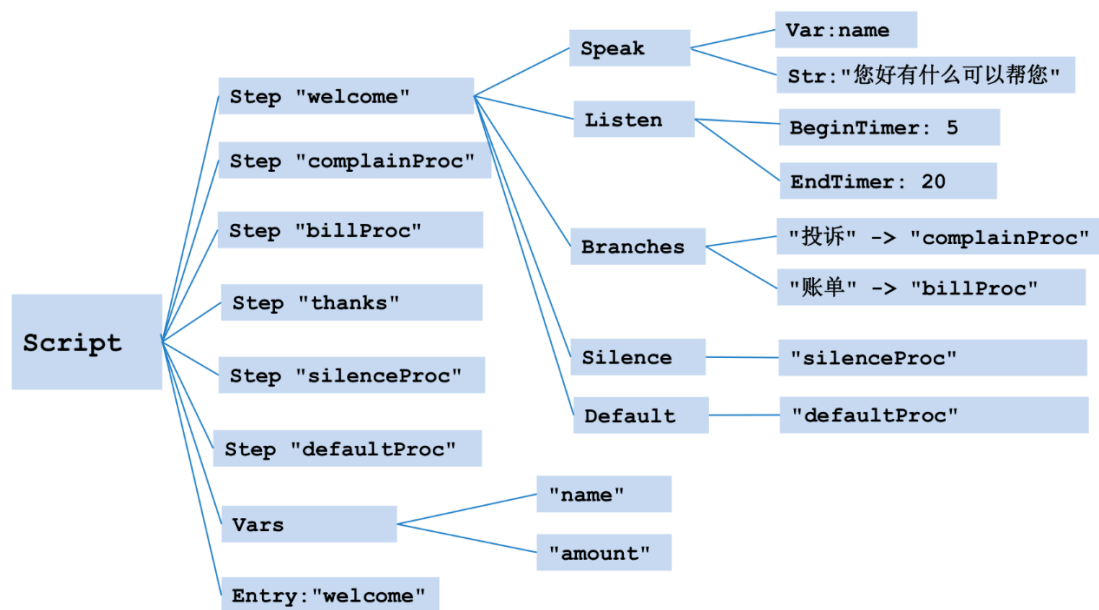
var\_table 用于存储变量，在处理前先预处理变量，将该程序的变量先从 database.txt 中读取出来。p 记录此时的 parser 实例。key\_words 用于记录此时通过命令行传入的主键。inter\_expression 翻译此时表达式的内容，inter\_listen 用于执行此时的读取输入操作。

## 3 设计说明

1、首先，我们设计出了一个自己的脚本，我们需要通过 parse 解析脚本，构建脚本语法树。

接下来我们需要设计如何实现构建语法分析树：通过 parse\_file() 读取文件，在文件中一行一行读取，并忽略行尾行首的空白，忽略空行，忽略“#”开头的注释行，一行一行处理脚本。parse\_line 读取一行中空白分割，将他们构建为不同的 token。根据现在 token 的情况选择调用，即 token[0] 分情况处理。若为 Step 则调用 process\_step() 创建一个新的 step，若为 Speak，则将对表达式存入当前的 Expression。若为 Listen，则调用 process\_listen()，若为 Branch，则建立对应的回答和步骤名称，存入当前的哈希表中。若为 Silence, Default，则记录他们需要跳转的 stepid。若为 Exit，则将 step 的 exit 置为 1，表示此步骤需要退出。

最后解析出的语法树如下：



2、在得到语法分析树以后，我们需要翻译该语法树。翻译设计思路如下：

获取脚本语法树，创建执行环境。当前 Step 置为 entryStep，循环针对当前 Step 做：执行 Speak（输出到标准输出）。如果本步骤是终结步骤，则结束循环，断开通话。否则执行 Listen（直接从标准输入读入用户意愿），获得下一个 StepId；如果用户沉默，那么获得 Silence 的 StepId。根据用户意向查找 HashTable，获得 StepId。如果查不到，则获得 Default 的 StepId，将当前 Step 置为刚才获得的 StepId 对应的 Step。

那么我们如何获得语法树，**parser** 和 **interpreter** 之间的接口如何设定。我们通过实例 parser 来传递语法树，这个参数为 parser 的实例，通过这个实例，我们在解析的过程构建了语法树，并将其存储在了 parser 的实例中，而这个实例时在翻译过程中调用的，这样就把语法树传递过来了。

根据以上步骤即可完成语法树的翻译，下面为 interpreter 的主要过程：

```

void interpreter::run() {
    p.parse_file("test.rsl");
    init();
    tstep = p.Script.steps[p.Script.entry];
    if (var_table.find(key_words) == var_table.end()) {
        std::cout << "用户不存在\n";
        return;
    }
    hThread1 = (HANDLE)_beginthreadex(NULL, 0, Fun1Proc, NULL, 0, NULL);
    SuspendThread(hThread1);
    int repeat = 0; //repeat记录沉默的次数，连续超过三次则退出
    std::string ans = "";
    while (true) {
        inter_expression(tstep.expression); //先翻译speak
        std::cout << "\n";
        if (tstep.exit == 1) { //如果是结束步骤则断循环
            break;
        }
    }
}
  
