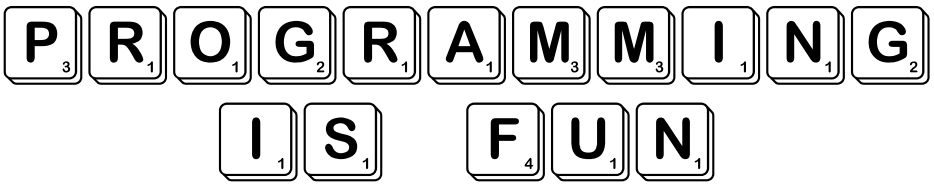
MSc (Computing Science) 2019-2020
C/C++ Laboratory Examination

Imperial College London

**Tuesday 7 January 2020, 10h00–12h00**




- ☞ You are advised to use the first 10 minutes to read through the questions.
- ☞ Log into the Lexis exam system using your DoC login as both your login and as your password (**do not use your usual password**).
- ☞ You must add to the pre-supplied header file **scrabble.h**, pre-supplied implementation file **scrabble.cpp** and must create a **makefile** according to the specifications overleaf.
- ☞ You will find source files **scrabble.cpp**, **scrabble.h**, and **main.cpp**, and data file **words.txt** in your Lexis home directory (/**exam**). If one of these files is missing alert the invigilators.
- ☞ **Save your work regularly.**
- ☞ Please log out once the exam has finished. No further action needs to be taken to submit your files.
- ☞ No communication with any other student or with any other computer is permitted.
- ☞ You are not allowed to leave the lab during the first 30 minutes or the last 15 minutes.
- ☞ **This question paper consists of 5 pages.**

## Problem Description

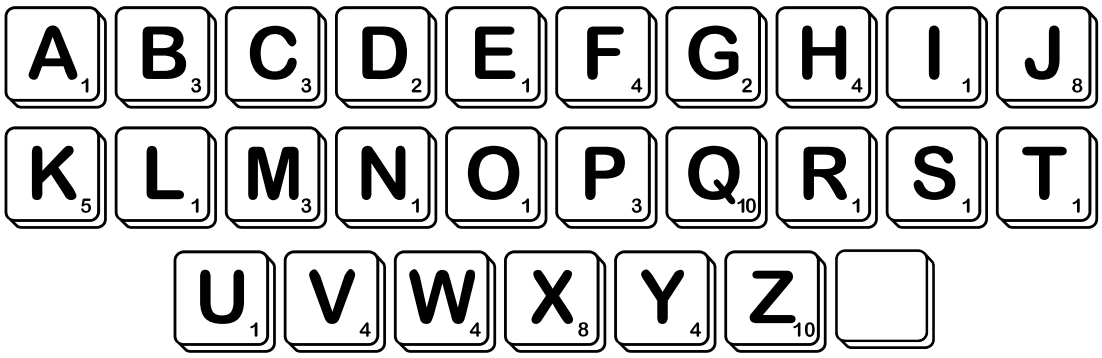
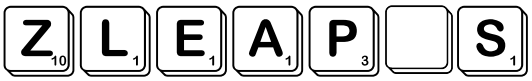



Figure 1: Scrabblegram Tiles

Scrabblegram is a quick-play variant of the world-famous Scrabble[1] word game. The aim of the game is to form a high-scoring English word from a collection of seven *tiles*, each of which represents a letter of the alphabet (see Figure 1). Each tile is annotated with a numerical *tile score* according to the letter it represents. There is also a special "blank" tile which can represent any letter, but which has a tile score of zero.

**When forming a word, each of the seven tiles in a collection may be used only once.** For example, given the tiles:




words that can be formed include **LEAP**, **LEAPS**, **PLEASE**, **APPLES**, **WASP** and **SPATZLE**. It is not possible, however, to make the word **PIZZA** from this collection of tiles.

