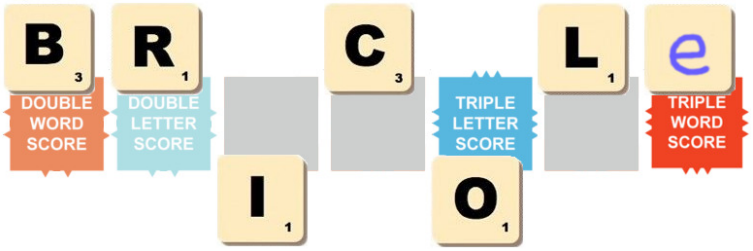
Programmation — Examen

Durée : 2 heures, Calculatrices et Documents Interdits



Dans ce devoir, on s'intéresse au calcul du score d'un mot dans un jeu comme le Scrabble. Dans ce type de jeu, les joueurs posent des tuiles sur des cases pour former des mots, en s'aidant éventuellement des tuiles déjà posées aux tours précédents. Chaque tuile porte une lettre à laquelle est associé une valeur (une même lettre ayant toujours la même valeur, par exemple tous les `'B'` valent 3). La tuile blanche est spéciale : elle remplace une lettre au choix mais sa valeur est nulle.

Le score du mot formé par le joueur est essentiellement la somme des valeurs des lettres formant le mot. Cependant, certaines cases (*Lettre Compte Double*, *Lettre Compte Triple*) agissent comme des bonus multiplicateurs de valeur pour la lettre qui est posée dessus, à condition que celle-ci vienne d'être posée (a contrario, si la lettre a été posée lors d'un tour précédent, le bonus n'est pas actif, ce qui revient à dire que le multiplicateur est 1). Une fois cette somme calculée, certaines cases (*Mot Compte Double*, *Mot Compte Triple*) agissent comme des bonus multiplicateurs de score pour cette somme. Là encore, il faut que la lettre vienne d'être posée pour activer le bonus.

Par exemple, supposons le mot `"BRICOLe"` vient d'être joué tel que montré sur la figure ci-dessus, avec le `'I'` et le `'O'` déjà posés aux tours précédents, et avec le `'e'` minuscule dénotant une tuile blanche de valeur nulle. Au vu de la valeur des lettres et des bonus de cases, le score du mot est égal à $(3+1\times 2+1+3+1+1+0)\times 2\times 3$, c'est-à-dire $11\times 6 = 66$. Ici, la tuile `'R'` vient d'être posée et active son bonus *Lettre Compte Double*, alors que la tuile `'O'` était déjà posée et n'active pas son bonus *Lettre Compte Triple*. Enfin, les deux bonus *Mot Compte Double* et *Mot Compte Triple* sont actifs, car les tuiles `'B'` et `'e'` viennent d'être posées, et donc on multiplie le total 11 par $2\times 3 = 6$, pour un score final de 66.

Dans tout le devoir, pour implémenter une fonction, vous devez RÉUTILISER les fonctions des questions précédentes lorsque cela est possible, même si vous ne les avez pas implémentées.

Par ailleurs, dans tout l'examen, on utilisera pour les Booléens le type `bool`, et les constantes `true` et `false`, tous trois définis dans `<stdbool.h>`, que l'on supposera inclus comme les autres entêtes standards.

1 Tuile et jeu de tuiles

On représente une tuile par la structure `Tile` suivante, où le champ `letter` est la lettre portée par la tuile, et le champ `value` est la valeur de la tuile :

```
typedef struct Tile {
  int value;
  char letter;
} Tile;
```

Question 1. Implémenter la fonction `Tile_Make()` qui fabrique et retourne une tuile à partir de ses paramètres.

```
Tile Tile_Make (int value, char letter);
```

On appelle *jeu de tuiles* (ou *set*), un ensemble constitué d'exactement une tuile pour chaque lettre de l'alphabet, ce qui permet de fixer la valeur de chaque lettre. En effet, les valeurs des lettres diffèrent d'une langue à une autre. Par exemple, le français et l'anglais utilisent des jeux différents.

