形式语言与自动机课程实验报告

姓名	学号	邮箱
吕玉龙	191220076	1931015836@qq.com

一、实验完成度

完成了实验的所有任务；

完成了解析器，并实现了在给定`-v|--verbose`参数时给出不符合语法的图灵机程序片段及相关提示的任务；

同时也对参数的个数以及格式不对的输入情况给出了简单的提示；完成了模拟器的普通模式和`-v|--verbose`模式；

实现了求两数最大公约数的多带图灵机程序

二、实验结果

解析器的错误处理：在实现时设定为检测到有一处语法错误整个程序就停止报错以防止因某些错误导致程序运行产生连锁错误。

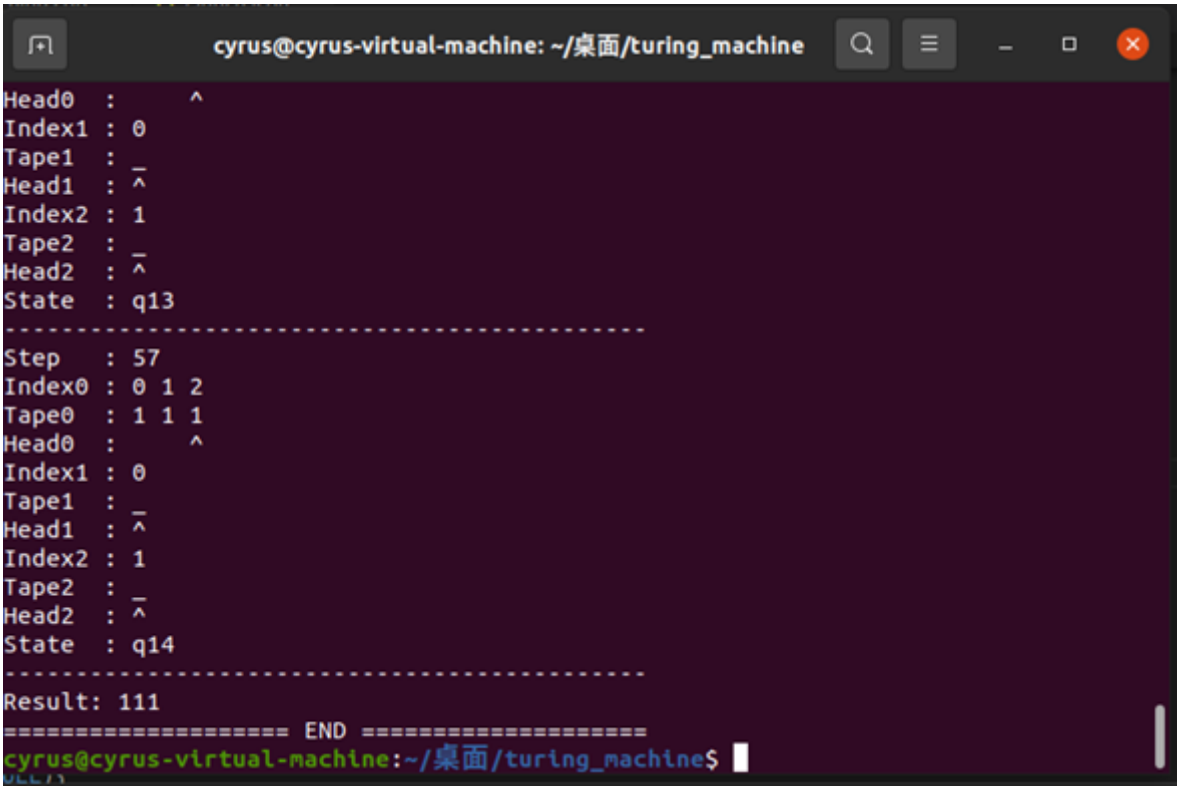
命令行工具使用：

```
cyrus@cyrus-virtual-machine:~/桌面/turing_machine$ ./turing --help
usage: turing [-v|--verbose] [-h|--help] <tm> <input>
cyrus@cyrus-virtual-machine:~/桌面/turing_machine$
```