```

```

    }

    inter_listen(tstep.listen);
    ans = input_str;
    if (ans == "") { //如果什么都
没读到为silence
        if (repeat < 3) {
            repeat++;
        }
        else {
            break;
        }
        tstep = p.Script.steps[tstep.silence_to];
    }
    else if (tstep.ans_step.find(ans) != tstep.ans_step.end()) { //如果在
branch中能找到
        repeat = 0;
        tstep = p.Script.steps[tstep.ans_step[ans]];
    }
    else { //如果无法识
别要求则转default
        repeat = 0;
        tstep = p.Script.steps[tstep.default_to];
    }
}

std::cout << "程序结束\n";
}

```

3、接下来我们需要设计**人机接口**，我们将人机接口设置为命令行模式。我们通过调用参数-k 或者—key 来调用参数，后面跟着标识符主键，用于标识唯一的一个用户。

```

int main(int argc ,char ** argv) {

    if (argc < 3) {
        std::cout << "missing parameter\n";
    }
    else {
        std::string t_arg = std::string(argv[1]);
        if (t_arg == "-k" || t_arg == "--key") {
            interpreter inter;
            inter.key_words = argv[2];
            inter.run();
        }
        else {
            std::cout << "wrong parameter\n";
        }
    }
}

```

```

    }

}

return 0;
}

```

运用这个接口最后调用程序的过程如下：

```

F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>脚本语言解释器.exe -k 2019211424
胡敏臻您好，请问有什么可以帮助您？请在听到滴的一声时开始说话
滴
您的语言说话时间结束
听不清，请你大声一点可以吗？请在听到滴的一声时开始说话
滴
查成绩
您的语言说话时间结束
您的语文成绩为99分，您的数学成绩为98分。祝你能收获一个满意的成绩。再见！
程序结束

```

若接口调用错误会报错。会根据情况，报参数错误或者参数丢失。

```

F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>脚本语言解释器.exe -key 2019211424
wrong parameter

F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>脚本语言解释器.exe --key
missing parameter

```

## 4 测试说明

### 4.1 测试脚本设计：

根据老师提供的的投诉和查账单的客服机器人问答外，还设计了另一个相似功能的客服机器人。用于给学生们查询成绩和给课程提供建议。具体实现如下：

```

#起始步骤
Step Welcome
    Speak $name + "您好，请问有什么可以帮助您？请在听到滴的一声时开始说话"
    Listen 5 20
    Branch "查成绩" scoreProc
    Branch "提建议" adviceProc
    Silence silenceProc
    Default defaultProc

#处理提建议的步骤
Step adviceProc
    Speak "请问您有什么想与我们交流的吗？请提出你宝贵的建议。请在听到滴的一声时开始说话"
    Listen 5 50
    Silence silenceProc
    Default thanks

#结束退出的步骤
Step thanks
    Speak "感谢您的来电，再见"

```



```

Exit

#处理查成绩的步骤
Step scoreProc
    Speak "您的语文成绩为" + $chinese + "分，您的数学成绩为" + $math + "分。祝你能收获一个满意的成绩。再见！"
    Exit

#如果用户不说话的步骤
Step silenceProc
    Speak "听不清，请你大声一点可以吗？请在听到滴的一声时开始说话"
    Listen 5 20
    Branch "查成绩" scoreProc
    Branch "提建议" adviceProc
    Branch "再见" thanks
    Silence silenceProc
    Default defaultProc

#如果没有匹配的的步骤
Step defaultProc
    Speak "抱歉，暂时没有理解您的要求，能再说一遍吗？请在听到滴的一声时开始说话"
    Listen 5 20
    Branch "查成绩" scoreProc
    Branch "提建议" adviceProc
    Branch "再见" thanks
    Silence silenceProc
    Default defaultProc

```

## 4.2 测试样例说明

### • 测试样例 1

基本功能提建议测试

提建议  
希望可以加强实践教学

### • 测试样例 2

基本功能查成绩测试

什么

查成绩

### • 测试样例 3

连续 3 次输入为空，检验是否正常退出

- 测试样例 4

不存在的主键输入，在数据库中找到

脚本语言解释器.exe -k 2019211423 <input1.txt >output4.txt

- 测试样例 5

多次输入错误后，输入再见退出

想查成绩  
有点犹豫  
算了不查吧  
再见

### 4.3 自动脚本测试

因为要实现自动脚本，想到可以通过批处理的方式来实现多个测试的自动运行。在之前先准备好测试 input 文件和对应的答案 ans 文件，再比较 output 文件与 ans 文件是否一致。


- ans 文件

 ans1.txt	2021/12/18 11:07
 ans2.txt	2021/12/18 11:21
 ans3.txt	2021/12/19 17:27
 ans4.txt	2021/12/19 17:27
 ans5.txt	2021/12/19 17:27

- input 文件

 input1.txt	2021/12/18 11:10
 input2.txt	2021/12/18 11:21
 input3.txt	2021/12/19 17:27
 input5.txt	2021/12/19 17:27

