Štruktúry údajov

Moderný číslicový počítač bol vyvinutý na uľahčenie a zrýchlenie zložitých a najmä časovo náročných výpočtov. Vo väčšine prípadov použitia počítača zohráva rozhodujúcu úlohu jeho schopnosť zapamätať si a znovu sprístupniť množstvo informácií. Táto vlastnosť počítača je snáď najcharakteristickejšou črtou. Naproti tomu schopnosť počítača počítať, vykonávať rôzne aritmetické a iné výpočty, sa v mnohých prípadoch stala takmer bezvýznamnou.

Vo všetkých uvedených prípadoch veľké množstvá informácií, ktoré je potrebné spracovať, predstavujú určitú abstrakciu reálneho sveta. Informácia vstupujúca do počítača pozostáva z vybranej množiny údajov reálneho sveta, o ktorej sa domnievame, že je podstatná a súčasne postačujúca na vyriešenie daného problému. Údaje predstavujú abstrakciu určitej reality. Abstrakcia je tým vlastne i zjednodušenie skutočnosti.

Ako príklad zoberme kartotéku zamestnancov nejakého podniku. Každý zamestnanec je v tejto kartotéke reprezentovaný (abstrahovaný) množinou údajov, ktoré sú dôležité pre evidenciu a účtovníctvo jeho zamestnávateľa. Tieto údaje obsahujú niektoré identifikačné informácie o pracovníkovi, napr. jeho meno, priezvisko a plat. S najväčšou pravdepodobnosťou sa však medzi týmito údajmi nebudú vyskytovať také informácie, ako sú napr. farba vlasov, váha či výška pracovníka. Tieto informácie sú pre zamestnávateľa bezvýznamné. Pri riešení konkrétneho problému, či už pomocou počítača alebo bez neho, je dôležité zvoliť abstrakciu skutočnosti, to

znamená definovať množinu údajov reprezentujúcu reálnu situáciu. Tento výber sa musí uskutočniť so zreteľom na riešený problém. Až potom nasleduje voľba reprezentácie vybraných informácií. V tejto etape už je potrebné brať do úvahy aj zariadenie, na ktorom sa budú vybrané informácie spracúvať. Pri údajoch treba mať na mysli vždy aj operácie, ktoré sa budú s danými údajmi vykonávať.

Typ údajov:

- Typ údajov určuje množinu hodnôt, do ktorej daná konštanta prináleží, alebo ktoré môže daná premenná či výraz nadobudnúť, alebo ktoré môže daným operátorom alebo funkciou vypočítať.
- Typ hodnoty konštanty, premennej alebo výrazu sa dá určiť z ich deklarácie alebo zápisu bez nevyhnutnosti uskutočniť vlastný výpočtový proces programu.
- Každý operátor alebo funkcia predpokladá argumenty presne definovaných typov a poskytuje výsledok tiež presne definovaného typu. V prípade, že operátor pripúšťa argumenty rôznych typov (napr. ak + sa používa na sčítanie celých i reálnych čísel), dá sa typ výsledku určiť podľa špecifických pravidiel príslušného programovacieho jazyka).

Prirodzene, v konečnom dôsledku budú údaje reprezentované veľkým počtom binárnych číslic bez ohľadu na to, či bol program

pôvodne napísaný v jazyku vyššej úrovne, zahrňujúcom pojem typu údajov, alebo v beztypovom jazyku symbolických inštrukcií.

Programovací jazyk Pascal

Vo väčšine prípadov sa nové typy údajov definujú pomocou už definovaných typov. Hodnoty takéhoto typu sú obyčajne konglomerátmi hodnôt prvkov už definovaných typov. Hovoríme im štruktúrované typy. Vzhľadom na to, že typy jednotlivých prvkov štruktúry môžu byť štruktúrované, je možné vytvárať celé hierarchie štruktúr. Pochopiteľne, prvotné prvky celej štruktúry už nesmú byť deliteľné (štruktúrované). Preto je dôležité, aby existoval nejaký spôsob zápisu takýchto primitívnych, neštruktúrovaných typov. Jednou priamočiarou metódou je vymenovanie hodnôt tvoriacich daný typ. Okrem takýchto typov definovaných programátorom musia existovať aj nejaké štandardné typy, o ktorých hovoríme, že sú preddefinované. Takéto typy obyčajne zahŕňajú čísla a logické hodnoty.

