Projecte Programació: Bloc C

Identifica els blocs que componen l'estructura d'un programa informàtic.

Un programa informàtic és una secuencia d'accions que manipulen un conjunt d'objectes.

Es pot dividir un programa informàtic en dos blocs principals:

- **Bloc de declaracions**: on es declaren tots els objectes que utilitza el programa.
- **Bloc d'instruccions**: conjunt d'accions que s'han de dur a terme per aconseguir els resultats esperats.

Aquest últim bloc està a la vegada compost per tres parts, tot i que no sempre es poden diferenciar clarament i apareixen mesclades en la secuencia del programa, es poden localitzar segons la seva funció. Són les següents:

- ***Entrada de dades***: instruccions que guarden a la memoria interna dades procedents d'un dispositiu extern. (DADES)
- ***Procés o algoritme***: instruccions que modifiquen els objectes d'entrada i, en ocasions, creant altres de nous. (CÀLCULS)
- ***Sortida de resultats***: conjunt d'instruccions que agafen les dades finals de la memoria interna i els envien als dispositius externs. (RESULTATS)

A continuació, veurem l'estructura bàsica d'un programa informàtic:

- ***Capçalera***: se sol especificar el nom del programa i les dades d'entrada i de sortida.
- ***Funcions***: definició de les funcions pròpies crades pel programador per a utilitzar-les en repetides ocasions.
- ***Declaracions***: definicions i tipus de variables, constants i nous tipus de dades.
- ***Assignacions***: valors inicials dels identificadors declarats prèviament.
- ***Entrades***: instruccions per a guardar en la memòria els valors d'alguns identificadors.
- ***Control***: instruccions de control de flux del programa que poden ser alternatives o repetitives.
- ***Sortides***: instruccions per a retornar els resultats obtinguts.

Escriu i prova codi que faci ús de les estructures de selecció.

Les estructures de control de flux en un llenguatge de programació especifiquen l'ordre en el que es realitza el processament de les dades.

Les estructures de selecció permeten realitzar una o una altra acció en base en una expressió lògica. Les accions possibles a realitzar són mutuament excluients, és a dir, només es pot executar una a la vegada dins de tota l'estructura.

A continuació veurem dos exemples d'estructura de selecció, en aquest cas en llenguatge Java, que és el que farem servir aquest curs per programar::

- ***Estructura if***: s'anomena estructura de selecció única perque executa un bloc de sentències només quan es compleix una condició específica. Si aquesta condició és verdadera s'executa el bloc de sentències, si és falsa el flux del programa continúa en la sentencia immediatament posterior al if.
- ***Estructura if-else***: es denomina de selciió doble ja que selecciona entre dos blocs de sentències mutuament excluients. Si es compleix la condició, s'executa el bloc de sentències associat al *if*. Si la condició no es compleix, llavors d'executa el bloc de sentències associat al *else*.

Utilitza correctament les diferents estructures de repetició disponibles.

Les estructures de repetició són les anomenades estructures cícliques, iteratives o de bucles. Permeten executar un conjunt d'instruccions de manera repetida (o cíclica) mentre que la expressió lògica a evaluar es compleixi (sigui verdadera).

Tres exemples d'estructures de repetició en llenguatge Java són:

- ***while***: indiquem una condició que s'executarà fins que la condició sigui falsa, és important que la variable que estigui en la condició varii per a que no es produeixi un bucle infinit.
- ***for***: aquest tipus de bucle, és com l'anterior per ademés de la condició, inclou una inicialització d'una variable i un increment o decrement d'aquesta variable, en principi, no és necessari que inclogui les tres parts, es poden inicialitzar o incrementar varies variables separades per comes. Quan sortim del bucle, la variable inicialitzada no existeix.
- ***do-while***: aquesta estructura repetitiva, té algo especial i és que com a mínim les instruccions dins de l'estructura repetitiva al menys una vegada, mentre que el while les seves instruccions es poden executar 0 o N vegades. Es surt del bucle quan la condició del while no es compleix, si això passa comença des del do. S'utilitza per a la validació de dades.

Reconeix les possibilitats de les sentències de salt.

Aquestes sentències permeten transferir el control del programa de forma incondicional.

En Java, existeixen quatre d'aquestes sentències:

- ***break***: s'utilitza per a sortir de la forma incondicional dels bucles do, for i while, així com de les sentències switch de multi-divisió. Es té que tenir en compte que en el cas de bucles aniuats, break fa sortir del bucle anterior.
- ***continue***: s'utilitza als bucles for, do-while, while… Als primers el control salta al final del bucle, amb la qual cosa el comptador s'incrementa i comença una altra comprovació. En el cas dels while, el control passa immediatament a la condició de control.
- ***goto***: permet transferir el control d'execució a la etiqueta especificada per l'identificador. L'etiqueta té que estar en la mateixa funció que el goto.
- ***return***: retorna el control d'execució des de la funció que conté el retorn a la rutina que l'ha invocat; opcionalmemt es pot retornar un valor.

