ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

Programació

DAW

Unitat 1.

Fonaments de Programació

Autors: Carlos Cacho y Raquel Torres

Revisat per: Lionel Tarazon - lionel.tarazon@ceedcv.es

Fco. Javier Valero – franciscojavier.valero@ceedcv.es

José Manuel Martí - josemanuel.marti@ceedcv.es

Guillermo Garrido Portes - g.garridoportes@edu.gva.es

Jose Cantó Alonso – j.cantoalonso@edu.gva.es

2023/2024

Llicència

Nomenclatura

Al llarg d'aquest tema s'utilitzaran diferents símbols per a distingir elements importants dins del contingut. Aquests símbols són:

Important

Atenció

Interessant

Contingut

1. Introducció 4
2. Algoritme 5
3. Cicle de vida d'un programa 6
3.1. Fase de definició 6
3.2. Fase de desenvolupament 6
3.3. Fase de manteniment 7
4. Documentació 8
5. Elements d'un Programa 9
5.1. Dades 9
5.1.1. Constants 9
5.1.2. Variables 9
5.2. Expressions 9
5.3. Operadors 10
5.3.1. Operadors Relacionals 10
5.3.2. Operadors Aritmètics 11
5.3.3. Lògics o booleans 11
5.3.4. Parèntesi () 12
5.3.5. Operador Alfanumèric (+) 13
5.3.6. Ordre d'avaluació dels operadors 13
6. Estructures alternatives 14
6.1. Diagrames de flux (ordinogrames) 14
6.1.1. Símbols d'operació 14
6.1.2. Símbols de decisió 15
6.1.3. Símbols de connexió 15
6.1.4. Exemple 16
6.2. Pseudocodi 16
6.2.1. Exemple 17

1. Introducció

La raó principal per la qual una persona utilitza un **ordinador** és per a **resoldre problemes** (en el sentit més general de la paraula), o en altres paraules, processar una informació per a obtindre un resultat a partir d'unes dades d'entrada.

Els ordinadors resolen els problemes **mitjançant la utilització de programes escrits** pels programadors. Els programes d'ordinador no són llavors més que mètodes per a resoldre problemes. Per això, per a escriure un programa, el primer és que el programador sàpiga resoldre el problema que estem tractant.

El programador ha d'**identificar** quines són les **dades d'entrada** i a partir d'ells **obtindre** les **dades d'eixida**, és a dir, la solució, a la qual s'arribarà per mitjà del processament de la informació que es realitzarà mitjançant la utilització d'un mètode per a resoldre el problema que denominarem algoritme.

2. Algoritme

Per algoritme entenem un **conjunt ordenat i finit d'operacions que permeten resoldre un problema** que a més compleixen les següents característiques:

- Té un nombre finit de passos
- Acaba en un temps finit. Si no acabara mai, no es resoldria el problema.
- Totes les operacions han d'estar definides de manera precisa i sense ambigüitat.
- Pot tindre diverses dades d'entrada i com a mínim una dada d'eixida.

Un clar exemple d'algoritme és una recepta de cuina, on tenim uns passos que cal seguir en un ordre i han d'estar ben definits, té un temps finit i té unes dades d'entrada (ingredients) i una eixida (el plat).

Per exemple, l'algoritme per a fregir un ou podria ser el següent:

- Dades d'entrada: Ou, oli, paella, foc.
- Dades d'eixida: ou caigut.

Procediment:

1. Posar l'oli en la paella.
2. Posar la paella al foc.
3. Quan l'oli estiga calent, trencar l'ou i introduir-lo.
4. Cobrir l'ou d'oli.
5. Quan l'ou estiga fet, retirar-lo.

La codificació d'un algoritme en un ordinador es denomina **programa**.

3. Cicle de vida d'un programa

La creació de qualsevol programa (programari o software) implica la realització de **tres passos genèrics**:

- Definició: Què cal desenvolupar?
- Desenvolupament.
- Manteniment.

3.1. Fase de definició

S'intenta **caracteritzar el sistema que s'ha de construir**. S'ha de determinar la informació que ha de usar el sistema, quines funcions ha de realitzar, quines condicions existeixen, quines són les interfícies del sistema (mig comú perquè els elements no relacionats es comuniquen entre si) i quins criteris de validació s'utilitzaran.