<u>Štandardné primitívne typy</u>

- sú vo väčšine existujúcich počítačov súčasťou ich základného programového vybavenia. Sú to celé čísla, logické pravdivostné hodnoty, reálne čísla so zodpovedajúcou množinou jednoduchých

operátorov a množina vytlačiteľných znakov (*integer, boolean, real, char*).

Typ **integer** zahŕňa podmnožinu množiny celých čísel, ktorej veľkosť je rôzna pre rôzne typy počítačov. O všetkých operáciách vykonávaných nad údajmi tohto typu sa predpokladá, že sú presné a spĺňajú bežné aritmetické pravidlá. Okrem toho platí zásada, že výpočet programu sa preruší v okamihu, keď výsledok niektorej z operácií je mimo reprezentovanej podmnožiny (v Pascale to neplatí - program beží ďalej, i keď výsledok operácie "pretiekol" z podmnožiny)..

Typ **real** označuje podmnožinu reálnych čísel. Zatiaľ čo pri celočíselnej aritmetike očakávame presné výsledky, pri aritmetike s hodnotami typu real pripúšťame nepresnosti v rámci limitu chýb spôsobených zaokrúhľovaním nadol. Chyby sú dôsledkom výpočtov vykonávaných s konečným počtom číslic. Toto je zásadný dôvod na presné rozlišovanie typov integer a real vo väčšine programovacích jazykov.

Štandardný typ **boolean** obsahuje dve hodnoty, ktoré sa označujú identifikátormi true a false.. Boolovskými operátormi sú logický súčin (konjunkcia) - and , logické zjednotenie (disjunkcia) - or a logická negácia - not. Poznamenávame, že porovnávania sú operátory, ktoré davajú výsledok typu boolean. Ak sú napr. dané boolovské premenné p a q a celočíselné premenné x=5, y=8, z=10, tak výsledkom príkazov p:=x=y; q:=(x<y) and (y<=z) sú hodnoty p=false a q=true.

Štandardný typ **char** zahŕňa množinu znakov vytlačiteľných na danom počítači. Najrozšírenejšou je množina znakov definovaná Medzinárodnou organizáciou pre štandardizáciu (ISO) a najmä jej americká verzia ASCII (Americký štandardný kód pre výmenu informácií).

Podmnožiny písmen a číslic sú usporiadané a súvislé.

Typ **interval** - často sa stáva, že nejaká premenná nadobúda hodnoty určitého typ, ale iba v rámci stanoveného intervalu. Túto skutočnosť možno vyjadriť tým, že danú premennú definujeme ako premennú typu *interval*. Definíciu potom možno napísať takto:

type T=min..max

Symboly min a max reprezentujú hranice intervalu.

Príklady: **type** rok = 1900..1999

type písmeno ='A'..'Z'

type číslica ='0'..'9'

Štruktúra pole

Pole je homogénna štruktúra; pozostáva z prvkov, ktoré sú všetky jediného typu. Tento typ prvku poľa sa nazýva bázový alebo základný typ. Pole pokladáme za *štruktúru s náhodným prístupom;* všetky prvky môžu byť vybraté náhodne a sú rovnako sprístupniteľné. Na referenciu (volanie) individuálneho prvku poľa je

potrebné použiť meno celej štruktúry rozšírené o *index*, určujúci vybratý prvok. Index nadobúda hodnoty typu, definovaného ako *typ indexu* poľa.

Skutočnosť, že indexy polí, t.j. "mená" prvkov polí, musia byť určitého definovaného typu, je mimoriadne dôležitá. Indexy možno vypočítať; namiesto indexovej konštanty možno dosadiť indexový výraz, vyhodnotí sa a výsledok určuje vybraný prvok poľa.

Vo väčšine aplikácií sa na typoch pole nepredpokladá žiadne usporiadanie. Kardinalita štruktúrovaného typu je súčinom kardinalít jeho prvkov. Vzhľadom na to, že všetky prvky typu poľa A sú toho istého základného typu B, dostávame, že $kardinalita(A)=(kardinalita(B))^n$, kde n=kardinalita(I), pričom I je typ indexu poľa.

Štruktúra záznam

Najbežnejšou metódou vytvárania štruktúrovaných typov je spájanie ľubovoľných prvkov (zložiek), môžu byť aj štruktúrované, do zložených typov.

