



EXAMEN DE LABORATOIRE ASSEMBLEUR JUIN 2006

NOM :
Prénom :
Groupe :
Professeur : BEJ - HAL - JCJ – NVS - PBT

L'examen se déroule comme suit :

- Vous disposez de quatre (4) heures pour cet examen.
- Vous travaillez sur les machines de l'école (pas de portable).
- Vous avez accès au contenu de vos répertoires de travail sous MS-DOS et LINUX.
- Vous pouvez utiliser vos notes de cours.
- Veillerez à taper du code lisible. Pour vous y aider, nous imposons la présence de commentaires (rôle des variables, brève description de chaque fonction).
- Déposez régulièrement vos sources dans le eCasier de votre professeur. Rassemblez vos sources dans un dossier portant votre nom et suffixé par V1, V2, etc.

Bon travail !

PREMIÈRE PARTIE : PARTIE ASSEMBLEUR SOUS MS-DOS

1-Présentation

Votre programme consiste en l'affichage d'un rectangle de couleur à l'écran.

2- Description détaillée

Le rectangle que vous devez afficher est en réalité une succession de caractères disposés de telle sorte que l'ensemble forme un rectangle. Ce dernier est d'une longueur de quatorze (14) caractères et d'une hauteur de six (6).

On vous impose le caractère à afficher et son attribut (c'est-à-dire sa couleur). Ces informations sont stockées dans un fichier binaire qui vous est fourni pendant cet examen.

3- Le fichier `carac.hal`

On met à votre disposition (dans le eDistri HAL) un fichier binaire nommé « `carac.hal` » de taille 168 octets contenant une succession de codes binaires représentant respectivement l'attribut du caractère puis le caractère à afficher.

4- L'affichage

Vous travaillez dans un environnement MS-DOS en mode d'affichage texte (80 colonnes, 25 lignes). Le premier caractère formant le rectangle a comme coordonnées : colonne 30, ligne 10.

Remarque :

-Si un problème survient lors du traitement du fichier, votre programme doit afficher le message d'erreur suivant :

```
Erreur dans le traitement du fichier, le programme va
se terminer
```

-Si l'affichage se fait correctement, un message de fin (de votre choix) est affiché avant de terminer le programme.

5- Exigence de programmation

On vous impose d'utiliser un minimum de variables globales. En fait, à part les variables de type chaîne que vous devez utiliser, deux autres variables sont admises : une variable *handle* et une variable *buffer* pour gérer les accès au fichier. Toutes les autres données sont soit des variables locales, soit des constantes, soit définies par registre.

6- Liste des interruptions

Changement de mode vidéo : _____ int 10h,00h

AH – 00h

AL – numéro du mode vidéo

*Pour ce travail utilisez le mode 03h : 80 *25 caractères, 16 couleurs*

Ecrire un caractère à l'écran : _____ int 10h,09h

AH – 09h

AL – code ASCII du caractère

BL – l'attribut du caractère

BH – numéro de page (0)

CX – nombre de répétition

Afficher une string _____ int 21h,09h

AH – 09h

DX – *offset buffer*

Changer la position du curseur : _____ int10h,02h

AH – 02h

BH – 00h (mode graphique)

DH – ligne

DL – colonne

Marquer une pause _____ int 16h,00h

AH – 00h

DEUXIÈME PARTIE : PARTIE ASSEMBLEUR SOUS LINUX

1. Directives générales

Toutes les fonctions qui vous sont demandées reçoivent tous leurs arguments par la pile. Aucune d'elles ne nettoie la pile, c'est toujours la procédure appelante qui s'en charge.

Aucune variable globale ne peut être utilisée, sauf celles stockant les chaînes de caractères constantes ou résultat (voir plus bas).

Toutes les valeurs numériques considérées, qu'il s'agisse de données ou de résultats doivent être considérées comme des entiers non signés codés sur quatre (4) octets.

Pour vos tests, n'hésitez pas à recourir à la fonction `printw` dont le source est fourni.

2. Description détaillée

1.Écrivez une fonction `estEntierNonSigne` qui reçoit en argument l'adresse du premier caractère d'une chaîne de caractères terminée par un zéro (0) binaire. La fonction ne modifie pas cette chaîne de caractères. Elle retourne la valeur binaire un (1) dans `eax` si le chaîne de caractères représente un nombre entier non signé en base dix (10), zéro (0) autrement.

