计算理论初步课程实验报告

姓名：崔子寒 学号：161220026

实验概述

项目	内容
完成情况	• 使用C++编写了多纸带图灵机程序解析器，可以根据给定的.tm图灵机描述文件，生成一个与之对应的图灵机模拟器。并且可以读如输入文件input.txt的内容，将运行过程和运行结果输出至文件中。 • 实现了判定语言{0^k | k是斐波那契数}和{ww | w是01串}的多纸带图灵机，并编写了测试用例进行测试。 • 项目使用Makefile构建，使用git进行管理。
操作系统	Ubuntu 18.04.1 LTS 64bit
编译器	g++ (Ubuntu 5.5.0-12ubuntu1) 5.5.0 20171010
提交说明	按照实验说明给定的方式设定提交文件的结构，my-project文件夹下是完整的代码。有可能因为环境更改导致turing缺少执行权限，请重新make生成可执行文件。

编译运行说明

编译运行

项目使用**Makefile**进行构建。编译过程比较简单。执行 `make` 后，会生成可执行文件 `turing` ,通过运行 `./turing path/to/case` 进行测试。程序将读入**case**目录下的**test.tm**和**input.txt**，并在同一目录下生成**console.txt**和**result.txt**。

例子:

```
user: ~/my-project/$ make
g++ -c ./main.cpp ./Tape.cpp ./utils.cpp ./Transition.cpp ./TuringMachine.cpp
g++ -std=c++11 ./main.o ./Tape.o ./utils.o ./Transition.o ./TuringMachine.o -o turing
user: ~/my-project/$ ./turing ../case1
```

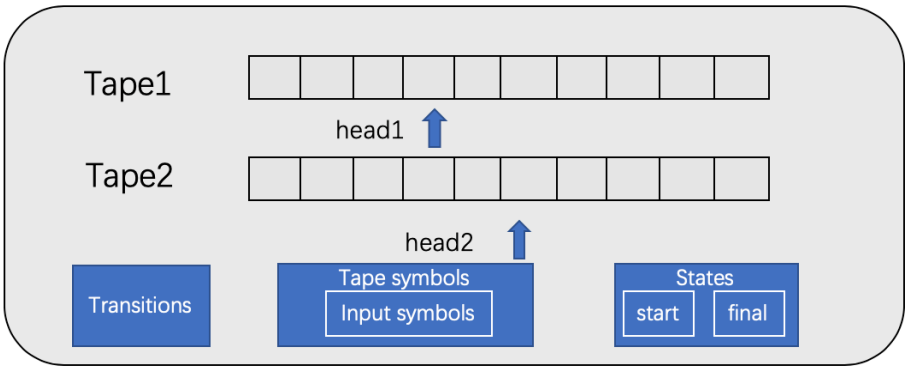
Make选项

Makefile也提供了其他的Makefile目标，可以通过执行 `make case1` 或 `make case2` 自动编译生成可执行文件，并进行测试。（要求：case1和case2应该位于Makefile的上一层目录）

设计思路

模块设计

一个多纸带的图灵机的结构如下图所示：



从面向对象的角度，一个图灵机对象由若干个**Tape**对象，若干个**Transition**对象，以及标定磁头位置的**head**变量，状态集**states**和符号集构成。其中，磁头、状态以及符号都可以用C++的已有类型表示，因此我们需要将**Tape**和**Transition**抽象成类。

在项目中，我设计了三个类，它们的基本信息描述如下：

类名	包含文件	功能描述
TuringMachine	TuringMachine.h TuringMachine.cpp	图灵机模拟器类，可以读入一个图灵机配置文件，生成图灵机模拟器。
Tape	Tape.h Tape.cpp	图灵机纸带类，模拟无限长的纸带，提供读写函数。
Transition	Transition.h Transition.cpp	转移函数类，接受状态和当前纸带符号，给出新状态，新符号，以及磁头移动方向。