Príkladmi z matematiky sú komplexné čísla zložené z dvoch reálnych čísel alebo súradnice bodov pozostávajúce z dvoch alebo viacerých reálnych čísel podľa rozmernosti priestoru vymedzeného súradnicovou sústavou. Príkladom z oblasti spracovania údajov môže byť opis osoby pomocou niekoľkých podstatných charakteristických

údajov, akými sú meno a priezvisko, dátum narodenia, pohlavie a manželský stav. V matematike sa takýto zložený typ nazýva *karteziánsky súčin* jeho zložkových typov. Vychádza sa zo skutočnosti, že množina hodnôt definovaných týmto zloženým typom je tvorená všetkými možnými kombináciami hodnôt braných z jednotlivých množín hodnôt každého typu zložiek. Potom počet takýchto kombinácií, často nazývaných n-ticami, sa rovná súčinu počtu prvkov každej množiny zložiek, čo znamená, že kardinalita zloženého typu je daná súčinom kardinalít typov zložiek.

*Kardinalita (T)=kardinalita(T_1)*kardinalita(T_2)*...*kardinalita(T_n)*.

Štruktúra množina

Treťou základnou štruktúrou údajov, po poli a zázname, je *množina*.

Definuje sa **type** T=**set of** T_0.

Možnými hodnotami premennej x, ktorá je typu T, sú množiny prvkov typu T_0. Množinu všetkých podmnožín prvkov množiny T_0 nazývame *potenčná množina T_0*. Typ T teda zahŕňa potenčnú množinu jeho *základného typu T_0*.

Príklady: **type** celočíselnámnožina = **set of** 0 . . 30

type množinaznakov = **set of** char

type farbaočí = **set of** farba

V druhom prípade je typ množina znakov definovaný prostredníctvom štandardnej množiny znakov označenej identifikátorom typu char; v treťom prípade je typ farbaočí definovaný na základe množiny farieb

type farba=(modrá, hnedá, zelená, čierna)

Nech: **var** is : celočíselnámnožina

cs : množinaznakov

Potom je možné vytvoriť a priradiť napr. nasledujúce hodnoty uvedených typov množina:

is:=[1, 4, 9, 16, 25]

cs:=['+', '-', '*', '-']

Kardinalita množinového typu T je $kardinalita(T) = 2^{kardinalita(T0)}$, čo sa dá jednoducho odvodiť zo skutočnosti, že každý z prvkov T_0, ktorých počet sa rovná kardinalite(T_0), musí byť reprezentovaný jednou z dvoch možných hodnôt "prítomný" alebo "neprítomný" a že prvky sú navzájom nezávislé. Z hľadiska efektívnej a ekonomickej implementácie množinových typov je zrejmé, že základný typ množiny musí byť konečný a súčasne jeho kardinalita čo najmenšia.

Pre každý typ množina sú definované štyri základné operátory:

* množinový prienik

\+ množinové zjednotenie

\- množinový rozdiel

in množinová príslušnosť

Najvyššiu prioritu má prienik, potom zjednotenie a rozdiel. Operátor množinovej príslušnosti má najnižšiu prioritu a zaraďujeme ho medzi relačné operátory (vzťahové).

Reprezentácia štruktúr pole, záznam a množina

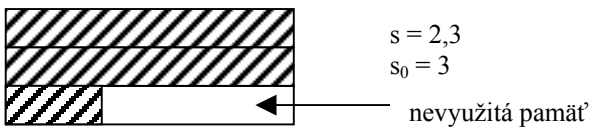
Problém reprezentácie údajov sa transformuje na problém zobrazenia abstraktných štruktúr do pamäti počítača. Pamäť počítača si môžeme - v prvom priblížení - predstaviť ako pole individuálnych pamäťových buniek nazývaných *slová*. Indexy týchto slov sa nazývajú *adresy:* **var** *pamäť :* **array** [adresa] **of** slovo

Kardinalita typov adresa a slovo sa menia v závislosti od počítačov. Zvláštnym problémom je však veľká variabilita kardinality slova. Jeho logaritmus sa nazýva *dĺžka slova*, pretože vyjadruje počet bitov v jednej pamäťovej bunke.

Reprezentácia polí

Pod reprezentáciou štruktúry pole rozumieme zobrazenie (abstraktného) poľa, ktoré prvky sú typu T, do pamäti počítača, ktorú predstavuje pole s prvkami typu *slovo*. Pole má byť zobraziteľné takým spôsobom, aby sa adresy jednotlivých prvkov poľa dali vypočítať najjednoduchším (a tým aj najefektívnejším) možným spôsobom. Adresa alebo pamäťový index j-tého prvku poľa sa vypočíta na základe lineárnej funkcie zobrazenia $i=i_0+j*s$, kde i_0 je adresa prvého prvku poľa a s je počet slov, ktoré zaberá jeden prvok poľa.

