ECE1

Année 2011-2012

Cours d'informatique

<i>1. Prise en main du langage Pascal</i>	<i>p.3</i>
<i>2. Mathématiques et Pascal</i>	<i>p.7</i>
<i>3. Les structures de base. Applications aux suites et sommes</i>	<i>p.9</i>
<i>4. Simulation d'une loi de probabilité</i>	<i>p.15</i>
<i>5. Résolution approchée de l'équation $f(x) = 0$</i>	<i>p.17</i>
<i>6. Les fonctions</i>	<i>p.20</i>
<i>7. Tableaux, matrices et polynômes</i>	<i>p.22</i>
<i>8. Les procédures</i>	<i>p.25</i>

Quelques messages d'erreur classiques

Observez toujours la position du curseur. C'est à cet endroit que le compilateur reconnaît une erreur.

Numéro	Description	Explication
...	'... ' expected	Il manque ce symbole à l'endroit du curseur ou juste avant.
3	Unknown identifier	Vous avez oublié de déclarer une variable (ou vous avez mal orthographié une variable).
4	Duplicate identifier	Vous avez nommé deux objets avec le même nom.
8	String constant exceeds line	Vous avez ouvert une apostrophe sans la refermer.
26	Type mismatch	Une de vos variables n'a pas le bon type pour l'opération voulue (stocker un réel dans un entier, compteur de boucle réel, etc...).
113	Error in statement	(Très souvent) Vous avez mis un point-virgule avant ELSE.

1. Prise en main du langage Pascal

1.1 Un langage informatique

Un programme est une succession d'instructions dans un certain langage : ici le langage Pascal. A chaque lancement du programme, l'ordinateur effectue l'une après l'autre les différentes instructions. La syntaxe et la ponctuation y sont très strictes.

Un programme informatique peut être comparé à une recette de cuisine :

- _ on fait la liste des ingrédients dont on aura besoin
- _ on détaille, étape par étape, les différentes opérations à effectuer

L'élaboration d'un programme s'effectue en deux étapes :

- _ L'édition du programme : dans un éditeur de texte, on tape les instructions du programme sous forme de texte (c'est le code source). Le fichier est au format .pas
- _ La compilation : les instructions sont transformées en langage machine et forment ainsi un programme exécutable. Le fichier est au format : .exe

Ici, l'éditeur et le compilateur sont regroupés en un seul logiciel : turbo.exe.

Pour écrire un programme et tester un programme Pascal :

- _ lancer **turbo.exe** situé dans le dossier
- _ écrire le programme
- _ enregistrer votre programme dans le répertoire ...
- _ cliquer sur **Run (ou CTRL+F9)** pour lancer le programme

Attention, enregistrer TOUJOURS votre programme avant de le lancer, car s'il y a une erreur, (programme qui tourne à l'infini), il n'est plus possible d'y revenir !

1.2 Les variables

En informatique, une variable est un espace de stockage en mémoire associé à un nom.

La valeur stockée peut être modifiée au fur et à mesure du programme.

Attention, **seule la valeur actuelle est stockée.**

Quand on appelle une variable dans une ligne de programme, la variable contient la valeur au moment de l'appel, et non plus la valeur initiale.

En Pascal, une variable doit être définie avant d'être utilisée, et son type doit être précisé.

Les différents types que l'on utilisera pour une variable sont :

integer : les entiers entre -32768 et 32767

longint : les entiers entre $\approx -2.10^9$ et 2.10^9

real : les réels en écriture scientifique (attention, ils sont stockés en valeur approchée !)

boolean : booléen, c'est-à-dire true ou false

char : caractère, entre apostrophes : 'A', ...

array : tableau

Remarques : _ les réels sont stockés en écriture scientifique (et en valeur approchée !).

Si les valeurs sont entières, vous avez donc tout intérêt à définir une variable entière.

Exemples : - 12 est stocké 1.2000000000E+01

- pour entrer la valeur 10^{-4} , on peut écrire 1E-04

_ Les types boolean et array seront précisés ultérieurement.

1.3 Structure d'un programme

Un programme Pascal se présente toujours sous la forme :

- _ l'en-tête
- _ la partie déclarative
- _ la partie exécutive appelée corps du programme.

Pour la lisibilité du programme, penser à mettre des commentaires : ils s'écrivent entre { }.

A) L'en-tête s'écrit toujours : PROGRAM Nom ;

B) La partie déclarative

En Pascal, il faut déclarer au départ les variables et les constantes, qu'on va utiliser, ainsi que leur type. Pour nommer une variable, n'utiliser ni accent, ni espace, ni mot réservé. Pensez à utiliser des noms évocateurs.