L'estudi i definició del problema donen lloc al plantejament del problema que s'escriurà en la documentació del programa. Si no se sap el que es busca, no se'l reconeix si s'ho troba. És a dir que, si no sabem amb claredat què és el que hem de resoldre, no podrem trobar una solució. Ací es declara quina és la situació de partida i l'entorn de dades d'entrada, els resultats desitjats, on han de registrar-se i quina serà la situació final a la qual ha de conduir el problema després de ser implementat.

3.2. Fase de desenvolupament

En aquesta fase es dissenyen estructures de dades i dels programes, s'escriuen i documenten aquests, **i es prova** el software.

En aquesta etapa del cicle de vida de desenvolupament de programes, els analistes treballen amb els requeriments del programari desenvolupat en l'etapa d'anàlisi. Es determinen totes les tasques que cada programa realitza, com així també, la forma en què s'organitzaran aquestes tasques quan es codifique el programa. Els **problemes** quan són **complexos**, es poden resoldre més eficientment amb l'ordinador quan **es descomponen en subproblemes que siguen més fàcils de solucionar** que l'original. La descomposició del problema original en subproblemes més simples i a continuació dividir aquests subproblemes en uns altres més simples que poden ser implementats per a la seua solució en l'ordinador es denomina **disseny descendent** (top-down design). Els avantatges més importants del disseny descendent són:

- El problema es compren més fàcilment en dividir-se en parts més simples denominades mòduls.
- Les modificacions en els mòduls són més fàcils.
- La comprovació del problema es pot verificar fàcilment.

En aquesta etapa a més, s'utilitzen auxiliars de disseny, que són diagrames i taules que faciliten la delineació de les tasques o passos que seguirà el programa, per exemple: diagrames de flux, pseudocodi, etc.

En aquesta fase, **es converteix l'algoritme en programa**, escrit en un llenguatge de programació d'alt nivell com a C, Java, etc. La codificació del programa sol ser una tasca pesada que requereix un coneixement complet de les característiques del llenguatge triat per a aconseguir un programa eficaç. No obstant això, si el disseny de l'algoritme s'ha realitzat detalladament amb accions simples i amb bona llegibilitat, el procés de codificació pot reduir-se a una simple tasca mecànica. Les regles de sintaxis que regulen la codificació variaran d'un

llenguatge a un altre i el programador haurà de conéixer en profunditat aquestes regles per a poder dissenyar bons programes.

Per a augmentar la productivitat, és necessari adoptar una sèrie de normes, com ser:

- **Estructures acceptables** (programació estructurada)
- **Convencions de nominació**: maneres uniformes de designació d'arxius i variables
- Convencions de comentaris

3.3. Fase de manteniment

Una vegada obtingut el **programa font**, és necessària la seua **traducció al codi màquina**, ja que els programes escrits en un llenguatge d'alt nivell no són directament executables per l'ordinador. Segons la mena de traductor que s'utilitze, els llenguatges d'alt nivell es classifiquen en **llenguatges interpretats i llenguatges compilats**.

- Són **llenguatges interpretats** aquells en els quals el sistema tradueix una instrucció i l'executa, i així successivament amb les restants.
- Són **llenguatges compilats** aquells en els quals, primer es tradueix el programa font complet, obtenint-se un codi intermedi després, es fusiona aquest amb rutines o llibreries necessàries per a la seua execució en un procés anomenat **llinkat** i que obté com a resultat un mòdul executable (**programa executable**). **L'avantatge dels llenguatges compilats**, enfront dels interpretats, són la seua **ràpida execució** i, en cas de necessitar posteriors execucions del mateix programa, es farà de l'executable emmagatzemat.

La **posada a punt** consta de les següents etapes:

- Detecció d'errors.
- Depuració d'errors.
- Prova del programa.

En cadascuna d'aquestes fases es poden detectar problemes que ens fan replantejar-nos conceptes de la fase anterior i refer el programari creat amb les oportunes correccions.

4. Documentació

La major part dels projectes exigeixen la realització d'una **planificació prèvia**. Aquesta planificació ha de determinar el model de cicle de vida a seguir, els terminis per a completar cada fase i els recursos necessaris a cada moment. Tot això s'ha de plasmar en una documentació completa i detallada de tota l'aplicació.

La **documentació associada** al programari pot classificar-se en interna i externa.