Vzhľadom na to, že podľa definície je slovo najmenšou adresovateľnou jednotkou pamäti počítača, je, samozrejme, veľmi žiaduce, aby s bolo celé číslo. Ak s nie je celé číslo (čo je normálne), tak sa obyčajne zaokrúhľuje na najbližšie väčšie celé číslo s_0. Každý prvok poľa potom zaberá s_0 slov, pričom $s_0 - s$ slov pamäti počítača ostáva nevyužitých. Zaokrúhľovanie potrebného počtu slov pamäti smerom k najbližšiemu väčšiemu celému číslu sa nazýva *vyplňovanie*.



1. Vyplňovanie pamäti zníži jej využitie.
2. Vyhnutie sa vyplňovaniu môže zapríčiniť neefektívny prístup k častiam slova.
3. Prístup k častiam slov môže zase spôsobiť, že vygenerovaný cieľový kód (skompilovaného programu) sa natoľko zväčší, že sa vlastne zneutralizuje zisk z vynechania vyplňovania.

Úvahy 2 a 3 sú tak dominantné, že kompilátory sú konštruované tak, že automaticky vykonávajú vyplňovanie pamäťového priestoru. Vo väčšine programovacích jazykov programátor nemá možnosť ovplyvniť mechanizmus určovania reprezentácie abstraktných štruktúr údajov.

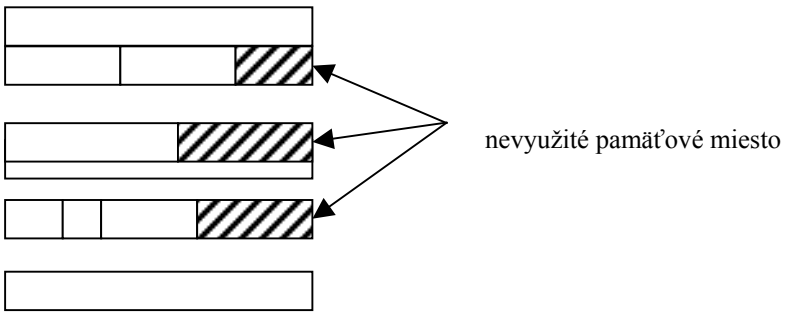
Reprezentácia štruktúr záznam

Záznamy sú zobrazované do pamäti počítača (je im pridelená pamäť) postupným zobrazovaním (ukladaním) ich jednotlivých zložiek. Adresu záznamovej zložky r_i vzhľadom na začiatočnú adresu záznamu r nazývame *posunutie* (alebo *ofset*) k_i zložky r_i. Vypočíta sa podľa vzorca $k_i = s_1 + s_2 + . . + s_{i-1}$, pričom s_j je veľkosť (v slovách) *j*-tej zložky záznamu.

Zo skutočnosti, že všetky prvky poľa sú rovnakého typu, vyplýva $s_1 = s_2 = . . . = s_n$ a preto $k_i = (i-1)*s$.

Ale vzhľadom na všeobecnú povahu štruktúry záznam takéto zjednodušenie funkcie pre výpočet posunutia zložky záznamu nie je možné. Preto sú zložky záznamu prístupné jedine prostredníctvom presne stanovených identifikátorov. Toto obmedzenie prináša zase jednu výhodu, ktorou je skutočnosť, že posunutia zložiek záznamu sú známe už v čase kompilácie. Preto nie je potrebné generovať pre výpočet posunutia kód (ako napr. v prípade indexov poľa) a tento potom vykonávať v čase realizácie programu. Dosiahne sa dobre známa efektívnosť prístupu k položkám štruktúr typu záznam.

Problém *zhusťovania* začne vystupovať v tých prípadoch, keď sa niekoľko zložiek záznamu zmestí do jedného strojového slova.



