SDA1 Travaux Pratiques n°3 Année scolaire 2012-2013

Remarque : Ce TP se déroulera en 4 parties. Les 3 premières parties d'une durée de 3h ne sont pas notées. Lors de la dernière heure, il vous sera donné un sujet qui lui sera noté.

Objectifs

- Manipulation des tableaux
- Manipulation des fonctions

Compétences attendues

- Comprendre les étapes de développement d'une application informatique
- Etre capable de coder en langage C une analyse en pseudo-code

Etapes de développement d'une application en programmation procédurale

L'approche classique de développement repose sur un cycle en V. Elle passe par 3 phases : analyse, codage et tests.

Phase 1 : L'analyse

L'analyse consiste dans un premier temps à comprendre le besoin afin d'envisager une solution pouvant y répondre. La compréhension nécessite souvent de faire des schémas qui permettent de mieux comprendre le problème posé. La solution se traduit ensuite dans le choix de structures de données et la définition d'un algorithme par exemple sous forme de pseudo-code.

Dès que l'analyse a été faite, il faut immédiatement définir les tests à effectués pour vérifier le code qui sera obtenu. C'est la compréhension de la solution qui permet d'envisager les différents cas à tester : le cas général, mais également des cas limites. Dans tous les cas les tests devront répondre à trois objectifs : valider le besoin (ou le cahier de charges), vérifier la robustesse de programme (pas de plantage intempestif), vérifier l'ergonomie du programme.

Phase 2 : Codage

On peut ensuite passer à la phase de codage. C'est dans cette phase que l'on choisit la plateforme de développement et le langage de prorgrammation.

- La plateforme de développement est basé sur la distribution **linux Debian**. Ce choix est justifié par des raisons pédagogiques : vous habituez à utiliser un système libre qui favorise l'insertion de l'outil informatique dans les PME. Il est également à la base des applications embarquées que l'on retrouve dans de nombreux systèmes multimedia grand-public mais également de plus en plus dans des applications industrielles.
- Le langage de programmation et le **langage C** pour son utilisation dans tous les secteurs de l'entreprise, notamment pour le développement des applications industrielles.

La phase de codage est d'autant plus rapide que l'analyse a bien été réalisée. Le code doit être écrit de manière à faciliter la maintenance de l'application. La relecture par une personne tiers doit être facilitée. Cet objectif est atteint par :

- un nommage judicieux des variables correspondant à leur rôle,
- l'indentation du programme,
- l'insertion de commentaires dans le code.

La phase de codage comprend 3 étapes :

Etape 1 : Vous éditerez votre fichier source à l'aide d'un éditeur pleine page de type « kate » ou « gedit ».
#kate <source>.c &

Etape 2 : Compilation et edition de liens du fichier source en code exécutable avec gcc
#gcc <source>.c –o <executable.exe>

La compilation sert à transformer le code source en code objet → permet de vérifier la syntaxe
L'édition de lien sert à inclure le code des fonctions des bibliothèsques et à transformer le code objet en code exécutable
Avec gcc, l'option « -o » permet de faire la compilation suivi de l'édition de lien

Etape 3 : Lancer l'exécution de votre programme
./<executable.exe>

Phase 3 : Les tests

Une fois le code réalisé, il faut effectuer les tests définis dans la phase d'analyse. Ces tests doivent donner des jeux d'essais qui doivent être commentés pour indiquer en quoi il vérifie la conformité du code ou besoins ou ils prouvent sa robustesse.

Comment faire des jeux d'essais
Tous les cas doivent être testés. Les tests doivent être commentés si possibles en renvoyant à des parties de code.

Capture du test :
#script <fichier_resultats>
Script demarré
=> que tout ce qui sort à l'écran est redigéré vers ce fichier. A la fin arrêter le script

#exit

Partie I – Ecriture de sous-programmes en langage C

Remarques :

- Vous devez envisager à chaque fois tous les cas de tests prouvant la robustesse et le respect du cahier de charge.
- Ne pas utiliser les fonctions de maths.h dans ce TP.

