6 STRUCTURI DE DATE

6.1 Tipuri şi structuri

Un sistem de calcul este programat pentru prelucrarea instrucţiunilor. Un program este un ansamblu de instrucţiuni şi date. În capitolele precedente au fost introduse conceptele esenţiale ale programării, instrucţiuni în cod maşină şi limbaje de programare. Prezentul capitol se referă la organizarea informaţiilor de prelucrat în memoria calculatorului.

Sistemul de calcul defineşte la nivel hardware un set de instrucţiuni elementare şi câteva reprezentări ale datelor elementare.

O dată structurată se poate defini ca fiind o colecţie de date elementare. O astfel de colecţie se mai numeşte **structură de date**.

Structurile de date au apărut la nivelul limbajelor de programare evoluate care definesc structuri abstracte asupra instrucţiunilor, de exemplu, macrodefiniţii, proceduri, funcţii, subprograme, etc. Ele definesc de asemenea structuri de date precum variabile dublă precizie, numere complexe, vectori, şiruri de caractere etc.

Structurile de date sunt construite pornind de la datele elementare realizate la nivel maşină. Utilizatorul poate să lucreze şi cu informaţii mai complexe , deci la un nivel superior de abstractizare, fără să ţină seama de detaliile implementării fizice a acestor structuri care sunt **"**ascunse**"** de limbajul de programare şi software-ul asociat (traducător, interpretor, etc.).

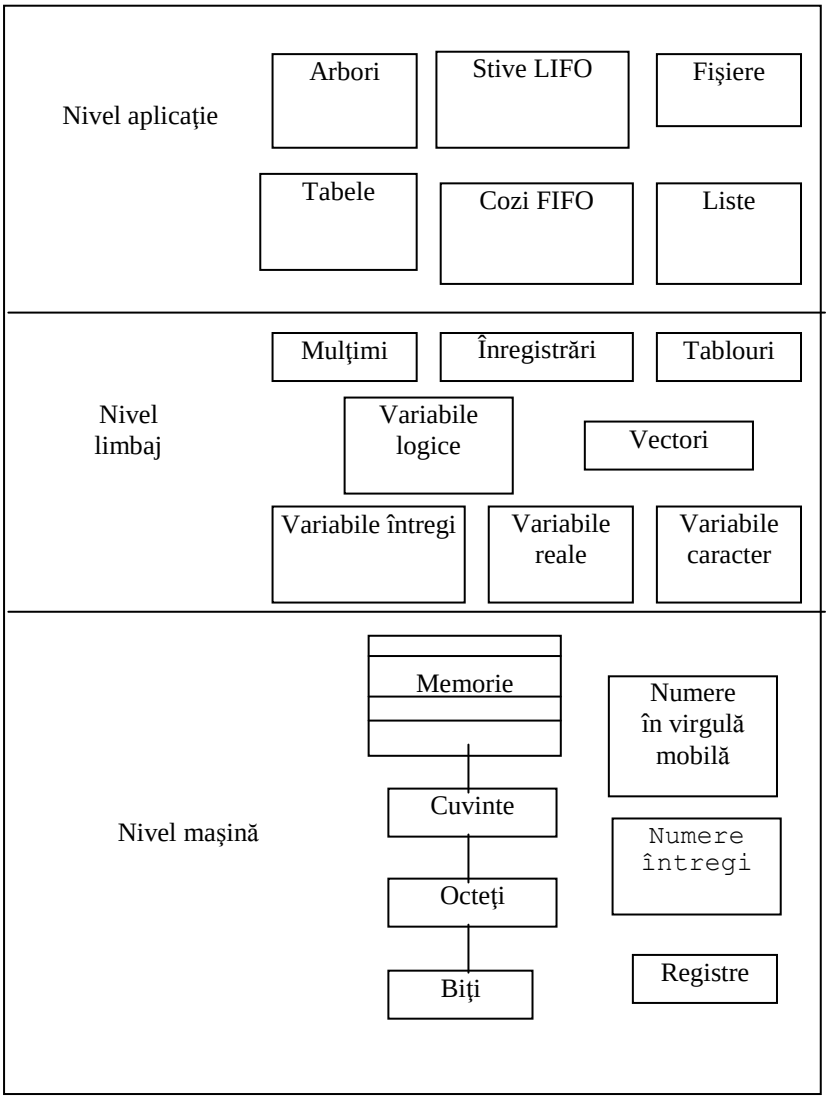
Utilizatorul poate defini structuri de date şi mai complexe (arbori, tabele, liste, etc.) utilizând datele structurate la nivelul limbajului de programare.

