Algorithmes paramétrés (1) [ss]
Exercices de cours

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 16 mai 2018

Table des matières

C++ - Exercices de cours (Solution)

Mots-Clés Algorithmes paramétrés ■
Difficulté • ○ ○ (1 h 30) ■

1 Appréhender le cours

1.1 Autour de la fonction evalBool / pgevalbool

Écrivez le **profil** d'une fonction `evalBool(b)` qui renvoie 1 si `b` est `Vrai`, 0 sinon.

Solution Paramètres

Entrants : Un booléen `b`

Sortants : Un entier

Écrivez le corps de la fonction de sorte que :

```
evalBool(true) ==> 1
evalBool(false) ==> 0
```

Validez votre fonction avec la solution.

Solution C++

```
/**
  Booléen en un entier (vrai=1, faux=0)
  @param[in] b - un booléen
  @return 1 si b est vrai, 0 sinon
*/

int evalBool(bool b)
{
  return (b ? 1 : 0);
}
```

Écrivez un programme qui teste votre fonction.

Testez. Résultat d'exécution :

```
==> evalbool(false) est 0
==> evalbool(true) est 1
```

Validez votre programme avec la solution.

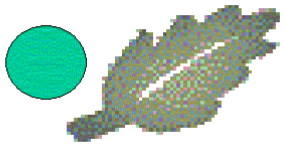
Solution C++ @[pgevalbool.cpp]

```
int main()
{
  cout<<"==> evalbool(Faux) est "<<evalBool(false)<<endl;
  cout<<"==> evalbool(Vrai) est "<<evalBool(true)<<endl;
}
```

1.2 Bulbes du logo d'Ascq / pgparterre

Objectif

C'est le moment de planter les bulbes. La mairie de Villeneuve d'Ascq désire en mettre dans tous les parterres de la ville où fleurit habituellement son logo constitué essentiellement d'une feuille verte et d'un cercle bleu. Seuls cette feuille et ce cercle doivent être fleuris :



Vous êtes embauché pour aider le service des espaces verts à calculer le nombre de bulbes nécessaires pour chaque parterre :

- On sait que le diamètre d du cercle doit être égal à la moitié de la largeur ℓ du parterre.
- On admet que la feuille à la même surface qu'un losange dont la grande diagonale $d1$ est égale à la moitié de la longueur L du parterre, et la petite diagonale $d2$ à la moitié de la largeur ℓ du parterre.
- Le nombre de bulbes au m^2 est de 100.

Cet exercice calcule et affiche le nombre de bulbes nécessaires pour un parterre.

Donnez les expressions de d, $d1$ et $d2$ en termes de ℓ et L.

Écrivez une fonction `surfaceC(r)` qui calcule et renvoie la surface d'un cercle de rayon `r`. Utilisez la formule $\pi \cdot r^2$.

Outil C++

Selon les compilateurs :

GNU C++ La constante `M_PI` de π est définie dans la bibliothèque `<cmath>`.
(inutile de la définir si vous incluez la bibliothèque)

ISO C++ La constante `M_PI` n'est pas standard.
Il faut la définir :

```
const double PI=3.141592653589793116;
```

Écrivez une fonction `surfaceL(d1,d2)` qui calcule et renvoie la surface d'un losange de diagonales `d1` et `d2`. Utilisez la formule $\frac{1}{2} \cdot d1 \cdot d2$.

Écrivez une fonction `surface(L,lgr)` qui calcule et renvoie la surface à fleurir d'un parterre de longueur `L` et largeur `lgr`.

Écrivez une fonction `bulbes(L,lgr)` qui calcule et renvoie le nombre de bulbes nécessaires d'un parterre de longueur `L` et largeur `lgr`.

Écrivez un programme qui saisit la longueur et la largeur d'un parterre puis calcule et affiche la surface à fleurir et le nombre de bulbes nécessaires.

Validez votre programme avec la solution.

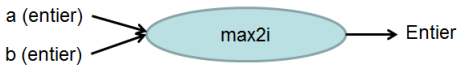
Solution C++

2 Appliquer le cours

2.1 Autour de la fonction max2i / pgminmax2i

Objectif
Cet exercice détermine le plus grand de deux, trois ou quatre entiers.

