TD4

Exercice 1 : Fonction sur les piles

a) Structure d'une pile

Algorithme :

```
//structure d'un élément de la pile :
Début
        struct cell *next ;
        entier value ;
Fin
CELL ;

//structure de la pile :
Début
        CELL *first ;
Fin
PILE ;
```

b) Fonction créer

Algorithme :

```
Fonction create() : Pile*
Début
        PILE *pile = allouer(*pile) ;
        pile → first = NULL ;
        Retourner(pile) ;
Fin
```

c) Fonction empiler

Algorithme :

```
Fonction push(Pile* pile, entier val) : vide
Début
  CELL *cell = allouer(tailleDe(*cell)) ;
  Si (pile != NULL && cell != NULL) Alors :
    Cell→value <= val ;
    Cell→next <= pile→first ;
    Pile→first <= cell ;
  Fin si
Fin
```

d) Fonction dépiler

Algorithme :

```
Fonction pop(Pile* pile) : vide
Début
  Si (pile != NULL) Alors :
    CELL *cellToUnstack = pile→first ;
    Si (cellToUnstack != NULL) Alors :
```

```
    pile→first  <= cellToUnstack→next ;
    deallouer(cellToUnstack) ;
   Fin si
  Fin si
Fin
```

e) Fonction vider

```
Algorithme :

Fonction clear(Pile *pile) : vide
Début
  Tant Que (pile→first != NULL) Faire :
    pop(pile) ;
  FinTantQue
Fin
```

f) Fonction sommet

```
Algorithme :

Fonction top(Pile *pile) : entier
Début
  Si (pile→ first != NULL) Alors :
    Retourner(pile→ first→value) ;
  Sinon :
    Retourner(-1) ;
  Fin si
Fin
```

g) Fonction pileVide

```
Algorithme :

Fonction isEmpty(Pile *pile) : entier
Début
  Retourner (pile→first = NULL ? 1 : 0) ;
Fin
```

h) Fonction taille

```
Algorithme :

Fonction size(Pile* pile) : vide
Entier :  cpt;
Début
  cpt = 0;
  Si (pile != NULL) Alors :
    Cell *cur = pile→first ;
    Si (cur != NULL) Alors :
      Tant que (cut != NULL) Faire:
        cpt <= cpt + 1;
        cur <= cur→next;
      Fin tant que
    Fin si
```

```
  Fin si
Fin
```

i) Fonction contient

```
Fonction contain(Pile* pile, entier : value) : entier
Début
  Si (pile != NULL) Alors :
      Cell *cur = pile→first ;
      Si (cur != NULL) Alors :
        Tant que (cur != NULL) Faire:
         Si(cur→value = value) Alors:
           Retourner(1);
         Fin si
        cur <= cur→next;
       Fin tant que
        Retourner(0);
      Fin si
  Fin si
Fin
```

j) Fonction égale

Algorithme :

```
Fonction equals(Pile* p1, Pile* p2) : entier
Début
  Si (p1 != NULL ET p2 != NULL) Alors :
    Si (size(p1) = size(p2) Alors:
      Cell p1_val <= p1→first;
      Si (p1_val != NULL) Alors:
        Tant que (p1_val != NULL) Faire:
          Si (!contain(p2, p1_val->value) Alors:
            Retourner(0);
        Fin tant que
        Retourner(1);
      Fin si
    Fin si
  Fin si
Fin
```

k) Fonction recherche et restaure

Algorithme :

```
Fonction searchAndRestore(Pile* p, entier : val) : vide
Début
  Si (p != NULL) Alors :
    Pile *finalPile = create() ;
    Cell *val = pile->first ;
    Si (finalPile != NULL ET val != NULL) Alors :
      Tant que (val != NULL) Faire :
        Si (val->value != val) Alors :
          push(finalPile, val->value) ;
        Fin si
```

```
      val = val->next ;
      Fin tant que
    Fin si
  Fin si
  *pile = *finalPile ;
Fin
```

I) Fonction trier

Exercice 2 :

Soit la formule d'Ackermann :

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m,n-1)) & \text{si } m > 0 \text{ et } n > 0. \end{cases}$$

1) Calculer A(2,2).

A(2,2) = A(1, A(2,1))

= A(1, A(1, A(2,0)))

= A(1, A(1, A(1,1)))

= A(1, A(1, A(0, A(1, 0))))

= A(1, A(1, A(0, A(0, 1))))

= A(1, A(1, A(0, 2)))

= A(1, A(1, 3)) //A(0,2)

= A(1, A(0, A(1,2)))

= A(1, A(0, A(0, A(1,1))))

= A(1, A(0, A(0, A(0, A(1,0)))))

= A(1, A(0, A(0, A(0, A(0,1)))))

= A(1, A(0, A(0, A(0, 2))))

= A(1, A(0, A(0, 3)))

= A(1, A(0, 4))

= A(1, 5)

= A(0, A(1,4))

= A(0, A(0, A(1,3)))

= A(0, A(0, A(0,A(1,2))))

= A(0, A(0, A(0,A(0, A(1,1)))))

= A(0, A(0, A(0,A(0, A(0, A(1,0))))))

= A(0, A(0, A(0,A(0, A(0, A(0,1))))))

= A(0, A(0, A(0,A(0, A(0, 2)))))

= A(0, A(0, A(0,A(0, 3))))

= A(0, A(0, A(0,4)))

= A(0, A(0, 5))

= A(0, 6)

= 7

2) Écrire l'algo itératif d'Ackermann.

Algorithme Ackerman itératif avec pile :

```
Fonction ack(n,m :entier) : entier

pile p;
entier n,m;

Début

   créer_pile (p);
   empiler(p,m);
   empiler(p,n);

   Tant que (non_vide(p)) faire
      n <- sommet(p);
      depiler(p);
         Si(non_vide(p)) alors
           m <- sommet(p);
           dépiler(p);
         Sinon
           retourner(n);
         Fin Si
      Si (m==0) alors
        empiler(p, n+1);
      Sinon si(n==0) alors
        empiler(p, m-1);
        empiler(1);
      Sinon
        empiler(p, m-1);
        empiler(p, m);
        empiler(p, n-1);
      Fin Si
      Fin Si
   Fin Tant que
   retourner(sommet(p));
Fin
```

3) Exécuter Ackermann pour m=1, n=2.

Pile	N	M	Pile	Test
1 2	1	2	1 2	Oui
	1	2	Rien	
X 1 X 2 0	1	2	0	
X 0 X 1 2 0	0	1	2 0	
X 1 X 0 2 0	1	0	2 2 0	
X 2 X 2 0	2	2	1 2 1 0	
X 1 X 2 1 0	1	2	0 2 1 1 0	
X 0 X 2 1 1 0	0	2	1 1 1 1 0	
X 1 X 1 1 1 0	1	1	0 1 0 1 1 0	
X 0 X 1 0 1 1 0	0	1	1 0 0 1 1 0	
X 1 X 0 0 1 1 0	1	0	2 0 1 1 0	
X 2 X 0 1 1 0	2	0	3 1 1 0	
X 3	3	1	2	

X 1 1 0			1 0 1 0	
X 2 X 1 0 1 0	2	1	1 1 0 0 1 0	
X 1 X 1 0 0 1 0	1	1	0 1 0 0 0 1 0	
X 0 X 1 0 0 0 1 0	0	1	1	

4) Calculer la complexité de l'algo.