形式语言与自动机课程实验报告

姓名：王瀚霖
学号：181860092
邮箱：603102048@qq.com

一、实验完成度

完成了实验的所有任务：完成了解析器，并实现了在给定了 `-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；也对参数个数以及格式不对的情况给出了简单的提示；完成了模拟器的普通模式与`verbose`模式；完成了L_1与L_2两个多带图灵机程序。

二、实验结果

解析器的错误处理：

实现了如下的图灵机程序`palindrome_detector_2tapes_wrong.tm`对解析器的错误处理进行部分测试：

```
; the finite set of states
   #Q = {0,cp,cmp,mh,accept,accept2,accept3,accept4,halt_accept,reject,reject2,reject3,reject4,reject5,halt_reject,}
//错误：halt_reject,}有多余的','

; the finite set of input symbols
#S = {0,1,,2}
//错误：有',,'

; the complete set of tape symbols
#G = {0,1,_,t,r,u,e,f,a,ls}
//错误：字符集中出现了字符串ls

; the start state
#q0 = 2
//错误：初始状态不在状态集中

```

```
; the blank symbol
#B = x
//错误：'x'不在纸带字符集中

; the set of final states
#F= {halt_accept}
//错误：'='前少了空格

; the number of tapes
#N = 2

; the transition functions

;State 0: start state
   0 0_ 0_ **
//错误：转移函数的组成元素个数为4，少了new state
0 1_ 1_ ** cp x
//错误：转移函数的组成元素个数为6，多了
whl __ __ ** accept ; xxxx
  0 0_ 0_ ** whl
//错误：不存在状态'whl'
0 1 1_ ** cp
//错误：当前读写头所指的字符个数与纸带数不一致
0 _q ___ ** accept
//错误：要写到纸带上的字符个数与纸带数不一致
    //且'q'不属于纸带字符集
  0 0_ 0__ ** cp
0 1_ 1_ t* cp
//错误：不存在head的移动方式为't'
0 __ __ **y accept ; xxx
//错误：描述head移动方式的字符串长度与纸带数不一致，
    //且不存在head的移动方式为'y'
```

执行命令`./turing -v palindrome_detector_2tapes_wrong.tm 110011`，结果为：

```
whl@ubuntu:~/fla$ ./turing -v palindrome_detector_2tapes_wrong.tm 110011
syntax error :nothing exits before '}';the last char in line:#Q = {0,cp,cmp,mh,accept,accept2,accept3,accept4,halt_accept,reject,reject2,reject3,rejec
t4,reject5,halt_reject,}
syntax error :nothing exits before ',';the 10th char in line:#S = {0,1,,2}
syntax error :not string but char appears in G: #G = {0,1,_,t,r,u,e,f,a,ls}
syntax error :#F= {halt_accept} doesn't match '#F = {f1,f2,...,fn}'
syntax error :0 0_ 0_ **; as a delta the num of parameters is wrong
syntax error :0 1_ 1_ ** cp x; as a delta the num of parameters is wrong
syntax error :2 appears in q0 but not in Q
syntax error :x is blank symbol but not in G
syntax error :state whl appears in delta:whl __ __ ** accept but not in Q
syntax error :state whl appears in delta:0 0_ 0_ ** whl but not in Q
syntax error : symbols 1 don't match tape number in delta:0 1 1_ ** cp
syntax error :q is current/rewrite symbol in delta: 0 _q ___ ** accept , but not in G
syntax error : symbols ___ don't match tape number in delta:0 _q ___ ** accept
syntax error : symbols 0__ don't match tape number in delta:0 0_ 0__ ** cp
syntax error :t is move symbol in delta: 0 1_ 1_ t* cp , but not l or r or *
syntax error :move symbols **y don't match tape number in delta:0 __ __ **y accept
syntax error :y is move symbol in delta: 0 __ __ **y accept , but not l or r or *
whl@ubuntu:~/fla$
```

