Algorithmes et Programmation Structurée

CREE PAR *KAMAL ELAYOUNI*

SOMMAIRE

ALGORITHMES ET PROGRAMMATION STRUCTUREE

1- Les structures de base d'un langage de programmation :

Dans ce qui suit nous allons utiliser un pseudo-langage, comportant toutes les structures de base d'un langage de programmation.

Un programme est une suite d'instructions exécutées par la machine.

Ces instructions peuvent soit s'enchaîner les unes après les autres, on parle alors de SEQUENCE D'INSTRUCTIONS, ou bien s'exécuter dans certains cas et pas dans d'autres, on parle alors de STRUCTURE ALTERNATIVE, ou se répéter plusieurs fois, on parle alors de STRUCTURE REPETITIVE.

1.1- La séquence d'instructions :

Une instruction est une action que l'ordinateur est capable d'exécuter.

Chaque langage de programmation fournit une liste des instructions qui sont implémentées et que l'on peut donc utiliser sans les réécrire.

Dans notre pseudo-langage, nous n'aurons que la liste minimum d'instructions, nécessaire et suffisante pour les programmes que nous aurons à écrire.

1.2- L'affectation :

Variable ◄..... *Valeur*

Ce qui se lit « variable reçoit valeur » et qui signifie que l'on mémorise la valeur à un endroit nommé variable.

Par exemple : *i*◄..... *1 Termine* ◄..... *Vrai*

1.2.1- Les opérations :

Les opérations arithmétiques courantes, addition, soustraction, multiplication et division s'écrivent ainsi :

Variable ◄..... *Val 1 + Val 2*
Variable ◄..... *Val 1 - Val 2*
Variable ◄..... *Val 1 * Val 2*
Variable ◄..... *Val 1 / Val 2*

On doit toujours affecter le résultat d'une opération dans une variable.

1.2.2- Le dialogue avec l'utilisateur :

Pour permettre au programme de dialoguer avec l'utilisateur, c'est à dire d'afficher un résultat à l'écran et de lire une entrée au clavier, il faut au moins deux instructions une pour lire, l'autre pour afficher.

Dans le pseudo-langage, elles s'écrivent ainsi :

LIRE (Variable)
AFFICHER "Texte" Variable Variable "Texte" ...

La première lit tous les caractères qui sont saisis au clavier, jusqu'à ce que l'utilisateur appuie sur la touche Entrée, et stocke le résultat dans la variable.

La seconde affiche sur l'écran le ou les textes et la valeur des variables.

Exemple :

AFFICHER "Quel est ton nom ?"
LIRE (nom_utilisateur)
AFFICHER "Ton nom est : " nom_utilisateur "Le mien est TOTO"

1.3- La structure alternative :

Il est souvent nécessaire lorsque l'on écrit un programme de distinguer plusieurs cas conditionnant l'exécution de telles ou telles instruction. Pour ce faire, on utilise une structure alternative : si on est dans tel cas, alors on fait cela sinon on fait ceci.

La syntaxe de cette instruction est la suivante :

SI condition ALORS actions [SINON actions]FSI

Les crochets signifient que la partie SINON actions est facultative.

Exemple :

*SI (a < 0) ALORS ABS ◄..... a * (-1) SINON ABS ◄..... a FSI*
SI (a > 0) ALORS RES ◄..... B/a FSI

1.3.1- Les conditions :

Pour exprimer les conditions on utilise les opérateurs conditionnels suivants :

= égal
< inférieur
> supérieur
<= inférieur ou égal
>= supérieur ou égal
<> différent

Exemple : (a < 1) (toto<> titi)

On peut combiner des conditions à l'aide des opérateurs logiques suivants : ET OU

Exemple : (a < 2) ET ((b = 0) OU (c <> a))

Lorsque l'on écrit de telles conditions, il est recommandé de mettre toutes les parenthèses afin d'éviter les erreurs.

1.3.2- Les actions :

Les actions qui suivent le SINON ou le ALORS peuvent être :

- Une simple instruction
- Une suite d'instructions séparées par des ;
- Une autre alternative
- Une répétitive

1.4- La structure de choix : Cas … de :

Exemple :

Lire caractère
Cas caractère **de**
'0' … '9': écrire "chiffre";
'a' … 'z': écrire "minuscule";
'A' … 'Z': écrire "majuscule";
Fin cas

1.5- La structure répétitive :

Un programme a presque toujours pour rôle de répéter une même action un certain nombre de fois. Pour ce faire on utilise une structure permettant de dire « Exécute telles actions jusqu'à ce que telle condition soit remplie ».

