PROGRAMOZÁS

Felhasználói típus megvalósítása

dr. Gregorics Tibor
egyetemi docens
ELTE Informatikai Kar
http://people.inf.elte.hu/gt/prog

Feladat

Hány különböző szám található abban a tömbben, amelynek elemei 0 és 99 közé eső egész számok lehetnek?

A tömb elemei egy szövegfájlban vannak: az első sorban az elemek száma, utána soronként a tömb egy-egy eleme.

Megoldási terv

Bemenő adat: $x : \{0..99\}^n$
Eredmény adat: $s : \mathbb{N}$

Vezessünk be egy olyan halmazt, amelynek elemei 0 és 99 közé eső egész számok lehetnek

Dobáljuk bele a halmazba a tömb elemeit: $\bigcup_{i=1}^{n} \{x[i]\}$

Végül nézzük meg a halmaz számosságát!

Programterv

$A = (x:\{0..99\}^n , \ s:\mathbb{N})$
$Ef = (x = x')$

$Uf = (x = x' \wedge s = \left| \bigcup_{i=1}^{n} \{x[i]\} \right|)$

$Uf = (x = x' \wedge h = \bigcup_{i=1}^{n} \{x[i]\} \wedge s = |h|) \quad h : 2^{\{0..99\}}$

```
h:=∅
    i = 1 .. n
        h:=h∪{x[i]}
s:=|h|
```

Fő program

```
#include <iostream>
#include <fstream>
#include "halmaz.h"
using namespace std;

int main()
{
   vector<int> x;
   read(x);

   Halmaz h;

   for(int i=0;i<(int)x.size();++i)
   {
      h.be(x[i]);
   }

   cout << "A megadott tömbben " << h.darab()
        << " különböző szám található!" << endl;
   return 0;
}
```

main.cpp

Típus

	Típus-specifikáció	
típus-értékek	T	$F_1 \ \ldots \ F_n$ állapotterükben: T
reprezentáció	R $repr: R \to T$ $inv: R \to \mathbb{L}$	$S_1 \ \ldots \ S_m$ állapotterükben: R
	Típus-implementáció	

műveletek

implementáció

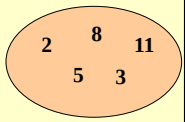
Halmaz típus-specifikációja

Típus-specifikáció

típus-értékek

$2^{\{0..99\}}$

pl: h



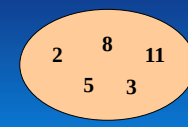
műveletek

Létrehoz egy üres halmazt
$h:=\varnothing$
Betesz egy elemet a halmazba
$h:=h\cup\{e\}$ $e:\{0..99\}$
Megadja a halmaz elemszámát
$s:=|h|$ $s:\mathbb{N}$

Halmaz típus reprezentációja

típus-értékek

$2^{\{0..99\}}$

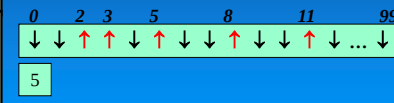


reprezentáció

$v : \mathbb{L}^{0..99}$

$db : \mathbb{N}$

$db = \sum_{i=0}^{99} v[i]$



5

Halmaz típus műveleteinek implementációja

$h:=\varnothing$

$k=0 .. 99$
$v[k]:=hamis$
$db:=0$

műveletek

$h:=\varnothing$,
$h:=h\cup\{e\}$
$s:=|h|$

implementáció

műveletek programjai

$h:= h \cup\{e\}$

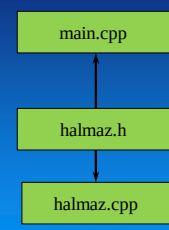
$v[e]$

SKIP	$v[e]:=igaz$ $db:=db+1$

$s :=|h|$

$s:=db$

Típus-orientált modul szerkezet



Halmaz típus osztálya

A **`Halmaz h`** hatására jön létre a **h** objektum, amelynek két adata van: **v** és **db**.
Ezekre az adatokra hajtódnak végre a metódusok.

```
#ifndef HALMAZ_H
#define HALMAZ_H

class Halmaz{
   public:
      Halmaz();
      void be(int e);
      int  darab() const;
   private:
      bool v[100];
      int  db;
};

#endif
```

konstans metódus

A privát adattagokra nem lehet **`h.v`** vagy **`h.db`**-ként közvetlenül hivatkozni, de a publikus metódusokat meghívhatjuk a **h**-ra (pl. **`h.be(5)`**).