- La **documentació interna** correspon a la que s'inclou dins del codi font dels programes. Ens aclareixen aspectes de les pròpies instruccions del programa.
- La **documentació externa** és la que correspon a tots els documents relatius al disseny de l'aplicació, a la descripció de la mateixa i els seus mòduls corresponents, als manuals d'usuari i els manuals de manteniment..

5. Elements d'un Programa

5.1. Dades

Les **dades** són la matèria primera de qualsevol programa informàtic i són utilitzades per realitzar operacions i generar resultats. La gestió adequada de les dades és fonamental per a un bon funcionament del programa.

S'assignen a variables o constants que tindran els següents **atributs**:

- Nom: l'identificador.
- Tipus: conjunt de valors que pot prendre.
- Valor: element del tipus que se li assigna.

5.1.1. Constants

En el mòdul que ens ocupa haurem de documentar el codi font que desenvolupem durant l'elaboració dels diferents programes.

Són elements el **valor dels quals roman invariable** al llarg de l'execució d'un programa. Una constant és la denominació d'un valor concret, de tal forma que s'utilitza el seu nom cada vegada que es necessita referenciar-lo.

Per exemple: *pi = 3.14.1592* *e = 2.718281*

També són utilitzades les constants per a facilitar la modificació dels programes, és a dir per a fer més independents unes certes dades del programa.

Per exemple: suposem un programa en el qual cada vegada que es calcula un import al qual s'ha de sumar l'IVA utilitzàrem sempre el valor 0.16, en cas de variar aquest índex hauríem d'anar buscant al llarg del programa i modificant aquest valor, mentre que si li donem nom i li assignem un valor, podrem modificar aquest valor amb molta més facilitat.

5.1.2. Variables

Són **elements** el valor dels quals **pot ser modificat** al llarg de l'execució d'un programa.

Per exemple: una variable per a calcular l'àrea d'una circumferència determinada, una variable per a calcular una factura, etc.

5.2. Expressions

Les expressions són combinacions de valors, operadors i funcions que produeixen un resultat. Les expressions segons el resultat que produïsquen es classifiquen en:

- Numèriques: Són les que produeixen resultats de tipus numèric. Es construeixen mitjançant els operadors aritmètics.

  Per exemple: pi*sqr(x) (2*x)/3

- Alfanumèriques: Són les que produeixen resultats de tipus alfanumèric. Es construeixen mitjançant operadors alfanumèrics.

  Per exemple: *"Don " + "José"*

- Booleanes o lògiques: Són les que produeixen resultats de tipus Vertader o Fals. Es construeixen mitjançant els operadors relacionals i lògics.

  Per exemple: *a < 0* *(a > 1) and (b < 5)*

5.3. Operadors

Els operadors són símbols que s'utilitzen per a realitzar operacions entre valors o variables dins d'una expressió.

5.3.1. Operadors Relacionals

S'usen per a formar expressions que en ser avaluades **retornen un valor booleà**: vertader o fals.

OPERADOR	DEFINICIÓ
<	Menor que
>	Major que
==	Igual que
>=	Major o igual que
<=	Menor o igual que
<>	Distints que

Exemples:

Expressió	Resultat
A'<b'	Vertader, ja que en còdic ASCII la A està abans que la B
1<6	Vertader
10<2	Fals

ASCII (acrònim anglés d'American Standard Code for Information Interchange — Codi Estàndard Estatunidenc per a l'Intercanvi d'Informació), és un codi de caràcters basat en l'alfabet llatí, tal com s'usa en anglés modern. Pots veure la taula en el següent enllaç: http://ascii.cl/es/

5.3.2. Operadors Aritmètics

S'utilitzen per a realitzar operacions aritmètiques.

OPERADOR	DEFINICIÓ
+	Suma
-	Resta
*	Multiplicació
^	Potència
/	Divisió
%	Reste de la división

Exemples:

Expressió	Resultat
3 + 5 - 2	6
24 % 3	0
8 * 3 - 7 / 2	20.5

5.3.3. Lògics o booleans

La combinació d'expressions amb aquests operadors produeixen el resultat vertader o fals.

OPERADOR	DEFINICIÓ
NO	Negació
I	Conjunció
O	Disjunció

El comportament d'un operador lògic es defineix mitjançant la seua corresponent **taula de veritat**, en ella es mostra el resultat que produeix l'aplicació d'un determinat operador a un o dos valors lògics. Les operacions lògiques més usuals són:

- NO lògic (**NOT**) o negació:

A	NOT A
V	F
F	V

- Taula de veritat del NOT -

L'operador NOT inverteix el valor: Si és vertader (V) retorna fals (F), i viceversa.

