Iut Informatique Matière: C & SD Date: 9/11/99 Feuille n°: 1

Cette feuille est à rendre séparément >> **Cette feuille doit être rendue même si elle est inutilisée..<<** En cas de place insuffisante un exercice pourra être prolongé au verso, puis sur une feuille supplémentaire à joindre à celle-ci.

NOM PRENOM GROUPE

I 1) Réécrire chacune des expressions suivantes en supprimant les parenthèses inutiles (c'est à dire les parenthèses que l'on peut supprimer sans changer l'interprétation de l'expression) :

`k=(5* db)/((k/x)*(z-w))`	
`z=((*tpt)[7])`	
`w=(y+=(*(*tpt))*(3+x))`	

2) On considère les déclarations suivantes pour les variables de la première expression :

```
char k,w;
long z;
float x;
double db;
```

Décrire les conversions de type effectuées lors du calcul de cette expression :

II Quel est le nom de l'objet déclaré ou défini par la ligne suivante ; décrire cet objet ; comment le compilateur interprète-t-il précisément l'objet **r** dans cette déclaration :

```
typedef bool (* pfft)(struct cnst xyz, float r[4]);
```

Iut Informatique Matière: C & SD Date: 9/11/99 Feuille n°: 2

Cette feuille est à rendre séparément >> **Cette feuille doit être rendue même si elle est inutilisée..<<** En cas de place insuffisante un exercice pourra être prolongé au verso, puis sur une feuille supplémentaire à joindre à celle-ci.

NOM PRENOM GROUPE

III

1) Ecrire le début d'un programme principal (main) permettant d'appeler une fonction fonct1 pour chacun des arguments de la ligne de commande

```
main(                              )
{




/* la suite du programme */
}
```

2) Donner une déclaration (en style moderne) de la fonction fonct1 sachant que celle-ci ne possède qu'un seul argument.

3) Indiquer (dans la case ci-contre) combien de fois la fonction fonct1 aura été appelée dans ce début de programme, sachant que le programme exécutable s'appelle **prog**, qu'on l'a lancé sous UNIX, dans un répertoire contenant les seuls fichiers **tric.c**, **tric.h**, **tric.bat**, **troc.c**, **tric2**, **tric**, **truc.c**, et ceci par la commande suivante :

```
prog tric.* truc*
```

Cette feuille est à rendre séparément >> **Cette feuille doit être rendue même si elle est inutilisée..<<** En cas de place insuffisante un exercice pourra être prolongé au verso, puis sur une feuille supplémentaire à joindre à celle-ci.

NOM [ ] PRENOM [ ] GROUPE [ ]

IV Dire dans les cadres prévus à cet effet ce qu'affiche le programme suivant sur une machine pour laquelle les pointeurs sont codés sur **4** octets, les 'int' sur **2** octets et les 'float' sur **8** octets, et sachant que les tableaux **tbi** et **tbf** sont respectivement implémentés aux adresses **0x0F3E** et **0x0F54** (rappelons que la spécification d'affichage **%.4X** affiche en <u>hexa</u> sur 4 chiffres minimum, en complétant par des zéros à gauche):

```
#define NBL  9
#include <stdio.h>
```

/* <u>Deux questions en passant</u> : Ce qui suit est ce une déclaration ou une définition ? En style classique ou moderne ? */

```
fonct8 ( float *a , int b )   /* [                    ] */
{
     int i;
     for (i=0 ; i<3 ; i++)
        {
        b += a[i] ;
        a[i+1]  =  b /(int) a[i] ;
        }
}


main()
 {
  int  tbi[NBL] , * pti = tbi + 2  , b ;
  float  tbf[NBL] , * ptf , a[4] ;
  printf( "%d %d %d\n ", sizeof(*pti) , sizeof(ptf) , sizeof(tbf) );
                                                   [              ]

  ptf = tbf + 7 ; * a = 7.2 ;  b = 8 ;
  printf( "%.4x %.4x\n " , pti , ptf - 2 );  [              ]
  fonct8( a , b );
  printf("%g %g %g %d\n ",a[0],a[1],a[2],b);  [              ]
 }
```

Iut Informatique Matière: C & SD Date: 9/11/99 Feuille n°: 4

Cette feuille est à rendre séparément >> **Cette feuille doit être rendue même si elle est inutilisée..<<** En cas de place insuffisante un exercice pourra être prolongé au verso, puis sur une feuille supplémentaire à joindre à celle-ci.

