

Dans ce petit document vous trouverez un condensé du cours d'ALSDS articulé sur 3 parties :

- La première partie traite de la démarche et du formalisme
- Dans la seconde partie, on trouve 2 exemples traités et commentés dans les moindres détails. Un exemple sur des données élémentaires et le second sur les tableaux.
- Et une dernière partie composée de trois exercices que vous devez traiter pour mettre en œuvre les concepts présentés et valider vos connaissances.

Nota : Les idées essentielles et les concepts sont surlignés en jaune, les commentaires sur la démarche sont en violet.

1 - LA DEMARCHE MODULAIRE ET LE FORMALISME

1ère ETAPE : COMPREHENSION DU PROBLEME POSE

Lisez attentivement l'énoncé et prenez le temps de bien comprendre ce qui vous est demandé. Garder en tête « qu'un problème bien posé est à moitié résolu ». Pour cela n'hésitez pas à mettre en relief (en surlignant, soulignant, entourant) les mots clés, les phrases ou expressions qui méritent d'être explicitées davantage, les buts à atteindre. Il ne sert à rien de se précipiter à entamer la solution avec le risque d'être totalement hors sujet. **Une bonne compréhension de votre sujet est une excellente amorce de votre analyse.** Assez souvent les idées maîtresses de la solution se trouvent dans l'énoncé lui-même et il faut apprendre à les détecter. Mais cela n'est pas toujours le cas, parfois des efforts de réflexion et de créativité sont indispensables.

2ième ETAPE - ANALYSE DU PROBLEME

a) Découpage du problème.

A ce niveau, on tente de faire ressortir les modules à construire et d'avoir une idée de l'architecture globale de la solution. Il y a des modules qui sont évidents et que l'on peut détecter rapidement, grâce à la première étape, si elle est bien menée, et des modules secondaires qui ne sont pas apparents au premier abord.

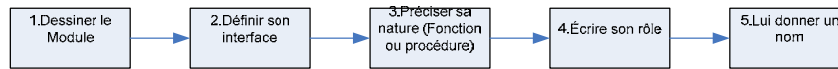
Ne perdez pas votre temps, ni votre énergie, à vouloir à tout prix recenser tous les modules nécessaires. **Commencez par les modules évidents et enrichissez votre découpage au fur et à mesure que d'autres modules vous paraîtront utiles.** C'est lors de la construction des modules évidents que d'autres modules peuvent apparaître, c'est ce que l'on appelle les modules secondaires. Ainsi le découpage initial sera enrichi progressivement. Le découpage n'est pas une suite de tâches séquentielles. On peut à tout moment modifier, rajouter ou supprimer un module, **l'essentiel est de garder la logique et la cohérence de notre solution.** Le découpage est parfois implicite, c'est-à-dire qu'une simple lecture attentive du sujet vous aidera à les recenser mais, dans d'autres cas, il nécessitera un effort d'analyse et de créativité.

Garder en tête qu'un module doit répondre tout simplement au QUOI ? Evitez, à ce niveau, de réfléchir au COMMENT ? Par exemple si nous voulons chercher le Nième élément de la suite de Fibonacci. On propose un module auquel on donne N et il nous fournit le Nième élément de la suite (le QUOI ?). On peut imaginer qu'à ce moment précis on ne sait même pas ce qu'est une suite de Fibonacci, ni comment nous allons procéder.

Aussi, **n'oubliez pas de justifier votre découpage.** Vous pouvez, à la limite, vous passer de cela si votre découpage est tellement évident qu'il ne nécessite aucun commentaire, sinon il faut l'accompagner des explications nécessaires qui permettront au lecteur de comprendre votre approche pour résoudre le problème donné.

Au niveau du découpage on peut constater que des modules existent déjà. Il s'agira simplement de les réutiliser. On ne fera que les signaler et on ne construira donc que les modules qui n'existent pas. C'est **l'un des avantages de la modularité, qui consiste à réutiliser des modules déjà faits, ce qui permet d'être plus efficace et d'aller plus vite.** Pour cela, il faut avoir à votre disposition un petit catalogue qui contient tous les modules dont vous disposez de même que celui des modules standards du langage utilisé.

Une fois que vous avez détecté les modules, on n'en a qu'une idée globale. Ensuite, il faut les reprendre un à un et les formaliser clairement et sans aucune ambiguïté. Pour chaque module il faut :



- Dessiner le module : il s'agit de dessiner un simple rectangle
- Définir son interface : **l'interface est constituée par les paramètres formels d'entrée et de sortie.** La liste des paramètres formels doit être exhaustive, il faut donc s'interdire de mettre des : Etc. ou des pointillés ... Posez-vous les deux questions élémentaires. Que doit-on fournir au module ? ce sera ses paramètres d'entrée. Que va-t-il nous retourner ? ce seront ses paramètres de sortie. Donc, sur le plan théorique si votre module a 15 paramètres en entrée et 35 paramètres en sortie (dans la pratique ce genre de module est très rare), il faudra les énumérer un à un avec leur nom et leur type bien entendu. De plus il faut définir le mode de passage de chacun de ces paramètres. La aussi, nous avons vu dans le cours, qu'**en ce qui concerne les fonctions, il est recommandé de procéder à un passage par valeur de tous les paramètres d'entrée et quand il s'agit d'une procédure, à un passage par variable de tous ses paramètres de sortie.** On observe ainsi que seuls les paramètres d'entrée d'une procédure font l'objet d'une attention particulière. **En théorie, nous pouvons aussi effectuer un passage par variable pour les paramètres d'entrée d'une fonction, mais il est fortement décommandé de le faire.**
- Préciser sa nature : (procédure ou fonction). **La définition de la nature d'un module est très simple du fait qu'elle découle tout simplement du nombre de paramètres de sortie.** Si le module n'a qu'UNE SEULE sortie et si en plus c'est un **objet élémentaire** (entier, réel, booléen, caractère) il s'agit d'une Fonction, dans tous les autres cas c'est une procédure.