- O lògica (**OR**) o disjunció:

A	B	A OR B
V	V	V
V	F	V
F	V	V
F	F	F

- Taula de veritat del OR -

L'operador OR retorna vertader (V) si algun dels dos valors és vertader. En cas contrari, retorna Fals (F).

- I lògica (**AND**) o conjunció:

A	B	A AND B
V	V	V
V	F	F
F	V	F
F	F	F

- Taula de veritat del AND –

L'operador **AND retorna Vertader (V) només si tots dos valors són vertaders**. En qualsevol altre cas retorna Fals (F).

Exemples:

Expressió	Resultat
9 == (3 * 3)	Vertader
3 <> 2	Vertader
9 = (3 * 3) Y 3<>2	Vertader
3>2 Y b<a	Vertader Y Fals → Fals
3>2 O b<a	Vertader O Fals → Vertader
NO (a<b)	NO Vertader → Fals
5 > 1 Y NO(b>a)	Vertader Y NO Fals → Vertader

5.3.4. Parèntesi ()

Els parèntesis () són símbols utilitzats en programació per agrupar o delimitar expressions dins d'una expressió més gran.

Exemples: *Operació (3*2) + (6/2) ⟶ Resultat 9*

5.3.5. Operador Alfanumèric (+)

Uneix dades de tipus alfanumèric. També dit **concatenació**.

Exemples:

Expressió	Resultat
“Ana ” + “Ferrandis”	Ana Ferrandis
“boca” + “badat”	bocabadat

5.3.6. Ordre d'avaluació dels operadors

A l'hora de resoldre una expressió, l'ordre a seguir és el següent:

1. Parèntesi ()
2. Potència ^
3. Multiplicació i divisió * /Sumes i restes + -
4. Concatenació +
5. Relacionals < <= > >= == <>
6. Negació NOT
7. Conjunció AND
8. Disjunció OR

L'**avaluació d'operadors d'igual ordre** es realitza **d'esquerra a dreta**. Aquest ordre d'avaluació té algunes modificacions en determinats llenguatges de programació.

6. Estructures alternatives

Existeixen diverses formes de representació d'algoritmes. Les més importants són els diagrames de flux (també anomenats 'ordinogrames') i el pseudocodi.

6.1. Diagrames de flux (ordinogrames)

Durant el disseny d'un programa i en les seues fases d'anàlisis i programació, sorgeix la necessitat de representar d'una manera gràfica els fluxos que seguiran les dades manipulades per aquest, així com la seqüència lògica de les operacions per a la resolució del problema.

Aquesta representació gràfica ha de tindre les següents qualitats:

1. Senzillesa en la seua construcció.
2. Claredat en la seua compressió.
3. Normalització en el seu disseny.
4. Flexibilitat en les seues modificacions.

En la pràctica se solen utilitzar indistintament els termes **diagrama de flux, organigrama** i **ordinograma** per a referenciar qualsevol representació gràfica dels fluxos de dades o de les operacions d'un programa.

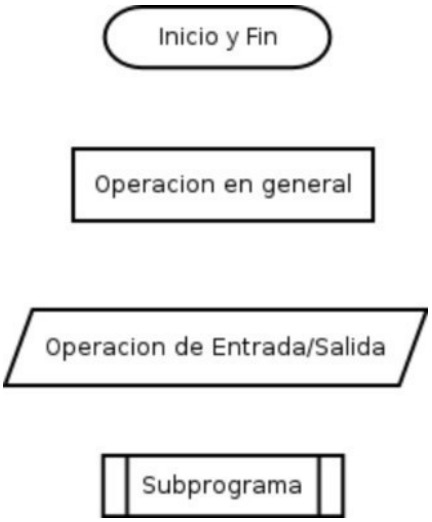
És important diferenciar-los perquè no corresponen a les mateixes fases de disseny dels programes. Encara que utilitzen alguns símbols comuns, el significat d'aquests no és el mateix.

