Gestion dynamique de la mémoire

Matériel fourni :
- les notes de cours sur la gestion dynamique de la mémoire.

1 Programmes avec pointeurs

1.1 Adresses de variables

Question 1. Recopiez le programme ci-dessous, complétez-le en incluant la clause

```
uses SysUtils;
```

ainsi que la déclaration nécessaire de n.

```
begin
   n := strToInt(paramstr(1));
   writeln('Adresse de n : ',cardinal(@n));
end.
```

Q 1–1. Que signifie l'affichage obtenu ?
Q 1–2. Remplacez `cardinal(@n)` par `@n`. Que se passe-t-il à la compilation ? Pourquoi ?
Q 1–3. Ajoutez à la fin du programme l'instruction

```
writeln('Adresse de n : ',intToHex(cardinal(@n),8));
```

Quel est l'affichage produit ? Comparez-le au précédent ! Quel lien y a-t-il entre les deux ?

1.2 Pointeur vers une variable

Question 2. Recopiez le programme ci-dessous, complétez-le en incluant la clause

```
uses SysUtils;
```

ainsi que les déclarations nécessaires de variables. Quel est le type de n ? et celui de p ?

```
begin
   n := strToInt(paramstr(1));
   p := @n;
   writeln('Au debut : ');
   writeln('n = ',n);
   writeln('p^ = ',p^);
   inc(p^);
   writeln('Apres incrementation de la zone pointee par p :');
   writeln('n = ',n);
   writeln('p^ = ',p^);
end.
```

Exécutez ce programme et observez les affichages qu'il produit.

1.3 Arithmétique sur les pointeurs

Question 3. Recopiez le programme ci-dessous, complétez-le en incluant la clause

```
uses SysUtils;
```

ainsi que les déclarations nécessaires de variables.

```
begin
   n := strToInt(paramstr(1));
   writeln('Taille d''un CARDINAL : ',sizeof(n));
   p := @n;

   writeln('Avant incrementation de p : ');
   writeln('Valeur de p : ',cardinal(p));
   writeln('Adresse de n : ',cardinal(@n));
```

```
   inc(p);
   writeln('Apres incrementation de p : ');
   writeln('Valeur de p : ',cardinal(p));
   writeln('Adresse de n : ',cardinal(@n));

end.
```

Exécutez ce programme et observez l'affichage qu'il produit.

Question 4. Modifiez le programme pour

1. le type REAL (utilisez alors **strToFloat** au lieu de **strToInt**) ;
2. et le type CHAR.
3. mettez vos observations en rapport avec le résultat de la fonction **sizeof** sur les types CARDINAL, REAL et CHAR.

1.4 Allocation dynamique de la mémoire

Question 5. Recopiez le programme ci-dessous, complétez-le en incluant la clause

```
uses SysUtils;
```

ainsi que les déclarations nécessaires de variables.

```
begin
   new(p);
   p^ := strToInt(paramstr(1));

   writeln('Valeur pointee par p : ',p^);
   dispose(p);
end.
```

Q 5–1. Exécutez ce programme et onservez l'affichage qu'il produit.

Q 5–2. Mettez en commentaire l'instruction **new(p)**. Le programme compile-t-il ? Que se passe-t-il à l'exécution ? Expliquez !

Q 5–3. Ajoutez après l'instruction **dispose(p)** l'instruction

```
   writeln(p^);
```

Qu'obtient-on à l'exécution ?

1.5 Une fonction avec des pointeurs

Voici un programme nommé `factorielle1.pas` qui déclare une focntion `factorielle`, et qui calcule et affiche la factorielle du nombre passé sur la ligne de commande.

```
// auteur : EW
// date   : octobre 2008
// objet  : programme factorielle
//   a decliner ensuite en versions pointeurs
program factorielle1;
uses SysUtils;

function factorielle(n : CARDINAL) : CARDINAL;
var
   i : CARDINAL;
   f : CARDINAL;
begin
   f := 1;
   for i := 1 to n do
      f := f*i;
   factorielle := f;
end {factorielle};

begin
   writeln(factorielle(strToInt(paramstr(1))));
end.
```

