

Date du lendemain [dt08] - Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel

sciel

algotprog

UNIVERSITÉ
HAUTE-ALSACE

Version 21 mai 2018

Table des matières

1	Énoncé	2
2	Algorithmique, Programmation	2
2.1	Analyse	2
2.2	Fonction bissextile (test de bissextilité)	2
2.3	Fonction dernierJour (dernier jour d'un mois et année)	3
2.4	Test : Dernier jour d'un mois et année donnés	4
2.5	Type DateJMA	5
2.6	Procédure calculerDemainDT (lendemain d'une date)	6
2.7	Fonction saisirGregDT (saisie d'une date)	7
2.8	Procédure afficherDT (affichage d'une date)	8
2.9	Test : Lendemain d'une date	8
2.10	Application : Nombre de jours écoulés	9
3	Références générales	11

C++ - Date du lendemain (Solution)



Mots-Clés Dates et Heures ■

Requis Axiomatique impérative (sauf Fichiers) ■

Difficulté ●○○ (1 h 30) ■



Objectif

Cet exercice calcule le lendemain d'une date (jour, mois, année) puis calcule le nombre de jours écoulés entre deux dates.

1 Énoncé

Une date est mémorisée dans trois variables de type entier, une pour le numéro du jour, une pour le numéro de mois et une pour le millésime de l'année. Par exemple, la date du 12 février 2013 est composée des trois entiers (12, 2, 2013).

Objectif

Demander les trois entiers composant la date d'un jour puis calculer et afficher les trois entiers composant la date du lendemain. En application, calculer le nombre de jours écoulés entre deux dates.

2 Algorithmique, Programmation

2.1 Analyse

Ce problème se simplifie si on essaie de dégager des règles de changement de date sans s'attacher aux cas particuliers des jours et des mois précis. En effet, il apparaît qu'il y a trois types de changement de dates :

- **année suivante** si la date saisie est le dernier jour de décembre
- **mois suivant** si la date est celle du dernier jour d'un mois (sauf décembre)
- **jour suivant** dans les autres cas

Le point le plus délicat est celui du changement de mois (sauf décembre). Pour éviter de tester chacun des onze mois dans l'année, on aimerait trouver un critère plus général. Or ce critère existe si on est capable de calculer le *nombre de jours du mois* d'une date. Il suffit alors de comparer ce nombre avec la valeur saisie dans la variable représentant le jour. Si ces deux nombres sont égaux, alors il faut changer de mois.

Pour résoudre ce problème il faut donc gérer :

- Les années bissextiles pour le mois de février.
- Le nombre de jours dans le mois.
- Le changement de mois.
- Le changement d'année.

2.2 Fonction bissextile (test de bissextilité)



Définition

Soit une année postérieure à 1582 (début du calendrier grégorien). Elle est **bissextile** si et seulement si son millésime est :

- Divisible par 4 mais **non** divisible par 100.
- **Ou** divisible par 400.

Exemples

- 1986 : non (non divisible par 4)
- 1988 : oui (divisible par 4 et non divisible par 100)
- 1900 : non (divisible par 4 et par 100)
- 2000 : oui (divisible par 400)



Écrivez une fonction `bissextile(an)` qui teste et renvoie `Vrai` si le millésime d'une année `an` (entier supérieur à 1582) est bissextile, `Faux` sinon.

Solution Paramètres

Entrants : Un entier `an`

Résultat de la fonction : Un booléen

Solution simple

En tant que *prédicat* (fonction booléenne), le corps de la fonction est simplement constitué de l'**expression booléenne** qui traduit l'énoncé.



Validez votre fonction avec la solution.

Solution C++ @[UtilsDT.cpp]

```
/**
 * Prédicat d'année bissextile
 * @param[in] an - millésime d'une année > DEBUTGREGCALENDRIER=1582
 * @return vrai si an est bissextile
 */
bool bissextile(int an)
{
    return (an % 4 == 0 && !(an % 100 == 0)) || (an % 400 == 0);
}
```

2.3 Fonction dernierJour (dernier jour d'un mois et année)

**Propriété**

Pour les années postérieures à 1582 (année du calendrier Grégorien) :

- Les mois numéros 1, 3, 5, 7, 8, 10 et 12 ont 31 jours.
- Ceux de numéros 4, 6, 9 et 11 en ont 30.
- Le mois numéro 2 a 29 jours si l'année est bissextile, 28 dans le cas contraire.