这三个类的成员变量和成员函数的具体描述如下：

- Tape

成员变量	`vector <char> content`：记录初始状态下，磁头右侧的内容，下标0对应纸带的第0块，依次递增
	`vector <char> leftContent`：记录初始状态下，磁头左侧的内容，下标0对应纸带上的-1，依次递减
	`char blank`：纸带上的空白符号定义
	`int index`：纸带的编号
成员函数	`void init(string input)`：初始化纸带内容，将input填入content中
	`char get(int index)`：读取纸带第index块的内容，如果index超出当前已记录的内容范围，返回blank
	`void set(int index, char symbol)`：设置纸带第index块的内容
	`void setBlank(char ch)`：设置代表空白的字符
	`void setIndex(int index)`：设置纸带的编号
	`string result()`：返回纸带第一个非空符号到最后一个非空符号之间的内容

- Transition

成员变量	`string state`：输入状态
	`string tapeSymbols`：当前纸带符号
	`string newSymbols`：改写后的符号
	`string directions`：磁头移动方向
	`string newState`：转移后的状态
成员函数	`Transition(string description)`：构造函数，接受.tm文件中转移函数的一个描述，设置转移函数变量，返回值等
	`bool match(string state, string tapeSymbols)`：接受当前状态和当前纸带符号，判断是否匹配
	`string getNewSymbols()`：返回改写后的符号
	`string getNewState()`：返回转移后的状态
	`string getDirections()`：返回磁头的移动方向

