고급프로그래밍
3차 과제 보고서

학과	소프트웨어학부
학번	2020203090
이름	한용옥

24.06.14

결과 요약

구현한것	The Table, The Tape, The Machine
구현하지 못한것	없음

실행 사진

The Table

```
(*) List of commands
- add [curr_s] [read_s] [write_s] [move] [next_s]
- find [curr_s] [read_s]
- initialize [name] (name = palindrome, addition, parenthesis)
- load [path]
- clear
- print
- quit
- help
> load bin_dec.txt
0 * * * 1
1 _ _ r 1
1 * * r 1a
1a * * r 1a
1a _ _ l 2
1b _ _ r 1
1b * * r 1b
2 1 0 l 3
2 0 1 l 2
2 _ _ r 20
3 * * l 3
3 _ _ l 4
4 0 1 r 1b
4 1 2 r 1b
4 2 3 r 1b
4 3 4 r 1b
4 4 5 r 1b
4 5 6 r 1b
4 6 7 r 1b
4 7 8 r 1b
4 8 9 r 1b
4 9 0 l 4
4 _ 1 r 1b
20 _ _ l 21
20 * _ r 20
21 _ _ l 21
21 * * l 21a
21a * * l 21a
21a _ _ r halt
```

bin_dec.txt

파일 편집 보기

```
; Converts a number from binary to decimal.
; It does this by successively incrementing the decimal count
is reached.
; Input: A single number in binary.

0 * * * 1
1 _ _ r 1
1 * * r 1a
1a * * r 1a
1a _ _ l 2   ; found the end of the input
1b _ _ r 1   ; (used later) skip the output, go to end of input
1b * * r 1b
2 1 0 l 3    ; decrement input
2 0 1 l 2    ; decrement & carry
2 _ _ r 20   ; finished. clean up
3 * * l 3    ; find start of output
3 _ _ l 4    ; found end of input
4 0 1 r 1b   ; increment the output
4 1 2 r 1b
4 2 3 r 1b
4 3 4 r 1b
4 4 5 r 1b
4 5 6 r 1b
4 6 7 r 1b
```

줄 1, 열 1 848자 100%

주석과 공백이 많지만 잘 무시하고 로드하는 모습

```
> find 4 *
4 0 1 r 1b

> find 1 *
1 * * r 1a

> find 21 21
21 * * l 21a

> find 21a _
21a _ _ r halt
```

와일드문자를 포함한 검색을 잘 수행하는 모습

The Tape

```
<<< Turing::Tape Test Program >>>

(*) List of commands
- construct
- destroy
- randomize
- initialize [s]
- read [i]
- write [i] [c]
- push_back [c]
- push_front [c]
- extend_right [n]
- extend_left [n]
- print
- quit
- help
```

```
> construct


> initialize abcde
abcde

> push_back 0
abcde0

> push_front 1
1abcde0

> write 3 3
1ab3de0

> read 3
3

> extend_right 2
1ab3de0_
1ab3de0__

> extend_left 3
_1ab3de0__
__1ab3de0__
___1ab3de0__

> randomize
jXMu^GuKsoVP

> destroy

> print
Error: tape has not yet been constructed

>
```

모든 기능이 정상적으로 수행된다

The Machine

```
<<< Turing::Machine Test Program >>>

(*) List of commands
- load_table [path]
- init_table [name] (name=palindrome, addition, parenthesis)
- init_tape [data]
- start [initial state] [accept state] [reject state]
- run
- step
- quit
- help

> load_table primality.txt
0 * * l 1
1 * a r 2
2 _ b l 3
2 * * r 2
3 a a r 4
3 x x r 4
3 y y r 4
3 * * l 3
4 0 x r 5x
4 1 y r 5y
4 b b l 9
9 x 0 l 9
9 y 1 l 9
9 a a r 10
5x b b r 6x
5x * * r 5x
5y b b r 6y
606 * _ r 607
607 * n r 608
608 * o r 609
609 * t r 610
610 * _ r 611
611 * p r 612
612 * r r 613
613 * i r 614
614 * m r 615
615 * e r 616
616 * . * halt


> init_tape 101111
101111

> start 0 halt error
101111 [0/NORMAL]
^

> |
```

primality.txt를 잘 로드한 모습

```
_101111_is_prime___________ [511/NORMAL]
               ^
_101111_is_prime!__________ [halt/ACCEPT]
               ^
_101111_is_prime!__________ [halt/ACCEPT]
               ^

> |
```

