Soros gyakorlat
●
sor folytonos ábrázolással

2015-16.

1. kérdés: Mi a sor?

Válaszok

a) szűkebb („hagyományos”) válasz: egy **típus**, amelyre az alábbi jellegzetes műveletek alkalmazhatók:

`Sorba` – a bent lévő adatok „mögé” rakja az új értéket

`Sorból` – a legelső (azaz a legrégebben betett) elemet adja eredményül, és eközben a sorból ki is veszi

b) egy **típuskonstrukció**, amely egy típussal (és esetleg mérettel) paraméterezhető, és az alábbi műveletgarnitúrával kezelhető:

```
Exportmodul TSor(Típus TElem):
  Eljárás
    Üres[legyen](Változó s:TSor)
  Függvény
    ÜresE(Konstans s:TSor):Logikai
  Eljárás
    Sorba(Változó s:TSor, Konstans e:TElem)
  Függvény
    Sorból(Változó s:TSor):TElem
    Első(Változó s:TSor):TElem
    Hossz(Konstans s:TSor):Egész
    TeleE(Konstans s:TSor):Logikai
    HibásE(Változó s:TSor):Logikai
Modul vége.
```

A `Hossz` függvényt, ami a verembeli `Mélység` párja, már az alap megoldásnál a sor műveletei közé soroljuk.

2. kérdés: Hogyan ábrázolható?

Válasz

Sokféleképpen, de mi az ún. **folytonos ábrázolás**t választjuk.

```
Modul TSor(Típus TElem):
  Reprezentáció
    Konstans
      MaxHossz:Egész(???)
       [megadja, h. maximálisan hány eleme lehet egy sornak]
    Típus
      TSorElemek=Tömb(1..MaxHossz:TElem)
    Változó
      se:TSorElemek
      eleje,vége,hossza:Egész
      hiba:Logikai
     [A modul „globális” adatai együttesen jelentik a sor ábrázolását.
      Bonyolultabban, de szokványosabban ezt így is írhattuk volna:
      TSor=Rekord(
             ve:TSorElemek
             eleje,vége,hossza:Egész
             hiba:Logikai)
      Jegyezzük meg tehát a fenti jelölés lényegét, jelentését!]
  Implementáció
    Eljárás Üres[legyen](Változó s:TSor):
      [„alapállapotba” hozza a sort:
       Ef: -
       Uf: eleje=1 ∧ vége=1 ∧ hossza=0 ∧ hiba=Hamis]
      eleje:=1; vége:=1; hossza:=0; hiba:=Hamis
    Eljárás vége.
    Függvény ÜresE(Konstans s:TSor):Logikai
      [0 elemű-e a sor:
       Ef: -
       Uf: ÜresE(s)=hossza=0]
      ÜresE:=hossza=0
    Függvény vége.
    Eljárás Sorba(Változó s:TSor, Konstans e:TElem):
      [az s sor végére teszi az e értéket;
       ha nem fér bele, hiba-állapotba kerül:
       Ef: -
       Uf: hossza=MaxHossz → hiba ∧
           hossza<MaxHossz → se'eleje.©.vége=seeleje.©.vége ∧ se'vége'=e ∧
                             vége'=vége⊕1 ∧ hossza'=hossza+1]
    …
```

