Langage informatique

Définition

Il sert à décrire les actions que le programmeur veut faire faire à son ordinateur afin d'arriver à trouver une solution à un problème.
C'est un langage qui n'est pas forcement compris **directement** par le microprocesseur mais, dans le meilleur des cas, qui doit être comprit et maîtrisé par le programmeur.

Il a différent niveau de complexité et d'abstraction.

- Au plus bas niveau
  Il est directement compris par le microprocesseur : c'est le binaire exécutable. Si votre programme fait 1Mo de code il faut écrire 1Mo de binaire représentant les instructions du microprocesseur. Ce n'est plus guère possible actuellement.
- Au niveau intermédiaire
  Il décrit des ensembles de contraintes et d'actions de manière symbolique. Il faut le traduire pour que le code binaire soit généré. Mais on peut aussi décrire des actions au niveau du microprocesseur.
- Au plus haut niveau
  Il décrit des ensembles de contraintes et d'actions de manière symbolique. On a besoin de le traduire pour que le code binaire soit généré.

Historique

Il est très malcommode de coder du binaire directement. Outre que l'on doit connaître toutes les instructions binaires du micro, on doit aussi connaître parfaitement le matériel sur lequel on code. Ces contraintes empêchent le programmeur de se concentrer sur le programme.
Le besoin d'un langage s'est fait sentir dès que l'ordinateur a été inventé. Il semble permettre de ne se consacrer qu'à l'écriture de programme et faire abstraction de la machine qui l'exécute. Dans la réalité cela est faux mais c'est une vision commode pour travailler.
Le premier langage est un langage pour les mathématiciens le ForTran (Formula Translator). Ce qui est bien normal car le premier ordinateur a été inventé par des mathématiciens.

Nous allons essayer de décrire l'évolution des langages ainsi que celle des machines et des systèmes.

année	langage	machine	système	commentaire
783				Al-Khawarizmi mathématicien perse qui inventa l'algorithmique[1]
1840				1er programme par Ada Lovelace[2]
1943		ENIAC		1er ordinateur à tube
1951-2	A-0	UNIVAC		Chargeur de programme binaire
1954	ForTran[3]	Main Frame	propriétaire	1er "vrai" langage informatique
1958	Algol 58			Notion de séquence et de récursivité
1959	Lisp			Langage de recherche en IA[4]

[1] http://fr.wikipedia.org/wiki/Abou_Jafar_Muhammad_Ibn_M%C5%ABsa_al-Khuw%C4%81rizm%C4%AB
[2] http://fr.wikipedia.org/wiki/Ada_Lovelace
[3] Définition des langages voir : http://fr.wikipedia.org/wiki/<nom_du_langage> ex. http://fr.wikipedia.org/wiki/Fortran
[4] Intelligence Artificielle

(Suite)

année	langage	machine	système	commentaire
1960	Cobol	IBM	propriétaire	Langage de comptabilité et de gestion
1962	APL			Langage pour les mathématiques
1964	PL1			Langage système des ordinateurs IBM
1964	Basic			**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode. Simple d'utilisation
1967	BCPL	mini		Basic Combined Programming Language
1968	Forth			Programmation interactive utilisée en astronomie et par les machines SUN
1968	Logo			Langage d'IA d'objets réflexifs utilisé pour l'apprentissage avec une tortue qui se déplace
1969	C	mini	Unix	Langage impératif structuré de haut et bas niveau. Utilisé pour construire Unix et inspira de nombreux langages actuels
1971	Pascal			Langage structuré utilisé dans l'enseignement
1972	Smalltalk			1er langage objet
1972	prolog			IA
1974	SQL			Langage des bases de données
1975	Sheme			IA
1975	Altair Basic	µordinateur		1er Basic pour µ ordinateur Altair fait pas Bill Gates et Paul Allen
1979	awk			Traitement de ligne sous Unix
1981		PC	DOS	
1983	C++			Langage objet. Systèmes et Programmes
1983	ADA			Langage objet. Militaire et spatial
1985	PostScript			Langage de description de page (PDF)
1985	Caml			Langage objet pour étudiant français (INRIA)
1986	Eiffel			Langage objet
1987	Perl			Langage interprété pur la manipulation de fichiers de texte
1988	Tcl			Langage de script. Evolution en 90 avec TK (graphisme). On parle de Tk-Tcl
1988	Charme			Langage de contraintes
1991	Python			programmation structurée et objet
1991		PC	Linux	
1993	Ruby			Langage objet utilisé sous Unix
1995	Java			Langage objet portable sur différentes machines
1995	JavaScript			Script pour le WEB
1995	PHP			Langage de script pour produire des pages WEB
2000	C#	Microsoft		Langage objet pour Microsoft .NET

