SUPPORT DE COURS B1-DEV

Introduction à la programmation

Généralités

Date	Révision
16/09/2018	Publication initiale BTS SIO
11/08/2019	Corrections et améliorations (exemple + simplifications)
03/10/2019	Remise au standard DR2019
03/08/2021	Amélioration des titres et ajout d'explications sur compilation versus interprétation

TABLE DES MATIÈRES

Icon made by Freepik from www.flaticon.com

1 Les différents langages

Il existe plusieurs langages de programmation, certains sont propres à une technologie, d'autres n'existent plus tandis que de nouveaux langages naissent pour répondre à de nouveaux besoins.

Il est difficile de classer les langages entre eux, car ils sont très nombreux et ont tous des spécificités qui sont à l'origine de leur existence : il serait en effet idiot de créer plusieurs langages fonctionnant de manière identique.

Voici quelques exemples de classification.

1.1 Classement par Performances

On peut classer les langages par performance : les plus rapides étant les plus intéressants.

Dans ce cadre, on trouvera :

- Le langage machine (binaire)
- L'assembleur
- Les langages compilés
- Les langages interprétés

1.1.1 Le langage machine

Il consiste en une écriture binaire des instructions. La machine ne comprend en effet que ce langage. Humainement, c'est le langage le plus difficile à écrire. Il faut être capable de calculer en binaire et il est plus concis d'écrire en hexadécimal. Programmer dans ce langage est très fastidieux.

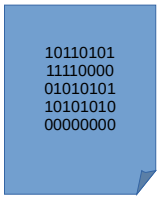
10100001 = 0xA1

00000000 = 0x00

11111111 = 0xFF

11110000 = 0xF0

etc.



Programme.exe

1.1.2 L'ASSEMBLEUR

Il s'agit du langage le plus proche du langage binaire de la machine. Le microprocesseur ne peut comprendre que des codes binaires mais les programmeurs préfèrent des mots.

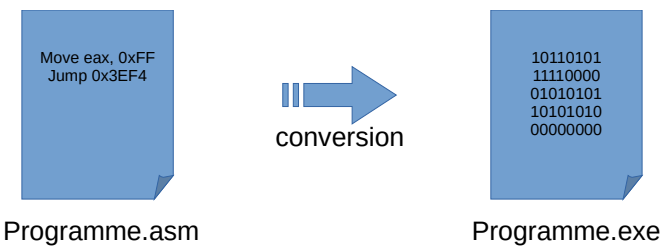
L'assembleur est un programme qui lit un code "humain" et le convertit en binaire. Il utilise donc un dictionnaire dans lequel chaque mot correspond à un code binaire compréhensible par le processeur. Le compilateur traduit les instructions en instructions binaires.

L'exemple ci-après montre une instruction en assembleur Intel et son code hexadécimal dans la mémoire.

`MOV EAX, 0x01020304` *(déplace EAX dans mémoire 01020304)*	A101020304

Si c'est le langage le plus rapide, il a comme inconvénient que chaque processeur utilise ses propres codes (EAX est un registre sur les processeurs Intel, qui n'existe pas sur les CPU ARM, Motorola…)

L'assembleur est donc un simple traducteur mot à mot.



1.1.3 LES LANGAGES COMPILÉS

Pour éviter l'inconvénient de devoir taper plusieurs codes pour faire le même programme sur plusieurs processeurs, sont apparus les langages compilés.

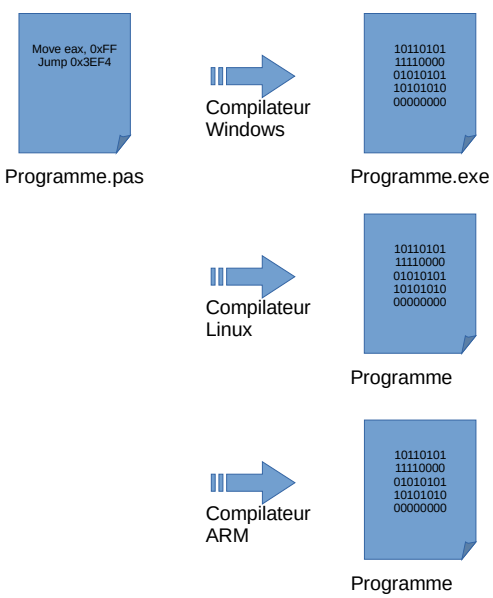
Le langage de développement utilise des instructions proches des mots humains (en anglais) : **for**, **if**, **else**, **print.** Il est aussi capable de créer des procédures, c'est-à-dire des morceaux de codes réutilisables, avec des mots clés comme **procedure** ou **function**.