Nota : **en Pascal** , une fonction peut aussi avoir comme sortie une chaîne de caractères, car cette dernière peut être considérée comme un objet élémentaire ou structuré (Cf . cours sur les chaînes de caractères)

- Ecrire son rôle : **exprimer à travers une phrase simple, précise et concise ce que fait le module** (le QUOI ?). Vérifiez systématiquement que les objets constituant l'interface du module apparaissent d'une manière ou d'une autre dans son rôle. Cela relève en fait du bon sens, car ce n'est pas à l'utilisateur de deviner le sens de chaque objet de l'interface du module. **Le rôle est assez souvent négligé alors qu'il est très important.** Lors d'une utilisation d'un module nous avons besoin de savoir ce qu'il fait et non comment il est construit. Lorsque l'on a un catalogue de plusieurs centaines de modules et que l'on y cherche un module qui peut être utilisé dans notre solution, sans avoir à le reconstruire et donc à gagner en temps et en efficacité, il est clair que l'on se basera seulement sur son rôle. Et si ce dernier est vague, ambiguë ou incomplet, le module ne sera d'aucune utilité. Il faut exclure absolument des rôles du genre : « donne N3 » !!! ou « règle le problème de l'oiselier » !!!!! ou « tresse 2 nombres » ou « donne la solution de la question 2 du problème ». Car, vous conviendrez que de tels modules sont totalement inutilisables et tous les efforts dépensés à les construire ont été inutiles.
- Lui donner un nom : **Prendre des noms explicites, en évitant si possible de dépasser 8 caractères, qui permettront par leur simple lecture d'avoir une idée de ce fait le module.** Appeler par exemple un module FACTO s'il vous donne la factorielle d'un nombre ou NB_POS s'il fournit le nombre de positions d'un nombre. Eviter de donner des noms qui ne vous renseignent pas sur le rôle de vos modules tels que Fathy, USMA, Mimosa, Zx036M ou autres fantaisies. Des noms explicites vous permettront de retrouver facilement un module, par exemple une simple recherche à travers la chaîne de caractères 'base' vous permettra d'avoir tous les modules dont le nom contient cette chaîne : Base_Dec, Dec_Base , BaseN_M, Hexabase,....
Une idée toute simple, pour donner un nom à un module, consiste à retrouver les mots clés du rôle de ce module et à construire un nom à partir de la contraction de ces mots clés. Par

exemple si le rôle est « convertir un nombre décimal en binaire », les deux mots clés étant convertir, décimal et binaire, on appellera le module ConvDBin ou Dec_Bin ou DecVersBin. Ne perdez pas de vue que le nombre de modules va augmenter très rapidement. Quand il s'agit de 20 à 30 modules il est aisé de se rappeler leur nom et même leur rôle, mais lorsque l'on dispose de plusieurs centaines de modules, le problème sera surtout de les retrouver rapidement. Pour cela, il est absolument nécessaire d'être organisé. Il est fortement recommandé de réunir les différents modules dans un même répertoire ou dans une ou plusieurs bibliothèques (Unités) et de disposer d'un catalogue de tous les modules classés par ordre alphabétique par exemple, que l'on enrichira au fur et à mesure, et qui peut avoir la structure suivante (ensemble des informations nécessaires pour chaque module) :

Fonction	:	NB_POS (N)
Unité	:	Ma_Biblio
Types des paramètres	:	N de type entier long
Type du résultat	:	Entier
Fonctionnement	:	Donne le nombre de chiffres composant N
Exemple	:	après la séquence
		X := 4589645 ;
		R := Nb_Pos(X) ;
		on aura dans R le nombre 7

Il faut aussi, garder toujours en tête qu'un module doit avoir les qualités suivantes :

- **La réutilisation ultérieure.** **Penser toujours à généraliser au maximum vos modules.** La aussi, il faut relever l'importance de la généralisation. Car il arrive, plus souvent qu'on ne le pense, d'utiliser des modules, qui au moment de leur construction, on n'avait aucune idée de leur utilisation future. Par exemple si nous avons besoin à un moment donné besoin de convertir un nombre décimal en binaire (donc base 10 vers base 2), on peut construire ce module. Mais si plus tard nous aurons besoin de convertir un nombre décimal en octal (base 10 vers base 8), le module précédent ne pourra pas être utilisé. Par contre si on moment de sa construction, on l'avait généralisé, en construisant un module de conversion de la base X vers la base Y, la solution est plus intéressante car son usage est multiple.
- **L'indépendance.** Pour cela, il faut **éviter**, sauf obligation :
 - **de mettre des lectures ou des écritures dans un module**
 - **d'utiliser des variables globales dans un module** (car c'est en plus c'est le meilleur moyen d'éviter les effets de bord).
- **La simplicité.** **Un module doit remplir UNE tâche précise.** Il suffit de lire le rôle pour s'en apercevoir. Si on vous demande de trouver le nombre d'arrangements et le nombre de combinaisons de N objets pris P à P , ne construisez pas un module qui fait les 2 en même temps mais plutôt un module qui vous donne le nombre d'arrangements et un autre qui vous donne le nombre de combinaisons. Il n'est pas normal d'imposer à l'utilisateur de vos modules, des sorties dont il n'a pas besoin à un instant t .

