15 Bien les listes. Revois !

Mercredi 25 Juin

DE Programmation C-2

25/3

Exercice 1. Taille d'une chaine de caractères

1) int longueurChaine (char * s); const 0,25/0 et s == NULL ?

2) int longueurChaine1 (const char * s)

```
{
    int i = 0;
    while (s[i] != '\0')
        i++;
    i++;            // on compte le '\0'.
    return i;
}
```

0,75/1

Nul ! faire longueur 4 sockets en mémoire.

3) int longueurChaine2 (const char * s)

```
{
    char * pcourant;
    pcourant = s;
    int i = 1;
    for (; *pcourant != '\0'; pcourant++)
        i++;
    return i;
}
```

pcourant ! c'est 1 copie !

0,5/0,5

4) int longueur chaine 3 (const char * s)

```
{
    if (*s == '\0')
        return 0;
    return 1+longueur chaine 3 (s+1);
}
```

1/1

Exercice 2. Ensemble d'entiers dynamique 10/11

Question 1. Cours 2/2

1) (void *) malloc (size_t n) ;

la fonction malloc alloue de la mémoire dans le "tas" ("heap") si c'est possible, n octets, et renvoie un pointeur universel pointant vers cette zone allouée dynamiquement. on s'en sert pour créer des tableaux de taille que l'on calcule.

2) void* realloc (void* p, size_t n) ; ou diminuer !!

la fonction realloc sert à agrandir un tableau alloué dynamiquement au préalable, passé en paramètres par un pointeur universel, et il renvoie l'adresse de cette nouvelle zone mémoire dynamique. Soit il crée une nouvelle zone mémoire de la bonne taille, copie l'ancien tableau puis le libère de la mémoire, soit il "prolonge" le tableau si c'est possible.

Dans les deux fonctions, un échec renvoie NULL.

Question 2. Création

1) int Insertion 1 (int tab[], int* taille_utile, int taille, int valeur)

```
{
    if (taille == *taille_utile)        return 0;
```

```
int i = 0;
while ( i<*taille_utile || tab[i] > valeur)
{
  if( tab[i] == valeur)
      return 0;
  i++;          // on repère où insérer la valeur.
}
int j;
for (j= *taille_utile - 1 ; j != i-1; j--)
      tab[j+1]= tab[j];          // décalage des valeurs.
tab[i] = valeur;
*taille_utile ++;
return 1;
}
```

i+1

2/2

2) `int Creation( Ensemble * ensemble)`

```
{
  if(ensemble == NULL)
      return 0;
  if((ensemble -> cardinal != 0) && (ensemble -> nombreMax != 0))
      return 0;          // l'ensemble n'est pas vide.
  printf (" Nombre maximal d'elements de l'ensemble: ");
  scanf (" %d", &(ensemble -> nombreMax));
  (void*)(ensemble -> elements) = (int *) malloc ((ensemble -> nombreMax) * sizeof(int));
  if(ensemble -> elements == NULL)
      return 0;
  return 1;
}
```

le CONTRAIRE !! (voir la fonction EstVide_)

Confusion

et l'insertion !!

1/2

3) `int Insertion2 (int tab[], int * taille_utile, int taille, int valeur)`

```
{
  if(*taille == *taille_utile)
```

*

ERREUR !!

```
{
    int * p = NULL;
    (void*) p = (int*) realloc(tab, (taille + 10) * sizeof (int));
    if (p == NULL)
        return 0;
    tab = p;
    taille += 10;
return insertion1( tab, taille - utile, taille, valeur);
}
```

1/1,5

Question 3. Ensemble vide

1/1

```
int EstVide (const Ensemble * ensemble)
{
  if (ensemble == NULL)
      return 1;
  if ((ensemble -> cardinal == 0) && (ensemble -> nombre_max == 0) && (ensemble -> elements == NULL))
      return 1;
  return 0;
}
```

plus lisible !

Question 4. Appartenance.

1/1

```
int Appartient (const Ensemble * ensemble, int valeur)
{
  int test;
  test = EstVide (ensemble);
  if (test == 1)
      return 0;
  test = 0;
  while (test != ensemble -> cardinal)
  {
    if ((ensemble -> elements)[test] == valeur)
        return 1;
    test++;
  }
  return 0;
}
```

Attention : la variable test sert à 2 choses différentes ⇒ PEU LISIBLE !

Remède : 2 variables Entier OU BIEN une seule variable de boucle et `if (EstVide (ensemble) == 1) return 0;`

Question 5. Ajout.

```
int Ajout (const Ensemble * ensemble, int valeur)
{
  return Insertion1(ensemble->elements, &(ensemble->cardinal), ensemble->nombreMax, valeur);
}
```

et si ensemble == NULL ?

0,5/0,5

Question 6. Libération.

```
void Vide (Ensemble * ensemble)
{
  free(ensemble->elements);
  ensemble->cardinal = 0;
  ensemble->nombreMax = 0;
  ensemble->elements = NULL;
}
```

il faut aussi vérifier que ce pointeur ≠ NULL

Utile !

1/1

Exercice 3. Listes d'entiers.

2,5/6

1)
```
void doubleVal(Node * l)
{
  l->val = 2 * (l->val);
  if (l->next == NULL)
    return;
  doubleVal(l->next);
}
```

si l == NULL ? → oui

récursif, oh

0,75/1

2)
```
int lengthList (Node * l)
{
  if (l->next == NULL)
    return 1;
  return 1 + lengthList(l->next);
}
```

count

il y a une erreur !

0,5/0,5

3)
```
void reversePrintListRec(Node* l)
{
  if (l->next == NULL)
    printf("%d\n", l->val);
```

return ?

```
    reversePrintListRec(l->next);
    printf("%d\n", l->val);
}
```

oui

1/1

6)
```
void removeFirst(Node * l, int valeur)
{
    Node * tempo = NULL;
    for(tempo = l; tempo != NULL; tempo = tempo->next)
    {
        if(tempo->val == valeur)
        {
```

Inachevé !!

0,5/1

Question 1 : 4) 0/0,5

5) 0/1

7) 0/1