Écrivez une fonction `dernierJour(mm,an)` qui calcule et renvoie le dernier jour d'un numéro de mois `mm` (entier compris entre 1 et 12) d'une année `an` (entier supérieur à 1582), selon l'algorithme suivant :

- Si le mois vaut 2 : le dernier jour est 28 (ou 29 si l'année est bissextile).
- Sinon si le mois est impair et ≤ 7 ou pair et > 7 : le dernier jour est 31.
- Sinon le dernier jour est 30.

Solution simple

On utilise une structure `Si`. Le test sur l'imparité se fait en examinant le reste de la division entière par 2 : s'il vaut 1, l'entier est impair.



Validez votre fonction avec la solution.

Solution C++

@[UtilsDT.cpp]

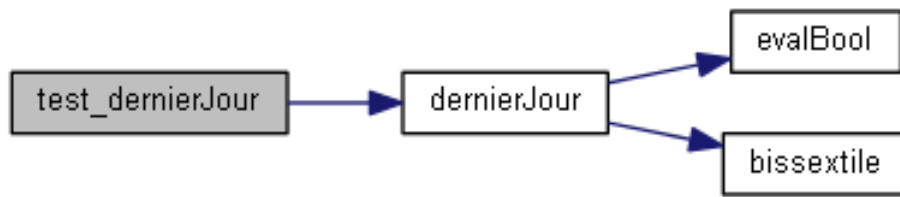
```
/**
 * Dernier jour d'un mois et année
 * @param[in] mm - un numéro de mois dans [1..12]
 * @param[in] an - millésime d'une année > DEBUTGREGCALENDRIER
 * @return le dernier jour de (mm,an)
 */
int dernierJour(int mm, int an)
{
    int rs;
    if (mm == 2)
    {
        rs = (bissextile(an) ? 29 : 28);
    }
    else if (mm <= 7)
    {
        rs = (mm % 2 == 1 ? 31 : 30);
    }
    else
    {
        rs = (mm % 2 == 0 ? 31 : 30);
    }
    return rs;
}
```

2.4 Test : Dernier jour d'un mois et année donnés



Écrivez une procédure `test_dernierJour` qui saisit un numéro de mois et le millésime d'une année puis calcule et affiche le dernier jour du mois dans l'année. Affichez les invites :

```
Numero de mois ([1..12])?
Millesime de l'annee (>1582)?
```



Testez. Exemples d'exécution :

Mois, annee? 2 2000
==> Dernier jour est le 29

Mois, annee? 8 2010
==> Dernier jour est le 31



Validez votre procédure avec la solution.

Solution C++ @[pgdmain.cpp]

```

/**
 * @test
 */
void test_dernierJour()
{
    int annee, mois;
    cout<<"Mois, annee? ";
    cin>>mois>>annee;
    cout<<"==> Dernier jour est le "<<UtilsDT::dernierJour(mois,annee)<<endl;
}
  
```

2.5 Type DateJMA



Définissez le type `DateJMA` sous la forme d'un triplet d'entiers contenant : le numéro de jour, le numéro de mois et le millésime de l'année.



Validez votre définition avec la solution.

Solution C++ @[UtilsDT.cpp]

```

/**
 * Type Date
 */
struct DateJMA
{
    /// numéro de jour
    int jr;
    /// numéro de mois
  
```

```
int mm;
/// millésime de l'année
int an;
};
```

2.6 Procédure calculerDemainDT (lendemain d'une date)



Écrivez le **profil** d'une procédure `calculerDemainDT(dt)` qui passe au lendemain d'une `DateJMA dt` valide.

