Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar

Bevezetés a programozásba I

11. gyakorlat

C++: rekordok, típusok megvalósítása

© 2011.11.29. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Rekordok

Fontossága

- Előfordulhat, hogy egyszerre több, összetartozó adatot is szeretnénk kezelni egyszerre, egy változón belül, pl.:
  - egy racionális szám számlálóból és nevezőből áll
  - egy 3 dimenziós koordináta 3 számból áll
  - egy dátum évet, hónapot, napot jelent
  - egy telefonkönyvi bejegyzésben név, cím és számok szerepelnek
- A C++ lehetőséget biztosít, hogy már létező típusainkból létrehozott változóinkat összegyűjtsük egy közös változóba, ezt *rekord*nak nevezzük, a rekord tehát különböző típusú változók gyűjteménye (szemben a tömbbel, amely azonos típusú változók gyűjteménye)

Rekordok

Szerkezete

- Rekord segítségével saját típusokat gyárthatunk, amelyeket a későbbiekben ugyanúgy használhatunk, mint a beépített típusokat, és ezen felül hivatkozhatunk a benne található további változókra, amiket *adattagok*nak nevezünk
- A rekordban bármilyen, már ismert típusú változó lehet bármilyen kombinációban, lehetnek beépített típusok, már létrehozott saját típusok (rekordok) és tömbök is
- A rekord szerkezete:

```
struct <rekord név>{
    <típus 1> <változó 1>;
    <típus 2> <változó 2>;
    …
};
```

Rekordok

Használata

- A rekordokat függvényen kívül kell deklarálnunk, és a végén kell pontosvesszőt raknunk
- A deklarációt követően felhasználhatjuk saját típusként, tehát deklarálhatunk belőle változókat bármely utána következő függvényben:
  <rekord név> <változónév>;
- A behelyezett változókra tudunk hivatkozni, így egyenként módosíthatjuk, lekérdezhetjük az értékeiket
  <változónév>.<tagnév> formában
- A rekordokból tömböket is hozhatunk létre a megszokott módon, és ekkor is hivatkozhatunk az adattagokra:
  <rekord név> <változónév>[<méret>];
  <változónév>[<index>].<tagnév>;

Rekordok

Példák

Feladat: Valósítsuk meg a komplex számok típusát. Olvassunk be 10 komplex számot billentyűzetről, majd írjuk ki az összegüket.

- a komplex szám egy valós és egy képzetes részből áll, ez két valós szám, ezeket tegyük rekordba
- komplex számnál külön adjuk össze, kérjük be, íratjuk ki a valós és a képzetes részt
- használjunk egy alprogramot az összeadáshoz (*osszegez*), amely paraméterben megkapja a számok vektorát, és visszaadja az eredmény komplex számot

Rekordok

Példák

Megoldás:

```
…
struct komplex {  // komplex szám típusa
   float re;  // valós rész
   float im;  // képzetes rész
};
/* komplex számok összegzése
- bemenet: komplex számok vektora (szamok)
- kimenet: a számok összege
*/
komplex osszegez(vector<komplex> szamok);
/* főprogram
- bemenet: 20 valós szám a bemeneten
- kimenet: 10  komplex szám összege */
```

Rekordok

Példák

Megoldás:

```
int main() {
   vector<komplex> szamok(10);
   // 10 elemű tömb, komplex számokból
   for (int i = 0; i < 10; i++) {
      cout << i+1 << ". szám, valós: ";
      cin >> szamok[i].re; // bekérés
      cout << "képzetes: "; cin >> szamok[i].im;
   }
   komplex eredmeny = osszegez(szamok);
   cout << "Az összeg: " << eredmeny.re
        << " + i*" << eredmeny.im;
   return 0;
}
```

Rekordok

Példák

Megoldás:

```
komplex osszegez(vector<komplex> szamok) {
   komplex eredmeny;
   eredmeny.re = 0; eredmeny.im = 0;
   // eredmény kinullázása
   for (int i = 0; i < szamok.size(); i++) {
      // összegzés tagonként
      eredmeny.re += szamok[i].re;
      eredmeny.im += szamok[i].im;
   }
   return eredmeny;
}
```

Rekordok

Példák

Feladat: Valósítsuk meg egy névjegyzéket, amely névből, és születési dátumból áll, a születési dátum pedig évből, hónapból és napból. A program olvasson be tetszőleges számú adatot egy fájlból, és legyen lehetőségünk a mai napon született emberek neveit lekérdezni.

