

Les premières questions sont l'occasion de rappeler le fonctionnement d'un processus (notions de pile d'exécution et de tas). Les listes chaînées sont reprises dans plusieurs TP (listes de `char`, mais aussi listes de chaînes de caractères). Il est important de comprendre les implantations et les fonctions de base à leur sujet.

## 1 Allocation dynamique

**Question 1.** On souhaite affecter à une variable  $p$  une zone mémoire allouée dynamiquement. Pour chacun des cas suivants, donner la déclaration de  $p$  et l'instruction qui effectue l'allocation dynamique : un tableau de dix `double` ; un tableau de cinq structures de type `struct rationnel` ; une zone capable de recevoir la chaîne de caractères "elephant".

**Question 2.** Chacune des fonctions suivantes gère la mémoire de façon incorrecte. Dans chaque cas, indiquer l'erreur, le comportement à l'exécution et proposer une correction. Indiquer les directives d'inclusion `#include` manquantes.

```

/* a.c */
int main ()
{   int* x;
    scanf ("%d", x);
}

/* b.c */
char* zone (void)
{   char tab [256];
    return tab;
}

int main ()
{   char* t;
    t = zone ();
    strcpy (t, "elephant");
}

/* c.c */
char* zone (void)
{   char* tab;
    tab = (char*)malloc (256 * sizeof (char));
    return tab;
}

int main ()
{   char* t;
    t = zone ();
    strcpy (t, "elephant");
}

/* d.c */
int main ()
{   double *t;
    int i;
    t = malloc (10);
    for (i = 0; i < 10; i++)
        t [i] = 3.14;
    free (t);
}

```

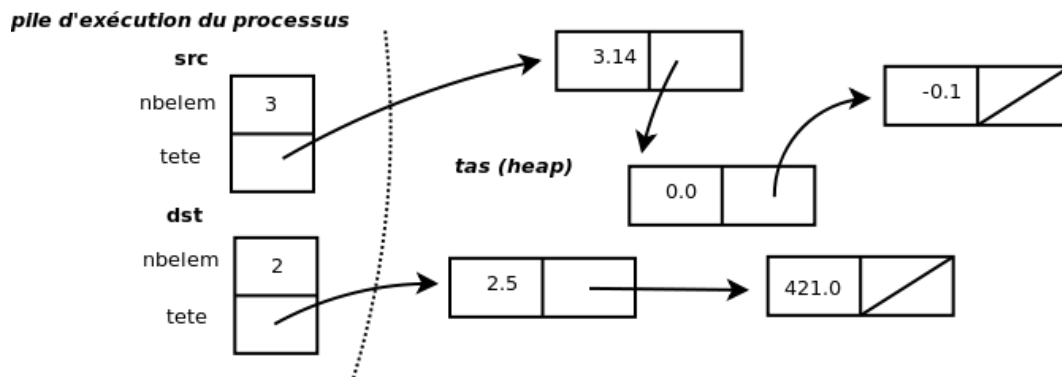
## 2 Algorithmique des listes chaînées

On rappelle l'implantation des listes chaînées.

```
struct maillon_double {
    double valeur;
    struct maillon_double* suivant;
};

struct liste_double {
    struct maillon_double* tete;
    int nbelem;
};

#define NIL (struct maillon_double*)0
```



Les fonctions qui suivent seront utiles pour implanter les piles et les files au moyen de listes.

**Question 3.** Écrire une fonction `ajouter_en_queue_liste_double`, paramétrée par une liste  $L$ , un double  $d$  et qui ajoute  $d$  en queue de  $L$ .

**Question 4.** Modifier l'implantation du type `struct liste_double` en lui ajoutant un nouveau champ, `queue`, qui pointe en permanence sur le dernier maillon de la liste. Réécrire les fonctions qui doivent l'être.

**Question 5.** Écrire une fonction `inserer_liste_triee_double`, paramétrée par une liste  $L$  supposée triée par ordre croissant, un double  $d$  et qui insère  $d$  à la bonne place dans  $L$ .

### Recherche dans une liste

Les questions qui suivent seront utiles lors des TP sur les tables de hachage.

**Question 6.** Écrire une fonction `rechercher_liste_double`, paramétrée par un double  $d$ , une liste de doubles  $L$ , qui retourne `true` si  $d \in L$  et `false` sinon.

**Question 7.** Dans le cas où  $d \in L$ , il peut être utile, non seulement de le savoir, mais aussi de retourner à la fonction appelante l'adresse du maillon qui contient  $d$ . Adapter `rechercher_liste_double` en ce sens.

**Question 8.** Écrire une version de la fonction `rechercher_liste_double` pour le cas où la liste  $L$  est triée.

**Question 9.** Comment savoir, sans la parcourir, si une liste est triée ? Si tous ses éléments ont été ajoutés par `insérer_liste_triee_double` alors on est sûr que la liste est triée. Comment modifier l'implantation des listes pour mémoriser cette information ? Comment modifier `rechercher_liste_double` pour en tirer parti ?

## Compléments

**Question 10.** Même question pour des listes avec des maillons à double chaînage : chaque maillon dispose non seulement d'un champ `next` qui pointe vers le maillon suivant, mais aussi d'un champ `prev`, qui pointe vers le maillon précédent.

**Question 11.** Adapter le type `struct liste_double` pour obtenir des listes de `char`.

La question suivante est utile pour de nombreux projets de SD/graphes.

**Question 12.** Adapter le type `struct liste_double` pour obtenir des listes de chaînes de caractères. Discuter de l'intérêt de dupliquer les chaînes de caractères lors des ajouts et des insertions. Penser au destructeur.