Dernier jour d'un mois/année [ss02]
Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 16 mai 2018

Table des matières

C - Dernier jour d'un mois/année (Solution)

Mots-Clés Algorithmes paramétrés ■
Utilise Fonction, Procédure de test ■
Requis Structures de base, Structures conditionnelles ■
Difficulté ●●○ (1 h) ■

Objectif
Cet exercice calcule le dernier jour d'un mois et année donnés.

1 Dernier jour d'une date / pgsjours

1.1 Énoncé

Objectif
Déterminer le dernier jour d'un mois et année donnés.

Résultat attendu
Voici un exemple du résultat attendu :

```
De quel mois s'agit-il? 2
De quelle année? 2000
==> En 2000 le dernier jour du mois 2 est 29
```

1.2 Fonction dernierJour0

Commençons par déterminer le dernier jour d'un numéro de mois donné **indépendamment** d'une année.

Écrivez le **profil** d'une fonction `dernierJour0(mm)` qui renvoie le dernier jour d'un numéro de mois `mm`. Le mois est donné sous la forme d'un entier (1 pour janvier...).

Solution Paramètres

Entrants : Un entier `mm` (numéro de mois)

Résultat de la fonction : Un entier (nombre de jours)

Nombre de jours
Les mois à 31 jours sont : janvier, mars, mai, juillet, aout, octobre, décembre ; les mois à 30 jours : avril, juin, septembre, octobre ; le cas de février : 28 jours ; et dans tous les autres cas : c'est 0. (image : `http:\www.maxicours.com`)

Il convient donc de déclarer une variable `rs` de type entier (type de la fonction) qui stockera le résultat, de la calculer en utilisant une structure `Selon` ou une structure `Si-Sinon-Si` (par regroupement de mois) puis de renvoyer sa valeur.

Écrivez son corps.

Validez votre fonction avec la solution.

Solution C @[sjours.c]

```c
int dernierJour0(int mm)
{
  int rs;
  switch(mm)
  {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
      rs = 31;
      break;
    case 4:
    case 6:
    case 9:
    case 11:
      rs = 30;
      break;
    case 2:
      rs = 28;
      break;
    default:
      rs = 0;
  }
  return rs;
}
```

1.3 Procédure de test

Écrivez une procédure `test_sjours0` qui saisit un numéro de mois (entier).
Affichez l'invite :

```
De quel mois s'agit-il?
```

Mémorisez le dernier jour du mois dans une variable (entier).

Affichez l'un des deux messages (où `[x]` désigne le contenu de `x`) :

```
==> Le dernier jour du mois [mois] est [njours]
==> Impossible ce mois n'existe pas
```

Écrivez un programme qui se contente d'appeler la procédure de test.

Testez. Exemples d'exécution :

```
De quel mois s'agit-il? 5
==> Le dernier jour du mois 5 est 31
```

```
De quel mois s'agit-il? 15
==> Impossible ce mois n'existe pas
```

Validez votre procédure avec la solution.

Solution C @[pgsjours.c]

```c
void test_sjours0()
{
  int mois;
  printf("De quel mois s'agit-il? ");
  scanf("%d",&mois);
  int njours = dernierJour0(mois);
  if (njours != 0)
  {
    printf("==> Le dernier jour du mois %d est %d\n",mois,njours);
  }
  else
  {
    printf("==> Impossible ce mois n'existe pas\n");
  }
}
```

1.4 Bissextilité en terme de divisibilité

Pour tenir compte du mouvement réel de la Terre, un Romain d'avant notre ère décréta qu'il y aurait des années bissextiles environ tous les quatre ans. Il fut décidé que les années de notre ère dont le numéro est multiple de 4 seraient bissextiles et que le jour supplémentaire serait le 29 février.

Mais cette correction est un peu trop forte, et un pape du nom de Grégoire décida que :

Propriété
Les années de siècles (c.-à-d. multiples de 100) ne sont pas bissextiles, sauf si elles sont multiples de 400.

Soient deux entiers n et d. Dire que d est multiple de n équivaut à dire que... (à vous de compléter) en terme de divisibilité.

