Méthodes de programmation

Guillaume CONNAN

MPSI - Lycée Saint-Stanislas

29 septembre 2010

Sommaire

Sommaire

- Algorithme d'Euclide.
- Algorithme d'addition posée.
- Recette de cuisine.
- Rechercher un mot dans le dictionnaire.

- Algorithme d'Euclide.
- Algorithme d'addition posée.
- Recette de cuisine.
- Rechercher un mot dans le dictionnaire.

- Algorithme d'Euclide.
- Algorithme d'addition posée.
- Recette de cuisine.
- Rechercher un mot dans le dictionnaire.

- Algorithme d'Euclide.
- Algorithme d'addition posée.
- Recette de cuisine.
- Rechercher un mot dans le dictionnaire.

- Machine abstraite.
- Langage algorithmique.
- L'algorithme est-il correct ? Peut-on le prouver ?
- L'algorithme est-il efficace ? Quel est son « coût » ?
- Programmer dans un certain langage : le PASCAL.
- Syntaxe et sémantique : « *Le lapin mange le chasseur* ».

- Machine abstraite.
- Langage algorithmique.
- L'algorithme est-il correct ? Peut-on le prouver ?
- L'algorithme est-il efficace ? Quel est son « coût » ?
- Programmer dans un certain langage : le PASCAL.
- Syntaxe et sémantique : « *Le lapin mange le chasseur* ».

- Machine abstraite.
- Langage algorithmique.
- L'algorithme est-il correct ? Peut-on le prouver ?
- L'algorithme est-il efficace ? Quel est son « coût » ?
- Programmer dans un certain langage : le PASCAL.
- Syntaxe et sémantique : « *Le lapin mange le chasseur* ».

- Machine abstraite.
- Langage algorithmique.
- L'algorithme est-il correct ? Peut-on le prouver ?
- L'algorithme est-il efficace ? Quel est son « coût » ?
- Programmer dans un certain langage : le PASCAL.
- Syntaxe et sémantique : « *Le lapin mange le chasseur* ».

- Machine abstraite.
- Langage algorithmique.
- L'algorithme est-il correct ? Peut-on le prouver ?
- L'algorithme est-il efficace ? Quel est son « coût » ?
- Programmer dans un certain langage : le PASCAL.
- Syntaxe et sémantique : « *Le lapin mange le chasseur* ».

- Machine abstraite.
- Langage algorithmique.
- L'algorithme est-il correct ? Peut-on le prouver ?
- L'algorithme est-il efficace ? Quel est son « coût » ?
- Programmer dans un certain langage : le PASCAL.
- Syntaxe et sémantique : « *Le lapin mange le chasseur* ».

Sommaire

- *On doit porter un casque sur une moto.*
- *On doit porter les animaux sur les escaliers mécaniques.*
- Langage naturel / langage formel.
- Spécification / implantation

- *On doit porter un casque sur une moto.*
- *On doit porter les animaux sur les escaliers mécaniques.*
- Langage naturel / langage formel.
- Spécification / implantation

- *On doit porter un casque sur une moto.*
- *On doit porter les animaux sur les escaliers mécaniques.*
- Langage naturel / langage formel.
- Spécification / implantation Énoncé / démonstration.
- Spécifier, ce n'est pas dire comment sont effectués les calculs mais ce qu'ils calculent.

- *On doit porter un casque sur une moto.*
- *On doit porter les animaux sur les escaliers mécaniques.*
- Langage naturel / langage formel.
- Spécification / implantation Énoncé / démonstration.

→ Spécifier, ce n'est pas dire comment sont effectués les calculs mais ce qu'ils calculent.

- *On doit porter un casque sur une moto.*
- *On doit porter les animaux sur les escaliers mécaniques.*
- Langage naturel / langage formel.
- Spécification / implantation Énoncé / démonstration.
- Spécifier, ce n'est pas dire comment sont effectués les calculs mais ce qu'ils calculent.

