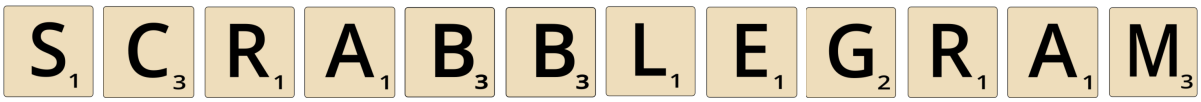
C Practice Test

Imperial College London

Department of Computing

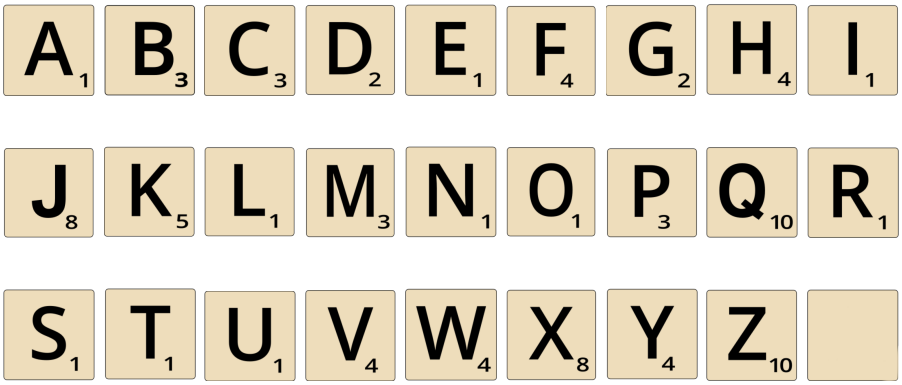


Thanks are due to Will Knottenbelt who came up with this test.

Friday 9th June 2023
16h00 to 17h00
ONE HOUR

- This practice test is **unassessed**. However, if it were assessed, the three questions would have carried *30%, 40% and 30% of the marks* respectively.
- Try to attempt all questions. If you cannot get one of the questions to work, try the next one.
- Your code needs to compile and work correctly in the test environment which will be the same as the lab machines. Comment out any code that does not compile before you finish.
- **Important:** You should only modify the file `player.c`, which should contain all your functions. `player.c` must be in the top-level ∼/**sg** directory, i.e. where it currently is.
- Start by studying the main program **scrabble.c**, the (incomplete) implementation **player.c**, the header **player.h** and the data **words.txt**.
- Feel free to define any of your own helper functions which would help to make your code more elegant; make them static.
- Remember the standard header `<ctype.h>`. It may prove useful.
- You may compile your code however you like; we recommend using **CBuild (aka cb)** as usual.

Problem Description



Scrabblegram is a simplified variant of the world famous Scrabble word game[1]. The aim of the game is to form a high-scoring English word from a *hand* (a collection) of 7 *tiles*, each of which represents a letter of the alphabet (see above). Each tile is annotated with a numerical *tile score* according to the rareness of the letter it represents. There is also a special blank tile which can represent any letter - and which has a tile score of zero.

When forming a word, each of the 7 tiles in a hand may be used only once. For example, given the following hand (which we'd write textually as **ZLEAP?S**):

some words that can be formed without using the blank include **LEAP, LEAPS, PEAL, PEALS, PLEA, PLEAS** and some words that can be formed using the blank include **PLEASE** (using tiles **PLEAS?**), **APPLES** (using tiles **AP?LES**), **WASP** or **GASP** (using tiles **?ASP**) and the unusual word **SPATZLE** (using tiles **SPA?ZLE**). It is not possible, however, to make the word **PIZZA** from this hand, because the tiles include one blank, and neither an "I" nor a second "Z".

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 The contribution of the tile towards the word score is twice the tile score.

Triple letter The contribution of the tile towards the word score is three times the tile score.

Double word The contribution of the tile towards the word score is the tile score. However, the total word score should be doubled at the end.

Triple word The contribution of the tile towards the word score is the tile score. However, the total word score should be trebled at the end.

[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.

Note that any number of double and triple letter modifiers may be applied (to different letters), and also that any number of double and triple word modifiers may be 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 following list of score modifiers:

{None, Triple Letter, None, None, Double Word, 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 modifier), **?ASP** (ie WASP using a blank) scores 7 (0+3*1+1+3) points (since blank tiles score 0), **PLEAS?** (ie PLEASE using a blank) scores 18 ((3+3*1+1+1+1+0)*2) points, and **SPA?ZLE** (ie SPATZLE using a blank) scores 96 ((1+3*3+1+0+10+1+1)*2+50) points.