Exemple de programme en Delphi (Pascal)

```
procedure TForm1.Draw();
var
  i, j, k: Integer;
begin
  for j:=0 to 4 do
    for i:=0 to 4 do
    begin
      k:=tab_int[i,j];
      img_screen.Canvas.Draw(i*100,j*100,tab_img[k].Picture.Graphic);
    end;
end;
```

Il faut ensuite utiliser un compilateur, qui va traduire chaque instruction en plusieurs instructions binaires (langage binaire). En effet, dans ces langages, les instructions sont parfois complexes et obligent le compilateur à écrire plusieurs actions en langage machine pour obtenir le même résultat.

Ce type de langages est beaucoup plus facile à lire pour un humain et se rapproche énormément de l'algorithme. De plus on peut avoir un seul fichier source et créer des exécutables pour chaque architecture en créant un compilateur pour celle-ci.



1.1.4 Les langages interprétés

Les langages compilés ont beaucoup d'avantages mais il faut créer autant de compilateur qu'il existe de processeur. Pour le développeur, c'est transparent (c'est la société qui vend les compilateurs qui fait le travail).

Il reste cependant un inconvénient : pour chaque architecture, il peut y avoir plusieurs configurations : matériels différents, définitions d'écran différentes, emplacements mémoires différents. Il reste donc difficile de créer un programme compatible avec plusieurs architectures.

Une solution se trouve dans les langages interprétés :

- ceux qui utilisent d'un runtime interpreter (un interpréteur d'exécution)
- ceux qui dépendent d'une machine virtuelle

1.1.4.1 Runtime

Les langages de cette catégorie sont PHP, Ruby, JavaScript, Python…

Chaque fois qu'un nouveau programme est lancé, il faut d'abord lancer son interpréteur, avec le nom du programme en paramètre.

Ex : py monProgramme.py

Le programme interpréteur se lance, puis interprète les instructions données dans le fichier joint. Comme le décodage se fait en temps réel, si une erreur est présente, le programme s'arrêtera de fonctionner à ce moment-là.

1.1.4.2 Machine virtuelle

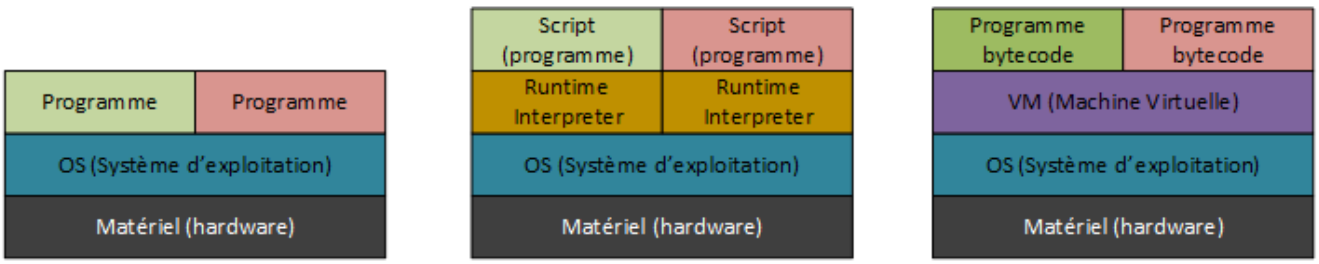
Pour devenir indépendant du matériel, certains éditeurs ont eu l'idée de créer les machines virtuelles : Java Runtime Environnement (Oracle) et framework.net (Microsoft) sont des exemples de machines virtuelles ! *C'est un peu la même chose qu'un émulateur de console qui fonctionne sur un PC.*

Le code écrit dans le langage correspondant (Java ou C#) est exécuté en temps réel sur la VM (Virtual Machine). Les différents codes sont donc interprétés par l'unique machine virtuelle.

Notez toutefois la différence suivante avec la catégorie précédente : un programme peut d'abord être pré-compilé, ce qui implique une vérification du code avant d'avoir un programme utilisable sur la machine virtuelle.

1.1.5 Différence entre programme compilé, interprété et en VM

Le schéma montre les couches nécessaires pour l'exécution d'un programme dans les 3 cas.