On représente un jeu de tuiles par la structure `Set` ci-dessous. Le champ `length` est le nombre de tuiles effectivement stockées, celles-ci se trouvant dans les `length` premières cases de son champ-tableau `tiles`. Par commodité, on se contentera d'ajouter dans la structure les tuiles de valeurs strictement supérieures à 1. Une tuile absente de la structure sera donc réputée avoir la valeur 1 lorsqu'elle est non-blanche.

```
#define SET_CAPACITY 26
typedef struct Set {
  Tile tiles [SET_CAPACITY];
  int length;
} Set;
```

Question 2. Implémenter la fonction `Set_InitEmpty()` qui initialise à vide un set passé par adresse. Implémenter aussi le prédicat `Set_IsFull()` qui teste si un set a atteint sa capacité de stockage.

```
void Set_InitEmpty (Set * set);
bool Set_IsFull (Set const * set);
```

Question 3a. Implémenter la fonction `Set_FindTileWithLetter()` qui recherche dans un set la tuile portant une lettre donnée. Si la tuile est trouvée, la fonction retourne son adresse. Si la tuile n'est pas trouvée, la fonction retourne `NULL`.

```
Tile const * Set_FindTileWithLetter (Set const * set, char letter);
```

Question 3b. En déduire le prédicat `Set_ContainsLetter()` qui teste si un set contient une lettre donnée.

```
bool Set_ContainsLetter (Set const * set, char letter);
```

Question 4a. Implémenter la fonction `Set_AddTile()` qui ajoute une tuile (passée par valeur) à la fin d'un set (passé par adresse). La fonction vérifie préalablement avec deux `assert()` qu'il reste de la place dans le set et que le set ne contient pas déjà la lettre de la tuile ajoutée.

```
void Set_AddTile (Set * set, Tile tile);
```

Question 4b. En déduire la fonction `Set_AddGroup()` qui permet d'ajouter dans un set un groupe de plusieurs lettres `letters` de même valeur `value`. Son utilisation est illustrée ci-dessous par `Set_InitStandardFrench()` et `Set_InitStandardEnglish()` qui initialisent respectivement des sets standards français et anglais.

```
void Set_AddGroup (Set * set, int value, char const letters[]);
```

```
void Set_InitStandardFrench (Set * set) {
  Set_InitEmpty (set);
  Set_AddGroup (set,  2, "DMG");
  Set_AddGroup (set,  3, "BCP");
  Set_AddGroup (set,  4, "FHV");

  Set_AddGroup (set,  8, "JQ");
  Set_AddGroup (set, 10, "KWXYZ");
}
```

```
void Set_InitStandardEnglish (Set * set) {
  Set_InitEmpty (set);
  Set_AddGroup (set,  2, "DG");
  Set_AddGroup (set,  3, "BCMP");
  Set_AddGroup (set,  4, "FHVWY");
  Set_AddGroup (set,  5, "K");
  Set_AddGroup (set,  8, "JX");
  Set_AddGroup (set, 10, "QZ");
}
```

Question 5. Implémenter la fonction `Set_LetterValue()` retournant la valeur d'une lettre donnée pour un set donné. La fonction accepte les lettres sur tuiles blanches, qui sont de valeur nulle et représentées par les lettres minuscules, ce que l'on détectera préalablement par une clause de garde grâce au prédicat `Letter_IsBlank()` donné ci-dessous. On rappelle que tout autre lettre absente de la structure `Set` est réputée de valeur 1.

```
#include <ctype.h>

bool Letter_IsBlank (char letter) {
  return islower (letter);
}
```

```
int Set_LetterValue (Set const * set, char letter);
```

2 Bonus multiplicateur et empreinte de coup

On décide de représenter les bonus des cases par le type énumératif `Bonus` suivant :

```
typedef enum Bonus {
  BONUS_NONE,
  BONUS_LETTER_X2,
  BONUS_LETTER_X3,
  BONUS_WORD_X2,
  BONUS_WORD_X3
} Bonus;
```

Question 6a. À l'aide de `switch`, implémenter la fonction `Bonus_LetterMultiplier()` qui retourne le multiplicateur de valeur de lettre associé à un bonus (la fonction retourne 1, 2, ou 3).

```
int Bonus_LetterMultiplier (Bonus bonus);
```