示例图灵机程序正确运行：例如对于正确输入：

```
cyrus@cyrus-virtual-machine:~/桌面/turing_machine$ ./turing gcd.tm 1110111111
111
cyrus@cyrus-virtual-machine:~/桌面/turing_machine$
```

verbose运行结果如下：



3和6的最大公约数是3

三、实验分析与设计思路

实验首先分三个部分完成：读取参数、解析器、模拟器

读取参数

在main函数中利用传参，读取参数之后，合成一个string，进行处理，这里是为了可以处理空串的情况。然后丢给splitcommand处理。

```
223
224 int main(int argc, char* argv[])
225 {
226     string str;
227     string turingMachine,Input;
228     for (int i = 0;i<argc;i++){
229         string temp = argv[i];
230         temp.push_back(' ');
231         str+=temp;
232     }
233     //cout<<str<<endl;
234     //cout<<"bad\n";
235     splitCommand(str,&turingMachine,&Input);
236
237     //cout<<"good\n";
```

splitcommand函数判断完输入是否合法之后，返回图灵机的名称以及input。重点在于利用strtok(command," ");这个strtok函数进行字符串分割，然后进行分割之后的检查。

解析命令之后，如果是合法输入，则尝试载入图灵机，利用get_TuringMachine来读取图灵机，如果图灵机不合法，则会直接exit（-1）并打出"syntax error"，否则用得到的图灵机检查是否是合法输入，如果是，则交给图灵机的run函数去模拟即可。

```
    //cout<<"good\n";
    //cout<<"turingmachine = "<<turingMachine<<" input = "<<Input<<endl;
    if(!ishelp){
        TuringMachine TM = get_TuringMachine(turingMachine,isverbose);
        if(TM.checkinput(Input)){
            TM.run(Input);
        }
        else{
            exit(-1);
        }
    }
    return 0;
}
```

解析器

进入splitcommand函数之后，进行解析器的运行

首先需要对输入的命令进行解析，包括命令的错误输入、-h、-v等参数的获取。

先判断是否输入./turing；随后判断跟在后面的参数是-h，-v还是图灵机，如果是-h，那么后面的串要为空，否则认为它不合法，如果是verbose，那么将会标记verbose模式，对于之后的图灵机初始化有用。还要判断之后的图灵机参数末尾是否为".tm"，不是的话，也认为是不合法的命令。

针对不合法命令，输出提示：“illegal command”

以下为get_TuringMachine函数，它尝试打开图灵机文件并读入图灵机

```
    ifstream in(turingmachine);
    vector<string> turingText;
    string
setOfState,setOfInput,setOfTape,startState,blankSymbol,setOfFinish,numOfTape;
    vector<string> transfunc;
    if(in.is_open()){
        while(!in.eof())
        {
            string temp;
            getline(in,temp);
            turingText.push_back(temp);
        }
    }
    else{
        cout<<"syntax error\n";
        exit(-1);
    }
```

读入之后，则会将各个参数传递给TM类进行图灵机初始化

```
TuringMachine TM =
TuringMachine(setOfState,setOfInput,setOfTape,startState,blankSymbol,setOfFinish
,numOfTape,transfunc,isVerbose);
    if (!TM.isTM()){
        //cout<<"wrong tm\n";
        cout<<"syntax error";
        cout<<endl;
        exit(-1);
    }
    else{
        //cout<<"correct tm\n";
        return TM;
    }
```

图灵机类展示如下：

```
class TuringMachine
{
private:
    vector<string> State;
    string startState;
    vector<string> inputSymbol;
    vector<string> tapeSymbol;
    string blankSymbol;
    vector<string> endState;
    int tapeCount;

    vector<Tape> setOfTape;

    vector<pair<string,vector<Transition>>> transFunc;
    bool isverbose;
    string currentState;
    bool successBuild;
    int stepCount;
public:
    TuringMachine(string setOfState,string setOfInput,string setOfTape,string
startState,string blankSymbol,string setOfFinish,string numOfTape,vector<string>
transfunc,bool isVerbose);
    string deleteExplanation(string str);
    void stringSplit(string str,int pos,string name);
    bool checkLegalTapeAndInput();
    bool checkStartstateAndFinishstateInState();
    bool isTM();
    void printState();
    void printTransitionFunc();
    bool checkinput(string inputString);
    void run(string inputString);
    Transition findNextMove();
    void moveToNext(Transition transition);
    void inVerboseMode();
    bool checktransition();
};
```

deleteExplanation函数用来消除行注释以及注释，对于消除完行注释的string参数（除去迁移函数），都使用stringsplit函数进行分割成子string并存在对应的vector中。例如setOfState串分割完之后会存在vector State中。

