Sommaire

L'objectif de cette série d'exercices est de comprendre les notions de : expressions et leurs évaluations, déroulement d'algorithme, teste alternatif simple et double ainsi que la notion de boucles.

Vous allez aborder les points suivants :

- Déterminer le type d'une valeur (entier, réel, caractère, chaîne ou booléen)
- Écrire correctement les valeurs et les expressions dans le langage PASCAL
- Déroulement d'un algorithme
- Organigramme
- Utilisation du teste alternatif double et simple
- Utilisation des boucles Pour, Tant-que et Répéter (*for*, *while* et *repeat*).
- Calcul de sommes et produits itératives

Exercices Supplémentaires : Niveau 02

Exercice 1 : Types de valeurs

Donner le type des constantes suivantes : *2010 ; 124.5 ; 667.0E-8 ; 'A' ; 'erreur : divisions par zéro' ; TRUE ; FALSE*

Réponse :

Valeur	***Type (Algorithme)***	***En Pascal***
2010	Entier	Integer
124.5	Réel (avec une virgule fixe)	Real
667.0E-8	Réel (avec une virgule flottante)	Real
'A'	Caractère	Char
'erreur:division par zéro'	Chaîne	String
TRUE	Booléen	Boolean
FALSE	Booléen	Boolean

En algorithmique et en langage PASCAL, il y 5 types de base.

Exercice 2 : Nombres

Exprimer les nombres suivants dans un langage (PASCAL) : *8,50 96,2 10^9 0,39 10^{-16}*

Réponse :

Nombres	***En PASCAL***
8,50	*8.50*
96,2 10^9	*96.2 E+9*
0,39 10^{-16}	*0.39 E-16*

Exercice 3 : Expressions

Pour les expressions mathématiques (arithmétiques / logiques), la transcription manuelle n'est pas toujours valables pour le langage PASCAL. Le tableau suivant montres comment écrire correctement quelques opérations / fonctions mathématiques

Expression normale	***Expression en Pascal***
$\frac{a}{b}$	*a / b* (division réel : résultat avec la virgule) *a* ***div*** *b* (division entière : résultat entier)
a^2	*sqr(a)*
$\sqrt{a}$	*sqrt(a)*
$\|a\|$	*abs(a)*
e^x	*exp(x)*
Ln(x)	*ln(x)*
a^n	*exp(n*ln(x))*

Exprimer les expressions suivantes dans un langage (PASCAL) :

$$\frac{-b+\sqrt{b^2-4ac}}{2a} \qquad 2x\,e^x \qquad \frac{(x-y)^{2k}}{ab}$$

Réponse :

Expression	***En Pascal***
$\frac{-b+\sqrt{b^2-4ac}}{2a}$	*(-b + sqrt(sqr(b) – 4*a*c)) / (2 * a)*
$2x\,e^x$	*2 * x * exp(x)*
$\frac{(x-y)^{2k}}{ab}$	*exp(2*k*ln(x-y)) / (a*b)*

Il faut faire attention lors d'écriture des expressions mathématiques (arithmétiques ou logiques).

Exercice 4 : Identificateurs valides / non valides

Indiquer les identificateurs valides et non valides parmi la liste suivante : *1A ; R? ; K2 ; T280 ; 12R ; Hauteur ; Prix-HT ; Prix_HT ; Code prod ; Code_prod*

Réponse :

Identificateurs	***Valide / Non Valide***
1A	*Non valide : il commence par un chiffre*
R?	*Non valide : il contient le point d'interrogation*
K2	*Valide*

T280	*Valide*
12R	*Non valide (comme le premier)*
Hauteur	*Valide*
Prix-HT	*Non valide : il contient tiré 6 (signe de moins)*
Prix_HT	*Valide*
Code prod	*Non valide : il contient un blanc*

- ✔ Il faut revoir la définition de l'identificateur en cours
- ✔ Les identificateurs sont utilisés pour donner un nom à un programme, constante ou variables

Exercice 5 : Déclaration de variables (Affectation / Lecture)

Écrire un algorithme / programme PASCAL permettant d'initialiser des variables par affectations ou par lectures les données (valeurs) suivantes et les affiche à l'écran :

135 -125 150,0 127543,50 96,2 10^9 0,39 10^{-16} 'A' 'Informatique 1' true false

(Il y a 10 valeurs, donc vous devez déclarer 10 variables = 10 espaces mémoires)

Première solution : Par affectation

L'algorithme :

```
Algorithme exercie_5
     Variables
          a, b : entier
          c, d, e, f : réel
          g : caractère
          h : chaîne
          i, j : booléen
Début
     a ← 135 ;  b ← -125 ;  c ← 150.0 ;  d ← 127543.50
     e ← 96.2 E 9 ; f ← 0.39 E-16 ; g ← 'A' ;
     h ← 'Informatique 1' ; i ← true ; j ← false ;
     Écrire(a, b, c, d, f, g, h, i, j)
Fin
```

Le programme PASCAL :

```
1  Program exercie_5;
2  Uses wincrt ;
3  var
4      a, b : integer ; c, d, e, f : real ; g:char ;  h:string ;
5      i, j : boolean ;
6  Begin
7      a:=135; b:=-125; c:=150.0;  d:=127543.50; e:=96.2 E9;
8      f:=0.39E-16;   g:='A';  h:='Informatique 1'; i:=true ;
9      j:=false;
10     write (a, b, c, d, e, f, g, h, i, j) ;
11 End.
```

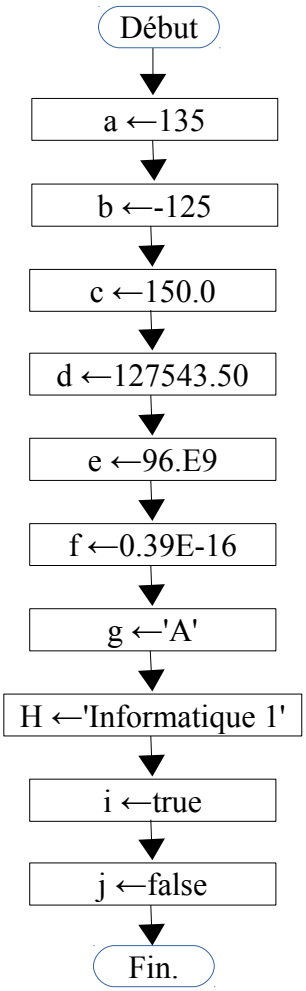
✔ Dans cette première solution, on écrit les valeurs dans le code sources (dans le programme lui même). Et les valeurs sont connus avant d'exécuter le programme.

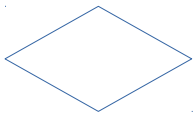
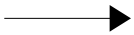
✔ La partie gauche d'une affectation est toujours une variables

✔ Vous pouvez ajouter des chaîne de caractères (message) à la fonction *write* pour montrer les noms des variables et leurs valeurs respectives : *write ('a = ', a, ' b=', b, ….etc.*

Organigramme :

L'organigramme est une façon de montrer un algorithme (ou un programme) sous forme d'actions ***schématisées*** et leurs enchaînement (***flèches***)



Les différentes figures d'organigramme	
Figure	***Sémantique / Sense***
	Représente le début et la Fin de l'organigramme
	Entrées / Sorties : Lecture des données et écriture des résultats.
	Calculs, Traitements
	Tests et décision : on écrit le test à l'intérieur du losange
	Ordre d'exécution des opérations (Enchaînement)
	Connecteur

Deuxième solution : Par lecture

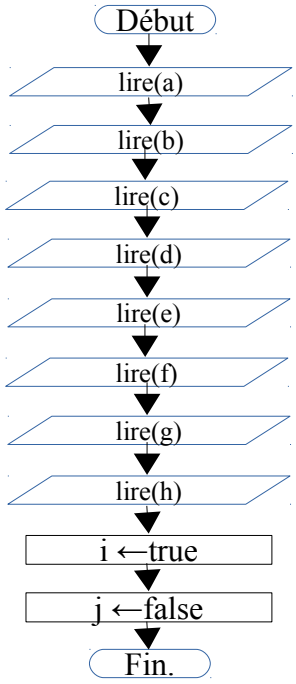
L'algorithme :

```
Algorithme val_absolue_carre
     Variables
          a, b : entier
          c, d, e, f : réel
          g : caractère
          h : chaîne
          i, j : booléen
Début
     lire(a) ;  lire(b) ;  lire(c) ;  lire(d) ;
     lire(e) ; lire(f) ; lire(g) ;
     lire(h) ; i ← true ; j ← false ;
     Écrire(a, b, c, d, f, g, h, i, j)
Fin
```

Le programme PASCAL :

```
1  Program val_absolue_carre;
2  Uses wincrt ;
3  var
4      a, b : integer ; c, d, e, f : real ; g:char ;  h:string ;
5      i, j : boolean ;
6  Begin
7      read(a); read(b); read(c);  read(d); read(e);
8      read(f);   readln(g);  readln(h); i:=true ;
9      j:=false;
10     write (a, b, c, d, e, f, g, h, i, j) ;
11 End.
```

Organigramme :



Exercice 6 : Exécution / Déroulement d'un algorithme

Exécuter les séquences d'instructions suivantes manuellement et donner les valeurs finales des variables A, B, C et celle de X, Y, Z.

a) A←5 ; B←3 ; C←B+A ; A←2 ; B←B+4 ; C←B-2

b) X←-5 ; Y←2*X ; X←X+1 ; Y←sqr(-X-Y) ; Z←sqr(-X+Y) ; X←-(X+3*Y)+2

Réponse :

Exécuter manuellement une séquences d'instructions (algorithmes / programme) nous permet de voir l'évolution des variables (changement de valeurs pour les variables) et ça nous permet de comprendre l'algorithme et de prouver qu'il donne un bon résultat.