Pre-supplied functions and files

You are supplied with a main program in **scrabble.c**, and a data file **words.txt** containing a dictionary of uppercase English words no longer than 7 letters. An extract of **words.txt** is presented below:

```
...
ABLATE
ABLATED
...
JUGS
JUGSFUL
JUGULA
...
WRY
WRYBILL
WRYER
...
```

You are also supplied with a header file **player.h** containing your function prototypes, and a skeletal version of the implementation file **player.c** (for your functions to go in).

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

```
typedef enum { NONE, DOUBLE_LETTER, TRIPLE_LETTER, DOUBLE_WORD, TRIPLE_WORD } ScoreModifier;
```

Specific Tasks

1. In **player.c**, write a function: `int tile_score( char tile )`

   which returns the tile score for a given tile. If `tile` is a letter (whether uppercase or lowercase) then the function should return the tile score associated with that letter – see the tiles in the Problem Description. If `tile` is `' '` or `'?'` (either of which can be used to represent a blank tile), the function should return 0. Otherwise the function should return -1.

For example, the code:

```
printf( "Tile score for ‘P’ is %d\n", tile_score(’P’) );
```

should display the output

```
Tile score for ‘P’ is 3
```

Similarly, the code

```
printf( "Tile score for ‘2’ is %d\n", tile_score(’2’) );
```

should display the output

```
Tile score for ‘2’ is -1
```

2. In **player.c**, write a function

   `bool form_word( char *word, char *tiles, char *played_tiles )`

   which determines whether a given word can be made from a given collection of tiles. Here `word[]` is a string (char *) describing the target word and `tiles[]` is a string describing the tiles in the collection. If the target word can be formed from the tiles according to the rules described in the Problem Description, then the tiles played from the hand should be written (as a null-terminated string) into `played_tiles[]`, and the function should return `true`, Otherwise the function should return `false`.

   This function may be recursive or iterative, as you prefer.

   For example, the code:

   ```
   char played_tiles[10];
   bool success = form_word("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[10];
   bool success = form_word("APPLES", "ZLEAP?S", played_tiles);
   ```

   should result in `success` having the value `true` and `played_tiles[]` having value “AP?LES”[2].

   As a final example, the code:

   ```
   char played_tiles[10];
   bool success = form_word("PIZZA", "ZLEAP?S", played_tiles);
   ```

   should result in `success` having the value `false`, and `played_tiles[]` not being modified.

3. In **player.c**, write a function

   `int compute_score( char *played_tiles, ScoreModifier *score_modifiers )`

   which returns the word score given a `played_tiles[]` string and an array `score_modifiers[]`.

   For example, the code:

[2] “A?PLES” is also acceptable, although it is preferable to match the concrete letter first, in preference to the blank.

```
ScoreModifier sm0[]={NONE, TRIPLE_LETTER, NONE, NONE, DOUBLE_WORD, 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 `26`.

As a final example, the code

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

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

4. We have written the following function for you:

   `int highest_score( char *tiles, ScoreModifier *scoremodifiers, char *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[]` is a string input parameter, `scoremodifiers[]` is an input parameter - an array of `ScoreModifier`. You may assume that `scoremodifiers[]` is as long as the tiles.

   If it not possible to make any word in the supplied dictionary from the tiles then the function returns -1. Otherwise the word that attains the highest word score is written into the output parameter `word[]`, and the function returns that highest word score. For example:

```
ScoreModifier sm1[]={NONE, NONE, DOUBLE_LETTER, NONE, NONE, NONE, NONE};
char word[512];
score = highest_score("WBNNOER", sm1, word);
printf( "The highest scoring word that can be made from the tiles 'WBNNOER'\n" );
printf( "with a double letter score on the third letter is: " );
if( score < 0 )
  printf( "(no word found)\n" );
else
  printf( "'%s' (%d points)\n", word, score );
```

   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)
```

   If you've written the first 3 functions correctly, the tests for `highest_score()` should now find the following answers:

```
'NEWBORN' (66 points)
'HARSHEN' (98 points)
'BILAYER' (82 points)
```

Good luck!