图灵机程序正确的运行结果举例如下：

执行命令`./turing -v palindrome_detector_2tapes.tm 110011`，结果为：

```
Tape0  : 1 1 0 0 1 1
Head0  : ^
Index1 : 0 1 2 3 4 5
Tape1  : 1 1 0 0 1 1
Head1  :           ^
State  : mh
-------------------------------------------
Step   : 14
Index0 : 1 0 1 2 3 4 5
Tape0  : _ 1 1 0 0 1 1
Head0  : ^
Index1 : 0 1 2 3 4 5
Tape1  : 1 1 0 0 1 1
Head1  :           ^
State  : mh
-------------------------------------------
Step   : 15
Index0 : 0 1 2 3 4 5
Tape0  : 1 1 0 0 1 1
Head0  : ^
Index1 : 0 1 2 3 4 5
Tape1  : 1 1 0 0 1 1
Head1  :           ^
State  : cmp
-------------------------------------------
Step   : 16
Index0 : 1 2 3 4 5
Tape0  : 1 0 0 1 1
Head0  : ^
Index1 : 0 1 2 3 4
Tape1  : 1 1 0 0 1
```

```
-------------------------------------------
Step   : 24
Index0 : 6 7 8
Tape0  : t r _
Head0  :     ^
Index1 : 1
Tape1  : _
Head1  : ^
State  : accept3
-------------------------------------------
Step   : 25
Index0 : 6 7 8 9
Tape0  : t r u _
Head0  :       ^
Index1 : 1
Tape1  : _
Head1  : ^
State  : accept4
-------------------------------------------
Step   : 26
Index0 : 6 7 8 9
Tape0  : t r u e
Head0  :       ^
Index1 : 1
Tape1  : _
Head1  : ^
State  : halt_accept
-------------------------------------------
Result: true
==================== END ====================
whl@ubuntu:~/fla$
```

三、分析与设计思路

- ## 图灵机的数据结构：

首先编写程序`turing.h`，在其中定义要使用的数据结构；之后功能函数的声明也在这里完成。

参考实验手册中对图灵机程序语法的描述，可以得到一个图灵机要包含状态集、输入字符集、纸带字符集、初始状态、空格符、终结状态集与转移函数。一个状态可以用一个`string`进行描述，一个字符则用`char`进行描述，所以我分别使用了`string`与`char`来存储初始状态与空格符；对于状态集与终结状态集我使用了结构`vector<string>`进行描述，对于输入字符集、纸带字符集我使用了`vector<char>`结构进行存储；这样就与除转移函数之外图灵机的内容完成了对应。

转移函数的存储较为复杂，无法利用单一的数据结构进行描述，所以应该使用自定义结构进行描述。根据功能部分不同，我们可以将一个转移函数拆分为两部分：第一部分包括当前的状态与当前所有纸带的`head`指向的内容，它指明了在确定性图灵机下该转移函数何时会被执行；第二部分包括要写的字符、`head`移动的方向以及转移函数的目标状态，它指明了接下来图灵机的行为状态。考虑到在图灵机运行的时候每次转移都要寻找当前状态下可以执行的转移函数，所以转移函数应该采用一种容易根据第一部分进行搜索的数据结构。按照这个思路，我选择了`C++`的`map`结构存储转移函数，其中key是一个二元组`pair<string,string>`，表示第一部分的内容；`value`则是第二部分的内容，在实现中构造了结构`trans_way`来存储第二部分的内容：

```
struct trans_way //the trans way for a certain delta
{
    string rewrite_symbols;
    //symbols that will take place of current_symbols
    string move;
    //tells how heads will move next
    string new_state;
};
```

这样转移函数就可以构造为：

```
struct delta
{
    map<pair<string,string>,trans_way> deltas;
    //pair<string,string> is
    //<old_state,current_symbols
    //(symbols processed by heads of types now)
    //>,the state before transforming with a certain delta
};
```

利用以上结构就可以对图灵机的各种初始状态信息进行存储了。这里要注意的是在解析器中要对转移函数是否合法(例如涉及的状态是否都在状态集中)进行判断，而这样的判断需要先存储状态集、纸带字符集、纸带数等信息。如果在`<tm>`文件中转移函数定义在这些内容之前，则无法在处理转移函数的同时进行合法性验证，所以我添加了`vector<vector<string>> delta_read`这个二维`string`数组结构用来暂存文件中转移函数的相关内容；数组的每一行存储一个转移函数。

