Université Paris Diderot – Paris 7
Langage C (LC4)
Examen du 27 mai 2010 - Durée : 3 heures
Documents, calculettes, ordinateurs et téléphones non autorisés.

Exercice 1. Écrire un programme qui lit au clavier un nombre entier non signé et affiche le nombre de bits de valeur 1 dans sa représentation machine (représentation binaire sur 32 bits). Les entiers non signés peuvent être lus avec le format `u`.
Plus difficile : même question pour compter le nombre de bits valant 1 dans la représentation d'un nombre entier quelconque ou d'un nombre de type `float`.

Exercice 2. On considère le programme suivant :

```
#include <stdio.h>
#define swap1(x,y) {int z; z = x; x = y; y = z;}
void swap2(int x, int y) {int z; z = x; x = y; y = z;}
int main() {
  int a = 2, b = 3; swap1(a,b);
  printf("%d %d\n", a, b);
  a = 2, b = 3; swap2(a,b);
  printf("%d %d\n", a, b);
  return 0;
}
```

Expliquez pourquoi l'appel à `swap1` réalise effectivement l'échange des valeurs de `a` et `b` alors que l'appel à `swap2` ne modifie pas les valeurs de `a` et `b`.

Exercice 3. Qu'affiche le programme suivant (justifiez votre réponse) ?

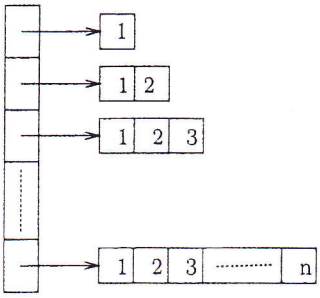
```
#include <stdio.h>
#define f1(x) ((x)++)
void f2(int x) {x++;}
void f3(int *x) {(*x)++;}
int main() {
  int x = 0;
  int *px = &x;
  f1(x); printf("%d\n", x);
  f2(x); printf("%d\n", x);
  f3(&x); printf("%d\n", x);
  f1(*px); printf("%d\n", x);
  f2(*px); printf("%d\n", x);
  f3(px); printf("%d\n", x);
  return 0;
}
```

Exercice 4. L'appel système **stat** permet à un programme de récupérer à une adresse donnée de son espace d'adressage des informations sur un fichier de nom donné.
La fonction qui permet de le réaliser dans un programme C a comme prototype

```
int stat(const char *file_name, struct stat *buf);
```

Ce prototype est défini, ainsi que la structure `stat` (destinée à recevoir les informations sur le fichier) dans le fichier standard `sys/stat.h`.
Indiquer, et dans l'ordre, **tout** ce que devra contenir un programme réalisant un appel à `stat` pour récupérer les informations relatives au fichier contenant le programme en cours d'exécution.

Exercice 5. Écrire, **sans utiliser l'indexation (la solution ne devra donc pas comporter un seul caractère** `[]`), une fonction ayant en argument un nombre entier et renvoyant un élément de type `int **` correspondant à :



Exercice 6. Le but de cet exercice est de mettre en place une représentation des figures à trois dimensions comme ensemble de points de l'espace.

6.1. Définir le type `point` représentant un point par ses trois coordonnées supposées réelles.

6.2. Définir le type `figure3d` permettant la mémorisation d'un ensemble de points quelconques, le nombre total de points de la figure et le nombre de points actuellement initialisés.

6.3. Écrire la fonction

```
figure3d *creerFigure(int n)
```

qui crée une figure de `n` points tous non initialisés et renvoie un pointeur sur cette figure. En cas d'impossibilité de créer la figure, la fonction renverra `NULL`.

6.4. Écrire la fonction

```
void supprimerFigure(figure3D *figure)
```

qui supprime la figure pointée.

6.5. Écrire la fonction

```
figure3d ajouterPoint1(float x, float y, float z, figure3d fig)
```

qui place le point spécifié dans la figure à la première place non initialisée et renvoie la figure ainsi modifiée. La fonction n'insérera le point que si tous les points n'ont pas encore été initialisés. Le prototype ci-dessus est imposé.

6.6. Donner une version `ajouterPoint2` de la fonction précédente dont le type de résultat est `void`. Cette fonction permettra la modification directe de la figure transmise. À vous de choisir les paramètres de cette fonction.

6.7. Écrire une fonction `concatenerFigure` qui, étant données deux figures, construit la concaténation de ces deux figures (libre choix des paramètres et du type du retour) et en permet l'accès.

6.8. Écrire une fonction `void sauverFigure(char *nom, figure *f)` qui sauvegarde une figure dans un fichier texte. Les points composant la figure apparaitront chacun comme une ligne du fichier, les coordonnées d'un point étant séparées par un espace.