Écrivez le **profil** d'une fonction `max2i(a,b)` qui renvoie le plus grand des entiers `a` et `b`.



Solution Paramètres
Entrants : Deux entiers `a` et `b`
Résultat de la fonction : Un entier

Écrivez le corps de la fonction.

Validez votre fonction avec la solution.

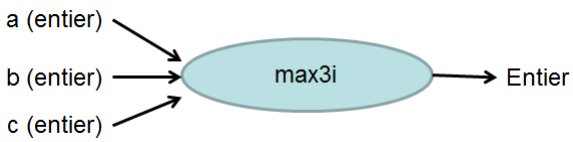
Solution C++

```
/**
  Maximum de deux entiers
  @param[in] a - un entier
  @param[in] b - un entier
  @return le maximum de a et b
*/

int max2i(int a, int b)
{
  return (a > b ? a : b);
}
```

Solution commentée
On teste si `a` est plus grand que `b`, auquel cas on renvoie `a`, sinon c'est `b`. Notez qu'en cas d'égalité, la fonction donne `b` qui vaut `a`. Il est donc inutile de faire le test d'égalité.

Écrivez le **profil** d'une fonction `max3i(a,b,c)` qui renvoie le plus grand des trois entiers `a`, `b` et `c`.



Solution Paramètres

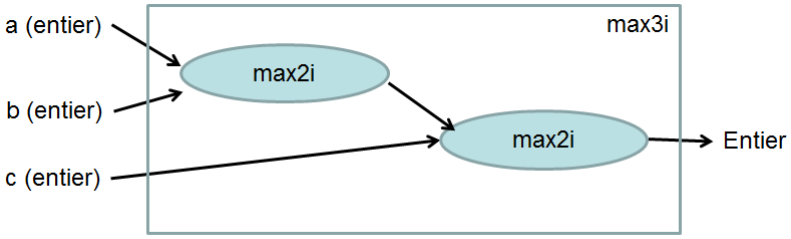
Entrants : Trois paramètres entiers

Résultat de la fonction : Un entier

Écrivez le corps de la fonction en utilisant la fonction `max2i`

Aide méthodologique

Composez deux à deux.



Validez votre fonction avec la solution.

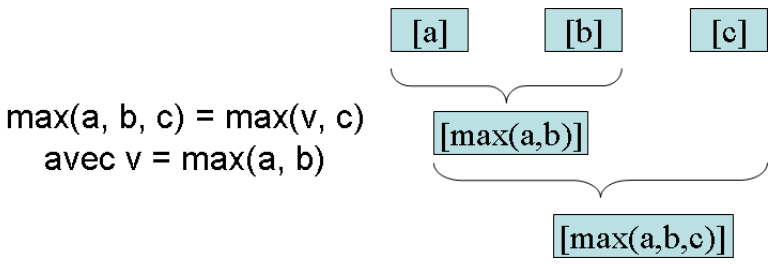
Solution C++

```
/**
  Maximum de trois entiers
  @param[in] a - un entier
  @param[in] b - un entier
  @param[in] c - un entier
  @return le maximum de (a,b,c)
*/

int max3i(int a, int b, int c)
{
  int tmp = max2i(a,b);
  return max2i(tmp,c);
}
```

Solution commentée

On écrit :



La version abrégée de la solution ci-dessus s'écrit :

Solution C++

```
/**
  Maximum de trois entiers
  @param[in] a - un entier
  @param[in] b - un entier
  @param[in] c - un entier
  @return le maximum de (a,b,c)
*/

int max3iv2(int a, int b, int c)
{
  return max2i(max2i(a,b),c);
}
```

Écrivez une fonction `max4i(a,b,c,e)` qui calcule et renvoie le plus grand des quatre entiers `a`, `b`, `c` et `e` en utilisant les fonctions `max2i` et/ou `max3i`.

Validez votre fonction avec la solution.

