Systèmes Embarqués 1 & 2

Classes T-2/I-2 // 2018-2019

a.07 – C - Pointeurs

Exercices

Exercice 1

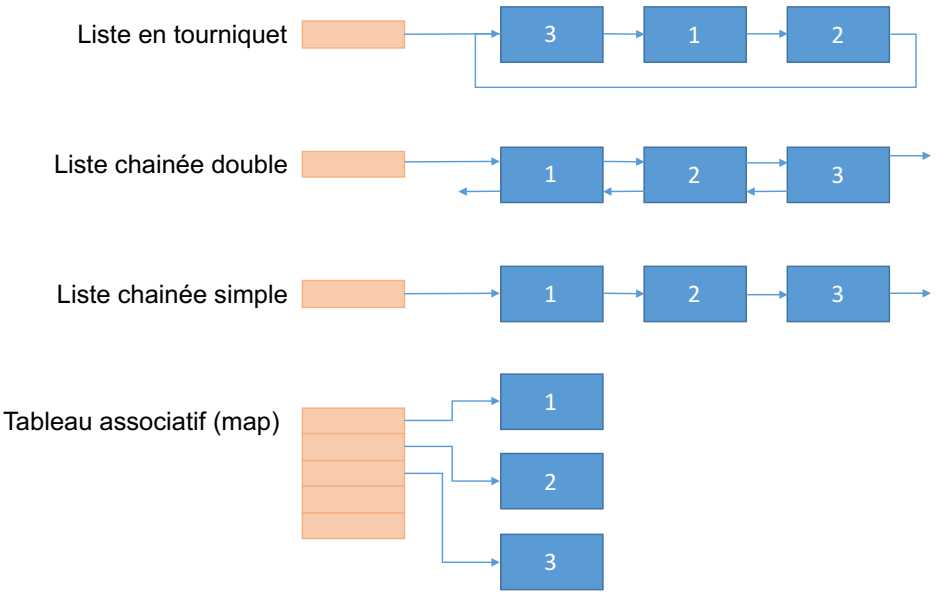
Indiquer la valeur des pointeurs pour les expressions ci-dessous

```
struct a_struct {int a; int b; int c;}; //int est codé sur 32 bits
    /* 1 */ struct a_struct* pa = (struct a_struct*)1000;
    /* 2 */ struct a_struct* pb = &pa[10]
    /* 3 */ struct a_struct* pc = pa + 10;
    /* 4 */ struct a_struct* pd = pc--;
    /* 5 */ struct a_struct* pe = ++pb;
```

Exercice 2

Pour chacun des types de listes chainées ci-dessous, développer un programme permettant de créer, supprimer, insérer, rechercher et extraire des objets du type

```
struct element {
    int key;
    int value;
};
```



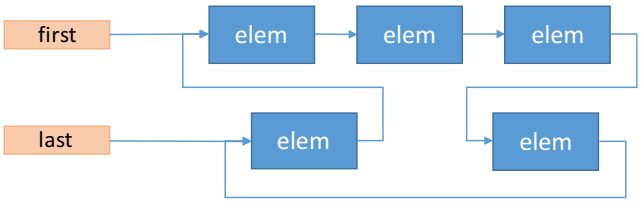
Citer les avantages et désavantages de ces types de listes.
Citer d'autres représentations.

Exercice 3

Soient les deux structures suivantes

```
struct elem {
    char name[100];
    struct elem* next;
};

struct container {
    struct elem* first;
    struct elem* last;
};
```

et la liste chainée correspondant au schéma suivant



coder en C les fonction suivantes

(a) fonction capable de compter le nombre d'éléments

```
/**
 *@param container pointeur sur la structure container
 *@return nombre d'éléments
 */
int get_nb_of_eles (const struct container* container);
```

(b) fonction capable d'enlever un élément sans casser la chaine

```
/**
 *@param container pointeur sur la structure container
 *@param index position dans la chaine
 *@return pointeur sur l'élément enlevé, sinon 0
 */
struct elem* delete (struct container* container, int index);
```

Exercice 4

Considérer la structure suivante

```
struct node {
   unsigned char ident;
   long          size;
   struct node*  next;
};
```

Ces structures de données peuvent être chainées les unes aux autres de façon simple. La fin de la chaine sera identifiée par un camp *“next”* à *“null” (0)*.