另外，图灵机在动态执行的过程中需要确定当前自己的各个`head`的位置、纸带上的内容、执行的步数以及目前所处的状态，即要知道当前自己的`id`，因此构造如下结构：

```
struct id
{
    int step;
    int* index;//record the start of each tape
    string* tape;
    int* head;//record offset of tape start
    string state;
};
```

这里的`index`用于记录目前每个纸带的`head`相对于开始的0号位置的偏移量，在之后的`run`中会用到，这里暂时不作解释。

最后，定义一个`bool`变量用来指示`tm`文件中的图灵机是否合乎语法，主程序可通过查看该值决定返回正确值还是错误值。由此，程序中图灵机的结构定义为如下的类：

```
class TM
{
private:
    vector<string> Q; //states
    vector<char> S; //input symbols
    vector<char> G; //tape symbols
```

```
    string q_0; //start symbol
    char B; //blank symbol
    vector<string> F; //final states
    int n; //number of tapes
    vector<vector<string>> delta_read;
    delta trans; //trans functions
    bool normal; //tell if setup success
    id current_id; //record current id
};
```

- ## 命令行输入的处理：

在`turing_parser.cpp`中实现了函数`command_read`进行命令行输入的处理，函数声明为：

```
int command_read(int c,char** v,string& file_name,string& input)
```

返回值指明了命令行输入的处理结果：−1表示命令行内容不合法，2表示输入内容为`turing -h|--help`，3表示输入内容为`turing <tm> <input>`，4表示输入内容为`turing -v|--verbose <tm> <input>`；参数`c`是输入命令行内容的长度(以空格分割)，`v`是命令行的内容，`file_name`与`input`用于存储处理命令行内容后得到的文件名与输入字符串，将用于之后的图灵机运行。

函数的实现思路为：根据输入命令行内容的长度进行分类并进行合法性判断。如果`c`为2，则合法的输入只可能是`turing -h|--help`，因此对`v[1]`的内容进行判断，只有`-h|--help`是合法的；如果`c`为3，则合法的输入只可能是`turing <tm> <input>`，因此视`v[1]`、`v[2]`为图灵机程序文件名与输入字符串，将他们的值赋给`file_name`与`input`并返回3；如果`c`为4，则合法的输入只可能是`turing -v|--verbose <tm> <input>`，因此首先要检验`v[1]`是否为`-v|--verbose`，如果满足则视`v[2]`、`v[3]`为图灵机程序文件名与输入字符串，将他们的值赋给`file_name`与`input`并返回4，否则返回−1；如果`c`为其他值则不可能是合法命令行输入，因此返回−1.

- ## 解析器的实现：

在`turing_parser.cpp`中实现了`TM`类的成员函数`setup`函数来进行解析器的工作，该函数的声明为：

```
void setup(string file_name,bool whether_v);
//setup a turing machine
```

其中`file_name`是图灵机程序名，`whether_v`是经过命令行解析后判断是否有`-v|--verbose`命令的变量，为真说明是在`verbose`模式下运行的。`setup`的功能就是每次从图灵机程序中读取一行，去掉前面无用的空格、注释的内容(`;`之后的内容)以及去除注释后末尾无用的空格。经过这些处理后如果内容不为空，说明需要对该行进行解析，则`setup`函数就会调用`process`函数对该行的内容进行解析。在`setup`中，首先要打开图灵机程序文件，若打开失败则报错；每处理一行内容后要检查一下`TM`类中`normal`的值，如果为`false`说明图灵机程序有错误。在我的实现中，如果是在`verbose`模式下运行，则我会报出发现的所有错误，因此在`verbose`模式下即使`normal`为`false`，对文件的处理也不会终止；但如果是普通模式，不需要保错，则发现一个错误就可以终止循环处理结束了。最后，在处理完图灵机程序的内容后，调用函数`check_grammar`对存储的信息进行合法性检测(例如检测初始状态是否在状态集中、输入字符集是否是纸带符号集的子集这些)。

