Série de TD et TP – 09

[Les Listes chainées]

---------- Semaine N° 1 ----------

Exercice 1

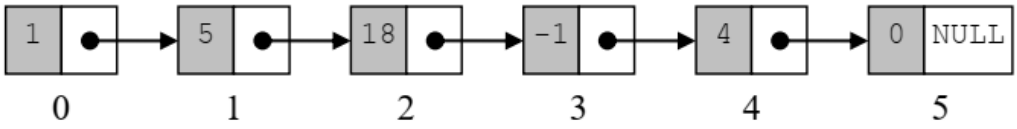
Écrivez une fonction ***int longueur_liste(liste l)*** qui renvoie le nombre d'éléments présents dans la liste l passée en paramètre.

Solution

```
int longueur_liste(liste l)
{ element *temp = l.debut;
int resultat = 0;
while(temp != NULL)
{ resultat++;
temp = temp->suivant;
}
return resultat;
}
```

Exercice 2

Écrivez une fonction ***int recherche_valeur(liste l, int v)*** qui recherche une valeur v dans une liste simplement chaînée l (qui n'est pas ordonnée). La fonction doit renvoyer la position de la première occurrence de la valeur dans la liste, ou -1 si la liste ne contient pas d'éléments de valeur v.
Exemple :



Si on recherche la valeur -1 dans cette liste, la fonction doit renvoyer 3. Si on recherche la valeur 99, elle doit renvoyer -1.

Solution

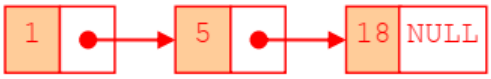
```
int recherche_valeur(liste l, int v)
{ int i = 0;
int pos = -1;
element *temp = l.debut;
while(pos==-1 && temp!=NULL)
{ if(temp->valeur == v)
pos = i;
else
{ temp = temp->suivant;
i++;
}
}
return pos;
}
```

Exercice 3

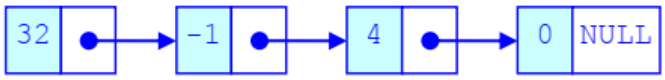
Écrivez une fonction ***void fusionne_listes(liste *l1, liste l2)*** qui prend en paramètres un pointeur sur une liste simplement chaînée l1, et une liste simplement chaînée l2. La fonction doit rajouter la liste l2 à la fin de la liste l1.

Exemple :

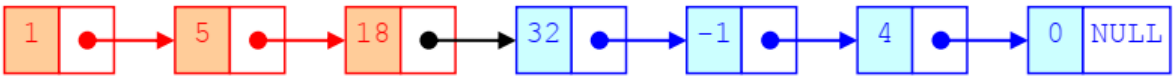
liste **1** avant la fusion



liste **2**



liste **1** après la fusion



Solution

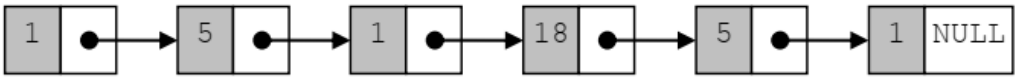
```
void fusionne_listes(liste *l1, liste l2)
{ element *temp = l1->debut;
// si la liste 1 est vide, la fusion est la liste 2
if(temp == NULL)
l1->debut = l2.debut;
else
{ // sinon on va à la fin de la liste 1
while(temp->suivant != NULL)
temp = temp->suivant;
//et on rajoute la liste 2
temp->suivant = l2.debut;
}
}
```

Exercice 4

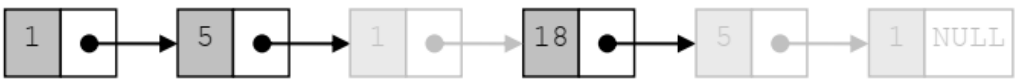
Des doublons sont des éléments d'une liste qui ont la même valeur. Écrivez une fonction void supprime_doublons(liste l) qui prend une liste simplement chaînée l en paramètre, et la traite de manière à ce que la même valeur n'apparaisse pas deux fois (i.e. il faut supprimer des doublons en trop)

Exemple :

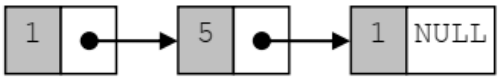
liste de *départ*



suppression des doublons



liste *résultat*



Solution

```
void supprime_doublons(liste l)
{ element *temp=l.debut,*temp2;
```

```
element *avant,*apres;
while (temp!=NULL)
{ avant = temp;
temp2 = avant->suivant;
while (temp2!=NULL)
{ apres = temp2->suivant;
if (temp->valeur == temp2->valeur)
{ avant->suivant = apres;
free(temp2);
}
else
avant = temp2;
temp2 = apres;
}
temp = temp->suivant;
}
}
```

Exercice 5

On dit qu'une liste l2 préfixe une liste l1 si l2 apparaît entièrement au début de l1.
Exemple :

l1	l2	préfixe ?
{1, 2, 4, 8, 0, 3}	{1, 2, 4}	oui
{1, 2, 4, 8, 0, 3}	{1}	oui
{1, 2, 4, 8}	{1, 2, 4, 8}	oui
{}	{}	oui
{1, 2, 4, 8, 0, 3}	{}	oui
{1, 2, 4, 8, 0, 3}	{5, 6}	non
{1, 2, 4, 8, 0, 3}	{4, 8, 0}	non
{1, 2, 4, 8, 0, 3}	{1, 2, 4, 8, 0, 3, 7}	non

Écrivez une fonction récursive ***int est_prefixe(liste l1, liste l2)*** qui prend deux listes l1 et l2 en paramètre, et renvoie un entier indiquant si l2 est préfixe de l1. L'entier doit valoir 1 pour oui et 0 pour non.

Solution

```
int est_prefixe(liste l1, liste l2)
{ element *temp1=l1.debut, *temp2=l2.debut;
liste l1_bis, l2_bis;
int resultat;
if (temp2==NULL)
resultat = 1;
else
if (temp1==NULL)
resultat = 0;
else
if (temp1->valeur == temp2->valeur)
{ l1_bis.debut = temp1->suivant;
l2_bis.debut = temp2->suivant;
resultat = est_prefixe(l1_bis,l2_bis);
}
else
resultat = 0;
return resultat;
}
```

Exercice 6

Écrivez une fonction int copie_liste(liste l1, liste *l2) qui réalise une copie de la liste l1. Cette copie doit être renvoyée via le paramètre l2, qui doit être initialement une liste vide. La fonction renvoie −1 en cas d'erreur et 0 en cas de succès.

Solution

```
int copie_liste(liste l1, liste *l2)
{ element *temp=l1.debut,*e,erreur=0;
int i=0;
while(temp!=NULL && erreur!=-1)
{ if((e=cree_element(temp->valeur))==NULL)
erreur = -1;
else
{ temp = temp->suivant;
insere_element(l2, e, i);
i++;
}
}
return erreur;
}
```

Exercice 7

Écrivez une fonction ***void inverse_liste(liste *l)*** qui inverse la liste doublement chaînée l passée en paramètre.

Solution

```
void inverse_liste(liste *l)
{ element *temp = l->debut, *temp2;
if(temp != NULL)
{ l->debut = l->fin;
l->fin = temp;
while(temp != NULL)
{ temp2 = temp->suivant;
temp->suivant = temp->precedent;
temp->precedent = temp2;
temp = temp2;
}
}
}
```