Conception d'un programme

Un programme est à la base un texte d'instructions écrit dans un langage donné. Pour écrire on doit savoir quelles instructions il faut utilisées et dans quel ordre les écrire. C'est "l'art du programmeur". **Dans tous les cas** le programme est **un fichier de texte** contenant le code.

Regardons quelques exemples de langages :

Nous voulons saisir 2 valeurs numériques et afficher la somme de ces 2 valeurs.

Basic

```
10 print "saisir A"
20  input $a
30  print "saisir B"
40  input $b
50  Print "la valeur de A+B est : " , $a+$b
60 END
```

C

```
#include <stdio.h>        /* I/O  library */

int main ( ) {
int a , b ;
   printf( "saisir A " ) ;
   scanf( "%d", &a ) ;
   printf( "saisir B " ) ;
   scanf( "%d", &b) ;
   printf( "la valeur de A+B est :  %d " , a+b ) ;
}
```

C++

```
#include <iostream>        // I/O  library
using namespace std;

int main ( ) {
int a , b ;
   cout << "saisir A " ;
   cin >> a ;
   cout << " saisir B " ;
   cin >> b ;
   cout << "la valeur de A+B est : " << a+b ;
}
```

Php

```
<?php
  echo "saisir A" ;
  $a = fgets(STDIN);
  echo "saisir B" ;
  $b = fgets(STDIN);
  echo "la valeur de A+B est : ", $a+$b
?>
```

java

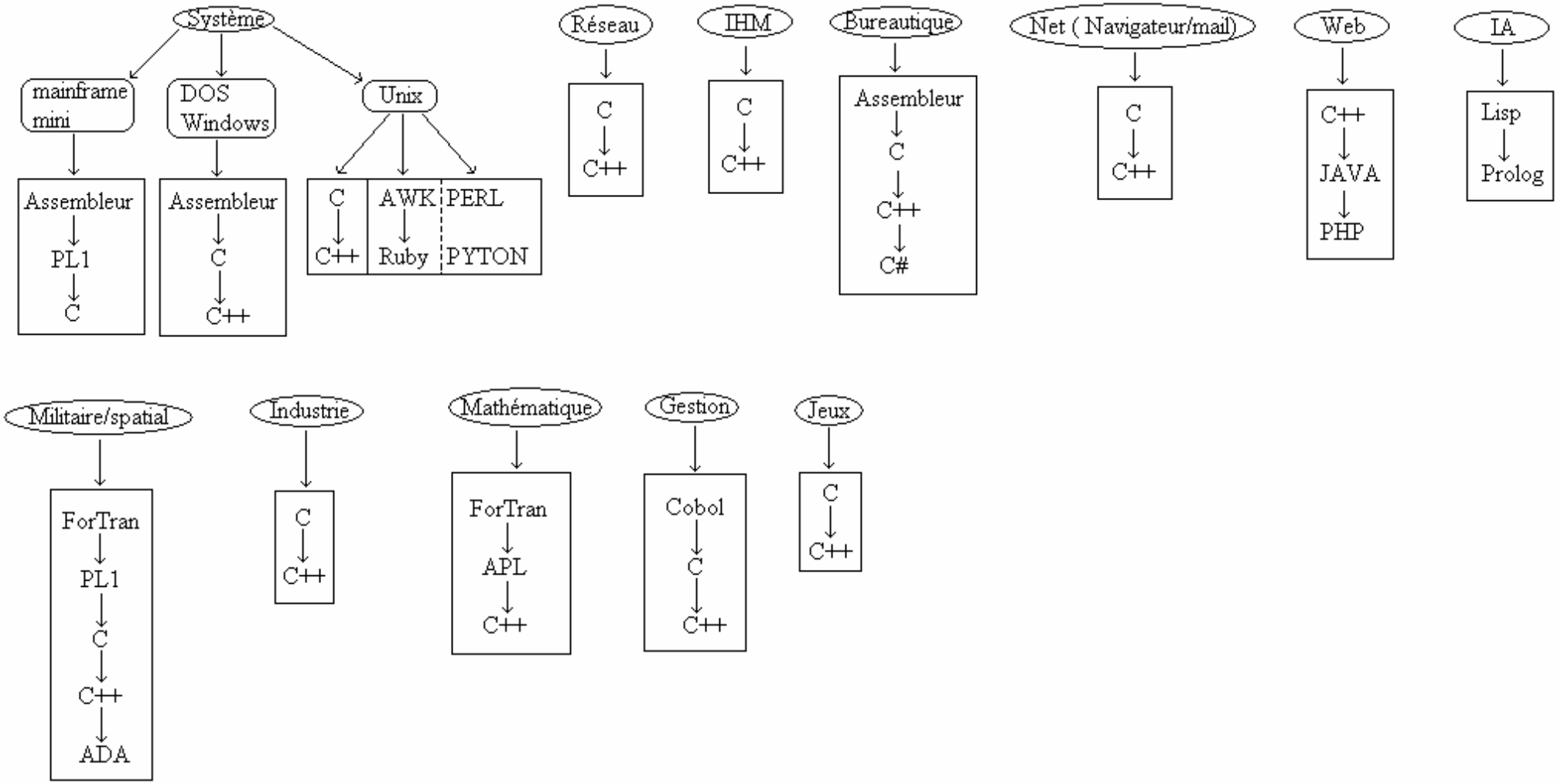
```
public class addition {
  public static void main(String args[]) {
    System.out.println("saisir A");
    BufferedReader entree = new BufferedReader(new InputStreamReader(System.in));
    int a = Integer.parseInt( entree.readLine );

    System.out.println("saisir B");
    BufferedReader entree = new BufferedReader(new InputStreamReader(System.in));
    int b = Integer.parseInt( entree.readLine );

    System.out.println("la valeur de A+B est : ", a+b );
  }
}
```

