Langage C_ Présentation

I/ Un peu d'histoire

Le **langage C** possède trois ancêtres : le *CPL*, le *BCPL* et surtout le *langage B*.

- Le langage **CPL** (***C**ombined **P**rogramming **L**anguage*) a été conçu au début des années 1960 par les universités de Cambridge et de Londres. C'était un grand projet académique consistant à créer un langage englobant beaucoup de concepts. Le **CPL** devait notamment être **fortement typé** (*), avec de nombreux types comme des nombres *entiers*, *réels*, *booléens*, *caractères*, *tableaux*, *listes* ...

Le CPL était trop complexe pour l'époque et il semble que personne n'ait réussi à terminer l'écriture d'un **compilateur** (**). Ce langage a disparu sans laisser de trace quelque part dans les années 1970.

(*) Langage typé : il s'agit d'un langage informatique dans lequel le **programmeur** indique **explicitement la nature de la variable à sa déclaration** (nombre entier, à virgule flottante, caractère, tableau …), il s'agit d'un *typage* dit *« fort »* ou *typage* dit *« statique »*.
Passer d'un type à un autre nécessite alors *en principe* une **conversion** (*cast* en anglais) **explicite** de la part du programmeur.

Il existe un risque toutefois d'une *conversion implicite* (notamment en langage C) dont il faudra tenir compte mais les **compilateurs récents lancent des avertissements**.
Exemple ici :

```
int x = 3, y = 2; // déclaration de deux variables entières x et y et initialisation float z; // déclaration d'une variable flottante z z = x/y;
```

```
int x = 3, y = 2;
// déclaration de deux variables entières x et y et initialisation
float z; //
déclaration d'une variable flottante z
z = x/y;
```

Le résultat sera z = 1 et non z = 1.5 comme attendu car la variable `z` sera convertie en nombre entier.

A l'inverse, des *langages faiblement typés* ou dits *typés « dynamiquement »* comme **Python** dédient à l'ordinateur de déterminer le type adéquat de la variable en fonction du contexte pendant l'exécution du programme.

Quelques langages fortement typés : **Java**, **OCaml** (à la compilation), **C++**, **C#**.
Le langage **C** est lui moyennement typé à cause des *conversions implicites* et du passage au type *void** que l'on verra ultérieurement.

Faut-il utiliser ***langage typé statique*** ou ***dynamique*** ? Laisser au programmeur ou à l'ordinateur de gérer le type des variables ? Débat en cours 😊.

(**) Compilateur : il sert à traduire un *code source* écrit dans un langage de programmation en un autre langage, habituellement un langage d'assemblage ou un langage machine. Le programme en langage machine produit par un compilateur est appelé code objet. Il assure en outre l'édition de liens vers d'autres fichiers (fichiers *.dll* qui sont des bibliothèques dynamiques par exemple).

Le premier compilateur, A-0 System, a été écrit en 1951 par Grace Hopper.

En savoir plus sur Grace Hopper, grande figure de l'informatique, lien ici : https://www.techno-science.net/glossaire-definition/Grace-Hopper.html
On lui doit le langage COBOL en 1959, les normes du langage FORTRAN et plus généralement l'utilisation d'un langage proche de l'anglais plutôt que du langage machine (assembleur).

- Le **langage BCPL** (pour *Basic CPL*) a été conçu à Cambridge en 1966 par **Martin Richards**. L'année suivante, il alla au **MIT** et écrivit un premier compilateur. Le BCPL est une version fortement simplifiée du langage CPL avec notamment un seul type de donnée, le mot machine, c'est-à-dire le nombre typique de bits que le processeur d'un ordinateur traite en une instruction-machine (addition, soustraction, multiplication, copie...). Du temps du BCPL, on trouvait des architectures à mots de 40 bits, 36 bits, 18 bits, 16 bits...
  Le BCPL opère sur les mots de la machine. Il est donc à la fois portable et proche du matériel, donc efficace pour la programmation système. Le langage BCPL a servi à écrire divers systèmes d'exploitation de l'époque, dont un (TripOS) qui s'est retrouvé partiellement porté sur l'Amiga pour devenir la partie Amiga DOS du système.
  Mais aujourd'hui, le BCPL ne semble plus être utilisé que par son inventeur. C'est sans doute dû au fait que le langage C a repris et étendu la plupart des qualités de BCPL.

