Yann.Secq@univ-lille.fr
Fabien.Delecroix@univ-lille.fr

TP n°5

Objectifs: Écrire des fonctions utilisant des boucles à compteur et des boucles à évènement.

Attention : dans vos fonctions, le mot-clé `return` ne doit apparaître qu'une fois au maximum et à la fin de la fonction (et jamais pour les procédures !).

Exercice 1 : IUT [FUN-DEF, FUN-APPEL, REP-FOR-COUNT]

On souhaite réaliser un programme `DessineIUT` permettant de produire les affichages suivant :

taille : **5**

```
IIIII
  I
  I
  I
IIIII

U   U
U   U
U   U
U   U
UUUUU

TTTTT
  T
  T
  T
  T
```

taille : **3**

```
III
 I
III

U U
U U
UUU

TTT
 T
 T
```

Chacune des lettres I, U et T est ici dessinée dans un carré dont la taille est le nombre saisi par l'utilisateur.

1. Pour y parvenir, on va découper le programme en plusieurs procédures [1]. Vous devez tester séparément chacune des procédures. C'est à dire, après avoir écrit une procédure, appelez-la plusieurs fois dans la fonction principale `algorithm` avec des paramètres différents pour vous assurer qu'elle fait bien ce qu'on attend.
   Si on analyse chacune des lignes de chacune des lettres, on remarque qu'il existe en fait 3 types de lignes :
   (a) les lignes pleines, par exemple
   ```
   TTTTT
   ```
   On peut ici reprendre la procédure dessineLigne vue en TD :
   ```
   1  void dessineLigne(int n, char c) {
   2    for (int i = 0; i < n; i = i + 1) {
   3      print(c);
   4    }
   5  }
   ```
   (b) les lignes avec un caractère au milieu du carré, par exemple
   ```
     I
   ```
   Réalisez une procédure `void dessineMilieu(int n, char c)` qui dessine le caractère reçu `c` au milieu d'une ligne de taille `n`. On peut remarquer que dans ce cas, il n'est pas nécessaire d'afficher les espaces en fin de ligne.
   (c) les lignes avec des caractères aux extrêmités, par exemple

1. rappel : ce sont des fonctions qui ne retournent rien, dont le type de retour est `void`, comme `algorithm`

```
U   U
```

Réalisez la procédure `dessineExtrémités` qui corresponde.

2. Maintenant qu'on a nos briques de base, on va pouvoir dessiner chacune de nos lettres en les assemblant. Avant cela, définissez une procédure `println()` qui effectue juste un retour à la ligne, puis créez trois procédures, `dessinerI`, `dessinerU`, `dessinerT`, paramétrées par un caractère et une taille.
3. Il ne nous reste plus qu'à remplir notre `void algorithm()` en récupérant auprès de l'utilisateur la taille souhaitée et à appeler nos procédures de manière à obtenir l'affichage successif des 3 lettres I, U et T en utilisant le caractère correspondant à la lettre qu'on dessine ('I' pour dessiner le I, etc.).
4. N'oubliez pas de lancer les tests une fois que l'affichage correspond à ce qui est indiqué au début de l'exercice.

Exercice 2 : Un programme de saison [FUN-DEF, FUN-APPEL, ALT-COMB, EXPR-BOOL]

Dans cet exercice, on souhaite créer le programme `Saisons` définissant quelques fonctions utiles pour travailler avec des dates.

1. Commençons par la fonction `saisonMeteorologique` qui détermine la saison à partir du numéro du mois. Rappelons qu'en France métropolitaine le printemps météorologique commence le 1 mars, l'été météorologique commence le 1 juin, etc.

   Copiez le programme qui suit et complétez sa fonction de signture **`String saisonMeteorologique (int mois)`** de manière à ce qu'il donne l'exécution attendue montrée plus bas.

```
class Saisons extends Program {

  String saisonMeteorologique (int mois) {
      ...
  }

  void algorithm () {
    for (int m=0;m<=13;m=m+1){
      println("mois␣" + m + "␣:␣" + saisonMeteorologique(m));
    }
  }
}
```

```
mois 0 : erreur
mois 1 : hiver
mois 2 : hiver
mois 3 : printemps
mois 4 : printemps
mois 5 : printemps
mois 6 : été
mois 7 : été
mois 8 : été
mois 9 : automne
mois 10 : automne
mois 11 : automne
mois 12 : hiver
mois 13 : erreur
```

2. Continuons avec la fonction `nombreJoursMois`. Elle retourne le nombre de jours en fonction du numéro du mois. Un peu de poésie pour vous aider :

   « Trente jours ont novembre,
   Avril, juin et septembre.
   De vingt-huit, il y en a un,
   Tous les autres ont trente et un. »

   Pour simplifier, nous ignorons pour le moment les années bissextiles et considérons donc que le mois de février a 28 jours.

