Algorithmes et structures de données : TD 9 Corrigé

Piles - Complexité asymptotique

Rappel :
`SetLength(tableau, n)` est de complexité O(n)
`SetLength(tableau, 1)` est de complexité O(1)
`New(element)` est de complexité O(1) quand `element` est d'un type de taille fixe

Dans ce TD, on va implémenter une nouvelle structure de donnée, **la pile**, en utilisant d'abord un tableau dynamique, et après une liste chaînée. On comparera les complexités asymptotiques respectives.

Exercice 9.1 *Tableaux dynamiques*

Dans cet exercice, on écrira les fonctions et procédures nécessaires pour implémenter une pile avec un tableau dynamique. Pour cela, considérer le programme incomplet suivant :

```
var pile        : array of string;
var no_elements : integer;

procedutre creer_pile;
{ créer une pile vide }
begin
  no_elements := 0;
  SetLength(pile, no_elements);
end;

function est_vide() : boolean;
{ tester si la pile est vide }
begin
  if no_elements = 0 then
    result := true
  else
    result := false;
end;

procedure push(nom : string);
{ empiler }
var
```

```
begin
  no_elements := no_elements+1;
  SetLength(pile, no_elements);
  pile[no_elements-1] := nom;
end;

function pop() : string;
{ depiler et retourner l'élément en haut de la pile}
var
   element_ancien : string;
begin
  result := pile[no_elements-1];
  no_elements := no_elements-1;
  SetLength(pile, no_elements);
end;

function top() : string;
{ retourner l'élément en haut de la pile }
begin
  result := pile[no_elements-1];
end;

procedure afficher();
{ afficher les elements de la pile }
var i : integer;
begin
  for i:=0 to no_elements-1 do
    WriteLn(pile[i]);
end;

begin

  creer_pile;

  push('Axel');
  afficher;           {1}
  push('Jerome');
  push('Nathalie');
  afficher;           {2}
  nom := Pop;
  WriteLn(nom);
  afficher;           {3}

  Readln; {sert uniquement pour ne pas fermer la fenetre}
end.
```

1. Ecrire les fonctions et procédures qui sont "à faire". **Remarque :** Ne changez pas les

en-tetes/definitions des fonctions/paramètres !

2. Déterminer la complexité de chaque fonction.

fonction/procédure	complexité asymptotique
est-vide	O(1)
push	O(n)
pop	O(n) ou O(1), suivant le langage (et le ramasse-miettes)
top	O(1)
afficher	O(n)
vider	O(n) ou O(1), suivant le langage (et le ramasse-miettes)

3. Qu'est-ce qui est affiché à l'écran lors du premier appel de `afficher` ?

```
Axel
```

4. Qu'est-ce qui est affiché à l'écran lors du deuxième appel de `afficher` ?

```
Axel
Jerome
Nathalie
```

5. Qu'est-ce qui est affiché à l'écran lors du troisième appel de `afficher` ?

```
Axel
Jerome
```

6. Ecrire une procédure `procedure vider()` qui vide la pile entièrement.

```
procedure vider();
{ vide la pile }
begin
  SetLength(pile, 0);
end;
```

Exercice 9.2 *Pile avec liste linéaire simplement chaînée*

Dans cet exercice, on écrira les fonctions et procédures nécessaires pour implémenter une pile.

Pour cela, considérer le programme incomplet suivant :

```
type
   p_t_pile  =  ^t_pile;
   t_pile    = record
```

```
        nom        : string;
        suivant    : p_t_pile;
  end;

var pile : p_t_pile;

function est_vide() : boolean;
{ tester si la pile est vide }
begin

   if (pile = NIL) then
        result := true
   else
        result := false;

end;

procedure push(nom : string);
{ empiler }
var
   element_nouveau : p_t_pile;
begin

   New(element_nouveau);
   element_nouveau^.nom := nom;
   element_nouveau^.suivant := pile;
   pile := element_nouveau;

end;

function pop() : string;
{ depiler et retourner l'élément en haut de la pile}
var
   element_ancien : p_t_pile;
begin

   if NOT(est_vide=true) then
   begin
        element_ancien := pile;
        result := pile^.nom;
        pile := element_ancien^.suivant;
        Dispose(element_ancien);
   end
   else
        result := 'La pile est vide';

end;
```

```
function top() : string;
{ retourner l'élément en haut de la pile }
begin

   if NOT(est_vide=true) then
   begin
      result := pile^.nom;
   end
   else
        result := 'La pile est vide';

end;

procedure afficher();
{ afficher les elements de la pile }
var temp : p_t_pile;
begin

   temp := pile;
   while NOT(temp = NIL) do
   begin
      WriteLn(temp^.nom);
      temp := temp^.suivant;
   end;
   WriteLn;

end;

begin

  push('Axel');
  afficher;           {1}
  push('Jerome');
  push('Nathalie');
  afficher;           {2}
  nom := Pop;
  WriteLn(nom);
  afficher;           {3}

  Readln; {sert uniquement pour ne pas fermer la fenetre}
end.
```

1. Ecrire les fonctions et procédures qui sont "à faire". **Remarque :** Ne changez pas les en-tetes/definitions des fonctions/paramètres !
2. Déterminer la complexité asymptotique de chaque fonction/procédure que vous avez écrits.

fonction/procédure	complexité asymptotique
est-vide	O(1)
push	O(1)
pop	O(1)
top	O(1)
afficher	O(n)
vider	O(n)

3. Qu'est-ce qui est affiché à l'écran lors du premier appel de `afficher` ?
Axel
4. Qu'est-ce qui est affiché à l'écran lors du deuxième appel de `afficher` ?
Nathalie
Jerome
Axel
5. Qu'est-ce qui est affiché à l'écran lors du troisième appel de `afficher` ?
Jerome
Axel
6. Ecrire une procédure `procedure vider()` qui vide la pile entièrement.

```
procedure vider();
begin
    WHILE NOT(est_vide=true) DO
    begin
        pop();
    end;
end;
```

Exercice 9.3 *Complexités*

1. Comparer les complexités asymptotiques dans le tableau suivant :

fonction/procédure	complexité asymptotique tableau dynamiques	liste chaînée
est_vide	O(1)	O(1)
push	O(n)	O(1)
pop	O(n) ou O(1), suivant le langage (et le ramasse-miettes)	O(1)
top	O(1)	O(1)
afficher	O(n)	O(n)
vider	O(n) ou O(1), suivant le langage (et le ramasse-miettes)	O(n)

2. Est-ce que c'est plus efficace d'implémenter la pile avec une liste chaînée ou avec un tableau dynamiques ? Expliquer.

En terme de complexité temporelle et mémoire, il est plus efficace d'implémenter la pile avec une liste chaînée, surtout parce que l'allocation de la mémoire se fait uniquement en rajoutant un élément pour cet élément même.