Reprezentácia množín

Množinu *s* možno vhodne reprezentovať v pamäti počítača pomocou jej *charakteristickej funkcie C(s)*. Je to pole logických hodnôt, ktorého i-ty prvok vyjadruje prítomnosť alebo neprítomnosť prvku i v danej množine. Veľkosť poľa je určená kardinalitou typu množina. $C(s_i) \equiv (i \textbf{ in } s)$

Napríklad množinu malých celých čísel s=[1, 4, 8, 9] možno reprezentovať postupnosťou logických hodnôt F (false) a T (true):

C(s) = (FTFFTFFFTT) ak základným typom množiny s je celočíselný interval 0..9. Postupnosť logických hodnôt je v pamäti počítača reprezentovaná reťazcami bitov

0 1 0 0 1 0 0 0 1 1

0 1 2 3 4 5 6 7 8 9

Reprezentácia množín prostredníctvom ich charakteristických funkcií má výhodu v tom, že operácia zjednotenia, prieniku a rozdielu dvoch množín sa dajú vhodne implementovať pomocou elementárnych logických operácií.

Z doterajších úvah vyplýva, že štruktúra množina by sa mala používať hlavne v prípade *krátkych základných typov* (v zmysle typovej kardinality). Horná hranica hodnoty kardinality základného typu, pre ktorú je ešte zaručená efektívna implementácia, sa určí z dĺžky strojového slova príslušného počítača. Je zrejmé, že z tohto hľadiska budú výhodnejšie počítače s veľkou dĺžkou strojového slova. Ak je veľkosť slova pomerne malá, tak na implementáciu typu množina sa môže použiť niekoľkoslovná reprezentácia.

Sekvenčný súbor

Spoločnou vlastnosťou doteraz uvedených štruktúr údajov (poľa, záznamu a množiny) je, že ich *kardinalita je konečná* (samozrejme, za predpokladu, že kardinalita typov ich prvkov je konečná). Z toho dôvodu ani nespôsobujú väčšie implementačné problémy; vhodné reprezentácie sa dajú bez ťažkostí nájsť a realizovať na každom číslicovom počítači.

Väčšina zložitých štruktúr, akými sú napr. postupnosti, stromy, grafy, zoznamy atď., sú charakteristické tým, že ich kardinalita je nekonečná. Dôsledok existencie takýchto štruktúr, ktorý treba brať do úvahy i pri voľbe ich reprezentácií, je, že v čase kompilácie nie je známy počet strojových slov, ktoré treba vyhradiť na reprezentáciu hodnôt typov týchto štruktúr. Veľkosť pamäti potrebnej na ich reprezentáciu sa v čase priebehu realizácie programu môže dokonca meniť. To si vyžaduje *dynamické prideľovanie pamäti* pre prípad, že hodnoty narastajú, resp. *uvoľňovanie pamäti* pre opačné prípady.

Spomedzi zložitých štruktúr spomenieme jednu, ktorá má výnimočné postavenie, a to je *postupnosť*. Tým, že jej kardinalita je nekonečná, radíme ju medzi zložité, ale vzhľadom na jej veľmi časté a rozsiahle používanie ju takmer prirodzene zaraďujeme k základným štruktúram. Nazývame ju *sekvenčný súbor* alebo skrátene *súbor*:

Typ súbor je definovaný pomocou formuly **type** T = **file of** T_0, z ktorej vyplýva, že každý súbor typu T sa skladáz 0 alebo viacerých prvkov typu T_0.

Podstata sekvenčného prístupu spočíva v tom, že v každom okamihu je bezprostredne prístupný iba jediný, špecifický prvok postupnosti, ktorý je určený momentálnou pozíciou prístupového mechanizmu. Zmenu momentálnej pozície možno dosiahnuť pomocou operátorov typu súbor obyčajne tak, aby to bola pozícia nasledujúceho alebo predchádzajúceho prvku postupnosti. Pozícia v súbore sa formálne vyjadruje tak, že súbor x pokladáme za celok, ktorý pozostáva z dvoch častí, ľavej x_L a pravej x_R.

Druhým dôležitým dôsledkom sekvenčného prístupu je, že procesy vytvárania a prehľadávania postupností sú rozdielne a nemôžu sa striedať v ľubovoľnom poradí. Súbor sa vytvára tak, že sa k nemu neustále pridávajú prvky (na jeho koniec). Zisťovanie výskytu konkrétneho prvku sa potom uskutočňuje sekvenčným prehľadávaním súboru (od jeho začiatku). Preto je súbor charakterizovaný jedným z dvoch stavov: buď je v stave jeho vytvárania (zápisu), alebo v stave prehľadávania (čítania).

Existujú pamäťové médiá, pri ktorých je sekvenčná prístupová metóda jedinou možnou metódou prístupu. Sekvenčný prístup je základnou charakteristikou každého počítačového zariadenia, ktoré sa mechanicky pohybuje.