Advanced Programming

Programming Assignment #3

Kang Hoon Lee

Kwangwoon University

Turing Machine

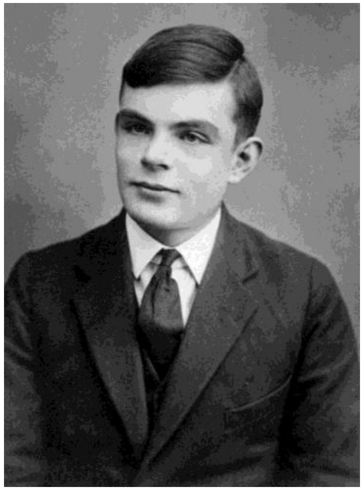
- **Invented by Alan Turing in his groundbreaking paper**
  - Turing, A. M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem". Proceedings of the London Mathematical Society.

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

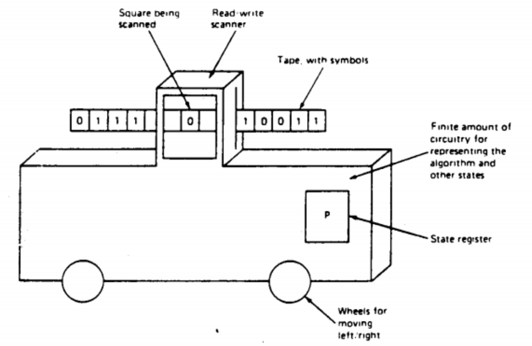
The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

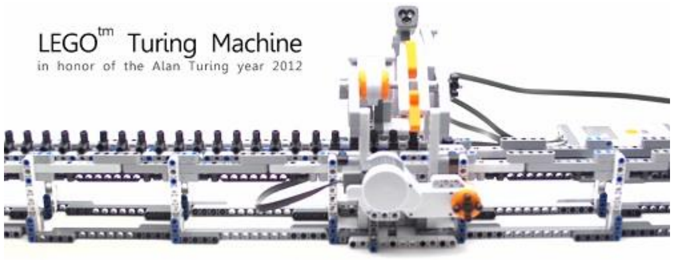


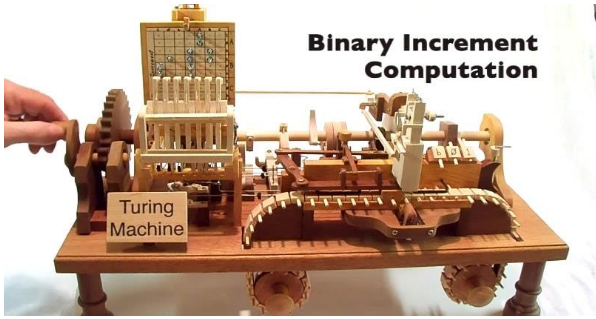
Turing Machine

- Physical Machine?

Figure B-2 An imaginary, physical Turing machine.







Turing Machine

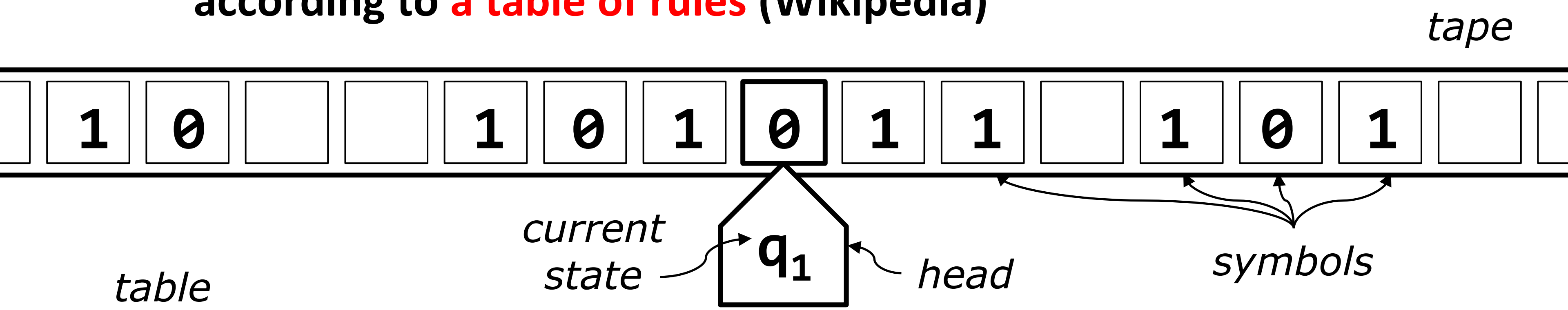
- **A mathematical model of computation that defines an abstract machine, which manipulates symbols on a strip of tape according to a table of rules (Wikipedia)**

$$M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$$

  - Q is a finite, non-empty set of states
  - Γ is a finite, non-empty set of tape alphabet symbols
  - $b \in \Gamma$ is the blank symbol
  - $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of input symbols
  - $q_0 \in Q$ is the initial state
  - $F \subseteq Q$ is the set of final states or accepting states
  - $\delta: (Q \setminus F) \times \Gamma \nrightarrow Q \times \Gamma \times \{L, R, N\}$ is called a transition function, where L is left shift, R is right shift, and N is no move

Turing Machine

- A mathematical model of computation that defines an abstract machine, which manipulates symbols on a strip of tape according to a table of rules (Wikipedia)



table

Current State	Tape Symbol	Write Symbol	Moving Direction	Next State
q_0	0	1	Right	q_1
q_0	1	0	Left	q_2
q_1	*	*	Left	q_0
q_2	*	*	Right	q_0

Turing Machine

- **Example: 3 state, 2 symbol busy beaver**

$$M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$$

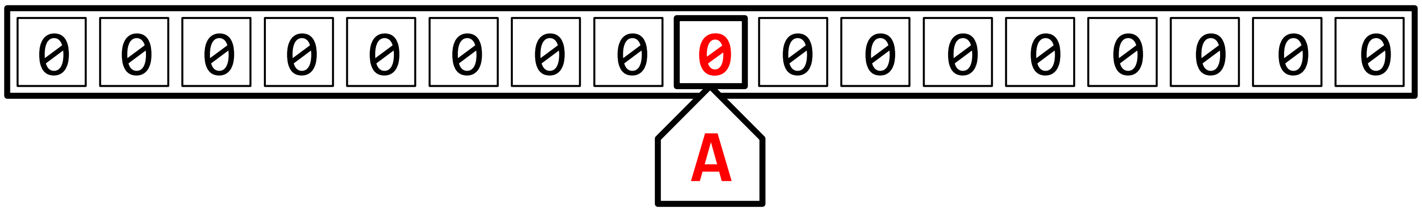
- 상태 $Q = \{A, B, C, HALT\}$
- 심볼 $\Gamma = \{0,1\}$
- 공백 $b = 0$
- 공백이아닌심볼 $\Sigma = \{1\}$
- 초기상태 $q_0 = A$
- 종료상태 $F = \{HALT\}$
- 상태 전이 함수 $\delta = \cdots$