Bien qu'une seule soit nécessaire, la plupart des langages de programmation proposent trois types de structure répétitive.

Voici celles de notre pseudo-langage.

1.5.1- Le TantQue :

TantQue condition
Faire actions
FTQ

Ce qui signifie : tant que la condition est vraie, on exécute les actions.

Exemple :

```
Fini ◄..... Faux ;
TantQue (NON fini)
Faire
        i◄..... (- i) + 1 ;
        SI (x/i < epsilon)
        ALORS fini◄..... VRAI
        SINON x ◄..... x/i
        FSI
        FTQ
```

1.5.2- Le faire jusqu'à :

FAIRE
actions
JUSQUA conditions

Ce qui signifie que l'on exécute les actions jusqu'à ce que la condition soit vraie.

Exemple :

```
Fini ◄..... FAUX ;
Faire
        i ◄.....( - i) + 1 ;
        SI (x/i < epsilon)
        ALORS fini◄..... VRAI
        SINON x ◄..... x/i
        FSI
        JUSQUA (fini = VRAI) ;
```

1.5.3- Le Pour :

Très souvent, on utilise une structure répétitive avec un compteur et on s'arrête lorsque le compteur a atteint sa valeur finale.

Exemple :

```
i ◄..... 1
TantQue (i < 10)
Faire
        p ◄..... p * x
        i ◄..... i + 1
FTQ
```

C'est pourquoi la plupart des langages de programmation offrent une structure permettant d'écrire cette répétitive plus simplement.

Dans le pseudo-langage c'est la structure POUR :

POUR variable ALLANT DE valeur initiale A valeur finale [PAS valeur du pas]
FAIRE actions
FinPour

Lorsque le PAS est omis, il est supposé égal à + 1.

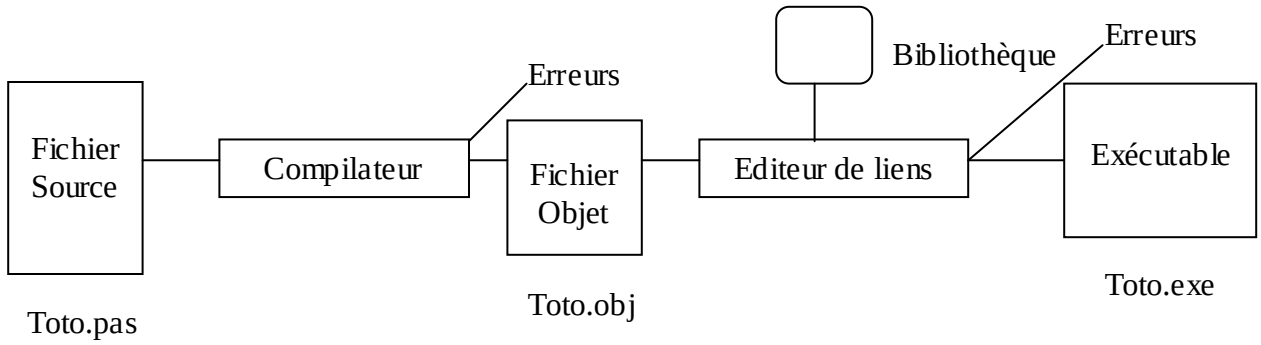
Exemple :

POUR i ALLANT DE 1 A 10
FAIRE
*p ◂..... p * x*
FinPour

1.6- La compilation :

Un langage de programmation sert à écrire des programmes de manière à les exécuter. Des outils permettent de traduite le langage écrit par le programme en langage machine.

Ils fonctionnent de la manière suivante :



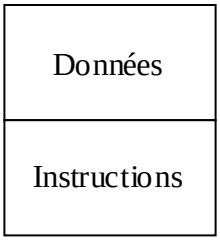
Le compilateur analyse le langage source afin de vérifier la syntaxe et de générer un fichier objet en langage intermédiaire assez proche du langage machine. Tant qu'il y a des erreurs de syntaxe, le compilateur est incapable de générer le fichier objet.

Souvent, on utilise dans un programme des fonctions qui, soit ont été écrites par quelqu'un d'autre, soit sont fournies dans une bibliothèque (graphique par exemple). Dans ce cas, le compilateur ne les connaît pas et ne peut donc pas générer le langage intermédiaire correspondant.

C'est le travail de l'éditeur de liens que d'aller résoudre les références non résolues. C'est à dire que lorsqu'il fait appel dans le fichier objet à des fonctions ou des variables externes, l'éditeur de liens recherche les objets ou bibliothèques concernés et génère l'exécutable. Il se produit une erreur lorsque l'éditeur de liens ne trouve pas ces références.