Question 6. Réalisez un programme nommé `factorielle2.pas` dans lequel la variable locale `f` est remplacée par un pointeur vers un `CARDINAL`.
Question 7. Réalisez un programme nommé `factorielle3.pas` dans lequel la variable locale `i` est aussi remplacée par un pointeur vers un `CARDINAL`.
Question 8. Réalisez un programme nommé `factorielle4.pas` dans lequel en plus des variables locales, le paramètre `n` de la fonction est de type pointeur vers un `CARDINAL`, ainsi que la valeur retournée. Il sera nécessaire de déclarer préalablement le type

```
type
   POINTEUR = ^CARDINAL;
```

1.6 Tableaux et pointeurs

Question 9. Écrivez un programme qui initialise un tableau déclaré par

```
var
  t : array[1..10] of INTEGER;
```

avec les valeurs $\texttt{t}[i] = 2 * i + 1$ et qui produit l'affichage des adresses des et des valeurs des différents éléments du tableau, à l'aide de l'opérateur `@`.

```
t[1] a pour adresse : 6446720 et contient 3
t[2] a pour adresse : 6446724 et contient 5
t[3] a pour adresse : 6446728 et contient 7
t[4] a pour adresse : 6446732 et contient 9
t[5] a pour adresse : 6446736 et contient 11
t[6] a pour adresse : 6446740 et contient 13
t[7] a pour adresse : 6446744 et contient 15
t[8] a pour adresse : 6446748 et contient 17
t[9] a pour adresse : 6446752 et contient 19
t[10] a pour adresse : 6446756 et contient 21
```

(il est possible que les adresses affichées soient différentes pour vous).
Question 10. Quelle conclusion pouvez-vous formuler sur la représentation en mémoire du tableau `t` ?
Question 11. Modifiez votre programme pour que l'affichage soit obtenu grâce à un sul pointeur initialisé à `p := @t[1]`.

2 Simulation de pointeurs

On souhaite simuler la gestion dynamique de la mémoire, c'est-à-dire créer les équivalents des opérations **new**, **dispose**, ^, ...

Pour cela, on va considérer que
- la mémoire (le tas) est un tableau d'entiers
- les adresses sont les indices du tableau
- la table d'allocation est un tableau de booléens indexé par les adresses et telle que `table[i]` $\iff$ `tas[i]` non alloué (libre).

Question 12. Écrire une procédure `initTable` qui initialise la table et donc rend libre le tas entier.

```
// initTable : rend libre le tas entier
procedure initTable ;
```

Question 13. Écrire une fonction qui teste si une adresse est libre.

```
// libre(i)  <=>  tas[i] non alloué
function libre(i: ADRESSE): Boolean ;
```

On définit un pointeur comme étant une ADRESSE ou NUL (car NIL est un mot réservé du Pascal).
Question 14. Écrire une procédure `allouer(p)` qui réserve une zone libre du tas et attribue l'adresse de cette zone à `p`.

```
// CU : le tas est supposé ne pas être complètement alloué
procedure allouer(out p: POINTEUR) ;
```

Question 15. Écrire une procédure `desallouer(p)` qui libère la zone d'adresse `p` et attribue la valeur NUL à `p`.

```
procedure desallouer(var p : POINTEUR);
```

Question 16. Écrire une fonction `valeurPointee(p)` qui donne accès à la valeur de la zone pointée par `p`.

```
function valeurPointee(p : POINTEUR): CARDINAL;
```

Question 17. Écrire une procédure `attribuer(n,p)` qui attribue la valeur `n` à la zone pointée par `p`.

```
// CU p doit pointer vers une zone allouée
procedure attribuer(const n : CARDINAL; const p : POINTEUR);
```

Question 18. On veut maintenant réaliser une unité `U_Pointeur`. Que doit-on mettre
- dans la partie *interface*
- dans la partie *implementation*
- dans la partie *initialization* et
- dans la partie *finalization*

2.1 Applications :

Question 19. Écrire un programme qui calcule le carré d'un entier stocké dans le tas.
Question 20. Écrire un programme qui échange les valeurs de deux zones pointées.
Question 21. Écrire une fonction qui calcule la somme des entiers compris entre 1 et un entier $n \geqslant 0$ stocké dans le tas.
Question 22. Écrire une fonction qui calcule la factorielle d'un entier $n \geqslant 0$ stocké dans le tas.