Retranscriptions algorithmiques du TP 2 : Compactage

Comptacter

On cherche à calculer la taille totale de notre résultat. A chaque fois qu'on trouve une lettre différente, le résultat est censé grandir de 2 (ben oui haha, il faut 2 caractères à chque fois : 1 caractère pour la lettre et 1 caractètre pour le compte de la lettre). Conformément à la règle de l'énoncé, on en déduit qu'il est impossible qu'une lettre apparaissent plus de 9 fois.

```
 taille: nombre <- 2
@Pour i: nombre <- 0, i < E.longueur -1, i <- i +1
@DebutBloc
    @Si E[i] != E[i +1]
    @DebutBloc
        taille <- taille + 2
    @FinBloc
@FinBloc
```

```
 int taille = 2;
for (int i = 0; i < E.length -1; ++i)
{
    if (E[i] != E[i +1])
    {
        taille = taille + 2;
    }
}
```

On crée la chaine qui nous serviera de résultat.

```
 S: lettre[] <- créer char[taille]
```

```
 char[] S = new char[taille];
```

On parcourt la chaîne en entrée pour déduire le compte de chacun des caractères. Comme vu ensemble, chaque caractère a une position dans la table ASCII. Ainsi, on peut facilement transformer le glyphe « 1 » en sa quantité associée. Dans cet algorithme, on ajoute un à chaque fois qu'on voit une lettre identique.

```
 j: nombre <- 0
S[j] <- E[0]
j <- j + 1
S[j] <- '1'

@Pour i: nombre <- 1, i < E.longueur, i <- i +1
    @DebutBloc
    @Si E[i] != E[i -1]
    @DebutBloc
        j <- j +1
        S[j] <- E[i]
        j <- j +1
        S[j] <- '1'
    @FinBloc else @DebutBloc
        S[j] <- char S[j] +1
    @FinBloc
@FinBloc
```

```
 int j = 0;
S[j] = E[0];
j = j + 1;
S[j] = '1';

for (int i = 1; i < E.length; ++i) {
    if (E[i] != E[i -1]) {
        ++j;
        S[j] = E[i];
        ++j;
        S[j] = '1';
    } else {
        S[j] = (char) (S[j] +1);
    }
}
```

On n'oublie pas de retourner notre petit résultat ! :)

```
@Résultat: S
```

```
return S;
```

Décomptacter

Pour calculer la longueur du résultat final, on parcourt les positions 2 par 2. En faisant la somme de tous les comptes dans une chaîne qui a été compactée, on obtient la longueur du résultat final.

```
taille: nombre <- 0
@Pour i: nombre <- 1, i < S.longueur, i <- i +2
@DebutBloc
    somme: nombre <- S[i] - 48
    taille <- taille + somme
@FinBloc
```

```
int taille = 0;
for (int i = 1; i < S.length; i = i +2)
{
    int somme = S[i] - 48;
    taille = taille + somme;
}
```

On peut créer le tableau résultat final.

```
F: lettre[] <- créer char[taille]
```

```
char[] F = new char[taille];
```

On remplit la chaîne du résultat final en inscrivant la lettre autant de fois que le compte le permet. Ici, il faut donc 2 boucles, une première boucle pour parcourir le résultat final et une 2nde boucle pour pouvoir répéter l'inscription du même caractère, tant que le compte le permet.

```
k: nombre <- 0
@Pour i: nombre <- 1, i < S.longueur, i <- i +2
    @DebutBloc
    @Pour j: nombre <- 0, j < S[i] - 48, j <- j +1
    @DebutBloc
        F[k] <- S[i -1]
        k <- k +1
    @FinBloc
@FinBloc
```

```
int k = 0;
for (int i = 1; i < S.length; i = i +2) {
    for (int j = 0; j < S[i] - 48; ++j) {
        F[k] = S[i -1];
        ++k;
    }
}
```

Et on n'oublie pas finalement de retourner notre résultat !

```
@Résultat: F
```

```
return F;
```