1.7- La déclaration des variables :

Un programme exécutable et composé de deux parties :



La partie instructions contient les instructions à exécuter.

La partie données contient toutes les variables utilisées par le programme.

Un programme exécutable est chargé dans la mémoire centrale de l'ordinateur, les valeurs que l'on a affectées aux variables doivent être conservées tout le temps du déroulement du programme. Par conséquent, il faut que le programme soit capable de réserver la place nécessaire aux variables.

Pour ce faire, les variables doivent être déclarées afin que le compilateur sache quelle place elles vont occuper.

1.7.1- Les types :

Les variables que l'on utilise dans les programmes ne sont pas toutes de même nature, il y a des nombres, des caractères, … On dit que les variables sont typées.

Il est nécessaire de donner un type aux variables, car cela permet d'une part de contrôler leur utilisation (ex : on ne peut pas diviser un caractère par un entier) et d'autre part car cela indique quelle place il faut réserver pour la variable.

Généralement les langages de programmation offrent les types suivants :

ENTIER : Il s'agit des variables destinées à contenir un nombre entier positif ou négatif.

Dans notre langage, la déclaration des variables de type entier est la suivante:

ENTIER variable, variable,…;

Généralement un entier occupe 2 octets, ce qui limite les valeurs de -32768 à +32768. Cependant cela dépend des machines, des compilateurs, et des langages.

Certains langages distinguent les entiers courts (1 octet), les entiers longs (4 octets) et les entiers simples (2 octets).

REEL : Il s'agit des variables numériques qui ne sont pas des entiers, c'est à dire qui comportent des décimales. Généralement un nombre réel est codé sur 4 octets

Dans notre langage, la déclaration des variables de type réel est la suivante:

REEL variable, variable,…;

CARACTERE : Les variables de type caractères contiennent des caractères alphanumériques ou numériques (de 0 à 9), mais dans ce cas ils ne sont pas considérés comme étant des nombres et on ne peut pas faire d'opérations dessus.

Un caractère occupe un octets.

Dans notre langage, la déclaration des variables de type caractère est la suivante:

CAR variable, variable,…;

N.B. : Les chaînes de caractères, dans notre langage sont des tableaux de caractères (voir 1.7.2)

BOOLEEN : Il est souvent nécessaire lorsque l'on écrit un programme d'introduire des variables qui prennent les valeurs VRAI ou FAUX ou les valeurs OUI ou NON.

Pour cela, il existe un type particulier dont les variables ne peuvent prendre que 2 valeurs : VRAI ou FAUX.

Dans notre langage, la déclaration s'écrit:

BOOLEEN variable, variable,…;

LES CONSTANTES: Une constante est inchangable durant toute l'exécution du programme.
Exemple: CONST p =3.14

1.7.2- Les tableaux :

On peut regrouper plusieurs variables sous un même nom, chacune étant alors repérée par un numéro. C'est ce que l'on appelle un tableau. On peut faire un tableau avec des variables de n'importe quel type.

Dans tous les cas le ième élément d'un tableau appelé TAB sera adressé par TAB (i). Généralement on fait des tableaux à une dimension, mais il existe également des tableaux à deux dimensions, dans ce cas TAB (i,j) représente la jème colonne et la ième ligne.

TABLEAU type variable [longueur] ;

Exemple :

TABLEAU CAR mot [10] ;
TABLEAU ENTREE liste_nb [25] ;
TABLEAU CAR MOTS [10] [20] ;

1.8- Les fonctions et procédures :

Lorsque l'algorithme devient trop compliqué, on aura envie de le découper, de manière à ce que chaque partie soit plus simple et donc plus lisible.

De même, lorsque une partie de code doit être exécutée plusieurs fois à de endroits différents, ou réutilisée ultérieurement on pourra ne l'écrire qu'une fois et lui donner un nom en faisant une **fonction** ou une **procédure**.

1.8.1- Procédure :

Une **procédure** est une suite d'instructions servant à réaliser une tâche précise en fonction d'un certain nombre de paramètres.

Les paramètres sont de deux types.

Les paramètres de type VAL : ils contiennent une valeur qui sera utilisée dans la procédure.

Les paramètres de type VAR : ils représentent une variable du programme appelant qui pourra être lue et modifiée si nécessaire.

