Piles et applications

Matériel fourni :
- les notes de cours sur les Piles.

1 L'unité `U_Element`

Dans cette partie vous allez réaliser une unité nommée `U_Element` qui sera utilisée dans l'unité `U_Pile` et plus tard dans d'autres unités que vous serez amené à écrire. Il vous faudra donc conserver cette unité qui vous servira encore dans des TP prochains.

Cette unité sert essentiellement à définir un type nommé `ELEMENT` et deux procédures, l'une destinée à la lecture (saisie) d'un `ELEMENT`, l'autre à l'écriture (affichage) d'un `ELEMENT`. Voici la partie interface de cette unité :

```
unit U_Element;

interface

type
   ELEMENT = < A PRECISER > ;

   // ecriture du paramètre e sur la sortie standard
   procedure ecrireElement(const e : ELEMENT);


   // saisie de la valeur paramètre e sur la sortie standard
   procedure lireElement(out e : ELEMENT);
```

Question 1. Compléter cette unité avec le type `CHAR`.

Question 2. Réalisez un programme nommé `element1.pas` qui, en utilsant les deux procédures de l'unité `U_Element`, se contente de lire un élément et de l'écrire.

Question 3. Dans l'unité `U_Element`, changez le type `CHAR` en `INTEGER`.
Devez-vous changer l'implémentation des deux procédures d'entrée/sortie ? Si oui, faîtes-le !

Question 4. Recompilez le programme `element1.pas` sans le modifier, puis vérifiez son bon fonctionnement.

2 L'unité `U_Pile`

Dans cette partie, vous allez réaliser une unité nommée `U_Pile` pour construire un type `PILE`.
Voici la partie interface de cette unité (qui est celle présentée en cours[1]).

```
unit U_Pile;

interface

uses U_Element;

const MAX = 100; // taille maximale de la pile
type
   PILE = record
      sommet : 0..MAX;
      contenu : array[1..MAX] of ELEMENT;
   end {record};


const
```

[1] à la différence près que le type se nomme `PILE` au lieu de `T_PILE`.

```
PILE_VIDE : PILE = (sommet : 0);


// estPileVide(P) ssi P est vide
function estPileVide(P : PILE) : BOOLEAN;

// estPilePleine(P) ssi P est pleine
function estPilePleine(P : PILE) : BOOLEAN;

// sommet(P) = élément situé au sommet de P
// CU : déclenche une exception si la pile est vide
function sommet(P : PILE) : ELEMENT;

// empile x au sommet de la pile P
// CU : déclenche une exception si la pile est pleine
procedure empiler(const x : ELEMENT;
                  var P : PILE);

// dépile x la pile P
// CU : déclenche une exception si la pile est vide
procedure depiler(var P : PILE);
```

Question 5. Complétez la partie implementation de cette unité.

Question 6. En supposant que le type `ELEMENT` est `CHAR`, écrivez un programme nommé `pile1.pas` qui

1. lit un à un une suite de caractères saisis au clavier, ces caractères étant empilés succesivement dans une pile initialement vide. La suite des caractères lus est terminée lorsque l'utilisateur entre le caractère `'.'` (ce dernier caractère ne pouvant donc pas faire partie de la suite lue, et n'étant donc pas empiler).
2. puis vide la pile en écrivant les caractères un à un.

Testez votre programme avec les lettres de votre nom.

Question 7. En supposant que le type `ELEMENT` est `INTEGER`, écrivez un programme nommé `pile2.pas` qui fait la même chose que le précédent. La suite de nombres lus est terminée lorsqu'on entre un nombre négatif.

3 Une calculette

Dans cette partie, vous allez utiliser la structure de Pile pour réaliser une calculette quatre opérations sur les nombres entiers positifs ou négatifs.

Question 8. Quel doit-être le type `ELEMENT` ? Assurez-vous que c'est bien le cas dans votre unité `U_Element`.

La calculette va évaluer des expressions arithmétiques postfixées (cf le cours). Dans une utilisation de cette calculette, l'utilisateur pourra à chaque étape entrer un nombre entier ou une opération. En réponse la calculette affiche la valeur du sommet de la pile qu'elle utilise. Les lignes qui suivent montrent une session d'utilisation de cette calculette (le symbole > indiquant une attente de la calculette, et le symbole = indiquant sa réponse)

```
> 12
= 12
> 3
= 3
> 5
= 5
> *
= 15
> +
= 27
```

Ci-dessous le code du programme principal qu'il vous est demandé de réaliser.

```
 var
   op : STRING;
   p : PILE;
begin
```

```
   p := PILE_VIDE;

   while true do begin
      lire(op);
      traiter(op,p);
      afficherReponse(p);
   end {while};
end.
```

Comme vous pouvez le remarquer, la boucle tant que est infinie, autrement dit la calculette ne s'arrête jamais (la seule façon de l'interrompre est la combinaison de touche CTRL+C).

Question 9. Réalisez les trois procédures `lire`, `traiter` et `afficherReponse`.

La procédure `lire` lit une chaîne de caractères au clavier.

La procédure `traiter` va distinguer le cas où la saisie a donné un nombre entier ou le symbole d'un opérateur. Pour cette distinction, la procédure prédéfinie **`val`** est utile. En voici la spécification :

```
// Val(s,n,r) attribue à n l'entier représenté par
// la chaîne de caractères s.
// Si la conversion échoue, r est égal à l'indice
// où la conversion a échoué
procedure Val(const s : STRING; out n : LONGINT; out r : LONGINT);
```

La procédure `afficherReponse` ... affiche ... la ... euh ... réponse.

Question 10. Avec votre programme calculez les expressions

1. $$3 \times (4 - 7)$$
2. $$1 + \frac{8}{5} - 2$$
3. $$\frac{1+8}{5-2}$$

Question 11. Ajoutez une opération supplémenataire à votre calculette : la puissance.