47은 소수라고 잘 계산하는 모습이다

```
_110001_is_not_prim_________ [615/NORMAL]
                   ^
_110001_is_not_prime________ [616/NORMAL]
                   ^
_110001_is_not_prime._______ [halt/ACCEPT]
                   ^
_110001_is_not_prime._______ [halt/ACCEPT]
                   ^

> |
```

합성수인 49를 잘 계산하는 모습이다

The Table

데이터 관리

Table은 Transition의 집합으로 생각할 수 있고,
Table의 핵심기능인 findTransition의 반환형이 Transition*이므로
Table을 Transition*형을 담은 벡터를 이용해 관리하기로 했다.

```
private:
    vector<Transition*> table;
```

생성자

`Table::Table() {}` 하는일이 없어 비워두었다

규칙 추가하기

포인터로 규칙을 관리할 것이므로 동적할당을 이용한다
이미 구현되어있는 Transition의 생성자를 이용한다

```
void Table::addTransition(const std::string& curr_s, char read_s, char write_s,
                          Move move, const std::string& next_s)
{
    Transition* temp = new Transition{ curr_s, read_s, write_s, move, next_s };
    table.push_back(temp);
}
```

필요한 원소를 받고 원소를 동적할당으로 초기화해 table에 넣는다

텍스트로 초기화하기

텍스트의 조건은 아래와 같다
1. 텍스트는 ₩n으로 나뉘어져있다
2. 주석은 문장 시작, 규칙 끝에만 온다
3. 단어들은 공백으로 구분되어있다

3가지 조건을 생각할 때 문장을 분해하는 방법은 아래의 방법이 적당하다고 생각했다
1. util::split을 이용해 ₩n 기준으로 분리
2. 분리된 원소를 util::split을 이용해 공백 기준으로 분리
3. 첫 원소가 ;나 빈 문자면 건너뜀 (마지막 원소의 문장이 문장₩n이므로 문장 , ""으로 분리됨)
4. 아니면 처음 5개 원소만 뽑아 규칙에 추가

```
void Table::initialize(const std::string& rule_script) {
    this->clear();
    vector<string> temp1 = split(rule_script, '\n');
    for (const string& x : temp1) {
        vector<string> temp2 = split(x, ' ');
        if (temp2[0] == ";" || temp2[0] == "") continue;
        this->addTransition(temp2[0], (char)temp2[1][0], (char)temp2[2][0],
                            chartoMove((char)temp2[3][0]), temp2[4]);
    }
}
```

먼저 저장된 것을 없앤 다음, 인수로 받은 문장을 분해해 addTransition으로 계속 추가한다

파일로 초기화하기

파일로 문자열을 읽은 다음 구현된 initialize를 쓴다

```
bool Table::load(const std::string& path) {
    ifstream ist{ path };
    string rule_script, temp;
    while (getline(ist, temp)) {
        rule_script += temp;
        rule_script += "\n";
    }
    if (!ist.eof()) return false;

    this->initialize(rule_script);
    return true;
}
```

initialize의 인수는 ₩n까지 들어간 통 문자열이므로 파일을 줄 단위로 읽고 ₩n으로 결합했다
다 읽었을 때 ist가 eof가 아니면 못읽은 것이므로 false를 반환한다
읽은 문자열에 대해 initialize를 실행하고 true를 반환한다

출력하기

Transition에 구현된 print를 이용한다

```
void Table::print(ostream& os) const {
    for (Transition* const& x : table) {
        x->print(os);
        os << '\n';
    }
}
```