- valósítsuk meg a dátum és a bejegyzés típusokat, és ezek felhasználásával oldjuk meg a feladatot
- a műveleteket írjuk meg függvény segítségével
- a fájlnevet adjuk meg argumentumként, feltételezzük, a fájl minden sora a következő szerkezetű:

  `<vezetéknév> <keresztnév> <év>.<hó>.<nap>`, a beolvasást ennek megfelelően végezzük, hibaellenőrzéssel

Rekordok

Példák

- a műveleteket alprogramok segítségével írjuk meg, az alábbi alprogramokra van szükségünk:
  - *fajlBeolvas*: adatok beolvasása fájlból
    - megkapja paraméterként a fájlnevet, valamint a feltöltendő vektort (cím szerint), visszatérési értékként megadja, sikeres volt-e a fájl megnyitása
    - amennyiben hibás, vagy üres sor van a fájlban, azokat átugorja
  - *bejegyzesBeolvas*: adatok beolvasása egy sorból
    - megkapja a fájlt (cím szerint), valamint a beolvasandó bejegyzést (cím szerint), visszatérési értékben megadja, sikeres volt-e a beolvasás

Rekordok

Példák

- *szulinapKeres*: adott születésnap keresése (lineáris kereséssel)
  - megkapja az adatokat tartalmazó vektort, valamint a dátumot, visszaadja a szülinaposok neveit egy vektorban

Megoldás:

```
…
struct datum { // dátum típusa
   int ev; int ho; int nap;
};
```

Rekordok

Példák

Megoldás:

```
struct bejegyzes { // névjegyzék típusa
   string nev;
   datum szulIdo; // felhasználjuk a dátumot
};

/* adatok beolvasása fájlból (elemenkénti
   feldolgozás)
- bemenet: fájnév (fnev), névjegyzék vektor
  (adatok)
- kimenet: igaz, ha sikerült beolvasni az adatokat
*/
bool fajlBeolvas(string fnev, vector<bejegyzes>
&adatok);
```

Rekordok

Példák

Megoldás:

```
/* adott születésnap keresése a névjegyzékben
   (lineáris keresés)
- bemenet: névjegyzék vektor (adatok), keresett
  dátum (d)
- kimenet: az adott hónap/napon születettek
  neveinek vektora
*/
vector<string> szulinapKeres(vector<bejegyzes>
adatok, datum d);
```

Rekordok

Példák

Megoldás:

```
/*
főprogram
- bemenet: névjegyzéket tartalmazó fájl neve
- kimenet: a november 29-i születésűek nevei
*/
int main(int argc, char* argv[]) {
   vector<bejegyzes> adatok;
   if (fajlBeolvas("adatok.txt", adatok)) {
      datum d;
      d.ho = 11; d.nap = 29;
      cout << "Ma ünnepli szülinapját: "
           << endl;
```

Rekordok

Példák

Megoldás:

```
        vector<string> nevek =
           szulinapKeres(adatok, d);

        for (int i = 0; i < nevek.size(); i++)
           cout << nevek[i] << endl;
    }
    else {
         cout << "Az adatok beolvasása sikertelen!"
              << endl;
     }
     return 0;
}
```

Rekordok

Példák

Megoldás:

```
vector<string> szulinapKeres(vector<bejegyzes>
    adatok, datum d) {
    vector<string> nevek;
    for (int i = 0; i < adatok.size(); i++) {
        if (adatok[i].szulIdo.ho == d.ho &&
            adatok[i].szulIdo.nap == d.nap) {
            // ha a hónap és a nap stimmel
            nevek.push_back(adatok[i].nev);
        }
    }
    return nevek;
}
```

Típusok megvalósítása

Az adattípus

- A C++ lehetőséget biztosít saját típusok létrehozására
- *Adattípus = értékhalmaz + művelethalmaz*, ahol
  - *értékhalmaz*: a típusban szereplő adatok (változók) gyűjteménye, amit rekord segítségével valósítunk meg
  - *művelethalmaz*: az adatszerkezeten értelmezett műveletek, amelyeket alprogramokkal (függvényekkel) valósítunk meg
- Nyilván először mindig a rekordot deklaráljuk, majd ezt követően a rá alkalmazott függvényeket
- A függvények paraméterben megkapják a rekordot, így használhatják azt a függvény törzsében (cím szerinti paraméterátadásnál módosulhat a rekord értéke, és módosult értéket adjuk vissza a függvény lefutását követően)

Típusok megvalósítása

Példa

Feladat: Készítsük el a racionális szám típusát a beolvasás, kiírás, összeadás és szorzás műveletével. A főprogramban olvassunk be 5 racionális számot, és adjuk meg a szorzatukat és összegüket.