- _ les variables (dont la valeur pourra changer au cours du programme) sont précédées de VAR
 - _ les constantes (dont la valeur est fixée immédiatement) en écrivant : CONST nom = valeur;
- Déclarer des constantes permet d'obtenir un programme plus lisible (grâce à un nom évocateur) et très facilement modifiable (il suffit de changer la valeur de la constante).

Exemple :

```
CONST n = 100;
VAR
    i, j : integer ;
    x : real ;
```

déclare qu'on utilisera dans le programme principal, un entier n fixé à 100, deux entiers appelés i et j, et un réel x.

C) La partie exécutive

Elle s'écrit sous la forme suivante :

```
BEGIN
    instruction1 ;
    instruction2 ;
    ...
END.
```

Une instruction se termine toujours par ';' . Le seul '.' du programme est le point final. Lors du lancement du programme, l'ordinateur effectue successivement instruction1, puis instruction2, ...

Conclusion : Structure d'un programme :

```
PROGRAM nom;
CONST ....
VAR ....
BEGIN
    ...
    ...
END.
```

1.4 Les procédures d'entrée-sortie

Dans un programme, l'ordinateur est capable de dialoguer avec l'utilisateur.

A) Les procédures de "sortie" : (Ordinateur -> Utilisateur) :

write (parametre, parametre2, ...); (Ecrit à l'écran)

writeln (parametre, parametre2, ...); (Ecrit à l'écran... puis passe à la ligne)
--

parametre peut être :

- _ le nom d'une variable : dans ce cas, l'ordinateur affiche la valeur de la variable
- _ une expression mathématique : dans ce cas, l'ordinateur affiche la valeur du résultat
- _ du texte placé entre apostrophes.

Exemples :

writeln('Bonjour'); affiche Bonjour à l'écran

Si x est une variable qui vaut 5, alors : writeln(x); affiche

writeln('x'); affiche

writeln(2*x-3); affiche

Remarque :

Pour une meilleure compréhension du programme, pensez à présenter ce que vous affichez.

Par exemple, si on veut afficher la variable n, on écrira :

B) Les procédures d'"entrée" : (Utilisateur -> Ordinateur)

readln (var1,var2, ...); (Lit les valeurs tapées par l'utilisateur et les stocke dans les variables var1, var 2, ...)
--

Attention, pour une utilisation aisée du programme, il faut toujours poser une question (avec writeln) avant d'attendre la réponse !

Ex : Demander son âge à l'utilisateur (la variable age est définie comme integer dans la partie déclarative)

Remarques :

_ Dans une instruction readln, il ne peut y avoir que des variables !

_ Pour que la console ne se ferme pas à la fin du programme, écrivez readln; juste avant END.

_ Pour ne pas confondre readln et writeln :

Lorsque vous écrivez une instruction, vous donnez un ordre à l'ordinateur :

writeln lui ordonne d'écrire

readln lui ordonne de lire (donc c'est l'utilisateur qui doit écrire)

1.5 Exemple

Problème : Ecrire un programme qui demande à l'utilisateur deux nombres entiers, et qui affiche la somme.

2. Mathématiques et Pascal

2.1 Affectation

variable := valeur ; (stocke la valeur de droite dans la variable de gauche)
--

A droite, on peut avoir un nombre, une variable ou une opération arithmétique.

Exemples :

$i := 3 * 5$; affecte 15 dans i

$i := j$; affecte la valeur de j à i . (la valeur de j ne change pas)

2.2 Opérateurs arithmétiques et fonctions

Pour les entiers, on a les opérations suivantes : +, -, *, DIV, MOD

DIV donne le quotient de la division euclidienne, MOD donne le reste de cette division.

Exemple : $13 \text{ DIV } 4$ vaut 3 $13 \text{ MOD } 4$ vaut 1

Remarque : Soit n un entier. n est pair si et seulement si $n \text{ MOD } 2 = 0$

Pour les réels, on a : +, -, *, /

Les fonctions mathématiques prédéfinies

ln (réel ou entier) $\rightarrow$ réel

exp (réel ou entier) $\rightarrow$ réel

sqrt (réel ou entier) $\rightarrow$ réel (racine carrée/square root)

abs (réel ou entier) $\rightarrow$ comme le paramètre (valeur absolue)

trunc (réel) $\rightarrow$ entier (partie entière)

Attention, il n'existe pas de fonctions prédéfinies en Pascal pour calculer une puissance !