모든 원소에 대해 print를 실행한다

clear()

table의 원소는 동적할당된 원소들이므로 배열을 비우기 전에 delete를 해 메모리 누수를 막는다
그 다음에 배열을 비워준다.

```
void Table::clear() {
    for (Transition* const& x : table) delete x;
    table.clear();
}
```

소멸자

```
Table::~Table() {
    for (Transition* const& x : table) delete x;
}
```

동적할당된 원소들을 해제해준다

규칙 찾기

규칙을 찾는 조건은 다음과 같다
1. 상태와 문자가 동시에 맞으면 그 규칙 우선 반환
2. 문자 검색을 와일드 문자로 하면 상태가 맞는 것 중 맨 위 규칙 반환
3. 상태와 문자가 동시에 맞지 않아도 상태만 맞는 규칙중 문자가 와일드 문자인 규칙 반환

조건을 생각하여 아래와 같은 방법을 생각했다
1. 조건이 맞는 것 중에서 맨 위 규칙 반환이므로 벡터의 앞에서 시작
2. 상태와 문자가 동시에 맞으면 그 즉시 반환
3. 상태만 맞고 검색 문자가 와일드 문자거나 원소의 문자가 와일드 문자면
위의 2,3번 조건이므로 값 한 번만 저장
4. 배열 다 봤으면 저장된 값 반환

```
Transition* Table::findTransition(const std::string& curr_s, char read_s) {
    bool wild = true;
    Transition* result = nullptr;
    for (Transition* const& x : table) {
        if (x->getCurrState() == curr_s && x->getReadSymbol() == read_s) {
            return x;
        }
        if (wild && (x->getCurrState() == curr_s) &&
            ((read_s == WILDCARD_SYMBOL) || (x->getReadSymbol() == WILDCARD_SYMBOL))) {
            result = x;
            wild = false;
        }
    }
    return result;
}
```

3번에서 저장을 한번만 하기 위해 bool형 변수를 추가해
저장이 되면 변수를 false로 바꿔 저장을 못하게 했다
따라서 맨 위의 값이 나오고 상태와 문자가 동시에 맞는 게 뒤에 있더라도
정상적으로 반환 될 수 있다.

The Tape

Tape의 구조는 수업시간에 구현한 벡터와 구조가 거의 동일하다

멤버 변수

```
private:
    int sz;
    int space;
    char* elem;
```

sz는 원소의 개수, space는 할당받은 공간의 크기
elem은 할당받은 연속공간의 첫 번째 주소이다

기본 생성자

```
Tape::Tape()
    :sz{ 0 }, space{ 0 }, elem{nullptr}
{
}
```

테이프는 다른 함수에 의해 내용이 채워지므로
기본 생성자에서는 멤버변수만 초기값으로 설정한다

복사 생성자

```
Tape::Tape(const Tape& t)
    :sz{ t.sz }, space{ t.sz }, elem{new char[t.sz]}
{
    for (int i = 0; i < t.sz; i += 1) elem[i] = t.elem[i];
}
```

테이프를 받아 그 테이프의 멤버변수를 복사한뒤, 실제 내용도 복사한다

복사 대입 연산자

```
Tape& Tape::operator=(const Tape& t) {
    if (t.sz <= space) {
        for (int i = 0; i < t.sz; i += 1) elem[i] = t.elem[i];
        sz = t.sz;
        return *this;
    }
    char* temp = new char[t.sz];
    for (int i = 0; i < t.sz; i += 1) temp[i] = t.elem[i];
    delete[] elem;
    elem = temp;
    sz = t.sz;
    space = sz;
    return *this;
}
```

테이프를 받아 멤버변수를 복사해주고, 이미 있는건 해제한다.