Exercice 1 :

Ecrire des sous-programmes permettant la saisie d'une valeur entière. On envisagera deux versions dont les prototypes sont donnés ci-dessous :

- Cas 1 : int saisirUneValeur1()
- Cas 2 : void saisirUneValeur2(int *adV)

Ecrire un sous-programme permettant d'afficher une valeur entière. Son prototype est le suivant : void afficherUneValeur1(int V)

1. **Tester ces sous-programmes.**

2. Quels sont les avantages respectifs de ces deux méthodes de saisie d'une valeur ?

Exercice 2 :

Ecrire deux sous-programmes permettant de calculer la puissance d'un nombre entier a élevé à la puissance b. On considère de nouveau ceux cas :

- Cas 1 : int puissance1(int a,int b);
- Cas 2 : void puissance2(int a,int b,int *c) ; avec c l'adresse de la variable ou l'on stocke le resultat de a puissance b.

Utiliser les sous-programmes de l'exercice 1 pour saisir et afficher les nombres entiers permettant de tester cette fonction puissance.

Exercice 3 :

Ecrire deux sous-programmes de calcul de la factorielle d'un nombre entier. Utiliser les prototypes suivants :
Cas1 : int fact(int n) ; // La valeur retournée est la valeur de factoriel
Cas2 : void fact (int n, int *c) ; avec c l'adresse de la variable où l'on stocke le résultat

Utiliser les sous-programmes de l'exercice 1 pour saisir et afficher les nombres entiers permettant de tester cette fonction puissance.

Partie II – Transcription d'une analyse en pseudo-code en code C

Un tableau de dimension n est dit magique si les sommes des valeurs placées sur ses lignes, ses
colonnes et ses deux diagonales principales sont égales. Si de plus, ce tableau ne contient que les
nombres entiers de 1 à n^2, tous les termes étant différents, on parle alors de carré magique.
La somme des valeurs contenues sur chaque ligne, sur chaque colonne ainsi que sur les deux diagonales principales est alors égale à
$n*(n^2+1$
$)/2$.

ligne \ colonne				
				15
	6	1	8	15
	7	5	3	15
	2	9	4	15
	15	15	15	15

Carré magique

Exercice 4 : Evalué si un carré est magique, on utilisera l'algorithme suivant

```
programme magique ;
debut
MAXELT =10 ;
entier Tableau(MAXELT+1, MAXELT+1) ;
entier taille ;
entier ligne, colonne ;
faire
afficher (« saisir la taille ») ;
lire(taille),
TQ (taille<0) ou (taille >MAXELT-1) ;

saisirtcarre(Tableau, taille) ;

affichercarre(Tableau, taille) ;

//Calcul de la somme des lignes
pour ligne allant de 1 à  taille faire,
        Tableau(ligne,taille+1)<-sumligne(Tableau, taille, ligne);
finpour ;

//Calcul de la somme des colonnes
pour colonne allant de 1 à  taille faire,
        Tableau(taille+1,colonne)<-sumcolonne(Tableau, taille, colonne);
finpour;
```

```
// Calcul de la somme des diagonales
  Tableau(taille+1,taille+1)<-sumdiag1(Tableau, taille);
  Tableau(0,taille+1)<-sumdiag2(Tableau, taille);

// On verifie si le carre est magique
colonne ← 1 ;
pasmagique<-faux;
TQ (pasmagique=faux) et (colonne <=taille) faire
        pasmagique<-Tableau(taille+1, colonne)<>Tableau(taille+1, colonne+1) ;
        colonne<-colonne+1 ;
finTQ ;

ligne<-0
TQ (pasmagique=faux) et (ligne <=taille)faire
        pasmagique<-ligneTableau(ligne, taille+1)<>Tableau(ligne+1, taille+1) ;
        ligne=<-ligne+1 ;
finTQ ;

si (pasmagique=faux) et Tableau(taille+1,taille+1=n(n²+1)/2)
                        alors afficher(«le carre est magique) ;
                        sinon afficher(«le carre n'est pas magique) ;
finsi ;

fin.
```

1. Ecrire les sous-programmes suivants :

- saisircarre
- affichercarre
- sumligne
- sumcolonne
- sumdiag1 qui calcule la valeur de la diagonale descendante allant de de Tableau(1,1) à Tableau(taille,Taille)
- sumdiag2 qui calcule la valeur de la diagonale montante allant de de Tableau(taille,1) à Tableau(1,Taille)

On déclarera les prototypes des fonctions et leurs définitions

2. Proposer des améliorations de l'algorithme permettant de le rendre plus performant et tester les.

Partie 3 : tester vos algorithmes

Etablir des jeux de tests comme :

Test 1 :

```
int Tableau[MAXELT+1] [MAXELT+1]=
 {
 { 1, 2, 3},

                                                { 2, 3, 1},

                                                { 3, 1, 2},
};


 int taille= 3; /* dimension effective du carré magique  */
```