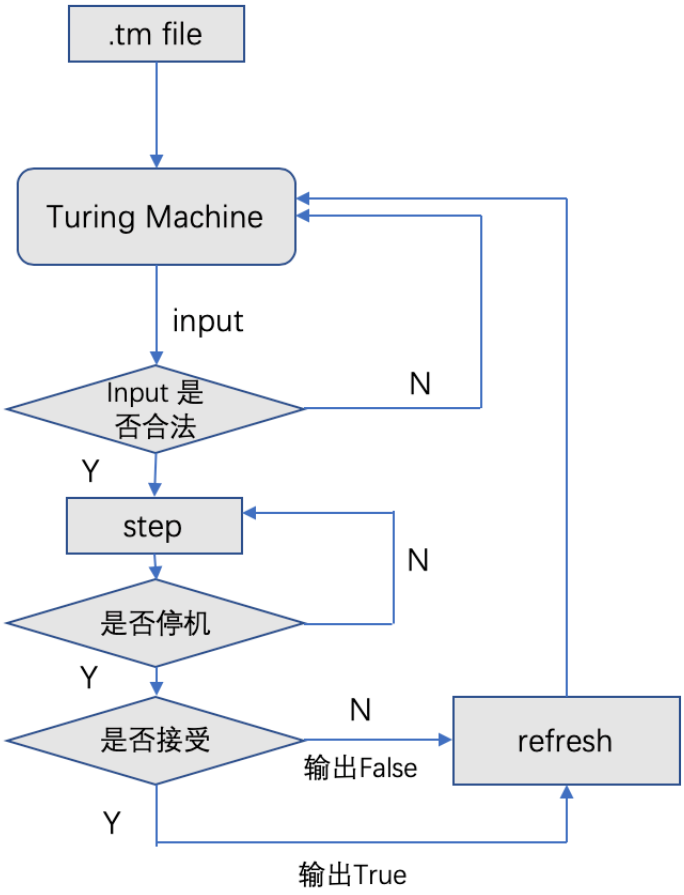
- TuringMachine

成员变量	`int numOfTapes`：纸带数目
	`int stepCount`：当前步数
	`char blank`：空白符号
	`string startState`：开始状态
	`string currentState`：当前状态
	`vector<string> states`：所有状态
	`vector<string> finalStates`：接受状态
	`vector<char> inputSymbols`：输入符号
	`vector<char> tapeSymbols`：纸带符号
	`vector<Transition> transitions`：转移函数
	`vector<Tape> tapes`：纸带
	`vector<int> heads`：磁头
成员函数	`TuringMachine(string tmConfig)`：构造函数，参数是.tm文件的路径。构造函数解析.tm文件，设置图灵机模拟器的各个成员变量。
	`bool input(string str)`：读取输入的字符串
	`bool step()`：图灵机执行一步，如果返回true，表明尚未停机，如果返回false，表明已经停机
	`string getID()`：返回图灵机的一个Instantaneous Description
	`string toString()`：返回图灵机的完整描述
	`void refresh()`：重置图灵机模拟器，准备接受下一个输入
	`bool accept()`：当前处于接受状态
	`string result()`：返回第一条纸带上的内容

流程设计

在读取了图灵机配置文件后，我们的图灵机模拟器就可以按照文件描述的那样运行了。我们首先调用input函数，读取输入并判定输入的合法性；然后不断的执行step，直到停机。此时，再调用accept函数，判断是否接受，如果是，则输出True，否则输出False。再处理完一条输入后，可以调用refresh，准备接受下一个输入。

整个处理流程可以用下面的流程图描述：



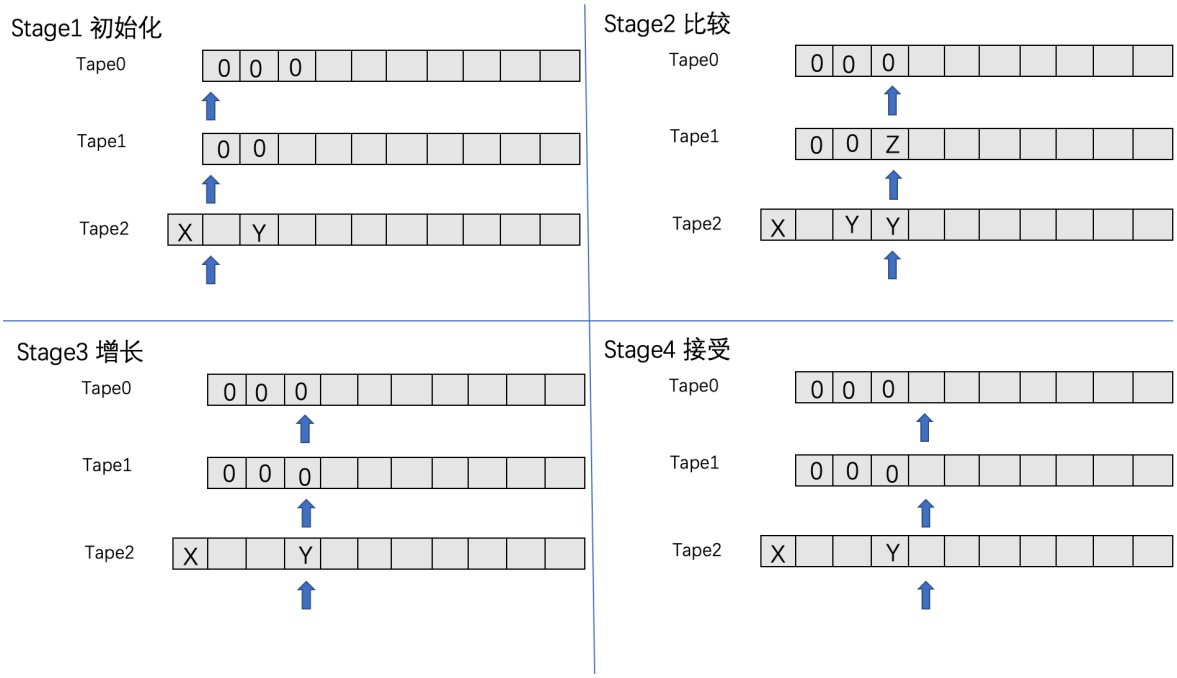
对应的代码在**main.cpp**中，关键代码如下：

```
for (int i = 0; i < inputLines.size(); i++)
    {
        inputLine = inputLines[i];
        if (tm.input(inputLine))
        {   //输入合法
            do
            {
                writeConsoleLog << tm.getID();
            } while (tm.step());
            if(tm.accept()) {
                writeResult << "True" << endl;
            }
            else {
                writeResult << "False" << endl;
            }
        }
        else
        {
           //输入非法
           ...
        }
        tm.refresh();
    }
```