Solution simple

Dire que d est multiple de n équivaut à n est divisible par d, donc que le reste de la division entière de n par d (c.-à-d. le modulo) est nul.

Traduisez l'énoncé en terme de divisibilité : « [...] années multiples de 4 sont bissextiles. [...] mais que les années de siècles (c.-à-d. multiples de 100) ne sont pas bissextiles, sauf si elles sont multiples de 400. »

Solution simple

L'énoncé dit que :

1. Les années divisibles par 4 sont bissextiles sauf si elles sont divisibles par 100.
2. Les années divisibles par 400 sont bissextiles.

Une année est donc *bissextile* si elle satisfait le premier **ou** le deuxième cas. Le premier cas est satisfait si l'année est divisible par 4 mais **non** divisible par 100.

1.5 Fonction divisible (test de divisibilité)

Écrivez le **profil** d'une fonction `divisible(n,d)` qui renvoie `Vrai` si un entier `n` est divisible par un entier `d`, `Faux` sinon.

Propriété

Un entier n est **divisible** par un entier d si (et seulement si) le reste de la division entière de n par d (c.-à-d. le modulo) est nul.

Écrivez le corps de la fonction.

Validez votre fonction avec la solution.

Solution C

```c
bool divisible(int n,int p)
{
  return n%p == 0;
}
```

1.6 Fonction bissextile

Définition

Une année postérieure à 1592 (début du calendrier grégorien) est **bissextile** si elle est divisible par 4 mais **non** divisible par 100, ou si elle est divisible par 400.

Écrivez une fonction `bissextile(an)` qui teste et renvoie `Vrai` si le millésime d'une année `an` (entier), supposée postérieure à 1592, est bissextile, `Faux` sinon.

Aide simple

Écrivez une expression booléenne dépendant de `an` qui traduit l'énoncé.
Attention aux parenthèses de l'expression.

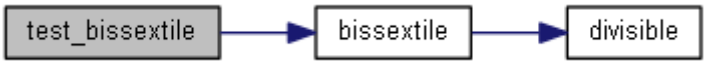
Validez votre fonction avec la solution.

Solution C @[sjours.c]

```
bool bissextile(int an)
{
  return (divisible(an,4) && !divisible(an,100)) || divisible(an,400);
}
```

1.7 Procédure de test

Écrivez une procédure `test_bissextile` qui saisit le millésime d'une année (entier) puis affiche si elle est ou non bissextile.



Testez. Exemples d'exécution :

```
Annee? 2004
==> Vrai
```

```
Annee? 1900
==> Faux
```

```
Annee? 2000
==> Vrai
```

Validez votre procédure avec la solution.

Solution C @[pgsjours.c]

1.8 Fonction dernierJour

Ce problème revisite la fonction `dernierJour0` de sorte à tenir compte des années bissextiles.

Copiez/collez la fonction `dernierJour0` en la fonction `dernierJour(mm,an)` qui renvoie le dernier jour d'un numéro de mois `mm` (entier) et année `an` (entier) donnés.

Modifiez le cas de février (cas 2) :

- Si l'année est bissextile, mettez 29 dans le résultat `rs`, 28 sinon.

Validez votre fonction avec la solution.

Solution C @[sjours.c]

```c
int dernierJour(int mm, int an)
{
  int rs;
  switch(mm)
  {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
      rs = 31;
      break;
    case 4:
    case 6:
    case 9:
    case 11:
      rs = 30;
      break;
    case 2:
      rs = (bissextile(an) ? 29 : 28);
      break;
    default:
      rs = 0;
  }
  return rs;
}
```

1.9 Procédure de test revisitée

Copiez/collez la procédure de test `test_sjours0` en la procédure `test_sjours` puis complétez-la de sorte qu'elle saisit également le millésime d'une année (entier).
Affichez l'invite :

```
De quelle année?
```

Modifiez le message en (où `[x]` désigne le contenu de `x`) :

```
En [annee] le dernier jour du mois [mois] est [njours]
```

Lancez-le avec les exemples d'exécution.

```
De quel mois s'agit-il? 5
De quelle année? 2010
==> En 2010 le dernier jour du mois 5 est 31
```

```
De quel mois s'agit-il? 15
De quelle année? 2010
==> Impossible ce mois n'existe pas
```

```
De quel mois s'agit-il? 2
De quelle année? 2000
==> En 2000 le dernier jour du mois 2 est 29
```

Validez votre procédure avec la solution.

