INS ESTEVE TERRADAS

Xavier Gómez <xgomez@xtec.cat>

Curs 2019-2020

Mòdul 5 Entorns de desenvolupament

UF1. Desenvolupament de programari

NF1. Desenvolupament de programari ('software')

A3. Tipus de llenguatges i paradigmes de programació

Tipus de llenguatges de programació

Establert el concepte de programa informàtic i els conceptes de codi font, codi objecte i codi executable (així com el de màquina virtual), cal ara establir les diferències entre els diversos tipus de codi font existents, a través dels quals s'arriba a obtenir un programa informàtic.

Un **llenguatge de programació** és un llenguatge que permet establir una comunicació entre l'home i la màquina. El llenguatge de programació identificarà el codi font, que el programador desenvoluparà per indicar a la màquina, una vegada aquest codi s'hagi convertit en codi executable, quins passos ha de donar.

Al llarg dels darrers anys ha existit una evolució constant en els llenguatges de programació. S'ha establert una creixent evolució en la qual es van incorporant elements que permeten crear programes cada vegada més sòlids i eficients. Això facilita molt la tasca del programador per al desenvolupament del programari, el seu manteniment i l'adaptació. Avui en dia, existeixen, fins i tot, llenguatges de programació que permeten la creació d'aplicacions informàtiques a persones sense coneixements tècnics d'informàtica, pel fet d'existir una creació pràcticament automàtica del codi a partir d'unes preguntes.

Els diferents tipus de llenguatges són:

- Llenguatge de primera generació o llenguatge màquina.
- Llenguatges de segona generació o llenguatges d'assemblador.
- Llenguatges de tercera generació o llenguatges d'alt nivell.
- Llenguatges de quarta generació o llenguatges de propòsit específic.
- Llenguatges de cinquena generació.

El primer tipus de llenguatge que es va desenvolupar és l'anomenat **llenguatge de primera generació o llenguatge màquina**. És l'únic llenguatge que entén l'ordinador directament.

La seva estructura està totalment adaptada als circuits impresos dels ordinadors o processadors electrònics i molt allunyada de la forma d'expressió i anàlisi dels problemes propis dels humans (les instruccions s'expressen en codi binari). Això fa que la programació en aquest llenguatge resulti tediosa i complicada, ja que es requereix un coneixement profund de l'arquitectura física de l'ordinador. A més, s'ha de valorar que el codi màquina fa possible que el programador utilitzi la totalitat de recursos del maquinari, amb la qual cosa es poden obtenir programes molt eficients.

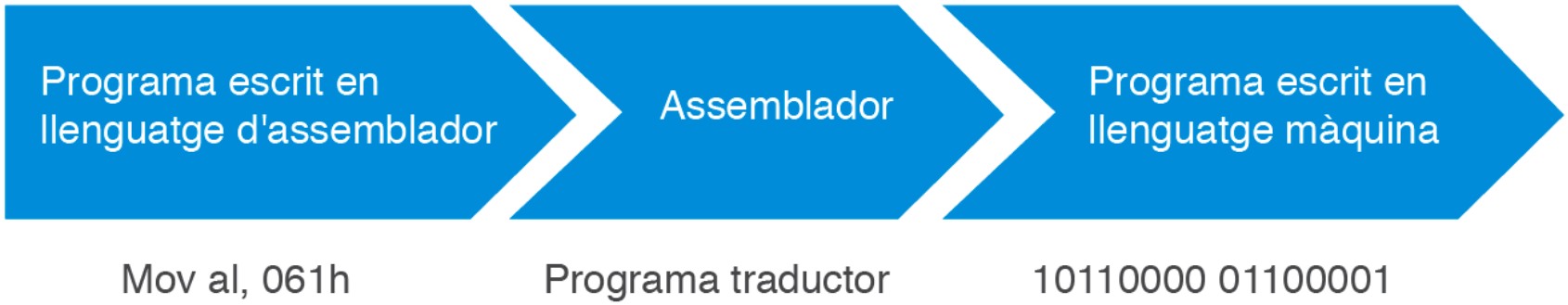
Aquesta línia conté una instrucció que mou un valor al registre del processador.
10110000 01100001

Actualment, a causa de la complexitat del desenvolupament d'aquest tipus de llenguatge, està pràcticament en desús. Només es farà servir en processadors molts concrets o per a funcionalitats molt específiques.