상태×심볼→심볼×방향×상태

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Turing Machine

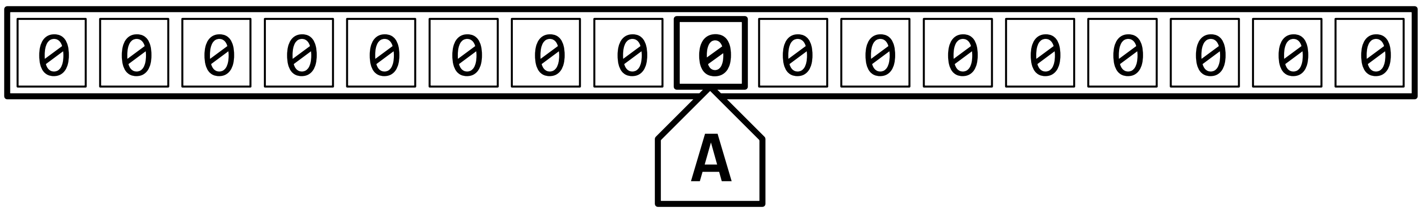
- Example: 3 state, 2 symbol busy beaver



Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Turing Machine

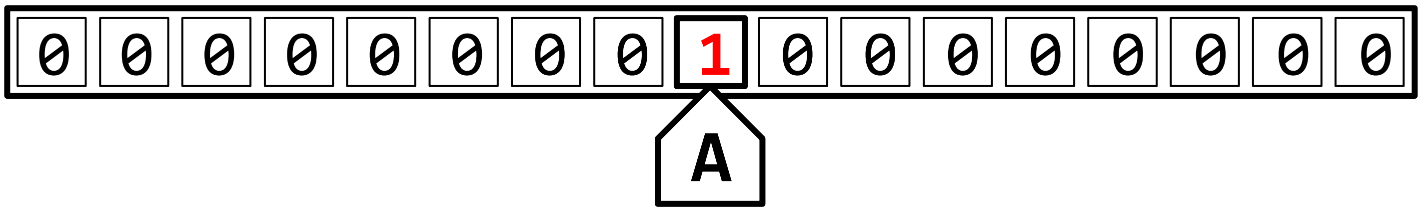
- Example: 3 state, 2 symbol busy beaver



Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Turing Machine

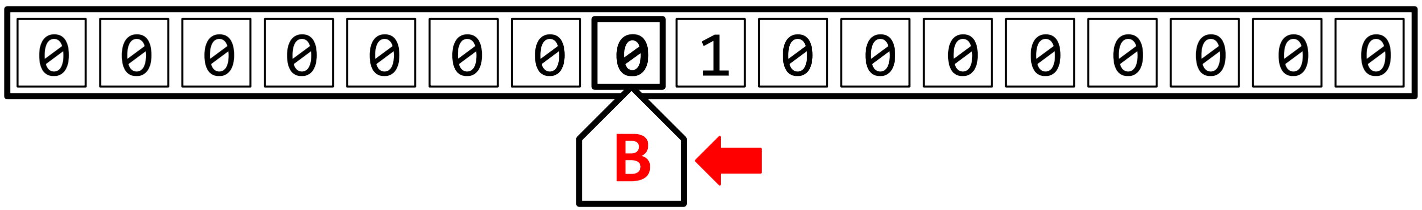
- Example: 3 state, 2 symbol busy beaver



Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Turing Machine

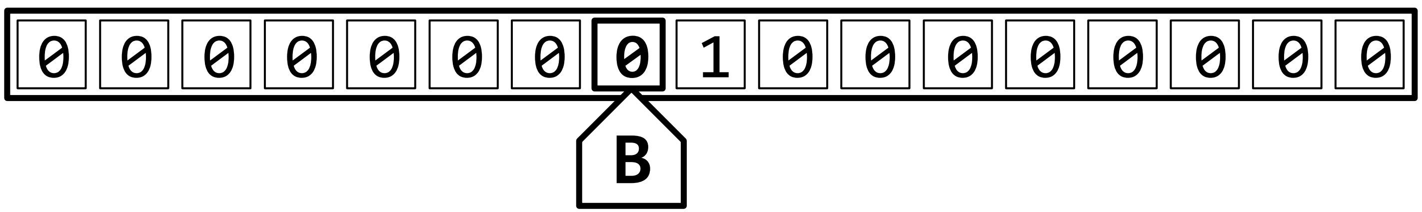
- Example: 3 state, 2 symbol busy beaver



Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Turing Machine

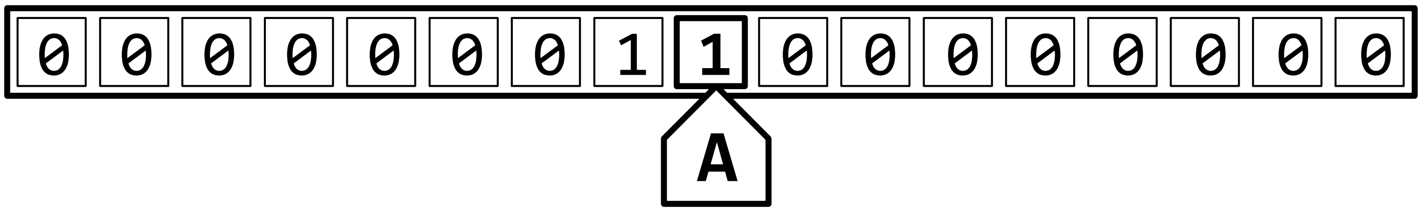
- Example: 3 state, 2 symbol busy beaver



Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Turing Machine

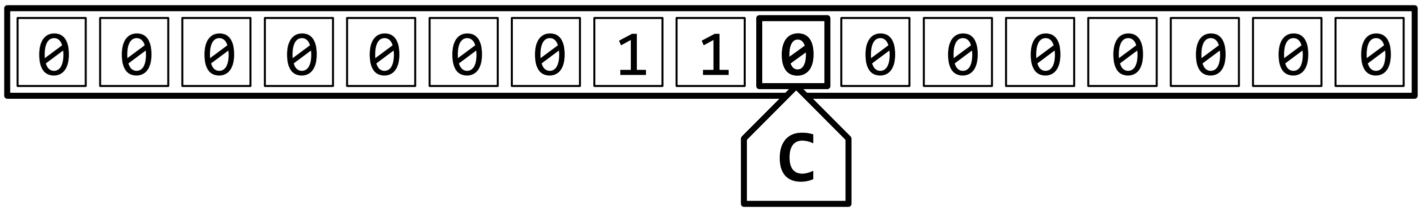
- Example: 3 state, 2 symbol busy beaver



Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Turing Machine

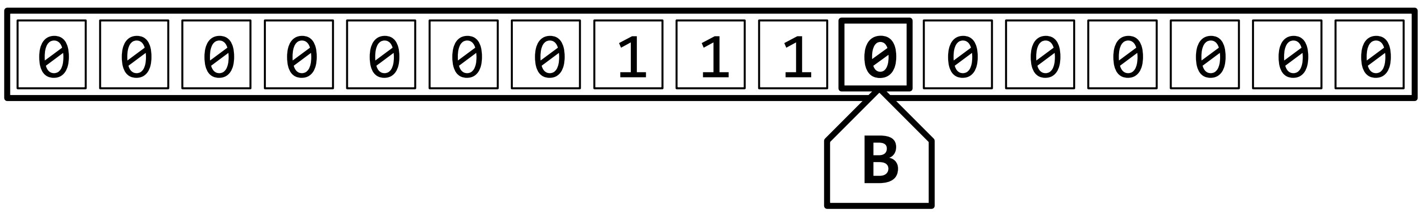
- Example: 3 state, 2 symbol busy beaver



Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Turing Machine

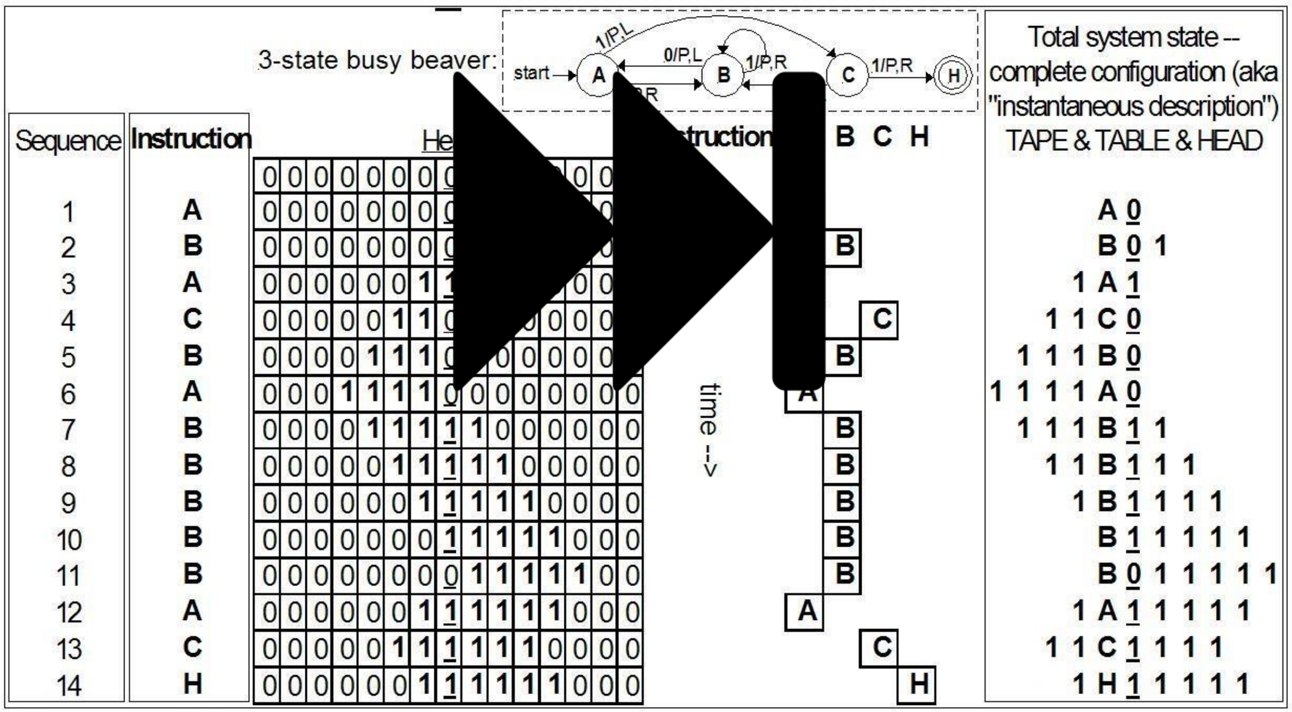
- Example: 3 state, 2 symbol busy beaver



Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Turing Machine

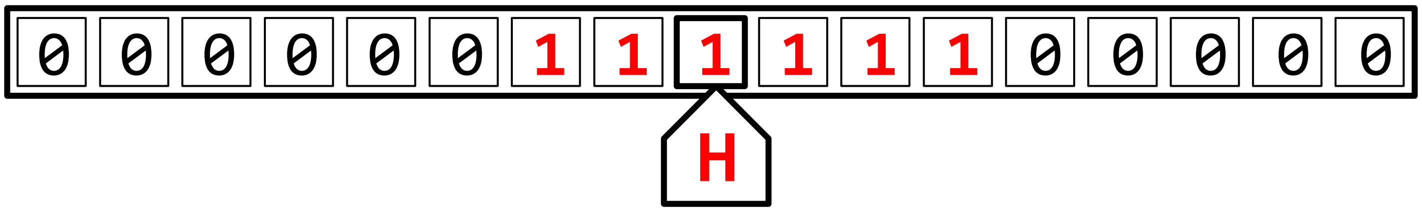
- Example: 3 state, 2 symbol busy beaver



Progress of the computation (state-trajectory) of a 3-state busy beaver

Turing Machine

- Example: 3 state, 2 symbol busy beaver



Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

https://www.youtube.com/watch?v=28pnk2JIBSE

Turing Machine Simulator

- **Write a C++ program to simulate the behavior of Turing machine**

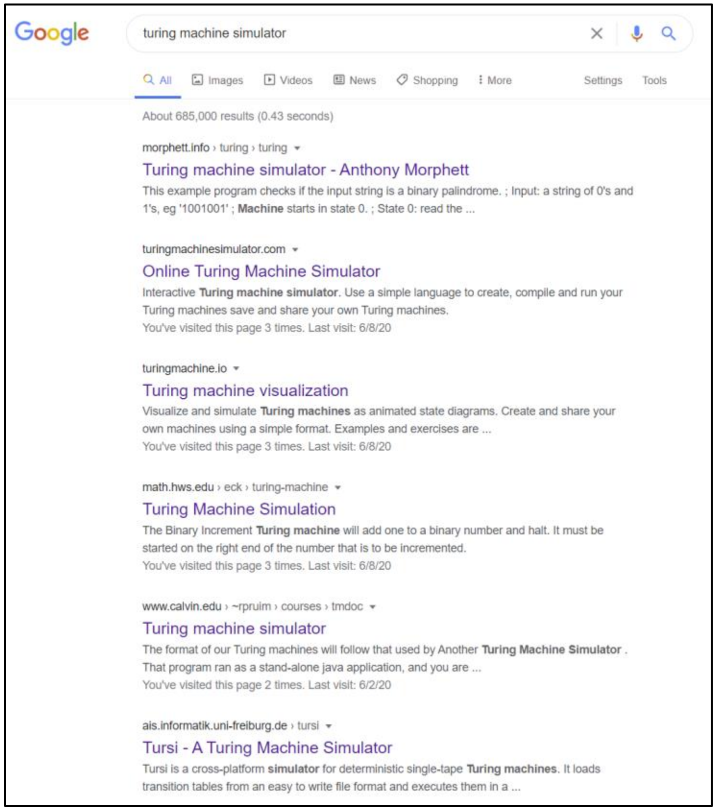
- **Input**
  - A set of transition rules (table)
  - Initial symbols on the tape
  - Initial position of the head
  - Initial state
  - Halting states (accept, reject)