   Ajouter à votre programme une fonction de signature **`int nombreJoursMois(int numeroMois)`** et modifier l'algorithme principal pour qu'il affiche en plus le bon nombre de jours pour chaque mois valide et qui compte en plus le nombre de jours total dans une année (non bissextile). Par exemple

```
mois 0 : erreur, 0 jours
mois 1 : hiver, 31 jours
mois 2 : hiver, 28 jours
...
mois 9 : automne, 30 jours
...
nombre de jours total : 365
```

3. Dans le calendrier, on utilise plutôt la saison astronomique que météorologique. Le printemps astronomique commence aux alentours du 21 mars (cette date varie d'une année à l'autre), l'été astronomique commence aux alentours du 21 juin, etc. Dans l'exercice nous prendrons les dates 21/03, 21/06, 21/09 et 21/12 comme débuts des différentes saisons. Remarquez-vous un lien entre saison astronomique et saison météorologique ? Si oui, vous en servir peut rendre votre code moins redondant, plus clair et plus concis. Ajouter une fonction de signature `String saisonAstronomique (int jour, int mois)` qui calcule la saison astronomique à partir du numéro du mois et du jour.
4. Modifiez votre fonction principale afin qu'elle affiche pour chaque jour de l'année sa saison astronomique sous la forme suivante :

```
1/1 : hiver
2/1 : hiver
...
20/6 : printemps
...
31/12 : hiver
```

Vérifiez que la saison affichée est la bonne notamment pour les dates suivantes :

Date	Saison Astronomique
11/11	automne
20/3	hiver
31/3	printemps
20/6	printemps
20/9	été
20/12	automne
21/12	hiver

Exercice 3 : Détecter la présence d'une sous-chaine [REP-WHILE, EXPR-STR, FUN-DEF, FUN-APPEL]

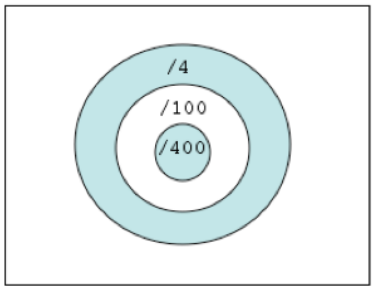
Écrire le programme `SousChaine` définissant la fonction `boolean contient(String chaine, String motif)` qui prend en paramètre deux chaines de caractères et détermine si la deuxième est sous-chaîne de la première. Vérifiez votre fonction en l'appelant avec les exemples données ci-dessous dans `algorithm()`.

Chaîne	Motif recherché	Sortie
"blablabla"	"lab"	trouvé
"blablabla"	"bal"	pas trouvé
"abc"	""	trouvé
""	""	trouvé
"abcdef"	"defg"	pas trouvé
"abc"	"abcdef"	pas trouvé

Exercice 4 : Années bissextiles [ALT-COMB, FUN-DEF]

Pour déterminer si une année est bissextile, il faut appliquer les règles suivantes :
— Les années divisibles par 4 sont bissextiles,
— Exception : les années divisibles par 100 ne sont pas bissextiles,
— Exception à l'exception : les années divisibles par 400 sont bissextiles.
— Sauf application des règles ci-dessus, toute année n'est pas bissextile.

Il est possible de représenter graphiquement ces contraintes avec en bleu les zones pour lesquelles une année est bissextile et en blanc pour celles qui ne le sont pas:



1. Concevez le programme `Bissextile` contenant une fonction booléenne, `boolean estBissextile(int annee)`, permettant de déterminer si une année est bissextile.
2. Optionnel : Implémentez quatre résolutions différentes de la fonction précédente en respectant les contraintes suivantes :
   — avec quatre alternatives indépendantes (`estBissextile1`),
   — avec une seule alternative constituée d'une expression conditionnelle complexe (`estBissextile2`),
   — avec une cascade de si alors sinon en allant des conditions les plus spécifiques au plus générales (`estBissextile3`),
   — avec une cascade de si alors sinon en allant des conditions les plus générales au plus spécifiques (`estBissextile4`).
3. Pour vérifier le bon fonctionnement de votre fonction, ajoutez la fonction `algorithm()` qui affiche les 33 dernières années bissextiles en partant de 2022. Vous devriez obtenir l'affichage suivant :

```
2020 2016 2012 2008 2004 2000 1996 1992 1988 1984 1980 1976 1972
1968 1964 1960 1956 1952 1948 1944 1940 1936 1932 1928 1924 1920
1916 1912 1908 1904 1896 1892 1888
```

;

Exercice 5 : Bissextilité

1. Maintenant que vous disposez d'une fonction qui permet de tester si une année est bissextile, en réutilisant votre fonction `nombreJoursMois`, proposez-en une autre avec le même nom mais en ajoutant l'année en paramètre supplémentaire afin de calculer le bon nombre de jours pour les mois de février.
2. Écrivez désormais le programme `Bissextilite` qui affiche tous les jours d'une année saisie par l'utilisateur ou une utilisatrice.

Prolongements

Exercice 6 : Plus de dessins ASCII

Enrichissez le programme `DessinsASCII` de l'exercice 1 pour qu'en plus d'écrire `IUT`, il écrive ensuite `LILLE` et `ASCII` sur le même principe.

Exercice 7 : Un programme qui fera date

On souhaite écrire le programme `FaireDate` qui propose les fonctionnalités suivantes :

1. En repartant des fonctions produites durant les précédents exercices, affichez le calendrier d'une année donnée avec les jours de la semaine sous la forme LMMJVSD. Voici un exemple d'exécution attendu :

```
Année souhaitée : 2022
1er jour de l'année L(1), M(2), M(3), J(4), V(5), S(6) ou D(7) ? 6
Mois 1
S 1
D 2
L 3
...
Mois 10
S 1
D 2
L 3
...
V 30 S 31
```

2. Créez une nouvelle fonction proposant presque la même fonctionnalité, mais maintenant sans avoir besoin de demander le 1er jour de l'année.
3. Écrivez une fonction qui à partir d'une date donnée et d'un nombre de jours souhaité (positif ou négatif) indique la date atteinte.