En la representació de ordinogrames és convenient seguir les següents regles:

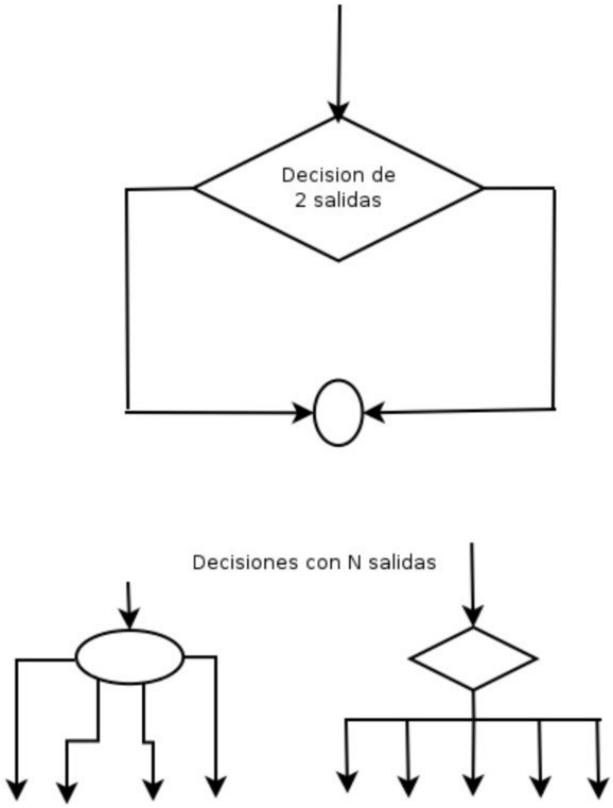
- El començament del programa figurarà en la part superior de l'ordinograma.
- El símbol de començament haurà d'aparéixer una sola vegada en l'ordinograma.
- El flux de les operacions serà, sempre que siga possible de dalt a baix i d'esquerra a dreta.
- S'evitaran sempre els encreuaments de línies utilitzant connectors.

Aquesta serà la representació que utilitzarem durant el curs.