Pour calculer x^2 par exemple, on écrit : $x * x$

2.3 Compteurs

Incrémentation :

Soit i une variable entière. Pour ajouter 1 à i (= incrémenter), on écrit l'instruction suivante :

$i := i + 1 ;$ (rajoute 1 à i)

Pour compter le nombre de fois qu'un événement se produit, on utilise un compteur (i par exemple), qui est une variable entière :

A retenir :

Compteur : Initialisation (au départ) : $i := 0 ;$
--

A chaque occurrence de l'événement : $i := i + 1 ;$

A la fin, le compteur contient le nombre d'occurrences de l'événement.

2.4 Test / Condition booléenne

On peut comparer deux nombres grâce aux opérateurs suivants :

= < > <= (≤) >= (≥) <> (≠)

Le résultat est un booléen (true ou false).

Exemples : (3 = 2) vaut (3 >=2) vaut

Attention à ne pas confondre :

x = 3 (on compare la valeur de x à 3, le résultat est un booléen)

x := 3 (on affecte la valeur 3 à x).

Les tests seront utilisés essentiellement avec la structure IF que l'on verra ultérieurement.

Opérateurs logiques : NOT, AND, OR (opérateurs logiques)

Règles de calcul :

p	NOT p
true	
false	

p	q	p AND q	p OR q
true	true		
true	false		
false	true		
false	false		

3. Les structures de base

3.1 L'instruction conditionnelle : IF

Le but d'une instruction conditionnelle est d'effectuer ou non une instruction selon qu'une condition soit réalisée ou non.

Première syntaxe :

IF condition THEN instruction1; (si ... alors ...)

condition est un test, donc un booléen.

si condition est vraie, alors instruction1 est effectuée. Sinon, il ne se passe rien, et l'ordinateur passe à la ligne suivante.

Deuxième syntaxe :

IF condition THEN instruction1 (si ... alors ... ELSE instruction2; sinon ...)

si condition est vraie alors instruction1 est effectuée. Sinon, instruction2 est réalisée.

Attention, pas de point-virgule avant ELSE, car le tout est considéré comme une instruction !

Exemple :

x est un nombre entier. On doit afficher à l'écran si x est pair ou impair.

Attention :

Si on veut effectuer plusieurs instructions après le test, il faut créer un bloc d'instruction juste après THEN, délimité par : BEGIN et END; , sinon seule la première instruction sera effectuée ou non en fonction du test !

```
IF .... THEN BEGIN
    instruction1;
    instruction2;
    ...
END;
```

Remarque : un programme doit toujours contenir autant de BEGIN que de END !

Exemple : n est un entier. Si n est supérieur ou égal à 5, on demande une nouvelle valeur de n à l'utilisateur.

3.2 L'instruction répétitive FOR

On doit souvent faire plusieurs fois une instruction ou un bloc d'instruction.

L'instruction FOR s'emploie **lorsque le nombre d'itérations est connu à l'avance.**

Syntaxe :

FOR compteur := debut TO fin DO instruction;
(pour toutes les valeurs de compteur de debut à fin faire ...)

_ compteur est une variable entière, qui va varier dans la boucle

_ debut et fin sont des entiers

La variable compteur prend toutes les valeurs entières entre debut et fin.

La boucle est donc effectuée fin – debut + 1 fois.

Comment agit cette instruction ?

_ compteur prend la valeur debut, instruction est faite une fois.

_ compteur prend la valeur debut + 1, instruction est faite une fois.

...

_ compteur prend la valeur fin, instruction est faite une fois.

Exemple :

Afficher tous les nombres de 11 à 20 :

Remarques :

_ FOR ... TO est une appelée boucle ascendante. La boucle descendante est FOR ... DOWNTO
Ex FOR i:=10 DOWNTO 1 DO ... (i prend toutes les valeurs de 10 à 1)

_ Pour effectuer plusieurs instructions dans la boucle, il faut créer un bloc d'instruction
délimité par BEGIN et END; :

```
FOR ... TO ... DO
  BEGIN
    instruction1 ;
    instruction2 ;
    ...
  END ;
```

Sans BEGIN ... END;, seule la première instruction est dans la boucle.

Attention, BEGIN se place tout de suite après DO.

_ Attention, le compteur doit obligatoirement être défini comme un entier.

3.3 Application au calcul de somme ou de produit

Problème : Soient $u_1, \dots, u_N$ des nombres réels. On veut calculer $S = \sum_{k=1}^N u_k = u_1 + \dots + u_N$.