Pentru a introduce o nouă structură abstractă se procedează astfel :

- se defineşte structura informaţiei;
- se definesc operaţiile aplicabile acelei structuri;
- se lucrează cu această informaţie structurată la acest nivel de abstractizare, fără a se ţine seamă de detaliile de implementare;

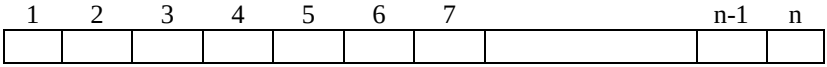
Există un mare număr de structuri de date. Vom prezenta câteva exemple dintre structurile cele mai utilizate : vectori, tablouri, liste, arbori, cozi, stive şi tabele.

Exemple de structuri de date întâlnite la diferite nivele de abstractizare sunt prezentate în figura următoare:



Vectorii [vectors] sunt structuri foarte utile, folosite de majoritatea limbajelor de programare (procesoarele vectoriale suportă vectorii la nivel hardware). Un vector este o mulţime finită şi ordonată de elemente, toate de acelaşi tip. El este definit prin numărul de elemente, ca şi prin tipul şi mărimea lor, stabilindu-se o corespondenţă imediată între structura abstractă

şi organizarea în memorie. Figura următoare prezintă schematizat structura unui vector:

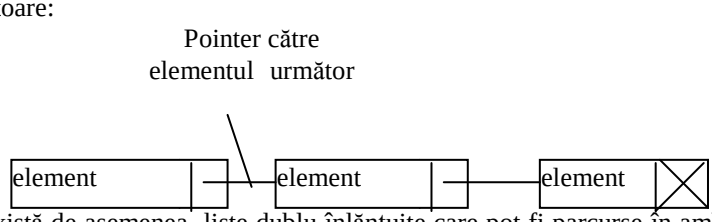


Tablouri

Un tablou [array] este o colecţie multidimensională de obiecte de acelaşi tip. Un tablou cu o singură dimensiune este un vector. Fiecare element al tabloului este reperat printr-un set de indici [subscripts]. Accesul la informaţia conţinută într-un tablou este deci aleator. Realizarea unui tablou multidimensional utilizează vectori, iar specificarea dimensiunilor necesită o declaraţie în cadrul programului.

Liste

Listele [lists] sunt structuri suple capabile să conţină un număr nedeterminat de obiecte. Dacă fiecare obiect este accesibil prin cel precedent datorită unui pointer, atunci avem de a face cu o listă legată sau înlănţuită [linked list]. Un astfel de exemplu este prezentat în figura următoare:



Există de asemenea liste dublu înlănţuite care pot fi parcurse în ambele sensuri, fiecare element fiind compus dintr-un obiect şi doi pointeri. Evident, realizarea unei astfel de liste se bazează pe noţiunea de pointer. Se spune că un pointer înlănţuieşte elementele sau pointează, asupra unui element. Pointerul este o variabilă a cărei valoare corespunde unei adrese.

Schimbând un pointer, se poate insera un nou element într-o listă, iar prin simpla modificare a pointerilor se pot suprima elemente dintr-o listă.

Arbori

Arborele este o structură constituită din noduri legate prin arce. Fiecare nod conţine un obiect al colecţiei. Arborele este o structură ierarhică. În vârf

există un nod numit **rădăcină**, de la care pornesc **ramurile**, către **nodurile** nivelului inferior. Nodurile terminale se numesc **frunze**.

Arborele binar este un caz particular în care fiecare nod posedă cel mult două ramuri către nodurile nivelului inferior.

Implementarea în memorie a unui arbore se poate realiza cu ajutorul pointerilor. Fiecărui nod îi corespunde un ansamblu de cuvinte de memorie care conţin datele şi pointerii reprezentând ramurile care pleacă din nod.

Cozi

O coadă este un fir de aşteptare (FIFO = First In First Out). Inserările se realizează la o extremitate iar extragerile se fac la celălalt capăt. Realizarea este posibilă cu ajutorul unui vector şi a doi pointeri.

Pentru a evita deplasarea cozii în memorie, se poate defini o zonă de memorie circulară, adică prima adresă să urmeze după ultima adresă. Un pointer pointează asupra ultimului element din coadă, unde se adaugă datele, iar al doilea pointer indică primul element, unde se extrag datele.

Stive

O sivă [LIFO = Last In First Out] este o structură liniară particulară, în care datele sunt adăugate sau extrase la o extremitate predeterminată.

