Algorithmique

TD N°3 : Listes Piles, Listes Chaînées

Listes Piles

1.

```
Fonction Swap (l : liste) retourne Liste
	Si EstVide(l) ou estVide(suite(l)) alors
		Rendre l
	Fsi
	Elt1 = premier(l)		// elt1← le premier élément de l
	Elt2 = premier(suite(l))	// elt2 ← le 2e élément de l

	depile(l)
	depile(l)
	empile( elt1, l)		// L2 ← elt1 :: L2
	empile( elt2, l)		// L2 ← elt2 :: L2

	retourne l
```

2.

```
procedure affiche (l)
	L2 = l
	tant que non estvide(L2) faire
		afficher premier(L2)
		L2 = suite (L2)
	ftq
```

3.

```
// Affiche une liste à partir de la fin.
procedure ehciffa (l)
	si non estVide(l) faire
		ehciffa(suite(l))
		premier(l)
```

4.

```
fonction kieme(k : entier ,l : Liste d'elts) retourne elts
	entier : cpt ← 0
	elts : eltk
	booleen : sarretter ← 0
	si estVide(l) alors
		rendre inexistant
	fsi
	L2 = l
	tant que cpt ≤ k et non sarretter faire
		si cpt=k faire
			eltk = premier(L2)
		fsi
		si estVide(L2) faire
			sarretter ← 1
		fsi
		depile(L2)
```

```
            cpt ++
      ftq
      si sarretter faire
            rendre inexistant
      fsi
      rendre eltk
```

5.

Procédure

```
Procédure ajoutetriee (l : liste d'élément (inout), e : élément )
      L2←l
      L3 ← []
      Tant que non estVide(L2) faire :
            Si premier(L2) ≥ e alors
                  Empile (e,L3)
                  Tant que non estVide(L3)  faire
                        Empile(premier(L2), L3)
                        Depile(L2)
                  Ftq
             sinon
                  empile(premier(L2),L3)
                  Depile(L2)
            Fsi
      Ftq
      L=L3
```

fonction :

```
fonction ajouteTriee(l :liste, e : élément ) retourne liste
      L2←l
      L3 ← []
      Tant que non estVide(L2) faire :
            Si premier(L2) ≥ e alors
                  Empile (e,L3)
                  Tant que non estVide(L3)  faire
                        Empile(premier(L3), L2)
                        Depile(L2)
                  Ftq
             sinon
                  empile(premier(L2),L3)
                  Depile(L2)
            Fsi
      Ftq
      Rendre L3
```

recursif :

```
      Si estVide(L) ou x<premier(l)
      alors rendre ajoute(x,l)
      Sinon rendre ajoute(premier(L), ajoutetriee(x, suite(L))
```

6.

```
premierePosition(l :liste, e :élément )
	si estVide(l)
		rendre 0
	si premier(l) = e
		alors rendre 1
	sinon
		x← premiereposition(suite(l),e)
		si x ≠ 0 alors
			rendre x + 1
		Sinon
			Rendre 0
```

7.

```
tueDoublonDansTriee(l : liste(inout))
	l1← l
	lres←[]

	si estVide(l)
		rendre l
	sinon
		prec←premier(l1)
		tant que non estVide (l1) alors
			depile(l1)
			si prec ≠ premier(l1) alors
				ajoutetriee(premier(l1), lres)
			fsi
			prec = premier(l1)
		ftq
		rendre l←lres
	fsi
```

8. Mj

```
// supprime les éléments en double. Garde les premières occurrences
tueDoublon (inout l)
	si non estVide(L) alors
		// rend vrai si l'élément est dans le tableau, faux sinon.
		si estDans(Premier(L), suite(L))
			depile(L)
			tueDoublon(L)
		sinon
			tueDoublon(PointeurSuite(L))


// supprime les éléments en double. Garde les premières occurences
Fonction tueDoublons(l :liste , P/D)
	Si ¬ estVide(l)    alors
		tueDoublons(pointeurSuite(l)
		elimine(Premier(L),pointeurSuite(L)

fonction elimine (IN : x, inout : liste l)
```

Listes chainées

1.

a. initialiseNull(List *L)
b. TestNul(Liste L)
c. Premier(Liste L)
d. Empile(int x, List *L)

```
Liste l2← malloc (sizeof(*L))
*L2.valeur ← x
*l2.suite ← L
*L←L2
```

e. Depile(List *L)

```
Si *L=null alors
	Erreur
Sinon
	Tmp ← *L
	*L ←  (*L) →suite
	Free(tmp)

PointeurSuite(Liste L)
	Si L=null alors
		Erreur
	Sinon
		Rendre 2(L→suite)
```

2.

```
Swap (*L)
	Si L.suite ≠ null && L.suite.suite ≠ null
		Tmp ← *L.suite
		*L.suite ← *L.Suite.suite
		*Tmp .suite ← *L
	Sinon
		Erreur
```

3.

```
freeAll(*L)
	si *L≠ Null
	alors
		freeAll(&(*L).suite)
		free(*L)
```

4.

```
ehciffa(Liste L)
	ffa(L)
	print ‘\n’;
ffa(Liste L)
	si !estVide(L)
	ffa(suite(L))
	print premier(L)
```

5.

```
prox AjouteTrie(IN : x, inout L)
	si ¬ estvide(L) et x<premier(L) alors
		empile(x,L)
	sinon ajoutertrie(x,*L.suite)
```

6.

```
tueDoublon(Liste *L)
	si !estVide(*L) alors
		si estDans(L→valeur, L→suite) alors
			depile(L)
			tueDoublon(L)
		sinon
			tueDoublone (L→suite)
```