Méthode : On utilise une variable S qui va contenir progressivement tous les termes de la somme :

_ on initialise S à 0.

_ pour k de 1 à N , on ajoute le terme u_k à S : $S := S + u_k$;

(dans le programme u_k est remplacée par son expression)

A la fin de la boucle, S contient $\sum_{k=1}^N u_k$.

A RETENIR :

Initialisation : $S:=0$;

Relation de récurrence : $S := S + u_k$;

Exemple :

Calcul de $\sum_{k=1}^{10} \sqrt{k+3}$:

Pour un produit : $P = \prod_{k=1}^N u_k$, la méthode est la même, mais attention :

_ l'initialisation est $P:=1$;

_ la relation de récurrence : $P := P \times u_k$;

Ex : Calcul de 3^n

3.4 Application aux suites récurrentes

1) Suite u définie par : $\begin{cases} u_0 \in \mathbb{R} \\ u_{n+1} = f(u_n) \end{cases}$ et N un entier.

Pour calculer u_N , on est obligé de calculer successivement $u_1, u_2, \dots, u_{N-1}, u_N$.

Pour cela, on utilisera une variable réelle u , qui va prendre successivement les valeurs de $u_0, u_1, \dots, u_N$.

Les valeurs successives ne seront pas gardées, mais seulement utilisées pour le calcul du terme suivant.

Méthode :

_ on initialise u à la valeur u_0 (donné dans le programme) : $u:=u_0$;

_ on effectue une boucle FOR pour i allant de 1 à N :

 A chaque étape, on remplace la valeur de u par le terme suivant $f(u)$,
 par l'instruction : $u:=f(u)$;

 (dans le programme f est remplacée par son expression)

_ puisque u commence à u_0 et que, N fois, on calcule le terme suivant, à la fin, u contient u_N .

A RETENIR :

Initialisation : $u := u_0$;

Relation de récurrence : $u := f(u)$;

Exemple :

Soit (u_n) la suite définie par $\begin{cases} u_0 = 2 \\ u_{n+1} = 3u_n - 1 \quad \forall n \in \mathbb{N} \end{cases}$.

Ecrire un programme qui calcule u_{10} et affiche sa valeur.

$$2) \text{ Suite définie par : } \begin{cases} u_0 \in \mathbf{R} \\ u_1 \in \mathbf{R} \\ \forall n \in \mathbf{N}, u_{n+2} = f(u_n, u_{n+1}) \end{cases}$$

On veut calculer u_N où N est un entier fixé.

Le problème dans ce cas, est de garder en mémoire non pas un, mais deux termes consécutifs de la suite. Pour cela, on utilisera deux variables réelles u et v .

- _ au départ, on initialise u à u_0 et v à u_1 . (donc v est en avance sur u)
- _ dans une boucle, on calcule à chaque fois le terme suivant pour u et v
- _ si on a fait $N-1$ itérations, à la fin, u vaut u_{N-1} et v vaut u_N .

Schéma explicatif de la boucle :

Si avant la boucle, u vaut u_n et v vaut u_{n+1} , après la boucle, u doit valoir u_{n+1} et v doit valoir u_{n+2} . Pour ne pas perdre de valeur, on utilisera une variable intermédiaire w .

u	v	w
u_n	u_{n+1}	
u_{n+1}	u_{n+2}	

Conclusion :

```

u:=u0;v:=u1;
FOR i:=2 TO N DO BEGIN
    w:=f(u,v);
    u:=v;
    v:=w;
END;
writeln(' u',N,' vaut ',v);

```

Exemple :

$$\text{Soit } u \text{ la suite définie par : } \begin{cases} u_0 = 2 \\ u_1 = -1 \\ \forall n \in \mathbf{N}, u_{n+2} = 3u_{n+1} - u_n \end{cases}$$

On veut calculer u_{10} :

3.5 Les instructions répétitives REPEAT et WHILE

Les instructions REPEAT et WHILE s'utilisent **quand on ne connaît pas à l'avance le nombre d'itérations**.

Syntaxe :

REPEAT instruction1 UNTIL condition1; (répéter ... jusqu'à ce que ...)
WHILE condition2 DO instruction2 ; (tant que ... faire ...)

Différences entre ces boucles :

- _ avec REPEAT, l'instruction est effectuée au moins une fois, mais pas avec WHILE.
- _ avec REPEAT, la boucle s'arrête quand la condition est vraie, avec WHILE, quand elle est fausse.