Dans notre pseudo-langage, une procédure se déclare de la manière suivante :

PROCEDURE nom de la procédure (VAL type nom, type nom, ...
VAR type nom, type nom, ...)
Déclarations de variables locales
DEBUT
Actions
Fin

Les variables que l'on déclare localement à la procédure ne sont connues que dans cette procédure.

Pour utiliser cette procédure dans un programme appelant, on écrit :

Nom de la procédure (variable 1, variable 2, ..., variable 3, variable 4, ...) ;

A chaque paramètre de type VAR on associe un nom de variable connue dans le programme appelant, à chaque paramètre de type VAL, on associe une variable ou une valeur.

Exemples :

```
PROCEDURE remplir_chaîne (VAL CAR caractère, VAL ENTIER longueur,
                          VAR TABLEAU CAR chaîne [MAXCAR])
ENTIER i, long ;
DEBUT
    SI longueur > MAXCAR
    ALORS long ◄.....MAXCAR
    SI NON long ◄.....longueur
```

```
        FSI
        POUR i ALLANT DE 1 A long
        FAIRE
            Chaîne (i) ← caractère ;
        FinPour
FIN
```

On l'utilise ainsi :

```
Remplir_chaîne ("*",12, ligne) ;
Remplir_chaîne (longmot, rep, message) ;
```

1.8.2- Fonction :

une **fonction** est une procédure dont le but est de déterminer une valeur et de la retourner au programme appelant.

Dans notre pseudo-langage, elle se déclare de la manière suivante :

Type FONCTION nom de la fonction (VAL type nom, type nom, ...
VAR type nom, type nom, ...)
Déclarations de variables locales
DEBUT
Actions
RETOURNE valeur
FIN

Les mêmes remarques que pour la procédure s'appliquent.

De plus, il faut noter que la fonction retourne une valeur (ou le contenu d'une variable) et que donc à la déclaration on doit indiquer son type, c'est à dire le type de cette valeur.

L'appel d'une fonction s'écrit :

Variable ← nom de la fonction (valeur 1, valeur 2, ...) ;

Une fonction ne peut donc pas retourner un tableau.

Exemple :

```
ENTIER FONCTION valeur_absolue (VAL ENTIER nombre)
DEBUT
SI (nombre < 0)
ALORS
        RETOURNE (nombre * (-1))
SINON
RETOURNE nombre
FSI
FIN
```

On l'utilise ainsi :

```
i ← valeur_absolue (i) ;
limite ← valeur_absolue (-120) ;
```

2- Règles de programmation :

Un programme doit être **le plus lisible possible**, de manière à ce que n'importe qui d'autre que l'auteur soit capable de comprendre ce qu'il fait rien qu'en le lisant. Pour cela il faut suivre les quelques règles suivantes :

- Le nom des variables doit être significatif, c'est à dire indiquer clairement à quoi elles servent.
- Un algorithme ne doit pas être trop long (une page écran). S'il est trop long, il faut le découper en fonctions et procédures (voir 1.8)

- Les structures de contrôle doivent être indentées, c'est à dire, par exemple, que les instructions qui vient après le ALORS doivent toutes être alignées et décalées d'une tabulation par rapport au SI. Il en est de même pour les répétitives.
- A chaque imbrication d'une structure de contrôle, on décale d'une tabulation.

Exemple :

Mal écrit :

```
ENTIER a, b, i, j, k
TABLEAU CAR tab (10) (20)
i ◄..... 1 ; a ◄..... 0 ; b ◄..... 0 ;
FAIRE
SI (tab (i,1) <>'')
ALORS
c ◄..... 0 ; a ◄.....  a + 1
j ◄..... 1
TANTQUE ((j < 20) ET (c <> 0)) FAIRE
SI tab (i,j) = 'X' ALORS c ◄..... 0 FSI
j ◄..... j + 1 ;
FTQ FSI
SI (c = 0) ALORS b◄..... b + 1 FSI
i ◄..... i + 1 ;
JUSQUA (i > 10) ;
```

Mieux écrit :

```
ENTIER i, j ;
ENTIER nbmots, nbvecx ;
BOOLEEN trouve ;
TABLEAU CAR mots (10) (20)
i ◄..... 1 ; nbmots ◄.....0 ; nbavecx◄..... 0 ;
FAIRE
    SI (mots (i,1) <>'')
    ALORS
      Trouve ◄..... FAUX ;
      Nbmots ◄..... nbmots + 1
      j◄..... 1 ;
      TANTQUE ((j<20) ET (NON trouve))
      FAIRE
        SI (mots (i,j) = 'X')
        ALORS
          Trouve ◄.....VRAI
        FSI
      FTQ
    SI trouve ALORS nbavecx◄..... nbavecx + 1 ; FSI
    FSI
    i ◄..... i + 1 ;
JUSQUA (i > 10) ;
```