- *On doit porter un casque sur une moto.*
- *On doit porter les animaux sur les escaliers mécaniques.*
- Langage naturel / langage formel.
- Spécification / implantation Énoncé / démonstration.
- Spécifier, ce n'est pas dire comment sont effectués les calculs mais ce qu'ils calculent.

- Un exemple : spécifions une fonction qui a pour données un nombre en base dix et qui a pour résultat son écriture en base deux.
  Précisons un peu.
  - La donnée est un nombre entier positif écrit en base dix.

- Un exemple : spécifions une fonction qui a pour données un nombre en base dix et qui a pour résultat son écriture en base deux.
  Précisons un peu.
  - La donnée est un nombre entier positif écrit en base dix.
  - Le résultat est une liste d'entiers appartenant à $\{0;1\}$, dans l'ordre décroissant des puissances de deux associées. On convient de commencer par la plus grande puissance dont le coefficient est non nul.
  - Par exemple, si on entre 11, on obtient $[1,0,1,1]$.

- Un exemple : spécifions une fonction qui a pour données un nombre en base dix et qui a pour résultat son écriture en base deux.
  Précisons un peu.
- La donnée est un nombre entier positif écrit en base dix.
- Le résultat est une liste d'entiers appartenant à $\{0; 1\}$ dans l'ordre décroissant des puissances de deux associées. On convient de commencer par la plus grande puissance dont le coefficient est non nul.
- Par exemple, si on entre 11 on obtiendra... $[1, 0, 1, 1]$.
- En effet, $11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$.

- Un exemple : spécifions une fonction qui a pour données un nombre en base dix et qui a pour résultat son écriture en base deux.
  Précisons un peu.
- La donnée est un nombre entier positif écrit en base dix.
- Le résultat est une liste d'entiers appartenant à $\{0; 1\}$ dans l'ordre décroissant des puissances de deux associées. On convient de commencer par la plus grande puissance dont le coefficient est non nul.
- Par exemple, si on entre 11 on obtiendra... $[1, 0, 1, 1]$.
- En effet, $11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$.

- Un exemple : spécifions une fonction qui a pour données un nombre en base dix et qui a pour résultat son écriture en base deux.
  Précisons un peu.
- La donnée est un nombre entier positif écrit en base dix.
- Le résultat est une liste d'entiers appartenant à $\{0; 1\}$ dans l'ordre décroissant des puissances de deux associées. On convient de commencer par la plus grande puissance dont le coefficient est non nul.
- Par exemple, si on entre 11 on obtiendra... $[1, 0, 1, 1]$.
- En effet, $11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$.

- Un exemple : spécifions une fonction qui a pour données un nombre en base dix et qui a pour résultat son écriture en base deux.
  Précisons un peu.
- La donnée est un nombre entier positif écrit en base dix.
- Le résultat est une liste d'entiers appartenant à $\{0; 1\}$ dans l'ordre décroissant des puissances de deux associées. On convient de commencer par la plus grande puissance dont le coefficient est non nul.
- Par exemple, si on entre 11 on obtiendra... $[1, 0, 1, 1]$.
- En effet, $11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$.

On ne sait pas encore *comment* calculer le résultat mais on sait déjà ce que calcule cette fonction et on peut donc l'intégrer dans des calculs plus complexes.

Sommaire

Sommaire

Prenons un exemple simple : étant donné un entier relatif a et un entier naturel n, on voudrait obtenir a^n qui sera un entier relatif.

Spécifions notre problème de manière plus formelle :

```
puissance(a:entier;n:entier positif):entier
puissance(a,n) renvoie a à la puissance n
```

Spécifions notre problème de manière plus formelle :

```
puissance(a:entier;n:entier positif):entier
 puissance(a,n) renvoie a à la puissance n
```

Sommaire

```
Fonction puissance(a,n : entier) : entier
Début
    Si n=0 Alors
        1
    Sinon
        a*puissance(a,n-1)
    FinSi
Fin
```

- Que fait cet algorithme ?
- Peut-on le prouver ?
- Comment caractériser son efficacité ?