Bevezettünk két jelölést, **a ciklikus növelés operátor**ét (⊕1) és a **ciklikus intervallum**ét (.©.):

```
Operátor cInc(Konstans x:Egész):Egész
  Másként x⊕1
  Ha x=MaxHossz akkor cInc:=1 különben cInc:=x+1
Operátor vége.

eleje.©.vége=eleje..vége, ha eleje≤vége
eleje.©.vége=1..vége ∪ eleje..MaxHossz, ha eleje>vége
```

```
    Függvény Sorból(Változó s:TSor):TElem
      [az s sorból kiveszi az első elemet, és az e-be teszi;
       ha üres a sor, hiba-állapotba kerül]
       Uf: hossza=0 → hiba ∧
           hossza>0 → Sorból(s)=se_eleje ∧ se'_eleje'.©.vége=se_eleje'.©.vége ∧
                      eleje'=eleje⊕1 ∧ hossza'=hossza-1]
      …
    Függvény Első(Változó s:TSor):TElem
      [az s sor első elemét az e-be másolja;
       ha üres a sor, hiba-állapotba kerül]
      …
    Függvény Hossz(Konstans s:TSor):Egész
      [hány elemű a sor]
      …
    Függvény TeleE(Konstans s:TSor):Logikai
      [fér-e még a sorba érték]
      …
    Függvény HibásE(Változó s:TSor):Logikai
     [hibás állapotban van-e a sor]
      …
  Inicializálás
    [ez történik minden ilyen típusú adattal a létrejöttekor,
     automatikusan:]
    eleje:=1; vége:=1; hossza:=0; hiba:=Hamis
Modul vége.
```

1. feladat

A műveletekből hiányzó **specifikációk**at pótolja! Segítségül meghagytam az informális leírását.

Majd ennek felhasználásával **algoritmizálja** az egyes sorműveleteket! De ügyeljen arra, hogy az egyes műveletek legyenek egymástól maximálisan függetlenek, azaz még ha nagy is a kísértés, akkor se hívjon más műveletet az egyes műveletek implementációjában! Ez az elv a biztonságosságot szolgálja! Gondolkodjon el azon, hogy miért!

Mindenekelőtt alkossuk meg a **sor osztály**át, felhasználva a fenti modul-leírást!

```
Konstans
  MaxHossz:…
Típus
  TSorElemek=…
  OSor=Osztály(
        [értékhalmaz reprezentációja:]
         se:TSorElemek
         eleje,vége,hossza:Egész
         hiba:Logikai
        [típushoz asszociált műveletek:]
         Eljárás  Üres
         Függvény ÜresE:Logikai
         Eljárás  Sorba(Konstans e:TElem)
         Függvény Sorból:TElem
         Függvény Első:TElem
         Függvény Hossz:Egész
         Függvény TeleE:Logikai
         Függvény HibásE:Logikai
       )
```

3. Kódolás

2. feladat

Alkossuk meg a **Pascal** párját a fenti **osztály**nak! Fejezze be az alább megkezdett Pascal **osztály**-definíciót!

```
Const
  MaxHossz=10;//persze a 10 helyett más szám is lehet
Type
  TSorElemek=Array [1..MaxHossz] of TElem;
  OSor=Object
         Private          {értékhalmaz reprezentációja:}
           se:TSorElemek;
           eleje,vege,hossza:Integer;
           hiba:Boolean;
         Public           {típushoz asszociált műveletek:}
           Procedure Ures;
           Function  UresE:Boolean;
           Procedure Sorba(Const e:TElem);
           Function  Sorbol:TElem;
           Function  Elso:TElem;
           Function  Hossz:Integer;
           Function  TeleE:Boolean;
           Function  HibasE:Boolean;
       End;
  Procedure cInc(Var x:Integer);
  Begin
    If x=MaxHossz then x:=1 else Inc(x);
  End;
  Procedure OSor.Ures;
  Begin
    eleje:=1; vege:=1; hossza:=0; hiba:=False;
  End;
  Function  OSor.UresE:Boolean;
  Begin
    UresE:=hossza=0;
  End;
  Procedure OSor.Sorba(Const e:TElem);
  Begin
    If hossza<MaxHossz then
    Begin
      se[vege]:=e; cInc(vege); Inc(hossza);
    End
      else
    Begin
      hiba:=True;
    End;
  End;
  Function  OSor.Sorbol:TElem;
  …
  Function  OSor.Elso:TElem;
  …
  Function  OSor.Hossz:Integer;
  …
  Function  OSor.TeleE:Boolean;
  …
  Function  OSor.HibasE:Boolean;
  Begin
    HibasE:=hiba; hiba:=False;
  End;
```