Stiva este una dintre structurile cele mai importante şi mai utilizate (de exemplu: recursivitate, întreruperi multinivel, maşină pe 0 adrese etc.).

Implementarea în memorie a unei stive se poate face cu ajutorul unui vector şi a unui pointer care indică vârful stivei, unde datele sunt depuse sau de unde sunt extrase.

Tabele

În capitolul precedent am descris tabelele utilizate de asamblor şi compilator pentru păstrarea simbolurilor şi valorilor corespunzătoare.

Se poate defini o **tabelă** ca o structură în general neordonată de date de acelaşi tip, unde fiecare dată are asociată o informaţie unică numită cheie şi care serveşte la identificarea elementului tabelei. De exemplu, dacă vom considera o agendă telefonică, cheia este numele abonatului iar informaţiile asociate pot fi adresa şi numărul de telefon.

Căutarea unui element în tabelă se realizează prin specificarea cheii. Deşi au fost elaborate mai multe metode de căutare în tabele, nu există o tehnică universală şi optimală, alegerea depinzând de tipul şi talia tabelei, precum şi de specificul utilizării sale.

Vom prezenta trei algoritmi de căutare în tabele utilizaţi frecvent în programare: căutarea liniară, căutarea binară şi adresajul dispersat.

Căutarea liniară

Tabelele de simboluri, construite în timpul primei treceri ale unui asamblor şi în timpul fazelor de analiză lexicală şi sintactică ale unui compilator, asociază numele simbolic al unei variabile sau al unui identificator oarecare utilizat drept cheie a tabelei un anumit număr de atribute, ca de exemplu: tip, adresă, protecţie, etc. Atunci când traducătorul întâlneşte un simbol, el trebuie să verifice dacă există deja în tabelă, altfel trebuie să stabilească o nouă intrare în tabelă.

Deoarece nici simbolurile şi nici ordinea de prezentare nu sunt cunoscute în prealabil, traducătorul introduce datele în tabelă, pe măsură ce întâlneşte noi simboluri în programul sursă, deci tabela este nesortată.

Pentru a găsi un element în tabelă se poate utiliza metoda de căutare liniară. Este cea mai simplă metodă, dar şi cea mai puţin performantă. Ea constă în a parcurge tabela începând cu primul element şi comparând fiecare nume cu cheia de căutare. Cu o tabelă de n elemente, timpul mediu de căutare este proporţional cu $n / 2$.

Dezavantajul unei astfel de metode este deci numărul mare de comparaţii. Avantajele se referă la uşurinţa de realizare, aplicarea ei pentru tabele nesortate (sortarea unei tabele ia mult timp) şi faptul că pot fi efectuate căutări în timpul încărcării tabelei.