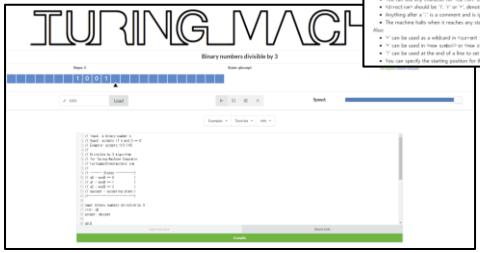
- **Output** (for each step)
  - Current symbols on the tape
  - Current position of the head
  - Current state

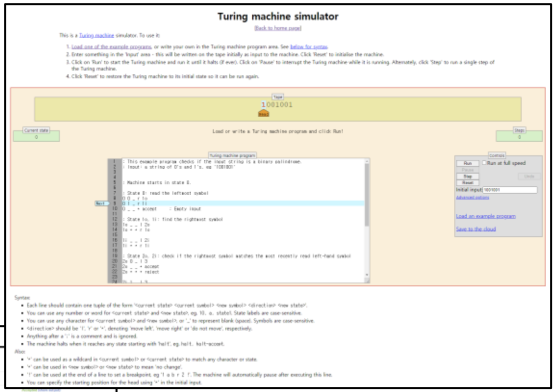
- If the current state reaches one of the halting states, no further output is produced

Turing Machine Simulator

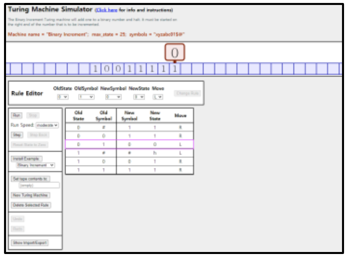
- There are a lot of simulators on the web, but we will create our own new ones

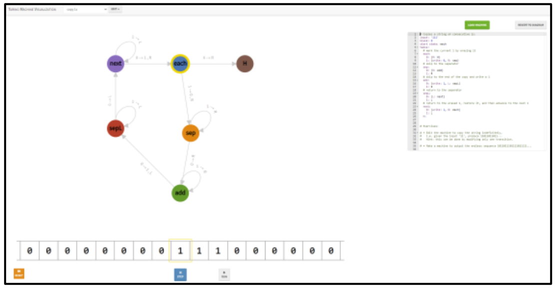






http://morphett.info/turing/turing.html





Turing Machine Simulator

- **Goal**
  - Build the simulator step by step by implementing and testing the following **three key classes** in sequence
- **Step 1: The Table**
  - Encapsulate the table of transition rules, to which an arbitrary number of rules can be added
- **Step 2: The Tape**
  - Encapsulate the strip of tape, which is infinitely extendable in both the left and the right directions
- **Step 3: The Machine**
  - Encapsulate the Turing machine which successively updates the symbols on the tape based on the rules stored in the table

Turing Machine Simulator

- **Template project**
  - **`main.cpp`**
    - Just calls one of the test functions
  - 세 갈래
  - **`test_[table/tape/machine].cpp`**
    - Console-based drivers for testing each class
  - **`util.[h/cpp]`**
    - Utility functions and variables for easier implementation and test
  - **`TuringMachine.[h/cpp]`**
    - Where you will do your homework!
    - Some constants and **`struct Transition`** are pre-defined
    - You should define **`Table`**, **`Tape`**, and **`Machine`** **`class`**es

Step 0: The Constants

```
namespace Turing
{
  const char WILDCARD_SYMBOL = ‘*’;
  const char EMPTY_SYMBOL = ‘_’;

  enum class Move
  {
    NONE = 0, LEFT, RIGHT
  };

  // ... struct Transition, class Table, class Tape, class Machine ...

};
```

Step 0: The Transition

전변된걸

```
struct Transition
{
public:
  Transition(const std::string& curr_s, char read_s, char write_s, Move move, 
const std::string& next_s);

  void print(std::ostream& os) const;

  const std::string& getCurrState() const;
  const std::string& getNextState() const;
  char getReadSymbol() const;
  char getWriteSymbol() const;
  Move getMove() const;

private:
  // ...
};
```

① 3번째에 쓰이는 것.

Step 1: The Table

```
class Table
{
public:
  Table();
  ~Table();
    추가하기
  void addTransition(const std::string& curr_s, char read_s, char write_s, Move move, const std::string& next_s);
  Transition* findTransition(const std::string& curr_s, char read_s);
  void clear(); 다 없애기
  void print(std::ostream& os) const; 저장정보 출력

  void initialize(const std::string& rule_script);
  bool load(const std::string& path);

private:
  // ...
};
```

Step 1: The Table

- **Usage: `addTransition`**
  - Add a new rule (`curr_s, read_s, write_s, move, next_s`)

```
Table table;
table.addTransition("A", '0', '1', Move::LEFT, "B");
table.addTransition("A", '1', '1', Move::RIGHT, "C");
table.addTransition("B", '0', '1', Move::RIGHT, "A");
table.addTransition("B", '*', '1', Move::LEFT, "B");
table.addTransition("C", '0', '1', Move::RIGHT, "B");
table.addTransition("C", '1', '1', Move::NONE, "HALT");
```

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 1: The Table

- **Usage: `print`**
  - Print every rule (`curr_s, read_s, write_s, move, next_s`)

```
table.print(std::cout);
```

move write

```
===
(A, 0, 1, l, B)
(A, 1, 1, r, C)
(B, 0, 1, r, A)
(B, *, 1, l, B)
(C, 0, 1, r, B)
(C, 1, 1, *, HALT)
```

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	*	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 1: The Table

- **Usage: `clear`**
  - Clear all of the existing rules

```
table.clear();
table.print(std::cout);

===
(nothing is printed)
```

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	*	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 1: The Table

- Usage: `findTransition`
  - **Exact match**: Find the first rule that exactly matches both the state and the symbol arguments

```
Transition* t = table.findTransition("C", '1');
t->print(std::cout);

===
(C, 1, 1, *, HALT)
```

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	*	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 1: The Table