Nota : dans tous les langages, les "entrées/sorties" sont les instructions les plus difficiles à implémenter : elles dépendent du matériel et du système sur lequel on travail. L'exemple, simple conceptuellement, est très complexe informatiquement parlant. Il fait appel à des ressources systèmes très sophistiquées.

Domaine d'utilisation des langages



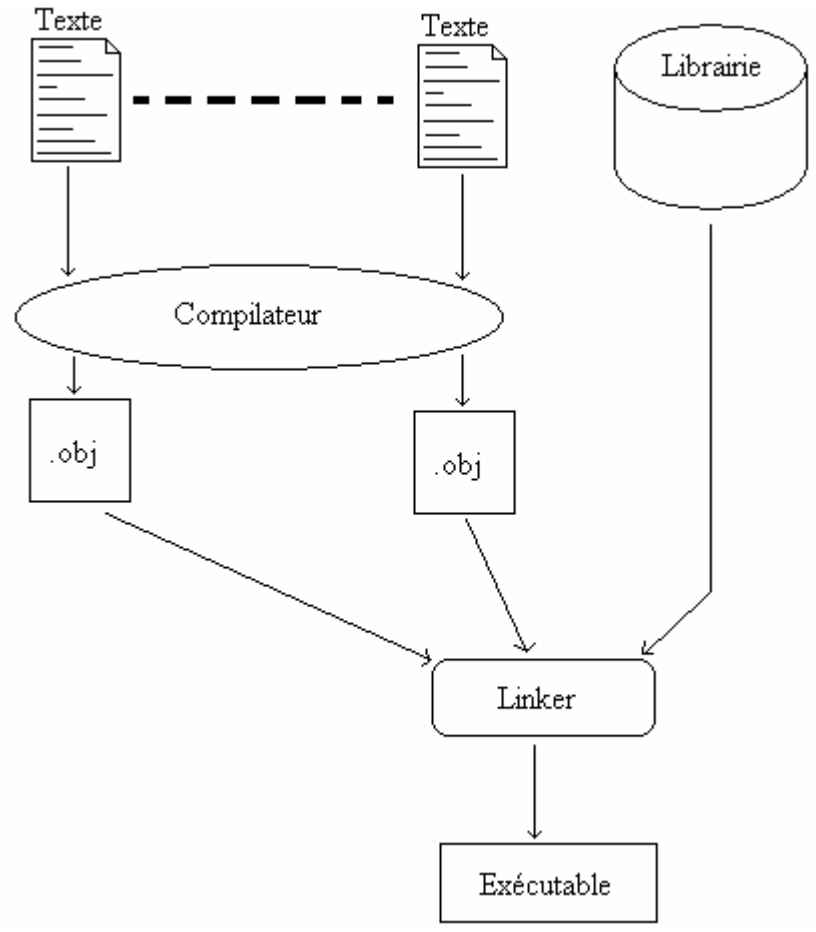
Exécution d'un programme

Ca y est, on a réussit à écrire un fichier de texte contenant le code source de notre programme, maintenant il faut l'exécuter pour vérifier s'il fonctionne correctement.
Actuellement il y a 2 manières de faire :
Soit on travail avec un langage compilé, genre C, C++ etc., et on passe par un compilateur.
Soit on travail avec un langage interprété, genre php, perl, etc. et on passe par un interprète.

Un compilateur va prendre le source qu'on lui donne pour générer un exécutable. On a donc un fichier texte au départ et un binaire à l'arrivé.
Lorsque l'on compile, le résultat est l'exécutable. Il faut "lancer" cet exécutable pour faire fonctionner le programme.
Un interprète va prendre le source qu'on lui donne pour exécuter les instructions qu'il lit dans le fichier de code.

Compilateur

C'est un programme qui traduit les sources en un exécutable.



Les fichiers sources, du texte, x.cpp sont compilés afin de produire des fichiers x.obj.
Les fihciers x.obj sont des fichiers ou tout ce qui a pu être traduit en langage machine l'a été. Seul reste les appels de fonctions externes au fichier source : librairies, appel à des fonctions dans d'autre fichiers sources, etc.

Une librairie est une collection de fonctions qui ont toutes une relation. Par ex. la librairie d'IO, la librairie mathématique, etc.

Les fichiers x.obj sont linker (liés) avec les librairies utilisées