En ce qui concerne les fonctions et procédures, il y a aussi des règles à respecter :

- On doit toujours être capable de donner un nom significatif à une procédure ou à une fonction.
- Le nombre de paramètres ne doit être trop grand (en général inférieur à 5) car cela nuit à la lisibilité du programme.
- Une procédure ou une fonction doit être la plus générale possible de manière à pouvoir être réutilisée dans d'autres circonstances.
- Si le but d'une procédure est de calculer une valeur simple, il est préférable d'en faire une fonction.
- Il est souvent plus claire d'écrire une fonction booléenne plutôt qu'une condition complexe.

3- La syntaxe du pseudo-langage :

Programme : Déclarations ;
DEBUT
Actions
FIN

Déclarations : Déclaration de variables OU
Déclaration de fonction OU

Déclaration de procédure

Déclaration de variables : Type nom OU
TABLEAU type nom (entier) [(entier)] ;

Type : ENTIER OU **REEL** OU **CAR** OU **BOOLEEN**

Déclaration de fonction : Type **FONCTION** (**VAL** type nom, type nom, …
VAR type nom, type nom, …)
Déclaration de variables
DEBUT
Actions
RETOURNE valeur
FIN

Déclaration de procédure : **PROCEDURE** nom (**VAL** type nom, type nom, …
VAR type nom, type nom, …)
Déclaration de variables
DEBUT
Actions
FIN

Actions : Suite d'instructions OU
Alternative OU
Répétitive

Suite d'instructions : Instruction ; instruction ; …

Alternative : **SI** condition **ALORS** actions [**SINON** actions] **FSI**

Répétitive : Tantque OU
Faire OU
Pour

Tantque : **TantQue** condition **Faire** actions **FTQ**

Fair : **Faire** actions **Jusqua** condition

Pour : **POUR** variable **ALLANT DE** entier **A** entier [**PAS** entier]
Faire actions
FinPour

Condition : Condition ope_logique condition OU
Variable ope_cond valeur

Ope_logique : **ET, OU, NON,**

Ope_cond : =, <, >, <=, >=, <>

Instruction : Variable ◂····· valeur OU
LIRE variable OU
AFFICHER [texte valeur] N OU
Fonction OU
Procédure

Fonction : Variable ◂····· nom (valeur, valeur, …) ;

Procédure : Nom (valeur, valeur, …) ;

Valeur : Variable OU
Constante OU
(Valeur opération valeur)

Opération : =, -, *, /

4- Exercices :

4.1- L'alternative :

Un patron décide de calculer le montant de sa participation au prix du repas de ses employés de la façon suivante :

S'il est célibataire participation de 20 %
S'il est marié participation de 25 %
S'il a des enfants participation de 10 % supplémentaires par enfant

La participation est plafonnée à 50%

Si le salaire mensuel est inférieur à 6000F la participation est majorée de 10%

Ecrire l'algorithme qui lit les informations au clavier et affiche pour chaque salarié, la participation à laquelle il a droit.

4.2- La boucle :

1. Corriger l'exercice précédent pour que l'on ne soit pas obligé de relancer le programme pour chaque employé.
2. Ecrire un programme qui saisit des entiers et en affiche la somme et la moyenne (on arrête la saisie avec la valeur 0)

4.3- Les statistiques :

On saisit des entiers (comme dans l'exercice précédent) et on les range dans un tableau (maximum 50).

Ecrire un programme qui affiche le maximum, le minimum et la valeur moyenne de ces nombres.

4.4- Les tris :

Dans tous les exercices qui suivent on étudie différents algorithmes permettant de trier un tableau de 10 entiers.

Afin d'expliquer les algorithmes, on prendra en exemple le tableau suivant :

52	10	1	25	62	3	8	55	3	23

4.4.1- Le tri par sélection :

Le premier algorithme auquel on pense pour effectuer ce tri est celui-ci :

On cherche le plus petit élément du tableau et on le place en 1er, puis on cherche le plus petit dans ce qui reste et on le met en second, etc …

52	10	1	25	62	3	8	55	3	23
1	52	10	25	62	3	8	55	3	23
1	3	52	10	25	62	8	55	3	23
1	3	3	52	10	25	62	8	55	23
1	3	3	8	52	10	25	62	55	23
1	3	3	8	10	52	25	62	55	23
1	3	3	8	10	23	52	25	62	55
1	3	3	8	10	23	25	52	62	55
1	3	3	8	10	23	25	52	62	55
1	3	3	8	10	23	25	52	55	62

Ecrire l'algorithme qui permet de réaliser ce tri.