이동 생성자

```
Tape::Tape(Tape&& t)
    :sz{ t.sz }, space{ t.space }, elem{t.elem}
{
    t.sz = 0;
    t.space = 0;
    t.elem = nullptr;
}
```

테이프를 받아 멤버변수를 복사하고, 인수의 포인터를 nullptr로 만든다

이동 대입 연산자

```
Tape& Tape::operator=(Tape&& t){
    delete[] elem;
    elem = t.elem;
    sz = t.sz;
    space = sz;
    t.sz = 0;
    t.space = 0;
    t.elem = nullptr;
    return *this;
}
```

이동 생성자와 내용이 같다

initialize()

초기화는 initialize()에 의해 이루어진다

```
void Tape::initialize(const std::string& s) {
    delete[] elem;
    sz = s.size();
    space = sz;
    elem = new char[space];
    for (int i = 0; i < sz; i += 1) elem[i] = s[i];
}
```

테이프가 이미 만들어져도 문자열대로 초기화 되어야하기 때문에
저장된 테이프를 해제 해주고, 문자열 안의 내용을 복사한다

clear()

```
void Tape::clear() {
    sz = 0;
}
```

sz만 0 으로 바꿔도 원소가 없는것처럼 기능한다

read()

```
bool Tape::read(int i, char& c) const {
    if (i < 0 || i >= sz) return false;
    c = elem[i];
    return true;
}
```

범위가 벗어나면 false, 범위 안이면 외부 문자에 읽은 값 대입 후, true 반환

write()

```
bool Tape::write(int i, char c) {
    if (i < 0 || i >= sz) return false;
    elem[i] = c;
    return true;
}
```

범위 벗어나면 false, 범위 안이면 쓰고 true 반환

print()

```
void Tape::print(std::ostream& os) const {
    for (int i = 0; i < sz; i += 1) os << elem[i];
}
```

원소를 순서대로 os에 출력

reserve()

```
void Tape::reserve(int newalloc) {
    if (space >= newalloc) return;
    char* temp = new char[newalloc];
    for (int i = 0; i < sz; i += 1) temp[i] = elem[i];
    delete[] elem;
    elem = temp;
    space = newalloc;
}
```

테이프의 메모리 공간을 인수만큼 늘린다
이때 이미 할당된 공간이 더 크면 종료하므로 늘리는 역할만 한다
인수개 만큼의 공간을 새로 만든 뒤, 복사하고 원래 공간을 해제 후 바꾼다

resize()

```
void Tape::resize(int newsize) {
    reserve(newsize);
    for (int i = sz; i < space; i += 1) elem[i] = EMPTY_SYMBOL;
    sz = space;
}
```

테이프의 길이를 인수로 맞춘다 reserve가 늘리는 용도로만 사용되어 resize또한 늘리는 용도로만 사용된다
이때 남는 공간은 EMPTY_SYMBOL로 채워넣는다

push_back()

```
void Tape::push_back(char c) {
    if (space == 0) {
        initialize(string(1,c));
        return;
    }
    else if (sz == space) reserve(2 * space);
    elem[sz] = c;
    sz += 1;
}
```

테이프가 빈 테이프라면 길이 1 내용은 c인 테이프를 생성한다
여유공간이 없다면 두 배로 늘린다
마지막 원소를 인수로 채운다

push_front()

```
void Tape::push_front(char c) {
    if (space == 0) {
        initialize(string(1, c));
        return;
    }
    else if (sz == space) reserve(2 * space);
    for (int i = sz; i > 0; i -= 1) elem[i] = elem[i - 1];
    elem[0] = c;
    sz += 1;
}
```

테이프가 빈 테이프라면 길이 1 내용은 c인 테이프를 생성한다
여유공간이 없다면 두 배로 늘린다
기존 원소를 한 칸씩 오른쪽으로 밀면 맨 앞이 빈다
맨 앞에 인수를 채워넣는다.