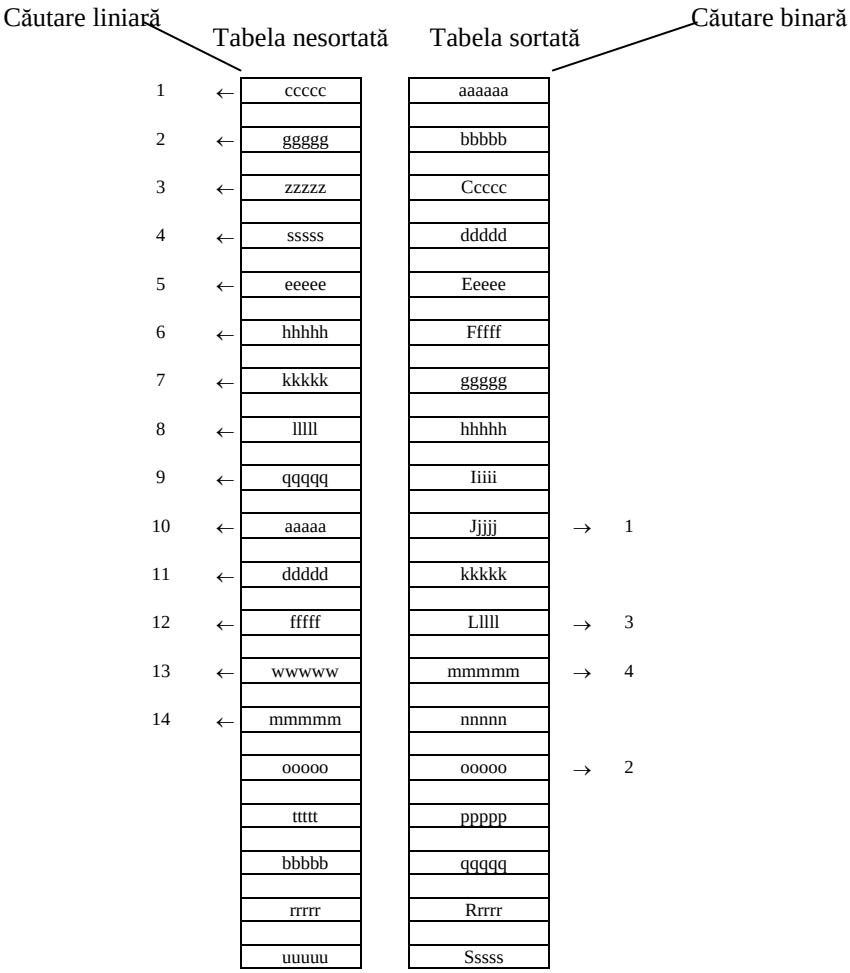
Căutarea binară

Printre tehnicile rapide, dar care impun existenţa unei tabele ordonate, se poate cita **căutarea binară** sau **dihotomică** [binary search].

Ideea constă în a considera cheile ca fiind uniform distribuite şi deci de a începe căutarea prin compararea cheii date cu cea plasată în mijlocul tabelei. Această comparare permite eliminarea dintr-o dată a unei jumătăţi a tabelei. Se poate continua în acelaşi mod aplicând metoda pentru elementele rămase ş.a.m.d. Metoda converge rapid şi durata căutării este proporţională cu $\log_2(n)$.

Avantajul acestei tehnici provine din numărul redus de comparaţii necesare pentru a găsi obiectul dorit. Pentru o tabelă cu 1000 de date, căutarea binară va putea găsi orice cheie din 10 comparări, în timp ce căutarea liniară ar necesita în medie 500 de comparări.

Figura următoare prezintă comparativ cele două căutări şi se observă că metoda de căutare liniară necesită 13 comparări în timp ce metoda comparării binare necesită doar 4 comparări ale cheii (caută: mmmmm).



Adresaj dispersat

O soluţie ideală ar fi ca să existe un mecanism de căutare rapidă care să nu impună ca tabela să fie ordonată. Memoriile asociative realizează o căutare paralelă pe toate cheile, dar o memorie asociativă de mare capacitate este costisitoare. Pentru simularea comportamentului unei memorii

asociative (acces aleator) se poate utiliza metoda de **adresaj dispersat** (hashing, hash code).

Această tehnică foarte utilizată constă în evidenţierea unei funcţii de hashcode, care se aplică cheii şi furnizează o adresă sau un indice în tabela asociată. Funcţia de hashcode este importantă, ea trebuie să furnizeze adrese destul de uniform distribuite în intervalul de adrese disponibile.

Dacă aplicarea funcţiei de hashcode asupra a două nume de simboluri alese drept cheie dă acelaşi rezultat, adică cele două obiecte ar trebui să fie plasate la aceeaşi adresă, spunem că s-a realizat o **coliziune**. O bună funcţie de hashcode trebuie să genereze un număr minim de coliziuni.

Este foarte clar că spaţiul de adrese din cadrul tabelei trebuie să fie suficient pentru a conţine toate obiectele.

Pentru rezolvarea coliziunilor se pot utiliza mai multe strategii:

- liste înlănţuite pentru adresarea obiectelor implicate, fiecare listă să înceapă la adresa comună;
- repetarea calculului de adresă cu o a doua funcţie de hashcode;
- plasarea obiectului care găseşte locul ocupat la adresa următoare sau la prima adresă disponibilă (presupune o tabelă generos dimensionată, şi în plus o căutare liniară).

Metoda adresajului dispersat permite stabilirea unei relaţii funcţionale cheie-adresă, de aceea căutarea în tabele se limitează la o singură comparare (exceptând coliziunile) după calculul adresei cu ajutorul funcţiei de hashcode.

6.2 Fişiere şi de baze de date

Structurile de date prezentate sunt adaptate organizării memoriei centrale, adresabile prin cuvinte sau octeţi.

Fişierele sunt o structură adaptată memoriilor auxiliare (unităţi de bandă şi discuri magnetice).

Un fişier este o colecţie de date reprezentând o entitate pentru utilizator. În mediul informatic, datele unui fişier sunt înregistrate astfel încât să fie facilitată citirea şi prelucrarea acestora cu calculatorul. În acest context, un fişier este deci o colecţie de date, rezultate, un text, un program sursă, un program binar executabil etc. Un fişier este conceput ca o colecţie de înregistrări. Prezentarea detaliată a conceptului de fişier va fi realizată în capitolul următor.