Il s'agit d'un algorithme *récursif* car la fonction est définie en fonction d'elle-même.

Traduction possible en Pascal :

```
FUNCTION Puissance(a:INTEGER;n:INTEGER):INTEGER;
 BEGIN
  IF n=0
  THEN Puissance:=1
  ELSE Puissance:=a*Puissance(a,n-1);
 END;
```

Mais Pascal ne fonctionne pas en mode interactif. Il faut donc inclure cette fonction dans un programme.

Traduction possible en Pascal :

```
FUNCTION Puissance(a:INTEGER;n:INTEGER):INTEGER;
 BEGIN
  IF n=0
  THEN Puissance:=1
  ELSE Puissance:=a*Puissance(a,n-1);
 END;
```

Mais Pascal ne fonctionne pas en mode interactif. Il faut donc inclure cette fonction dans un programme.

Programme Puissance_rec utilisant la fonction Puissance

```
PROGRAM Puissance_rec;
VAR a:INTEGER; n:INTEGER;

 FUNCTION Puissance(a:INTEGER;n:INTEGER):INTEGER;
  BEGIN
   IF n=0
   THEN Puissance:=1
   ELSE Puissance:=a*Puissance(a,n-1);
  END;

BEGIN {programme principal}
  WRITE('Entrer a : ');READLN(a);
  WRITE('Entrer n : ');READLN(n);
  WRITELN;
  WRITE(a,'^',n,' = ',Puissance(a,n));
  READLN;
END.
```

Listing 1 – puissance(a,n) : version récursive 1 en Pascal

et cela donne :

Réponse du logiciel

```
Entrer a : 2
Entrer n : 5

2^5 = 32
```

Dans un autre langage comme Caml, l'idée est la même mais la manière de l'implanter et de l'utiliser diffèrent :

```
# let rec puissance(a,n)=
      if n==0
      then 1
      else a*puissance(a,n-1);;
      val puissance : int * int -> int = <fun>
# puissance(2,5);;
- : int = 32
```

Sommaire

```
Fonction puissance(a,n : entier) : entier
Variable
    res,k : entier
Début
    res ← 1
    Pour k variantDe 1 à n Faire
        res ← res*a
    FinPour
    Retourner res
Fin
```

- Que fait cet algorithme ?
- Peut-on le prouver ?
- Comment caractériser son efficacité ?

En Pascal :

```
FUNCTION Puiss(a:INTEGER;n:INTEGER):INTEGER;
VAR res:INTEGER; k:INTEGER;
 BEGIN
  res:=1;
  FOR k:=1 TO n DO res:=res*a;
  Puiss:=res;
 END;
```

Et si on l'introduit dans un programme :

```
PROGRAM Puissance_it;
VAR a:INTEGER; n:INTEGER;

FUNCTION Puissance(a:INTEGER;n:INTEGER):INTEGER;
VAR res:INTEGER; k:INTEGER;
 BEGIN
  res:=1;
  FOR k:=1 TO n DO res:=res*a;
  Puiss:=res;
 END;

BEGIN {programme principal}
  WRITE('Entrer a : ');READLN(a);
  WRITE('Entrer n : ');READLN(n);
  WRITELN;
  WRITE(a,'^',n,' = ',Puissance(a,n));
  READLN;
END.
```

Que la fonction de calcul de la puissance soit récursive ou itérative, la manière de l'utiliser dans le programme principal est la même.

