Correction

1 Tuile et jeu de tuiles

Question 1.

```
Tile
Tile_Make (int value, char letter) { // with C89
  Tile tile;
  tile.value= value;
  tile.letter= letter;
  return tile;
}
```

Question 2.

```
void
Set_InitEmpty (Set * set) {
  set->length= 0;
}
```

```
bool
Set_IsFull (Set const * set) {
  return set->length == SET_CAPACITY;
}
```

Question 3.

```
Tile const *
Set_FindTileWithLetter (Set const * set, char letter) {
  for (int k= 0; k < set->length; k++) {
    if (set->tiles[k].letter == letter)
      return & set->tiles[k];
  }
  return NULL;
}
```

```
bool
Set_ContainsLetter (Set const * set, char letter) {
  return Set_FindTileWithLetter (set, letter) != NULL;
}
```

Question 4.

```
void
Set_AddTile (Set * set, Tile tile) {
  assert ( ! Set_IsFull (set));
  assert ( ! Set_ContainsLetter (set, tile.letter));
  set->tiles [set->length]= tile;
  set->length++;
}
```

```
void
Set_AddGroup (Set * set, int value, char const letters[]) {
  for (int k= 0; letters[k] != '\0'; k++) {
    Set_AddTile (set, Tile_Make (value, letters[k]));
  }
}
```

Question 5.

```
int
Set_LetterValue (Set const * set, char letter) {
  if (Letter_IsBlank (letter)) return 0;
  Tile const * tile= Set_FindTileWithLetter (set, letter);
  if (tile == NULL) return 1;
  return tile->value;
}
```

2 Bonus multiplicateur et empreinte de coup

Question 6.

```
int
Bonus_LetterMultiplier (Bonus bonus) { // with switch
  switch (bonus) {
  case BONUS_LETTER_X3: return 3;
  case BONUS_LETTER_X2: return 2;
  default:              return 1;
  }
}
```

```
int
Bonus_WordMultiplier (Bonus bonus) { // with if-ladder
  if      (bonus == BONUS_WORD_X3) return 3;
  else if (bonus == BONUS_WORD_X2) return 2;
  else                             return 1;
}
```

Question 7.

```
Footprint
Footprint_Make (char letter, bool live, Bonus bonus) { // with C99
  return (Footprint) { .letter= letter, .bonus= bonus, .live= live };
}
```

Questions 8.

```
int
Footprint_EffectiveLetterMultiplier (Footprint f) { // with ternary op
  return f.live ? Bonus_LetterMultiplier (f.bonus) : 1;
}
```

```
int
Footprint_LetterScore (Footprint f, Set const * set) {
  int value= Set_LetterValue (set, f.letter);
  int mult= Footprint_EffectiveLetterMultiplier (f);
  return value * mult;
}
```

Question 9.

```
int
Footprint_LetterTotalScore (Footprint const fs[], int length, Set const * set) {
  int score= 0;
  for (int k= 0; k < length; k++) {
    score+= Footprint_LetterScore (fs[k], set);
  }
  return score;
}
```

Questions 10.

```
int
Footprint_EffectiveWordMultiplier (Footprint const fs[], int length) {
  int word_mult= 1;
  for (int k= 0; k < length; k++) {
    if (fs[k].live)
      word_mult *= Bonus_WordMultiplier (fs[k].bonus);
  }
  return word_mult;
}
```

```
int
Footprint_WordScore (Footprint const fs[], int length, Set const * set) {
  int total= Footprint_LetterTotalScore (fs, length, set);
  int mult = Footprint_EffectiveWordMultiplier (fs, length);
  return total * mult;
}
```