- készítsünk rekordot a racionális számnak, amely tartalmazza a számlálót és a nevezőt
- a beolvasás és kiírás függvénye kapja meg cím szerinti paraméterátadással az aktuális racionális számot, az összeadás és szorzás paraméterben kapjon két racionális számot, és a visszatérési értékük legyen az eredmény

Típusok megvalósítása

Példa

Megoldás:

```
…
// racionális szám típusa:
struct racionalis{ // racionális szám rekordja
   int szamlalo;
   int nevezo;
};

/* beolvasás
- bemenet: racionális szám (rac)
- kimenet: rac-ba beolvassuk az értékeket */
void beolvas(racionalis &rac){
   cin >> rac.szamlalo >> rac.nevezo;
}
```

Típusok megvalósítása

Példa

Megoldás:

```
/* kiírás
- bemenet: racionális szám (rac)
- kimenet: rac.szamlalo/rac.nevezo */
void kiir(racionalis &rac){
   cout << rac.szamlalo << "/" << rac.nevezo;
}
/* szorzás
- bemenet: két racionális szám (r1, r2)
- kimenet: r1 * r2 */
racionalis szoroz(racionalis r1, racionalis r2){
   …
   return eredmeny;
}
```

Típusok megvalósítása

Példa

Megoldás:

```
/* összeadás
- bemenet: két racionális szám (r1, r2)
- kimenet: r1 + r2 */
racionalis osszead(racionalis r1, racionalis r2){
   racionalis eredmeny;
   eredmeny.szamlalo = r1.szamlalo * r2.nevezo +
                       r2.szamlalo * r1.nevezo;
   eredmeny.nevezo = r1.nevezo * r2.nevezo;
   return eredmeny;
}
```

Típusok megvalósítása

Példa

Megoldás:

```
/* főprogram
   bemenet: 5 racionális szám (t)
   kimenet: t eleminek összege, szorzata */
int main(){
   racionalis t[5];
      // 5 elemű tömb racionális számokból
   cout << "Számok megadása: " << endl;
   for (int i = 0; i < 5; i++)
      beolvas(t[i]);
         // használjuk a beolvasó függvényt
      racionalis osszeg, szorzat;
      osszeg.szamlalo = 0; osszeg.nevezo = 1;
         // a 0 racionális szám a 0/1
```

Típusok megvalósítása

Példa

Megoldás:

```
        szorzat.szamlalo = szorzat.nevezo = 1;
           // 1/1 racionális szám
        for (int i = 0; i < 5; i++){
           osszeg = osszead(osszeg, t[i]);
           szorzat = szoroz(szorzat, t[i]);
        }
        cout << "Összeg: "; kiir(osszeg); cout << endl;
        cout << "Szorzat: "; kiir(szorzat);
        cout << endl;
        return 0;
    }
```

Típusok megvalósítása

Példa

- Megjegyzések:
  - a rekord szerkezete tetszőleges lehet
  - tetszőleges számú, tetszőleges viselkedésű függvényeket írhatunk egy típushoz
  - az egyszerű értékadás műveletét használhatjuk két rekord között, ekkor a jobbérték megfelelő értékei átmásolódnak a balérték megfelelő értékeibe, pl.:

    ```
    osszeg = osszead(osszeg, t[i]);
    ```

    ekkor az **`osszead`** által visszaadott **`racionalis`** rekord **`szamlalo`** adattagja bekerül az **`osszeg szamlalo`** mezőjébe, a **`nevezo`** adattag az **`osszeg nevezo`** adattagjába
  - az érték szerinti paraméterátadásnál is értékadás történik
  - de más operátort (**`+`**, **`*`**, **`<<`**, **`>>`**) nem használhattunk

