S3

Programmation impérative en langage C

feuille de TP n°1 : Environnement de travail, spécifications et tableaux dynamiques

---

Objectifs du TP :

- *Éditer compiler et exécuter un programme C en ligne de commande*
- *Spécifier un programme en triplet de Hoare*
- *Mettre en œuvre des tableaux dynamiques*

1 L'environnement de travail

1. **Connection**
   - Procédure de connexion :
     Le système d'exploitation que nous utilisons pour les TP est Linux Fedora. Pour se connecter («logger») sous linux, deux modes sont possibles : le mode console (texte), et le mode graphique (avec des fenêtres).
     Par défaut, il y a 6 consoles et un mode graphique. Pour aller du mode graphique à la console, taper CTRL+ALT+Fi où i varie de 1 à 6. Pour repasser en mode graphique, taper CTRL+ALT+F7 ou ALT+F7. Pour passer d'une console à l'autre, taper CTRL+ALT+Fi où i est la console où vous désirez aller.
     Nous nous connectons en mode graphique.
   - Procédure de déconnexion :
     Pour se déconnecter, il faut choisir l'item déconnexion du menu "Mandriva" en bas à gauche de l'écran, puis choisir le bouton éteindre l'ordinateur. A partir de là **on ne touche plus à rien**, la machine s'éteint seule sans action sur le bouton marche/arrêt.
2. **Édition d'un programme C**
   On utilise un éditeur de texte (kate, gedit...) conviennent. En particulier kate possède les utilitaires standards pratiques (numérotation des lignes de code, coloration syntaxique, indentation automatique, terminal intégré...)
3. **Compilation d'un programme C**
   Nous utiliserons le compilateur gcc de Gnu. Le comportement de ce compilateur peut être modifié à l'aide d'options. Nous utiliserons systématiquement l'option *-Wall* (warnings all) ou l'option *pedantic* pour rendre la compilation stricte et faciliter la recherche d'erreurs.
   La compilation se fait en deux phases :
   — La compilation proprement dite qui consiste à traduire le code C en code objet (exécutable) en gardant les références symboliques.
   — L'édition de liens qui consiste à remplacer dans le code objet les références symboliques (sous programmes, itérations, etc.) par les adresses réelles dans ce code.
   Des erreurs peuvent survenir dans l'une ou l'autre de ces phases.

— Les erreurs de compilation sont anoncées par un message et un numéro de ligne précédés des mots : *error* (l'erreur est fatale) ou *warning* (l'erreur n'empêche pas la génération de code objet).
— Les erreurs d'édition de liens sont annoncées par un message précédé des lettres *ld*.

La compilation du fichier nom_du_fichier.c se fait dans une fenêtre terminal à l'aide de la commande :

```
gcc -Wall nom_du_fichier.c -o  nom_du_fichier_exécutable_sans_extension  [-lm]
```

par exemple : `gcc -Wall hello.c -o hello`

L'option -lm est nécessaire pour l'utilisation de fonctions mathématiques : cos,pow,fabs...

4. **Exécution**
On utilise :

```
./nom_du_fichier_exécutable
```

dans notre exemple : `./hello`

◇ Exercice **1** : Éditez, compilez , (corrigez peut-être...) et exécuter un programme d'affichage du message "Hello world" .

2 Spécifications

On spécifie un programme par :
— un prédicat d'entrée PE réalisé par les données du programme (hormis le type des données)
— un prédicat de sortie PS réalisé par les résultats du programme
— des prédicats intermédiaires si nécessaire
Dans le programme, ces prédicats prendront la forme de commentaires en français ou bien de formules de la logique des prédicats.
On utilisera les symboles habituels vus en cours :
$\forall$ (quantificateur universel) $\forall I :\ P_1 \rightarrow P_2$
$\exists$ (quantificateur existentiel) $\exists I :\ P_1 \wedge P_2$
ν (quantificateur numérique) $\nu I : (I \in E) \wedge P(I)$ nombre des I dans E vérifiant la propriété P
$\neg$ (négation)
$\rightarrow$ (implication)
$\wedge$ (conjonction)
$\vee$ (disjonction)
$\in$ (appartenance)
avec le parenthèsage nécessaire.