Ca y est ! Nous avons ramené notre problème en n petits problèmes distincts. Nous allons maintenant prendre ces petits problèmes un par un, et les solutionner.

b) Construction des modules ?

Une fois notre découpage terminé, même partiellement, on va s'atteler à **construire les modules nouveaux et séparément**, autrement dit, on va s'attaquer au **Comment ?** Il n'y a aucun ordre particulier dans la construction des modules. Ils seront construits un à un, et même par des personnes distinctes s'il le faut, pour aller plus vite. Chaque module sera considéré comme un problème à part et à aucun moment sa construction ne doit être influencée par un autre module.

D'ailleurs, ils seront testés comme des problèmes distincts et totalement indépendants les uns des autres. Mais si un module M_1 nécessite l'utilisation d'un autre module M_2 , on pourra l'utiliser dans M_1 même si M_2 n'existe pas encore, et on fera comme s'il existe.

On commencera par **l'analyse du module, qui doit OBLIGATOIREMENT se faire AVANT la construction de l'algorithme**. Il s'agit, ici de donner la ou les idées de base, à les formuler avec des phrases simples, concises et claires. Et ensuite à les structurer. Afin d'éviter l'amalgame avec l'algorithme, il y a lieu peut être de ne pas utiliser à ce niveau les mots réservés utilisés par les algorithmes. Ne pas hésiter à utiliser nos propres conventions (dessin, schéma, narrations, couleurs, ...). Utiliser le subterfuge pédagogique auquel ont recours les enseignants lorsqu'ils sont à court d'arguments : l'exemple commenté. Prenez un exemple, bien choisi, et expliquer votre idée à travers cet exemple. Faites comme si vous avez affaire à un non informaticien. Valider votre analyse en vous posant systématiquement les questions suivantes :

1. est-ce que l'idée de base de mon raisonnement est exprimée simplement ?
2. est-ce que les idées qui découlent de l'idée de base sont correctement structurées ?
3. est-ce que je ne suis pas rentré dans des détails qui risquent de rendre plus confuse mon analyse ?
4. est-ce qu'un non informaticien peut comprendre comment je procède ?
5. est-ce que je peux extraire facilement l'algorithme de mon analyse ?

On **construira ensuite l'algorithme correspondant en appliquant strictement le formalisme étudié**. En principe, votre algorithme doit découler naturellement de notre analyse, si elle est bien structurée, bien entendu.

L'algorithme est un moyen d'expression et de communication, il faut donc en respecter les règles.

Dès que l'algorithme est construit il faudra bien entendu le valider en le déroulant à la main à travers un jeu d'essai judicieusement choisi.

Comme pour les divers modules, l'algorithme principal peut être construit à n'importe quel moment. Si cela vous semble plus facile, ou plus naturel, de le construire au tout début, faites-le sans aucun problème.

3^{ème} ETAPE - REALISATION

Nous arrivons à la phase technique dans laquelle il faudra :

- Traduire les algorithmes des divers modules dans un langage donné
- **Programmer et tester SEPARÉMENT les modules**. Dans un premier temps, le module concerné est mis sous forme de module interne et lorsqu'il fonctionne correctement, après lui avoir appliqué un jeu d'essai complet on le catalogue sous forme de module externe ou en le rajoutant à notre bibliothèque (Cf. Comment cataloguer un module en Pascal ?).

ATTENTION à ce stade la construction des modules devra se faire selon un ordre bien établi. Commencer toujours par construire les modules qui ne font appel à aucun autre module et veiller à ce que si un module fait appel à d'autres modules, les modules appelés doit être construits en priorité.

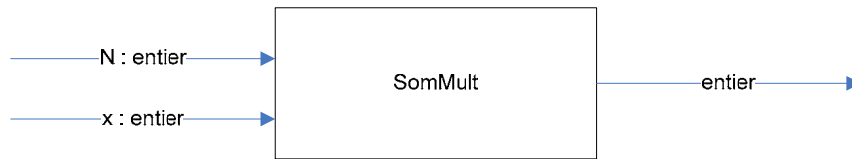
- Finalement, tester et mettre au point le programme principal.
- Veillez à ce que vos programmes soient aussi bien structurés. Pour cela, utiliser l'indentation et mettez des commentaires.

2 - EXEMPLES COMPLETS D'APPLICATION COMMENTES

Exemple 1 : utilisation des objets élémentaires

Enoncé : Etant donné un nombre N , nous voulons avoir la somme de ses chiffres qui sont des multiples de 3 et celle de ses chiffres qui sont des multiples de 4.

FONCTION



Merci d'envoyer vos commentaires et observations à : B.Chergou@ESI.Dz

CONSEIL : ne jamais construire une analyse à partir de cas particulier mais du cas général. Une fois votre algorithme construit, commencer par le dérouler avec le cas général. Et ensuite avec les cas particuliers. Très souvent, le cas général règle le cas particulier. Dans le cas contraire reprendre l'analyse et la compléter. Avec un peu d'expérience, on pourra effectuer un déroulement (plus général) de notre analyse et éviter donc de construire un algorithme pour le modifier ensuite.