Típusok megvalósítása

Operátorok saját típusokra

- Láthatjuk, hogy bármilyen feladatra készíthetünk függvényt egy típushoz, de ezek meghívása, kezelése nem egyezik meg a beépített típusok operátorműveleteihez
- Jó lenne, ha hasonló módon, operátorokkal tudnánk kezelni saját adattípusainkat, akárcsak a beépített típusok esetében
- A C++ lehetőséget biztosít, hogy saját típusainkhoz operátorokat készítsünk, és a működésüket teljes egészében mi határozzuk meg
  - az operátorokat ugyanúgy függvényként adhatjuk meg, mint más műveleteinket
  - a függvénynévként **`operator<jel>`**-t kell megadnunk, ahol a jel helyén az operátor jelét kell megadnunk (pl. **`operator+`**)

Típusok megvalósítása

Operátorok saját típusokra

- Az operátorok esetében a paraméterek száma és a visszatérési érték megléte kötött (hiszen csak az adott formában tudjuk használni őket), a paraméterek és a visszatérési érték típusa viszont tetszőlegesen szabályozható
  - a beépített operátorokat nem írhatjuk felül, ezért paraméterként saját típust kell megadnunk
  - amikor saját típushoz (rekordhoz) deklarálunk operátort, akkor a paraméterek között
- Az operátor deklarációját követően a meghívás teljesen ugyanúgy történik, mint a beépített operátorok esetében
  - ekkor ugyanazt a paraméterezést kell követnünk, amit a függvény definíciójában megadtunk (azaz a formális paraméternek megfelelő típust kell átadnunk)

Típusok megvalósítása

Definiálható operátorok

- A következő operátorokat deklarálhatjuk, illetve írhatjuk felül:
  - 1 paraméterrel:
    - matematikai: **`++, --`**
    - logikai: **`!`**
  - 2 paraméterrel (ekkor az első paraméter lesz a kifejezés balértéke, a második a jobbértéke):
    - matematikai: **`+, -, *, /, %`**
    - logikai: **`|| (OR), && (AND), ^ (XOR), & (bitenkénti és), | (bitenkénti vagy)`**
    - értékadás: **`+=, -=, *=, /=`**
    - összehasonlítás: **`<, >, <=, >=, !=, ==`**
    - adatátvitel: **`<<, >>`**

Típusok megvalósítása

Bináris művelet

- Vannak különleges operátorok (egyszerű értékadás, indexelés), amelyeket szintén felül lehet definiálni, de csak speciálisan (ezekről majd jövőre)
- Pl. az összeadás műveletét egy saját típusra így deklarálhatjuk:

```
<típus> operator+(<típus> <név1>, <típus> <név2>){
    … // függvénytörzs
    return <visszatérési érték>;
    // a típusnak megfelelő
}
```

- A deklarációt követően használhatjuk az operátort:

```
<típus> <vált1>, <vált2>, <vált3>;
…
<vált1> = <vált2> + <vált3>;
```

Típusok megvalósítása

Példa

Feladat: Valósítsuk meg az előbbi racionális típust úgy, hogy az összeadás és szorzás műveleteit operátorral fogalmazzuk meg.

- a két függvénynevet lecseréljük operátorra, a működésük marad az előbbi
- a főprogramban már operátorként használjuk őket

Megoldás:

```
racionalis operator+(racionalis r1,
                     racionalis r2){
   racionalis eredmeny;
   …
   return eredmeny;
}
```

Típusok megvalósítása

Példa

Megoldás:

```
racionalis operator*(racionalis r1,
                     racionalis r2){
   racionalis eredmeny;
   …
   return eredmeny;
}
…
for (int i = 0; i < 5; i++){
   osszeg = osszeg + t[i];
   szorzat = szorzat * t[i];
   // az általunk írt operátorok hívódnak meg
}
…
```