Pour dérouler un algorithme / programme on utilise un tableau dans le quel les colonnes représentent les variables et les lignes représentent les instructions.

a) Pour la première séquence, nous avons 3 variables : A, B et C. et selon les affectations on déduit qu'ils sont de type entier.

Variables / Instructions	A	B	C
A←5	5	/	/
B←3	5	3	/
C←B+A	~~5~~	3	8
A←2	2	~~3~~	8
B←B+4	2	7	~~8~~
C←B-2	2	7	5

Les valeurs finales sont : A=2 B=7 C=5

b) Pour la deuxième séquence, nous avons 3 variables : X, Y et Z. et selon les affectations on déduit qu'ils sont de type entier.

Variables / Instructions	X	Y	Z
X←-5	-5	/	/
Y←2*X	~~-5~~	-10	/
X←X+1	-4	~~-10~~	/
Y←sqr(-X-Y)	-4	196	/
Z←sqr(-X+Y)	~~-4~~	196	40000
X←-(X+3*Y)+2	-582	196	40000

Les valeurs finales sont : X=-582 Y=196 Z=40000

L'algorithme :

```
Algorithme exo6_a
     Variables
          a, b, c : entier
Début
     A←5; B←3; C←B+A; A←2; B←B+4; C←B-2;
     Écrire(A, B, C)
Fin
```

Le programme PASCAL :

```
1  Program exo6_a;
2  Uses wincrt ;
3  var
4      a, b, c : integer ;
5  Begin
6      A:=5 ; B:=3 ; C:=B+A ; A:=2 ; B:=B+4 ; C:=B-2 ;
7      Write ('A = ', A, '    B = ', B, '    C = ', C) ;
8  End.
```

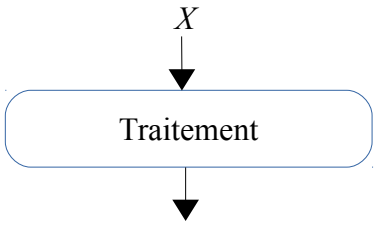
N.B. : ça sera la même chose pour la deuxième séquence.

Exercice 7 : Le signe d'un nombre

Écrire l'algorithme puis le programme PASCAL qui lit un nombre réel X, détermine et affiche son signe. Selon le cas : il affiche '*X est positif*', '*X est négatif*' ou '*X est nul*'.

Réponse :

Cette algorithme peut être schématisé comme suit :



Message (l'un des 3 messages selon le cas)

On introduit un nombre réel quelconque, et l'algorithme affiche une message indiquant si le nombre est positif, négatif et bien nul. Pour introduire ce nombre on a besoin d'une variable réelle X. Donc la première instruction est lire(x). à la fin on doit afficher un message.

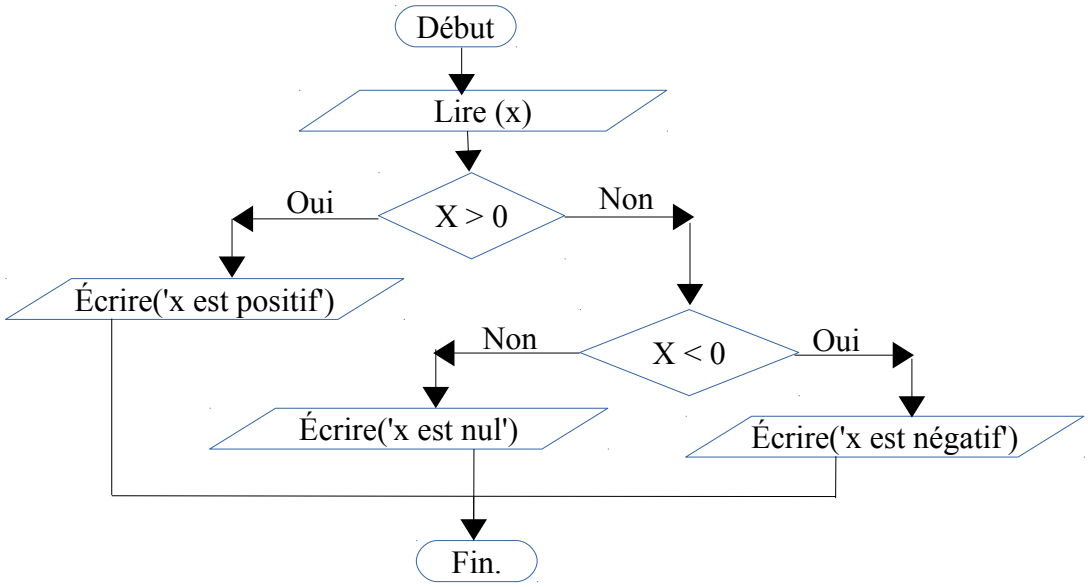
L'algorithme :

```
Algorithme val_absolue_carre
     Variables
          x : réel
Début
     lire(x)
     Si x > 0 alors
          Écrire('x est positif')
     Sinon
          Si x < 0 alors
               Écrire ('x est négatif')
          Sinon
               Écrire ('x est nul')
          FinSi
     FinSi
Fin
```

Le programme PASCAL :

```
1  Program val_absolue_carre;
2  Uses wincrt ;
3  var
4      X : real ;
5  Begin
6      Read(X);
7      if x>0 then
8           Write('X est positif')
9      else
10          if x < 0 then
11               Write('X est négatif')
12          else
13               Write('X est nul') ;
14 End.
```

Organigramme :



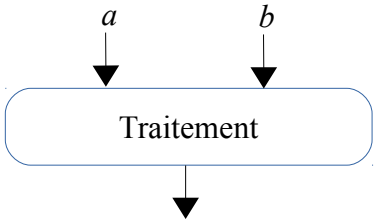
Exercice 8 : Comparaison entre deux nombres

Écrire l'algorithme puis le programme PASCAL qui lit deux nombres réels *a* et *b*, et affiche :

'a est plus grand que b'	*si a > b*
'a est plus petit que b'	*si a < b*
'a est égal à b'	*si a = b*

Réponse :

Cette algorithme peut être schématisé comme suit :



Message (l'un des 3 messages selon le cas)

On introduit deux nombres réels quelconques, et l'algorithme affiche une message indiquant si le premier nombre est plus grand, plus petit ou égale au deuxième nombre. Pour introduire ces deux nombres on a besoin de deux variables réelle *a* et *b*. Donc la première instruction sera *lire(a, b)*. À la fin on doit afficher un message : *'a est plus grand que b'*, *'a est plus petit que b'* ou *'a est égal à b'*.

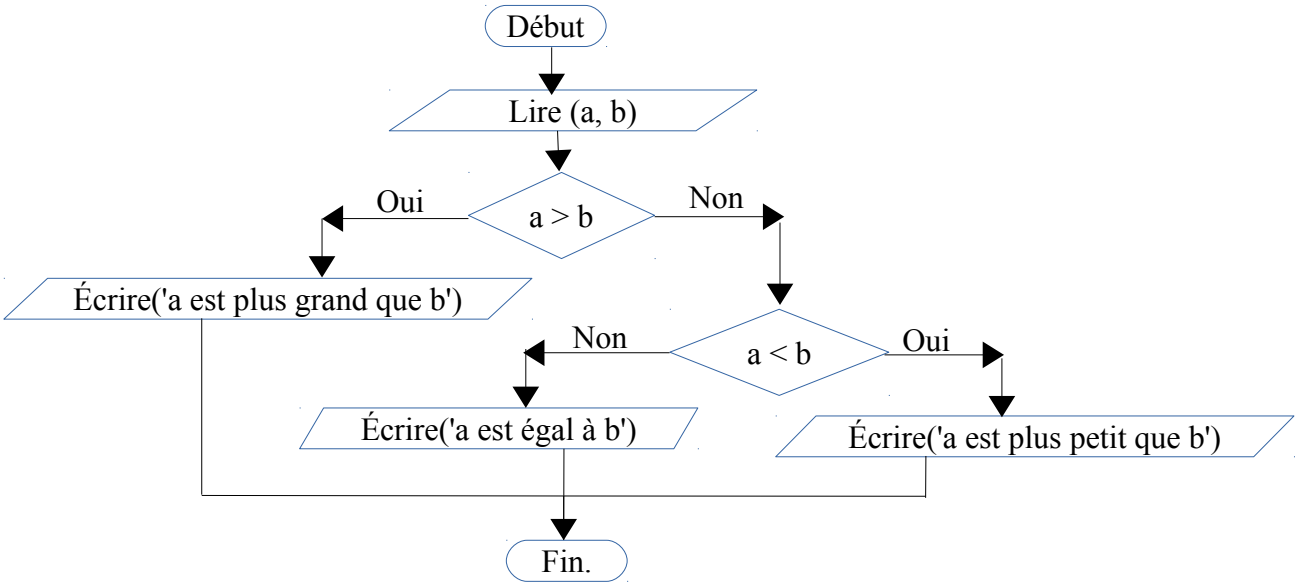
L'algorithme :

```
Algorithme val_absolue_carre
     Variables
          a, b : réel
Début
     lire(a, b)
     Si a > b alors
          Écrire('a est plus grand que b')
     Sinon
          Si a < b alors
               Écrire ('a est plus petit que b')
          Sinon
               Écrire ('a est égal à b')
          FinSi
     FinSi
Fin
```