Algorithme :

En principe, si l'analyse est bien structurée, l'algorithme en est extrait très facilement. Il suffit de repérer les structures de bases qui composent généralement un algorithme, c'est-à-dire : les expressions (arithmétiques, conditionnelles, relationnelles et/ou mixtes), les alternatives (ou conditionnelles), les répétitives, les lectures et les écritures.

Dans notre cas nous observons rapidement qu'il y a :

1. une affectation (initialisations de M)
2. une répétitive dont on ne connaît pas le nombre d'itérations, donc cela ne peut être un POUR. Elle sera de la forme TANT QUE car le nombre d'itérations peut être égal à 0, dans le cas où $N=0$.
3. cette répétitive contiendra :
 - a. une conditionnelle
 - b. Une expression arithmétique pour calculer le prochain N

Donc, il faut exercer votre esprit à cette démarche et au bout d'une dizaine d'algorithmes vous verrez la facilité avec laquelle vous allez construire votre algorithme. Il est clair que cela ne sera possible que si votre analyse est bien structurée.

Respectez **STRICTEMENT** le formalisme adopté et ne pas oublier que c'est un langage qui doit être commun, donc ne prenez aucune liberté le concernant.

Évitez de couper un algorithme. Arrangez-vous afin qu'il soit toujours sur une page. (Dans certains cas c'est impossible ! Non ! la modularité sera là pour remédier à ce problème à chaque fois que cela sera possible)

```

Fonction SomMult(N, x : entier) : entier
Variable M : entier

DEBUT
M ← 0
Tant que N <> 0 Faire
    DTQ
    Si N mod 10 mod x = 0 Alors M ← M + N mod 10
    N ← N div 10
    FTQ
SomMult ← M
FIN
  
```

Jeu d'essai :

Une fois l'algorithme écrit, il faut vérifier qu'il « tourne », c'est-à-dire qu'il donne le résultat attendu. Pour cela, il faut construire un jeu d'essai. Ce dernier est très souvent confondu avec le déroulement alors que leurs rôles sont totalement différents.

Le jeu d'essai est tout simplement, un ou plusieurs exemples avec leurs résultats et un petit commentaire pour justifier chacun d'eux, qui vont permettre de valider le résultat de l'algorithme. Il est évident que si vous prenez plusieurs exemples, il ne s'agit pas de mettre des cas identiques qui ne seront d'aucune utilité.

Son élaboration exige de la réflexion. Parfois un seul exemple suffit mais d'autres fois, plusieurs sont nécessaires. De façon générale il faut prévoir le cas général et ensuite les cas particuliers s'il y en a.

Ainsi, pour notre exemple le jeu d'essai peut être :

N = 8237349	M3 =15 , M4=12	{nombre quelconque}
N = 69336	M3 =27 , M4=0	{que des multiples de 3}
N = 48884	M3 =0 , M4=32	{que des multiples de 4}
N = 544712	M3 =0 , M4=8	{aucun multiple de 3}
N = 66291	M3 =21 , M4=0	{aucun multiple de 4}
N = 2157	M3 =0 , M4=0	{aucun multiple de 3 et de 4}
N = 0	M3 =0 , M4=0	{nombre nul}

Vous observez le rôle des commentaires qui expliquent l'intérêt de chacun des exemples choisis.

NOTA : dans certains cas, il ne sera pas possible de proposer un jeu d'essai. Dans ce cas, il faut valider les résultats après leur programmation. Par exemple dans l'exercice où il s'agissait de retrouver les nombres tels que la somme des cubes des chiffres les composant est égale au nombre lui-même. Il n'est pas évident de trouver un exemple, sauf s'il faut dérouler notre algorithme, au risque de perdre beaucoup de temps. Dans ce cas, une fois que notre programme donne des résultats. Il faut vérifier qu'ils sont justes.

Déroulement

Le déroulement sert à valider l'algorithme. Le déroulement est constitué d'un tableau dont chaque colonne représente un objet de l'environnement. Au départ il faut dessiner le tableau et reprendre, TOUS et SEULEMENT, les objets déclarés dans l'algorithme. Dans le cas où vous déroulez un module, il faut prendre les paramètres formels du module et ses variables locales. Dans notre cas : N, x, M.

Durant le déroulement, il faut avoir, en même temps, 2 attitudes : machinale et d'observation.

Attitude machinale : effectuer strictement les actions de l'algorithme.

1. on donne une valeur à N et à x (passage des paramètres)
2. on met 0 dans M
3. si N est différent de 0
 - ✓ si $N \bmod 10 \bmod x = 0$ on rajoute x dans M
 - .
 - .

Attitude d'observation : en même temps, on observe, au fur et à mesure, que :

- les différents chiffres de N sont extraits, un à un.
- La détection des multiples de x est bien faite
- Que la construction de M est correcte
- Et le résultat est juste (conforme au résultat du jeu d'essai)

Cette attitude d'observation permet d'arrêter rapidement le déroulement afin de reprendre l'analyse et/ou l'algorithme. Si par exemple on observe que les différents chiffres de N ne sont pas extraits correctement ou que le contenu de M contient des valeurs insolites, qu'une variable n'a pas été initialisée, qu'une colonne est vierge tout le temps, que l'on a oublié une variable, On arrête le déroulement pour apporter les corrections nécessaires à notre algorithme.

Observez **ATTENTIVEMENT** comment se fait le déroulement de notre fonction SomMult. On met d'abord les paramètres formels de la fonction (N et x), ensuite les variables locales de la fonction (ici il n'y en a qu'une seule M) et finalement le nom de la fonction pour s'assurer que la valeur renvoyée est correcte.

