CS210-001: Assignment 3
Fall 2019 (201930)

Due Date and Time: Tuesday, October 1, 2019 at 11:20 AM

(100 marks) Before the invention of the electronic programmable computer (i.e., circa 1935), many researchers were already studying important problems related to what it means for a task to be computable. One of these researchers was *Alan Turing* (1912 – 1954). In an attempt to better understand the theoretical abilities and limitations of computability, Turing developed, in 1937, a mathematical model of a hypothetical computing machine that used a predefined set of rules to determine a result from a set of input symbols. Hypothetical computing machines adhering to Turing's mathematical model became known as *Turing machines*. As a result of his work, he is widely considered to be one of the fathers of computer science.

A Turing machine consists of *four* components:

a. A *tape*, divided into cells, into which symbols can be written and from which symbols can be read. Each cell can hold one symbol. The tape is assumed to be arbitrarily extendable to the left and to the right (i.e., a Turing machine is always supplied with as much tape as it needs for its computation).

b. A *movable read/write pointer* that reads symbols from cells on the tape and writes symbols to cells on the tape.

c. A *state register* that stores the state of the Turing machine. At any time, a Turing machine can be in only one of many (possibly infinite) states. One special state is called the *start state*.

d. An *action table* containing a set of rules that tells the Turing machine what to do.

Generally, a Turing machine works as follows. First, it will read the symbol from the tape in the cell currently under the read/write pointer. Then, given the current state of the Turing machine and the value of the symbol read from the tape, the Turing machine will write a symbol to the tape in the cell currently under the read/write pointer, (possibly) change state, and (possibly) move the read/write pointer to the cell on the left or right of the cell currently under the read/write pointer. The symbol written to the tape, the (possible) change of state, and the direction that the read/write pointer is (possibly) moved is determined by the rule in the action table that corresponds to the current state and the value of the symbol read from the tape.

PART 1

Implement a Turing machine that adheres to the following requirements. The state register should be declared as an `int`. The tape should be declared as a variable of a *doubly-linked list type* defined as a C++ class, where each node corresponds to one cell on the tape. The action table should be declared as a variable of an *array-based list type*

defined as a C++ class, where each item corresponds to a `struct` consisting of *five* members corresponding to: (1) the current state, (2) the symbol read from the tape, (3) the next state, (4) the symbol to be written to the tape, and (5) the direction to move the read/write pointer, respectively. Each item in the action table corresponds to a "command" with the following format:

(*current state*, *symbol read*) -> (*next state*, *symbol to write*) -> *direction to next cell*

The *current state* and *next state* members should be of type `int`. The *symbol read* and *symbol to write* members should be of type `char`. The values of *current state*, *symbol read*, *next state* and *symbol to write* are problem dependent. The symbol `B` is a *reserved* symbol used to indicate that a cell is empty (i.e., blank). The *direction to next cell* member should be of type `char`, where `L`, `-` (i.e., the dash character), `R`, and `H` correspond to moving the read/write pointer to the left, no movement, moving to the right, and halting the computation (i.e., the problem is solved), respectively. Your C++ classes should be implementations of the doubly-linked list algorithms given on the CS210 Algorithms web page. Do not waste your time implementing any algorithms that you don't actually need.

Prior to solving a problem using your Turing machine, the tape, state register, and action table must be initialized, and the read/write pointer must be positioned over a cell. To make your Turing machine capable of solving multiple problems, read the initial values from a text file containing data of the following format:

initial_state_register_value
initial_tape_values
first_row_of_action_table_values
.
.
.
last_row_of_action_table_values
`-1` // action table terminator
initial_read_write_pointer_position

For example, a text file containing the following values will "program" your Turing machine to count to 16 in binary.

```
0 // initial state
B 0 0 0 0 B // initial cell values on tape
0 0 0 0 R // move right
0 1 0 1 R // move right
0 B 1 B L // when a blank at the right is found, change to
state 1 and move left
1 0 0 1 R // if a 0 is changed to 1, change to state 0 and
move right
1 1 1 0 L // if a 1 is changed to 0, move left
```

```
1 B 0 1 H // if a B is changed to 1, stop
-1 // end of action table
6 // initial position of read/write pointer
```

Your program should print out the contents of the tape (and the position of the read/write pointer), from left to right: (1) after it is initialized, (2) after each intermediate computation (i.e., after an individual action is completed), and (3) after the computation halts. For example, the initial cell values on the tape and the initial position of the read/write pointer (represented by the character ^) would print as follows:

```
B 0 0 0 0 B
          ^
```

PART 2

For programming problems, the Results are worth 70%. So, for this problem, the Results are worth 70 marks out of the 100 marks available. Demonstrate that your Turing machine is correct. You can demonstrate that it works by using the `count.txt` and `subtract.txt` files in the `/home/venus/hilder/cs210/assignment3/datafiles` directory as input. The `count.txt` file will "program" your Turing machine to count to 16 in binary. The `subtract.txt` file will "program" your Turing machine to subtract one integer from another, where the value of an integer is represented as a sequence of `1`s. For example, the five digit sequence `11111` represents the integer 5. Your demonstration should be captured in script files.

WHAT TO SUBMIT

If you are working alone, submit to UR Courses: (1) all your source code files (i.e., *only* the `.cpp` and `.h` files) zipped into a single file called `cppandhfiles`, (2) a script file called `countscript` showing the compilation and execution of your program using `count.txt` as input, and (3) a script file called `subtractscript` showing the compilation and execution of your program using `subtract.txt` as input.

If you are working with a partner, *one of the partners* should submit to UR Courses: (1) a file named `partners` that provides the names and student numbers of the partners and the relative contributions of the partners (should total 100%), (2) all your source code files (i.e., *only* the `.cpp` and `.h` files) zipped into a single file called `cppandhfiles`, (3) a script file called `countscript` showing the compilation and execution of your program using `count.txt` as input, and (4) a script file called `subtractscript` showing the compilation and execution of your program using `subtract.txt` as input. *The other partner* should submit to UR Courses: (1) a file named `partners` that provides the names and student numbers of the partners and the relative contribution of the partners (should total 100%). Note that *both* partners need to submit the `partners` file.