Remarques :

_ Attention ! Il faut toujours s'assurer que le test d'arrêt arrivera à un moment donné. Sinon, le programme bouclera indéfiniment ! (pensez à enregistrer votre programme)

_ Contrairement à FOR, il n'y a pas de compteur. Pour compter le nombre d'itérations, il faut initialiser un compteur à 0 avant la boucle, et rajouter 1 à chaque itération ($n := n + 1$); De même, pour une suite (u_n) , il faut initialiser n et ne pas oublier de l'incrémenter (= ajouter 1) à chaque fois.

_ Pour effectuer tout un bloc d'instruction avec WHILE, il est nécessaire de le délimiter par BEGIN et END :

```
WHILE condition DO BEGIN
    ....
END;
```

Ce n'est pas nécessaire avec REPEAT, car le bloc d'instruction est délimité par UNTIL.

Exemple : Soit u la suite définie par :
$$\begin{cases} u_0 = 1 \\ u_{n+1} = \frac{u_n}{2} \forall n \in \mathbb{N} \end{cases}$$

On veut calculer les valeurs de u_n jusqu'à ce que $|u_n| \leq 10^{-3}$, et afficher la valeur de n correspondante.

Avec REPEAT :

Avec WHILE :

4. Simulation d'une loi de probabilité

Preliminaire :

Soit X une V.A.R. à valeurs réelles. Si $X(\Omega) = [0;1[$ et si $P(X \in [a;b]) = b - a$, on dit que X suit la loi uniforme sur $[0;1[$.

Le Pascal permet la génération de nombres aléatoires de deux types : réels et entiers.

Au début du programme, il faut écrire l'instruction `Randomize`; qui permet d'initialiser le générateur.

Deux fonctions sont prédéfinies : `random` et `random(n)` que nous allons étudier.

Attention, chaque appel d'une de ces fonctions renvoie une nouvelle valeur ! Pour garder une valeur, il faut la stocker dans une variable.

On peut considérer que les valeurs renvoyées par des **appels successifs** sont **indépendantes** entre elles.

Rappel : Pour compter le nombre d'occurrences d'un événement, on utilise un **compteur** :

Initialisation (avant la boucle) : $n := 0$; Incréméntation (dans la boucle) : $n := n + 1$;

4.1 Random(n)

Soit $n \in \mathbb{N}$. A retenir :

random(n) est un nombre entier aléatoire entre 0 et n - 1
--

On peut considérer que la valeur renvoyée suit la **loi uniforme sur $\{0, \dots, n - 1\}$**

Pour obtenir un nombre aléatoire entre 1 et n , il faut donc utiliser `random(n) + 1`.

A retenir :

Si $X \rightarrow U(\{1, \dots, n\})$, alors $x := \text{random}(n) + 1$;
--

Attention : Ne pas confondre :

- `random(n) + 1` : loi uniforme sur $\{1, \dots, n\}$, probabilité $1/n$
- `random(n + 1)` : loi uniforme sur $\{0, \dots, n\}$, probabilité $1/(n + 1)$

Exemple : On lance un dé équilibré. On note X le nombre obtenu.

Simulation :

4.2 Random

A retenir :

random est un nombre réel aléatoire de [0;1[.

On peut considérer que la valeur renvoyée suit la **loi uniforme sur [0;1[**.

Remarque : La probabilité que Random appartienne à $[0;p[$ est p
La probabilité que Random appartienne à $[p;1]$ est $1 - p$.

A retenir :

Si $p \in]0;1[$ l'événement $\text{random} < p$ est de probabilité p .

Exemple :

Une urne contient deux boules bleues et une boule rouge.

On tire une boule au hasard. On décide de noter le résultat 'B' ou 'R', stocké dans la variable boule de type char.

Simulation :

Loi de Bernoulli de paramètre p :

Rappel : si on considère une épreuve dont le succès a pour probabilité p , et si X vaut 1 quand il y a un succès et 0 sinon, alors $X \rightarrow B(1,p)$.

A retenir :

**Si $X \rightarrow B(1,p)$: if random < p then x:=1
 else x:=0;**

Loi binomiale de taille n et de paramètre p :

Rappel : si on répète n épreuves indépendantes de probabilité de succès p , et si X est le nombre de succès, alors $X \rightarrow B(n,p)$.

A retenir :

**Si $X \rightarrow B(n,p)$: x:=0;
 for i:=1 to n do if random < p then x:=x+1;**

Exemple : On considère l'urne précédente (2 boules bleues, 1 boule rouge).