- **Usage: `findTransition`**
  - **Exact match**: Find the first rule that exactly matches both the state and the symbol arguments (even if the symbol is `‘*’`)

```
Transition* t = table.findTransition(“B”, ‘*’);
t->print(std::cout);
```

```
===
(B, *, 1, *, HALT)
```

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	*	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 1: The Table

- **Usage:** `findTransition`
  - **Wildcard match**: If there is no exact match, search for an partially-matched rule instead based on the wildcard symbol **'*'**
    - If the symbol argument is **'*'**, find the first rule that matches only the state argument

상태 A, 임의의 Tape symbol인 첫번째 규칙반환

```
Transition* t = table.findTransition("A", '*');
t->print(std::cout);
```

```
===
(A, 0, 1, l, B)
```

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 1: The Table

- **Usage: `findTransition`**
  - **Wildcard match**: If there is no exact match, search for an partially-matched rule instead based on the wildcard symbol **'*'**
    - Otherwise, find the first rule whose current state equals to the state argument and whose tape symbol is **'*'**

```
Transition* t = table.findTransition("B", '1');
t->print(std::cout);
```

```
===
(B, *, 1, l, B)
```

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	*	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 1: The Table

- **Usage: `initialize`**
  - Clear the existing rules, split the input string into a sequence of lines, remove comments from lines, skip white-lines, and add the rule for each remaining line

```
string script =
  “; this script encodes the 3-state busy beaver\n”
  “A 0 1 l B ; comments can be added anywhere\n”
  “A 1 1 r C\n”
  “B 0 1 r A\n”
  “C 0 1 r B\n”
  “C 1 1 * HALT\n”;

Table table;
table.initialize(script);
```

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 1: The Table

- **Usage: `initialize`**
  - Clear the existing rules, split the input string into a sequence of lines, remove comments from lines, skip white-lines, and add the rule for each remaining line

util.h

```
string script =
  "; this script encodes the 3-state busy beaver\n"
  "A 0 1 l B ; comments can be added anywhere\n"
  "A 1 1 r C\n"
  "B 0 1 r A\n"
  "C 0 1 r B\n"
  "C 1 1 * HALT\n";

Table table;
table.initialize(script);
```

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 1: The Table

- **Usage: `load`**
  - Clear the existing rules, open the file stream for the given path, combine lines into a string, call **`initialize`** with the string

`3_beaver.txt`

```
; this script encodes the 3-state busy beaver
A 0 1 l B ; comments can be added anywhere
A 1 1 r C
B 0 1 r A
C 0 1 r B
C 1 1 * HALT
```

```
Table table;
table.load("3_beaver.txt");
```

Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	1	R	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 1: The Table

- **util.h/cpp (wrapped with the namespace `Util`)**
  - The following functions can help for implementing **`initialize()`**

✓ `std::vector<std::string> split(const std::string& s, char c = ‘ ’);` (파이썬 스플릿)

✓ `std::string stripComment(const std::string& s);` – 주석 제거

✓ `bool isWhiteLine(const std::string& s);` 빈 문장 제거

  - The following strings can be used for testing **`initialize()`**

✓ `const std::string binary_palindrome_code;`

✓ `const std::string binary_addition_code;`

✓ `const std::string parenthesis_check_code;`

테스트 돌리면

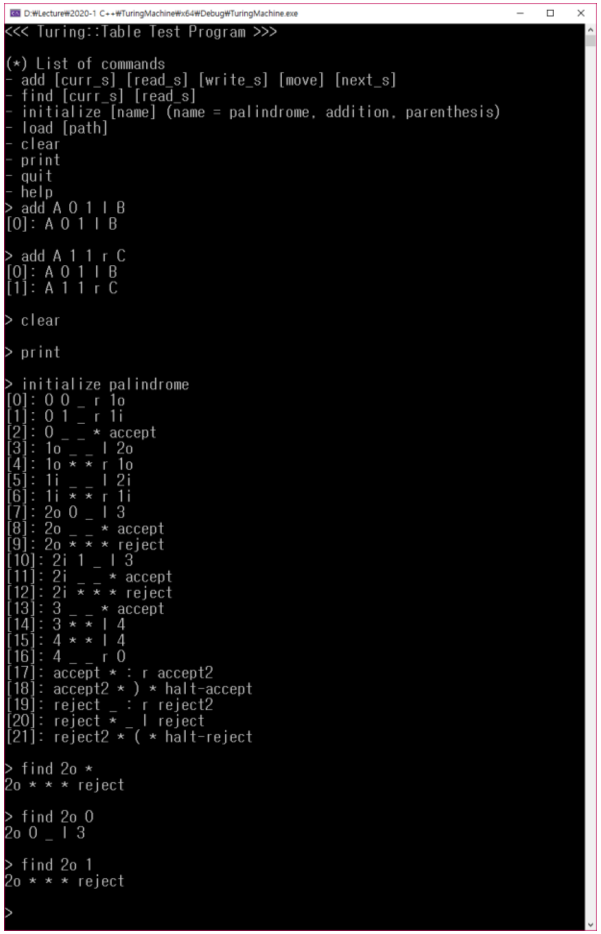
Step 1: The Table

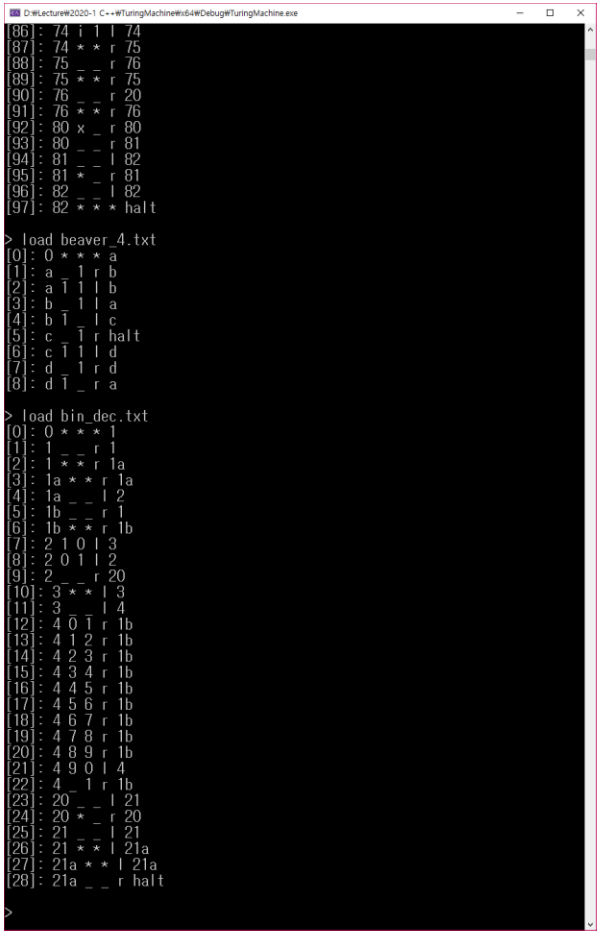
- Test by calling `testTable()` defined in test_table.cpp

```
int main()
{
   testTable();
}
```