Exemple pris de notre jeu d'essai N = 8237349 x =3 résultat : 15

N	x	M	SomMult
8237349	3	0	
		9	
823734			
82373			
		12	
8237			
823			
		15	
82			
8			
0			15

NOTA : pour le déroulement il faut reprendre tous les exemples du jeu d'essai. C'est une assurance que la phase de conception est correcte et que lors de la réalisation les erreurs de logique seront inexistantes.

Programmation :

Cette étape consiste à traduire l'algorithme dans un langage de programmation. Elle peut être fastidieuse lors de l'apprentissage du langage. Mais il faut d'une part avoir à portée de main la documentation sur le langage utilisé et surtout s'y référer systématiquement au départ. Et d'autre part apprendre à corriger soi même les différentes erreurs de programmation. Utiliser aussi l'aide en ligne.

« Un bon programmeur n'est pas celui qui ne fait pas d'erreurs, mais celui qui sait les corriger rapidement ».

Vos programmes doivent être lisibles, pour cela :

- *Eviter de mettre plusieurs instructions par ligne*
- *Aligner verticalement les instructions d'un même bloc*
- *Utiliser l'indentation pour faire ressortir les divers blocs du programme*
- *Mettre des commentaires pertinents*

```
Function SomMult(N ,x : longint): integer;
// -----
// Donne la somme des chiffres de N qui sont des multiples de x
// -----
Var m : integer;
Begin
m := 0;
while N <> 0 do
    Begin
        if N mod 10 mod x = 0 then m := m + N mod 10;
        n := n div 10;
    End;
SomMult := m;
END;
```

```
program sommult;
var A,y:longint;
{$i e:\algo\modules\sommult.fon}
BEGIN
Write (' Donner votre nombre : ');
Readln (A);
write ('Somme des mutiples de 3: ',somMult(A,3), ', de 4:',SomMult (A,4));
Readln;
END.
```

Une fois les erreurs de programmation corrigées. Il faut exécuter le programme en reprenant les exemples du jeu d'essai (et du déroulement) afin de valider votre programme.

```
Donner votre nombre : 823349
Somme des mutiples de 3: 15, de 4:12
```

```
Donner votre nombre : 69336
Somme des mutiples de 3: 27, de 4:0
```

```
Donner votre nombre : 48884
Somme des mutiples de 3: 0, de 4:32
```

```
Donner votre nombre : 544712
Somme des mutiples de 3: 0, de 4:8
```



```
Donner votre nombre : 66291
Somme des multiples de 3: 21, de 4:0
```

```
Donner votre nombre : 2157
Somme des multiples de 3: 0, de 4:0
```

```
Donner votre nombre : 0
Somme des multiples de 3: 0, de 4:0
```

Exemple 2 : utilisation d'objets structurés (tableaux)

Énoncé : On voudrait remplir automatiquement un tableau (R) de 10 lignes et 20 colonnes avec des nombres aléatoires compris entre 0 et 5000.

Puis les éléments de R seront mis dans un tableau (X) à une dimension que l'on triera.

Une fois que X sera trié on affichera, tous les éléments qui sont premiers et ceux qui sont composés de 2 parties identiques (exemples : 77, 4949), de la manière suivante :

```
23      Premier
1414    2 parties identiques
```

1ère ETAPE : COMPREHENSION DU PROBLEME POSE

On met en évidence les éléments importants de l'énoncé

On voudrait remplir automatiquement un tableau (R) de 10 lignes et 20 colonnes avec des nombres aléatoires compris entre 0 et 5000.

Puis les éléments de R seront mis dans un tableau (X) à une dimension que l'on triera.

Une fois que X sera trié on affichera, tous les éléments qui sont premiers et ceux qui sont composés de 2 parties identiques (exemples : 77, 4949), de la manière suivante :

```
23      Premier
1414    2 parties identiques
```

On constate que le travail demandé est très clair :

1. le tableau R (10,20) est rempli automatiquement (donc les éléments ne seront pas saisis), avec des nombres aléatoires [0,1,2,...,5000],
2. R (qui est un tableau à 2 dimensions) sera ensuite transvasé dans un tableau X à une dimension. Que l'on procède ligne par ligne ou colonne par colonne n'a aucune importance puisque X sera par la suite trié,
3. on trie X
4. puis on parcourt X et on affiche les nombres premiers et ceux qui sont composés de 2 parties identiques

2ième ETAPE - ANALYSE DU PROBLEME

Découpage modulaire et sa justification :

Le travail préliminaire effectué au niveau de l'étape de compréhension du sujet va être d'une importance fondamentale au niveau de l'analyse, car il nous permettra de faire ressortir les modules dont nous avons besoin tout en justifiant notre découpage. Dans notre cas :

On doit :

- remplir notre tableau, à deux dimensions, avec des nombres aléatoires (module **T2_Aleat**). Ces nombres aléatoires seront générés grâce au module standard **RANDOM**
- puis on transvasera ce tableau à 2 dimensions vers un tableau à 1 dimension (module **CopyT2T1**)
- le tableau obtenu sera trié grâce au module (**Tri_Sel**)

- Le module (**Prem**) nous permettra de détecter si un nombre est premier
 - Et le module (module **P_Ident**) de détecter si un nombre est composés de 2 parties identiques
- Sur les 6 modules nécessaires, on constate que les 3 modules en bleu existent déjà, il suffit donc de les réutiliser. Il faut donc construire les trois autres modules.