Question 6b. À l'aide de `if`, implémenter la fonction analogue `Bonus_WordMultiplier()` qui retourne le multiplicateur de score de mot associé à un bonus (la fonction retourne 1, 2, ou 3).

```
int Bonus_WordMultiplier (Bonus bonus);
```

Par la suite, on appelle *empreinte de coup* (ou *footprint*) une lettre posée sur un bonus, et participant à la création d'un mot joué sur la grille. Le bonus est dit *actif* (ou *live*) si la lettre a été posée au moment où le mot est joué (et non à un tour précédent). On décide de représenter une empreinte par la structure `Footprint` suivante :

```
typedef struct Footprint {
  char letter;
  bool live;
  Bonus bonus;
} Footprint;
```

Remarque : On a volontairement omis la valeur de la lettre dans l'empreinte.

Question 7. Implémenter la fonction `Footprint_Make()` qui fabrique et retourne une empreinte à partir de ses trois arguments. Utiliser une approche différente de celle que vous avez utilisée à la question 1 (C89 vs C99).

```
Footprint Footprint_Make (char letter, bool live, Bonus bonus);
```

Dans les questions suivantes, on cherche à calculer, pour un set donné, le score d'un mot représenté par un tableau d'empreintes. Le but final est d'écrire la fonction `Footprint_WordScore()` dont l'utilisation est illustrée ci-dessous :

```
int main (void) {
  Set set;
  Set_InitStandardFrench (& set);

  int const length= 7;
  Footprint footprints[]= {
    Footprint_Make ('B', true , BONUS_WORD_X2),
    Footprint_Make ('R', true , BONUS_LETTER_X2),
    Footprint_Make ('I', false, BONUS_NONE),
    Footprint_Make ('C', true , BONUS_NONE),
    Footprint_Make ('O', false, BONUS_LETTER_X3),
    Footprint_Make ('L', true , BONUS_NONE),
    Footprint_Make ('e', true , BONUS_WORD_X3)
  };

  int score= Footprint_WordScore (footprints, length, & set);
  printf ("Word␣score␣is␣%d\n", score);
  return 0;
}
```

```
Word score is 66.
```

Question 8a. Implémenter la fonction `Footprint_EffectiveLetterMultiplier()` qui retourne le multiplicateur effectif de lettre d'une empreinte donnée, c'est-à-dire en prenant en compte si le bonus est actif ou non. On demande ici d'utiliser l'opérateur ternaire conditionnel.

```
int Footprint_EffectiveLetterMultiplier (Footprint f);
```

Question 8b. En déduire la fonction `Footprint_LetterScore()` qui calcule et retourne le score de la lettre de cette empreinte pour un set donné.

```
int Footprint_LetterScore (Footprint f, Set const * set);
```

Question 9. Implémenter la fonction `Footprint_LetterTotalScore()` qui calcule et retourne la somme des scores de lettres d'une série d'empreintes `fs` de longueur `length` pour un set donné. Les multiplicateurs de mot n'entrent pas encore en jeu ici, cette fonction retourne donc $3+1\times2+1+3+1+1+0=11$ dans notre exemple.

```
int Footprint_LetterTotalScore (Footprint const fs[], int length, Set const * set);
```

Question 10a. Implémenter la fonction `Footprint_EffectiveWordMultiplier()` qui retourne le multiplicateur effectif de mot pour un tableau d'empreintes de longueur `length`. Il s'agit du produit des multiplicateurs de mot qui sont actifs, c'est-à-dire $2\times3=6$ dans notre exemple.

```
int Footprint_EffectiveWordMultiplier (Footprint const fs[], int length);
```

Question 10b. En déduire la fonction `Footprint_WordScore()` qui calcule et retourne le score d'un mot représenté par un tableau d'empreintes de longueur `length` pour un set donné. Les multiplicateurs de mot entrent maintenant en jeu, cette fonction retourne donc $11\times6=66$ dans notre exemple.

```
int Footprint_WordScore (Footprint const fs[], int length, Set const * set);
```