Retranscriptions algorithmiques du TP 1 : Justification d'une ligne

1. Supprimer les espaces avant la virgule

En supposant que l'utilisateur ne mettent pas plusieurs virgules l'une à la suite de l'autre (voir les solutions alternatives au besoin), on peut avoir l'idée de décaler la virgule de la droite vers la gauche jusqu'à voir une lettre.

```
@Pour i: int <- line.longueur -1, i > 0, i <- i -1
@DebutBloc
    j: int <- i -1
    car: char <- line[i]
    @Si car = ',' && line[j] = ' '
    @DebutBloc
        line[j] <- car
        line[i] <- ' '
    @FinBloc
@FinBloc
```

```
for (int i = line.length -1; i > 0; i = i -1)
{
    int j = i -1;
    char car = line[i];
    if (car == ',' && line[j] == ' ')
    {
        line[j] = car;
        line[i] = ' ';
    }
}
```

2. Positionner les marges

Positionner les marges (d'un point de vue algorithmique) n'est qu'une boucle pour répéter l'écriture d'un espace. On fait usage de l'accès direct pour modifer la chaine.

```
resultat: char[] <- new char[80]
@Pour i: int <- 0, i < mg, i <- i +1
@DebutBloc
    resultat[i] <- ' '
@FinBloc

@Pour i: int <- resultat.longueur - md, i < resultat.longueur, i <- i + 1
@DebutBloc
    resultat[i] <- ' '
@FinBloc
```

```
 char[] resultat = new char[80];
for (int i = 0; i < mg; i = i +1)
{
    resultat[i] = ' ';
}

for (int i = resultat.length - md; i < resultat.length; i = i + 1)
{
    resultat[i] = ' ';
}
```

3. Calculer le nombre de vides à remplir

On calcule le nombre de vides qui restent après avoir mis les marges.

```
 nbVides: int <- resultat.longueur - md - mg
```

```
 int nbVides = resultat.length - md - mg;
```

4. Compter le nombre de caractères dans line

Compter le nombre de caractères nous servira à déduire les espaces à distribuer afin de justifier notre ligne.

```
 int nbCar <- 0
@Pour i: int <- 0, i < line.longueur, i <- i +1
@DebutBloc
    car: char <- line[i]
    @Si car != ' '
    @DebutBloc
        nbCar <- nbCar +1
    @FinBloc
@FinBloc
```

```
 int nbCar = 0;
for (int i = 0; i < line.length; i = i +1)
{
    char car = line[i];
    if (car != ' ')
    {
        nbCar = nbCar +1;
    }
}
```

5. Compter le nombre de mots

Compter les nombres de mots permet de savoir combien de "blocs d'espaces" seront nécessaires. En effet, si on a trouvé 5 mots, il faut mettre un bloc d'espace entre chaque mot, donc 4 blocs d'espaces.

```
nbMots: int <- 0
last: char <- ' '
@Pour i: int <- 0, i < line.longueur, i <- i +1
@DebutBloc
    car: char <- line[i]
    @Si car != ' ' && last = ' '
    @DebutBloc
        nbMots <- nbMots + 1
    @FinBloc
    last <- car
@FinBloc
```

```
int nbMots = 0;
char last = ' ';
for (int i = 0; i < line.length; i = i +1)
{
    char car = line[i];
    if (car != ' ' && last == ' ')
    {
        nbMots = nbMots + 1;
    }
    last = car;
}
```

6. Calculer le nombre de blocs, leur taille et le nombre d'espaces à répartir

Avec ces informations, on peut donc en déduire les grandeurs suivantes : - le nombre de blocs d'espaces qui seront mis en chaque mot - la taille d'un bloc d'espace (le nombre d'espace) - le nombre d'espace restant à distribuer (s'il y 3 mots, donc 2 blocs d'espace et qu'il y ait 5 vides à distribuer, il y'aura forcément un bloc plus grand que l'autre, car il y aurau 1 espace restant à distribuer)

```
nbBlocs: int <- nbMots -1
tailleBloc: int <- nbVides - nbCar / nbBlocs
nbEspaceARepartir: int <- nbVides - nbCar modulo nbBlocs
```

```
int nbBlocs = nbMots -1;
int tailleBloc = (nbVides - nbCar) / nbBlocs;
int nbEspaceARepartir = (nbVides - nbCar) % nbBlocs;
```

7. Remplir le tableau résultat

Il faut écrire les mots, les blocs espace en n'oubliant pas de distribuer les espaces restants.

```
 last <- ' '
j: int <- mg
@Pour i: int <- 0, i < line.longueur, i <- i +1
@DebutBloc
    @Si line[i] != ' '
    @DebutBloc
        resultat[j] <- line[i]
        j <- j +1
    @FinBloc
    else
    @DebutBloc
        @Si last != ' ' && j < resultat.longueur -md
        @DebutBloc
            @Pour int k <- 0, k < tailleBloc, k <- k +1
            @DebutBloc
                resultat[j] <- ' '
                j <- j +1
            @FinBloc
            @Si nbEspaceARepartir > 0
            @DebutBloc
                resultat[j] <- ' '
                j <- j +1
                nbEspaceARepartir <- nbEspaceARepartir -1
            @FinBloc
        @FinBloc
    @FinBloc
    last <- line[i]
@FinBloc
```

```
 last = ' ';
int j = mg;
for (int i = 0; i < line.length; i = i +1)
{
    if (line[i] != ' ')
    {
        resultat[j] = line[i];
        j = j +1;
    }
    else
    {
        if (last != ' ' && j < resultat.length -md)
        {
            for (int k = 0; k < tailleBloc; k = k +1)
            {
                resultat[j] = ' ';
                j = j +1;
            }
            if (nbEspaceARepartir > 0)
            {
                resultat[j] = ' ';
                j = j +1;
                nbEspaceARepartir = nbEspaceARepartir -1;
            }
        }
    }
    last = line[i];
}
```

8. Retourner le résultat

Oui, il faut retourner la nouvelle données ! :)

```
@Résultat: resultat
```

```
return resultat;
```