Realitza operacions bàsiques, compostes i de tractament de caràcters

Tot valor que es pot manipular mitjançant les instruccions d'un programa sempre pertany a algun tipus de dades. Els llenguatges de programació proporcionen un conjunt de tipus, anomenat tipus primitius, amb els quals podem operar amb conjunts de valors molt bàsics.

- En Java, les dades bàsiques són les de tipus booleà, enter, real i caràcter. Ara bé, per poder crear programes complexos, sovint cal manipular conjunts de dades, d'una manera no tan bàsica. Per això, els llenguatges també ofereixen, o permeten construir, els tipus de dada compostos, que permeten emmagatzemar més d'un valor dins d'una única variable.Projecte Programació: Bloc C
- El tipus compost més estès en tots els llenguatges és l'array, que permet emmagatzemar, en forma de seqüència, una quantitat predeterminada de valors pertanyents al mateix tipus de dades. Quan s'inicialitza un array, s'estableix quina és la seva mida, o sigui, el nombre màxim de valors que pot emmagatzemar.

Un cop concretada la mida, aquesta ja no pot canviar. Tots els valors emmagatzemats dins un array sempre pertanyen al mateix tipus de dades.

Cada valor ocupa una posició dins la seqüència, de manera que el primer valor es considera la posició 0 i el darrer seria a la posició (mida − 1). Per accedir als valors emmagatzemats dins l'array s'usa un índex, un valor enter que indica la posició amb la qual es vol treballar. Per mitjà d'aquest índex és possible tant llegir el valor que hi ha en aquella posició com desar-ne de nous. Com que cada posició només pot contenir un únic valor, si se n'escriu un de diferent, l'anterior es perd. En aquest sentit, cada posició d'un array es comporta exactament igual que una variable individual, i per tant, també poden ser usades dins d'expressions.

Tot i que els arrays es poden manipular de moltes maneres, hi ha un seguit d'esquemes fonamentals per treballar-hi. El domini d'aquests esquemes és molt útil, ja que la immensa majoria de vegades que s'hi treballa, la tasca per dur a terme està estretament vinculada amb un, només amb petites variacions. Els més rellevants són:

- Inicialització procedural: es refereix al fet de generar contingut algorísmicament, per mitjà de càlculs, en lloc de fer-ho manualment partint de valors preestablerts. És un terme especialment usat dins del disseny de videojocs. Per exemple, posar a cada posició d'un array un conjunt de valors de manera que cadascun sigui el doble de l'anterior.
- Recorregut: es tracta de fer un càlcul basat en tots els valors emmagatzemats a l'array, o si més no, una bona part, de manera que per dur-lo a terme cal llegir tots els valors. Per exemple, fer el càlcul de la mitjana de tots els valors.
- Cerca: es tracta de cercar la posició on es troba un valor entre tots els emmagatzemats dins l'array. Un cop s'ha trobat, ja no cal fer res més. Per exemple, mirar si en alguna de les posicions hi ha un valor parell.
- Còpia: al contrari que amb les variables de tipus primitiu, els tipus compostos no poden ser copiats entre si amb una assignació. Si es fa, en realitat no s'obtenen dos valors iguals emmagatzemats en diferents llocs de la memòria, sinó que s'obtenen dos identificadors a partir dels quals es pot accedir exactament al mateix valor a la memòria. Les modificacions fetes via una de les variables també afectaran l'altra.

Per tant, per copiar arrays cal declarar-ne un de nou i copiar cada valor, individualment, de les posicions d'un a l'altre.
- Canvi de mida: un array no pot canviar la seva mida. L'única manera de fer-ho és declarant un nou array més gran, copiant els valors des de l'original i després afegint els nous valors a les posicions restants.
- Ordenació: a l'hora de mostrar dades de manera que les persones les puguem entendre, sovint és útil que estiguin ordenades (de menor a major, alfabèticament). Hi ha algorismes d'ordenació, com el BubbleSort, que permeten, partint d'un array en què els seus elements estan desordenats, fer que estiguin ordenats simplement fent intercanvis entre els valors de les diferents posicions.

Normalment, als valors d'un array s'accedeix usant un únic índex. Però hi ha també els arrays multidimensionals, que són aquells en què per accedir a una posició concreta, en lloc d'usar un sol valor com a índex, s'usa una seqüència de diversos valors. Cada índex serveix com a coordenada per a una dimensió diferent. Tot i semblar un concepte una mica estrany, sovint n'usem. Un exemple d'array multidimensional que usa dos índexs (un array bidimensional), és una taula. Cada cel·la s'accedeix indicant la posició de la seva fila i columna.

A part de l'array, Java ofereix altres tipus compostos molt útils. Un altre tipus compost molt usat dins els llenguatges de programació són les cadenes de text, que en Java s'identifiquen amb la paraula clau String. Aquestes permeten emmagatzemar seqüències ordenades de caràcters. Això és molt útil, ja que les persones usem normalment paraules o frases per comunicar-nos, i no pas caràcters individuals. Des del punt de vista de les dades que gestiona, una cadena de text té una certa semblança a un array de caràcters, però hi ha certes diferències importants.