Step 1: The Table

- **Test example**
  - add
  - clear
  - print
  - initialize
    - palindrome
    - addition
    - parenthesis
  - find
    - wildcard (*)
  - load
    - beaver_4.txt
    - bin_dec.txt
    - bin_mul.txt





Step 2: The Tape

```
class Tape
{
  Tape();                              // default constructor
  Tape(const Tape& t);                 // copy constructor
  Tape(Tape&& t);                      // move constructor
  ~Tape();                             // destructor

  Tape& operator=(const Tape& t);      // copy assignment operator
  Tape& operator=(Tape&& t);           // move assignment operator

  bool read(int i, char& c) const;
  bool write(int i, char c);

  void push_back(char c);
  void push_front(char c);
```

Step 2: The Tape

```
  // ... continued from the previous slide ...

  void reserve(int newalloc);
  void resize(int newsize);

  int size() const;
  int capacity() const;

  void initialize(const std::string& s);
  void clear();
  void print(std::ostream& os) const;

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

Step 2: The Tape

- **Almost the same as `vector` class explained in Chap. 17~19**
- **Except:**
  - Contain only **`char`**-type elements
    - Neither **`double`**-type elements, nor template typed elements
  - Some kinds of constructors are removed
    - `vector(int n);`
    - `vector(std::initializer_list<double> lst);`
  - Elements can be accessed by calling functions instead of using []
    - `read(int i, char& c);`
    - `write(int i, char c);`
  - Elements can be pushed to back, as well as be pushed to front
    - `push_back(char c);`
    - `push_front(char c);`
  - Some tape-specific functions are added
    - `initialize(const std::string& s);`
    - `clear();`
    - `print(std::ostream& os);`

Step 2: The Tape

- **Usage: `initialize`**
  - Resize to the length of the input string, and copy each `i`-th character of the string into the `i`-th element

```
Tape tape;
tape.initialize("01001");
```

```
sz=5, space=5, elem= | 0 | 1 | 0 | 0 | 1 |
```

Step 2: The Tape

- **Usage: `print`**
  - Print the entire elements in sequence

```
Tape tape;
tape.initialize(“01001”);
tape.print();

===
01001
```

sz=5, space=5, elem=

0	1	0	0	1

Step 2: The Tape

- **Usage: `read`**
  - Get the **`i`**-th element if **`i >= 0 && i < sz`**

```
Tape tape;
tape.initialize("01001");

char c = ' ';
tape.read(4, c);
cout << c;
===
1
```

`sz=5, space=5, elem=`

0	1	0	0	1

Step 2: The Tape

- Usage: `write`
  - Set the `i`-th element if `i >= 0 && i < sz`

```
Tape tape;
tape.initialize("01001");

tape.write(4, '0');
tape.print(cout);
===
01000

sz=5, space=5, elem= | 0 | 1 | 0 | 0 | 0 |
```

Step 2: The Tape

- **Usage: `clear`**
  - Resize to zero (without need of manipulating allocated memory)

```
Tape tape;
tape.initialize("01001");
tape.clear();
tape.print();

===
(nothing is printed)
```

sz=0, space=5, elem=

0	1	0	0	1

Step 2: The Tape

- **Usage: push_back**
  - Reserve space, and add the given element at the back

```
Tape tape;
tape.initialize("01001");
tape.push_back('1');
tape.print();
```

```
===
010011
```

sz=6, space=10, elem=

0	1	0	0	1	1	0	0	0	0

Step 2: The Tape

- **Usage: `push_front`**
  - Reserve space, shift the existing elements to the right, and add the given element at the front

```
Tape tape;
tape.initialize("01001");
tape.push_front('1');
tape.print();

===
101001
```

shifted

`sz=6, space=10, elem=` | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

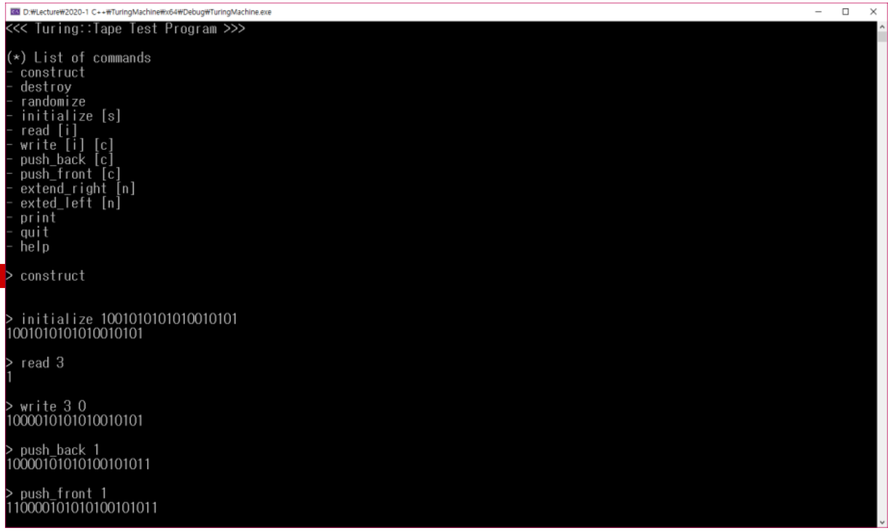
Step 2: The Tape

- Test by calling `testTape()` defined in test_tape.cpp

```
int main()
{
   testTape();
}
```

Step 2: The Tape

- **Test example**
  - construct
  - destroy
  - initialize
  - randomize
  - read
  - write
  - push_back
  - push_front
  - extend_right
  - extend_left



Step 3: The Machine

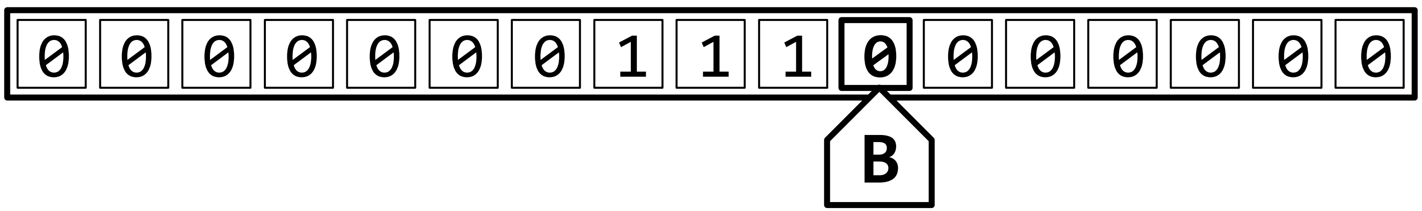
```
class Machine
{
public:
  enum class Mode { NONE, NORMAL, ACCEPT, REJECT, ERROR };

  void initTape(const std::string& initial_symbols);
  void initTable(const std::string& rule_script);
  bool loadTable(const std::string& path);

  void start(const std::string& start_state, const std::string&
accept_state, const std::string& reject_state);
  bool step();

  const Table& getTable() const { return table; }
  const Tape& getTape() const { return tape; }
```

Step 3: The Machine