Le programme PASCAL :

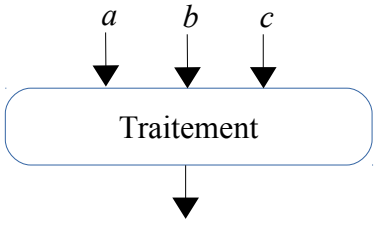
```
1  Program val_absolue_carre;
2  Uses wincrt ;
3  var
4      a, b : real ;
5  Begin
6      Read(a, b);
7      if a>b then
8           Write('a est plus grand que b')
9      else
10          if x < 0 then
11               Write('a est plus petit que b')
12          else
13               Write('a est égal à b') ;
14 End.
```

Organigramme :



Exercice 9 : Résolution de l'équation $ax^2+bx+c=0$

Écrire l'algorithme (et le programme PASCAL) permettant de résoudre, dans , l'équation $ax^2+bx+c=0$. Avec $a \neq 0$.



On a Trois cas :

- *si delta < 0 : Pas de solution dans IR*
- *si delta = 0 : Solution double x1=x2*
- *si delta > 0 : Deux solutions différentes*

La première étape dans l'algorithme est d'introduire les valeurs de a, b et c. En suite, on calcule delta. En utilisant les tests alternatifs doubles (*si … sinon …*), l'algorithme décide quoi afficher comme résultat.

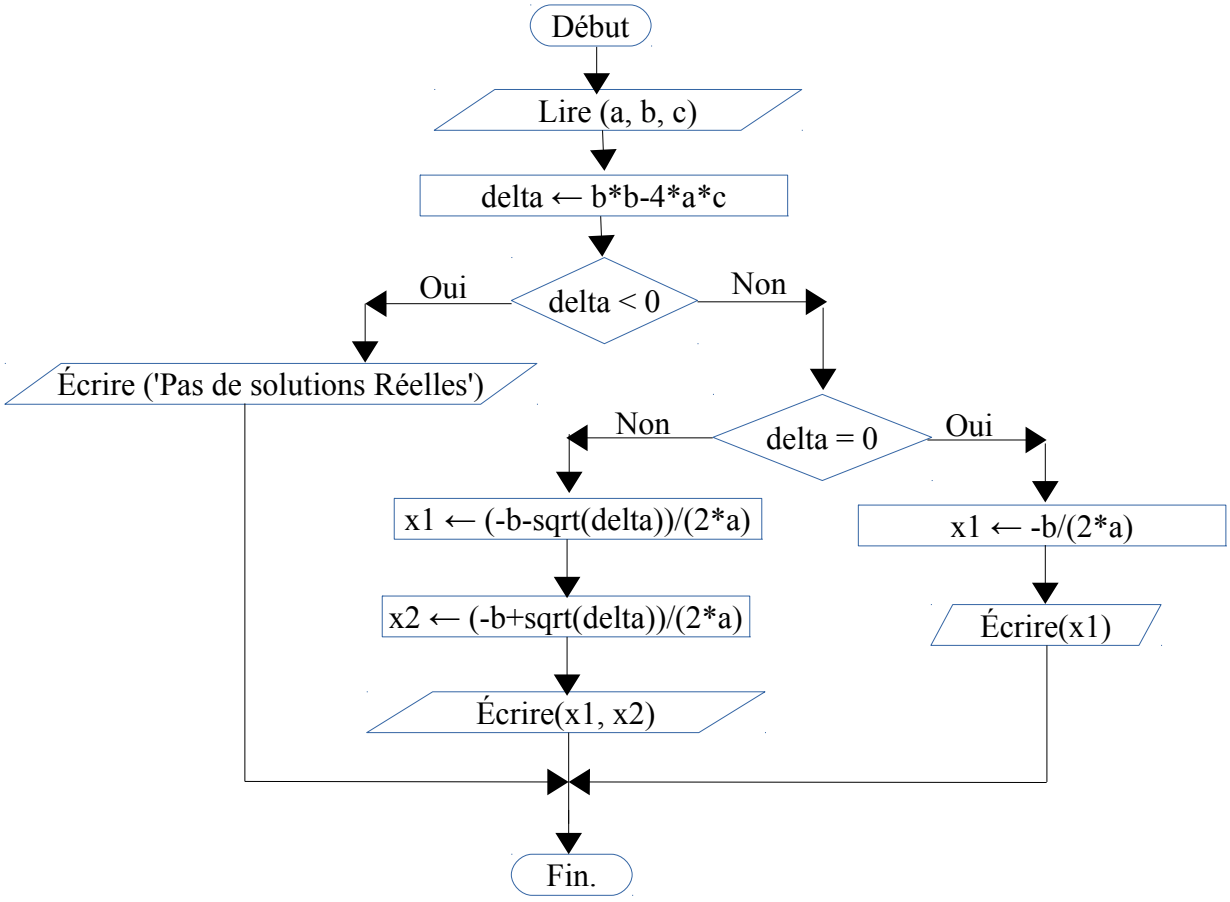
L'algorithme :

```
Algorithme val_absolue_carre
     Variables
          a, b, c, delta, x1, x2 : réel
Début
     lire(a, b, c)
     Delta ← b*b – 4*a*c
     Si delta < 0 alors
          Écrire('Pas de solutions réelles')
     Sinon
          Si delta = 0 alors
               x1 ← –b/(2*a)
               Écrire ('Solution double x1=x2=', x1)
          Sinon
               x1 ← (–b–sqrt(delta))/(2*a)
               x2 ← (–b+sqrt(delta))/(2*a)
               Écrire ('deux solutions. x1=', x1, '   x2=', x2)
          FinSi
     FinSi
Fin
```

Le programme PASCAL :

```
 1  Program val_absolue_carre;
 2  Uses wincrt ;
 3  var
 4      a, b, c, delta, x1, x2 : real ;
 5  Begin
 6      Read(a, b, c);  delta:=sqr(b)-4*a*c;
 7      if delta<0 then
 8           Write('Pas de solutions réelles')
 9      else
10           if delta = 0 then
11            begin
12                x1:= -b/(2*a);
13                Write('Solution double x1 = x2 = ', x1:6:2);
14            End
16           Else
17            Begin
18                X1:=(-b - sqrt(delta))/(2*a) ;
19                X2:=(-b + sqrt(delta))/(2*a) ;
20                Write('Deux solutions. x1=',x1:6:2,'x2=', x2:6:2);
21            End;
22  End.
```

Organigramme :



Exercice 10 : Déroulement d'un algorithme

Soit l'algorithme suivant :

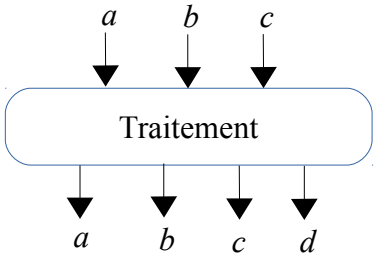
```
Algorithme Exo10
        Variables
                a, b, c, d : entier
Début
        lire(a, b, c) ; d ← 0
        Si a > b alors
                Si a < c alors
                        b ← b+c
                        a ← b-c
                        c ← a+b
                Sinon
                        Si b=c alors
                                c ← c+3
                        FinSi
                        d ← a-b+c
                FinSi
        Sinon
                Si a>c alors
                        a ← 2*b
                FinSi
                b ← b+c
        FinSi
        d ← sqr(a+b+d)
        écrire(a, b, c, d)
Fin
```

– - Faire le déroulement de cet algorithme et déduire les valeurs finale des variables *a, b, c* et *d* dans chacun des cas suivantes :

1) *a=1 ; b=2 ; c=1* ***2)*** *a=2 ; b=1 ; c=1* ***3)*** *a=1 ; b=1 ; c=-2* ***4)*** *a=2 ; b=1 ; c=3*

– Écrire le programme Pascal correspondant et l'exécuter.

