1.1 Blocs d'un programa informàtic

Un programa informàtic és una seqüència d'instruccions escrites per realitzar una tasca específica en un ordenador. Un programa té un format executable que l'ordinador pot utilitzar directament per executar les instruccions. El mateix programa en el seu format de codi font llegible per humans, de el qual es deriven els programes executables (per exemple, compilats), li permet a un programador estudiar i desenvolupar els seus algorismes.

Un programa pot considerar-se com una seqüència d'accions (instruccions) que manipulen un conjunt d'objectes (dades) podem dividir-lo per dos blocs:

- ➢ Bloc de **declaracions**: en aquest es detallen tots els objectes que utilitza el programa (constants, variables, arxius, etc.).
- ➢ Bloc **d'instruccions**: constituït pel conjunt d'operacions que s'han de realitzar per a l'obtenció dels resultats desitjats.

Dins del bloc d'instruccions hi trobem tres parts que es poden localitzar segons la seva funció:
Entrada de dades: la constitueixen totes aquelles instruccions que prenen dades d'un dispositiu extern, emmagatzemant-los en la memòria central perquè puguin ser processats.
Procés o algoritme: està format per les instruccions que modifiquen els objectes a partir del seu estat inicial fins a l'estat final, deixant aquests disponibles a la memòria central.
Sortida de resultats: conjunt d'instruccions que prenen les dades finals de la memòria central i els envien als dispositius externs.

2.4.2 Tipus de dades compostes

Qualsevol dada manipulada dins d'un programa pertany a algun dels tipus primitius, ja que són els únics que sap processar el maquinari de l'ordinador. Normalment, aquestes dades són emmagatzemades dins de variables, un dels elements bàsics d'un programa amb vista a manipular dades. Aquesta aproximació és útil sempre que el nombre de dades que es vol tractar sigui relativament limitat.

Els llenguatges de programació ofereixen uns tipus de dades especials, anomenats tipus **compostos**, que permeten aglutinar en una única variable una quantitat arbitrària de dades individuals pertanyents a algun tipus concret.
El tipus compost de dades més bàsic és **l'array**, disponible en la majoria de llenguatges d'alt nivell. Aquest permet emmagatzemar en una única variable un conjunt de valors en forma de seqüència, amb l'única restricció que tots pertanyin al mateix tipus de dades. D'aquesta manera, en una única variable podem disposar de qualsevol nombre d'enters, reals, etc.
A part dels arrays, hi ha altres tipus compost de dades de diferents graus de complexitat i amb l'objectiu de solucionar problemàtiques semblants, però amb certs matisos. Un exemple és la manipulació de text.

2.5 Estructures de selecció, repetició i salt

Les **estructures de selecció** permeten prendre decisions sobre quin conjunt d'instruccions cal executar en un punt del programa. O sigui, seleccionar quin codi s'executa en un moment determinat entre camins alternatius. Tota estructura de selecció es basa en l'avaluació d'una expressió que ha de donar un resultat booleà: true (cert) o false (fals). Aquesta expressió s'anomena la **condició lògica** de l'estructura.
En podem trobar de cuatre tipus:

- Una desviació temporal del camí: selecció simple

L'estructura de selecció simple permet controlar el fet que s'executi un conjunt d'instruccions si i només si es compleix la condició lògica (és a dir, el resultat d'avaluar la condició lògica és igual a true). En cas contrari, no s'executen.

- Dos camins alternatius: la sentència "if/else"

L'estructura de selecció doble permet controlar el fet que s'executi un conjunt d'instruccions, només si es compleix la condició lògica, i que se n'executi un altre, només si no es compleix la condició lògica

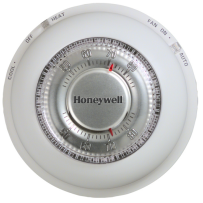
- Diversos camins: la sentència "if/else if/else"

L'estructura de selecció múltiple permet controlar el fet que en complir-se un cas entre un conjunt finit de casos s'executi el conjunt d'instruccions corresponent.

- Combinació d'estructures de selecció

Les sentències que defineixen estructures de selecció són instruccions com qualssevol altres dins d'un programa, si bé amb una sintaxi una mica més complicada. Per tant, res no impedeix que tornin a aparèixer dins de blocs d'instruccions d'altres estructures de selecció. Això permet crear una disposició de bifurcacions dins el flux de control per tal de dotar al programa d'un comportament complex, per comprovar si es compleixen diferents condicions d'acord amb cada situació.

- El commutador: la sentència "switch"



Hi ha una estructura de selecció una mica especial, motiu pel qual s'ha deixat per al final. El que la fa especial és que no es basa a avaluar una condició lògica composta per una expressió booleana, sinó que estableix el flux de control a partir de l'avaluació d'una expressió de tipus enter o caràcter (però mai real).
La sentència switch enumera, un per un, un conjunt de valors discrets que es volen tractar, i assigna les instruccions que cal executar si l'expressió avalua en cada valor diferent. Finalment, especifica què cal fer si l'expressió no ha avaluat en cap dels valors enumerats.

Sobre **<u>l'estructura de repetició</u>** o bucle podem dir fa possible l'execució repetida d'una o més instruccions. Les estructures de repetició ens permeten executar diverses vegades unes mateixes línies de codi.