Les langages compilés sont donc les plus rapides et les plus performants, car plus proche du matériel.

1.2 Classement par Paradigmes

Nous avons vu que le langage binaire est le plus proche de la machine, avec le risque de faire facilement planter le programme.

À l'inverse, les langages comme C# ou Java s'exécutent sur une machine virtuelle qui peut bloquer certaines actions et éviter le plantage total.

Nous pouvons donc effectuer un classement par niveau de capacités, c'est-à-dire, les fonctionnalités des langages.

1.2.1 Langages linéaires

Ce sont les langages les plus bas et donc les plus proches du matériel. Le langage machine et le binaire font partie de cette catégorie : les instructions sont simples et s'exécutent les unes après les autres.

1.2.2 Langages structurés

Ce sont des langages qui permettent d'utiliser des conditions et des itérations (répétitions). En réalité, tous les langages sont structurés, pour permettre des choix, des décisions et des modifications.

1.2.3 Les langages procéduraux

Ce sont les langages compilés et interprétés dans leur globalité : C, C#, C++, Java, Perl, Javascript, PHP, etc.

Ils permettent de créer des blocs de codes qu'il est possible de réutiliser et d'exécuter plusieurs fois : ce sont les procédures ou fonctions. L'intérêt est de pouvoir transmettre des valeurs a ces fonctions et d'obtenir une valeur en retour.

1.2.4 Les langages déclaratifs

Ces langages obligent le développeur à préparer son code, et notamment les variables dont il aura besoin, à l'avance.

L'objectif, est de réserver la place en mémoire pour les valeurs que le programme pourra utiliser. En termes de sécurité, cela permet aussi d'alerter le développeur sur une faute de syntaxe ou une erreur de frappe. On les considère comme plus sûrs : Java, C#, C++ et C en font partie.

Javascript, PHP, Perl n'en font pas partie.

1.2.5 Les langages objets

Java, C# et C++ (puis PHP et JavaScript) ont adopté une manière de programmer sous forme de bloc de code contenant les valeurs et les fonctions. C'est un paradigme très sécurisé et facile à utiliser, mais qui nécessite un peu plus d'apprentissage au début.

2 Vision globale des langages

Les langages évoluent, apparaissent et disparaissent en fonction des besoins, des intérêts et du suivi des communautés.

Ainsi, régulièrement la question se pose de savoir quel langage apprendre.

2.1 Au collège et au lycée

Le langage de prédilection est généralement **Scratch**, sinon **Python** pour la calculatrice

2.2 En BTS SIO

Le BTS SIO du lycée Aristide Bergès propose 5 langages :

- **Python** (en algorithmie)
- **C#** et **JavaScript** en développement
- **PHP** en base de données
- **Java** pour Android

2.3 En université et ailleurs

En études post BTS, on trouve souvent les langages suivants :

- **JAVA**
- **C** et **C++**
- **ERLang**

2.4 Les autres langages

Les différents systèmes sont :

- Assembleur (langage machine)
- Pascal et Delphi
- WinDev
- Go
- TypeScript
- Rust
- et des outsiders comme Cobol, Fortran, Prolog, Perl, Visual Basic, etc.

3 CLASSEMENT DES LANGAGES

Quelques sites proposent le classement des langages, mais si les tendances se ressemblent, encore une fois, la diversité reste présente, en fonction des projets, des envies et des fonctionnalités...

3.1 TIOBE

Le site https://www.tiobe.com/tiobe-index/ propose un classement selon plusieurs critères, et fournit donc plusieurs résultats. On notera que SQL est considéré comme un langage de programmation.

Jul 2022	Jul 2021	Change	Programming Language	Ratings	Change
1	3	↑	Python	13.44%	+2.48%
2	1	↓	C	13.13%	+1.50%
3	2	↓	Java	11.59%	+0.40%
4	4		C++	10.00%	+1.98%
5	5		C#	5.65%	+0.82%
6	6		Visual Basic	4.97%	+0.47%
7	7		JavaScript	1.78%	-0.93%
8	9	↑	Assembly language	1.65%	-0.76%
9	10	↑	SQL	1.64%	+0.11%
10	16	⇈	Swift	1.27%	+0.20%
11	8	↓	PHP	1.20%	-1.38%

3.2 DEVELOPPEZ.COM

Un autre classement, c'est celui du site developpez.com qui en outre propose, de l'actualité, des tutoriels, etc.