- Le **langage B** a été créé par **Ken Thompson** vers 1970 dans les **laboratoires Bell** d'**AT&T**. L'année précédente, il avait écrit en assembleur la première version d'**UNIX** sur un **PDP-7** contenant 8 kilo mots de 18 bits. Lorsqu'il voulut proposer un langage sur ce nouveau système d'exploitation, il semble qu'il ait d'abord pensé au **Fortran**, mais que très vite (en une semaine) il conçut son propre langage : B.
  Le B est une simplification de BCPL, mais la syntaxe très sobre du B (et des commandes **UNIX**) toute en lettres minuscules correspond surtout aux goûts de **Thompson**. Le langage C a repris la syntaxe du B avec un minimum de changements.
  Le langage B a été porté et utilisé sur quelques autres systèmes. Mais cette même année 1970, le succès du projet **UNIX** justifia l'achat d'un **PDP-11**. Celui-ci avait des mots machine de 16 bits, mais il était aussi capable de traiter des octets (24 ko de mémoire vive en tout) dans chacun desquels pouvait être stocké un caractère. Le B ne traitait que des mots machine, donc le passage de 18 à 16 bits n'était pas problématique, mais il était impossible de traiter efficacement les caractères de 8 bits. Pour bien exploiter les capacités de la machine, le B a donc commencé à être étendu en ajoutant un type pour les caractères...

  Un code en langage B :

```
infact (n)
{
  auto f, i, j;    /* no initialization for auto variables */
  extrn fact;      /* "What would I do differently if designing
                    *  UNIX today?  I'd spell creat() with an e."
                    *  -- Ken Thompson, approx. wording */

  f = 1.;          /* floating point constant */
  j = 0.;
  for (i = 0; i <= n; ++i) {
    fact[i] = f =#* j;       /* note spelling =#* not #*= */
    j =#+ 1.;                /* #+ for floating add */
  }

  return (f);      /* at least, I think the () were required */
}

TOPFACT = 10;    /* equivalent of #define, allows numeric values only */
fact[TOPFACT];
```

- Le **langage C** a été développé par un collègue de **Ken Thompson**, **Dennis Ritchie** qui pensait d'abord faire uniquement un New B ou NB. Mais en plus des caractères, Ritchie ajouta les tableaux, les pointeurs, les

nombres à virgule flottante, les structures ... 1972 fut l'année de développement la plus productive et sans doute l'année de baptême de C.
En 1973, le langage C fut suffisamment au point pour que 90 % d'**UNIX** puisse être réécrit avec. En 1974, les laboratoires Bell ont accordé des licences **UNIX** aux universités et c'est ainsi que le langage C a commencé à être distribué.

Un code en langage C (le code du langage B précédent) :

Déclaration d'une fonction (type du paramètre et de la valeur de retour).

Les **accolades** déterminent le bloc « *fonction* » et les « **;** » une instruction.

A noter la boucle « *for* » en langage C.

L'instruction « *define* » permet ici d'associer la valeur 10 à TOPFACT : le compilateur remplacera TOPFACT par 10.

```
float infact (n) int n;
{
  float f = 1;
  int i;
  extern float fact[];

  for (i = 0; i <= n; ++i)
    fact[i] = f *= i;

  return d;
}

#define TOPFACT 10
float fact[TOPFACT+1];
```

Le C est un langage que l'on peut qualifier de « **bas niveau** ». Cette appellation n'est pas péjorative, elle signifie simplement que le C manipule les mêmes sortes d'objets que la plupart des ordinateurs à savoir des ***caractères***, des ***nombres*** et des ***adresses mémoire***. On peut combiner ces objets entre eux et les manipuler grâce aux opérateurs arithmétiques et logiques que les machines fournissent réellement.

le langage C ne comporte aucune opération qui traite directement des objets composés, tels que les chaînes de caractères les ensembles les listes ou les tableaux. Il n'existe pas non plus d'opérations qui manipule l'intégralité d'un tableau ou d'une chaîne bien qu'il soit possible de copier les structures dans leur ensemble.

Le langage ne définit pas d'autre possibilité d'allocation de mémoire que la définition statique et la structure de pile où sont rangées les variables locales des fonctions ; il n'y a pas de segment de mémoire ni de récupération de la mémoire inutilisée (ramasse-miettes). Enfin le C proprement dit ne fournit aucun moyen de réaliser des entrées-sorties, ne comporte pas d'instruction READ ou WRITE ni de méthode d'accès aux fichiers.

La version définitive du langage C a été approuvée définitivement fin 1988, c'est la norme ANSI.