算法设计

判定语言{0^k | k是一个斐波那契数}

- 思路：斐波那契数的计算公式为：f(n) = f(n-1) + f(n-2) (n > 2), f(1)=f(2)=1。我们可以仿照斐波那契数的计算过程进行判断。首先我们需要3条纸带，Tape0储存输入的串，Tape1的串的长度按照斐波那契数进行增长，每增长一轮，就和Tape0进行比较，如果与Tape0长度相同，则接受；如果长度比Tape0短，则继续增长；如果长度比Tape0长，说明Tape0上的串的长度并不是一个斐波那契数，拒绝。Tape2用做标记，记录上一次Tape1的长度，也就是这一次要增长的长度。
- 为了方便起见，我们首先判断输入串的长度是否为1，如果长度是1，直接接受，如果输入是空串，直接拒绝。当长度大于1时，我们在Tape1上写入长度为2的串，在Tape2上标记上一次Tape1的长度为1。然后开始“比较->增长”的不断重复，直到停机。
- 算法示例：下图描述了图灵机接受“000”的过程

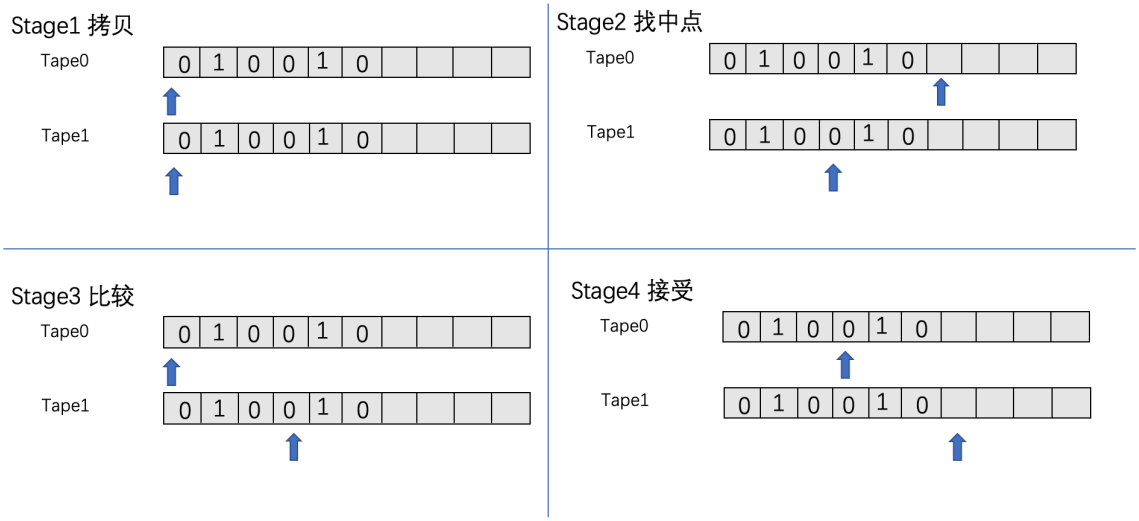


有关上图的一些解释：

1. 在初始化阶段，我们确定了输入串的长度大于1，所以Tape1中的串从“00”开始增长。Tape2中，我们在-1块上写入标记X，在第2块写入标记Y，规定X和Y之间的空格数就是Tape1中的串下次增长的长度。
2. 在比较阶段，三个head同步右移，直到发现了Tape1中的串比Tape0中的串要短。此时，我们将这个位置在Tape1中标记为Z，认为这是下次比较开始的位置，在Tape2中更新Y。
3. 增长阶段，我们把head2移到X标记处，然后head1和head2向右移动，开始增长，直到head2碰到第一个Y。然后head2清空这个Y，将head2移到下一个Y，将head1移到之前到Z，并将Z设置为0，然后开始下一次比较增长过程，如果比较时发现head0和head1同时指向了空，那么就接受。清空Tape0，并写入True。