4.4.2- Le tri par bulle :

Un tri bulle est un tri plus astucieux, son principe est de faire remonter petit à petit un élément trop grand vers le haut du tableau en comparant les éléments deux à deux.

Si l'élément de gauche est supérieur à son voisin de droit on les inverse et on continue avec le suivant. Lorsque l'on est en haut du tableau on repart au début et on s'arrête lorsque tous es éléments sont bien placés.

52	10	1	25	62	3	8	55	3	23
10	52	1	25	62	3	8	55	3	23
10	1	52	25	62	3	8	55	3	23
10	1	52	52	62	3	8	55	3	23
10	1	25	52	62	3	8	55	3	23
10	1	25	52	3	62	8	55	3	23
10	1	25	52	3	8	62	55	3	23
10	1	25	52	3	8	55	62	3	23
10	1	25	52	3	8	55	3	62	23
10	1	25	52	3	8	55	3	23	62

On a parcouru tout le tableau, on recommence, jusqu'à ce que tout soit bien placé.

Ecrire l'algorithme qui réalise ce tri.

4.4.3- Le tri par permutation :

Le tri par permutation est le tri du jeu de cartes.

On parcourt le tableau jusqu'à ce que l'on trouve un élément plus petit que le précédent, donc mal placé. On prend cet élément et on le range à sa place dans le tableau puis on continue la lecture. On s'arrête à la fin du tableau.

52	10	1	25	62	3	8	55	3	23
10	52	1	25	62	3	8	55	3	23
1	10	52	25	62	3	8	55	3	23
1	10	25	52	62	3	8	55	3	23
1	3	10	25	52	62	8	55	3	23
1	3	8	10	25	52	62	55	3	23
1	3	8	10	25	52	62	55	3	23
1	3	8	10	25	52	55	62	3	23
1	3	3	8	10	25	52	55	62	23
1	3	3	8	10	23	25	52	55	62

Ecrire l'algorithme qui réalise ce tri.

4.4.4.- Le tri par comptage :

Le tri par comptage consiste pour chaque élément du tableau à compter combien d'éléments sont plus petits que lui, grâce à ce chiffre on connaît sa position dans le tableau résultat.

	52	10	1	25	62	3	8	55	3	23
	7	4	0	6	9	1	3	8	1	5
Nbre position	8	5	1	7	10	2	4	9	3	6
	1	3	3	8	10	23	25	52	55	62

4.4.5- Le tri alphabétique :

Le programme consiste à saisir des mots (au maximum 10) de 20 caractères maximum et de les insérer dans un tableau dans l'ordre alphabétique. Puis d'afficher ensuite ce tableau.

Le tableau résultat est du type TABLEAU CAR [10, 20].

4.5- Le calcul des heures :

Le programme réalise l'addition de deux données exprimées en HH :MM :SS et affiche le résultat sous la même forme.

4.6- Le jeu du pendu :

Ecrire le programme du jeu du pendu.

Le principe est le suivant :

Un premier joueur choisit un mot de moins de 10 lettres.

Le programme affiche _ _ _ _ _ _ avec un _ par lettre.

Le deuxième joueur propose des lettres jusqu'à ce qu'il ait trouvé le mot ou qu'il soit pendu (11 erreurs commises).

A chaque proposition le programme réaffiche le mot avec les lettres découvertes ainsi que les lettres déjà annoncées et le nombre d'erreurs.

4.7- La structuration :

Réécrire le jeu du pendu en utilisant des fonctions et/ou procédures.

4.8- Le crible d'Erathostène :

Cet algorithme permet d'afficher progressivement la liste des nombres premiers inférieurs à une valeur donnée : MAX.

Pour ce faire on construit un tableau de MAX éléments, vide au départ, que l'on parcourt.

Chaque fois que la case est vide cela signifie que l'indice du tableau est un nombre premier, on l'affiche puis on remplit avec une valeur quelconque toutes les cases du tableau indices par un multiple de l'indice courant.

Exemple pour MAX = 10 :

Tableau = 0 0 0 0 0 0 0 0 0 0
Indice :
1 1 est un nombre premier (je ne marque rien !)
2 2 est un nombre premier ==> je marque
Tableau = 0 1 0 1 0 1 0 1 0 1
3 3 est un nombre premier ==> je marque
Tableau = 0 1 1 1 0 1 0 1 1 1
4 4 n'est pas un nombre premier
5 5 est un nombre premier ==> je marque etc …

Ecrire un programme qui demande un nombre et affiche tous les nombres premiers inférieurs au nombre donné.