Réponse



Nous avons dans l'algorithme *3* variables d'entrée et *4* variables de sortie. Pour dérouler l'algorithme, on doit connaître la valeur de chaque variable d'entrée.

– ***Déroulement***

1) *a=1 ; b=2 ; c=1*

Variables / Instructions	*a*	*b*	*c*	*d*
Lire(a, b, c) *d ← 0*	*1*	*2*	*1*	*0*
Si a>b → false *=> Sinon*	*1*	~~*2*~~	*1*	*0*
Si a>c → false *b ← b+c*	*1*	*3*	*1*	~~*0*~~
d←sqr(a+b+d)	*1*	*3*	*1*	*16*

Les valeurs finales sont : *a=1, b=3, c=1* et *d=16*

2) a=2 ; b=1 ; c=1

Variables / Instructions	*a*	*b*	*c*	*d*
Lire(a, b, c) *d ← 0*	*2*	*1*	*1*	*0*
Si a>b → true *si a<si → false* *=> sinon* *si b=c → true*	*2*	*1*	~~*1*~~	
c ← c+3	*2*	*1*	*4*	*0*
d ← a-b+c=2-1+4 = 5	*2*	*1*	*4*	~~*5*~~
d←sqr(a+b+d)=8^2	*2*	*1*	*4*	*64*

Les valeurs finales sont : *a=2, b=1, c=4* et *d=64*

3) a=1 ; b=1 ; c=-2

Variables / Instructions	*a*	*b*	*c*	*d*
Lire(a, b, c) *d ← 0*	~~*1*~~	*1*	*-2*	*0*
Si a>b → false *=> sinon* *si a>c → true* *a ← 2*b*	*2*	~~*1*~~	*-2*	*0*
b ← b + c	*2*	*-1*	*-2*	~~*0*~~
d←sqr(a+b+d)=-1^2	*2*	*-1*	*-2*	*1*

Les valeurs finales sont : *a=2, b=-1, c=-2* et *d=1*

4) a=2 ; b=1 ; c=3 **Les valeurs finales sont :** *a=2, b=-1, c=-2* et *d=1*

Variables / Instructions	*a*	*b*	*c*	*d*
Lire(a, b, c) *d ← 0*	*2*	~~*1*~~	*3*	*0*
Si a>b → true *si a< c → true* *b ← b + c*	~~*2*~~	*4*	*3*	*0*
a ← b - c	*1*	*4*	~~*3*~~	*0*

$c \leftarrow a + b$	*1*	*4*	*5*	*θ*
$d \leftarrow sqr(a+b+d)=-1^2$	*1*	*4*	*5*	*25*

- Le programme PASCAL correspondant :

```
1  Program Exo8;
2  Uses wincrt ;
3  var
4      a, b, c, d : real ;
5  Begin
6      Read(a, b, c);
7      d:=0;
8      if a>b then
9           if a<c then
10           begin
11               b:=b+c;
12               a:=b-c;
13               c:=a+b;
14           end
16         else
17           begin
18               if b=c then
19                    c:=c+3;
20               d:=a-b+c;
21           end
22     else
23      begin
24           if a>c then
25                a:=2*b;
26           b:=b+c;
17       end
28     D:=sqr(a+b+d);
29     Write('a=' , a, '  b=' , b, '  c=' , c, '  d=', d);
30 End.
```

Pas de point-virgule avant ***else***

Remarques :

- Il faut jamais mettre de point-virgule pour l'instruction qui précède ***else***.
- Si le bloc de ***if*** contient plusieurs instructions (plus d'une instruction) on délimite, obligatoirement, ces instructions par ***begin*** et ***end***
- Si le bloc de else contient plusieurs instructions (plus d'une instruction) on délimite, obligatoirement, ces instructions par ***begin*** et ***end***
- Si le bloc de ***if*** ou ***else*** contient une seule instruction, on peut enlever ***begin*** et **end**
- Dans le langage PASCAL, il n y a pas le mot clé *endIf* ou *endElse*

Exercice 11 : Évaluation des expressions arithmétiques et logiques

Une expression mathématiques peut être soit arithmétiques (donne un résultat numérique) ou bien logique (donne un résultat booléen). Évaluer un expression veut dire déterminer la valeur finale (le résultat) de cette expression après avoir remplacé toutes les variables avec leurs valeurs respectives. Cette évaluation doit respecter l'ordre de calcul des différents opérations utilisées dasn l'expression. On a 6 niveaux de priorité d'opérateurs :

1) Les parenthèse : on évalue tout d'abord les parenthèses les plus profondes
2) Les fonctions : comme par exemple *sqr, sqrt, abs, cos, sin, exp,* etc.
3) Les opérateurs unaires : - unaire et *not*
4) Multiplication *, division /, Div (division entière), Mod (reste de division) et le And (et logique)
5) Addition +, Soustraction –, OR (le ou logique)
6) Opérateurs relationnels (de comparaison) : < , > , =, >=, <=, <> (différent)

Dans le cas ou deux opérateurs ont la même priorité on calcul tout d'abord celui à gauche (de gauche à droite).

Expression 1 : *a + b / c + ((d / 3 + 4) / 3 + a) / b*

Expression 2 : *(a > b) or not (c >= d) and (b < c)*

Avec *a=1 ; b=2 ; c=4 ; d=6.*

Réponse

Expression 1

$$\underset{(7)}{a} + \frac{\underset{(5)}{b}}{\underset{(7)}{c}} + \left(\left(\frac{\underset{(1)}{d}}{\underset{(2)}{3}} + \underset{(4)}{4} \right) / \underset{(6)}{3} + \underset{(8)}{a} \right) / b$$

Dans l'expression 1, il y a huit opérateurs, donc on trouvera ***8*** étapes de calcul.

L'ordre des opérations sera défini donc comme suit :

$$\underset{(7)}{a} + \underset{(5)}{b} / \underset{(8)}{c} + ((\underset{(1)}{d} / \underset{(2)}{3} + \underset{(4)}{4}) / \underset{(6)}{3} + \underset{(8)}{a}) / b$$

Expression 2

$$\underline{(a > b)}\ or\ not\ \underline{(c >= d)}\ and\ \underline{(b < c)}$$
$$^{(1)}\ False \qquad ^{(2)}\ Flase \qquad ^{(3)}\ TRUE$$
$$^{(4)}\ TRUE$$
$$^{(5)}\ TRUE$$
$$^{(6)}\ TRUE$$

Dans l'expression 1, il y a 6 opérateurs, donc on trouvera ***6*** étapes de calcul.

L'ordre des opérations sera défini donc comme suit :

$$\underset{(1)}{(a > b)}\ \underset{(6)}{or}\ \underset{(4)}{not}\ \underset{(2)}{(c >= d)}\ \underset{(5)}{and}\ \underset{(3)}{(b < c)}$$

Exercice 12 : Calculer une somme d'une manière itératives

1- Écrire un algorithme pour calculer la somme suivante : $S = 1^2 + 3^2 + 5^2 + \ldots + (2m+1)^2$. Avec *m* donné. Résoudre l'exercice en utilisant chacune des boucles : ***(a)*** la boucle Pour ***(b)*** la boucle Tant-que ***(c)*** la boucle Répéter et déduire la plus appropriée dans ce cas.

2- Effectuer le déroulement (trace d'exécution) de l'algorithme pour m=4 dans chacun des cas ***(a)***, ***(b)*** et ***(c)***.

3- Traduire chacun des algorithmes en programme PASCAL

Remarque : On peut utiliser une boucle de *0* à *m* pour tous les cas. Une autre solution consiste à utiliser une boucle de *1* à *(2m+1)* mais en incrémentant le compteur de *2*.