On tire l'une après l'autre et en les remettant 5 boules de l'urne.

On note X le nombre de boules bleues obtenues.

Simulation :

5.2 Algorithme mathématique

On va construire deux suites (a_n) et (b_n) telles que $\forall n \in \mathbb{N}, \alpha \in [a_n; b_n]$.

Initialisation : $a_0 =$

$b_0 =$

Quand a_n et b_n sont calculés : Posons $c_n = \frac{a_n + b_n}{2}$

_ si $f(c_n) \geq 0$ alors on pose $\begin{cases} a_{n+1} = \\ b_{n+1} = \end{cases}$

_ sinon, on pose : $\begin{cases} a_{n+1} = \\ b_{n+1} = \end{cases}$

Propriétés des suites (a_n) et (b_n) :

_ on voit d'après la définition que (a_n) est croissante, (b_n) décroissante et que $\forall n \in \mathbb{N}, \alpha \in [a_n, b_n]$

_ $\forall n \in \mathbb{N}, b_n - a_n = \frac{b - a}{2^n}$

Démonstration : $b_n - a_n$ est la longueur de l'intervalle.

A chaque étape, cette longueur est divisée par 2.

Donc la suite $(b_n - a_n)$ est géométrique de raison $q = 1/2$.

Donc $\forall n \in \mathbb{N}, b_n - a_n = (b_0 - a_0) \times \left(\frac{1}{2}\right)^n = \frac{b - a}{2^n}$

_ (a_n) est croissante, (b_n) décroissante et $\lim_{n \rightarrow +\infty} b_n - a_n = 0$, donc les suites (a_n) et (b_n) sont adjacentes. Comme $\alpha \in [a_n; b_n] \forall n \in \mathbb{N}$, **(a_n) et (b_n) convergent vers α**

Remarque :

Comme α est compris entre a_n et b_n , les distances de α à a_n et b_n sont inférieures à $b_n - a_n$.

Pour que a_n ou b_n soient des valeurs approchées de α à ε près, il suffit donc que $b_n - a_n$ soit inférieur à ε .

5.3 Programmation

Pour écrire le programme, nous utiliserons trois variables réelles a , b , c .

On demande une précision epsilon sur α .

Remarque : $a_{n+1} = a_n$ ou $b_{n+1} = b_n$ n'a pas besoin de s'écrire en informatique, puisque la valeur reste inchangée.

Algorithme à retenir :

```
program dichotomie; {Dans le cas où f est croissante}

var a, b, c : real;

begin
  a:=... ; b:=...; {mettre les bornes de l'intervalle}
  repeat
    c := (a + b)/2;
    if f(c) > 0 then b:=c           {remplacer f par son expression si vous n'avez pas
      else a:=c;                   défini de fonction}
  until b - a < epsilon;
  writeln('Une valeur approchée de alpha est ', a); {ou b à la place de a}
  readln;
end.
```

Remarque :

Si f est **strictement décroissante** sur $[a;b]$, alors $f(a) > 0$, $f(b) < 0$ et si $f(c) > 0$, $\alpha \in [c ; b]$.

L'algorithme devient alors :

```
if f(c) > 0 then a:=c
  else b:=c;
```

Exemple :

Soit $f(x) = e^x + x - 2$. On admet toujours que la fonction définie par $f(x) = e^x + x - 2$ est strictement croissante sur $[0;1]$ et que l'équation $f(x) = 0$ admet une unique solution α sur $[0;1]$.

Ecrire un programme qui détermine une valeur approchée de α à 10^{-3} près.

6. Les fonctions

Problème :

Dans un programme, on effectue souvent le même calcul plusieurs fois. Pour cela, on peut créer des fonctions au début, et uniquement les appeler dans le programme principal. Ainsi, les calculs annexes sont faits à part, et le programme principal est plus simple et plus lisible.

Une fonction peut avoir aucun, un ou plusieurs paramètres. Par contre, elle donne un résultat de type entier, réel ou booléen (pour ce qui nous intéresse).

Exemple : On veut créer une fonction qui à un réel x et un entier n associe x^n .

_ La fonction dépend de deux paramètres : x (réel), et n (entier)

_ Le résultat est un réel.

6.1 Déclaration d'une fonction

La déclaration d'une fonction s'effectue dans la partie déclarative du programme principal. Une fonction est un "sous-programme", qui a la même structure qu'un programme principal.