Les modules déjà existant sont :

Procédure Tri_Sel (t : Tab2 ; tai : entier) (* trie le tableau à une dimension T selon la méthode par sélection *)

Fonction Prem (N : entier) : Booléen (* donne vrai le nombre N est premier *)

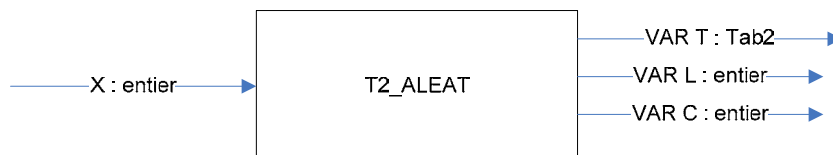
Fonction Random [(x: entier)] : réel ou mot (*module standard, génère un nombre aléatoire compris entre 0 et X*)

Il nous reste donc à construire les 3 modules suivants : T2_Aleat, CopyT2T1 et P_Ident

Dessin des modules. Nous avons vu plus haut comment procéder et nous allons l'appliquer en détail pour le premier module à savoir : T2_Aleat.

1. on dessine un rectangle
2. on définit l'interface. Comme les nombres aléatoires sont compris entre 0 et 5 000. On va généraliser en générant des nombres compris entre 0 et X. Donc X sera le paramètre d'entrée du module. Et en sortie on aura un tableau T de L lignes et C colonnes qui va contenir des nombres aléatoires compris entre 0 et X. On effectuera un passage par valeur pour X et un passage par variable
3. on précise sa nature. Puisque l'on a 3 sorties c'est donc une procédure
4. on précise le rôle. Le module fournit un tableau à 2 dimensions rempli avec des nombres aléatoires compris entre 0 et X
5. on lui donne un nom. Les mots clés étant : tableau, 2 dimensions, aléatoires. On prendra T pour désigner un tableau, 2 pour deux dimensions et aleat pour aléatoires, et on obtient T2_Aleat.

PROEDURE



Rôle : Fournit un tableau à 2 dimensions rempli avec des nombres aléatoires compris entre 0 et X

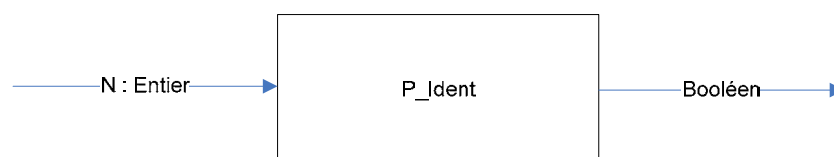
On procède de la même manière pour les autres modules. On a :

PROEDURE



Rôle : Recopie les éléments de T (tableau à 2 dimensions) dans R (1 dimension) , ligne par ligne

FONCTION



Rôle : Donne VRAI si N est composé de 2 parties identiques

Construction des modules

Notre découpage et sa justification étant faits nous allons construire les modules, un à un dans n'importe quel ordre. On aborde le Comment ?

Merci d'envoyer vos commentaires et observations à : B_Chergou@ESI.Dz

Construction du module T2_Aleat :**Analyse :**

- pour toutes les lignes (i) de T
 - et pour toutes les colonnes (j) de T on fait
 $T[i,j] = \text{random}(x)$ // on met dans une case de T un nombre aléatoire compris entre 0 et X

Algorithme :

```
procédure T2_aleat (x : entier ; var t : tab2 ; var l,c: entier)
variables i,j:entier
```

```
DEBUT
Ecrire ('Donnez le nombre de lignes : ')
Lire (L);
Ecrire ('Donnez le nombre de colonnes : ')
Lire (C)
Pour i Allant de 1 à L Faire
  Pour j allant de 1 à C Faire t[i,j] ← random(x)
FIN
```

Construction du module Copy_T2T1 :**Analyse :**

- $tai \leftarrow 0$ // c'est l'indice du tableau à une dimension R
- pour toutes les lignes i de T
 - et pour toutes les colonnes j de T on fait
 - $tai \leftarrow tai + 1$ // on incrémente l'indice
 - $R[tai] \leftarrow T[i, j]$ // on met dans R un élément de T

Algorithme :

```
procédure copyt2t1 (t : tab2 ; l, c: entier ; VAR R: tab1; VAR tai: entier )
variables i,j : entiere
```

```
DEBUT
Tai ← 0
pour i allant de 1 à l Faire
  Pour j allant de 1 à c Faire
    Dpour
    Tai ← tai+1
    R[tai] ← t[i,j]
    Fpour
FIN
```

Construction du module P_Ident :**Analyse :**

- Si le nombre de positions de N est pair // dans le cas contraire il ne peut être composé de 2 parties identiques
 - On extrait la 1^{ère} moitié de N (P1)
 - On extrait la 2^{ème} moitié de N (P2)
 - Si les deux moitiés sont égales alors N est composé de 2 parties identiques

Là, 2 modules déjà faits vont s'avérer très utiles :

Merci d'envoyer vos commentaires et observations à : B_Chergou@ESI.Dz

1. le module le module **Nb_Pos(X)** pour connaître le nombre de positions d'un nombre X
2. et le module **Extr_Nb(N, l, p)** nous permet d'extraire de N un nombre de L chiffres à partir de la position p (incluse)