Típusok megvalósítása

Adatátviteli operátor

- A C++-ban különböző adatátviteli csatornákat tartunk nyilván, ilyen például a konzol (**`cin`**, **`cout`**) és a fájl (**`ifstream`**, **`ofstream`**)
- A csatornák hierarchikus felépítésűek, és minden csatorna esetén megkülönböztetjük a kimenő és bemenő csatornát
  - ez azt jelenti, hogy minden bemeneti csatornának van egy közös gyűjtőneve, az **`istream`**, és minden kimeneti csatornának az **`ostream`**
  - tehát például egy **`f`** fájl (ha **`f ifstream`** típusú) és a **`cin`** speciális esete az **`istream`**-nek, ezért tudjuk például mindkettővel használni a **`getline`** függvényt
  - az **`istream`** és az **`ostream`** megtalálhatóak az **`<iostream>`** fájlban és az **`std`** névtérben

Típusok megvalósítása

Adatátviteli operátor

- Adatátviteli operátorokat (**<<**, **>>**) is deklarálhatunk, ezáltal egy lépésben elintézhetjük az adatok bekérését és kiíratását
- Az adatátviteli operátorok (beolvasásra és kiírásra) speciális paraméterezést igényelnek:
  - paraméterben meg kell kapniuk az adatot (amely saját rekordunk), valamint az adatfolyamot, ahol a műveletet el kell végezniük
  - visszatérési érték szintén az adatfolyam
  - az adatfolyam lehet bemeneti (**`istream`**) és kimeneti (**`ostream`**), amik csak cím szerinti paraméterátadással adhatóak át (különben elveszne az adatfolyam a hívó függvényben)

Típusok megvalósítása

Adatátviteli operátor

- Az operátor belsejében a paraméterben átadott adatfolyamot használhatjuk a beolvasásra és kiírásra, valamint a rekordunk adattagjainak megfelelő műveleteket (**<<**, **>>** operátorokat), valamint egyéb műveleteket is alkalmazhatunk az adatfolyamra (beolvasásnál pl. **getline**-t használhatunk, kiírásnál mást is kiírhatunk…)
- Pl. a kiírás operátorát a következőképpen írhatjuk:

```
ostream& operator<<(ostream& s, <típus> <név>){
    // s lesz az adatfolyam, tehát azt használjuk
    s << <név>.<adattag1> << <név>.<adattag2>;
    // operátorok már az adattagok típusára
    s << "üzenet"; // kiírás s-re
    return s; // adatfolyam visszaadása
}
```

Típusok megvalósítása

Példa

Feladat: Írjuk át a racionális szám típusunkat úgy, hogy a beolvasás és kiírás a megszokott operátorokon keresztül történjen.

- a többi függvényt változatlanul hagyjuk, a főprogramban pedig meghívjuk ezeket az operátorokat

Megoldás:

```
istream& operator>>(istream& s, racionalis &rac){
   // s lesz az adatfolyam
   s >> rac.szamlalo >> rac.nevezo;
   // s-ről beolvassuk a két értéket
   return s; // s-t visszaadjuk
}
```

Típusok megvalósítása

Példa

Megoldás:

```
ostream& operator<<(ostream &s, racionalis &rac){
   s << rac.szamlalo << "/" << rac.nevezo;
   // s-re kiírjuk a megfelelő alakot
   return s;
}
…
for (int i = 0; i < 5; i++)
   cin >> t[i]; // saját operátorunk használata
…
cout << "Összeg:" << osszeg << endl;
cout << "Szorzat:" << szorzat << endl;
…
```

Típusok megvalósítása

Alprogramok túlterhelése

- Előfordulhat, hogy egy adott alprogramot nem csak az először megadott paraméterezéssel szeretnénk meghívni
  - pl. a racionális szám esetében jó lenne az összeadást egész számmal is értelmezni
- C++-ban lehetőségünk van ugyanolyan nevű, de más paraméterezésű alprogramok létrehozására, ezt az *alprogram túlterhelésének* nevezzük
  - ekkor mindegyik alprogram bekerül a programba, és a hívás pillanatában dönti el a program, hogy melyiket futtatja
  - ennek a kiválasztása az aktuális paraméterektől függ, azaz, hogy milyen típusú, és hány változóval hívunk meg egy adott alprogramot

Típusok megvalósítása

Alprogramok túlterhelése

- A paraméterezéssel egyetemben a visszatérési értéken is változtathatunk, de külön csak azon nem, mert a program nem tudná, hogy melyik függvényt hívja
- Pl.:

```
<típus1> <függvénynév> (<típus2> <név1>);
// eredeti függvény
<típus3> <függvénynév> (<típus4> <név1>);
// itt változtattunk a visszatérési érték és a
// paraméter típusán is
<típus3> <függvénynév> (<típus5> <név1>);
// itt a paraméter típusán változtattunk
<típus3> <függvénynév> (<típus5> <név1>,
                        <típus6> <név2>);
// itt a paraméterek számán változtattunk
```