Syntaxe :

```
FUNCTION nomfonction (parametre1 : type1; ... ; parametreN : typeN) : typerésultat;  
VAR ....  
BEGIN  
    ...  
    nomfonction:=...;  
END;
```

_ parametre1, ..., parametreN sont des variables. Attention, elles sont séparées ici par des points-virgules.

_ Dans VAR, on déclare les variables locales : elles ne sont définies que dans la fonction pour aider au calcul du résultat (contrairement aux variables globales, définies dans tout le programme)

Les paramètres de la fonction ne sont pas des variables locales.

_ Corps de la fonction : Son unique but est de calculer la valeur de nomfonction. Il doit donc se terminer par une instruction du type nomfonction :=

On n'affecte qu'une fois la variable nomfonction. S'il y a des calculs intermédiaires, il faut utiliser une variable locale, et on affecte nomfonction seulement à la fin.

_ Une fonction ne sert qu'à calculer. Il est donc d'usage de ne pas mettre de writeln ni de readln dans une fonction.

Exemples :

1) Ecrire une fonction f qui à un réel x associe $f(x) = e^{2x}(x - 1)$

2) Ecrire une fonction `Somme`, qui à un entier n associe le nombre $\text{Somme}(n) = \sum_{k=1}^n \frac{1}{k}$

6.2 Appel de la fonction

Une fois la fonction déclarée, on peut l'utiliser de la même façon que les fonctions prédéfinies (`ln`, `exp`, `sqrt`, ...) par son nom suivi de ses paramètres.

Plus précisément, si $val1, val2, \dots, valN$ sont des valeurs ou des variables de type $type1, \dots, typeN$, alors `nom(val1,..., valN)` est une valeur de type `typeresultat`.

On peut aussi appeler une fonction dans une autre fonction, si cette dernière est déclarée après.

Remarques :

- _ Le résultat d'une fonction est un nombre, en aucun cas une instruction.
- _ Lors de l'appel d'une fonction, on doit respecter le nombre, l'ordre et le type des paramètres déclarés.
- _ Attention, lors de l'appel d'une fonction, les paramètres sont séparés par des virgules.

Exemple : Ecrire un programme qui affiche $f(3)$, où f est la fonction définie dans l'exemple précédent.

7. Tableaux, matrices et polynômes

7.1 Définition d'un tableau

En pascal, on peut définir un tableau à une, deux ou plusieurs dimensions.

Pour cela on crée un nouveau type dans la partie déclarative, avant VAR de la façon suivante :

TYPE nomtableau = ARRAY [debut..fin] of typeélément;

où debut et fin sont des entiers et typeélément un type tel que real, integer, ...

nomtableau est maintenant un type au même titre que real, integer, longint...

Une variable de type nomtableau sera un tableau dont les coefficients sont indexés de debut à fin et sont tous de type typeélément.

Si on définit des variables de type nomtableau :

VAR t : nomtableau;

t est alors un tableau dont les coefficients sont t[debut],..., t[fin], tous de type typeélément.

Exemple :

On veut travailler sur les vecteurs de $\mathbb{R}^3$ de la forme $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

On définira donc un type vecteur qui contient trois coordonnées de type real.

Dans le programme, on définit un vecteur $U = \begin{pmatrix} 3 \\ 4 \\ -2 \end{pmatrix}$.

On veut ensuite afficher U :

```
program exemple;
```

```
TYPE
```

```
VAR
```

```
Begin
```

```
End.
```

Tableau à deux dimensions :

TYPE tab = array[debut1..fin1, debut2..fin2] of typeélément.

Exemple : On veut travailler sur les matrices de $M_3(\mathbb{R})$.

On définit : TYPE matrice =

7.2 Opérations sur les tableaux

Attention, la seule opération acceptée directement sur un tableau est l'opération de transfert :

$U:=T;$, qui transfère tous les coefficients du tableau T dans le tableau U.

Pour toutes les autres opérations (calcul, lecture, affichage, ...), il faut utiliser une boucle FOR pour faire l'opération pour chaque cellule. (et deux boucles FOR imbriquées dans le cas d'un tableau à deux dimensions).

Exemple :

On veut demander les coefficients d'une matrice $A \in M_3(\mathbb{R})$ au clavier, puis les afficher à l'écran :

7.3 Application aux polynômes

Soit $n \in \mathbb{N}$. On veut travailler sur des polynômes de $\mathbb{R}_n[X]$.

Un tel polynôme s'écrit sous la forme : $P = p_0 + p_1X + \dots + p_nX^n$.

On représente P par ses coefficients $p_0, p_1, \dots, p_n$ de type real.