可以看到`setup`函数中真正对图灵机程序内容处理的是`process`函数，这个函数需要根据输入字符串的内容确定它对应的是图灵机的哪一部分并进行相应的处理来将输入字符串的信息存储到`TM`对象中。考虑到图灵机程序中除去注释与空行外的内容只有转移函数与以`#`开头的信息，所以`process`函数的实现方法为：如果输入字符串以`#`开头，则进入`switch`语句来对图灵机的状态集、字符集等进行处理；否则就认为是转移函数，并调用`process_delta`进行处理。因此`process`函数的结构为：

```
void TM::process(string line,bool whether_v)
{
    if(line[0] == '#')
    {
        if(line.length() < 6)
        {
            error_process(whether_v,line,15);
            //进行报错
            return;
        }
        switch(line[1])
        {
            case 'Q':
```

```
            ……
            default:
                error_process(whether_v,line,7);
        }
    }
    else
    {
        //process delta
        process_delta(line,whether_v);
    }

}
```

这里考虑到对于以`#`开头合法的内容，最短的是`#B = _`，其长度为6，所以长度小于6的内容不可能合法，直接调用错误处理函数`error_process`(之后介绍)进行报错。接下来介绍针对图灵机不同部分的处理方法：

1、**状态集Q的处理**：`process`函数会在`switch`语句的`case 'Q'`分支下调用`process_Q`函数对状态集Q进行处理。因为状态集内容格式为`#Q = {…}`，所以长度至少为7；在调用之前如果发现字符串内容小于7就进行报错：

```
            case 'Q':
                if(line.length() < 7)
                {
                    error_process(whether_v,line,16);
                    return;
                }
                process_Q(line,whether_v);
                break;
```

`process_Q`函数的声明为：

```
void TM::process_Q(string line,bool whether_v);
```

首先验证`line`是否符合状态集`Q`的格式`#Q = {…}`：

```
if(line[2] != ' ' || line[3] != '=' ||
   line[4] != ' ' || line[5] != '{' ||
   line[line.length()-1] != '}')
        error_process(whether_v,line,8);
```

如果格式符合要求，则遍历`{}`中的内容，根据`','`划分不同的状态，每得到一个状态就加入到`TM`对象的`Q`中。在`process_Q`中还需要考虑以下可能的错误：状态标签中是否包含了非法字符、是否有连续的`','`或是`',}'`(即缺少了状态标签，例如`{1,,3}`或是`{1,2,}`)。这些错误处理以及正确状态的存储实现思路为：遍历`'{'`之后的每一个字符1，用一个`string`变量`s`存储当前得到的状态标签。如果遍历到合法字符，就将该字符添加到`s`的尾部；如果遍历到`','`或者遍历到`line`的末尾，说明接下来是新的标签或者结束，因此要把当前得到的标签`s`加入到状态集中，并将`s`清空；如果在将`s`加入到状态集时发现`s`为空，则说明出现了之前提到的缺少状态标签的错误，故进行报错；如果遍历到的是非法字符，则也进行报错。这里将`s`加入到状态集的方法是调用函数`get_states(s)`，其实现为：

```
void TM::get_states(string state)
{
    Q.push_back(state);
}
```

2、**输入符号集S与纸带符号集G的处理：**`S`的处理思路与`Q`基本一致，即先检查`line`的长度是否不小于7，然后判断`line`的内容是否符合格式；接着遍历`line`的字符，用初始为空格的`char`变量`s`进行记录，由此将合法的字符加入到输入符号集中，或是在出现`',,'`或`',}'`时进行报错。与`Q`不同的是`S`是输入符号集，不可以有字符串出现，所以如果遍历到合法字符，要把它赋值给`s`，但是`s`不为空格时，说明出现了字符串(因为遇到`','`就会将`s`清空为空格，处理到合法字符但`s`不是空格就说明了这两个合法字符之间没有`','`)，进行报错：

```
        else if(line[i] >= 32 && line[i] <= 126 &&
                line[i] != ' ' && line[i] != ';' &&
                line[i] != '*' && line[i] != '_')
            //是合法字符
            {
                if(s != ' ') //string but char appears in
S
                    {
                        error_process(whether_v,line,11);
                        return;
                    }
                else s = line[i];
            }
```

