Notions avancées

1. La récursivité

Qu'est-ce que la récursivité ?

- ✓ Une fonction (procédure) est dite *récursive* si elle s'appelle elle-même;
- ✓ L'appel d'une fonction (procédure) à l'intérieur d'elle-même est dit *appel récursif*;
- ✓ Un appel récursif doit être dans une instruction conditionnelle (sinon la récursivité est sans fin);
- ✓ Il y a une *condition d'arrêt* (test de fin): dans ce cas, il n'y a pas d'appel récursif;
- ✓ On doit toujours tester en premier la condition d'arrêt, et ensuite, si la condition n'est pas vérifiée, on lance un appel récursif.

Résolution récursive d'un problème

Pour créer une fonction récursive, il faut :

- ✓ Décomposer un problème en un ou plusieurs sous-problèmes du même type;
- ✓ On résout les sous-problèmes par des appels récursifs;
- ✓ Les sous-problèmes doivent être de taille plus petite que le problème initial.
- ✓ La décomposition doit conduire à un cas élémentaire, qui, lui, n'est pas décomposé en sous-problème (condition d'arrêt).

Exemple

Problème : calculer $n!$ (n! = n * n-1 * n-2 * n-3 * … * 1)

Sous-problème : calculer $(n-1)!$ ((n-1)! = n-1 * n-2 * n-3 * … * 1)

Sous-problème : calculer $(n-2)!$ ((n-2)! = n-2 * n-3 * n-4 * … * 1)

.
.
.

Sous-problème : calculer 0! . C'est un problème élémentaire : le résultat vaut 1.

Structure d'une fonction récursive

```
fonction <nom> (paramètres): <type_résultat>
<partie déclarations>
début
  si (condition d'arrêt) alors
    <nom> ← <résultat>        (cas élémentaire)
  sinon
    …
    Appel récursif
    …
  FinSi
finFonction
```

***Exemple*:** fonction récursive qui calcul le factoriel d'un nombre entier.

```
fonction fact (n:entier): entier
début
  si (n=0) alors
    fact ← 1
  sinon
    fact ← n*fact(n-1)
  FinSi
finFonction
```

Exécution de cette fonction avec n = 3

Factorielle de 3	**n = 3** fact = 3*fact(2) fact = 3*2	 **n = 2** fact = 2*fact(1) fact = 2*1	 **n = 1** fact = 1*fact (0) fact = 1*1	 **n = 0** fact = 1

Exercice: La suite de *Fibonacci* est définie par récurrence par :

$u_0 = 0$; $u_1 = 1$

$u_n = u_{n-1} + u_{n-2}$ pour $n \geq 2$

- Donner un une fonction récursive pour calculer u_n.

Solution:

```
fonction Fibonacci (n:entier): entier
début
  si (n=0) alors
    Fibonacci ← 0
  sinon
    si (n=1) alors
      Fibonacci ← 1
    sinon
      Fibonacci ← Fibonacci(n-1)+ Fibonacci(n-2)
    FinSi
  FinSi
finFonction
```

2. Les pointeurs

- ✓ Les variables manipulées par un programme sont stockées dans la mémoire centrale (RAM);
- ✓ La mémoire peut être assimilée à un tableau dont chaque élément possède un numéro dit *adresse*;
- ✓ Chaque élément de ce tableau est une "case" mémoire; c'est à dire une zone où l'on peut stocker la valeur d'une variable atomique (par exemple un entier);
- ✓ Pour retrouver une variable, il suffit de connaitre l'adresse mémoire de l'élément où elle est stockée.

Le système d'exploitation alloue à chaque programme une *mémoire statique* et une *mémoire dynamique*.

- ***Mémoire statique*:** permet de stocker des *variables statiques*; c'est-à-dire des variables:
  - déclarées avant l'exécution d'un programme;
  - la place qui leur est réservée en mémoire est figée durant toute l'exécution du programme.
- ***Mémoire dynamique*:** permet de stocker des *variables dynamiques*; c'est-à-dire des variables:
  - qui prennent place en mémoire au fur et à mesure des besoins;

– libèrent de la place mémoire lorsqu'on n'en a plus besoin.

Un pointeur *est une variable qui stocke* ***l'adresse*** *(c'est-à-dire la position en mémoire) d'une variable dynamique.* On dit qu'un pointeur pointe vers une autre variable dynamique dite ***variable pointée***.

Déclaration des pointeurs

Méthode 1:

```
type <nom_type> = ↑<type_varible_pointée>
var <nom_pointeur> : <nom_type>
```

Avec:

- ✓ `<nom_type>` est le nom du type pointeur;
- ✓ `<type_varible_pointée>` est le type de la variable pointée;
- ✓ `<nom_pointeur>` est le nom de la variable pointeur.

Exemple:

```
type PTR = ↑entier
var P : PTR
```

- La variable `P` est de type `PTR` qui est en fait un pointeur pointant vers un entier.

Méthode 2:

```
var <nom_pointeur > : ↑<type_varible_pointée>
```

Exemple:

```
var P : ↑entier
```

- La variable `P` un pointeur pointant vers un entier.

Remarque:

- **`P`** est la variable pointeur (contenant l'adresse mémoire d'une variable). Elle sera stockée dans la mémoire statique;
- **`P`↑** est la variable pointée (contient la valeur de la variable pointée). Elle sera stockée dans la mémoire dynamique.

Création d'une variable dynamique

Soit la variable `P` déclarée comme suit:

```
var P : ↑entier
```

L'instruction:

`Allouer(P)` (en *Pascal* : `new(P);`) permet de:

- ✓ réserver un emplacement mémoire d'une taille correspondante au type `entier`,
- ✓ mettre dans la variable `P` l'*adresse* de la zone mémoire qui a été réservée.

Libération de la place occupée par une variable dynamique

L'instruction:

`Désallouer(P)` (en *Pascal* : `dispose(P);`) permet de:

- ✓ libérer la place de la zone mémoire dont l'adresse est dans P.

Remarque:

La valeur `NIL` est une constante qui désigne une adresse nulle.