Réponse

1- Pour faire une somme répétitive (itératives) il faut tout d'abord déduire le terme général (terme itératif). On peut écrire la somme *S* comme suit :

$S = 1^2 + 3^2 + 5^2 + \ldots + (2m+1)^2$ Qu'on peut écrire sous la forme abrégée suivante :

$$S = \sum_{i=0}^{m} (2i+1)^2$$

- Le terme répétitive (générale) est : $(2i+1)^2$
- *i* c'est un indice qui varie de la valeur initiale *0* jusqu'à la valeur finale *m*.
- Le symbole de la somme peut être remplacé, très facilement, par la boucle *Pour* comme suit : *Pour i ← 0 à m faire*

- à chaque itération de la boucle *Pour* on ajoute à *S* le terme répétitive : $(2i+1)^2$ comme suit : $S \leftarrow S + \mathbf{sqr(2*i + 1)}$ ce qui est en gras est le terme itérative (répétitive ou générale)

Bien évidemment, pour calculer la valeur de *S* on aura besoin de la valeur de *m*. *m* c'est valeur à introduire à l'algorithme (une variable d'entrée => *lire(m)*). Par contre *S* est le résultat à calculer (une variable de sortie => *écrire(s)*)

a) La solution en utilisant la boucle *Pour*

```
Algorithme Exo12_Pour
      Variables
            m, s, i:entier
Début
      lire(m) ;
      S ← 0 ;
      Pour i←0 à m faire
            S ← S + sqr(2*i+1)
      FinPour
      écrire(S)
Fin
```

b) La solution en utilisant la boucle *Tant-Que*

```
Algorithme Exo12_TantQue
      Variables
            m, s, i:entier
Début
      lire(m) ;
      S ← 0 ;
      i ← 0 ;
      Tant-que i<= m faire
            S ← S + sqr(2*i+1)
            i ← i + 1
      FinTant-Que
      écrire(S)
Fin
```

c) La solution en utilisant la boucle *Répéter*

```
Algorithme Exo12_Repeter
      Variables
            m, s, i:entier
Début
      lire(m) ;
      S ← 0 ;
      i ← 0 ;
      Répéter
            S ← S + sqr(2*i+1)
            i ← i + 1
      Jusqu'à i>m
      écrire(S)
Fin
```

Remarques :

- La boucle *Pour* fonctionne avec un compteur (indice) qui représente une variable entière (le compteur doit être obligatoirement une variable entière)

- Dans la boucle *Pour* on spécifie la *valeur initiale* et la *valeur finale* (deux valeurs entières) du compteur
- Si la *valeur finale* est plus petite que la *valeur initiale* donc **on n'exécute pas la boucle *Pour*** (*nombre d'itérations = 0*)
- Si la *valeur finale* est supérieur à la *valeur initiale* donc on exécute la boucle *Pour* avec un nombre d'itérations = ***<valeurFinale> - <valeurInitiale> + 1***. Dans l'exemple précédent, la boucle *Pour* sera exécutée pour *(m+1) itérations* (Si *m=4* ça sera alors *5 itérations*).
- Les deux boucles *Tant-que* et *Répéter* fonctionnent avec des conditions (expressions logiques qui donnent soit True soit False).
- D'une manière générale, traduire une boucle *Pour* à une boucle *Tant-que* se fait comme suit :

```
Pour <cpt>←<vi> à <vf> faire
      <instruction(s)>
FinPour
```

```
<cpt> ← <vi>
Tanque-Que <cpt> <= <vf> faire
      <instruction(s)>
      <cpt> ← <cpt> + 1
FinTant-que
```

Tel-que :

<*cpt*> : c'est une variable entière (le compteur de la boucle *Pour*)

<*vi*> : c'est une valeur entière (la valeur initiale de la boucle *Pour*)

<*vf*> : c'est une valeur entière (la valeur finale de la boucle *Pour*).

- Donc, pour transformer la boucle *Pour* à une boucle *Tant-que* il faut ajouter deux instructions : l'initialisation du compteur à la valeur initiale et l'incrémentation du compteur (de 1) à la fin de chaque itération. Bien évidemment, le boucle *Tant-que* contient la condition compteur inférieur ou égale à la valeur finale.
- Toute boucle *Pour* peut être remplacée par une boucle *Tant-que*. Cependant, on peut trouver des boucles *Tant-que* qu'on ne peut pas (ou bien difficile à) remplacer par une boucle *Pour*.
- Toute boucle *Tant-que* peut être remplacée par une boucle *Répéter* et vice-versa.
- *La boucle Tant-que et Répéter sont équivalentes*.
- La condition de la boucle *Tant-que* est évaluée au début de l'itération : si la condition est vraie on *entre à l'itération* sinon on *quitte* la boucle.
- La condition de la boucle *Répéter* est évaluée à la fin de l'itération : si la *condition est vraie* on *quitte* la boucle sinon on réalise une autre itération (*réitère : re-boucle*).
- Pour la boucle *Tant-que* la condition représente le *test d'entrée*, par contre, pour la boucle

Répéter la condition représente le *test de sortie*.

- Pour remplacer la boucle *Tant-que* par la boucle *Répéter*, il suffit de nier la condition de la boucle *Tant-que* et bien-sûr de respecter la syntaxe des deux boucles comme suit :

```
Tant-que <Condition_1> faire
      <instruction(s)>
FinTant-que
```

```
Répéter
      <instruction(s)>
Jusqu'à <Condition_2>
```

Tel-que : <*Condition_2*> et la négation de <*Condition_2*>, autrement dit :

<*Condition_2*> = ***Not*** (<*Condition_1*>)

2- Le déroulement pour m=4

a) La boucle *Pour*

Variables / Instructions	*m*	*i*	*s*
Lie (m)	*4*	*/*	*/*
S ← 0	*4*	*/*	*0*
Pour i=0	*4*	*0*	~~*0*~~
*S ← S + sqr(2*i+1) = 0 + sqr(2*0+1) = 1^2*	*4*	~~*0*~~	*1^2= 1*
Pour i=1	*4*	*1*	~~*1*~~
*S ← S + sqr(2*i+1) = 1^2 + sqr(2*1+1) = 1 + 3^2 = 10*	*4*	~~*1*~~	*1^2 + 3^2 = 10*
Pour i=2	*4*	*2*	~~*10*~~
*S←S+sqr(2*i+1)=10+sqr(2*2+1)= 10+5^2 = 35*	*4*	~~*2*~~	*1^2 + 3^2 +5^2 = 35*
Pour i=3	*4*	*3*	~~*35*~~
*S←S+sqr(2*i+1)=35+sqr(2*3+1)= 35+7^2 = 35+49 = 84*	*4*	~~*3*~~	*1^2 + 3^2 +5^2 +7^2 = 84*
Pour i=4 (la dernière itération i=m)	*4*	*4*	~~*84*~~
*S←S+sqr(2*i+1)=84+sqr(2*4+1)= 84+9^2 = 84+81 = 165*	*4*	*4*	*1^2 + 3^2 +5^2 +7^2 + 9^2= 165*
Écrire (s)	*4*	*4*	*165*

La valeur finale de l'algorithme est ***S = 165***

Remarque : *Les deux couleurs gris clair / foncé sont utilisé pour montrer et séparer les itérations de la boucle Pour.*

b) La boucle *Tant-Que*

Variables / Instructions	*m*	*i*	*s*
Lie (m)	*4*	*/*	*/*
S ← 0 *i ← 0*	*4*	*0*	*0*
Tant-que i<=m :0≤4 =>True=>entrer à la boucle	*4*	*0*	~~*0*~~
*S ← S + sqr(2*i+1) = 0 + sqr(2*0+1) = 1^2*	*4*	~~*0*~~	*1^2= 1*
i ← i +1	*4*	*1*	*1*
Tant-que i<=m :1≤4 =>True=>entrer à la boucle	*4*	*1*	~~*1*~~
*S ← S+sqr(2*i+1) = 1+sqr(2*1+1) = 1+3^2 = 10*	*4*	~~*1*~~	*1^2 + 3^2 = 10*
i ← i +1	*4*	*2*	*10*
Tant-que i<=m :2≤4 =>True=>entrer à la boucle	*4*	*2*	~~*10*~~
*S←S+sqr(2*i+1)=10+sqr(2*2+1)= 10+5^2 = 35*	*4*	~~*2*~~	*1^2 + 3^2 +5^2 = 35*
i ← i + 1	*4*	*3*	*35*
Tant-que i<=m :3≤4 =>True=>entrer à la boucle	*4*	*3*	~~*35*~~
*S←S+sqr(2*i+1)=35+sqr(2*3+1)=35+7^2=84*	*4*	~~*3*~~	*1^2 + 3^2 +5^2 +7^2 = 84*
i ← i + 1	*4*	*4*	*84*
Tant-que i<=m :4≤4 =>True=>entrer à la boucle	*4*	*4*	~~*84*~~
*S←S+sqr(2*i+1)=84+sqr(2*4+1)=84+9^2= 165*	*4*	*4*	1^2 + 3^2 +5^2 +7^2 + 9^2= *165*
i ← i + 1	*4*	*5*	165
Tant-que i<=m :5≤4 =>False=>quitter la boucle	*4*	*5*	165
Écrire (S)	*4*	*5*	165

La valeur finale de l'algorithme est ***S = 165***

Remarque : Comme dans le cas de la boucle Pour, la boucle Tant-que possède 5 itérations. Par contre, le teste i<=m est réalisé 6 fois (le dernier test n'aboutit pas => donne false pour quitter la boucle).