Ecrire une fonction C qui aura pour tâche de retrouver la structure dont le champ *“ident”* (qui d’ailleurs est unique) est égal au paramètre *“id”*. Cette fonction devra répondre à l’interface suivant

```
/**
 *@param list pointeur sur le premier élément de la chaine
 *@param id identification de l’élément à rechercher (champ ident de la structure)
 *@return référence sur la structure recherchée en cas de succès et null (0) si aucun
 *        élément n’a été découvert
 */
struct node* search (struct node *list, unsigned char id);
```

Remarque : si la chaine est vide le paramètre *“list”* sera *“null” (0)*.

Exercice 5

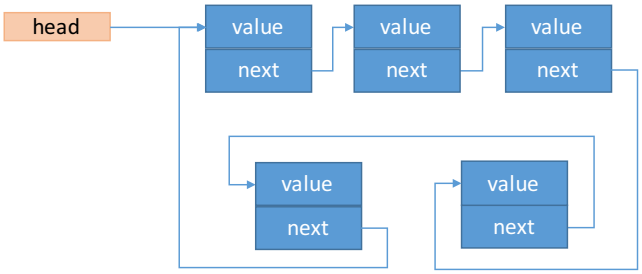
Soit les déclarations de variables et les initiations suivantes

```
#define ARRAY_SIZE(x) (sizeof(x)/sizeof(x[0]))

char   str1[]=”Programmation”;
char   str2[]=”Algorithmique”;
char   str3[]=”Microprocesseurs”;
char   str4[]=”Physique”;
char*  tab[] = {str1, str2, str3, str4};
char** ptab = tab;

int main()
{
    char* max = max_str(ptab, ARRAY_SIZE(tab));
    printf (”longest string=%s\n”, max);

    return 0;
}
```

Coder en C la fonction *“max_str”* capable de retourner la chaine de caractères (son adresse) la plus longue de celles contenues dans le tableau *“tab”*.

Exercice 6

Soit la liste chainée suivante



Ainsi que les déclarations suivantes

```
struct elem {
    short value;
    struct elem* next;
};

struct result {
    short max;
    short min;
};
```

Ecrire une fonction C capable de définir la valeur maximale et minimale *(short)* de la liste chainée présentée ci-dessus. Le retour se fera par référence *(pointer)* sur une structure de type *“struct result”* dans les champs *“max”* et *(min)*.

Conditions : si la chaine est vide, le pointeur *“first”* contient une valeur nulle (0).

Prototypage de la fonction

```
void compute_min_max (const struct elem* first, struct result* result);
```

Exercice 7

Voici la déclaration d'une union

```
struct str {
    uint16_t    val;
    struct str* next;
};

union exemple {
    uint32_t    word;
    uint16_t    hword;
    uint8_t     byte;
    struct str  elem;
} u;
```

Si *"u"* est logé à partir de l'adresse 0x2000, représentez pour chaque affectation les octets (sous forme hexadécimale) en mémoire

(a) u.word = 1536;

(b) u.hword = -3;

(c) u.byte = 96;

(d) u.elem.val = 127;

(e) u.elem.next = &u;

address	a)	b)	c)	d)	e)
0x2000					
0x2001					
0x2002					
0x2003					
0x2004					
0x2005					
0x2006					
0x2007					
0x2008					
0x2009					
0x200a					
0x200b					
0x200c					
0x200d					
0x200e					
0x200f					
0x2010					
0x2011					
0x2012					

P.S. les éléments mémoire dont on ne connait pas le contenu sont à marquer d'une croix