El segon tipus de llenguatge de programació són els **llenguatges de segona generació o llenguatges d'assemblador**. Es tracta del primer llenguatge de programació que utilitza codis mnemotècnics per indicar a la màquina les operacions que ha de dur a terme. Aquestes operacions, molt bàsiques, han estat dissenyades a partir de la coneixença de l'estructura interna de la pròpia màquina.

Cada instrucció en llenguatge d'assemblador correspon a una instrucció en llenguatge màquina. Aquests tipus de llenguatges depenen totalment del processador que utilitzi la màquina, per això es diu que estan orientats a les màquines.

A la següent figura es mostra un esquema del funcionament dels llenguatges de segona generació. A partir del codi escrit en llenguatge d'assemblador, el programa traductor (assemblador) ho converteix en codi de primera generació, que serà interpretat per la màquina.



En general s'utilitza aquest tipus de llenguatges per programar controladors (*drivers*) o aplicacions de temps real, ja que requereix un ús molt eficient de la velocitat i de la memòria.

Característiques dels llenguatges de primera i segona generació

Com a avantatges dels llenguatges de primera i segona generació es poden establir:

- Permeten escriure programes molt optimitzats que aprofiten al màxim el maquinari (*hardware*) disponible.
- Permeten al programador especificar exactament quines instruccions vol que s'executin.

Els inconvenients són els següents:

- Els programes escrits en llenguatges de baix nivell estan completament lligats al maquinari on s'executaran i no es poden traslladar fàcilment a altres sistemes amb un maquinari diferent.
- Cal conèixer a fons l'arquitectura del sistema i del processador per escriure bons programes.
- No permeten expressar de forma directa conceptes habituals a nivell d'algorisme.
- Son difícils de codificar, documentar i mantenir.

El següent grup de llenguatges es coneix com a **llenguatges de tercera generació o llenguatges d'alt nivell**. Aquests llenguatges, més evolucionats, utilitzen paraules i frases relativament fàcils d'entendre i proporcionen també facilitats per expressar alteracions del flux de control d'una forma bastant senzilla i intuïtiva.

Els llenguatges de tercera generació o d'alt nivell s'utilitzen quan es vol desenvolupar aplicacions grans i complexes, on es prioritza el fet de facilitar i comprendre com fer les coses (llenguatge humà) per sobre del rendiment del programari o del seu ús de la memòria. Els esforços encaminats a fer la tasca de programació independent de la màquina on s'executaran van donar com a resultat l'aparició dels llenguatges de programació d'alt nivell.

Els llenguatges d'alt nivell són normalment fàcils d'aprendre perquè estan formats per elements de llenguatges naturals, com ara l'anglès. A continuació es mostra un exemple d'algorisme implementat en un llenguatge d'alt nivell, concretament en Basic. Aquest algorisme calcula el factorial d'un nombre donat a la funció com a paràmetre. Es pot observar com és fàcilment comprensible amb un mínim coneixement de l'anglès.

En Basic, el llenguatge d'alt nivell més conegut, les ordres com :

IF comptador = 10 THEN STOP

poden utilitzar-se per demanar a l'ordinador que pari si la variable comptador és igual a deu.

```
' ---------------------------------------
' Funció Factorial
' ---------------------------------------
Public Function Factorial(num As Integer)As String
Dim i As Integer
        For i = 1 To num - 1
           num = num * i
           Factorial = num
        Next
End Function
```

Com a conseqüència d'aquest allunyament de la màquina i acostament a les persones, els programes escrits en llenguatges de programació de tercera generació no poden ser interpretats directament per l'ordinador, sinó que és necessari dur a terme prèviament la seva traducció a llenguatge màquina. Hi ha dos tipus de traductors: els compiladors i els intèrprets.