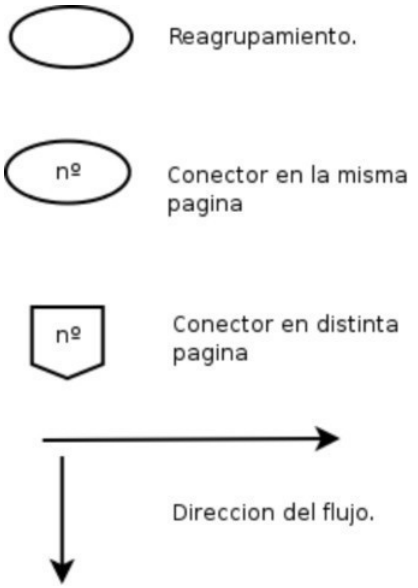
6.1.1. Símbols d'operació



6.1.2. Símbols de decisió



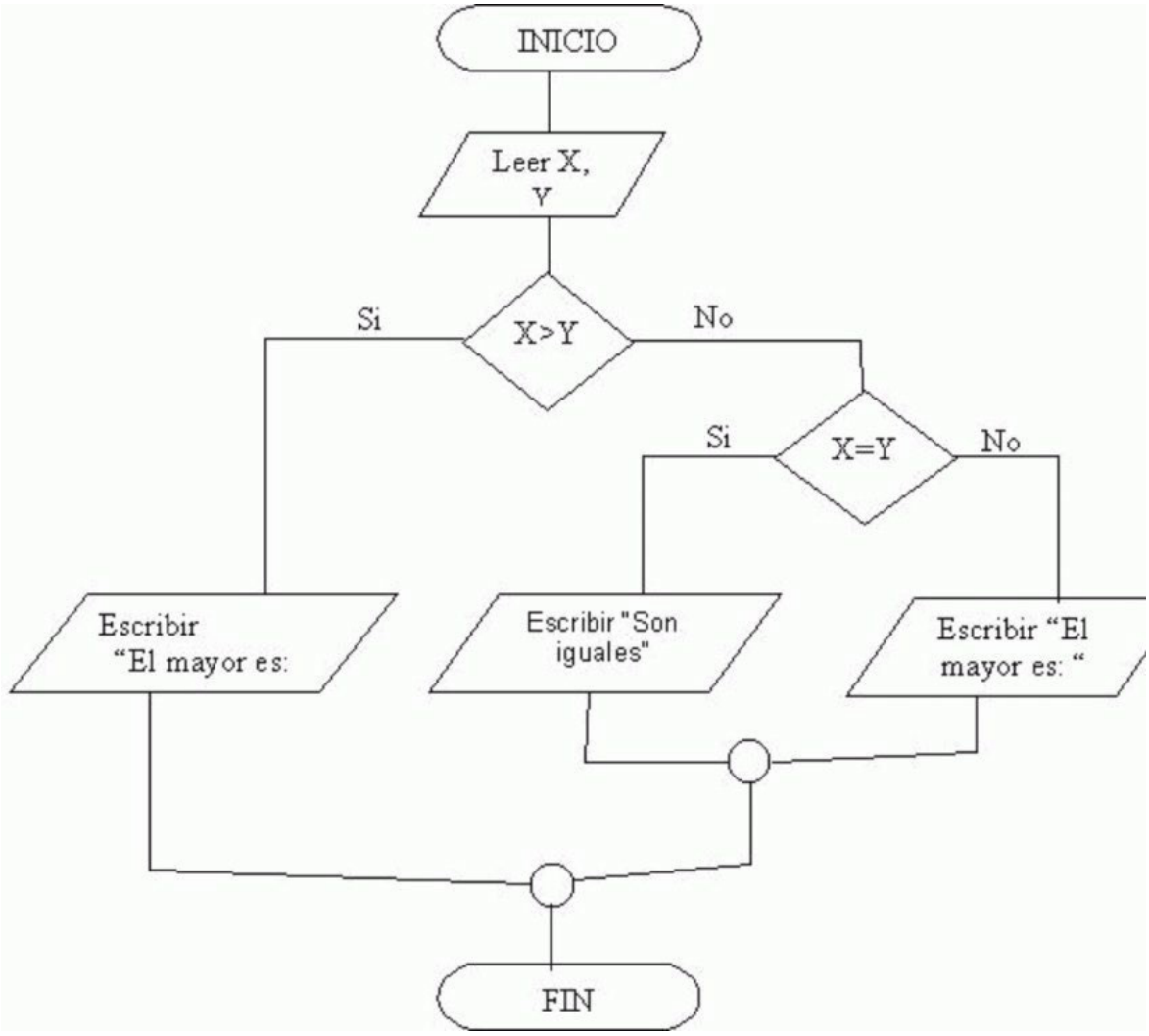
6.1.3. Símbols de connexió



6.1.4. Exemple

Algoritme que llig dos números “X” e “Y”, determina si són iguals, i en cas de no ser-ho, indica quin d'ells és el major.

La seua representació gràfica mitjançant ordinograma podem veure-la en el gràfic:



6.2. Pseudocodi

A més de les representacions gràfiques, un programa pot descriure's mitjançant un l**lenguatge intermedi entre el llenguatge natural i el llenguatge de programació**, de tal manera que permeta flexibilitat per a expressar les accions que es realitzaran i, també impose algunes limitacions, que tenen importància quan es vol codificar el programa a un llenguatge de programació determinat.

La notació en pseudocodi es caracteritza per:

1. Facilitar l'obtenció de la solució mitjançant la utilització del disseny descendent o Topdown.
2. Ser una manera de codificar els programes o algoritmes fàcil d'aprendre i utilitzar.

3. Possibilitar el disseny i desenvolupar els algoritmes d'una manera independent del llenguatge de programació que es vaja a utilitzar quan s'implemente el programa.
4. Facilitar la traducció de l'algoritme a un llenguatge de programació específic.
5. Permetre un gran flexibilitat en el disseny de l'algoritme que soluciona el problema, ja que es poden representar les accions d'una manera mes abstracta, no estant sotmeses a les regles tan rígides que imposa un llenguatge de programació.
6. Possibilitar futures correccions i actualitzacions en el disseny de l'algoritme per la utilització una sèrie de normes, que delimiten el treball del desenvolupador.

Quan s'escriu un algoritme mitjançant la utilització de pseudocodi, s'ha de **"sagnar"** el text respecte al marge esquerre, amb la finalitat que es comprenga més fàcilment el disseny que s'està realitzant.

6.2.1. Exemple

Algoritme que llig dos números "X" e "Y", determina si són iguals, i en cas de no ser-ho, indica quin d'ells és el major.

La seua representació gràfica mitjançant pseudocodi serà: