TIC1002—Introduction to Coputing and Programming II
National University of Singapore
School of Continuing and Lifelong Education

Practical Examination 2

20 April 2019

Time allowed: 2 hours

Instructions (please read carefully):

1. This is **an open-book exam**. You are allowed to bring in any course or reference materials in printed form. No electronic media or storage devices are allowed.
2. This practical exam consists of **one** questions. The time allowed for solving this test is **2 hours**.
3. The maximum score of this test is **20 marks** with 5 additional bonus marks. Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question.
4. You are advised to attempt all questions. Even if you cannot solve a question correctly, you are likely to get some partial credit for a credible attempt.
5. While you are provided with the template `pe2-template.cpp` to work with, your answers should be submitted on Coursemology. Note that you can **only run the test cases on Coursemology for a limited number of tries** because they are only for checking that your code is submitted correctly. You are expected to test your own code for correctness locally on VS Code and not depend only on the provided test cases. Do ensure that you pasted your answers correctly by running the test cases at least once.
6. In case there are problems with Coursemology.org and we are not able to upload the answers to Coursemology.org, you will be required to name your file `pe2-<student no>.cpp` where `<student no>` is your student number and leave the file in a `tic1002-pe2` folder on the Desktop. If your file is not named correctly, we will choose any .cpp file found at random.
7. Please note that it shall be your responsibility to ensure that your solution is submitted correctly to Coursemology or correctly left in the desktop folder at the end of the examination. Failure to do so will render you liable to receiving a grade of **ZERO** for the Practical Exam, or the parts that are not uploaded correctly.
8. Please note that while sample executions are given, it is **not sufficient to write programs that simply satisfy the given examples**. Your programs will be tested on other inputs and they should exhibit the required behaviours as specified by the problems to get full credit. There is no need for you to submit test cases.

ALL THE BEST!

Question 1: Snake [20 marks]

Background: *Snake is the common name for a video game concept where the player manoeuvres a line which grows in length, with the line itself being a primary obstacle. The concept originated in the 1976 arcade game Blockade, and the ease of implementing Snake has led to hundreds of versions (some of which have the word snake or worm in the title) for many platforms. After a variant was preloaded on Nokia mobile phones in 1998, there was a resurgence of interest in the snake concept as it found a larger audience. There are over 300 Snake-like games for iOS alone.* *Source: Wikipedia*

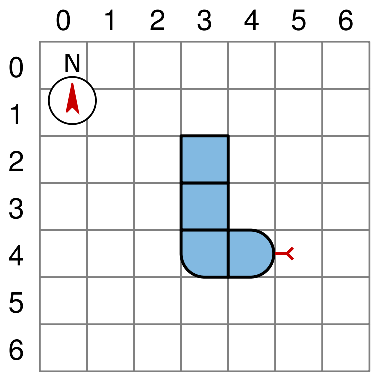
We will be implementing the snake game in this exam. Details are as follows:

Specifications

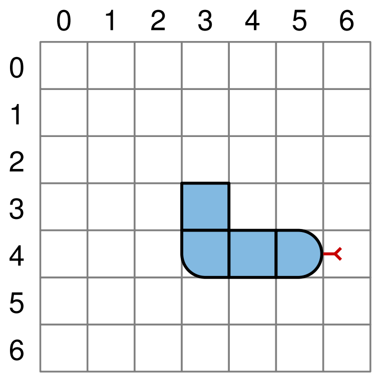
Playing Grid

- The game is played on an $N \times N$ square grid, with the top row and left column numbered as 0. Thus the bottommost row and rightmost column are numbered $N-1$.

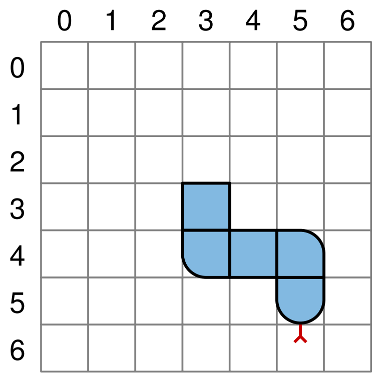
Movement



A snake of length 4, with the head at row 4, col 4.



The snakes moves east, with the head now at row 4, col 5. Note its length remains the same.