The Machine

멤버변수

```
enum class Mode { NONE, NORMAL, ACCEPT, REJECT, ERROR };
```

상태의 분류를 나타내는 Mode

```
Table table;
Tape tape;
```

규칙을 담을 table과 테이프를 담을 tape

```
Mode current_mode = Mode::NONE;
std::string current_state = "";
int current_pos = 0;
```

현재 상태와 위치

```
std::string accept_state = "";
std::string reject_state = "";
```

긍정, 부정 상태를 담을 변수

```
const Table& getTable() const { return table; }
const Tape& getTape() const { return tape; }
const std::string& getCurrentState() const { return current_state; }
int getCurrentPos() const { return current_pos; }
Mode getCurrentMode() const { return current_mode; }
```

각각의 멤버 변수를 반환하는 게터함수

머신은 생성자의 역할이 중요하지 않아 따로 정의하지 않았다

initTape()

```
void Machine::initTape(const std::string& initial_symbols) {
    tape.initialize(initial_symbols);
}
```

Tape 멤버함수이용, 문자열로 테이프를 초기화한다

initTable()

```
void Machine::initTable(const std::string& initial_symbols) {
    table.initialize(initial_symbols);
}
```

Table 멤버함수이용, 문자열로 테이블을 초기화한다

loadTable()

```
bool Machine::loadTable(const std::string& path) {
    return table.load(path);
}
```

Table 멤버함수이용, 파일로 테이블을 초기화한다

start()

```
void Machine::start(const std::string& start_state,
                    const std::string& accept_state,
                    const std::string& reject_state) {
    this->current_pos   = 0;
    this->current_state = start_state;
    this->accept_state  = accept_state;
    this->reject_state  = reject_state;
    this->current_mode  = Mode::NORMAL;
}
```

시작, 긍정종료, 부정종료, 시작 위치를 멤버변수에 저장한다

step()

튜링 머신의 움직임을 담당한다 튜링머신은 아래와 같이 행동한다
1. 테이프를 읽는다
2. 현재 상태와 읽은 문자에 맞는 규칙을 테이블에서 찾는다
3. 테이블 규칙대로 테이프를 쓰고, 헤드를 옮기고, 상태를 바꾼다
4. 상태가 정지면 멈춘다

1번은 테이프의 멤버함수를 이용한다
2번 역시 테이블의 멤버함수를 이용한다
따라서 3,4번만 따로 구현하면 된다

```
bool Machine::step() {
    if (current_state == accept_state || current_state == reject_state) return false;
```

현재 상태가 멈춤 상태 면 false를 반환한다

```
char read_s; tape.read(current_pos, read_s);
Transition* trans = table.findTransition(current_state, read_s);
```

테이프의 문자를 읽고, 테이블에서 현재 상태와 읽은 문자를 이용해 규칙을 찾아내 저장한다

```
if (trans->getWriteSymbol() != WILDCARD_SYMBOL) tape.write(current_pos, trans->getWriteSymbol());
```

쓸 문자가 와일드카드면 아무것도 안쓰므로, 와일드카드인지 검사 후 테이프에 문자를 쓴다

```
enum class Move
{
    LEFT = -1, NONE, RIGHT
};
```

```
current_pos += (int)trans->getMove();
if (current_pos < 0) {
    tape.push_front(EMPTY_SYMBOL);
    current_pos = 0;
}
else if (current_pos >= tape.size()) tape.push_back(EMPTY_SYMBOL);
```

튜링머신의 Move를 편하게 쓰기 위해 왼쪽은 –1 none은 0 오른쪽은 1로 바꿨다
따라서 다음 위치는 현재위치+테이블의move 다
이를 구현하였고 현재 위치가 기존 테이프의 범위를 벗어나면
테이프의 멤버함수 push_front, push_back을 이용해 EMPTY_SYMBOL를 채워 넣었다

```
current_state = trans->getNextState();
if (current_state == accept_state) current_mode = Mode::ACCEPT;
else if (current_state == reject_state) current_mode = Mode::REJECT;
```

현재 상태를 테이블에 적혀있는 다음 상태로 바꾸고
긍정멈춤인지 부정멈춤인지 판별해 저장한다

```
    return true;
}
```

한 주기 동안 할 일을 다했으므로 true를 반환한다.