```
// ...continued from the previous slide...
  const std::string& getCurrentState() const
                              { return current_state; }
  int getCurrentPos() const   { return current_pos; }
  Mode getCurrentMode() const { return current_mode; }

private:
  Table table;
  Tape tape;

  Mode current_mode = Mode::NONE;
  std::string current_state = “”;
  int current_pos = 0;

  std::string accept_state = “”;
  std::string reject_state = “”;
};
```

Step 3: The Machine

- **Some notes for implementing `step()`** [illegible]
  - When write symbol is **'*'**, do not write anything

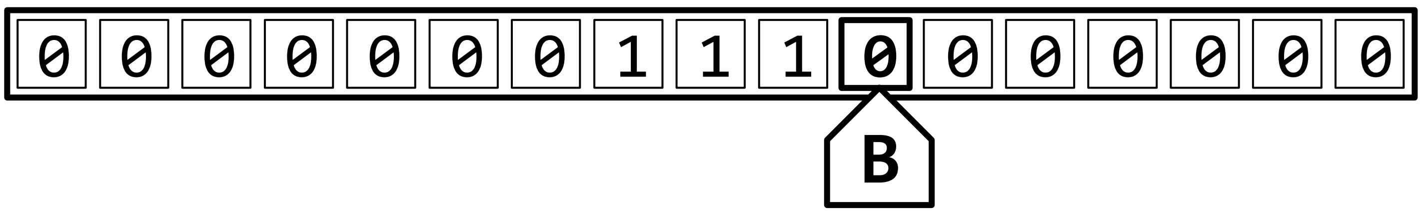


Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	*	*	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 3: The Machine

- **Some notes for implementing `step()`**
  - When move symbol is **'*'**, do not move (neither left nor right)

Move::NONE



Current State	Tape Symbol	Write Symbol	Moving Dir	Next State
A	0	1	L	B
A	1	1	R	C
B	0	*	*	A
B	1	1	L	B
C	0	1	R	B
C	1	1	N	HALT

Step 3: The Machine

- Test by calling `testMachine()` defined in test_machine.cpp

```
int main()
{
    testMachine();
}
```

테스트할 땐
1개의 cpp로!

Step 3: The Machine

```
D:\Lecture\2020-1 C++\TuringMachine\x64\Debug\TuringMachine.exe
<<< 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 [count]
- quit
- help

> init_table palindrome
[0]: 0 0 _ r 1o
[1]: 0 1 _ r 1i
[2]: 0 _ _ * accept
[3]: 1o _ _ l 2o
[4]: 1o * * r 1o
[5]: 1i _ _ l 2i
[6]: 1i * * r 1i
[7]: 2o 0 _ l 3
[8]: 2o _ _ * accept
[9]: 2o * * * reject
[10]: 2i 1 _ l 3
[11]: 2i _ _ * accept
[12]: 2i * * * reject
[13]: 3 _ _ * accept
[14]: 3 * * l 4
[15]: 4 * * l 4
[16]: 4 _ _ r 0
[17]: accept * : r accept2
[18]: accept2 * ) * halt-accept
[19]: reject _ : r reject2
[20]: reject * _ l reject
[21]: reject2 * ( * halt-reject


> init_tape 11100111
11100111

> start 0 halt-accept halt-reject
11100111 [0/NORMAL]
^

> step
_1100111 [1i/NORMAL]
 ^

>
```

```
D:\Lecture\2020-1 C++\TuringMachine\x64\Debug\TuringMachine.exe
      ^
__1001___ [3/NORMAL]
     ^
__1001___ [4/NORMAL]
    ^
__1001___ [4/NORMAL]
   ^
__1001___ [4/NORMAL]
  ^
__1001___ [4/NORMAL]
 ^
__1001___ [0/NORMAL]
  ^
___001___ [1i/NORMAL]
   ^
___001___ [1i/NORMAL]
    ^
___001___ [1i/NORMAL]
     ^
___001___ [1i/NORMAL]
      ^
___001___ [2i/NORMAL]
     ^
___00____ [3/NORMAL]
    ^
___00____ [4/NORMAL]
   ^
___00____ [4/NORMAL]
  ^
___00____ [0/NORMAL]
   ^
____0____ [1o/NORMAL]
    ^
____0____ [1o/NORMAL]
     ^
____0____ [2o/NORMAL]
    ^
_________ [3/NORMAL]
   ^
_________ [accept/NORMAL]
   ^
___:_____ [accept2/NORMAL]
    ^
___:)____ [halt-accept/ACCEPT]
    ^
___:)____ [halt-accept/ACCEPT]
    ^

>
```

Step 3: The Machine

D:₩Lectures₩2024-1 C++₩HW3 - 완성본₩x64₩Debug₩TuringMachine.exe

```
> 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)
(5y, *, *, r, 5y)
(6x, _, 0, l, 3)
(6x, *, *, r, 6x)
(6y, _, 1, l, 3)
(6y, *, *, r, 6y)
(10, _, c, l, 11)
(10, *, *, r, 10)
(11, b, b, r, 12)
(11, x, x, r, 12)
(11, y, y, r, 12)
(11, *, *, l, 11)
(12, 0, x, r, 13x)
(12, 1, y, r, 13y)
(12, c, c, l, 20)
(13x, _, 0, l, 11)
(13x, *, *, r, 13x)
(13y, _, 1, l, 11)
(13y, *, *, r, 13y)
(20, x, 0, l, 20)
(20, y, 1, l, 20)
(20, b, b, r, 21)
(21, _, d, l, 22)
(21, *, *, r, 21)
(22, 1, 0, r, 23)
(22, 0, 1, l, 22)
(22, c, !, r, error)
(23, d, d, r, 50)
```

D:₩Lectures₩2024-1 C++₩HW3 - 완성본₩x64₩Debug₩TuringMachine.exe

```
> init_tape 101
101

> start 0 halt halt
101 [0/NORMAL]
^

> run
_101 [1/NORMAL]
^
a101 [2/NORMAL]
 ^
a101 [2/NORMAL]
  ^
a101 [2/NORMAL]
   ^
a101_ [2/NORMAL]
    ^
a101b [3/NORMAL]
   ^
a101b [3/NORMAL]
  ^
a101b [3/NORMAL]
 ^
a101b [3/NORMAL]
^
a101b [4/NORMAL]
 ^
ay01b [5y/NORMAL]
  ^
ay01b [5y/NORMAL]
   ^
ay01b [5y/NORMAL]
    ^
ay01b_ [6y/NORMAL]
     ^
ay01b1 [3/NORMAL]
    ^
ay01b1 [3/NORMAL]
   ^
ay01b1 [3/NORMAL]
  ^
ay01b1 [3/NORMAL]
 ^
ay01b1 [4/NORMAL]
```

D:₩Lectures₩2024-1 C++₩HW3 - 완성본₩x64₩Debug₩TuringMachine.exe

```
a101b11__________ [501/NORMAL]
      ^
a101b1___________ [501/NORMAL]
     ^
a101b____________ [501/NORMAL]
    ^
a101_____________ [502/NORMAL]
   ^
a101_____________ [502/NORMAL]
  ^
a101_____________ [502/NORMAL]
 ^
a101_____________ [502/NORMAL]
^
_101_____________ [502a/NORMAL]
 ^
_101_____________ [502a/NORMAL]
  ^
_101_____________ [502a/NORMAL]
   ^
_101_____________ [502a/NORMAL]
    ^
_101_____________ [503/NORMAL]
     ^
_101_i___________ [504/NORMAL]
      ^
_101_is__________ [505/NORMAL]
       ^
_101_is__________ [506/NORMAL]
        ^
_101_is_p________ [507/NORMAL]
         ^
_101_is_pr_______ [508/NORMAL]
          ^
_101_is_pri______ [509/NORMAL]
           ^
_101_is_prim_____ [510/NORMAL]
            ^
_101_is_prime____ [511/NORMAL]
             ^
_101_is_prime!___ [halt/ACCEPT]
              ^
_101_is_prime!___ [halt/ACCEPT]
              ^
```

Step 3: The Machine

D:₩Lectures₩2024-1 C++₩HW3 - 완성본₩x64₩Debug₩TuringMachine.exe

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

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

> load_table bin_mul.txt
(0, *, *, l, 1)
(1, _, _, l, 2)
(2, _, 0, r, 3)
(3, _, _, r, 10)
(10, _, _, l, 11)
(10, x, x, l, 11)
(10, 0, 0, r, 10)
(10, 1, 1, r, 10)
(11, 0, x, r, 20)
(11, 1, x, r, 30)
(20, _, _, r, 20)
(20, x, x, r, 20)
(20, *, *, r, 21)
(21, _, 0, l, 25)
(21, *, *, r, 21)
(25, _, _, l, 26)
(25, *, *, l, 25)
(26, _, _, r, 80)
(26, x, x, l, 26)
(26, 0, 0, *, 11)
(26, 1, 1, *, 11)
(30, _, _, r, 30)
(30, x, x, r, 30)
(30, *, *, r, 31)
(31, _, _, l, 32)
(31, *, *, r, 31)
(32, 0, o, l, 40)
(32, 1, i, l, 50)
(32, o, o, l, 32)
(32, i, i, l, 32)
(32, _, _, r, 70)
(40, _, _, l, 41)
(40, *, *, l, 40)
(41, _, _, l, 41)
(41, *, *, l, 42)
(42, _, _, l, 43)
(42, *, *, l, 42)
```

D:₩Lectures₩2024-1 C++₩HW3 - 완성본₩x64₩Debug₩TuringMachine.exe

```
> init_tape 111_11
111_11

> start 0 halt halt
111_11 [0/NORMAL]
^

> run
_111_11 [1/NORMAL]
^
__111_11 [2/NORMAL]
^
0_111_11 [3/NORMAL]
 ^
0_111_11 [10/NORMAL]
  ^
0_111_11 [10/NORMAL]
   ^
0_111_11 [10/NORMAL]
    ^
0_111_11 [10/NORMAL]
     ^
0_111_11 [11/NORMAL]
    ^
0_11x_11 [30/NORMAL]
     ^
0_11x_11 [30/NORMAL]
      ^
0_11x_11 [31/NORMAL]
       ^
0_11x_11_ [31/NORMAL]
        ^
0_11x_11_ [32/NORMAL]
       ^
0_11x_1i_ [50/NORMAL]
      ^
0_11x_1i_ [50/NORMAL]
     ^
0_11x_1i_ [51/NORMAL]
    ^
0_11x_1i_ [52/NORMAL]
   ^
0_11x_1i_ [52/NORMAL]
  ^
0_11x_1i_ [52/NORMAL]
 ^
0_11x_1i_ [53/NORMAL]
^
```

D:₩Lectures₩2024-1 C++₩HW3 - 완성본₩x64₩Debug₩TuringMachine.exe

```
10101_xxx_11000 [26/NORMAL]
     ^
10101_xxx_11000 [80/NORMAL]
      ^
10101__xx_11000 [80/NORMAL]
       ^
10101___x_11000 [80/NORMAL]
        ^
10101_____11000 [80/NORMAL]
         ^
10101_____11000 [81/NORMAL]
          ^
10101______1000 [81/NORMAL]
           ^
10101_______000 [81/NORMAL]
            ^
10101________00 [81/NORMAL]
             ^
10101_________0 [81/NORMAL]
              ^
10101___________ [81/NORMAL]
               ^
10101___________ [82/NORMAL]
              ^
10101___________ [82/NORMAL]
             ^
10101___________ [82/NORMAL]
            ^
10101___________ [82/NORMAL]
           ^
10101___________ [82/NORMAL]
          ^
10101___________ [82/NORMAL]
         ^
10101___________ [82/NORMAL]
        ^
10101___________ [82/NORMAL]
       ^
10101___________ [82/NORMAL]
      ^
10101___________ [82/NORMAL]
     ^
10101___________ [82/NORMAL]
    ^
10101___________ [halt/ACCEPT]
    ^
10101___________ [halt/ACCEPT]
    ^

>
```

Note

- **Report (*.pdf)**
  - Title page
    - Course title, submission date, affiliation, student ID, full name
  - Begin with a summary of your results
    - Which requirements did you fulfill? And which didn't you? (present a simple table)
    - Did you implement some additional features? What are those?
  - For each requirement (basic/advanced/optional), explain how you fulfilled it
    - Do not just dump the entire code
    - It's okay to copy snippets of your code to complement written description
  - Conclude with some comments on your work
    - Key challenges you have successfully tackled
    - Limitations you hope to address in the future

Submission

- **Compress your code and report into a single *.zip file**
  - Code
    - The entire project folder including *.sln, *.cpp, *.h, etc.
    - The grader should be able to open the *.sln with Visual Studio 2019 and build/run the project immediately without any problems
    - Remove Debug, Release, and .vs subfolders for compactness
  - Report
    - A single *.pdf file
    - You should convert your word format (*.hwp, *.doc, *.docx) to PDF format (*.pdf) before zipping
  - Name your zip file as your student ID
    - ex) 2012726055.zip

- **Upload to homework assignment in KLAS**

- **Due at 6/17 (Mon), 11:59 PM**