c) La boucle *Répéter*

Variables / Instructions	*m*	*i*	*s*
Lie (m)	*4*	*/*	*/*
S ← 0	*4*	*0*	*0*

i ← 0			
Répéter	*4*	*0*	~~*0*~~
*S ← S + sqr(2*i+1) = 0 + sqr(2*0+1) = 1^2*	*4*	~~*0*~~	*1^2= 1*
i ← i +1	*4*	*1*	*1*
Jusqu'à i>m :1>4 =>False=>refaire la boucle	*4*	*1*	~~*1*~~
*S ← S+sqr(2*i+1) = 1+sqr(2*1+1) = 1+3^2 = 10*	*4*	~~*1*~~	*1^2 + 3^2 = 10*
i ← i +1	*4*	*2*	*10*
Jusqu'à i>m :2>4 =>False=>refaire la boucle	*4*	*2*	~~*10*~~
*S←S+sqr(2*i+1)=10+sqr(2*2+1)= 10+5^2 = 35*	*4*	~~*2*~~	*1^2 + 3^2 +5^2 = 35*
i ← i + 1	*4*	*3*	*35*
Jusqu'à i>m :3>4 =>False=>refaire la boucle	*4*	*3*	~~*35*~~
*S←S+sqr(2*i+1)=35+sqr(2*3+1)=35+7^2=84*	*4*	~~*3*~~	*1^2 + 3^2 +5^2 +7^2 = 84*
i ← i + 1	*4*	*4*	*84*
Jusqu'à i>m :4>4 =>False=>refaire la boucle	*4*	*4*	~~*84*~~
*S←S+sqr(2*i+1)=84+sqr(2*4+1)=84+9^2= 165*	*4*	*4*	1^2 + 3^2 +5^2 +7^2 + 9^2= *165*
i ← i + 1	*4*	*5*	165
Jusqu'à i>m :5>4 =>True=> quitter la boucle	*4*	*5*	165
Écrire (S)	*4*	*5*	165

La valeur finale de l'algorithme est ***S = 165***

3- Traduction des algorithmes en programme PASCAL

a) La boucle Pour

```
1  Program exo12_Pour;
2  Uses wincrt ;
3  var
4      m, i, s : integer ;
5  Begin
6      Write('Donner la valeur de m : '); Read(m);
7      S := 0;
8      For i:=0 to m do
9           S := S + sqr(2*i+1);
10     Write('La somme S = ', S);
11 End.
```

b) La boucle Tant-Que

```
1  Program exo12_Tantque;
2  Uses wincrt ;
3  var
4      m, i, s : integer ;
5  Begin
6      Write('Donner la valeur de m : ');
7      Read(m);
8      S := 0;
9      i:=0 ;
10     while i<=m do
11     begin
12           S := S + sqr(2*i+1);
13           i:=i+1;
14     end;
15     Write('La somme S = ', S);
16 End.
```

c) La boucle Répéter

```
1  Program exo12_Repeter;
2  Uses wincrt ;
3  var
4      m, i, s : integer ;
5  Begin
6      Write('Donner la valeur de m : ');
7      Read(m);
8      S := 0;
9      i:=0 ;
10     Repeat
11           S := S + sqr(2*i+1);
12           i:=i+1;
13     Until i>m;
14     Write('La somme S = ', S);
15 End.
```

Remarques :

– Pour le programme (avec la boucle Pour), on n'a pas mis *begin … end ;* ceci à cause de l'existence d'une seule instruction à l'intérieur de la boucle *Pour*. Rien n'empêche de les mettre comme suit :

```
For i:=0 to m do
Begin
          S := S + sqr(2*i+1);
End;
```

– *Begin .. end ;* se mettent après la structure de contrôle (soit *if, else, for, while*).

– S'il y a plus d'une instruction (nombre d'instructions > = 2), dans ce cas le *begin … end ;* sont obligatoires.

– Pour la boucle *Repeat*, on mis pas de *begin … end ;* puisque le bloc d'instruction de la boucle *Repeat* est délimité par les deux mots clé : *Repeat* et *Until*

– Il faut jamais mettre de point-virgule après *do* (Pour la boucle *While* ça génère une **boucle infinie**)

Exercice 13 : Calculs itératives - sommes, produits

Écrire un algorithme et le programme PASCAL pour chacun des cas suivants avec *N* donnée :

a) Calculer le factoriel de *N* soit : $1\times2\times3\times...\times N=N!$

b) Écrire un programme PASCAL qui permet de calculer le somme : $\frac{1}{2}+\frac{2}{3}+\frac{3}{4}+...+\frac{n}{n+1}$

c) Calculer la valeur de π de sachant que $\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+...(Faire\,100\,itérations)$

Réponse

a) On suppose que *P = N !*

Donc : $P=1\times2\times3\times4\times...\times N$

La forme abréagie : $P=\prod_{i=1}^{N} i$ (Le produit des *i* tel que *i* allant de *1* jusqu'à *N*) => le facteur général est : *i*. on aura alors la boucle *Pour* qui possède un compteur *i* avec la valeur initiale *1* et la valeur finale *N*. On multiplie P par le facteur général (facteur répétitif) *i* Comme suit :

```
Pour i ← 1 à N Faire
      P ← P * i;
FinPour;
```

Il reste à s'assurer que la valeur de *P* soit initialement neutre (il faut initialiser *P* à *1*. (*P←1*)).

```
Algorithme Exo13_a_factoriel
      Variables
            N, P, i : entier
Début
      lire(N) ;
      P ← 1 ;
      Pour i←1 à N faire
            P ← P * i
      FinPour
      écrire(P)
Fin
```

```
1  Program exo13_a_factoriel;
2  Uses wincrt;
3  var
4      N, i, P : integer ;
5  Begin
6      Write('Donner la valeur de N : ');
7      Read(N);
8      P:=1;
9      for i:=1 to N do
10         P := P * i;
11     Write('N! = ', P);
12 End.
```

b) On suppose que $S=\frac{1}{2}+\frac{2}{3}+\frac{3}{4}+...+\frac{n}{n+1}$

La forme abréagie : $S=\sum_{i=1}^{n}\frac{i}{i+1}$ (La somme termes $\frac{i}{i+1}$ tel que *i* allant de *1* jusqu'à *n*) => le terme général est : $\frac{i}{i+1}$. On aura alors la boucle *Pour* qui possède un compteur *i* avec la valeur initiale *1* et la valeur finale *n*. On ajoute à *S* par le terme général (terme répétitif) $\frac{i}{i+1}$ Comme suit :

```
Pour i ← 1 à n Faire
      S ← S * i/(i+1);
FinPour;
```

Il reste à s'assurer que la valeur de *S* soit initialement neutre (il faut initialiser *S* à *0*. (S←*0*)).

```
Algorithme Exo13_b_somme
      Variables
            n, i : entier
            S : réel
Début
      lire(N) ;
      S ← 0 ;
      Pour i←1 à n faire
            S ← S + i/(i+1)
      FinPour
      écrire(S)
Fin
```

```
1  Program exo13_b_somme;
2  Uses wincrt;
3  var
4      n, i : integer ;
5      S : real;
6  Begin
7      Write('Donner la valeur de N : ');
8      Read(N);
9      S:=0;
10     for i:=1 to n do
11     begin
12           S := S + i/(i+1);
14     end;
15     Write('La somme = ', S:6:3);
16 End.
```

c) On met $S=\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+...(Faire\,100\,itérations)$

Donc aura le terme générale suivant : $S=\sum_{k=0}^{99}\frac{(-1)^k}{2*k+1}$

Ainsi on aura une boucle *Pour* avec un compteur *k* allant de 0 jusqu'à 99 (100 itérations=*99–0+1*) Il reste à calculer $(-1)^k$. Il suffit de mettre une variable *signe* = $(-1)^k$. Cette variable bascule entre les deux valeur *1* et *-1*. Pour faire ça, il suffit de multiplier *signe* par *-1*. comme suit :

signe ← - *signe*. Et de cette façon, si *signe* = *1* donc il devient *-1*. Et si *signe* = *-1* il devient *1*.

L'objectif de l'algorithme et de calculer la valeur approximatif de $\pi = 4 * S$.

```
Algorithme Exo13_c_PI
      Variables
            k,signe : entier
            S, P : réel
Début
      S ← 0 ; signe ← 1
      Pour k←0 à 99 faire
            S←S+signe/(2*k+1)
            Signe ← - signe
      FinPour
      P ← 4*S;
      Écrire(P)
Fin
```

```
1  Program exo13_c_PI;
2  Uses wincrt;
3  var
4      k, signe : integer ;
5      S, P : real;
6  Begin
7      S:=0;
8      Signe:=1;
9      for k:=0 to 99 do
10     begin
11           S := S + signe/(2*i+1);
12           Signe:= - signe;
14     end;
15     P:=4*S;
16     Write('P = ', P:6:3);
17 End.
```

Exercice 14 : Calcul de somme en fonction de n et x

1) Écrire un algorithme pour calculer la somme suivante :

$S = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + ...+ \left((n+1)\,itérations\right)$ avec la valeur de n et de x données.