判定语言{ww | w in {0,1}*}

- 思路：判定一个串是否由两个相同的串拼接而成，我们需要寻找这个串的中点，然后比较中点前和中点后的内容是否相同。如果搜索中点的过程中发现串的长度为奇数，那么直接拒绝。搜索中点可以采用类似于搜索链表中点的方法，用两个磁头，从头开始，一个每次走一步，一个每次走两步，当第二个走到尽头时，第一个就是中点。
- 设计的图灵机拥有两条纸带，首先将输入从第一条纸带拷贝到第二条纸带；然后head0和head1开始右移，head1每右移一次，head0就右移2次，当head0指向空时停止；将head0移到输入最左端，然后比较head0和head1右侧的内容，直到head1指向空，接受，如果内容不相等，拒绝。
- 算法示例，下图描述了接受“010010”的过程：



运行展示

这里以case1为例，展示图灵机模拟器的完整运行过程：

1. 首先，在case1下准备我们的输入文件input.txt:

input.txt

case1 > input.txt

```
1
2   0
3   00
4   000
5   0000
6   00000
7   000000
8   0000000
9   00000000
10  000000000
11  0000000000
12  00000000000
13  000000000000
14  0000000000000
```

2. 编译执行：

```
cui@VM-0-7-ubuntu:~/TheoryOfComputation2019/src$ make clean
rm -f turing
rm -f *.o
cui@VM-0-7-ubuntu:~/TheoryOfComputation2019/src$ make
g++ -c ./main.cpp ./Tape.cpp ./utils.cpp ./Transition.cpp ./TuringMachine.cpp
g++ -std=c++11 ./main.o ./Tape.o ./utils.o ./Transition.o ./TuringMachine.o -o turing
cui@VM-0-7-ubuntu:~/TheoryOfComputation2019/src$ ./turing ../case1
```

3. 查看结果(case1/result.txt)：

case1 > result.txt

```
1   False
2   True
3   True
4   True
5   False
6   True
7   False
8   False
9   True
10  False
11  False
12  False
13  False
14  True
```

4. 查看运行过程(case1/console.txt)：

```
1606  ------------------------------------------------------------
1607  Step    :   129
1608  Index0  :   0   1   2   3   4   5
1609  Tape0   :   0   0   0   0   0   _
1610  Head0   :                       ^
1611  Index1  :   0   1   2   3   4   5
1612  Tape1   :   0   0   0   0   0   _
1613  Head1   :                       ^
1614  Index2  :   -1  0   1   2   3   4   5
1615  Tape2   :   X   _   _   _   Y   _   _
1616  Head2   :                           ^
1617  State   :   cmp
1618  ------------------------------------------------------------
1619  Step    :   130
1620  Index0  :   0   1   2   3   4
1621  Tape0   :   0   0   0   0   0
1622  Head0   :                   ^
1623  Index1  :   0   1   2   3   4   5
1624  Tape1   :   0   0   0   0   0   _
1625  Head1   :                       ^
1626  Index2  :   -1  0   1   2   3   4   5
1627  Tape2   :   X   _   _   _   Y   _   _
1628  Head2   :                           ^
1629  State   :   accept
1630  ------------------------------------------------------------
```

总结感想

经过这次实验，加深了对与图灵机运行过程的理解，也体会到了多纸带图灵机拥有更强的编程能力。将现实问题转换成用图灵机解决的问题需要巧妙的构思和设计。

另外，这次实验也锻炼了编程能力和动手能力。