Rappels de quelques prototypes

```
void *malloc(int nombreOctets);
void free(FILE *f);
FILE * fopen(char *nom, char *mode);
void fclose(FILE *f);
int fprintf(FILE *f, const char *format, ...);
int fscanf(FILE *f, const char *format, ...);
int fwrite(const void *buf, int taille,  int nombre,  FILE *f);
int fread(const void *buf, int taille,  int nombre,  FILE *f);
```

Exercice 7. On se propose ici d'implanter en langage C quelques fonctionnalités (sous forme simplifiée) offertes par le langage de programmation APL.
Toutes les fonctions qui renvoient des pointeurs renverront `NULL` en cas d'erreur.

7.1. Tout opérateur binaire (par exemple `+`, `*` ou `<`) peut être appliqué à des tableaux d'entiers de même dimension. Un nouveau tableau de même dimension est ainsi construit et un élément du nouveau tableau est le résultat obtenu en appliquant l'opérateur aux deux éléments de même indice dans les deux tableaux arguments.
Ainsi `[4 2 7] + [4 8 2]` est `[8 10 9]` et `[4 2 7] < [4 8 2]` est `[0 1 0]`
Écrire une fonction `appliquer` qui, étant donnés deux tableaux `t1` et `t2` contenant tous les deux `n` entiers et une fonction **alpha** réalisant l'opérateur binaire α, renvoie le tableau `t` de `n` entiers dans lequel `t[i]` a comme valeur `t1[i]`α`t2[i]`

7.2. L'opérateur *laisser* permet de construire un nouveau tableau déduit d'un tableau en y enlevant un certain nombre d'éléments en début ou en fin. Étant donné un tableau `t` de `n` entiers et un entier `k`, le tableau noté `k↓t`, est le tableau déduit de `t` en y enlevant ses `k` premiers éléments si `k` est positif et ses `k` derniers éléments si `k` est négatif.
Ainsi `3↓[3 7 2 8 9 10 4]` est le tableau `[8 9 10 4]` et `-3↓[3 7 2 8 9 10 4]` est le tableau `[3 7 2 8]`
Écrire une fonction `laisser` qui, étant donnés un tableau `t` de `n` entiers et un entier `k` renvoie le tableau correspondant à `k↓t`.

7.3. L'opérateur *ajouter* permet, à partir d'un tableau `t` de `n` entiers et un entier `x`, de construire un nouveau tableau noté `t,x` de `n+1` éléments, identique à `t` pour ses `n` premiers éléments et de dernier élément égal à `x`.
Écrire une fonction `ajouter` qui, étant donnés un tableau `t` de `n` entiers et un entier `x` renvoie le tableau correspondant à `t,x`.

7.4. La *réduction d'un tableau* permet, étant donnés un opérateur binaire α et un tableau `t` de n entiers, de calculer la valeur notée α`/t` de l'expression
`((...((t[0]` α `t[1])` α `t[2])` ... `)` α `t[n-1])`.
Ainsi, `+/t` calcule la somme des éléments du tableau `t`.
Écrire une fonction `reduction` qui, étant donnés une fonction **alpha** réalisant l'opérateur binaire α et un tableau `t` de `n` entiers, renvoie la valeur de α`/t`.

7.5. Le *produit externe de deux tableaux* permet, étant donnés un opérateur binaire α et des tableaux `t1` et `t2` contenant respectivement `n1` et `n2` entiers, de construire un tableau `t` noté `t1` $\circ\cdot\alpha$ `t2`, d'entiers de `n1` lignes et `n2` colonnes tel que $t[i]\,[j]$ a comme valeur `t1[`i`]` α `t2[`j`]`.
Écrire une fonction `produitExterne` qui, étant donnés une fonction `alpha` réalisant l'opérateur binaire α et des tableaux `t1` et `t2` contenant respectivement `n1` et `n2` entiers, renvoie l'élément de type `int **` correspondant au produit externe `t1` $\circ\cdot\alpha$ `t2` des deux tableaux.

7.6. Les cinq fonctions précédentes étant supposées définies respectivement dans les fichiers sources `appliquer.c`, `laisser.c`, `ajouter.c`, `reduction.c` et `produitExterne.c`, construire une bibliothèque d'archive `apl.a` les rassemblant et donner le contenu d'un fichier d'interface `apl.h` permettant d'utiliser correctement cette bibliothèque.

7.7. Application : écrire un programme principal et indiquer comment on construirait le binaire exécutable, ce programme

- lisant au terminal un tableau `ages` de 500 nombres entiers correspondant à l'âge de 500 personnes
- construisant un tableau `tranches` contenant dans cet ordre les valeurs
  `0 16 18 25 30 35 45 50 55 60 65 75`
- recherchant, dans le tableau `ages`, le nombre de personnes dont l'âge est dans chacune des tranches d'âges définies par le tableau `tranches`.
  Le vecteur recherché est obtenu en réalisant successivement :
  - le calcul du tableau `t` valeur de `+/(tranches` $\circ\cdot$ `< ages)`
  - le calcul du tableau `t - ((-1 ↓ t), 0)` qui est le résultat cherché.