Compiladors

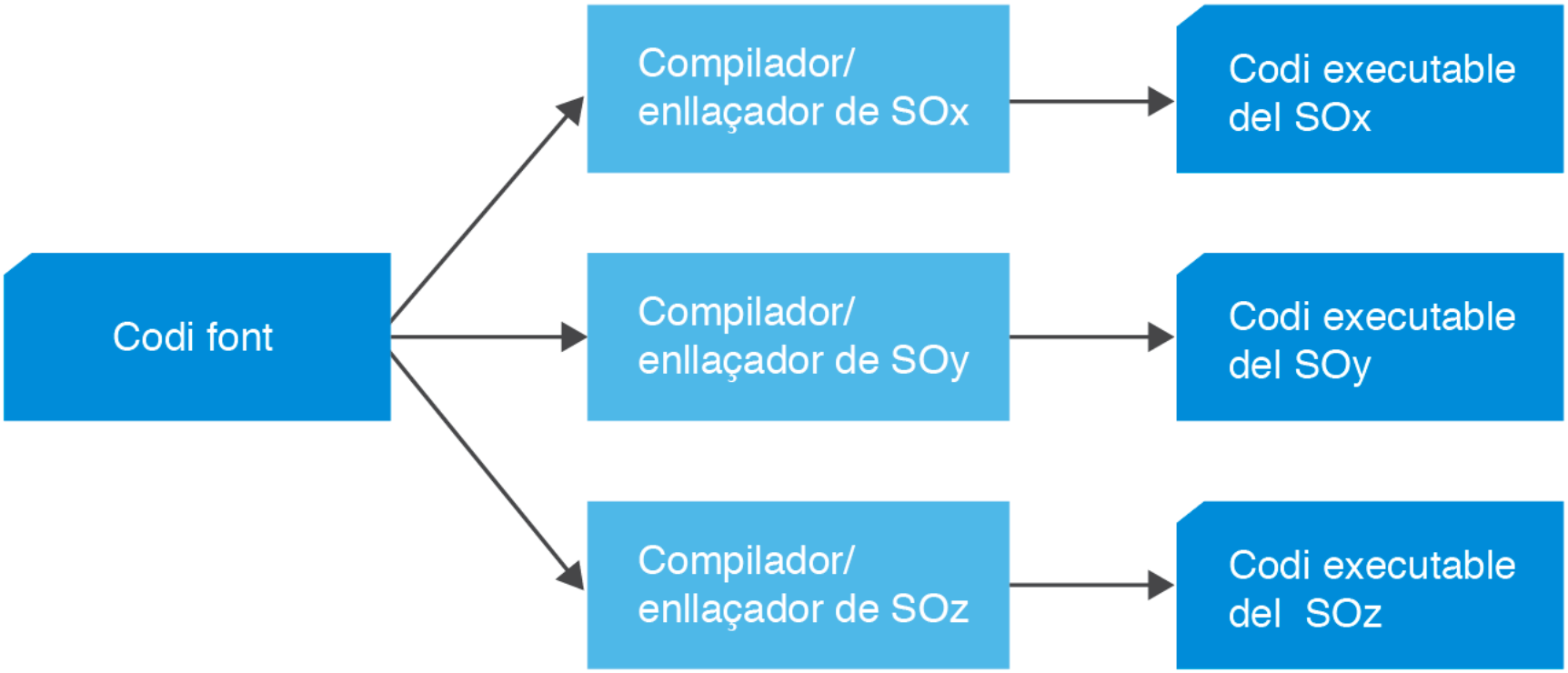
Són programes que tradueixen el programa escrit amb un llenguatge d'alt nivell al llenguatge màquina. El compilador detectarà els possibles errors del programa font per aconseguir un programa executable depurat.

Alguns exemples de codis de programació que hauran de passar per un compilador són: Pascal, C, C++, .NET, Go, Rust …

El procediment que haurà de seguir un programador és el següent:

- Crear el codi font.
- Crear el codi executable fent ús de compiladors i enllaçadors.
- El codi executable depèn de cada sistema operatiu. Per a cada sistema hi ha un compilador, és a dir, si es vol executar el codi amb un altre sistema operatiu s'ha de recompilar el codi font.
- El programa resultant s'executa directament des del sistema operatiu.

Esquema que representa la dependència del sistema operatiu a l'hora d'escollir i utilitzar compilador.



Codi compilat per SO

Els intèrprets

L'intèrpret és un programa que tradueix el codi d'alt nivell a codi de bytes, però, a diferència del compilador, ho fa en temps d'execució. És a dir, no es fa un procés previ de traducció de tot el programa font a codi de bytes, sinó que es va traduint i executant instrucció per instrucció.

Alguns exemples de codis de programació que hauran de passar per un intèrpret són: Java, Javascript, PHP, ASP…

Algunes característiques dels llenguatges interpretats són:

- El codi interpretat no és executat directament pel sistema operatiu, sinó que fa ús d'un intèrpret.
- Cada sistema té el seu propi intèrpret.

Compiladors davant intèrprets

L'intèrpret és notablement més lent que el compilador, ja que du a terme la traducció alhora que l'execució. A més, aquesta traducció es fa sempre que s'executa el programa, mentre que el compilador només la du a terme una vegada. L'avantatge dels intèrprets és que fan que els programes siguin més portables. Així, un programa compilat en un ordinador amb sistema operatiu Windows no funcionarà en un Macintosh, o en un ordinador amb sistema operatiu Linux, a menys que es torni a compilar el programa font en el nou sistema.