Típusok megvalósítása

Alprogramok túlterhelése

- Készítsünk két összeadó függvényt, az egyik két egész számot, a másik három egész számot tud összeadni:

```
int osszead(int a, int b){
    return a+b;
}
int osszead(int a, int b, int c){
    return a+b+c;
}
…
int x, y, z;
…
z = osszead(x,y,z); // ez a 2. fv-t hívja meg
z = osszead(x,y); // ez az 1. fv-t hívja meg
```

Típusok megvalósítása

Alprogramok túlterhelése

- Készítsünk egy olyan függvényt, amely megmondja, hogy melyik nagyobb a két paraméterből, amely lehet két szám, vagy két szöveg (szöveg esetén a hossz alapján döntsünk):

```
bool nagyobb(int a, int b){
    return (a>b);
}
bool nagyobb(string a, string b){
    return (a.length() > b.length());
}
int u,v; string x, y; bool l;
…
l = nagyobb(x,y);     // a 2. fv-t hívja meg
l = nagyobb(u,v);     // az 1. fv-t hívja meg
```

Típusok megvalósítása

Alprogramok túlterhelése

- Ügyeljünk arra, hogy minden, az alprogramot meghívni kívánó változókombinációnak legyen megfelelően paraméterezett függvénye, de aminek nem kell, ne csináljunk

```
l = nagyobb(u,x);
// fordítási hiba: nincs olyan függvény, hogy
// nagyobb(int, string)
```

- Ha nem pont ugyanolyan típusú paraméterrel hívtunk meg egy függvényt, akkor az automatikus típuskonverzió megpróbálja igazítani az aktuális paraméter típusát, ha ez nem sikerül, fordítási hibát kapunk.
- Akkor is hibaüzenetet kapunk, ha nem sikerül egyértelműen eldönteni, hogy melyik függvényt hívja

Típusok megvalósítása

Példa

Feladat: Készítünk a racionális típusnak egésszel való összeadás, és szorzás műveleteket is. A főprogramban az eredményhez adjunk kettőt, a szorzatot pedig szorozzuk meg kettővel.

- mindkét esetben a második paraméter típusán változtatunk
- a főprogramban mindkét változat meghívásra kerül

Megoldás:

```
/* összeadás
   bemenet: egy racionális szám (r) és egy egész
            szám (e)
   kimenet: r + e */
racionalis operator+(racionalis r, int e){
  racionalis eredmeny;
```

Típusok megvalósítása

Példa

Megoldás:

```
    …
    return eredmeny;
}

/* szorzás
   bemenet: egy racionális szám (r) és egy egész
            szám (e)
   kimenet: r * e */
racionalis operator*(racionalis r, int e){
    racionalis eredmeny;
    …
    return eredmeny;
}
…
```

Típusok megvalósítása

Példa

Megoldás:

```
int main(){
  …
  for (int i = 0; i < 5; i++){
     osszeg = osszeg + t[i];
     szorzat = szorzat * t[i];
     // ezek az eredeti változatokat hívják meg
  }
  cout << "Összeg:" << osszeg + 2 << endl;
  cout << "Szorzat:" << szorzat * 2 << endl;
  // ezek az új változatokat hívják meg
  return 0;
}
```

Típusok megvalósítása

Konstruktor műveletek

- Általában a rekordok létrehozása, és értékeinek beállítása elég körülményes, hiszen minden adattagjának külön kell értéket adni
- Ez egyszerűsíthető, ha készünk egy beállító függvényt, amely bizonyos paraméterek alapján beállítja a rekord tagjait
  - nem kötelező annyi paramétert adnunk, ahány tag van, bizonyos változóknak adhatunk alapértelmezett értéket
  - ellenben ezt a műveletet is meg kell hívnunk valahol manuálisan a kódban
- Azonban van egy olyan művelet, amely dedikáltan akkor hívódik meg, amikor létrehozunk egy változót a rekordból, ez művelet a *konstruktor*