halmaz.h

Halmaz metódusok

```
#include "halmaz.h"
using namespace std;

Halmaz::Halmaz(){
   for (int i=0; i<=99; ++i) v[i] = false;
   db = 0;
}
void Halmaz::be(int e){
   if (e<0 || e>99) return;
   if (!v[e]){
      v[e] = true;
      db++;
   }
}

int Halmaz::darab() const{
   return db;
}
```

A **`be()`** metódusnak két paramétere van (ez jól látszik majd a híváskor: **`h.be(5)`**)
- közönséges paraméter: **5**, amely az **`int e`** változóba kerül
- alapértelmezett paraméter: **h** amelynek adattagjaira (**v** , **db**) közvetlenül hivatkozunk.

halmaz.cpp

Feladat

Egy 0 és 99 közé eső egészeket tartalmazó tömbben melyik a leggyakoribb szám!

A tömb elemei egy szövegfájlban vannak: az első sorban az elemek száma, utána soronként a tömb egy-egy eleme.

Megoldási terv

Bemenő adat: $x : \{0..99\}^n$
Eredmény adat: $e : \{0..99\}$

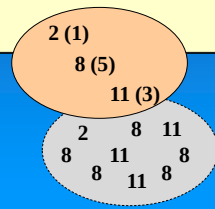
Vezessünk be egy olyan speciális halmazt (zsákot), amelynek elemei 0 és 99 közé eső egész számok lehetnek, és mindegyik eleméhez tartozik egy előfordulás-szám is.

Dobáljuk bele a zsákba a tömb elemeit: $h := \bigcup_{i=1}^{n} \{x[i]\}$

Keressük meg a zsák leggyakoribb elemét!

Programterv

$A = (x:\{0..99\}^n , e:\{0..99\})$
$Ef = (x = x' \wedge n>0)$
$Uf = (x = x' \wedge h = \bigcup_{i=1}^{n} \{x[i]\} \wedge e = max\_elem(h))$
$h : Zsák(0..99)$

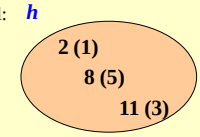



```
h:=∅
    i = 1 .. n
        h:=h∪{x[i]}
e := max_elem(h)
```

Zsák típus specifikációja

Típus-specifikáció

típusértékek: *Zsák(0..99)*
pl: ***h***

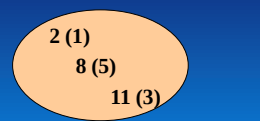


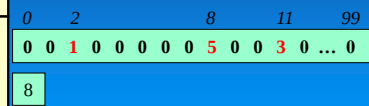
műveletek:
Létrehoz egy üres zsákot
h:=∅
Betesz egy elemet a zsákba
h:=h∪{e} *e:{0..99}*
Megadja a zsák leggyakoribb elemét
e:=max_elem(h)

Zsák típus reprezentációja

típus-értékek: *Zsák(0..99)*
$2^{\{0..99\}\times\mathbb{N}}$

reprezentáció:
$v : \mathbb{N}^{0..99}$
$ind : \mathbb{N}$
$v[ind] = \mathbf{MAX}_{i=0}^{99} v[i]$





Zsák típus implementációja

h:=∅
```
k=0 .. 99
    v[k]:=0
ind:=0
```

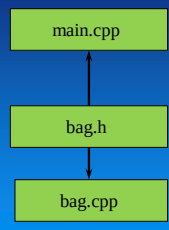
h:= h ∪{e}
```
v[e]:=v[e]+1
v[e]>v[ind]
ind:=e        SKIP
```

e:= max_elem(h)
```
v[ind]≠0
e:=ind        SKIP
```

műveletek: *h:=∅, h:=h∪{e}, e:=max_elem(h)*

implementáció: műveletek programjai

Típus-orientált modul szerkezet



Fő program

```
#include <iostream>
#include <fstream>
#include "bag.h"

using namespace std;

int main()
{
   vector<int> x;
   Bag h;

   read(x);
   for(int i=0;i<(int)x.size();++i){
      h.put(x[i]);
   }

   cout << "A leggyakoribb elem: " << h.max_elem();

   return 0;
}
```

main.cpp

Zsák típus osztálya

```
#ifndef BAG_H
#define BAG_H

class Bag{
   public:
      enum Exceptions{WrongInput, EmptyBag};

      Bag();
      void put(int e);
      int max_elem() const;

   private:
      static const int n = 100;
      int v[n];
      int ind;
};

#endif
```

