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

TP n°4

Objectifs: Écrire des programmes utilisant une boucle à compteur ou une boucle à événement.

Exercice 1 : Ticket de caisse [REP-WHILE-SAISIE, REP-ACC-NUM]

Dans cet exercice, un responsable de magasin vous mandate pour concevoir le programme `TicketDeCaisse` qui demande à l'utilisateur de saisir des entiers, dont il faut calculer la somme. La saisie doit s'arrêter lorsque l'utilisateur saisit le nombre 0. Voici un exemple d'exécution attendue :

```
5
10
15
30
0
Total=60
```

Exercice 2 : Prise de température [REP-WHILE-SAISIE]

Dans cet exercice, on demande à l'utilisateur de saisir des températures afin de lui afficher la valeur maximale. La saisie s'interrompt lorsque l'utilisateur entrera -273, qui correspond au zéro absolu (cette température n'étant pas à prendre en compte dans la liste). On supposera qu'il rentre toujours au moins un entier avant. Avant d'écrire l'algorithme, répondez aux questions suivantes, qui devraient vous aider à décomposer la résolution de ce problème.

1. Si le début de la liste est 5 -9 23 98 58 -11 89 8... que pouvez vous dire de la valeur maximale qui sera calculée ? Si l'on suppose que l'on ajoute 6 à la liste, quelle est la valeur maximale ? De même, que se passe-t-il si l'on ajoute 124 à la liste ?
2. Que pouvez-vous déduire des observations précédentes en ce qui concerne le nombre de variables à utiliser pour calculer le maximum ?
3. Décrivez le traitement qui doit être effectué à chaque nouvel entier saisi pour déterminer la valeur maximale de la liste.
4. Quand faut-t-il s'arrêter de saisir des entiers ?
5. Ecrivez maintenant le programme `Temperature` qui permet de saisir la suite d'entiers comme décrit ci-dessus, et affiche le plus grand parmi ces entiers. Voici un exemple d'exécution:

```
Saisir une suite de valeurs entières terminée par -273.
2
7
5
-273
Le maximum est 7.
```

6. Et en voici un second :

```
-2
-7
-5
-273
Le maximum est -2.
```

7. Modifiez votre algorithme pour calculer en plus du maximum, le miminum et la moyenne des nombres saisis (sans prendre en compte la valeur -273). Tester à nouveau avec les deux suites ci-dessus.

Exercice 3 : Diviseurs [REP-FOR-COUNT, REP-FILTRE, REP-ACC-NUM]

Le but de cet exercice est d'écrire le programme `Diviseurs` qui calcule la liste des diviseurs d'un entier donné. Avant de définir cet algorithme, répondez aux questions suivantes:

1. Donnez la liste des diviseurs de 10.
2. Donnez la liste des diviseurs de 36.
3. Comment avez-vous procédé ? Avez-vous répété une série d'actions? Laquelle ?
4. Quand vous êtes-vous arrêté ?
5. Ecrivez maintenant le programme `Diviseurs` qui affiche la liste des diviseurs d'un nombre donné.

Saisie	Affichage attendu
9	`Diviseurs : 9 3 1`
10	`Diviseurs : 10 5 2 1`
2	`Diviseurs : 2 1`

6. Un nombre n est dit parfait si la somme de ses diviseurs vaut $2 \times n$. Par exemple, 6 est un nombre parfait car $2 \times 6 = 12 = 1 + 2 + 3 + 6$. Modifiez votre programme de manière à ce qu'il affiche en plus si le nombre est parfait ou non.

```
6
Diviseurs : 6 3 2 1
Nombre parfait !
```

```
7
Diviseurs : 7 1
```

Exercice 4 : Ni oui ni non [REP-WHILE-SAISIE]

Écrire le programme `NiOuiNiNon` qui saisit des mots auprès de l'utilisateur et s'arrête quand celui-ci a saisi le mot oui ou le mot non. La fonction affiche alors le texte "Perdu !".

Optionnel: Améliorez l'algorithme pour qu'il reconnaisse les mots oui et non même s'ils sont écrits en majuscules, ou en combinaison de majuscules ou minuscules. Par exemple, ces mots seront reconnus comme oui ou non: `OUI`, `Non`, `ouI`, ... Vous pouvez vous aider de la documentation des fonctions sur les chaînes des caractères

Exercice 5 : Deviner un nombre [REP-WHILE]

Le but de cet exercice est de crééer le programme `DevinerNombre` qui permet de deviner un nombre entre 1 et 100 avec le plus petit nombre de tentatives, en utilisant des indications "plus petit" et "plus grand".

L'utilisateur a un nombre en tête que l'algorithme doit deviner en faisant des propositions. À chaque proposition, l'utilisateur indique plus grand, plus petit, ou égal.