Aquestes estructures descriuen processos que es repeteixen diverses vegades en la solució de el problema. El conjunt d'accions que es repeteixen conformen el cos del bucle i cada execució s'anomena iteració.
Com a l'estructura de selecció, també en podem trobar de cuatre tipus:

- <u>Repetir si es compleix una condició: la sentència "while"</u>

La sentència while permet repetir l'execució del bucle mentre es verifiqui la condició lògica. Aquesta condició es verifica al principi de cada iteració. Si la primera vegada, tot just quan s'executa la sentència per primer cop, ja no es compleix, no s'executa cap iteració.

- <u>Repetir almenys un cop: la sentència "do/while"</u>

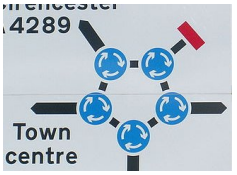
La sentència do/while permet repetir l'execució del bucle mentre es verifiqui la condició lògica. A diferència de la sentència while, la condició es verifica al final de cada iteració. Per tant, independentment de com avaluï la condició, com a mínim sempre es durà a terme la primera iteració.

- <u>Repetir un cert nombre de vegades: la sentència "for"</u>

La sentència for permet repetir un nombre determinat de vegades un conjunt d'instruccions.

- <u>Combinació d'estructures de repetició</u>

Com passava amb les estructures de selecció, res no impedeix combinar diferents estructures de repetició, imbricades unes dins de les altres, per dur a terme tasques més complexes. En darrera instància, la combinació i imbricació d'estructures de selecció i repetició conformarà el vostre programa.
Quan es combinen estructures de repetició cal ser molt acurats i tenir present quines variables de control estan associades a la condició lògica de cada bucle. Tampoc no us oblideu de sagnar correctament cada bloc d'instruccions, per poder així identificar ràpidament on acaba i comença cada estructura.



Les **<u>estructures de salt</u>** són instruccions que ens permeten trencar amb l'ordre natural d'execució dels nostres programes, podent saltar a un determinat punt o instrucció.
En trobem de tres tipus:

- <u>Break</u>

La instrucció *break* permet acabar l'execució d'una estructura condicional o iterativa, de manera que l'execució del nostre programa segueix per la instrucció següent a una d'aquestes estructures.

- Continue

La instrucció *continue* serveix per aturar l'execució d'un bucle, però si el break feia que s'executés la següent instrucció darrere del bucle, el *continue* ens porta de nou al bucle, que es tornarà a executar, sempre que es compleixi la condició d'entrada.

- Return

La sentència *return* s'utilitza sobretot per acabar l'execució d'un mètode o funció. D'aquesta manera tornem a la instrucció que va fer la crida a aquest mètode.

2.6 Tractament de cadenes

Fins ara, el tipus de dada caràcter no ha estat gaire rellevant a l'hora d'exposar exemples significatius de codi font de programes. És força habitual que en el vostre dia a dia tracteu amb conjunts de dades numèriques individuals, cadascuna independent de les altres: dates, distàncies, intervals de temps, quantitats, etc. Per tant, són dades que té sentit emmagatzemar i processar mitjançant un programa per automatitzar-ne el càlcul. En canvi, segurament no és gaire habitual que constantment useu conjunts de caràcters individuals i independents: lletres 'a', 'b', 'c', etc. La veritable utilitat dels caràcters és poder usar-los agrupats en forma de paraules o frases: un nom, una pregunta, un missatge, etc. Per tant, si un llenguatge de programació ha de ser útil, s'ha de tenir en compte aquesta circumstància.

A la programació, una **cadena de caràcters** és una seqüència ordenada (de longitud arbitrària, encara que finita) d'elements que pertanyen a un cert llenguatge formal o alfabet anàlogues a una fórmula o a una oració. En general, una cadena de caràcters és una successió de caràcters (lletres, nombres o altres signes o símbols). Si no es posen restriccions a l'alfabet, una cadena podrà estar formada per qualsevol combinació finita dels caràcters disponibles (les lletres de la 'a' a la 'z' i de la 'A' a la 'Z', els números del '0' al '9', l'espai en blanc ' ', símbols diversos '! ', '@', '%', etc).

Dins de les cadenes de caràcters podem trobar diverses funcions associades:

- **strcpy**: La funció strcpy es troba a la biblioteca <string.h> i s'utilitza per copiar una cadena de caràcters (font) en el lloc que ocupava una altra (destí). Aquesta còpia és destructiva, és a dir, que tots els caràcters que eren a la cadena destí desapareixen, encara que la cadena destí fos més llarga que la cadena font .La cadena destí es posa com a primer argument de la funció i la cadena font com a segon.
- **strcat**: Al programa anterior vam veure que la funció strcpy és desctructiva, però hi ha una altra funció a la llibreria <string.h> que copia una cadena (font) a una altra (destí) sense destruir aquesta, és a dir, que copia una cadena darrere de l'altra aquesta funció és coneguda com strcat.
- **strlen**: aquesta funció retorna el total (sencer) de caràcters que conformen una cadena (excloent el caràcter nul \ 0).
- **strcmp**: strcmp (abreviatura de ((string comparison))). La funció strcmp rep dues cadenes, a i b, retorna un enter.
  El sencer que resulta d'efectuar la crida strcmp (a, b) codifica el resultat de la comparació:

  és menor que zero si la cadena a és menor que b,
  és 0 si la cadena a és igual que b, i

és més gran que zero si la cadena a és major que b.
Naturalment, menor vol dir que va davant en ordre alfabètic, i més gran que va darrere.