O **bază de date** este un ansamblu structurat de date. Conţinutul unei baze de date este cunoscut sub numele de **bancă de date.**

O bază de date permite regruparea şi centralizarea informaţiilor necesare diverselor aplicaţii în vederea unei mai bune repartizări.

Din punct de vedere logic, o bază de date (BD) [data base] este constituită din mulţimea informaţiilor relative la un subiect dat. Această mulţime trebuie să respecte următoarele criterii:

- **exhaustiviate :** informaţii complete despre un subiect dat;
- **neredondanţă :** unicitatea informaţiilor în baza de date;
- **structură :** asigură o bună gestionare a bazei de date.

Din punct de vedere fizic, o bază de date este o virtualizare a noţiunii de fişier. În general, bazele de date sunt stocate pe discuri magnetice şi gestiunea lor se efectuează direct asupra discurilor. O bază de date este implementată cu fişiere dar acest aspect nu este transparent pentru utilizatori. Stocajul fizic al unei baze de date constă într-o mulţime de înregistrări fizice organizate cu ajutorul listelor, pointerilor şi diverselor metode de indexare.

Operaţiile care se pot efectua asupra unei baze de date sunt :

- interogare sau consultare;
- actualizare;
- inserare;
- suprimare.

Pentru efectuarea acestor operaţii sunt necesare programe adecvate, reunite sub denumirea de **Sistem de gestiune de baze de date** (SGBD) [DBMS : DataBase Management System].

Un aspect important al bazelor de date actuale este independenţa datelor faţă de aplicaţii, ceea ce constituie principalul obiectiv al bazelor de date. În vederea asigurării unei accesibilităţi sporite a informaţiilor se realizează o separare între problemele de stocaj şi întreţinere, pe de o parte, şi problemele de prelucrare ale utilizatorilor, pe de altă parte, realizându-se astfel o independenţă între date şi metodele de acces.

Pentru simplificarea problemelor de acces s-au elaboart mai multe modele logice de baze de date: modelul ierarhic, modelul reţea, modelul relaţional şi modelul pe obiecte.

Modelul ierarhic constă în organizarea datelor de manieră arborescentă, ceea ce constituie o structură simplă care este o ierarhie în care fiecare element nu are decât un superior, fără a exista conexiuni între ramurile de pe acelaşi nivel.

Modelul reţea este o extensie a modelului precedent deoarece permite stabilirea conexiunilor între elemente diferite. În acest mod se poate realiza un mare număr de interogări posibile, dar acestea trebuiesc prevăzute la construirea bazei de date. În această clasă de SGBD, modelul CODASYL este unul din cele mai răspândite.

Modelul relaţional (de exemplu: SGBD Ingres, SQL/DS, Oracle, Informix, Access etc.) permite eliminarea constrângerii de cunoaştere în avans a interogărilor care se vor efectua, permiţând stabilirea conexiunilor în momentul execuţiei.

Datele sunt stocate sub formă de relaţii în tabele, iar accesul la informaţii se efectuează prin aplicarea celor trei operaţii de bază: selecţia, proiecţia, compunerea.

Selecţia constă în extragerea dintr-o relaţie a acelor elemente care satisfac o condiţie.

Proiecţia permite izolarea unui număr de coloane dorite.

Compunerea produce noi tabele pornind de la tabele existente.

Bazele de date permit definirea unei structuri logice deasupra structurii fizice (cea de fişiere pe disc) care nu este adaptată manipulării datelor. Interfaţa dintre o aplicaţie şi o bază de date este efectuată cu ajutorul limbajelor specifice, dintre care cel mai cunoscut este SQL. Limbajul SQL [Structured Query Language] este un limbaj standard de definire şi manipulare a datelor din bazele de date relaţionale.

Modelul pe obiecte este un sistem de baze de date orientate obiect [OODB-Object Oriented DataBases]. Principiul acestui nou model este relativ simplu, el constă în a utiliza aceleaşi structuri de date în aplicaţie şi în baza de date, realizând astfel o corespondenţă directă între aplicaţie şi baza de date.

Aceste structuri de date sunt clase de obiecte şi baza de date este o bază de obiecte. Obiectele bazei de date sunt obiecte persistente şi care beneficiază de polimorfism şi moştenire ceea ce garantează o bună flexibilitate.

Conceptul de obiect a dat naştere unei noi tehnologii care se aplică în numeroase domenii, de exemplu: limbajele de programare, sistemele de operare şi bazele de date.