将`s`加入到输入符号集的方法是`get_input_symbols(s)`：

```
void TM::get_input_symbols(char s)
{
    S.push_back(s);
}
```

`G`的处理与`S`完全一致，只是将`s`加入到纸带符号集的方法是`get_tape_symbols(s)`，不再赘述。

3、**初始状态q0的处理**：q0只是一个状态，没有`{}`什么的需要处理，所以没有额外实现函数进行处理，直接在`switch`语句中进行实现：根据格式要求，首先长度要大于7，内容是`#q0 = …`，对这些进行验证，如果没问题的话后面的`...`内容存储到字符串中并调用`get_start_state`将它赋值给到`TM`对象的`q_0`：

```
void TM::get_start_state(string state)
{
    q_0 = state;
}
```

这里不进行初始状态的标签构成是否合法的检查，是因为后续要验证`q0`是否在状态集`Q`中，如果标签有非法内容自然不在`Q`中，从而会报错。

4、**空格符号B的处理**：考虑到空格符号只是一个符号，所以实际上图灵机程序中关于B的说明内容长度应该是固定的6。因此处理方法为：

```
case 'B':
    if(line.length() != 6)
    {
        //非法，报错
        error_process(whether_v,line,17);
        return;
    }
    if(line[2] != ' ' || line[3] != '=' || line[4] != ' ')
        error_process(whether_v,line,4);
    else
        //符合格式，最后一个字符就是空格符号
        get_blank_symbol(line[5]);
        //将它赋值给TM对象的q_0变量
    break;
```

5、**终结状态集F的处理**：与状态集Q的处理思路、方法完全一致，不再赘述。

6、**纸带数N的处理**：首先检查格式是否正确：

```
case 'N':
if(line[2] != ' ' || line[3] != '=' || line[4] != ' ')
    error_process(whether_v,line,5);
```

如果正确，则`line[5]`之后的内容就是纸带数`n`；因此对其进行循环处理，将`n`从字符串的形式转化为`int`；如果在处理过程中遍历到了不是'0-9'的字符，则说明`n`中出现了非数字字符，故报错：

```
else
{
    int num = 0;bool f = true;
    for(int i=5; i<line.length();++i)
    {
        if(line[i] >= '0' && line[i] <= '9')
        {
            num *= 10;
            num += line[i] - '0';
        }
        else
        {
            f = false;
            error_process(whether_v,line,6);
            break;
        }
    }
    if(f) get_num(num);
}
break;
```

7、**转移函数delta的处理**：如果输入内容不是以`#`开头，则将其视为转移函数，并调用函数`process_delta(line,whether_v)`进行处理。该函数的功能仅仅是将`line`的内容按照空格划分后存储到`TM`类的二维`string`数组`delta_read`中：

```
void TM::process_delta(string line,bool whether_v)
{
    vector<string> d;
    stringstream stin(line);
```

```
    string temp;
    int i = 0;
    while(stin >> temp)
    {
        d.push_back(temp);
    }
    //按照空格进行拆分，因此d[0]是当前状态，d[1]是当前head所指内容
    //d[2]是将要写的字符，d[3]是head的移动方式
    //d[5]是下一个状态
    if(d.size() != 5)
    //转移函数的内容数量与格式不符，报错
    {
        error_process(whether_v,line,2);
        return;
    }
    delta_read.push_back(d);
    //否则加入到delta_read中
}
```

