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

Practical Examination 2

17 April 2021

Time allowed: 1 hour 30 min

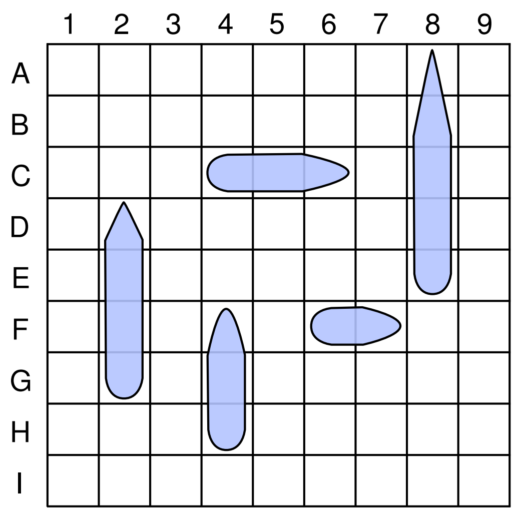
Instructions (please read carefully):

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** question. The time allowed for solving this test is **1 hour 30 min**.
3. The maximum score of this test is **10 marks** with 3 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: Battleship [10 marks]

Batleship is a game that is played on a square grid that contains a fleet of ships.



As illustrated in the above diagram, the ships have different lengths and occupies different number of points on the grid. No ship may occupy overlapping points.

After a player secretly places their ships, the opponent will fire a "shot" at a point in the grid. If a ship occupies that point, that particular part of the ship is destroyed. When every part of a ship is destroyed, the entire ship is sunk. Players take turns to fire shots at each other until a player's entire fleet is sunk.

For this practical exam, we will model the playing grid of a player's battleship game and implement the supporting functions.

A. `struct Grid`

Design and implement a `struct` (or `class` for bonus marks. See Bonus) to represent the state of the battleship grid. You may assume that the battleship grid is at most a 9×9 grid, and that it can contain any number of battleships.

You are free to incorporate any C++ STL type or implement your own ADTs and helper functions.

Hint: it might be useful to use multiple ADTs to keep track of different information.

[2 marks]

B. `bool add_ship(string name, vector<string> points, Grid & grid)`

The function takes the name of a ship, a vector of points which the ship will occupy, and the battleship grid. If all the points of this ship is unoccupied, the ship is added to the battleship grid at the given points and `true` is returned. Otherwise the grid remains unchanged and `false` is returned.

The points are given in a 2-character string, e.g. `"A1"`, with the first character being an uppercase letter from `A` to `I`, and the second character is a digit from `1` to `9`.

You may assume the points given are always in a straight line and do not exceed the bounds of the grid. [4 marks]

C. `string attack(string point, Grid & grid)`

The function takes as inputs a point and the battleship grid. It simulates the opposing player firing a shot at the point in the grid. The possible outcome are:

- If the point is currently unoccupied by any ship, the grid remains unchanged and the string `"Missed"` is returned.
- If the point is occupied by a previously damaged part of a ship, the grid remains unchanged and the string `"Already Hit"` is returned.
- If the point is occupied by an undamaged part of a ship, the grid is updated to reflect the new damage taken by the ship at the given point. If the ship still has undamaged parts, the string `"<ship name> Hit"` is returned. Otherwise the entire ship is sunk and the string `"<ship name> Sunk"` is returned. `<ship name>` in the string is the name of the ship that is hit.

[4 marks]

Sample Execution

```
Grid grid;
cout << boolalpha;  // set to print true/false for bool
cout << add_ship("Destroyer", {"A1", "A2", "A3"}, grid) << endl;
cout << add_ship("Submarine", {"A1", "B1"}, grid) << endl;
cout << add_ship("Submarine", {"C1", "D1"}, grid) << endl;

cout << attack("A1", grid) << endl;
cout << attack("B2", grid) << endl;
cout << attack("C1", grid) << endl;
cout << attack("D1", grid) << endl;
cout << attack("C1", grid) << endl;
```

Output

```
true
false
true
Destroyer Hit
Missed
Submarine Hit
Submarine Sunk
Already Hit
```

Bonus

You can get up to an additional 3 marks bonus if you implement `Grid` as an OOP class. Thus, you can potentially score 13 out of 10 marks for 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 `Grid` class as described:

- `bool add_ship(string name, vector<string> points, Grid & grid)` will be implemented as a method `bool add_ship(string name, vector<string> points)`
- `string attack(string point, Grid & grid)` will be implemented as a method `string attack(string point)`

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 —