Solution C++

```
/**
  Maximum de quatre entiers
  @param[in] a - un entier
  @param[in] b - un entier
  @param[in] c - un entier
  @param[in] e - un entier
  @return le maximum de (a,b,c,e)
*/

int max4i(int a, int b, int c, int e)
{
  return max2i(max2i(a,b),max2i(c,e));
}
```

Solution commentée

La fonction détermine le plus grand parmi `a` et `b` qui donne `m1`, puis le plus grand parmi `c` et `e` qui donne `m2`, enfin compare et renvoie le plus grand de `m1` et `m2` c.-à-d. le plus grand des quatre entiers.

Solution C++

```
/**
  Maximum de quatre entiers
  @param[in] a - un entier
  @param[in] b - un entier
  @param[in] c - un entier
  @param[in] e - un entier
  @return le maximum de (a,b,c,e)
*/
```

```
int max4iv2(int a, int b, int c, int e)
{
  return max2i(max3i(a,b,c),e);
}
```

Écrivez un programme qui demande quatre entiers puis calcule et affiche le plus grand des deux premiers, des deux derniers et des quatre entiers.

Testez. Exemple d'exécution :

```
Quatre entiers? 3 -2 8 4
Max des deux premiers  3
Max des deux derniers  8
Max des quatre entiers 8
```

Validez votre programme avec la solution.

Solution C++ @[pgminmax2i.cpp]

```
int main()
{
  int n1, n2, n3, n4;
  cout<<"Quatre entiers? ";
  cin>>n1>>n2>>n3>>n4;

  cout<<"Max des deux premiers  "<<max2i(n1,n2)<<endl;
  cout<<"Max des deux derniers  "<<max2i(n3,n4)<<endl;
  cout<<"Max des quatre entiers "<<max4i(n1,n2,n3,n4)<<endl;
}
```

Écrivez une fonction `min2i(a,b)` qui calcule et renvoie le minimum (= le plus petit) des entiers `a` et `b`.

Validez votre fonction avec la solution.

Solution C++

```
/**
  Minimum de deux entiers
  @param[in] a - un entier
  @param[in] b - un entier
  @return le minimum de a et b
*/

int min2i(int a, int b)
{
  return (a < b ? a : b);
}
```

2.2 Autour de la fonction divisible / pgdivisible

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++

```
/**
  Prédicat de divisibilité de deux entiers
  @param[in] n - un entier
  @param[in] d - un entier
  @return Vrai si n est divisible par d, Faux sinon
*/

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

Propriété
Deux entiers sont **multiples** l'un l'autre si l'un est divisible par l'autre.

Écrivez une fonction `multiple(a,b)` qui renvoie `Vrai` si deux entiers `a` et `b` sont multiples l'un l'autre, `Faux` sinon.

Validez votre fonction avec la solution.

Solution C++

```
/**
  Prédicat de multiplicité de deux entiers
  @param[in] a - un entier
  @param[in] b - un entier
  @return Vrai si a et b sont multiples l'un l'autre, Faux sinon
*/
```

```
bool multiple(int a, int b)
{
  return (divisible(a,b) || divisible(b,a));
}
```

Écrivez un programme qui saisit deux entiers.
Affichez l'invite :

```
Deux entiers?
```

Affichez les divisibilités et la multiplicité des deux entiers.

Testez. Exemple d'exécution :

```
Deux entiers? 132 33
==> divisible(132,33) est 1
==> divisible(33,132) est 0
==> multiple(132,33) est 1
```

Validez votre programme avec la solution.

Solution C++ @[pgdivisible.cpp]

```
int main()
{
  int n1, n2;
  cout<<"Deux entiers? ";
  cin>>n1>>n2;
  bool b1 = divisible(n1,n2);
  bool b2 = divisible(n2,n1);
  bool b3 = multiple(n1,n2);
  cout<<"b1="<<b1<<" b2="<<b2<<" b3="<<b3<<endl;
}
```

3 Approfondir le cours

3.1 Autour de la fonction signe / pgsigne

Écrivez une fonction `signe(n)` qui renvoie −1, 0 ou 1 selon qu'un entier `n` est strictement négatif, nul ou strictement positif. Exemples :

```
signe(5) ==> 1
signe(-8) ==> -1
signe(0) ==> 0
```

Validez votre fonction avec la solution.

Solution C++