Notre découpage, vu plus haut, va s'enrichir de ces 2 modules

Algorithme :

```

Fonction p_Ident (n: Entier ): Booléen
var p1,p2: entier
Fonctions extr_nb, nb_pos

DEBUT
Si Nb_pos (N) mod 2 <> 0 Alors P_ident:= FAUX
Sinon
  Dsin
  p1 ← Extr_nb(n,Nb_pos(n) div 2,1)
  p2 ← Extr_nb(n,(Nb_pos(n) div 2),Nb_pos(n) div 2 +1)
  Si p1 = p2 Alors P_ident ← VRAI
  Sinon P_ident ← FAUX
  Fsin
FIN

```

Construction de l'algorithme principal :

Analyse :

- On donne la borne Max pour générer des nombres aléatoires compris entre zéro et cette borne
- On appelle le module T2_Aleat pour créer un tableau R de 10 lignes et 20 colonnes comprenant des nombres aléatoires
- On transpose R dans le tableau à une dimension X
- On trie X
- On parcourt le tableau X, est à chaque fois qu'un élément est premier et / ou composé de 2 parties égales, on l'affiche.

On observe que la compréhension du problème a été très utile pour construire l'algorithme principal

```

Algorithme e2_1011
type tab2= Tableau [1..50,1..100] d' entiers
      tab1= tableau[1..5000] d'entiers
Variables borne, lig, col,taille : Entier
          R: tab2
          X: tab1
          i: Entier
Procédures tri_sel, T2_aleat, copyt2t1
Fonctions prem, p_Ident

DEBUT
Ecrire ('Donnez la borne max des nbres aleatoires : ')
Lire (borne)
T2_ALEAT (borne, R, lig, col)
COPYT2T1 (R, lig, col, x, taille)
Tri_sel (X, taille)
Pour i Allant de 1 à taille Faire
  Dpour
  Si prem(x[i]) = VRAI Alors Ecrire (x[i], ' Premier')
  Si P_Ident(x[i]) = VRAI Alors Ecrire (x[i], ' 2 parties identiques')
  Fpour
END.

```

3ième ETAPE - REALISATION

Au niveau de la réalisation, nos trois modules peuvent être programmés et testés dans n'importe quel ordre. Ils sont totalement indépendants les uns des autres ce qui n'est pas le cas du programme principal, car il ne peut être construit qu'une fois les trois modules réalisés.

```
(*-----*)
procedure T2_aleat(x:integer;var t :tab2;var l,c:integer);

// Fournit un tableau à 2 dimensions rempli avec des nombres aléatoires compris entre 0
et X

var i,j:integer;
BEGIN
write('Donnez le nombre de lignes : ');
readln(l);
write('Donnez le nombre de colonnes : ');
readln(c);
for i:= 1 to l do
    for j:=1 to c do
        t[i,j]:=random(x);
END;
(*-----*)
procedure copyt2t1(t :tab2;l,c:integer;var R:tab1;var tai:integer);

// Recopie les éléments de T (tableau à 2 dimensions) dans R (1 dimension) , ligne par
ligne

var i,j :integer;
BEGIN
tai:=0;
for i:= 1 to l do
    for j:=1 to c do
        BEGIN
            tai:=tai+1;
            R[tai]:=t[i,j];
        END;
END;
(*-----*)

Function p_Ident(n:longint): boolean;

// Donne Vrai si N est compos, de 2 parties identiques

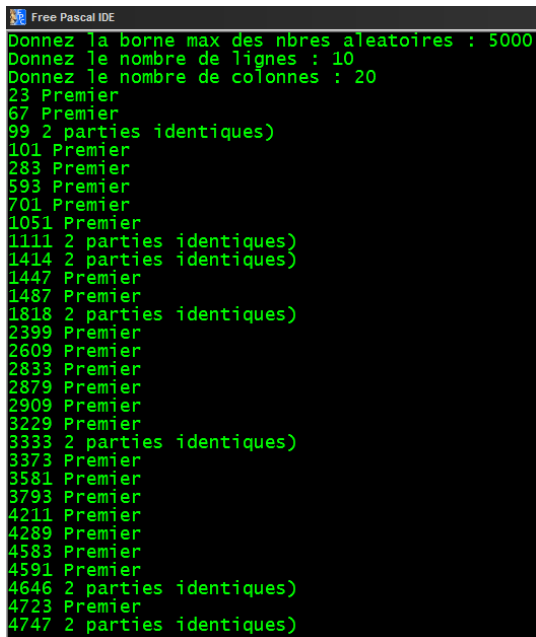
var p1,p2:longint;
{$i E:\algo\modules\extr_nb.fon}
{$i E:\algo\modules\nb_pos.fon}
BEGIN
if Nb_pos(N) mod 2 <> 0 then P_ident:=false
else
    BEGIN
        p1:= Extr_nb(n,Nb_pos(n) div 2,1);
        p2:=Extr_nb(n, (Nb_pos(n) div 2),Nb_pos(n) div 2 +1);
        if p1 = p2 then P_ident:=True
        else P_ident:=false ;
    END;
END;
(*-----*)
```