- output 文件

 output1.txt	2021/12/31 11:14
 output2.txt	2021/12/31 11:15
 output3.txt	2021/12/31 11:16
 output4.txt	2021/12/31 11:16
 output5.txt	2021/12/31 11:17

建立 script\_test.bat 文件，文件内容如下：

脚本语言解释器.exe -k 2019211424 <input1.txt >output1.txt

```

fc output1.txt ans1.txt
脚本语言解释器.exe -k 2019211424 <input2.txt >output2.txt
fc output2.txt ans2.txt
脚本语言解释器.exe -k 2019211424 <input3.txt >output3.txt
fc output3.txt ans3.txt
脚本语言解释器.exe -k 2019211423 <input1.txt >output4.txt
fc output4.txt ans4.txt
脚本语言解释器.exe --key 2019211424 <input5.txt >output5.txt
fc output5.txt ans5.txt
pause

```

第一个语句为运行程序，脚本语言解释器为生成的可执行文件，-k 表示需要输入的参数，<后跟输入文件，>后跟输出文件。fc 命令用于比较这两个文件的差异。pause 表示停止，防止运行后 bat 文件直接结束。运行结果如下：

```

C:\WINDOWS\system32\cmd.exe
F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>脚本语言解释器.exe -k 2019211424 0<input2.txt 1>output2.txt
F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>fc output2.txt ans2.txt
正在比较文件 output2.txt 和 ANS2.TXT
FC: 找不到差异

F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>脚本语言解释器.exe -k 2019211424 0<input3.txt 1>output3.txt
F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>fc output3.txt ans3.txt
正在比较文件 output3.txt 和 ANS3.TXT
FC: 找不到差异

F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>脚本语言解释器.exe -k 2019211423 0<input1.txt 1>output4.txt
F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>fc output4.txt ans4.txt
正在比较文件 output4.txt 和 ANS4.TXT
FC: 找不到差异

F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>脚本语言解释器.exe --key 2019211424 0<input5.txt 1>output5.txt
F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>fc output5.txt ans5.txt
正在比较文件 output5.txt 和 ANS5.TXT
FC: 找不到差异

F:\zzz\作业\程序设计\大作业\脚本语言解释器\Debug>pause
请按任意键继续...

```

根据结果可以看到此时每一个命令都找不到差异，说明此时运行的结果与预期相符，表示此时结果正确。

#### 4.4 测试桩说明

本实验内容在于设计一个脚本语言解释器，而我们不知晓此时的语音翻译结果和最后输出的媒体语音合成。于是我们设置自己的语音翻译结果，即为我们设计的 input 文件中的内容。而因为我们不知道媒体语音合成的结果，所以我们将最后的程序设计的结果输入到 output 文件中。

除将我们之间设计的脚本语言解释器看成一个整体之外，我们还可以将切其成两个部分，一个是语法树生成部分，一个是翻译部分。在生成语法树之后，做了 test\_parser() 操作，检验此时的语法树是否正确。

```

//测试parser
void parser::test_parser() {
    auto it = Script.steps.begin();
    for (it; it != Script.steps.end(); it++) {
        step tstep=(*it).second;
        std::cout << tstep.expression.size() << "\n" << tstep.listen.size()<<"\n";
    }
}

```

```

        std::cout << tstep.silence_to << "\n" << tstep.default_to << "\n\n";

    }

}

```

## 5 实验总结

关于本实验的记法见附件：RSL 脚本语言记法。

本次实验总花费时间较长。因为一开始没有明白需要怎么设计和翻译脚本语言。但是在上手之后就比较快了。根据老师提供的样例的设计脚本和设计思路，开始写的时候总花费还是比较快的。代码中花费时间最长的在 Listen 函数的翻译，过程中尝试了很多方法，最后是采用了多线程的方法解决的。

```

//工作线程，完成读取输入操作
unsigned __stdcall Fun1Proc(void* pArguments) {

    while (1) {
        if (flag == 1) {
            gets_s(str, maxsize-1);
            input_str = std::string(str);
            flag = 0;
        }
    }

    return 0;
}

```

这是读取工作的线程，若此时能够读取，则会读入 gets\_s，否则会一直在读取这里等待。在外部则需要中断此线程。

```

void interpreter::inter_listen(std::vector<int> listen) {
    Sleep(listen[0] * 1000);
    std::cout << "滴~~~\n";
    int lastime = listen[1] - listen[0];
    input_str = "";
    flag = 1; //置flag为1，激活输入
    ResumeThread(hThread1); //重新激活线程
    Sleep(lastime * 1000);
    SuspendThread(hThread1); //悬挂线程

    std::cout << "您的语言说话时间结束\n";
}

```

在外部 ResumeThread(hThread1)用于重新激活线程。而 SuspendThread(hThread1)用户悬挂线程。Listen 操作则不停对线程做激活和悬挂操作。

除了代码编写之外，对程序设计的过程也更为清晰了。特别是对代码的测试和测试桩更为了解了。第一次尝试用批处理的方式来测试程序，学习到了很多。并且对记法也更为了解。之前也从来也没有想过可以自己设计一门语言，最后完成的时候还是非常有成就感的。