```
/**
  Signe d'un entier
  @param[in] n - un entier
  @return le signe (-1,0,1) de n
*/

int signe(int n)
{
  return (n < 0 ? -1 : n > 0 ? 1 : 0);
}
```

Solution simple

La fonction teste d'abord si `n` est négatif : dans l'affirmative elle retourne −1. Sinon elle teste si `n` est plus grand que 0 (donc positif) : si c'est le cas elle retourne 1. Sinon c'est que `n` était nul : elle retourne 0.

Écrivez une fonction `vabs(n)` qui renvoie la valeur absolue d'un entier `n` par appel à la fonction `signe`.

Validez votre fonction avec la solution.

Solution C++

```
/**
  Valeur absolue d'un entier
  @param[in] n - un entier
  @return la valeur absolue de n
*/

int vabs(int n)
{
  return (n >= 0 ? n : -n);
}
```

Écrivez une fonction `signeProduit(a,b)` qui calcule et renvoie le signe du produit des entiers `a` et `b` sans calculer le produit.

Aide méthodologique

Pour le produit, on a :

$$\text{signe}(a \times b) = \text{signe}(a) \times \text{signe}(b)$$

En effet, lorsque les deux nombres sont non nuls mais de même signe, la fonction `signe` renvoie le même entier -1 ou 1 pour les deux, dont le produit vaut $+1$, indiquant ainsi un produit positif. Lorsque l'un des deux nombres est nul, la fonction `signe` renvoie 0 et le produit des signes est nul indiquant que le produit des nombres est nul. Enfin, lorsque les deux nombres sont de signes contraires, on obtient un résultat égal à -1.

Validez votre fonction avec la solution.

Solution C++

```
/**
  Signe du produit de deux entiers
  @param[in] a - un entier
  @param[in] b - un entier
  @return le signe (-1,0,1) de a*b
*/

int signeProduit(int a, int b)
{
  return signe(a) * signe(b);
}
```

De même, écrivez une fonction `signeSomme(a,b)` qui calcule et renvoie le signe de la somme des entiers `a` et `b` sans calculer la somme.

Outil C++

La fonction `abs(x)` de la valeur absolue est définie dans la bibliothèque `<cstdlib>`.

Aide méthodologique

Le signe d'une somme de deux nombres est le signe du nombre qui a la plus grande valeur absolue. Lorsque les valeurs absolues sont égales, la somme est nulle si les nombres ont des signes différents, comme -15 et 15 par exemple. Sinon, c'est le signe commun.

Validez votre fonction avec la solution.

Solution C++

```
/**
  Signe de la somme de deux entiers
  @param[in] a - un entier
  @param[in] b - un entier
  @return le signe (-1,0,1) de a+b
*/

int signeSomme(int a, int b)
{
  int sa = signe(a);
  int sb = signe(b);
  // Résultat: par défaut signe de a
  int rs = sa;
  // Analyse des modifications de rs
  if (std::abs(a) < std::abs(b))
  {
    rs = sb;
  }
  else if (sa == -sb)
  {
    rs = 0;
  }
  return rs;
}
```

Écrivez un programme qui saisit deux entiers puis teste vos fonctions.

Testez. Exemple d'exécution :

```
Deux entiers? -2 6
==> signe(-2) est -1
==> signe(6) est 1
==> signe(0) est 0
==> signeProduit est -1
==> signeSomme est 1
==> vabs(-2) est 2
==> vabs(6) est 6
```

Validez votre fonction et programme avec la solution.

Solution C++ @[pgsigne.cpp]

```
int main()
{
  int n1, n2;
  cout<<"Deux entiers? ";
  cin>>n1>>n2;

  cout<<"==> signe("<<n1<<") est "<<signe(n1)<<endl;
  cout<<"==> signe("<<n2<<") est "<<signe(n2)<<endl;
```

```
  cout<<"==> signe("<<0<<") est "<<signe(0)<<endl;

  cout<<"==> signeProduit est "<<signeProduit(n1,n2)<<endl;
  cout<<"==> signeSomme est "<<signeSomme(n1,n2)<<endl;

  cout<<"==> vabs("<<n1<<") est "<<vabs(n1)<<endl;
  cout<<"==> vabs("<<n2<<") est "<<vabs(n2)<<endl;
}
```