The *word score* for a formed word is computed taking into account the *tile score* for each of the tiles used and *score modifiers* specified for each position of the formed word. Possible score modifiers are:

- **None**. The contribution of the tile towards the word score is the tile score.
- **Double letter score**. The contribution of the tile towards the word score is twice the tile score.
- **Triple letter score**. The contribution of the tile towards the word score is three times the tile score.
- **Double word score**. The contribution of the tile towards the word score is the tile score. However, the total number of points should be doubled once the final score modifier has been applied.
- **Triple word score**. The contribution of the tile towards the word score is the tile score. However, the total number of points should be tripled once the final score modifier has been applied.

A bonus of 50 points is added to the word score if all seven tiles are used to make up the word.

For example, given the tiles above and the score modifiers:

{None, Triple Letter Score, None, None, Double Word Score, None, None}

The word **LEAP** scores 8 (1+3*1+1+3) points (since the word is not long enough to reach the **Double Word Score** score modifier), **WASP** scores 7 (0+3*1+1+3) points (since blank tiles score 0), **PLEASE** scores 18 ((3+3*1+1+1+1+0)*2) points, and **SPATZLE** scores 96 ((1+3*3+1+0+10+1+1)*2+50) points.

[1]SCRABBLE® is a registered trademark. All rights in the game are owned in the USA and Canada by Hasbro Inc. and throughout the rest of the world by J.W. Spear and Sons, a subsidiary of Mattel Inc.

## Pre-supplied functions and files

You are supplied with a main program in **main.cpp**, and a data file **words.txt** containing a dictionary of English words in uppercase. An extract of **words.txt** is presented below:

```
...
ABLATION
ABLATIONS
ABLATITIOUS
ABLATIVAL
...
JUGS
JUGSFUL
JUGULA
JUGULAR
...
WHEATIER
WHEATIEST
WHEATLAND
WHEATLANDS
...
```

You are also supplied with the beginnings of the header file **scrabble.h** (for your function prototypes) and the beginnings of the implementation file **scrabble.cpp** (for your function definitions).

Note **scrabble.h** contains the following enumerated type definition:

```
enum ScoreModifier { NONE, DOUBLE_LETTER_SCORE, TRIPLE_LETTER_SCORE,
  DOUBLE_WORD_SCORE, TRIPLE_WORD_SCORE };
```

## Specific Tasks

1. Write an integer-valued function `tile_score(tile)` which returns the tile score for a given tile. Here `tile` is an input parameter of character type. If `tile` is a letter (whether uppercase or lowercase) then the function should return the tile score for that letter. If `tile` is `' '` or `'?'` (both of which can be used to represent the blank tile), the function should return 0. Otherwise the function should return -1.

   For example, the code:

   ```
   cout << "Tile score for 'P' is " << tile_score('P') << endl;
   ```

   should display the output

   ```
   Tile score for 'P' is 3
   ```

   Similarly, the code

   ```
   cout << "Tile score for '1' is " << tile_score('1') << endl;
   ```

   should display the output

   ```
   Tile score for '1' is -1
   ```

2. Write a **recursive** Boolean function `can_form_word_from_tiles(word, tiles, played_tiles)` which determines whether a given word can be made from a given collection of tiles. Here `word` is an input parameter of string[2] type describing the target word while `tiles` is an input parameter of string type describing the tiles in the collection. You may assume that the target word is a valid English word. If the target word can be formed from the tiles according to the rules described in the Problem Description, the function should return `true`, and output parameter `played_tiles` should be a null-terminated string consisting of the ordered collection of tiles used. Otherwise the function should return `false`.

   For example, the code:

   ```
   char played_tiles[80];
   bool success = can_form_word_from_tiles("LEAP", "ZLEAP?S", played_tiles);
   ```

   should result in `success` having the value `true` and `played_tiles` having value "LEAP".

