Assignment1: scrabble. Due 01-03-2020 23:59

Victor Cionca

Implement a simple game of scrabble.

The user is given a set of **7** random letters, that can contain repetitions (several letters of the same type). The user can view the letters and then needs to start making words with the letters. Each letter has a score and based on that the score of the word will be calculated. The words can only be made from the set of letters available.

The program will keep track of the words entered and the total score. At the end, when the user is done entering words the program will list the words, their individual score and the total score.

Example run:

```
Your letters (and their count) are:
a --> 1
f --> 1
l --> 1
o --> 3
t --> 1
Input word:float
You used: float. Worth 8. Current total 8
Try again? (y/n)y
Input word:foot
You used: foot. Worth 7. Current total 15
Try again? (y/n)y
Input word:foal
You used: foal. Worth 7. Current total 22
Try again? (y/n)y
Input word:loot
You used: loot. Worth 4. Current total 26
Try again? (y/n)y
Input word:toll
Letter l not available
Not enough letters available to make: toll
Try again? (y/n)y
Input word:loaf
You used: loaf. Worth 7. Current total 33
Try again? (y/n)n
You provided the following words:
float -> 8
foot -> 7
foal -> 7
loot -> 4
loaf -> 7
Total score: 33
```

Functionality

Preparation

Generate characters

The set of letters is generated randomly at the start of the program. You should use the function below that generates a given number of letters and stores them in an array provided by the user.

The function uses the **random()** and **srandom()** functions from **stdlib.h**.

- **random()** generates a random positive long integer using a Pseudo Random Number Generator (PRNG)
- **srandom()** initialises the seed of the PRNG.

The function asks the user to provide a number that will be used a seed for the PRNG. Using the same seed will lead to the same state for the PRNG, and therefore the same set of letters generated.

The function also makes sure that half of the letters are vowels, so that words can be formed.

```
/**
 * Generates a number of random lower case letters.
 * The letters will be stored in the given array.
 */
void generate_letters(int number, char *destination){
    char vowels[] = {'a', 'e', 'i', 'o', 'u'};
    int i;
    unsigned int rand_seed;
    printf("Number < 1000:\n");
    scanf("%d", &rand_seed);    // initialise the PRNG
    getchar();
    srandom(rand_seed);

    // Half of the letters should be vowels
    for (i=0;i<number/2;i++, destination++) *destination = vowels[random()%5];
    for (;i<number;i++, destination++) *destination = random()%26+'a';
}
```

Read the letter scores

In this game (as in scrabble) each letter has a value (a score), and these values are summed over the letters of a word to determine the word score. The scores are provided in a separate file (`letter_values`), one value per line corresponding to a letter in alphabetic order. To read the scores you should use the function below that reads an array of a certain length from a file (one array element per line).

The function parameters are:

- path (name) of the file: `letter_values`
- array where the values will be copied
- maximum number of elements to read.

The function makes use of the `FILE` data type and the associated functions for opening and reading from files. **fscanf** should look familiar. These functions are declared in the `stdlib.h` header file, which should be included in your code.

```
/**
 * Tries to read the file into the array.
```

```
 * Each line has one value.
 *
 * Returns the number of lines read or -1 in case of file open error.
 */
int read_array_from_file(const char* filepath, int *array, int array_length){
    FILE *file;
    int i;

    if ((file = fopen(filepath, "r")) == NULL) return -1;
    for (i=0;i<array_length;i++){
        if (fscanf(file, "%d", &array[i]) == EOF) break;
    }

    return i;
}
```

Reading words

The game starts when the user starts making words from the letters available. A word is a sequence of **lowercase** letters that terminates in newline. The program must check that the letters in the word are actually available to the user. In real life the user would take letters from his pool and place them on the board, so by taking a letter from the pool it will no longer be available.

Example:

- consider the user has the letters AFL
- the user cannot make the word FALL because they are missing one "L".

After making one word

- the word score will be computed
- the letters making the word will be returned to the letter pool.

The program should keep track of the words entered by the user, their score, and the total score.

- hint: the word score can be calculated on the fly, instead of storing it.
- hint: when reading and allocating space for the letters, consider that the user only has 7 letters available which means they can't make longer words.

The program does not need to check the spelling or correctness of the word.

Recommended pseudocode

- setup and allocate variables
- read the letter scores from the file
- generate the letters (7 of them)
- go through the letters generated and count those that occur several times
- print the letters generated and each of their counts (in case there are repetitions)
- start reading words
  - read a word
  - calculate the word score
  - return the word letters to the pool
  - store the word entered and the score
  - ask the user if they want to continue; if yes, go back to reading words
- print all the words entered, their individual score, and the total score.

Marking

Basic functionality 70%

- setup and allocate variables 12
- read the letter scores from the file 3
- generate the letters 3
- go through the letters generated and count those that occur several times 10
- print the letters generated and each of their counts (in case there are repetitions) 3
- start reading words
  - read a word 20
  - calculate the word score, print the word and its score 5
  - return the word letters to the pool 8
  - ask the user if they want to continue; if yes, go back to reading words 3
- print the total score 3.

Additional functionality 30%

- keep track of the words entered and their scores; at the end print all the words and their scores
- change the program so it recognizes mixed-case letters within the words
  - hint: should convert all letters in the program either to lower or upper case
- at the end show a letter count over all the words entered.