Struktúrák/rekordok

Ionescu Klára

clara@cs.ubbcluj.ro

A struktúra/rekord adattípus

- A struktúra különböző típusú adatokat ***csoportosít***;
- Ezeket ***egyetlen adatként kezelhetjük***;
- Hozzáférünk a csoport elemeihez (***mezők***) is.
- A mező típusa lehet egyszerű, vagy összetett.
- A mezőt csupán annak a struktúra típusú változónak a megnevezésén keresztül érjük el, amelyhez tartozik.
- ***Nem lehet*** őket ***kiszámítani*** (úgy mint a tömbök indexeit), hanem le kell írnunk explicite ezeket az azonosítókat a program írásakor.

A struktúra/rekord adattípus

Példa:

Helyes

```
cin >> i;        // i = index
cout << a[i];  // a = tömb típusú változó
```

Helytelen

```
cin >> x;        // x = mezőnév
cout << struktura.x;
// „struktúra” = struct típusú változó
```

Pascal:

Helyes

```
Read(i);
Write(a[i]);
```

Helytelen

```
Read(x);
Write(struktura.x);
```

A struktúra AAT

- A **struktúra/rekord** típus a ***legkevésbé merev*** statikus adatstruktúra.
- Egy **struktúrá**nak lehet ***rögzített vagy változó számú mezője***.
- Ezeknek típusa lehet ***bármilyen***.
- A mezőket az azonosítójukon keresztül érjük el (***szelektor***).

```
struct struktura {
        mezőlista1: típus1;
        mezőlista2: típus2;
        …
        mezőlistan: típusn;
};
```

```
var struktura : record
        mezőlista1 : típus1;
        mezőlista2 : típus2;
        ….
        mezőlistan : típusn;
end;
```

Rögzített (fix) tételtípus

1) egyazon tételen belül az azonos szintű mezők azonosítóinak ***különbözők***nek kell lenniük;
2) ***a mezőazonosító (önmagában leírva!) nem kaphat értéket***; a mezőazonosító a komponens kiválasztására szolgál, tehát egy struktúratípusú változóval együtt fogjuk megnevezni;
3) a mezők típusa ***bármilyen*** lehet, tehát struktúratípus is;

```
struct nev{
        char n[20];
        int h;
};

struct datum{
    int ev;
    int ho;
    int nap;
};
```

```
struct szemely{
   nev neve;
   datum szdatum;
   nev beosztas;
   float fizetes;
   char hazas;
   nev hazastarsNeve;
   int gyereksz;
   nev gynev[12];
};

szemely alkalmazottak[100];
```

Műveletek a struktúrák szintjén

1) ***Értékátadás*** – azonos típusú tételek átadhatják egymásnak a tétel típusának megfelelő valamennyi mező értékét egyetlen utasítással.

2) ***Mezőkiválasztás*** – a tagkijelölő operátor ('**.**') után megadjuk a mezőazonosítót.

3) ***Kezdőérték-adás*** – felsoroljuk, rendre a mezőknek átadandó értékeket:

Példa:

struct típusNeve elem = { mezőÉrték1, mezőÉrték2, … }

A mezők

- A memóriában a mezők számára egymás után foglalódik le a hely.
- Mivel mindegyiknek megadtuk a típusát, a szükséges hely mérete ismert.
- Akármelyik ***m***-edik mező címe pontosan kiszámítható a tétel kezdőcíme (első mezőjének címe) és az ***m***-edik mező előtti mezők hosszából.
- ***Struktúrák mozgatása*** kerülendő, mivel a felcserélés, beszúrás stb. nagyon sok memóriaművelettel jár.

Változó rekordok
Variánsokkal rendelkező tételek Pascalban

- Van egy ***változó*** részük, amelynek a szerkezete többféle lehet.
- ***Két változó tételtípusú változó különböző szerkezettel rendelkezhet***.
- A két változónak ***különböző számú mezője lehet***, és/vagy ezek lehetnek ***különböző típusúak***.
- A tételtípus leírásában lesz egy speciális mező (***szelektor***). Ennek a mezőnek az ***értékei*** „irányítják” a variánsokat.
- ***C-ben, C++-ban nincs változó tétel, de egyszerűsíthetjük a kódban a struktúrák kezelését a referencia (&) használatával.***