Afin de produire un exécutable.

Nota : "Le compilateur" actuel est le GNU. C'est un programme développé par la FSF qui permet de travailler avec les langages les plus courant tel que le ForTran, C, C++, ADA, etc. Il est livré en natif sur toutes les distributions Linux. On l'appel par cc

Ex. de compilation

On a 2 fichiers sources : main.cpp qui contient le corps du programme que l'on veut exécuter et fnc.cpp qui contient plusieurs classes et fonctions appelées dans main.cpp.
Le fichier makefile de compilation sera :

main.o : main.cpp main.h fnc.h	*main.o va dépendre de main.cpp de main.h et de fnc.h*
cc – c main.cpp	*On compile main.cpp*
fnc.o : func.cpp fnc.h	*fnc.o va dépendre de fnc.cpp et de fnc.h*
cc – c main.cpp	*On compile fnc.cpp*
try : main.o fnc.o	*L'exécutable try va dépendre de main.o et de fnc.o*
cc try – o main.o fnc.o	*On link main.o et fnc.o pour produire try*

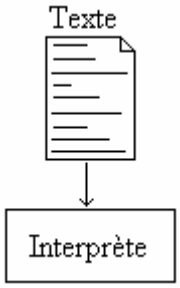
Il y a des simplifications syntaxiques pour le mafile.

```
OBJS = main.o fnc.o
.SUFFIXES :   .o   .c
# Regles implicites
.c.o:
        cc -c $<
# Regles explicites
try : $(OBJS)
        cc -o $@  $(OBJS) –l my_lib
main.o : main.h fnc.h
fnc.o : nc.h
```

Les langages compilés sont exécutés très rapidement. Ils sont utilisés quand les problèmes de vitesse sont critiques.

Interpréteur

C'est un programme spécial qui exécute les sources.



Les langages interprétés sont lents. L'interpréteur lit une ligne du source, l'exécute et passe à la ligne suivante.

Ils sont utilisés là ou la vitesse n'est pas le critère primordial, mais où la simplicité du langage, ou sa maniabilité, est primordiale. Un langage simple est plus productif qu'un langage complexe.

C'est parce que la vitesse des ordinateurs n'a cessée de croître que les langages interprétés ont eu un succès croissant.