Solution Paramètres

Modifiés : Une `DateJMA dt`

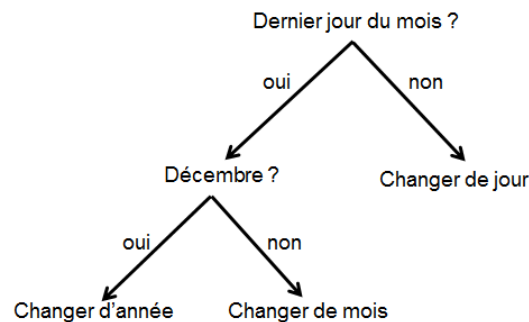
Solution C++

```
void calculerDemainJMA(DateJMA& dt)
```



Analyse

Passer au lendemain consiste en :



- **Changer d'année** consiste à : ajouter 1 à la variable de l'année et à donner 1 aux variables du mois et du jour.
- **Changer de mois** : ajouter 1 à la variable du mois, donner 1 à la variable du jour et laisser la variable de l'année inchangée.
- **Changer de jour** : ajouter 1 à la variable du jour en laissant inchangées les variables du mois et de l'année.



Écrivez le corps de la procédure.



Validez votre procédure avec la solution.

Solution C++ @[pgdemain.cpp]

```

/**
 * Calcule le lendemain d'une DateJMA
 * @param[in,out] dt - une DateJMA
 */
void calculerDemainDT(UtilsDT::DateJMA& dt)
{
    if (dt.jr < UtilsDT::dernierJour(dt.mm,dt.an))
    {
        ++dt.jr;
    }
    else
    {
        dt.jr = 1;
        if (dt.mm < 12)
        {
            ++dt.mm;
        }
        else
        {
            dt.mm = 1;
            ++dt.an;
        }
    }
}

```

2.7 Fonction saisirGregDT (saisie d'une date)



Écrivez une procédure `saisirGregDT(dt)` qui saisit une `DateJMA` dans `dt`. La date doit être demandée tant qu'elle n'est pas une date grégorienne valide. Affichez les invites :

Millesime de l'année?
 Numéro du mois?
 Numéro du jour?



Validez votre procédure avec la solution.

Solution C++ @[UtilsDT.cpp]

```

/**
 * Saisie contrainte d'une date
 * @param[out] dt - une DateJMA
 */
void saisirGregDT(DateJMA& dt)
{
    dt.jr = 0;
    dt.mm = 0;
    dt.an = 0;
    while (!(datumDT(dt) && gregorienneDT(dt)))
    {

```

```

    cout<<"Millesime de l'annee? ";
    cin>>dt.an;
    cout<<"Numero du mois? ";
    cin>>dt.mm;
    cout<<"Numero du jour? ";
    cin>>dt.jr;
}
}

```

2.8 Procédure afficherDT (affichage d'une date)



Écrivez une procédure `afficherDT(dt)` qui affiche une `DateJMA dt` sous la forme (où `[x]` désigne le contenu de `x`) :

```
[jr]/[mm]/[an]
```



Validez votre procédure avec la solution.

Solution C++ @[UtilsDT.cpp]

```

/**
 * Affiche une DateJMA
 * @param[in] dt - une DateJMA
 */
void afficherDT(const DateJMA& dt)
{
    cout<<dt.jr<<" / "<<dt.mm<<" / "<<dt.an<<endl;
}

```

2.9 Test : Lendemain d'une date



Écrivez une procédure `test_demain` qui saisit une `DateJMA`, passe au lendemain puis affiche la nouvelle date.



Testez. Exemple d'exécution :

```

Millesime de l'annee : 2010
Numero du mois : 12
Numero du jour : 31
==> Lendemain est le 1/1/2011

```



Validez votre procédure avec la solution.

Solution C++ @[pgdemain.cpp]

```
/**
 * @test
 */
void test_demain()
{
    UtilsDT::DateJMA dt {};
    UtilsDT::saisirGregDT(dt);
    calculerDemainDT(dt);
    cout<<"==> Lendemain est le ";
    UtilsDT::afficherDT(dt);
}
```

2.10 Application : Nombre de jours écoulés

Ce problème calcule et affiche le nombre de jours écoulés entre deux dates valides. Si la seconde date saisie est antérieure à la première, l'algorithme doit le signaler sans calculer le nombre de jours écoulés.