Sommaire

```
Fonction puissance(a,n : entier) : entier
Variable
    res,k : entier
Début
    res ← 1
    k ← n
    TantQue k>0 Faire
        k ← k-1
        res ← a*res
    FinTantQue
    Retourner res
Fin
```

```
PROGRAM Puissance_tq;
VAR a:INTEGER; n:INTEGER;

FUNCTION Puissance(a:INTEGER;n:INTEGER):INTEGER;
VAR res:INTEGER; k:INTEGER;
 BEGIN
  res:=1;
  k:=n;
  WHILE k>0  DO
    BEGIN
    k:=k-1;
    res:=res*a;
    END;
  Puiss:=res;
 END;

BEGIN {programme principal habituel}
END.
```

Sommaire

Le principe est simple :

$$\begin{cases} a^0 &= 1 \\ a^n &= \left(a^2\right)^{\frac{n}{2}} & \text{, si } n \text{ est pair et } n > 0 \\ a^n &= a \times \left(a^2\right)^{\frac{n-1}{2}} & \text{, si } n \text{ est impair} \end{cases}$$

Écrivez un algorithme utilisant ces relations de récurrence. Qu'y gagne-t-on ?

```
Fonction puissance_log(a,n : entier ) : entier
Début
    Si n=0 Alors
        1
    Sinon
        Si reste(n,2)=0 Alors
            puissance_log(a*a,n/2)
        Sinon
            a*puissance(a*a,(n-1)/2)
        FinSi
    FinSi
Fin
```

Avec Pascal

```
PROGRAM puissance_log;
VAR a:INTEGER; n:INTEGER;
FUNCTION Puiss_log(a:INTEGER;n:INTEGER):INTEGER;
BEGIN
 IF n=0 THEN
     Puiss_log:=1
   ELSE IF n MOD 2 = 0 THEN
            Puiss_log:=Puiss_log(a*a,n DIV 2)
        ELSE  Puiss_log:=a*Puiss_log(a*a,(n-1) DIV 2);
END;
BEGIN {programme principal}
  WRITE('Entrer a : ');READLN(a);
  WRITE('Entrer n : ');READLN(n);
  WRITELN;
  WRITE(a,'^',n,' = ',Puiss_log(a,n));
  READLN;
END.
```

Avec CAML

```
# let rec puissance_log(a,n)=
      if n==0 then 1
      else if n mod 2 == 0 then puissance_log(a*a,n/2)
           else a*puissance_log(a*a,(n-1)/2);;
```

Non, on ne rigole pas...

Avec CAML

```
# let rec puissance_log(a,n)=
      if n==0 then 1
      else if n mod 2 == 0 then puissance_log(a*a,n/2)
           else a*puissance_log(a*a,(n-1)/2);;
```

Non, on ne rigole pas...

Sommaire

L'instruction représentée par :

```
a ← 2
```

qui se lit « a reçoit 2 » réserve (affecte) une zone de la mémoire de l'ordinateur qui portera l'étiquette a et qui contiendra 2.

Donnez le contenu de a et b après la suite d'affectations suivante :

```
a ← 1
b ← 2
a ← a+b
b ← a-b
a ← a-b
```

Que contiennent x et y à la fin de cet enchaînement d'affectations :

```
x ← -5
x ← carre(x)
y ← -x-3
z ← carre(-x-y)
x ← -carre(x-y)+z
y ← puissance(z,x)*y
y ← -(z+y)
y ← x+y-z
y ← x+z
x ← carre(y-z)
y ← x-y
x ← (x+y)/(x/10)
y ← ((x*z)/y)*9
```

Écrire un algorithme qui calcule très rapidement x^{16}.

```
Fonction puissance_16( x : entier ) : entier
Variable
    res : entier
Début
    res ← x
    res ← res*res
    res ← res*res
    res ← res*res
    res ← res*res
    Retourner res
Fin
```

Écrire un algorithme qui calcule très rapidement x^{16}.

```
Fonction puissance_16( x : entier ) : entier
Variable
    res : entier
Début
    res ← x
    res ← res*res
    res ← res*res
    res ← res*res
    res ← res*res
    Retourner res
Fin
```

Déterminez un algorithme qui reçoit un nombre entier de secondes et qui renvoie quatre entiers correspondant à sa conversion en jours, heures, minutes et secondes. On pourra utiliser une fonction « quo » qui renvoie le quotient entier de la division de deux entiers.