对于这些参数处理的时候，只要有不符合规则的情况，那么 bool successBuild都会置成false，利用isTM函数返回successbuild来判断图灵机是否建立成功。

对于迁移函数vector中的结构思路是key-value，key为旧状态，value为这个状态对应的所有迁移函数。这里使用了另一个结构体

```
struct Transition
{
    string oldState;
    string newState;
    string oldSymbol;
    string newSymbol;
    string direction;
};
```

如果参数都读入正确，那么要判断这个图灵机是否符合逻辑，使用如下三个函数判断，分别是

```
bool checkLegalTapeAndInput();
bool checkStartstateAndFinishstateInState();
bool checktransition();
```

意义在于，tape符号有没有不合法的符号，同时input符号是否都在tape中；开始状态与接受状态是否都在状态集中，对于迁移函数，旧状态是否在状态集中，以及方向是否合法，新符号组是否合法。

如果都没有问题，那么successbuild应置为true，然后可以进行图灵机的模拟。

模拟器

这里利用一个tape结构体来维护每条带上的信息。

```
struct Tape
{
    string tape;
    int index_start;
    int index_end;
    int pointer;
};
```

tape表示磁带上两个index之间的所有符号，可以包括空串，pointer则标识磁带头的位置。

使用checkinput函数判断输入是否合法之后，使用run函数进行图灵机的模拟

run函数首先要对每条磁带进行初始化，随后包含三个流程：

```
while(true){
        if(isverbose){
            inVerboseMode();
        }
        Transition nextTransition = findNextMove();
        if (nextTransition.oldState == "notfound"){
            //cout<<"end\n";
            break;
        }
        else{
```

```
            moveToNext(nextTransition);
            stepCount++;
        }
        //if(stepCount == 14) break;
    }
```

若在verbose模式下，则逐步打印，随后根据当前磁带的读头指向，找到对应的迁移函数，随后进行移动即可。

verbose模式下的打印规则全部按照手册中的规则来，具体不详细展示。

找下一步的迁移函数也相对简单，因为使用了key-value对，所以很容易找到，麻烦一点的在于移动，因为需要考虑读头超过indexstart或者indexend以及读头至某个index之间都为blank symbol时需要对tape串进行更新的情况。

对于第一个情况，直接进行如下的处理

```
        if(setOfTape[i].pointer > setOfTape[i].index_end){
            setOfTape[i].tape.push_back('_');
            setOfTape[i].index_end = setOfTape[i].pointer;
        }
        if(setOfTape[i].pointer < setOfTape[i].index_start){
            string newb = "_";
            newb = newb + setOfTape[i].tape;
            setOfTape[i].tape = newb;
            setOfTape[i].index_start = setOfTape[i].pointer;
        }
```

直接扩展tape即可，对于第二种情况，处理的思路是检查从head处开始往两侧检查，如果发现从某个位置开始出现连续对blank symbol直到结束，那么删去这么多blank symbol即可。这里的代码量较多，就不展示了。

至此，模拟器的内容实现完毕。

多带图灵机程序

见gcd.tm文件中的注释

四、思考与感想

完成本次实验，我对于确定性图灵机这一模型有了更为直观的理解与掌握，也对确定性图灵机可以解决的问题有了更直接的了解。

此外，设计多带图灵机程序的任务也让我对于图灵机解决问题的思路设计以及转移函数的定义实现有了更多的体验。

当然，我觉得可以对于命令的格式做一个更严格的规范，例如这里

```
usage: turing [-v|--verbose] [-h|--help] <tm> <input>
```

可以说的更清楚一些，是否对于参数的顺序有要求。

同时可以给一个不带通配符的图灵机例子，因为我个人没有实现通配符，所以测试时候是自己写了另一个图灵机作为测试的。