Példa: *a 6. dián deklarált „személy” típusú alkalmazottak[i] tömbelem kiírása*

```
    cout << i + 1 << ". alkalmazott adatai: " << endl;
    szemely & sz = alkalmazottak[i];
    nev & ne = sz.neve;
    cout << "Nev = "; cout << ne.n; cout << endl;
    datum & d = sz.szdatum;
    cout << "Szuletes eve = "; cout << d.ev; cout <<
endl;
    cout << "Szuletes honapja = "; cout << d.ho; cout
<< endl;
    cout << "Szuletes napja = "; cout << d.nap; cout
<< endl;
    nev & b = sz.beosztas;
    cout << "Beosztas = "; cout << b.n; cout << endl;
```

Példák Pascalban/Delphiben stb.

```
type Diak = record
                nev : string[20];
                case osztondijas : Boolean of
                  true : (osztondij:140..180);
                  false : ()   { üres variáns }
                end;
  var x, y : Diak;
```

- A szelektor mező az **osztondijas**.
- Ez hozzátartozik a tételhez.
- Ha az értéke **true**, akkor a tételhez tartozik az **osztondij** mező;
- Ellenkező esetben nincs további mező.

```
type Komplex = record
                 case alak : (algebrai, trigonometriai) of
                   algebrai : (x, y : Real);
                   trigonometriai : (absz : Real; arg : Real)
                 end;
```

```
type alak_tipus = (algebrai, trigonometriai);
     Komplex = record
                  alak : alak_tipus;
                  case alak of
                  algebrai : (x, y : Real);
                  trigonometriai : (absz, arg : Real)
               end;
```

Megjegyzések:

1. a szelektor kötelezően ***sorszámozott*** típusú;
2. a **case** *állandóinak listája* ***minden*** megengedett értéket tartalmaz*;*
3. ha egy variáns ***üres***, akkor is le kell írni: **( )**;
4. a szelektor több értéke is vezethet ugyanazon variánshoz;
5. egy változó rész tartalmazhat más változó részt;
6. egy adott pillanatban csak egyetlen variáns aktív:
   a. $\exists$ szelektor mező $\Rightarrow$ a szelektor aktuális értéke szerinti variáns érvényes;
   b. nincs szelektor mező $\Rightarrow$ az utoljára kiválasztott variáns érvényes;
7. *minden mezőazonosító különböző.*

Megoldott feladat

Nyomtassuk ki egy cég adatbázisából az alkalmazottakra vonatkozó adatokat.
Három listánk lesz:

1. *a végzettség nélküliek listája,*
2. *a középvégzettségűeké, valamint*
3. *a felsőfokon képzettek listája.*

```
Program Listak;
type v_tetel = record
                  nev : string[30];
                  fizetes : Integer;
                  case tanulmanyok: (nincs, van) of
                    nincs : ();
                    van : (liceum : string[20]; erettsegi : Real;
                           case tan_tipus : (kozep, felso) of
                           kozep : ();
                     felso : (intezmeny, kar : string[20]; atlag : Real))
                  end;
       tetelek = array[1..50] of v_tetel;

var alkalmazottak:tetelek;
    n : Integer;
    nn : Integer;        { képzés nélküliek száma }
    kn : Integer;        { középfokon képzettek száma }
    fn : Integer;        { felsőfokon képzettek száma }
```

```
procedure Be(var n : Integer; var alkalmazottak : tetelek;
                                        var nn, kn, fn : Integer);
var i : Integer;      karakter : Char;
begin
  Write('Alkalmazottak szama n = '); Readln(n);
  nn := 0; kn := 0; fn := 0;
  for i := 1 to n do
    with alkalmazottak[i] do begin
      Write('Nev: '); ReadLn(nev);
      Write('Fizetes: '); ReadLn(fizetes);
      Write('Tanulmanyok (van = V, nincs = N): ');
      ReadLn(karakter);
```

```
case karakter of
'V', 'v' : begin
            tanulmanyok := van;
            Write('Liceum: '); ReadLn(liceum);
            Write('Erettsegi: '); ReadLn(erettsegi);
            Write('Tanulmanyok (kozep = k, felso = f):');
            ReadLn(karakter);
            case karakter of
              'K', 'k' : begin
                           tan_tipus := kozep;  kn := kn+1
                         end;
               'F', 'f' : begin
                            Write('Intezmeny:'); ReadLn(intezmeny);
                            Write('Kar:'); ReadLn(kar);
                            Write('Atlag:'); ReadLn(atlag);
                            tan_tipus := felso;   fn := fn+1
                          end
            end { case tan_tipus }
```

```
    'N','n': begin
                  tanulmanyok := nincs;    nn:=nn+1
             end
     end { case tanulmanyok }
   end; { with vege }
end; { Be vege }
procedure Ki_nn(nn: Integer; var alkalmazottak : tetelek);
var i : Integer;
begin
  WriteLn('Kepzes nelkuliek száma: ',nn);
  for i := 1 to nn do begin
     with alkalmazottak[i] do begin
       WriteLn('Nev: ',nev);   WriteLn('  Fizetes: ',fizetes)
     end;
   WriteLn;
   ReadLn
  end;
```

```
procedure Ki_kn(kn : Integer; var alkalmazottak : tetelek);
var i : Integer;
Begin
  WriteLn('Kozepfokon vegzettek szama: ',kn);
  for i := nn+1 to nn+kn do
     with alkalmazottak[i] do begin
        WriteLn('Nev: ',nev);
        WriteLn('  Liceum: ',liceum);
        WriteLn('  Erettsegi: ',erettsegi:5:2);
        WriteLn('  Fizetes: ',fizetes);
        WriteLn;
        ReadLn
     end;
end;
```

```
procedure Ki_fn(fn : Integer; var alkalmazottak : tetelek);
var i : Integer;
Begin
  WriteLn('Felsofokon vegzettek szama: ',fn);
  for i := nn+kn+1 to n do
    with alkalmazottak[i] do begin
      WriteLn('Nev: ',nev);
      WriteLn('  Fizetes: ',fizetes);
      WriteLn('  Liceum: ',liceum);
      WriteLn('  Erettsegi: ',erettsegi:5:2);
      WriteLn('  Intezet: ',intezmeny);
      WriteLn('  Kar: ',kar);
      WriteLn('  Atlag: ',atlag:5:2);
      WriteLn; ReadLn
    end;
end;
```

```
procedure Rendez(n : Integer; var alkalmazottak : tetelek);
var seged : v_tetel;
    i : Integer;
    rendben : Boolean;
begin
  repeat
    rendben := true;
    for i := 1 to n-1 do
      with alkalmazottak[i] do
        if (tanulmanyok > alkalmazottak[i+1].tanulmanyok) or
       ((tanulmanyok = van) and (alkalmazottak[i+1].tanulmanyok = van)
        and (tan_tipus > alkalmazottak[i+1].tan_tipus))   then begin
            seged := alkalmazottak[i];
            alkalmazottak[i] := alkalmazottak[i+1];
            alkalmazottak[i+1] := seged;
            rendben := false
        end
   until rendben;
end;
```

```
Begin
  Be(n,alkalmazottak,nn,kn,fn);
  Rendez(n,alkalmazottak);
  Ki_nn(nn,alkalmazottak);
  Ki_kn(kn,alkalmazottak);
  Ki_fn(fn,alkalmazottak);
End.
```

Feladat: Barátok

- Adott n személy, 1-től n-ig számozva ($2 \leq n \leq 100$);
- Adott p ($2 \leq p \leq 100$) természetes számpár (x_i, y_i, $2 \leq p \leq 500$);
- A számpár jelentése: az x_i sorszámú személy barátjaként nevezte meg az y_i sorszámú személyt ($i = 1, \ldots, p$).
- Ha egy bizonyos sorszám nem jelenik meg az x sorozatban, ez azt jelenti, hogy ez a személy nem nevezett meg egy barátot sem. Saját magát nem nevezheti meg!
- Ha egy bizonyos sorszám nem jelenik meg az y sorozatban, ez azt jelenti, hogy ezt a személyt senki nem nevezte meg, mint barátját.
- Ugyanaz a személy megnevezhet több barátot is, illetve egy bizonyos személyt megnevezhetnek barátként többen is.
- Ha egy x_i személy barátjaként nevezte meg az y_i személyt, innen ***nem*** következik, hogy y_i is barátként nevezte volna meg az x_i személyt.