4.9- La recherche dichotomique :

Cet algorithme permet de ranger un élément à sa place ou de le trouver dans une liste de manière très rapide.

On possède une liste de N entiers triés, on cherche la place d'un nombre X.

On compare X à l'élément du milieu de tableau, si il est inférieur on réduit le tableau à sa partie gauche, si il est supérieur on réduit le tableau à sa partie droite.

On réitère l'opération jusqu'à ce que le tableau ait moins de 2 éléments.

1 3 3 5 6 8 12 25 26 42 53 55, on veut y insérer le chiffre 7
$6^{ème}$ élément 8 ==> on insère 7 dans 1 3 3 5 6 8
$3^{ème}$ élément 3 ==> on insère 7 dans 5 6 8
$2^{ème}$ élément 6 ==> on insère 7 dans 6 8
==> 7 est entre 6 et 8

Ecrire le programme qui insère le chiffre à sa place dans le tableau en utilisant fonctions et/ou procédures.

Corrigés :

5.1- L'alternative :

```
AFFICHER " L'employé est-il marié (O/N) ? "
LIRE marié
```

```
AFFIHCER " Combien a-t-il d'enfants ? "
LIRE nb_enfants
AFFICHER " Quel et son salaire ? "
LIRE salaire
SI (marié = 'N') ALORS participation ← 0,2 SINON participation ← 0,25 FSI
Participation ← participation + nb_enfants*0,1
SI (salaire <= 6000) ALORS participation ← participation + 0,1
Si (participation > 0,5) ALORS participation ← 0,5 FSI
```

5.2- La boucle :

1. Il faut rajouter une boucle :

```
Faire
    … voir exercice précédent
    AFFICHER " Y a-t-il un autre employé (O/N) ? "
    LIRE rep
Jusqua (rep = 'N')
```

2.

```
Somme ← 0
Nb_nombres ← 0
FAIRE
    AFFICHER " entrer un entier "
    LIRE nombre
    SI (nombre <> 0)
    ALORS
            Nb_nombres ← nb_nombres + 1 ;
            Somme ← somme + nombre ;
    SINON
            Moyenne ← (somme / nb_nombres)
            AFFICHER " La somme = " somme " La moyenne  = " moyenne
            FSI
JUSQUA (nombre = 0)
```

5.3- Les statistiques :

```
nb_nombres ← 0
FAIRE
        AFFICHER « entrer un entier »
        LIRE nombre
        Si (nombre <> 0)
        ALORS
                Nb_nombres ← nb_nombres + 1 ;
                TAB (nb_nombres) ← nombre
        Si
JUSQUA ((nb_nombres = 50) ou (nombre = 0))
Somme ← 0
← TAB (1)
← TAB (1)
POUR i ALLANT DE 1 A nb_nombres
LIRE
```

```
Somme ← somme + TAB(i) ;
Si (TAB(i) < min) ALORS min ← TAB (i)
Si (TAB (i) > max) ALORS max ← TAB (i) ;
FinPour
Somme ← (somme / nb_nombres)
AFFICHER " La somme = " somme " la moyenne = " moyenne
AFFICHER " Le minimum = " min " le maximum = " max
```

5.4- Les tris :

5.4.1- LE tri par sélection :

```
POUR i ALLANT DE 1 A 9
FAIRE
        Petit ← TAB (i)
POUR j ALLANT DE i A 10
FAIRE
        Si (TAB (j) < petit) ALORS petit ← TAB (j) ; position ← j FSI
FinPour
POUR j ALLANT DE position A i+1 PAS –1
FAIRE
TAB(j) ← TAB (j-1) ;
FinPour
TAB (i) ← petit ;
FinPour
```

5.4.2- Le tri bulle :

```
FAIRE
Inversion ← FAUX
POUR i ALLANT DE 1 A 9
FAIRE
        Si (TAB (i) > TAB (i+1))
        ALORS
                Tampon ← TAB (i) ;
                TAB (i) ← TAB (i+1) ;
                TAB (i+1) ← Tampon
                Inversion ← VRAI
        FSI
        FinPour
JUSQUA (inversion = FAUX) ;
```