2.Écrivez une fonction `afficher` qui affiche à l'écran la chaîne de caractères terminée par un zéro (0) binaire qu'elle reçoit en premier argument. En second argument, elle reçoit une valeur numérique stocké sur un double mot. Si cette valeur est non nulle (différente de zéro), l'affichage de la chaîne est suivi de celui d'un passage à la ligne. Si le second argument est nul, aucun affichage n'est réalisé après celui du dernier caractère imprimable de la chaîne.

3.Écrivez une fonction `strVersBin` qui reçoit en argument une chaîne de caractères et retourne dans `eax` la valeur numérique représentée par cette chaîne. Pour convertir, on suppose que la chaîne représente un nombre non signé en base dix (10) et qu'aucune erreur ne survient.

4.Écrivez une fonction `binVersStr` qui reçoit en argument l'adresse du premier caractère d'une chaîne de caractères et un entier non signé sur quatre (4) octets. La fonction convertit le motif binaire en représentation textuelle en base dix (10). On suppose que la chaîne de caractères fournie en argument est suffisamment longue. Un zéro (0) binaire final est ajouté pour marquer la fin effective de la chaîne.

5.Utilisez les quatre fonctions dont le développement vous est demandé ci-dessus au sein d'une fonction principale `main` telle que le programme correspondant, `multiplier`, ait le comportement suivant. `multiplier` reçoit deux argument *par la ligne de commande*. Ces deux arguments doivent représenter des entiers non signés en base dix (10). Le produit de ces deux nombres est calculé et affiché, si les arguments sont valides en nombre et en nature. Dans le cas contraire, un message d'erreur adéquat est affiché.

De manière exhaustive :

- si le nombre d'arguments fournis par la ligne de commande n'est pas correct, la chaîne de caractères :

Usage : `./multiplier nb1 nb2`
suivie d'un passage à la ligne est affichée puis le programme retourne. Cette chaîne de caractères, sans passage à la ligne final, peut être déclarée comme variable globale. Ainsi par exemple, si l'utilisateur encode :

```
./multiplier 123 456 33
```

la chaîne :

```
Usage : ./multiplier nb1 nb2
```

est affichée ;

■ si le nombre d'arguments fournis par la ligne de commande est correct, mais si le premier argument facultatif ne représente pas un entier non signé en base dix (10), la chaîne de caractères :

Le 1er argument n'est pas un nombre valide
suivie d'un passage à la ligne est affichée puis le programme retourne. Cette chaîne de caractères, sans passage à la ligne final, peut être déclarée comme variable globale. Ainsi par exemple, si l'utilisateur encode :

```
./multiplier 12z3 456
```

la chaîne :

```
Le 1er argument n'est pas un nombre valide  
est affichée ;
```

■ si le nombre d'arguments fournis par la ligne de commande est correct ainsi que le premier argument facultatif, mais le second argument facultatif ne représente pas un nombre non signé en base dix (10), la chaîne de caractères :

Le 2è argument n'est pas un nombre valide
suivie d'un passage à la ligne est affichée puis le programme retourne. Cette chaîne de caractères, sans passage à la ligne final, peut être déclarée comme variable globale. Ainsi par exemple, si l'utilisateur encode :

```
./multiplier 123 456P
```

la chaîne :

```
Le 2è argument n'est pas un nombre valide  
est affichée ;
```

■ si les arguments fournis par la ligne de commande sont corrects en nombre et en nature, supposant que le premier argument facultatif est *nb1* et le second *nb2*, alors la chaîne de caractères :

```
nb1 * nb2 = resultat
```

où *resultat* est le résultat du produit de *nb1* par *nb2*, suivie d'un passage à la ligne, est affichée puis le programme retourne. Vous pouvez déclarer deux chaînes de caractères globales " * ", 0 et " = ", 0. De même, la chaîne destinée à accueillir le résultat peut être déclarée comme variable globale non initialisée. Attention : prenez-la assez grande ! Ainsi par exemple, si l'utilisateur encode :

```
./multiplier 123 456
```

la chaîne :

```
123 * 456 = 56088
```

est affichée.