Característiques dels llenguatges de tercera, quarta i cinquena generació

Els avantatges dels llenguatges de tercera generació són:

- El codi dels programes és molt més senzill i comprensible.
- Són independents del maquinari (no hi fan cap referència). Per aquest motiu és possible “portar” el programa entre diferents ordinadors / arquitectures / sistemes operatius (sempre que en el sistema de destinació existeixi un compilador per a aquest llenguatge d’alt nivell).
- És més fàcil i ràpid escriure els programes i més fàcil mantenir-los.

Els inconvenients dels llenguatges de tercera generació són:

- La seva execució en un ordinador pot resultar més lenta que el mateix programa escrit en llenguatge de baix nivell, tot i que això depèn molt de la qualitat del compilador que faci la traducció.

Exemples de llenguatges de programació de tercera generació: C, C++, Java, Pascal…

Els **llenguatges de quarta generació o llenguatges de propòsit específic** aporten un nivell molt alt d'abstracció en la programació, permetent desenvolupar aplicacions sofisticades en un espai curt de temps, molt inferior al necessari per als llenguatges de 3a generació.

S'automatitzen certs aspectes que abans calia fer a mà. Inclouen eines orientades al desenvolupament d'aplicacions (IDE) que permeten definir i gestionar bases de dades, dur a terme informes (p.ex.: Oracle reports), consultes (p.ex.: informix 4GL), mòduls… , escrivint molt poques línies de codi o cap.

Permeten la creació de prototipus d'una aplicació ràpidament. Els prototipus permeten tenir una idea de l'aspecte i del funcionament de l'aplicació abans que el codi estigui acabat. Això facilita l'obtenció d'un programa que reuneixi les necessitats i expectatives del client.

Alguns dels aspectes positius que mostren aquest tipus de llenguatges de programació són:

- Major abstracció.
- Menor esforç de programació.
- Menor cost de desenvolupament del programari.
- Basats en generació de codi a partir d'especificacions de nivell molt alt.
- Es poden dur a terme aplicacions sense ser un expert en el llenguatge.
- Solen tenir un conjunt d'instruccions limitat.
- Són específics del producte que els ofereix.

Aquests llenguatges de programació de quarta generació estan orientats, bàsicament, a les aplicacions de negoci i al maneig de bases de dades.

Alguns exemples de llenguatges de quarta generació són Visual Basic, Visual Basic .NET, ABAP de SAP, FileMaker, PHP, ASP, 4D…

Els **llenguatges de cinquena generació** són llenguatges específics per al tractament de problemes relacionats amb la intel·ligència artificial i els sistemes experts.

En lloc d'executar només un conjunt d'ordres, l'objectiu d'aquests sistemes és "pensar" i anticipar les necessitats dels usuaris. Aquests sistemes es troben encara en desenvolupament. Es tractaria del paradigma lògic.

Alguns exemples de llenguatges de cinquena generació són Lisp o Prolog.

Paradigmes de programació

És difícil establir una classificació general dels llenguatges de programació, ja que existeix un gran nombre de llenguatges i, de vegades, diferents versions d'un mateix llenguatge. Això provocarà que en qualsevol classificació que es faci un mateix llenguatge pertanyi a més d'un dels grups establerts. Una classificació molt estesa, atenent a la forma de treballar dels programes i a la filosofia amb què van ser concebuts, és la següent:

- Paradigma imperatiu/estructurat.
- Paradigma d'objectes.
- Paradigma funcional.
- Paradigma lògic.

El **paradigma imperatiu/estructurat** deu el seu nom al paper dominant que exerceixen les sentències imperatives, és a dir aquelles que indiquen dur a terme una determinada operació que modifica les dades guardades en memòria.

Alguns dels llenguatges imperatius són C, Basic, Pascal, Cobol…

La tècnica seguida en la programació imperativa és la **programació estructurada**. La idea és que qualsevol programa, per complex i gran que sigui, pot ser representat mitjançant tres tipus d'estructures de control:

- Seqüència.
- Selecció.
- Iteració.

D’altra banda, també es proposa desenvolupar el programa amb la tècnica de disseny descendent (*top-down*). És a dir, modular el programa creant porcions més petites de programes amb tasques específiques, que se subdivideixen en altres subprogrames, cada vegada més petits. La idea és que aquests subprogrames típicament anomenats funcions o procediments han de resoldre un únic objectiu o tasca.