Voici un exemple d'exécution, en supposant que le nombre à deviner est 28.

```
Est-ce que le nombre est 50 ?
-
Est-ce que le nombre est 25 ?
+
Est-ce que le nombre est 37 ?
-
Est-ce que le nombre est 31 ?
-
Est-ce que le nombre est 28 ?
=
```

La manière la plus efficace pour trouver le nombre est en utilisant une dichotomie. Le principe est le suivant. L'algorithme maintient l'intervalle dans lequel se trouve le nombre. Au début cet intervalle est $[0, 100]$. La proposition est toujours le milieu de l'intervalle. L'intervalle est mis à jour en fonction de la réponse de l'utilisateur.

Pour l'exemple ci-dessus, voici les valeurs successives de l'intervalle et la proposition de l'algorithme.

Étape	1	2	3	4	5
Intervalle	$[1, 100]$	$[1, 49]$	$[26, 49]$	$[26, 36]$	$[26, 30]$
Proposition	50	25	37	31	28

Exercice 6 : Plus grand commun diviseur [REP-WHILE]

L'algorithme d'Euclide permet de calculer le Plus Grand Commun Diviseur (PGCD) de deux entiers a et b à l'aide de divisions successives, comme suit.

On suppose que $a \geq b$ et que a et b sont tous deux positifs ou nuls. À chaque itération, on divise a par b, soit r le reste de cette division. Si r est 0, alors l'itération s'arrête et b contient le pgcd de a et b. Sinon, on continue l'itération en donnant à a la valeur de b et à b la valeur de r.

Le tableau ci-dessous présente les valeurs successives de a, b et r lors du calcul du pgcd de 80 et 62. La dernière valeur de b (sous-lignée) est la valeur du pgcd.

a	b	$r = a\%b$
80	62	18
62	18	8
18	8	2
8	<u>2</u>	0

1. Écrire le programme `PGCD` qui calcule le pgcd de deux nombres saisis, en supposant que ces nombres sont positifs. Voici deux exemples d'exécution :

```
80
70
Le pgcd est 10
```

```
5
3
Le pgcd est 1
```

2. Modifier le programme pour qu'il fonctionne aussi avec des nombres négatifs, sachant que le pgcd de a et b est égal au pgcd des valeurs absolues de a et b. Tester votre programme `pgcd` avec les valeurs ci-dessous.

Saisies		Affichage attendu
`-80`	`62`	`Le pgcd est 2`
`-5`	`-3`	`Le pgcd est 1`
`12`	`0`	`Le pgcd est 12`

Exercice 7 : La suite de Syracuse [REP-WHILE]

Rappelons la définition de la suite de Syracuse.

Prenez un entier positif ; s'il est pair, divisez-le par 2 ; s'il est impair, multipliez-le par 3 et ajoutez lui 1… Réitérez ce processus sur plusieurs exemples : que semble-t-il se passer ?

Partons de l'entier 7, et regardons la suite alors construite : 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1…. Cette suite devient cyclique, puisque l'obtention de la valeur 1 fait "boucler" indéfiniment l'algorithme.

On conjecture que l'on finit toujours par trouver la valeur 1 au fil des calculs quel que soit l'entier de départ… C'est la conjecture de Syracuse (encore appelée "problème $3n + 1$") ... qui attend toujours une preuve !

Reprenons l'exemple initial de l'entier 7. On appellera la suite (7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1) la trajectoire ou le vol de 7. Chaque entier de cette suite est une étape du vol, 52 est l'altitude maximale de la trajectoire. La durée d'un vol (16, ici) est le nombre d'étapes nécessaires avant l'apparition du premier 1 (s'il apparaît bien sûr !). La durée du vol en altitude est le nombre d'étapes entre le début du vol et le moment où le nombre courant passe sous la valeur de départ (11 dans notre exemple). Le facteur d'expansion est l'altitude maximale divisée par l'entier de départ (52/7 dans notre exemple).

Concevez le programme `Syracuse` permettant d'expérimenter la conjecture de Syracuse. Dans une première version de l'algorithme, ne calculez que la trajectoire de l'entier considéré. Enrichissez ensuite successivement votre algorithme pour calculer aussi l'altitude maximale, la durée de vol, le facteur d'expansion et finalement la durée de vol en altitude.

Par exemple, si l'utilisateur saisit **7**, le programme doit afficher

```
Trajectoire: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1,
Altitude max: 52
Durée de vol: 16
Durée de vol en altitude: 11
Facteur d'expansion: 7
```

Prolongements

Exercice 8 : The final countdown

À l'aide de la documentation de la fonction `getTime`, réalisez le programme `FinalCountdown` qui compte de 3 en 3 jusqu'à ce que 3ms se soient écoulées.

```
3
6
9
...
```

Lancez plusieurs fois votre programme, que constatez-vous ?