La primera diferència important és a escala de nomenclatura. En Java, exceptuant el cas de l'array, s'usa el terme classe per a referir-se a qualsevol tipus de dades compost. Per tant, en les fonts de documentació normalment es parla de "la classe String". A més a més, s'usa el terme objecte per a referir-se al valor assignat a una variable d'un tipus de dades compost. Per tant, habitualment es diria "a la variable hi ha emmagatzemat un objecte String".

Com en el cas de qualsevol variable, abans de poder treballar amb una cadena de text, cal declarar i inicialitzar-ne la variable. Els literals que representen una cadena de text en Java són qualsevol text envoltat entre dobles cometes ("). Per exemple, "Hi havia una vegada", o "La suma dóna 2". La sintaxi per a l'assignació del literal a una variable és idèntica a com es faria amb un tipus primitiu, mitjançant una igualtat directament.

Ara bé, la principal diferència en Java entre valors de tipus primitius o un array i la resta de tipus compostos és que no es poden manipular usant operacions. La manipulació d'objectes es fa mitjançant la invocació de mètodes sobre les variables en què es troben emmagatzemats. Per generar una instrucció en què es crida un mètode, s'usa l'identificador de la variable, seguida d'un punt, i llavors el nom del mètode que volem, escollit entre la llista que ofereix la classe a la qual pertany la variable. Depenent del mètode utilitzat, pot ser necessari indicar un seguit d'informació addicional, en forma de llista de valors entre parèntesis, separats per comes (…, …, …). Aquesta llista és el que s'anomenen els seus

paràmetres. Si el mètode no requereix cap paràmetre, només cal obrir i tancar parèntesis, ().

El resultat d'una instrucció en què es crida un mètode és equivalent a avaluar una expressió. El seu resultat és un valor, que pot ser assignat a una variable o utilitzat dins d'altres expressions més complexes. Entre els mètodes més usats d'aquesta classe es troben:

- charAt(posició): s'avalua a el caràcter que es troba a la posició indicada dins la cadena de text. Si es considera que una cadena de text és com un array de caràcters, seria l'equivalent a llegir una posició concreta.
- indexOf(caràcter): s'avalua a l'índex de la primera posició posició on es troba el caràcter especificat, o -1 si no n'hi ha cap. Resulta molt útil per fer cerques fàcilment.
- equals(text): compara la cadena de text desada a la variable sobre la qual s'invoca el mètode amb l'especificada entre els parèntesis. S'avalua al valor booleà true si són iguals i a false en cas contrari.
- substring(posicióInici, posicióFi): s'avalua a una nova cadena de text, subcadena de la que hi ha emmagatzemada a la variable sobre la qual s'invoca. Aquesta cadena resultant comprèn tots els caràcters entre posicióInici i (posicióFi - 1) de l'original.

Cap d'aquests mètodes no modifica la cadena de text que hi ha dins la variable usada per invocar el mètode. Això es deu al fet que les cadenes de text en Java són immutables.
Tot i que els tipus de dades compostos només admeten la invocació de mètodes per manipular les seves dades, la classe String és una mica especial, ja que permet un únic operador, la suma (+).

Quan se sumen dos cadenes de text, el resultat és que el segon operador es concatena immediatament a l'esquerra del primer operador. Per exemple "Hi havia"+"una vegada" s'avalua a "Hi havia una vegada". La suma és l'únic operador que es pot usar.
Normalment, l'ús de cadenes de text està associat a l'intercanvi de dades amb l'usuari, ja sigui d'entrada, usant el teclat, o de sortida, usant la pantalla. El darrer cas és senzill, ja que només requereix l'ús de la instrucció System.out.println(…). En canvi, per entrar informació hi ha dues possibilitats.

Per una banda, es poden entrar cadenes de text mitjançant la llista dels arguments del mètode principal d'un programa. Aquesta llista s'introdueix com un conjunt de paraules a continuació del nom del fitxer executable, separades per espais. Cadascun dels valors escrits està accessible dins el mètode principal en forma d'un array de cadenes de text, la variable args, emmagatzemats a les seves posicions en el mateix ordre en què s'han escrit en executar el programa.

D'altra banda, el més habitual és que l'usuari usi el teclat per escriure informació. Per això, cal usar la classe Scanner, i invocar els mètodes que ofereix, next…, que permeten anar llegint valors de diferents tipus de dades des del teclat. La lectura es fa de manera seqüencial d'acord amb tots els valors que ha escrit l'usuari. Si en algun moment es crida algun dels mètodes d'Scanner i no hi ha valors pendents de llegir, el programa s'atura i espera que l'usuari escrigui nous valors pel teclat.

En aquest procés de lectura, sempre cal garantir que el tipus que es vol llegir es correspon a l'escrit per l'usuari; en cas contrari, hi haurà un error. Per això, la classe Scanner ofereix uns mètodes addicionals, hasNext…, que permeten comprovar si el proper valor que es llegirà des del teclat pertany o no a un tipus. Aquests s'avaluen a cert o fals si és o no el cas.