Solution C @[pgsjours.c]

```c
void test_sjours()
{
  int mois;
  printf("De quel mois s'agit-il? ");
  scanf("%d",&mois);
  int annee;
  printf("Annee? ");
  scanf("%d",&annee);
  int njours = dernierJour(mois, annee);
  if (njours != 0)
  {
    printf("==> En %d le dernier jour du mois %d est %d\n",annee,mois,njours);
  }
  else
  {
    printf("==> Impossible ce mois n'existe pas\n");
  }
}
```

2 Compléments

2.1 Validité d'une date

Une date est mémorisée dans trois variables de type entier, une pour le numéro de jour, une pour le numéro du mois et une pour le millésime de l'année. Par exemple, la date

du 12 février 2013 est composée de trois entiers $(12, 2, 2013)$.

Définition
Une **date** (j, m, a) est **valide** si :

- L'année a est positive.
- Le mois m est compris entre 1 et 12.
- Le jour j est compris entre 1 et le nombre de jours du mois m de l'année a.

Écrivez une fonction `datum(jr,mm,an)` qui teste et renvoie `Vrai` si et seulement si le triplet d'entiers `(jr,mm,an)` représente une date, `Faux` sinon.

Validez votre fonction avec la solution.

Solution C @[sjours.c]

```
bool datum(int jr,int mm,int an)
{
  return (an>=0) && ((1<=mm) && (mm<=12)) && ((1<=jr) && (jr<=dernierJour(mm,an)));
}
```

Écrivez une procédure `test_saisir` qui demande et saisit une date jusqu'à ce qu'elle soit valide.

Testez.

Validez votre procédure avec la solution.

Solution C @[pgsjours.c]

3 Que retenir de cet exercice?

Un algorithme peut appeler une fonction `f1`, qui appelle une fonction `f2`, qui appelle à son tour une fonction `f3` et ainsi de suite. Il n'y a pas de limite théorique à cette chaîne d'appels de fonctions.

L'usage de fonctions permet aussi d'organiser les jeux d'essais de manière plus systématique. On teste d'abord les fonctions, en commençant par celles qui 'en appellent pas d'autres. Lorsqu'on est sûr de leur justesse, il n'est plus nécessaire de les tester en même temps que l'unité appelante.

C'est au programme appelant de vérifier qu'une fonction est appelée dans des conditions qui respectent sa spécification.

Les jeux d'essais ne sont pas réservés aux algorithmes. Il faut aussi tester les fonctions. Celles qui sont fournies dans des bibliothèques sont censées être justes.

Toutes les fonctions booléennes, nommées aussi *prédicats*, peuvent s'écrire de deux manières :

- En affectant explicitement la valeur booléenne `Vrai` ou `Faux` à une variable booléenne locale renvoyée par la fonction.
- En renvoyant directement une expression booléenne évaluée dans la fonction.

Une *variable locale* est déclarée à l'intérieur d'une fonction et n'est utilisable que par celle-ci. Elle est inconnue à l'extérieur de la fonction et disparaît de la mémoire dès que l'exécution de la fonction est terminée.

Il est intéressant d'utiliser les fonctions même si leur définition prend plus de place que les expressions qu'elles renferment. En effet :

- Si leur nom est bien choisi, il exprime en clair leurs propriétés, rendant ainsi l'algorithme immédiatement compréhensible sans qu'il soit nécessaire de connaître les détails des corps des fonctions.
- La programmation de ces fonctions peut être déléguée à des experts du domaine permettant ainsi un partage du travail dans une équipe de développement.

4 Références générales

Comprend [Routier-KM1 :c9 :et1] ■