5.4.3- Le tri par permutation :

```
POUR i ALLANT DE 1 A 9
FAIRE
        SI (TAB (i+1) < TAB (i))
        ALORS
                Abouger ← TAB (i+1)
                j ← 1 ;
                TanQue ((j < i) ET (TAB (j) < TAB (i+1)))
                Faire j ← j+1
                FTQ
```

```
                POUR k ALLANT DE i+1 A j+1 PAS –1
                Faire
                        TAB (k) ← TAB (k-1)
                FinPour
                TAB (j) ← abouger
        FSI
Fin Pour
```

5.4.4- Le tri par comptage :

```
POUR i ALLAN DE 1 A 10
FAIRE
        RES (i) ← 0
        NB (i) ← 0
        POUR j ALLANT DE 1 A 10
        FAIRE
                Si TAB (j) < TAB (i) ALORS NB (i) ← NB (i) + 1 FSI
        FinPour
FinPour
POUR i ALLANT de 1 A 10
FAIRE
        j ← NB (i)
        TantQue RES (j) <> 0
        Faire
                j ← j+1
        FTQ
        RES (j) ← TAB (i)
FinPour
```

5.4.5- Le tri alphabétique :

```
POUR nbmots ALLANT DE 1 A 10
Faire
        AFFICHER « Entrer le mot suivant »
        LIRE MOT
        Pluspetit ← VRAI
        j ← 1
        TANTQUE (pluspetit ET (j < nbmots))
        Faire
                k ← 1
                TantQue ((MOTS (j, k) = MOT (k)) ET k <= 20)
                Faire
                        k ← k+1
                FTQ
                Si (MOTS (j, k) < MOT (k))
                ALORS
                        j ← j+1
                SINON
                        Pluspetit ← FAUX
                FSI
        FTQ
```

```
        Si (j < nbmots)
        ALORS
                POUR i ALLANT DE nbmots A j+1 PAS –1
                Faire
                        POUR k ALLANT DE 1 A 20
                        Faire
                                MOTS (i, k) ← MOTS (i-1, k)
                        FinPour
                FinPour
        FSI
        POUR k ALLANT DE 1 A 20
        Faire
                MOTS (j, k) ← MOT (k)
        FinPour
FinPour
POUR i ALLANT DE 1 A nbmots
Faire
        AFFICHER MOTS (i)
FinPour
```

5.5- Le calcul de heures :

```
ENTIER i, Res J, Res H, Res M, Res S
TABLEAU ENTIER H[2], M[2], S[2]
Pour i allant de 1 a 2
Faire
        Faire
                AFFICHER « Entrer l'heure »
                LIRE H(i)
        Jusqua (H (i) < 24)
… idem pour minutes et secondes
FinPour
Res S ← S (1) + S (2)
Si (Res S > 60)
ALORS
        Res S ← Res S – 60;
        Res M ← 1
SINON
        Res M ← 0
FSI
… idem pour minutes et heures
AFFICHER H (1)"; " M (1)"; " S (1)
AFFICHER "+"
AFFICHER H (2)"; " M (2)"; " S (2)
AFFICHER "="
AFFICHER Resj " jours " ResH " heures " ResM " minutes et " ResS " secondes "
```

5.6- LE jeu du pendu :

```
TABLEAU CAR mot [10], devine [10]
```

```
TABLEAU CAR lettres [26]
ENTIER nblettres, nbechecs
BOOLEEN gagne
AFFICHER " entrer le mot a deviner "
LIRE mot
i ← 1
TANTQUE (mot (i) <> ‘ ‘) Faire i ← i+1 FTQ
Nblettres – i-1 ;
POUR i ALLANT DE 1 A nblettres
Faire
        Devine (i) ← ‘-‘
FinPour
Faire
        AFFICHER devine
        AFFICHER lettres " nombres d’échecs " nbéchecs
        AFFICHER " entrer une lettre "
        LIRE lettre
        Lettres (nbtrouve + nbéchecs + 1) ← lettre
        Absent ← VRAI
        POUR i ALLANT DE 1 A nblettres
        Faire
                Si mot (i) = lettre
                ALORS
                        Devine (i) = mot (i);
                        Nbtrouves ← nbtrouves + 1;
                        Absent ← FAUX;
                        Si (nbtrouve = nblettres)
                        ALORS gagne ← VRAI
                        FSI
                FSI
        FinPour
        Si (absent)
        ALORS
                Nbéchecs ← nbéchecs + 1
        FSI
Jusqua ((nbéchecs = 11) OU gane)
Si (gagne)
ALORS AFFICHER " Vous avez gagné "
SINON AFFICHER " Vous êtes pendu " FSI
```