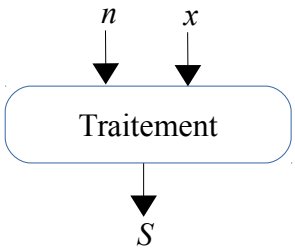
2) Dérouler l'algorithme jusqu'à $n=4$ (en fonction de x)

3) Traduire l'algorithme en programme.

Réponse

1) Algorithme pour le calcul de S

Pour cette algorithme, on veut calculer la valeur de S. Pour cela, il faut introduire la valeur de x et de n. Donc nous avons une variable de sortie S et deux variable d'entrée n et x.



Ainsi, au début de l'algorithme on aura *lire(n, x)* et à la fin de l'algorithme on aura *écrire (S)*.

La question qui se pose est quel est le traitement à réaliser pour avoir la valeur de S.

Nous avons : $S = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + ...+ \left((n+1)\,itérations\right) = \sum_{i=0}^{n} \frac{x^{2i+1}}{2i+1}$ La somme sera de $i=0$

jusqu'à *n*. (*n+1* itérations)

Pour le calculer de puissance x^{2i+1}, on mit $P = x^{2i+1}$. Initialement $P = x$. ensuite x^3, après x^5 …

Donc il suffit de faire $P \leftarrow P * sqr(x)$

```
Algorithme Exo_14
     Variables
          i, n : entier
          S, x, P : réel
Début
     Lire (n, x)
     S ← 0
     P ← x
     Pour i←0 à n faire
          S ← S+ P / (2*i+1)
          P ← P * sqr(x)
     FinPour
     Écrire(S)
Fin
```

2) Déroulement de l'algorithme pour $n = 4$ et x quelconque.

Variables / Instructions	*n*	*x*	*i*	*P*	*s*
Lie (n, x)	*4*	*x*	*/*	*/*	*/*
$S \leftarrow 0$ $P \leftarrow x$	*4*	*x*	*/*	*x*	*0*
Pour i=0	*"*	*"*	*0*	*"*	*"*
$S \leftarrow S + P / (2i+1)$	*"*	*"*	*"*	*"*	*x*
$P \leftarrow P * sqr(x)$	*"*	*"*	*"*	x^3	*x*
Pour i=1	*"*	*"*	*1*	*"*	*"*
$S \leftarrow S + P / (2i+1)$	*"*	*"*	*"*	*"*	$x + x^3/3$
$P \leftarrow P * sqr(x)$	*"*	*"*	*"*	x^5	*"*
Pour i=2	*"*	*"*	*2*	*"*	*"*
$S \leftarrow S + P / (2i+1)$	*"*	*"*	*"*	*"*	$x + x^3/3 + x^5/5$
$P \leftarrow P * sqr(x)$	*"*	*"*	*"*	x^7	*"*
Pour i=3	*"*	*"*	*3*	*"*	*"*
$S \leftarrow S + P / (2i+1)$	*"*	*"*	*"*	*"*	$x + x^3/3 + x^5/5 + x^7/7$
$P \leftarrow P * sqr(x)$	*"*	*"*	*"*	x^9	*"*
Pour i=4 (dernière itération i=n)	*"*	*"*	*4*	*"*	*"*
$S \leftarrow S + P / (2i+1)$	*"*	*"*	*"*	*"*	$x + x^3/3 + x^5/5 + x^7/7 + x^9/9$

$P \leftarrow P * sqr(x)$	"	"	"	x^{11}	"
Écrire (S)	"	"	"	"	$x + x^3/3 + x^5/5 + x^7/7 + x^9/9$

Donc la valeur finale S = $x + x^3/3 + x^5/5 + x^7/7 + x^9/9$ (le résultat est en fonction de x puisqu'on n'a pas donner une valeur pour x).

3) Traduction de l'algorithme en programme

```
1  Program exo_14;
2  Uses wincrt;
3  var
4      n, i : integer ;
5      x, s, p : real;
6  Begin
7      Read(n, x);
8      S:=0;
9      P:=x;
10     for i:=0 to n do
11     begin
12          S := S + P/(2*i+1);
14          P := P * sqr(x);
15     end;
16     Write('S = ', S:6:3);
17 End.
```

Exercice 15 : Calcul de salaire pour N individus

Supposant qu'une grandeur V en dinars est calculée en fonction de la situation familiale de l'individu. Soit $V = VB + R$, sachant que VB est une donnée et R est calculée en fonction de la situation familiale comme suit :

$$R = \begin{cases} VB \times 5\ \% & \text{si SF = 0 (célibataire)} \\ VB \times 3\ \% & \text{si SF = 1 (divorcé) et NF = 0} \\ VB \times 10\ \% & \text{si SF = 1 (divorcé) et NF > 0} \\ VB \times 10\ \% & \text{si SF = 2 (marié) et NF = 0} \\ VB \times 15\ \% & \text{si SF = 2 (marié) et } 1 \le \text{NF} \le 3 \\ VB \times 20\ \% & \text{si SF = 2 (marié) et NF > 3} \end{cases}$$

1) Écrire un algorithme permettant de saisir les valeurs les données *VB, SF, NF* et calcule la valeur de *V* pour *N* individus (*N* donnée). Sachant que V, VB et R sont des réels, N, SF, NF sont des entiers naturels. (Attention dans le cas où *SF=0*, l'algorithme ne doit pas lire *NF*). Faire les lectures et traitements dans une boucle de *1* à *N*.

2) Traduire l'algorithme en programme.

Réponse

1) Algorithme

```
Algorithme Exo_15
     Variables
          i, N : entier
          SF, NF : entier
          VB, R, V : réel
Début
     Lire (N)
     Pour i ← 1 à N faire
          Lire (VB, SF);
          si (SF = 1) OR (SF = 2) alors
               Lire (NF)
          FinSi

          Si SF = 0 alros
               R ← VB * 0.05
          Sinon
               Si SF = 1 alors
                    Si NF = 0 alros
                         R ←  VB * 0.03
                    Sinon
                         R ←  VB * 0.10
                    FinSi
               Sinon
                    Si SF = 2 alors
                         Si NF = 0 alors
                              R ←  VB * 0.10
                         Sinon
                              Si (NF >= 1) AND (NF <= 3) alors
                                   R ←  VB * 0.15
                              Sinon
                                   R ←  VB * 0.20
                              FinSi
                         FinSi
                    FinSi
               FinSi
          V ← VB + R
          Écrire(V)
     FinPour
Fin
```

Dans cet algorithme, on affichera *N* valeurs pour *V* (salaires des individus).

2) Traduction de l'algorithme en programme

```
Program exo_15;
Uses wincrt;
var
    N, i       : integer;
    SF, NF     : integer;
    VB, R, V   : real;
Begin
    Write ('Donner le nombre d'individus : ');
    Read(N);

    for i:=1 to N do
     beign
         Write ('Introduire le salaire de base : ');
         Read (VB);
         Write ('Indiquer la situation familiale : ');
         Read(SF);
         if (SF = 1) OR (SF = 2) then {Divorcé ou Marié}
          begin
              Write('Introduire le nombre d'enfants : ') ;
              Read (NF) ;
          end;

         if SF = 0 then
              R:= VB * 0.05
         else
              if SF = 1 then
                   if NF = 0 then
                        R:= VB * 0.3
                   else
                        R:= VB * 0.10
              Else
                   if SF = 2 then
                        if NF = 0 then
                             R:= VB * 0.10
                        else
                             if (NF >= 1) AND (NF <=3 ) then
                                  R:= VB * 0.15
                             else
                                  R:= VB * 0.20
         V:=VB + R;
         Writeln('Le salaire de l''individu ', i, 'est :', V:6:2);
     end;
End.
```

Exercices Supplémentaires :

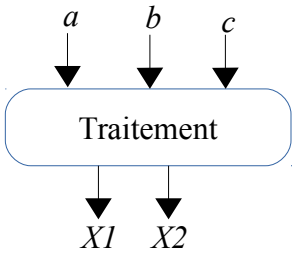
Exercice 16 : Solution d'une équation de deuxième degré

Écrire un algorithme (et programme) pour résoudre une équation du second ordre $ax^2+bx+c=0$

dans C (les nombres complexes). Dans le cas où le discriminant est négatif, on calcule la partie réelle X et la partie imaginaire Y en considérant la racine du discriminant en valeur absolue. Les solutions sont alors X, Y et X, Z avec Z=-Y. (Solution conjuguée).