Hibaesetek: Kivételek

Hátha módosul a terv

bag.h

Zsák metódusok

```
#include "bag.h"
using namespace std;

Bag::Bag()
{
   for (int k=0; k<n; ++k) v[k] = 0;
   if (n>0) ind = 0;
}
void Bag::put(int e)
{
   if (e<0 || e>n-1) throw WrongInput;
   ++v[e];
   if (v[e]> v[ind]){ ind = e; }
}
int Bag::max_elem() const
{
   if (n<1 || v[ind] == 0) throw EmptyBag;
   return ind;
}
```

Kivétel dobása

Kivétel dobása

bag.cpp

Fő program másképpen

```
vector<int> x;
Bag h;

Read(x);
for(int i=0;i<(int)x.size();++i){
   try{
      h.put(x[i]);
   }catch(Bag::Exceptions ex){
      if(ex==Bag::WrongInput){
         cout << "Hibás adat a tömbben! ";
      }
   }
}

try{
   cout << "A leggyakoribb elem: " << h.max_elem();
}catch(Bag::Exceptions ex){
   if(ex==Bag::EmptyBag){
      cout << "Üres a zsák! ";
   }
}
```

Kivétel figyelése

Kivétel elkapása és kezelése

Jó lenne többet tudni a hibáról

main.cpp

A kivételobjektum bármi lehet

```
class Bag{
public:
   enum Error{WrongInput, EmptyBag};
   struct Exception {
      Error message;
      int value;
   };
   ...
```

```
void Bag::put(int e)
{
   if (e<0 || e>n-1){
      Exception ex;
      ex.message = WrongInput;
      ex.value = e;
      throw ex;
   }
   ++v[e];
   if (v[e]> v[max]){ max = e; }
}
```

```
try{
   ...
   h.put(x[i]);
   ...
}catch(Bag::Exception ex){
      if(ex.message == Bag::WrongInput)
            cout << "A tömbbeli " << ex.value
           << " nem 0 és 99 közé eső szám!";
}
```

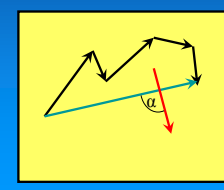
bag.cpp

Kivételkezelés

- Hibás, nem várt működés lekezelési jogának átadása.
  - Hibás működést okozó kódrészben (függvényben) csak észleljük és jelezzük a hibát, de nem reagálunk rá.
  - Hibakezelést abban a programrészben kódoljuk, ahonnan hibás működést okozó programrészt (függvényt) meghívtuk.
- Hibakezelés kiemelése a strukturált szerkezetből
  - a hiba kezelést nem sok, egymásba ágyazott elágazással oldjuk meg, hanem úgy, hogy a hiba észlelésénél kiugrunk a normális vezérlésből – kivételdobás (throw)
  - amit meghatározott helyen (try-catch) elkapunk és lekezeljük.

Feladat

Tekintsünk néhány síkvektort, és vizsgáljuk meg, hogy az összegük merőleges-e egy újabb adott síkvektorra.



Megoldási terv

Bemenő adatok: $t : Vector2D^n$
$p : Vector2D$
Eredmény adat: $l : \mathbb{L}$

1. Összeadjuk a *t* tömbbeli vektorokat.
   $z :=$ ***összegzés***$(i=1..n: t[i])$ $z : Vector2D$

2. Nulla-e a *v* és *s* vektorok skaláris szorzata?
   $l :=$ ***skalár***$(p,z) = 0$

Programterv

$A = (t : Vector2D^n , p : Vector2D, \quad l : \mathbb{L} \quad)$
$Ef = (t = t' \wedge p = p')$

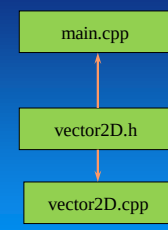
$$Uf = (Ef \wedge z{:}Vector2D \wedge \; z = \sum_{i=1}^{n} t[i] \wedge l = skalár(p,z)=0)$$

```
z:=nullvektor
    i = 1 .. n
        z:=z+t[i]
l := skalár(p,z) = 0
```

Síkvektorok típusa

típus-értékek	Típus-specifikáció	műveletek
síkvektorok `Vector2D`	*hozzáadás* (add) *v:=v+u* *(v += u;)* *létrehozás* *v := nullvector*	*skalárszorzás* (scalar) *d:= v1*v2* $d : \mathbb{R}$
reprezentáció	**Típus-implementáció**	**implementáció**
$x , y : \mathbb{R}$	*v.x, v.y:= v.x+u.x, v.y+u.y* *(v.x += u.x; v.y += u.y;)* *v.x, v.y := 0, 0*	*d := v1.x*v2.x+v1.y*v2.y* $d : \mathbb{R}$