Mettez au moins le
rôle du module

Utilisez l'indentation

```
program e2_1011;
uses crt;
type
  tab2=array[1..50,1..100] of longint;
  tab1=array[1..5000] of longint;
var
  borne, lig, col,taille :integer;
  R:tab2;
  X:tab1;
  i:longint;
{$i E:\algo\modules\T2_Aleat.pro}
{$i E:\algo\modules\CoptT2T1.pro}
{$i E:\algo\modules\prem.fon}
{$i E:\algo\modules\P_Ident.Fon}
BEGIN
clrscr;
write('Donnez la borne max des nbres aleatoires : ');
readln(borne);
T2_ALEAT(borne,R,lig,col); // on remplit le tableau R avec des nombres aléatoires
COPYT2T1(R,lig,col,x,taille); // on transpose R (2 dimensions) dans X (1 dimension)
Tri_sel(X,taille); // on trie X
for i := 1 to taille do //on affiche les nbres premiers et ceux composés de 2 parties
  identiques
    Begin
    if prem(x[i])=true then writeln(x[i], ' Premier',
    if P_Ident(x[i])=true then writeln(x[i], ' 2 parties identiques)');
    END;
readln;

END.
```



```
Free Pascal IDE
Donnez la borne max des nbres aleatoires : 5000
Donnez le nombre de lignes : 10
Donnez le nombre de colonnes : 20
23 Premier
67 Premier
99 2 parties identiques)
101 Premier
283 Premier
593 Premier
701 Premier
1051 Premier
1111 2 parties identiques)
1414 2 parties identiques)
1447 Premier
1487 Premier
1818 2 parties identiques)
2399 Premier
2609 Premier
2833 Premier
2879 Premier
2909 Premier
3229 Premier
3333 2 parties identiques)
3373 Premier
3581 Premier
3793 Premier
4211 Premier
4289 Premier
4583 Premier
4591 Premier
4646 2 parties identiques)
4723 Premier
4747 2 parties identiques)
```

Mettez des
commentaires

Et surtout, soignez votre travail, c'est une qualité essentielle pour un ingénieur .

Utilisez des couleurs différentes(en évitant si possible le rouge), des schémas, des exemples commentés aérés votre travail (et, bien qu'il n'est pas logique de le rappeler, utilisez une règle chaque fois que cela est nécessaire).

Un travail bien présenté entraine une image positive que fait de vous le lecteur (ou le correcteur).

3 - EXERCICES A TRAITER

Vous trouverez ci dessous trois exercices, essayez de les traiter en respectant la démarche présentée ci-dessus.

Exercice 1 :

On voudrait retrouver, dans un tableau à 2 dimensions (contenant des nombres entiers), le nombre, les coordonnées (ligne, colonne) de même que le(s) sens d'apparition d'une suite de nombres donnée.

On entend par sens le fait que la suite soit vers la droite ('D'), vers la gauche ('G'), vers le bas ('B') ou vers le haut (H) d'une case donnée du tableau.

Exemple : si dans le tableau suivant nous cherchons la suite 7, 11, 9

T	1	2	3	4	5	6	7
1	13	12	8	9	58	7	3
2	5	7	12	11	13	11	7
3	7	9	11	7	11	9	10
4	8	12	10	11	11	13	7
5	11	8	5	9	7	11	9
6	13	10	8	13	10	7	12

Nous constatons qu'elle existe : à la position T[1,6] vers le bas

à la position T[3,4] vers la droite, la gauche, le haut et le bas

à la position T[5,5] vers la droite

La suite existe donc 6 fois : 1 6 'B', 3 4 'D', 3 4 'G', 3 4 'H', 3 4 'B', 5 5 'D'

Proposez une solution complète (conception et réalisation) à ce problème.

Exercice 2 :

Soient 2 matrices :

- A contenant des éléments non nuls,
- et B, de même dimension que A, mais dont les éléments sont 0 ou 1.

La compression de A par B donne la matrice C dont les éléments sont ceux de A pour lesquels les valeurs correspondantes dans B (c'est à dire ceux qui se trouvent à la même ligne et à la même colonne) sont égales à 1.

A					B					C				
1	1	8	5	1	1	0	1	1	0	1	8	5	4	2
2	4	3	1	2	2	1	0	0	1	2	5	7	5	2
3	5	6	7	9	3	1	0	1	0	3	3	0	0	0
4	3	5	2	3	4	0	1	1	1	4	0	0	0	0
	1	2	3	4		1	2	3	4		1	2	3	4

Exemple

Comment compresser une matrice par une autre ? (conception et réalisation)

Exercice 3 :

Les formules de transformation utilisées dans les méthodes d'organisation des fichiers, sont des fonctions qui permettent de déterminer l'adresse physique à laquelle sera stocké sur le disque un enregistrement. Elles sont de la forme : F (indicatif) = adresse.

Dans notre cas, les indicatifs (ou clés) qui représentent les matricules des étudiants de l'ESI sont sur 6 positions (2 positions pour l'année d'entrée et 4 positions qui représentent le N° séquentiel d'inscription) et notre fichier contient au plus 1500 enregistrements c'est-à-dire 1500 étudiants.

La formule utilisée dans notre cas est appelée « formule de transformations par traduction ». Elle consiste à multiplier chaque position de l'indicatif respectivement par des nombres premiers les plus proches de 100, 200, 300,... et on additionne le tout. Puis on extrait à gauche autant de positions que l'adresse physique doit en avoir.

Notre adresse physique est de la forme : CCPPPSS (CC : N° de cylindre, PPP :N° de piste, SS :N° de secteur).

On voudrait :

1. générer le jeu d'essai des matricules, pour l'année 2014 et pour 500 étudiants
2. générer les adresses de notre jeu d'essai
3. trouvez le nombre de synonymes générés par la formule. Les synonymes sont des adresses identiques alors que les indicatifs (matricules) sont différents.

Travail a faire : proposer un découpage.