Imaginem que hem de fer una aplicació que registri les dades bàsiques del personal d’una escola, dades com poden ser el nom, el DNI, i que calculi el salari dels professors així com el dels administratius, on el salari dels administratius és el sou base (SOU_BASE) * 10 mentre que el salari dels professors és el sou base (SOU_BASE) + nombre d’hores impartides (numHores) * 12.

```
const float SOU_BASE = 1.000;

Struct Administratiu {
        string nom;
        string DNI;
        float Salari;
}

Struct Professor {
        string nom;
        string DNI;
        int numHores;
        float salari;
}

void AssignarSalariAdministratiu (Administratiu administratiu1) {
        administratiu1. salari = SOU_BASE * 10;
}

void AssignarSalariProfessor (Professor professor1) {
        professor1. salari = SOU_BASE + (numHores * 12);
}
```

El **paradigma d'objectes**, típicament conegut com a Programació Orientada a Objectes (POO, o OOP en anglès), és un paradigma de construcció de programes basat en una abstracció del món real. En un programa orientat a objectes, l'abstracció no són procediments ni funcions sinó els objectes. Aquests objectes són una representació directa d'alguna cosa del món real, com ara un llibre, una persona, una comanda, un empleat…

Alguns dels llenguatges de programació orientada a objectes són C++, Java, C#…

Un objecte és una combinació de dades (anomenades atributs) i mètodes (funcions i procediments) que ens permeten interactuar amb ell. En aquest tipus de programació, per tant, els programes són conjunts d'objectes que interactuen entre ells a través de missatges (crides a mètodes).

La programació orientada a objectes es basa en la integració de 5 conceptes: abstracció, encapsulació, modularitat, jerarquia i polimorfisme, que és necessari comprendre i seguir de manera absolutament rigorosa. No seguir-los sistemàticament, ometre'ls puntualment per pressa o altres raons fa perdre tot el valor i els beneficis que ens aporta l'orientació a objectes.

```
class Treballador {
   private:
      string nom;
      string DNI;
   protected:
      static const float SOU_BASE = 1.000;
   public:
      string GetNom() {return this.nom;}
      void SetNom (string n) {this.nom = n;}
      string GetDNI() {return this.DNI;}
      void SetDNI (string dni) {this.DNI = dni;}
      virtual float salari() = 0;
}
class Administratiu: public Treballador {
   public:
      float Salari() {return SOU_BASE * 10};
}

class Professor: public Treballador {
   private:
      int numHores;
   public:
      float Salari() {return SOU_BASE + (numHores * 15);}
}
```

El **paradigma funcional** està basat en un model matemàtic. La idea és que el resultat d'un càlcul és l'entrada del següent, i així successivament fins que una composició produeixi el resultat desitjat.

Els creadors dels primers llenguatges funcionals pretenien convertir-los en llenguatges d'ús universal per al processament de dades en tot tipus d'aplicacions, però, amb el pas del temps, s'ha utilitzat principalment en àmbits d'investigació científica i aplicacions matemàtiques.

Un dels llenguatges més típics del paradigma funcional és el Lisp. Vegeu un exemple de programació del factorial amb aquest llenguatge:

```
> (defun factorial (n)
(if (= n 0)
1
(* n (factorial (- n 1)))))
FACTORIAL
> (factorial 3)
6
```

El **paradigma lògic** té com a característica principal l'aplicació de les regles de la lògica per inferir conclusions a partir de dades.

Un programa lògic conté una base de coneixement sobre la que es duen a terme consultes. La base de coneixement està formada per fets, que representen la informació del sistema expressada com a relacions entre les dades i regles lògiques que permeten deduir conseqüències a partir de combinacions entre els fets i, en general, altres regles.

Un dels llenguatges més típics del paradigma lògic és el Prolog.

El paradigma és àmpliament utilitzat en les aplicacions que tenen a veure amb la Intel·ligència Artificial, particularment en el camp de sistemes experts i processament del llenguatge humà. Un sistema expert és un programa que imita el comportament d'un expert humà. Per tant conté informació (és a dir una base de coneixements) i una eina per comprendre les preguntes i trobar la resposta correcta examinant la base de dades (un motor d'inferència).

També és útil en problemes combinatoris o que requereixin una gran quantitat o amplitud de solucions alternatives, d'acord amb la naturalesa del mecanisme de tornada enrere (*backtracking*).

Xavier Gómez
xgomez@xtec.cat