NOM PRENOM GROUPE

V On considère la définition de structure ci-contre, supposée représenter un étudiant :
Le pointeur prec est supposé pointer vers un étudiant précédent déjà défini.

```
struct etud
    {
    char * nom;
    char * prénom;
    char classe;
    int age;
    struct etud * préc;
    }
```

1) Ecrire la définition d'un type **typpetu** correspondant à un pointeur vers un tel étudiant :

2) Ecrire la définition de deux pointeurs **ptetu1**, **ptetu2** du type défini à la première question.

3) On suppose que `sizeof(int)` donne **2** et que `sizeof(int *)` donne **4**.

Quelle valeur donnerait l'expression `sizeof(struct etud)` ?

On souhaite en fait lire une liste d'étudiants à partir d'un fichier texte **etudiants.dat** se présentant comme ci-contre, et construire ainsi une liste chaînée (inverse).

```
Selamer Michel 1A 19
Kaperdu Sancho 1A 19
Ekripar Alain 1A 20
Laf Annette 1A 19
...
```

4) Ecrire ci-dessous (sous la forme de la définition de la macro **QUEST4**) une suite d'instructions qu'il est absolument nécessaire d'exécuter avant de pouvoir utiliser le pointeur **ptetu1** pour stocker un ètudiant.

```
#define QUEST4
```

5) Ecrire ci-dessous (également comme définition de macro) une suite d'instructions qu'il est absolument nécessaire d'exécuter avant de mettre des données dans certains membres de l'objet pointé par **ptetu1**.

```
# define QUEST5
```

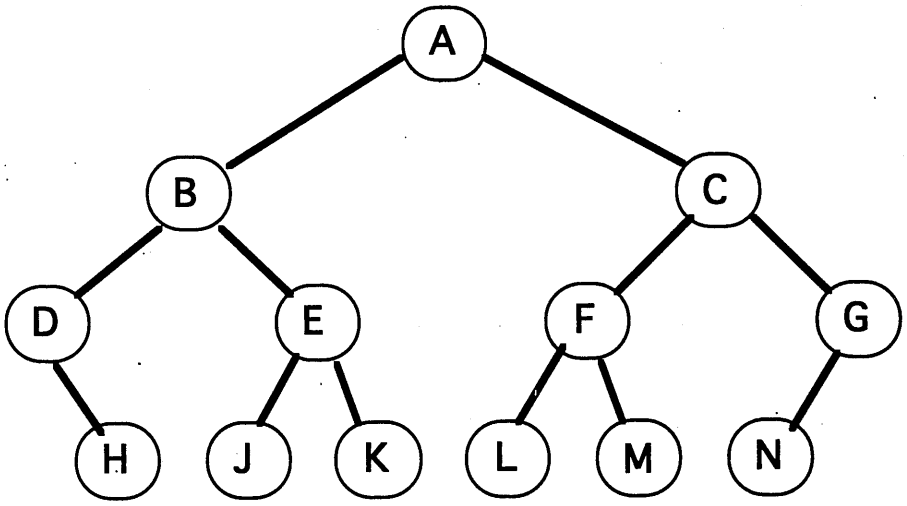
6) Ecrire enfin la fonction **lectfich** complète de lecture du fichier (*Vous pourrez utiliser les pointeurs et macros définis ci-dessus ; Vous pourrez continuer éventuellement au verso de cette feuille si la place est insuffisante*).