Megjegyzés:

Ez nem lesz teljesen ekvivalens az algoritmikus nyelvű változattal, hiszen a Pascal programban az **inicializálás** nem megy végbe automatikusan a típusdeklarációk alkalmával. Ezért egy alábbi szerű deklarációkor:

```
Var
  s:OSor;
```

az `s` sor értéke nem definiált marad a helyfoglalás után; így használat előtt a programozónak üresre kell állítania az `s`-et az

```
s.Ures;
```

művelettel.

Újrafelhasználhatóság

Meggondoljuk, miként tehetnék a fenti kódot a Pascalban minél könnyebben újrafelhasználhatóvá.

A Pascalbeli kódolás több úton történhet: 1) a modul mintájára egy **unit**tal, 2) egy beilleszthető kóddarabban (**include**-dal), ill. 3) típussal **paraméterezett osztály**al (generic/specialize fogalompárra építve). Az 1) előnye, hogy önálló fordítási egység, azaz előre lefordítható, elegendő tehát a sort használó főprogramhoz hozzáfordítani. Nagy baj viszont, hogy **nem paraméterezhető** az elemtípussal, így ahányféle elemtípusú sorra van szüksége a főprogramnak, annyi unitot kell készíteni hozzá. A 2) megengedi a módosítás nélküli újrafelhasználást, de csak akkor, ha a főprogram **egyetlen fajta sor**t igényel. (Miért csak egy típus esetén alkalmazható?) A 3) speciális szintaxisa teszi kissé idegenné számunkra.

A unitos megoldás

Megelégszünk egy olyan megoldással, amelyben az **elemtípus** a legáltalánosabb, azaz a **Szöveg** lesz. Így némi konverzió árán bármilyen típusú sor kezelésére alkalmassá tehető.

A unit exporttal (és –Pascal „logika” szerint– az ábrázolással) foglalkozó `Interface` része az alábbi:

```
Unit StrSor_Unit;
 Interface
   Type
     TElem=String;
   Const
     MaxHossz=10;//persze a 10 helyett más szám is lehet
   Type
     TSorElemek=Array [1..MaxHossz] of TElem;
     OSor=Object
           Private
             se:TSorElemek;
             eleje,vege,hossza:Integer;
             hiba:Boolean;
           Public
             Procedure Ures;
             Function  UresE:Boolean;
             Procedure Sorba(Const e:TElem);
             Function  Sorbol:TElem;
             Function  Elso:TElem;
             Function  Hossz:Integer;
             Function  TeleE:Boolean;
             Function  HibasE:Boolean;
         End;
… // és a unit folytatása
```

Az inklúdos megoldás

E megoldás arra épít, hogy 1) külön fájlban (`TSor.inc`) lesz a típuskonstrukció teljes definíciója, 2) a `TElem` típusazonosítóra vonatkozó típusdefiníció után inklúdáljuk a főprogramba. Valahogy így:

```
Program String_Soros_Program;
  Uses
    Crt, …;
  Type
    TElem=… típusdefiníció …;
  {$i TSor.inc - a TElem elemtípusú sor megvalósítása}
  Var
    s:TSor;
  …
Begin
  …
End.
```

A `TSor.inc` fájl tartalmának vázlata:

```
  Const
    MaxHossz=10;//persze a 10 helyett más szám is lehet
  Type
    TSorElemek=Array [1..MaxHossz] of TElem;
    TSor=Record
           se:TSorElemek;
           eleje,vege,hossza:Integer;
           hiba:Boolean;
         End;
    Procedure Ures;
    Begin
      …
    End;
    Function  UresE:Boolean;
    Begin
      …
    End;
    Procedure Sorba(Const e:TElem);
    Begin
      …
    End;
    … // és a többi művelet implementációja
```

Megjegyzés:

Persze a unitban alkalmazott ábrázolás (és a ráépülő implementáció) éppen olyan jó, mint az itt látható „hagyományos” rekordos ábrázolás… A lényeg, hogy e fájl tartalmazza mind az ábrázolást, mind a műveletek implementációját, de ne tartalmazza a „paraméterként működő” `TElem` típus definiálását.