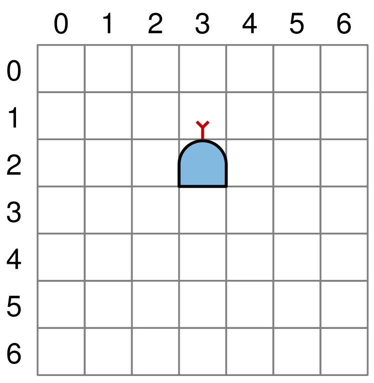


The snakes moves south and <u>grows</u>. The head is now at row 5, col 5 and its length is now 5.

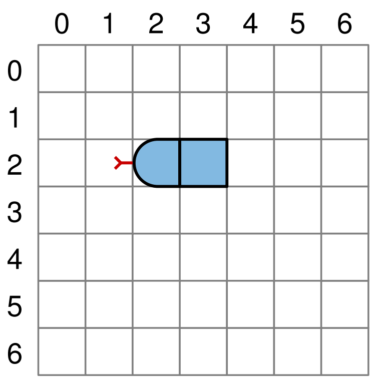
- The body of the snake contains n number of connected segments, each occupying a point on the grid. On one end of the body is the head of the snake.
- The head of the snake can move to an adjacent point on the grid, i.e. in a north, south, east and west direction, with north being the upward direction of decreasing row number. You may assume that the snake will never move out of the grid or collide with itself.
- When the head moves, the entire body will follow behind. Each segment takes up the position that the preceding segment has left behind.

- The snake may also grow during a move. When it grows, a new segment is added to the tail of the snake, that occupies the position where the previous tail was.

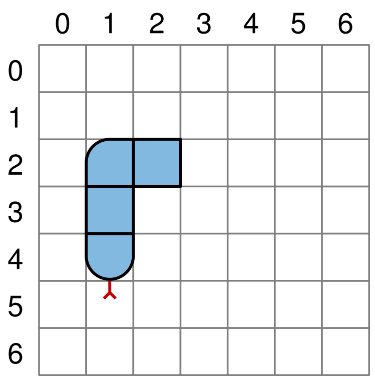
Starting



A snake of length 4 starts at row 2, col 3 as a single segment.



It grows as it moves. . .



. . . until it reaches its full length.

- A snake of length *n* starts as a single segment at a given starting coordinate.
- As it moves, it will grow until it reaches length *n*.

Your task will be to model, implement and manipulate a representation of the snake in the game. This question is rather open-ended. **You are strongly advised to read through the whole question and plan carefully before attempting.**

Implementation

The snake game is supported by the following functions:

A. `struct Snake`

Design and implement `Snake` as a `struct` (or `class` for bonus marks. See Bonus) to represent the state of the snake in the game. You are free to incorporate any C++ STL type or implement your own ADTs.

Note that your choice of implementation might affect the efficiency of the latter functions.

[3 marks]

B. `Snake new_game(int length, int row, int col)`

The function takes in the initial length of the snake, and the row and column of its starting position. It returns a Snake object that you have defined earlier. The snake object should represent the current state of the snake in the game. Provide an implementation of the function `new_game` [3 marks]

C. `void move(Snake &snake, char direction, bool grow)`

The function takes as inputs a snake, a direction in which it will move, and a flag indicating if it will grow.

Direction is specified as single character `'n'`, `'s'`, `'e'` or `'w'` which represents north, south, east and west, respectively. The function will update the snake according to the direction of movement (north in the direction of decreasing row and west of decreasing column).

If `grow` is `true`, then the snake grows by one segment, according to the rules stated above. If `grow` is `false` then the snake does not grow. Note that during the initial starting phase, the snake begins as a single segment, then automatically grows to its correct length with each move. You may assume that until the snake reaches its full length, the input `grow` will always be `false`.

You may also assume that the snake will never be made to move beyond the bounds of the grid.

Provide an implementation of the function `move`. You implementation should run as efficient as possible:

Efficiency	Max marks
Constant time e.g. $O(1)$	10 marks
Linear time	5 marks
Quadratic time	3 marks

You may assume that adding and removing from these STL containers are constant time: `stack`, `queue`, `map`. [10 marks]

D. `void get_state(Snake &snake, vector<vector<bool>> &grid)`

We need to be able to observe the state of the snake in the playing grid. The above function takes as inputs a snake and a playing grid represented as a vector of vectors of `bool`.

The given grid is 2D row-wise layout, i.e. the vector contains rows, which are represented by vectors of `bool`. Each element in the grid is initially set to `false`. The function should set every point the is occupied by the snake to `true`.

You may assume that the grid is large enough to contain the snake.