经过以上的处理之后，图灵机的所有信息就被存储到`TM`类的对象中了。如果上述处理都没有出现问题，则说明图灵机程序中的内容没有格式问题，因此接下来需要对存储信息的合法性进行验证，具体包括：检查输入字符集是否是纸带字符集的子集，初始状态是否在状态集中，空格符是否在纸带字符集中、终结状态集是否是状态集的子集以及转移函数的合法性。转移函数的合法性又包括：检查每一个转移函数的当前状态在不在状态集中、下一个状态在不在状态集中、当前head所指的内容以及要写到纸带上的内容的长度是否与纸带数一致，head所指的字符以及要写到纸带上的内容是否都在纸带符号集中、指明head移动方式的字符串的长度是否等于纸带数、是否都是'r,l,*'中的字符。

在代码中实现了函数`check_grammar`与`check_one_delta`来完成这部分功能。前者用以验证非转移函数部分内容的合法性，并调用后者处理所有暂存在`delta_read`中的转移函数信息，如果合法就将这些信息存储到`TM`对象的`trans`中：

```
//check delta
for(int i = 0;i < delta_read.size();++i)
{
    bool res = check_one_delta(delta_read[i],whether_v);
    if(res) //delta is legal
    {
        trans_way t;
        t.rewrite_symbols = delta_read[i][2];
```

```
            t.move = delta_read[i][3];
            t.new_state = delta_read[i][4];
            pair<string,string> t1 =
                make_pair(delta_read[i][0],delta_read[i][1]);
            trans.deltas[t1] = t;//add a delta
        }
        else if(!whether_v)
        //普通模式下运行，发现错误就报错并结束
            break;
    }
```

两个函数要检验的都是双方的包含关系，利用遍历即可进行验证，这里不进行过多描述。

最后介绍一下函数`error_process`，它用于在各种错误出现时进行相应的处理，其函数声明为：

```
void TM::error_process(bool whether_v,string line,int
kind);
```

参数`whether_v`表明了是否在`verbose`模式下运行，`line`是出现错误的图灵机程序内容，`kind`说明了错误的类型。如果是普通模式，该函数会直接输出"syntax error"并结束，否则函数会根据`kind`进行相应的报错。这里我考虑的错误类型有：格式不符、转移函数字符串的数目不为5、数字`n`中出现非数字字符、集合中出现',,'或',}'的情况、字符集合中出现字符串、图灵机程序得到的字符串长度不满足格式要求、出现非法字符、转移函数内容与纸带数不一致、输入字符集不是纸带字符集的子集、初始状态不在状态集中等一系列错误。

经过上述处理，如果图灵机程序存在问题，就会进行报错，否则就会在`TM`类的对象中存储与该程序对应的一个**多带图灵机模拟器**。

- ## 模拟器的实现：

经过解析器的处理后，`TM`类的对象中已经存储了图灵机程序中的各种信息，在模拟器中就是要根据这些信息对输入字符串进行处理。在`turing_simulator.cpp`中实现了函数`run`完成模拟器的功能，处理流程为：首先检查输入的字符串是否在输入符号集中，如果其中有不属于输入符号集的字符则进行报错。接着调用函数`get_start_id`给图灵机进行初始化，准备运行。图灵机的运行利用`while(1)`实现：在循环体中，首先判断当前状态是否属于终结状态集，是则退出循环。否则调用函数`search_trans`找到当前`id`下可以进行的转移函数，如果当

前的`id`没有可以执行的转移函数则退出循环，如果有则图灵机根据该转移函数进行状态转移。

进行一次状态转移要依次执行以下步骤：将图灵机的`current_id.state`更改为状态转移函数的新状态；将每一个纸带的`head`所指的字符改写为转移函数的新符号；根据转移函数指明的`head`移动方向更改`current_id.head`；前两步的实现非常简单，直接更改即可。第三步的实现较为复杂，因为`head`不总是在当前的纸带内容的范围内进行移动，即指向纸带内容开头的`head`可以左移，指向纸带内容末尾的`head`也可以右移。此外，在`verbose`模式下运行时有如下要求；