Pour alléger certaines écritures on pourra utiliser :
· $(T(I..J), <)$ ou $(T(I..J), <=)$ qui exprime la croissance de T entre les indices I et J
· $t = A$ pour exprimer que les 2 vecteurs t et A contiennent des valeurs identiques

· $t = permut(A)$ pour exprimer que le vecteur t contient toutes les valeurs de A mais peut-être dans le désordre

◇ Exercice **2** : Étant donné un vecteur de N(≤ 10) entiers entrés au clavier, écrire un programme qui vérifie si ce vecteur est un palindrome. Un vecteur est palindrome lorsque ses cases symétriques sont identiques.
Compléter le cartouche et écrire le programme.

Exo2 : Détermination du caractère palindrome d'un tableau
Données : (int N, int T[10]), $0 \leq N \leq 10$,
Résultat : (bool est_palin) , est_palin = ... à compléter

```
Donnez la dimension du tableau: 9
Donnez le tableau: 5 3 8 26 7 26 8 3 5
C'est un palindrome.
```

◇ Exercice **3** : Écrire un programme qui satisfait à la spécification suivante :

Exo3 : Que fait ce programme ? à compléter en langage naturel
Données : int N, $0 \leq N \leq 10$, int T[10]
Résultat : int t[10], $\forall i : 0 \leq i < N \rightarrow t[i] = T[N-1-i]$

Dans cette spécification t représente le tableau T modifié.
La solution ne doit pas faire intervenir de tableau auxiliaire.

3 Tableaux dynamiques

L'allocation dynamique permet :
· de dimensionner un tableau avec la valeur d'une variable entière
· d'accéder à une zone de mémoire appelée : tas

1. Cas d'un vecteur :
   Exemple :

```c
//création d'un vecteur dynamique de 100 réels dans le tas:
float* vec = malloc(100*sizeof(float));
//on vérifie ensuite que l'allocation s'est bien déroulée
//sinon on arrête le programme
if (vec ==NULL)
 { printf("Pb d'allocation."); exit(1);  }
//on peut modifier la taille du tableau:
vec=realloc(vec,200*sizeof(float));
//on vérifie toujours que l'allocation s'est bien déroulée:
if (vec ==NULL)
 { printf("Pb d'allocation."); exit(1); }
//en fin de traitement, on libère la mémoire:
free(vec);
```

◇ Exercice **4** : On dispose d'un fichier texte f.txt contenant un nombre inconnu de valeurs entières séparées par des caractères de contrôle. Écrire un programme qui place les entiers du fichier f.txt dans un tableau dynamique T .

2. Cas d'une matrice à N lignes et P colonnes :

   Deux étapes sont ici nécessaires à l'allocation dynamique.

   Exemple :

```
int N=2;
int P=3;
//allocation dans le tas d'un tableau dynamique de pointeurs sur int
int** m=malloc(N*sizeof(int*));
//on vérifie que l'allocation est ok
if (m==NULL)
{       printf("Pb allocation"); exit(1); }
int i,j;
for (i=0;i<N;++i)
{
    //allocation dans le tas d'un tableau dynamique m[i]
        m[i]=malloc(P*sizeof(int));
        //on vérifie que chaque allocation est ok
        if (m[i]==NULL)
        { printf("Pb allocation"); exit(1); }
}
```

◇ Exercice **5** : Une matrice avec n lignes et p colonnes est représentée par un tableau de dimension 2.

Par définition, le produit m1*m2 de deux matrices m1 et m2 s'effectue ligne par colonne. Ce produit n'est défini que si le nombre de colonnes de m1 est égal au nombre de lignes de m2.

par exemple :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 2 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 2 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1\times 1+2\times 0+3\times(-1)+4\times 1=2 & 4 & 9 & 13 \\ 1 & 1 & 0 & 2 \\ -2 & -2 & 1 & -1 \\ 2 & 2 & 3 & 5 \end{pmatrix}$$

ou encore :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 2 & 1 \end{pmatrix} \begin{pmatrix} -1 & 2 \\ 1 & 0 \\ 1 & 2 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 12 \\ 1 & 1 \\ 3 & 3 \end{pmatrix}$$

Écrire un programme qui fait le produit matriciel de deux matrices saisies par l'utilisateur. Ces deux matrices sont de dimensions quelconques ; si le produit n'est pas possible on affiche un message sinon on affiche la matrice produit.

◇ ◇ ◇◇ ◇ ◇◇