Típusok megvalósítása

Konstruktor műveletek

- A konstruktor olyan függvény, amely:
  - a rekord belsejében definiálható, a rekord változói mellett
  - tetszőlegesen felparaméterezhető és túlterhelhető
  - arra szolgál, hogy a rekord változóit kezdő értékekkel lássa el (úgymond *inicializál*ja a tagokat)
  - mindig automatikusan meghívódik, amikor létrehozunk egy új példányt a rekordból, paraméteres esetben ekkor átadhatunk neki értékeket
  - akkor is létezik, amikor nem írjuk meg, csak ekkor nem végez semmilyen inicializáló tevékenységet
  - neve megegyezik a rekorddal, és mivel létrehozza a példányt, a visszatérési értéke is, amit külön nem írunk le

Típusok megvalósítása

Konstruktor műveletek

```
struct <rekordnév>{
   <típus> <tag>;

   <rekordnév>() { <tag> = <érték>; }
   <rekordnév>(<típus> <változó>) {
      <tag> = <változó>;
   }
};

<rekordnév> <változónév>;
// ekkor lefut a 0 paraméteres konstruktor

<rekordnév> <változónév>(<érték>);
// ekkor lefut az 1 paraméteres konstruktor,
// amennyiben a típusok egyeznek
```

Típusok megvalósítása

Példa

Feladat: Egészítsük ki a racionális szám típusát két konstruktorral, egy 0 paraméteressel, amely a 0 racionális számot állítja elő, illetve egy 1 paraméteressel, amely egy egész szám alapján hozza létre a racionálist.

Megoldás:

```
struct racionalis{ // racionális szám rekordja
   int szamlalo; int nevezo;
   // konstruktorok:
   racionalis() { szamlalo = 0; nevezo = 1; }
   racionalis(int szam) {
      szamlalo = szam; nevezo = 1;
   }
};
```

Típusok megvalósítása

Példa

Megoldás:

```
…
racionalis osszeg;
   // 0 paraméteres konstruktor hívás
racionalis szorzat(1);
   // 1 paraméteres konstruktor hívás
for (int i = 0; i < 5; i++){
   osszeg = osszeg + t[i];
   szorzat = szorzat * t[i];
}
cout << "Összeg: " << osszeg + 2 << endl;
cout << "Szorzat: " << szorzat * 2 << endl;
…
```

IV. Rekordok:

1. Adott egy szövegfájl, ami egy recept hozzávalóit tartalmazza. A fájl minden sora egy számmal kezdődik, ami egy összetevőből szükséges mennyiség, majd vesszővel elválasztva tőle az összetevő neve jön.
   a) Add meg azt az összetevőt, amiből a legtöbb, és amiből a legkevesebb kell.
   b) Add meg, hány olyan összetevő van, amiből kevesebb, mint egy egységnyi kell.
   c) Add meg egy tetszőleges összetevőről, hogy mennyi kell belőle.

Alprogramok

Feladatok

IV. Rekordok:

4. Adott egy szövegfájl, ami egy hónap minden napjának hőmérsékleti adatait tartalmazza: minden sorban három szám van, egy napon mért reggeli, déli és esti hőmérsékletet.
   b) Add meg a legalacsonyabb napi középhőmérsékletet (és azt is, hogy hányadik napon volt).
   c) Add meg, hány reggel volt fagy.
6. Adott egy szövegfájl, ami egy áruház raktárkészletét tartalmazza. Minden sora egy árucikk adatait tartalmazza vesszővel elválasztva: az áru nevét,egységárát, a raktáron található mennyiséget, a mennyiségi egység nevét, a raktárban elfoglalt helyének a kódját, valamint egy minimális mennyiséget, amit szeretnénk a raktárban tartani.

IV. Rekordok:

6. b) (*) Add meg, hogy melyik áruból van a legértékesebb készlet.
   d) (*) Listázd ki azokat az árukat, amikből nincs meg az elvárt minimális mennyiség, add meg, mennyit kell még beszerezni belőlük, és hova kell őket tenni a raktárban.
   e) (*) Add meg, hogy mennyibe kerül a készletet kiegészítése úgy, hogy mindenből meglegyen az elvárt minimális mennyiség.
   f) (*) Add meg egy áruról a neve alapján, hogy mennyi van belőle, mekkora értékű a készlet, és hol található a raktárban.