   Likewise the code:

   ```
   char played_tiles[80];
   bool success = can_form_word_from_tiles("APPLES", "ZLEAP?S", played_tiles);
   ```

   should result in `success` having the value `true` and `played_tiles` having value "AP?LES"[3].

   As a final example, the code:

   ```
   char played_tiles[80];
   bool success = can_form_word_from_tiles("PIZZA", "ZLEAP?S", played_tiles);
   ```

   should result in `success` having the value `false`.

   **For full credit for this question, your function must be recursive. Note that you are free to use helper functions, which need not be recursive, in order to enhance the elegance of your solution.**

3. Write an integer-valued function `compute_score(played_tiles, score_modifiers)` which returns the word score given a `played_tiles` string and an array of `score_modifiers`. The individual elements in the `score_modifiers` array are of enumerated type `ScoreModifier`.

   For example, the code:

   ```
   ScoreModifier sm0[]={NONE, TRIPLE_LETTER_SCORE, NONE, NONE,
     DOUBLE_WORD_SCORE, NONE, NONE};
   int score = compute_score("LEAP", sm0);
   ```

   should result in `score` having the value `8`.

   Likewise, the code:

   ```
   score = compute_score("AP?LES", sm0);
   ```

   should result in `score` having the value `18`.

   As a final example, the code

   ```
   score = compute_score("SPA?ZLE", sm0);
   ```

   should result in `score` having the value `96`.

[2]We assume the use of C-style strings but you may also use C++-style strings if you wish.
[3]"A?PLES" is also acceptable.

4. Write an integer-valued function `highest_scoring_word_from_tiles(tiles, score_modifiers, word)` which returns the highest word score that can be achieved given a particular collection of tiles and score modifiers, using any of the words in the supplied dictionary. Here `tiles` and `score_modifiers` are input parameters of type string and `ScoreModifier[]` respectively.

   If it not possible to make any word in the supplied dictionary from the tiles then the function should return -1. Otherwise output parameter `word` should contain the word attaining the highest word score, and the function should return the achieved word score. For example, the code:

```
ScoreModifier sm1[]={NONE, NONE, DOUBLE_LETTER_SCORE, NONE, NONE, NONE, NONE};
char word[512];
score = highest_scoring_word_from_tiles("WBNNOER", sm1, word);
cout << "The highest scoring word that can be made from the tiles 'WBNNOER'" << endl;
cout << "  with a double letter score on the third letter is:" << endl;
if (score < 0)
  cout << "(no word found)" << endl;
else
  cout << "'" << word << "' (" << score << " points)" << endl;
```

   should result in the following output:

```
The highest scoring word that can be made from the tiles 'WBNNOER'
  with a double letter score on the third letter is:
'NEWBORN' (66 points)
```

*(The four parts carry, respectively, 20%, 30%, 25% and 25% of the marks)*

## What to hand in

Place your function implementations in the file **scrabble.cpp** and corresponding function declarations in the file **scrabble.h**. Use the file **main.cpp** to test your functions. Create a **makefile** which will compile your submission into an executable file entitled **scrabble**.

## Hints

1. You will save time if you begin by studying the main program in **main.cpp**, the pre-supplied implementation file **scrabble.cpp**, the pre-supplied header file **scrabble.h** and the given data file **words.txt**.
2. Feel free to define any of your own helper functions which would help to make your code more elegant. This applies to **all questions**.
3. You should feel free to exploit the answer to any earlier questions when answering a question.
4. The standard header `<cctype>` contains some library functions that you may find useful when answering **Question 1**. For example, `tolower(char ch)` returns the lowercase character corresponding to `ch` while `toupper(char ch)` returns the uppercase character corresponding to `ch`. Also `isalpha(ch)` returns true if `ch` is an alphabetical character.
5. Your solution to **Question 2** is required to be **recursive**, although any helper functions you use need not be. You are permitted to add default arguments, but this is not strictly necessary.
6. Try to attempt all questions. If you cannot get one of the questions to work, try the next one.