Université Paris 7
Licence 2, Développement en C.
3 mai 2007
Durée : 3 heures. Documents manuscrits, notes de cours, notes de TD/TP autorisés. Livres interdits.

Le sujet comporte 5 pages.

Votre code doit être écrit de façon lisible, avec des indentations et des accolades appropriées permettant de voir la fin de blocs de code (fin de boucles, etc.).

QUESTIONS

Question 1: Qu'est-ce que affiche le programme suivant ?

```
#include <stdio.h>
#include <string.h>

int main(void){
  char s[]="azertyqwty";
  char *d;
  int i,j,k;
  i=3;
  printf("B=%c\n", s[i++] );
  j=5;
  printf("C=%c\n", s[++j] );
  printf("longueur s=%d\n", strlen(s) );
  d=s+4;
  printf("longueur d=%d\n", strlen(d) );
  printf("D=%c\n", d[-2] );
  return 1;
}
```

Question 2: Soit

```
1 char *aaa(void *, int);
2 char *(*bbb)(void *, int);
3 a=b;
4 b=a;
```

un fragment d'un fichier source.

(A) Qu'est-ce que déclarent les lignes 1 et 2 ? (Déclaration d'une fonction ? Déclaration d'une variable ? Si une déclaration d'une variable alors quel est le type de cette variable ?)

(B) Pour chaque l'instruction dans les lignes 3 et 4 écrire si cette instruction est correcte ou non et pour une instruction correcte expliquer ce qu'elle est censée faire.

EXERCICES

Chaînes de caractères

Soit

```
struct elem{
    struct elem *suivant;
    char *val;
};
typedef struct elem* liste;
```

Le type `liste` représente une liste simplement chaînée de chaînes de caractères.
Dans cet exercice on suppose que les éléments sur la liste ce sont des mots (chaînes de lettres).
Exercice 1 Écrire la fonction

```
char *to_phrase(liste l)
```

qui retourne une chaîne de caractères obtenue en concaténant les mots sur la liste `l` et en les séparant par le caractère d'espace. Par exemple si la liste `l` est composée de trois mots "`le`", "`jeu`", "`infini`" alors la fonction produira la chaîne "`le jeu infini`".

Arbres

On définit le type `arbre` pour les arbres binaire :

```
struct elem_arbre{
    struct elem_arbre *fils_gauche;
    struct elem_arbre *fils_droit;
    void *valeur;
};
typedef struct elem_arbre *arbre;
```

Le champs `valeur` donne le pointeur vers une information associée avec un sommet.
Exercice 2 Écrire une fonction (récursive)

```
int nb_sommets(arbre a)
```

qui retourne le nombre de sommets de l'arbre `a`.

Listes doublement chaînées

Nous définissons

```
struct monome_elem{
    int degre;
    double coef;
};
typedef monome_elem *monome;
```

La structure `struct monome_elem` ci-dessus sera utilisée pour stocker les monômes à une variable x. Le type `monome` désigne un pointeur vers cette structure.

Exemple : le degré `degre` du monôme $-3.7x^4$ est 4 et son coefficient `coef` est -3.7.
Un polynôme sera codé avec une liste doublement chaînée de monômes, le type `polynome` est défini comme ci-dessous :

```
struct elem{
    struct elem *suivant;
    struct elem *precedent;
    monome       pval;
};
typedef struct elem *polynome;
```

à nombre variable d'arguments, tous de type `polynome` qui calcule et retourne la somme de tous ces polynômes. On supposera que tous les pointeurs sur la liste des argumets de `add_polys`, sauf le dernier, sont différents de `NULL`, c'est-à-dire le pointeur `NULL` marque la fin de la liste d'arguments.
Par exemple pour additionner trois polynômes `p`, `q`, `r`, tous les trois non `NULL`, on fera l'appel `add_polys(p,q,r, (polynome) NULL)`.

Exercice 5 Écrire une fonction

```
polynome tab_to_poly(int n, double tab[])
```

qui prend un tableau `tab` de `n` éléments et construit un polynôme tel que, pour tout indice `i`, `tab[i]` est le coefficient du monôme x^i du degré i. Notez que dans le tableau `tab` certains coefficient peuvent être 0 mais le polynôme construit doit être sous forme réduite.

Exercice 6 Écrire une fonction

```
double *poly_to_tab(polynome p)
```

inverse à la fonction de l'exercice 5. Cette fonction à partir d'un polynôme construit un tableau de ses coefficients et retourne le pointeur vers le premier élément de ce tableau.
Par exemple pour le polynôme `p` de la figure 1 la fonction `poly_to_tab` retournera le tableau

0	1.2	0	0	−4.6	0	0	3.1
0	1	2	3	4	5	6	7

Exercice 7 Écrire une fonction

```
polynome derivee(polynome p)
```

qui calcule et retourne la dérivée du polynôme `p`.
Rappel : $\frac{d}{dx}(x^i) = i \cdot x^{i-1}$

Exercice 8 Écrire une fonction

```
polynome supprimer_ieme(polynome p, int i)
```

qui supprime le monôme du degré i dans le polynôme `p`.
Remarque : Cette fonction doit effectivement supprimer le monôme correspondant dans la liste `p` (si ce monôme existe) et retourner le pointeur vers le premier élément de la liste après la suppression du élément désigné.

Exercice 9 Écrire une fonction

```
polynome add_monome(polynome p, monome m)
```

qui fait la somme du polynôme `p` et du monôme `m`. Contrairement à la fonction `add` de l'exercice 3 la fonction `ajouter_monome` ne produit pas une nouvelle liste mais modifie la liste `p` et retourne le pointeur vers le premier élément de la liste modifiée.
Par exemple si $p(x) = 4.1x^2 - 6.6x^6 + 5.8x^8$ alors l'addition du monôme $2.1x^3$ ajoutera un