注4：一般情况下，两端非空格符号外侧的空格符号和对应索引都不需要打印，只需打印纸带上最左非空格符号到最右非空格符号之间的符号及对应索引；

注5：**当读写头指向的位置为非空格符号外侧的空格符号时，需要打印必要的空格符号及对应索引，空格符号以`'_'`表示。**

所以实际上要打印的内容应该为[min(head,第一个非空格符的位置),max(head,最后一个非空格符的位置)]。对于这些要求采用了以下思路进行实现：

在之前介绍结构体`id`时，其内容为：

```
struct id
{
    int step;
    int* index;//record the start of each tape
    string* tape;
    int* head;//record offset of tape start
    string state;
};
```

其中`index`记录的是每一条纸带上要打印的内容是从哪里开始的，这里的位置是相对于初始的0号位置而言的；例如第一步`head`向左移动则对应的`index`就是-1，第一步向右移动对应的`index`就是1，不动则`index`为0。`head`用于记录对应的纸带的读写头相对于该纸带的`index`的偏移量。在此基础上进行实现：

首先更改每个纸带`head`的位置，即更新`current_id.head`的值；如果是左移且`head`的值小于0，说明`head`移动到了纸带内容左边界外的位置，因此将对应的`index`减1，同时给纸带的内容开头加一个空格符，并将`head`的值加1，恢复为0；这就表示`head`指向了非空格符号外

侧的空格符号，`index`减1是因为纸带要打印的内容左端多了一个空格符，所以相对于0位置开始打印的位置要减一；`head`恢复为0则是因为此时`head`指向了最左端的空格符，相对于`index`偏移为0。如果是右移且超出了当前纸带集上内容的长度，则说明`head`移动到了右边界以外，要在纸带的内容尾部添加空格符；由于纸带内容开始的位置不变，故`index`、`head`均保持不变；

```
if(current_trans.move[i] == 'l')
{
    current_id.head[i]--;
    //第i个纸带的head左移
    if(current_id.head[i] < 0)
    {
        current_id.index[i]--;
        current_id.tape[i] = B + current_id.tape[i];
        current_id.head[i]++;
    }
}
else if (current_trans.move[i] == 'r')
{
    current_id.head[i]++;
    if(current_id.head[i] >= current_id.tape[i].length())
        current_id.tape[i] += B;
}
```

这样处理后，就完成了一次`head`移动的任务。但如果只做这些的话，可能会对后续的打印产生影响：例如目前`head`指向了最左边的空格符，下一次向右移动且没有在该位置写上非空字符，则最左边的空格符在下一次就没必要打印了；但如果只进行上述操作，下一次打印还是会打印多余的最左边的空格符。因此还需要进行以下处理来删除多余的空格符：

遍历找到要打印的部分，将对应纸带原有的内容改写为需要打印的内容：

```
int left = 0,right = current_id.tape[i].length()-1;
while(current_id.tape[i][left] == B) left++;
if(left > current_id.head[i]) left = current_id.head[i];
while(current_id.tape[i][right] == B) right--;
if(right < current_id.head[i]) right = current_id.head[i];
//找到要打印的区域:
//[min(head,第一个非空格符的位置),max(head,最后一个非空格符的位置)]
string temp;
for(int l = left;l <= right;l++)
    temp += current_id.tape[i][l];

current_id.tape[i].clear();
current_id.tape[i] += temp;
```

根据改写后的内容对`index`进行调整：如果`left`的值大于0，说明纸带内容开始部分有`left`个多余的空格符。因此，将对应纸带的`index`加上`left`，`head`减去`left`(因为`head`记录的是相对于`index`的偏移量)即可。

执行了以上步骤就模拟了图灵机的一次转移。如果是在`verbose`模式下运行，则调用`show_id`函数进行一次打印。这样，退出循环就说明图灵机执行结束，故将`tape1`的纸带内容（首尾分别为纸带上最左和最右非空格符号)进行打印。由此完成了模拟器的实现。

下面介绍`run`中调用到的功能函数：

```
int TM::check_input(string input);
//return 1st illegal char if exits, else return -1
```

用于检查输入字符串中是否包含不在输入符号集的字符。

```
bool TM::search_trans(trans_way &x);
```

用于查找当前`id`下可以执行的状态转移函数，如果查找到返回$true$，并将找到的转移函数赋值给x；否则返回$false$;

```
void TM::get_start_id(string input,bool whether_v);
```