Iut Informatique Matière: C & SD Date: 9/11/99 Feuille n°: 5

Cette feuille est à rendre séparément >> **Cette feuille doit être rendue même si elle est inutilisée..<<** En cas de place insuffisante un exercice pourra être prolongé au verso, puis sur une feuille supplémentaire à joindre à celle-ci.

NOM PRENOM GROUPE

VI Indiquer la succession des nœuds visités lorsqu'on parcourt l'arbre suivant en parcours infixé :



Même question dans le cas d'un parcours postfixé :

VII Indiquez dans les cases de gauche, ce qui peut être (**OUI**), ou ne doit pas être (**NON**) dans un fichier **.h**

```
/* monprog.h : fichier d                    du programme monprog */
#define  SIZCH  256
int mafonc2(int a, int b, int c)
      { return (a+b)/ (float)c;}
typedef enum {FAUX,VRAI} bool;
extern  int buff[SIZCH],Tab[500];
int A,B=125;
#define  CARRE(x)   ((x)*(x))
extern C,D;
float mafonction1(int arg1,float arg2);
```

case à compléter

Cette feuille est à rendre séparément >> **Cette feuille doit être rendue même si elle est inutilisée..<<** En cas de place insuffisante un exercice pourra être prolongé au verso, puis sur une feuille supplémentaire à joindre à celle-ci.

NOM PRENOM GROUPE

VIII La solution proposée pour le TP **animaux** comportait un test de cohérence dont voici le listing partiel :

```
void testcohe(ptnod arb, int nn, char * nomfich)
{
 ptnod nod=arb; int i;

 for (i=0 ; i<nn ; i++,nod++) nod->indx=0;    /*
                                                */

 cpt=0;          /* mise à 0 du compteur de noeuds avant parcours */
 parcours(arb,nomfich);          /* test boucles */
 for (i=0,nod=arb ; i<nn ; i++,nod++) /* test noeuds non atteints et...*/
      {

           {
           fprintf(stderr,"erreur arbre (%s) : noeud %d (%s) non atteint\n",
                                                nomfich,i+1,nod->text); err=3;

           }
      else                ; /* ... remise à 0 des index (pour le parcours final) */

      }
 if (err) exit(err);
}
```

```
struct noeud
  { char * text;
    int indx:
    struct noeud * oui;
    struct noeud * non;
  }
```

On rappelle que la structure donnée dans l'énoncé a été complétée avec un membre **indx** comme indiqué ci-contre à gauche et que ce test de cohérence est appelé juste après le chargement de l'arbre initial dans un tableau, à partir du fichier **animaux.dat**.

```
17
2  11  Est ce un mammifère
3   8  Est ce un ruminant
4   7  Est ce un animal domestique
5   6  Donne t'il de la laine
un mouton
une vache
un buffle
9  10  A t'il des griffes rétractiles
un chat
un chien
12 15  Vit il dans l'eau
13 14  Est ce un poisson
une truite
une écrevisse
16 17  Est-ce un oiseau
un pinson
une abeille
```

On rappelle aussi que ce test est fait en deux étapes :

- un parcours récursif de l'arbre pour indexer les noeuds (de 1 à nn) et vérifier en passant que chaque noeud est atteint au maximum une fois,
- une vérification que tous les noeuds ont été atteints.

Le parcours est fait par l'appel à une fonction **parcours**() dont le rôle principal est d'indexer les noeuds pour préparer l'écriture du fichier dont la forme est rappelée ci-contre à droite.

1) Compléter le listing de la fonction **testcohe**() ci-dessus.

2) Quel sorte de parcours de l'arbre doit être fait par la fonction **parcours**() ?

parcours

3) Ecrire ci-dessous la fonction **parcours**() .

```
void parcours(ptnod arb,char * nomfich)
{
```

si vous continuez au verso, mettre une flèche dans cette case