A retenir :

Pour $i \in \{0, \dots, n\}$, $P[i]$ est le coefficient en X^i

Exemple : Si on travaille dans $\mathbb{R}_3[X]$

On peut définir : TYPE polynome = ARRAY[0..3] of real;

Dans ce cas le polynome $P(X) = 2X^3 + 3X - 7$ sera stocké sous la forme :

$P[0]:=$ $P[1]:=$ $P[2]:=$ $P[3]:=$

Remarque : Attention, ne pas confondre : $_P[1]$ qui contient le coefficient en X^1

$_P(1)$: valeur de P quand $X = 1$

7.4 Méthode de Horner

Soit $P(x) = p_0 + p_1x + \dots + p_{n-1}x^{n-1} + p_nx^n$ un polynôme de degré n élevé. Soit $a \in \mathbb{R}$.

Problème : Calculer rapidement le nombre $P(a)$.

Les additions étant très rapides, on cherche à minimiser le nombre de multiplications.

Si on calcule $P(a) = p_0 + p_1 \times a + p_2 \times a^2 + \dots + p_n \times a^n$,

on effectue $1 + 2 + \dots + n = \frac{n(n+1)}{2}$ multiplications

Méthode de Horner : Ecrivons $P(a)$ sous la forme :

$$P(a) = p_0 + a(p_1 + a(p_2 + a(\dots + a(p_{n-2} + a(p_{n-1} + ap_n))))).$$

On effectue alors $1 + 1 + \dots + 1 = n$ multiplications.

Cette méthode est donc beaucoup plus rapide pour n très grand.

Algorithme :

Soit $a \in \mathbb{R}$ Pour calculer $P(a)$, on calcule donc successivement :

$p_n, p_{n-1} + ap_n, p_{n-2} + a(p_{n-1} + ap_n), \dots, p_0 + a(p_1 + \dots)$

En stockant les résultats intermédiaires dans une variable réelle b , on obtient l'algorithme :

```
b:=p[n];
for i := n - 1 downto 0 do b := p[i] + a × b;
```

A la fin, b contient $P(a)$.

8. Les procédures

Problèmes :

- _ une fonction peut renvoyer un réel ou un entier, mais pas un tableau.
- _ dans une fonction, on ne peut pas dialoguer avec l'utilisateur.

On utilise alors une procédure :

- _ elle permet de ne pas répéter une même suite d'instruction dans un programme
- _ elle permet de séparer les tâches en sous tâches, ce qui permet de gagner en lisibilité.
- _ une procédure peut même être réutilisée telle quelle dans un autre programme.

Syntaxe : Dans la partie déclarative du programme :

```
PROCEDURE nom(paramètres);  
VAR  
    variables locales  
BEGIN  
    ....  
END;
```

Dans la partie exécutive du programme :

nom(paramètres); est une instruction (alors que pour une fonction, c'était une donnée du type défini dans la fonction)

Les différents types de paramètres :

_ passage par valeurs : les valeurs de ces paramètres sont utilisés dans la procédure, mais ne sont pas modifiées à l'intérieur de la procédure.

Dans le programme principal, lors de l'appel de la procédure, le paramètre peut être sous la forme d'une valeur, d'une expression ou d'une variable.

_ passage par variables : une modification de ces paramètres est répercutée dans le programme principal.

Dans la déclaration des paramètres, le paramètre est alors précédé de VAR (mais pas dans l'appel de la procédure).

Dans le programme principal, le paramètre doit être une variable.

Ex : PROCEDURE nom(x : real ; VAR y:real);
 passage passage
 par valeurs par variables

Dans le programme principal, si z et t sont des variables de type réel, on pourra appeler nom(z,t); nom(3,t); nom(2×z – 3,t); mais pas nom(2,1);

Remarque :

Comme pour les fonctions, il ne faut pas confondre les variables locales, définies uniquement dans la procédure, et les variables globales, définies dans tout le programme.

En particulier, les paramètres de la procédure ne sont pas des variables définies dans tout le programme.

Exemple :

On travaille sur les vecteurs de $\mathbb{R}^2$. Pour cela on crée un type vecteur à deux coordonnées réelles.

On veut écrire une procédure qui additionne deux vecteurs.

Dans le programme principal, on testera cette procédure avec les vecteurs $U = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ et $V = \begin{pmatrix} -1 \\ 3 \end{pmatrix}$.

```
program exemple;
```

```
type vecteur=array[1..2] of real;
```

```
procedure Addition(
```

```
begin
```

```
end;
```

```
VAR
```

```
begin
```

```
end.
```