Típus-orientált modul szerkezet



Fő program

```
int main()
{
      vector<Vector2D> t;
      fill(t);

      Vector2D z = sum(t);

      double x,y;
      cout << "x = " ; cin >> x;
      cout << "y = " ; cin >> y;
      Vector2D p(x,y);

      if( p.scalar(z) == 0.0)
            cout << "A vektor merőleges az összegre";
      else  cout << "A vektor nem merőleges az összegre";
}
```

t egy síkvektorokat tartalmazó tömb

feltöltjük a *t* tömböt síkvektorokkal

z a *t* tömb síkvektorainak összege

p egy *x*,*y* koordinátájú síkvektor Vector2D konstruktora

p és *z* síkvektorok skaláris szorzata

main.cpp

Fő program folytatása

```
void fill(vector<Vector2D> &t)
{
   int n;
   cout << "Vektorok száma: " ;
   cin >> n;
   t.resize(n);

   double x,y;
   for(int i=0; i<n; ++i){
      cout << "x = " ; cin >> x;
      cout << "y = " ; cin >> y;
      t[i].set_x(x);
      t[i].set_y(y);
   }
}
Vector2D sum(const vector<Vector2D> &t)
{
      Vector2D z;
      for(int i=0; i<t.size(); ++i) z.add(t[i]);
      return z;
}
```

Vector2D „üres” konstruktora

vektor helyzetének (koordinátáinak) megváltoztatása

hozzáadás művelete

main.cpp

Vector2D osztály 1. verzió

```
#ifndef VECTOR2D_H
#define VECTOR2D_H
class Vector2D{
   private:
      double x,y;
   public:
      Vector2D(): x(0.0), y(0.0){}
      Vector2D(double a, double b): x(a), y(b) {}

      void set_x(double r ){ x = r; }
      void set_y(double r ){ y = r; }

      void add(const Vector2D &u){
         x+=u.x; y+=u.y;
      }

      double scalar(const Vector2D& v){
         return x*v.x + y*v.y;
      }
};
#endif
```

túlterhelés

adattagok inicializálása

hívás:
`d = v1.scalar(v2)`

Szebb lenne:
`d = scalar(v1,v2)`
vagy
`d = v1 * v2`

vector2D.h

Változatok a Scalar() függvényre

```
class Vector2D{
   ...
double scalar(const Vector2D& v) { return x*v.x + y*v.y;  }

friend double scalar(const Vector2D& v1,const Vector2D& v2);

double operator*(const Vector2D& v2)
{return x*v2.x + y*v2.y;}

friend double operator*(const Vector2D& v1,
                        const Vector2D& v2);
};

double scalar(const Vector2D& v1, const Vector2D& v2)
{ return v1.x*v2.x + v1.y*v2.y; }

double operator*(const Vector2D& v1, const Vector2D& v2)
{ return v1.x*v2.x + v1.y*v2.y; }
```

hívása:
`d = v1.scalar(v2)`

hívása:
`d = v1 * v2`

hívása:
`d = scalar(v1,v2)`

hívása:
`d = v1 * v2`

Vector2D osztály 2. verzió

```
#ifndef VECTOR2D_H
#define VECTOR2D_H
class Vector2D{
   private:
      double x,y;
   public:
      Vector2D() {x = 0.0; y = 0.0;}
      Vector2D(double a, double b): x(a), y(b) {}
      void set_x(double r ){ x = r; }
      void set_y(double r ){ y = r; }
      ...
      Vector2D operator+=(const Vector2D &u);
   friend double operator*(
            const Vector2D& v1, const Vector2D& v2);
};
#endif
```

`s.add(v)`
helyett
`s += v`

vector2D.h

```
#include "vector.h"
Vector2D Vector2D::operator+=(const Vector2D &u){
   x+=u.x; y+=u.y; return *this;
}
double operator*(const Vector2D& v1 , const Vector2D& v2){
   return v1.x*v2.x + v1.y*v2.y;
}
```

vector2D.cpp

Fő program mégegyszer

```
int main()
{
   ...

      if( p*z == 0.0)
            cout << "A vektor merőleges az összegre" ;
      else  cout << "A vektor nem merőleges az összegre";
}

void fill(vector<Vector2D> &t)
{
   ...
}
Vector2D sum(const vector<Vector2D> &t)
{
      Vector2D z;
      for(int i=0; i<t.size(); ++i) z+=t[i])
      return z;
}
```

v és *s* vektorok skaláris szorzata

s vektorhoz való hozzáadás

main.cpp


VÉGE