用于初始化图灵机，并打印图灵机的初始状态：执行步数为0，状态为初始状态，`tape1`是输入字符串，其余`tape`为空，每个纸带的`head`与`index`均初始化为0;

```
void TM::show_id();
```

用于打印当前图灵机的`id`；满足注6、注7的要求：

注6：`Index` 中相邻两项之间以一个空格为间隔，`Tape` 上的符号与 `Index` 中对应的索引靠左对齐。

注7：若需要向 `0` 索引左边的纸带单元中读写字符，请在 `Index` 中按照 `"... 3 2 1 0 1 2 3 ..."` 的格式(即省略 `-3 -2 -1` 等负索引的负号)对索引进行描述。

实现思路为：在`run`中，调用`show_id`之前已经将多余的空格符处理掉了，所以这里只要将纸带上的内容与对应的位置编号打印出来就好了。首先确定`tape`内容的长度max，并利用数组`int *blanks = new int[max];`来存储在打印每一个纸带的字符之前应该先打印多少个空格以满足左对齐的要求。`blanks`的取值可以在打印索引时确定：

```
blanks[0] = 0;//打印首个字符前不必额外打印空格
cout << "Index" << i << " : ";
int j = 0;int t = 0;
for(j=current_id.index[i];
    j < max + current_id.index[i] -1;++j)
//current_id.index[i]记录了纸带内容的开始位置相对于0索引的偏移量
//因此要打印的索引范围是
//[current_id.index[i],current_id.index[i]+max-1]
{
    t = (j < 0) ? (-1*j) : j;
    //根据注7，索引为负数时省略负号
    cout << t << ' ';
    int n = le(t);
    //le是计算数字t的长度的函数
    blanks[j - current_id.index[i] + 1] = n;
}
t = (j < 0) ? (-1*j) : j;
cout << t << endl;
```

`blanks`的取值方式为`blanks[j - current_id.index[i] + 1] = n;`的理由是：在打印第j个字符时，因为要与对应索引靠左对齐且索引之间只有一个空格，所以只要在打印了第$j-1$个字符(长度为1)后再打印第$j-1$个索引长度个空格即可。例如当前打印索引为10，索引10对应字符为'a'，索引11对应字符为'b'，因为打印'a'之前输出位置是与'10'左对齐的，打印'a'后相当于输出位置右移一位，在打印'b'之前

打印'10'的长度，即2个空格，就相当于输出位置右移了三位，恰好与'11'左对齐。

这样，在打印纸带的第x个字符时，只要预先打印`blanks[x]`个空格再打印第x个字符就可以实现与索引靠左对齐了。对于读写头的打印，因为第i个纸带的读写头的位置是`current_id.head[i]`，即与第`current_id.head[i]`个字符对齐，所以利用相同的方式打印即可，只是要把`head`之前要打印的字符改为空格，`head`处的字符打印为'^';

模拟器实现时用到的一个小$trick$是如果纸带内容或输入字符串内容为空。则将内容赋值为一个空格符来表示其内容为空。

由此，实现了模拟器。

- ### main函数：

`main`函数通过构造`TM`类的对象实现要求的任务，其函数为：

```
int main(int argc, char* argv[])
```

`main`函数首先调用`command_read`判断输入的是哪种命令，如果返回值为3或4则需要运行图灵机程序，因此构造`TM`类的对象并执行`run`。若`command_read`返回值为-1则`main`返回错误码-1. `TM`类的构造函数为：

```
TM(string file_name,bool whether_v)
    {
        normal = true;
        setup(file_name,whether_v);
    }
```

在解析、模拟完成后，`main`会调用`get_state`获取`normal`的值，如果为$false$说明出现错误，执行`exit(1)`返回错误码；

- ### 多带图灵机程序的实现：

见`programs`下的`case1.tm`、`case2.tm`中的注释。

四、思考与心得

完成本次实验，使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。作为建议，可以在之后将实现下推自动机作为附加项目，通过找到一个可以被图灵机接收但无法被下推自动机接受的字符串来认识二者的关系。