**Nombre de jours écoulés**

Il existe différentes manières de calculer le nombre de jours écoulés entre deux dates. Celle que nous allons utiliser dans ce problème est loin d'être la plus efficace car très coûteuse en nombre de calculs. En effet, compter le nombre de jours entre deux dates revient à compter combien de fois il faut passer au lendemain, en partant d'une première date (j_1, m_1, a_1) pour la rendre égale à la deuxième (j_2, m_2, a_2).

Pour compter combien de fois il faut passer au lendemain, une boucle est nécessaire dans laquelle on incrémentera un compteur de jours initialisé à 0. On fera évoluer la première date (j_1, m_1, a_1) en la faisant passer au lendemain à chaque passage de la boucle. La répétition devra s'arrêter dès que les deux dates seront égales, c.-à-d. dès que ($j_1=j_2$ Et $m_1=m_2$ Et $a_1=a_2$). Pour traiter correctement le cas où les deux dates données sont égales, on utilisera un **TantQue** dont la condition d'entrée est la négation de la condition d'arrêt de la répétition.



Écrivez une fonction `njoursEntreDT(d1, d2)` qui calcule et renvoie le nombre de jours écoulés entre les `DateJMA d1` et `DateJMA d2`, la première étant supposée antérieure à la deuxième.



Validez votre fonction avec la solution.

Solution C++ @[pgdemain.cpp]

```
/**
 * Nombre de jours écoulés entre deux DateJMA
 * @param[in] d1 - une DateJMA
 * @param[in] d2 - une DateJMA
 * @return le nombre de jours entre d1 et d2 avec d1 antérieure à d2
 */
```

```

int njoursEntreDT(const UtilsDT::DateJMA& d1, const UtilsDT::DateJMA& d2)
{
    int nj = 0;
    UtilsDT::DateJMA dt = {d1.jr,d1.mm,d1.an};
    while (!(dt.jr == d2.jr && dt.mm == d2.mm && dt.an == d2.an))
    {
        ++nj;
        calculerDemainDT(dt);
    }
    return (nj + 1);
}

```



Antériorité d'une date

Il faut aussi vérifier l'antériorité de la première date saisie par rapport à la seconde. La valeur booléenne retournée par la fonction peut être calculée par une expression qui compare les années, puis les mois et enfin les jours.



Écrivez une fonction `posterieureDT(d1,d2)` qui teste et renvoie `Vrai` si la `DateJMA d1` est strictement postérieure à celle définie par `DateJMA d2`.



Validez vos fonctions avec la solution.

Solution C++ @[pgdemain.cpp]

```

/**
    Prédicat de DateJMA postérieure
    @param[in] d1 - première DateJMA
    @param[in] d2 - deuxième DateJMA
    @return Vrai si d1 est postérieure a d2
*/

bool posterieureDT(const UtilsDT::DateJMA& d1, const UtilsDT::DateJMA& d2)
{

```



Écrivez une procédure `test_ecoules` qui demande deux `DateJMA` valides puis calcule et affiche le nombre de jours écoulés entre les deux dates.



Testez.



Validez votre procédure avec la solution.

Solution C++ @[pgdemain.cpp]

```

/**
    @test
*/

```

```
void test_ecoules()
{
    UtilsDT::DateJMA d1 {};
    cout<<"Premiere date: ";
    UtilsDT::saisirGregDT(d1);
    UtilsDT::DateJMA d2 {};
    cout<<"Deuxieme date: ";
    UtilsDT::saisirGregDT(d2);
    if (posterieureDT(d1,d2))
    {
        cout<<"==> OUPS Premiere date posterieure a la deuxieme"<<endl;
    }
    else
    {
        cout<<"==> Nombre de jours ecoules = "<<njoursEntreDT(d1,d2)<<endl;
    }
}
```

3 Références générales

Comprend [Tartier-AL1 :c4 :ex11], [Tartier-AL1 :c6 :ex23], [Tartier-AL1 :c7 :ex26] ■