Solution

L'algorithme possède trois variables d'entrée *a, b* et *c* et deux variables de sortie *X1* et *X2*. (deux solutions réelles différentes (delta > 0), solution double (delta = 0) et deux solutions complexes *Z1* = *X1* + *i X2* et *Z2* = *X1* - *i X2*




```
Algorithme Exo_16
     Variables
          a, b, c : réel
          delta, X1, X2 : réel
Début
     Lire (a, b, c)
     Tant-que a = 0 faire
          lire(a);
     finTant-que

     delta ← sqr(b) – 4*a*b;
     si delta = 0 alors
          X1 = -b/(2*a)
          Écrire('Solution double X1 = X2 =', X1);
     sinon
          si (delta > 0) alors
               X1 = (-b-sqr(delta))/(2*a) ;
               X2 = (-b+sqr(delta))/(2*a) ;
               Écrire('Deux solutions réelles X1=', X1, '  X2=', X2);
          Sinon
               X1 = -b/(2*a);
               X2 = sqr(-delta)/(2*a)
               Écrire('Deux solutions Complexes : ');
               Écrire('Z1 = ',X1, ' + ', X2, ' i');
               Écrire('Z2 = ',X1, ' - ', X2, ' i');
          FinSi
     FinSi
Fin
```

```
Program exo_16;
Uses wincrt;
var
    a, b, c : real;
    Delta   : real;
    X1, X2  : real;
Begin
    Write ('Donner la valeur de a, b et c : ');
    Read(a, b, c);
    While a = 0 do {s'assurer que la valeur de a <> 0}
    begin
         Write('Veuillez donner une valeur non nulle pour a : ');
         Read(a);
    end;

    Delta := sqr(b) - 4*a*c;
    if delta = 0 then
     begin
         X1:= -b/(2*a);
         Write ('Une solution double X1=X2= ', X1:6:2);
     end
    else
         if delta > 0 then
          begin
              X1 := (-b-sqr(delta)) / (2*a);
              X2 := (-b+sqr(delta)) / (2*a);
              Write ('Deux solutions réelles :');
              Write ('X1=', X1:6:2, 'X2=', X2:6:2);
          end
         Else {delta est négatif : deux solutions complexes}
          begin
              X1 := -b / (2*a);
              X2 := sqr(-delta) / (2*a);
              Write ('Deux solutions complexes :');
              Write ('Z1=', X1:6:2, ' + ', X2:6:2, 'i');
              Write ('Z2=', X1:6:2, ' - ', X2:6:2, 'i');
          end;
End.
```

Exercice 17 : Sommes itératives

Écrire les algorithmes et programmes correspondants pour chacun des cas suivants avec la valeur de N et x données : (Dérouler chaque algorithme pou N=5).

(a) $S1 = \frac{x^2}{2} - \frac{x^4}{4} + \frac{x^6}{6} - \frac{x^8}{8} + \ \ldots \ \pm \frac{x^{2n}}{2n}$

(b) $S2 = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ \ldots\ + \frac{x^{2n+1}}{(2n+1)!}$

Réponse

(a) Il faut écrire la somme *S1* sous forme abrégée. Nous avons :

$S1 = \frac{x^2}{2} - \frac{x^4}{4} + \frac{x^6}{6} - \frac{x^8}{8} + \ \ldots\ \pm \frac{x^{2n}}{2n} = \sum_{i=1}^{n} signe \frac{x^{2i}}{2i}$ Tel-que signe prend, respectivement, les deux valeur *1* et *-1* d'une manière alternative. Donc, le terme général (répétitif) est : $x^{2i} / (2i)$

On réalise un changement de variable, on mettant $P = x^{2i}$. Initialement, lorsque que i=1, $P=x^2$ après passe x^4, x^6, … jusqu'à x^{2n}. Pour réaliser ça, il suffit de multiplie *P* par x^2.

Donc, L'algorithme possède deux variable d'entrée *n* et *x*. et une variable de sortie *S1*.

```
Algorithme Exo17_a
        Variables
                n,i,signe:entier
                x,P,S1 : réel
Début
        Lire (n, x)
        S1 ← 0 ;
        Signe ← 1
        P ← sqr(x);
        Pour i←1 à n faire
                S1←S1 + signe*P/(2*i)
                Signe ← - signe
                P ← P * sqr(x)
        FinPour
        Écrire(S1)
Fin
```

```
1  Program exo17_a;
2  Uses wincrt;
3  var
4      n, i, signe : integer ;
5      x, P, S1 : real;
6  Begin
7      Read(n, x);
8      S1:=0;
9      signe:=1;
10     P:=sqr(x);
11     for i:=1 to n do
12     begin
14         S1 := S1 + signe * P/(2*i);
15         Signe := - signe;
16         P := P * sqr(x);
17     end;
18     Write('S1 = ', S1:6:3);
19 End.
```

(b) Il faut écrire la somme *S2* sous forme abrégée. Nous avons :

$S2 = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ \ldots\ + \frac{x^{2n+1}}{(2n+1)!} = \sum_{i=0}^{n} \frac{x^{2i+1}}{(2i+1)!}$

Donc, le terme général (répétitif) est : $\frac{x^{2i+1}}{(2i+1)!}$

On réalise deux changements de variables, on mettant $P = x^{2i+1}$. Initialement, lorsque que i=0, $P=x$. On met aussi $f=(2i+1)!$. Initialement $f=1$.

La façon comment passer P de la valeur initiale x à x^3, x^5, … jusqu'à x^{2i+1} est de le multiplie à chaque itération par x^2 comme suit $P \leftarrow P * sqr(x)$.

La manière comment passer f de la valeur initiale *1* à *3!* puis *5!* … jusqu'à *(2n+1) !* est de le multiplie à chaque itération par *(2*(i+1))*(2(i+1)+1)* comme suit : $f \leftarrow f * ((2*(i+1))*(2*(i+1)+1))$.

*Pour i=0 ==> f ← f * 2 * 3 = 1 * 2 * 3 = 3!*

*Pour i=1 ==> f ← f * 4 * 5 = 3 ! * 4 * 5 = 5!*

*Pour i=2 ==> f ← f * 4 * 5 = 3 ! * 4 * 5 = 7!*

….

Donc, L'algorithme possède deux variable d'entrée *n* et *x*. et une variable de sortie *S2*.

```
Algorithme Exo17_b
      Variables
            n,i,f:entier
            x,P,S2 : réel
Début
      Lire (n, x)
      S2 ← 0 ;
      f ← 1
      P ← x;
      Pour i←1 à n faire
         S2←S2 + P/f
         P←P * sqr(x)
         f←f*(2*(i+1))*(2*(i+1)+1)
      FinPour
      Écrire(S2)
Fin
```

```
Program exo17_b;
Uses wincrt;
var
    n, i : integer ;
    f:longint; {Un entier plus grand}
    x, P, S2 : real;
Begin
    Read(n, x);
    S2:=0;
    f:=1;
    P:=x;
    for i:=0 to n do
    begin
          S2 := S2 + P/f;
          P:= P * sqr(x);
          f:=f*(2(i+1))* (2*(i+1)+1);
    end;
    Write('S2 = ', S2:6:3);
End.
```

Déroulement pour n=5 et x quelconque

(a) Pour la somme S1

Variables / Instructions	*n*	*x*	*i*	*signe*	*P*	*S1*
Lie (n, x)	*5*	*x*	*/*	*/*	*/*	*/*
S1← 0 *P ← sqr(x)* *signe=1*	*"*	*"*	*/*	*1*	x^2	*0*
Pour i=1	*"*	*"*	*1*	*"*	*"*	*"*
*S ← S + signe * P / (2i)*	*"*	*"*	*"*	*"*	*"*	$x^2/2$

signe ← -signe				*-1*		
*P ← P * sqr(x)*	"	"	"	"	x^4	$x^2/2$
Pour i=2	"	"	*2*	"	"	$x^2/2$
*S ← S + signe * P / (2i)*	"	"	"	"	"	$x^2/2 - x^4/4$
signe ← - signe				*1*		
*P ← P * sqr(x)*	"	"	"	"	x^6	"
Pour i=3	"	"	*3*	"	"	"
*S ← S + signe * P / (2i)*	"	"	"	"	"	$x^2/2 - x^4/4 + x^6/6$
signe ← -signe				*-1*		
*P ← P * sqr(x)*	"	"	"	"	x^8	"
Pour i=4	"	"	*4*	"	"	"
S ← S + signe P / (2i)*	"	"	"	"	"	$x^2/2 - x^4/4 + x^6/6 - x^8/8$
signe ← - signe				*1*		
*P ← P * sqr(x)*	"	"	"	"	x^{10}	"
Pour i=5 (dernière itération i=n)	"	"	*5*	"	"	"
*S ← S + signe * P / (2i)*	"	"	"	"	"	$x^2/2 - x^4/4 + x^6/6 - x^8/8 + x^{10}/10$
signe ← - signe	"	"	"	*-1*	"	
*P ← P * sqr(x)*	"	"	"	"	x^{12}	"
Écrire (S)	"	"	"	"	"	$x^2/2 - x^4/4 + x^6/6 - x^8/8 + x^{10}/10$