Provide an implementation for the function `get_state`. [4 marks]

Sample Execution

`Snake snake = new_game(4, 1, 3); get_state(snake, grid);`

Grid contains:

```
F F F F F F F
F F F T F F F
F F F F F F F
F F F F F F F
F F F F F F F
```

```
F F F F F F F
F F F F F F F
```

`move(snake, 's', false); get_state(snake, grid);`

Grid contains:

```
F F F F F F F
F F F T F F F
F F F T F F F
F F F F F F F
F F F F F F F
F F F F F F F
F F F F F F F
```

`move(snake, 's', false); get_state(snake, grid);`

Grid contains:

```
F F F F F F F
F F F T F F F
F F F T F F F
F F F T F F F
F F F F F F F
F F F F F F F
F F F F F F F
```

`move(snake, 'e', false); get_state(snake, grid);`

Grid contains:

```
F F F F F F F
F F F T F F F
F F F T F F F
F F F T T F F
F F F F F F F
F F F F F F F
F F F F F F F
```

`move(snake, 'e', true); get_state(snake, grid);`

Grid contains:

```
F F F F F F F
F F F T F F F
F F F T F F F
F F F T T T F
F F F F F F F
F F F F F F F
F F F F F F F
```

`move(snake, 'n', false); get_state(snake, grid);`

Grid contains:

```
F F F F F F F
F F F F F F F
F F F T F T F
F F F T T T F
F F F F F F F
F F F F F F F
F F F F F F F
```

`move(snake, 'n', true); get_state(snake, grid);`
Grid contains:

```
F F F F F F F
F F F F F T F
F F F T F T F
F F F T T T F
F F F F F F F
F F F F F F F
F F F F F F F
```

`move(snake, 'w', true); get_state(snake, grid);`
Grid contains:

```
F F F F F F F
F F F F T T F
F F F T F T F
F F F T T T F
F F F F F F F
F F F F F F F
F F F F F F F
```

`move(snake, 'w', true); get_state(snake, grid);`
Grid contains:

```
F F F F F F F
F F F T T T F
F F F T F T F
F F F T T T F
F F F F F F F
F F F F F F F
F F F F F F F
```

`move(snake, 'w', false); get_state(snake, grid);`
Grid contains:

```
F F F F F F F
F F T T T T F
F F F F F T F
F F F T T T F
```

```
F F F F F F F
F F F F F F F
F F F F F F F
```

Bonus

You can get up to an additional 5 marks bonus if you implement `Snake` as an OOP class. Thus, you can potentially score 25 out of 20 marks for the PE2.

To write a class, you should work with the template file `pe2-class.cpp` instead. All the supported functions will be implemented as methods of `Snake` class as described:

- `Snake new_game(int length, int row, int col)` will be implemented as the constructor for the class with the same inputs.
- `void move(Snake &snake, char direction, bool grow)` will be implemented as a method `void move(char direction, bool grow)`
- `void get_state(Snake &snake, vector<vector<bool>> &grid)` will be implemented as a method `void get_state(vector<vector<bool>> &grid)`

When submitting on Coursemology, you must remove all the existing functions and submit just your class definition, along with any other ADT that you need. Uncomment the line `#define CLASS` in Coursemology as this will signal the autograder that you are defining a class.

Appendix: ASCII Table

ASCII Control Characters

Num	Sym	Description
0	NUL	Null char
1	SOH	Start of Heading
2	STX	Start of Text
3	ETX	End of Text
4	EOT	End of Transmission
5	ENQ	Enquiry
6	ACK	Acknowledgment
7	BEL	Bell
8	BS	Backspace
9	HT	Horizontal Tab
10	LF	Line Feed
11	VT	Vertical Tab
12	FF	Form Feed
13	CR	Carriage Return
14	SO	Shift Out / X-On
15	SI	Shift In / X-Off
16	DLE	Data Line Escape
17	DC1	Device Control 1
18	DC2	Device Control 2
19	DC3	Device Control 3
20	DC4	Device Control 4
21	NAK	Negative Acknowl
22	SYN	Synchronous Idle
23	ETB	End of Trans Block
24	CAN	Cancel
25	EM	End of Medium
26	SUB	Substitute
27	ESC	Escape
28	FS	File Separator
29	GS	Group Separator
30	RS	Record Separator
31	US	Unit Separator

ASCII Printable Characters

Num	Sym	Num	Sym	Num	Sym
32		64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	<DEL>

— E N D O F P A P E R —