Feladat: Barátok

- Írjatok programot, amely meghatározza:

a) Azoknak a személyeknek a sorszámát (***maxBarátok***), akik a legtöbb személyt nevezték meg barátként (a sorozat hossza: k, $1 \leq k \leq n$);

b) Azoknak a személyeknek a számát (***párokSz***), akik kölcsönösen egymást nevezték meg *egyetlen* barátjukként;

c) A statisztikába felvett személyek halmazát (a ***mind*** sorozat hossza t, $1 \leq t \leq n$).

Az a) és c) követelmények esetében az elemek sorrendje tetszőleges.

Példa: ha $n = 10$, $p = 8$, $x = (1, 3, 5, 1, 8, 4, 3, 9)$ és $y = (2, 2, 6, 8, 2, 9, 6, 4)$, akkor:

- ***maxBarátok*** $= (1, 3)$, $k = 2$, (az 1-es és a 3-as személynek 2-2 barátja van);
- ***párokSz*** $= 1$ (4-nek csak 9 a barátja és 9-nek csak 4 a barátja);
- ***mind*** $= (1, 2, 3, 4, 5, 6, 8, 9)$, $t = 8$.

Specifikáció

Bemeneti adatok:

- ***n***: a személyek száma
- ***p***: a számpárok száma
- x y: egy számpár

Kimeneti adatok:

- ***maxBarátok***: k elemű tömb; tárolja a legtöbb baráttal rendelkező személyeket;
- ***párokSzáma****:* személyek száma, akik kölcsönösen csak egymást jelölték meg barátként
- ***mind****:* a statisztikában szereplő személyek halmaza (sorozata)

Funkciók: beolvasás, a bemenet kiírása, legkevesebb három alprogram, a követelményekben megfogalmazott funkciókkal, eredmények kiírása.

Tervezés

- Észrevesszük, hogy a számpárok elemei *összetartoznak*
- Deklarálunk egy struktúrát két mezővel:

```
struct par{
  int x, y;
};
```

- A párokat sorozatban tároljuk: `par szemely[maxDim];`
- Ahhoz, hogy meghatározhassuk a legtöbb baráttal rendelkező személyeket, szükségünk lesz arra, hogy minden személy esetében meghatározzuk a barátainak számát: `baratokszama` tömb;
- A leghatékonyabb megoldás érdekében a `baratokszama` statisztikavektor lesz:

```
Algoritmus maxBarátok_a(n, p, személy, k, maxBarátok, barátokSzáma):
    Minden i = 1, n végezd el:                          { még nem fedeztünk fel egy barátot sem }
        barátokSzáma[i] ← 0
    vége(minden)
    Minden i = 1, p végezd el:
        baratokSzáma[személy[i].x] ← barátokSzáma[személy[i].x] + 1
    vége(minden)
    k ← 1          { meghatározzuk a barátokSzáma minden olyan indexét, ahol a tömb maximuma található }
    maxBarátok[k] ← 1; m ← barátokSzáma[1]                          { m = aktuális maximum }
    Minden i = 2, n végezd el:
        Ha barátokSzáma[i] > m akkor                    { találtunk m-nél nagyobb barátszámot }
            m ← barátokSzáma[i]                                     { aktualizáljuk m-et }
            k ← 1; maxBarátok[k] ← i      { m-nek ez az első előfordulása, megjegyezzük a személy indexét }
        különben
            Ha barátokSzáma[i] = m akkor                { találtunk m-mel egyenlő barátszámot }
                k ← k + 1; maxBarátok[k] ← i
            vége(ha)
        vége(ha)
    vége(minden)
Vége(algoritmus)
```

A párok száma

Veszélyek:

- Észre kell venni, hogy a két barát kölcsönösen ***csak*** egymást nevezheti meg
- Ha a memória gazdaságos felhasználásának érdekében arra gondolunk, hogy az előbbi alprogramban használt/generált sorozatok közül (barátokSzáma és maxBarátok) bármelyiket felhasználhatnánk, ez nem vezet helyes irányba, mivel egyik sem tartalmaz a fent kiemelt kizárólagosságra vonatkozó információt.
- Így újabb tömböt generálunk, amelyben azt jegyezzük meg, hogy az illető személyt feldolgoztuk-e (egy barátja van)
- De a számpár második elemének is lehet több barátja, így ezeket is megszámoljuk
- **Lásd a programot a Programok mappában**