A típussal paraméterezhető osztály megvalósítása_____

A paraméteres sor **osztály**t most is egy unitba csomagoljuk. Ennek `Interface` része az alábbi:

```
Unit OSor_Sablon_Unit;
 {$mode objfpc}{$h+}
 Interface
   Const
     MaxHossz=10;//persze a 10 helyett más szám is állhat
   Type
     Generic OSor<TElem>=Class(TObject)
       Type
         TSorElemek=Array [1..MaxHossz] of TElem;
       Private
         se:TSorElemek;
         eleje,vege,hossza:Integer;
         hiba:Boolean;
         Procedure cInc(Var x:Integer);
       Public
         Constructor Create;
         Procedure Ures;
         Function  UresE:Boolean;
         Procedure Sorba(Const e:TElem);
         Function  Sorbol:TElem;
         Function  Elso:TElem;
         Function  Hossz:Integer;
         Function  TeleE:Boolean;
         Function  HibasE:Boolean;
     End;
… // és a unit folytatása
```

Megjegyzések:

Váratlan „kötelezettség”, hogy a `class`-alapú osztály definícióba a **`cInc`** eljárást be kell illeszteni. Így ez egy nem publikált (**`private`**) metódusa lesz az osztálynak.

Szétválasztottuk a (`Create` néven) és az üressé tétel (`Ures` néven) funkciókat. Indokok: 1) a létrehozás ennél a változatnál nem nélkülözhető, ui. ez által rendelődik futás közben az adott példányhoz megfelelő méretű memóriaterület; 2) az üressé tételre szükség lehet egy már létrehozott sor esetében is. A létrehozás természetesen gondoskodik az üres kezdőállapotról. is.

A főprogrambeli felhasználása, paraméterezése a sor osztálynak:

```
Program Soros_Program;
  Uses
    …OSor_Sablon_Unit…
  Type
    OSor_Int=Specialize OSor<Integer>;
    OSor_Str=Specialize OSor<String>;
  Var
    sE:OSor_Int;
    sS:OSor_Str;
  …
  sE:=OSor_Int.Create; sS:=OSor_Str.Create;
  …
```

Megjegyzés:

Ne feledje, itt is igaz a korábbi, `object`-alapú reprezentációra vonatkozó megjegyzés: a Pascal programban az **inicializálás** nem megy végbe automatikusan a típusdeklarációk alkalmával. Sőt –`class` esetén– a példánynak még helyet is kell foglalni: használat előtt a deklarált osztály-példányt az osztályhoz tartozó `Create`-tel **létre kell hozni** (melynek során az inicializálás végbe mehet):

```
peldany:=Osztaly.Create;
```

3. feladat

Fejezze be a `String` elemű sor unitját! Felhasználhatja a mellékelt keretet, de –természetesen– „0-ról” is kezdheti. Letöltés:

people.inf.elte.hu/szlavi/AlgAdat/1felev/Sor_folytonosan/StrSor_Unit_Keret.pas

A kész unitnak és fájljának legyen a neve: `StrSor_Unit`, amelyből „minden” kiderül.

4. feladat

Készítse el a unitból kiindulva az `OSor.inc` inklúd-állományt!

5. feladat

Készítse el (akár a 3. feladatbeli unitból kiindulva) az `OSor_Sablon_Unit.pas` class-alapú unitot! Letöltés:

people.inf.elte.hu/szlavi/AlgAdat/1felev/Sor_folytonosan/OSor_Sablon_Unit_Keret.pas

A kész unitnak és fájljának legyen a neve: `OSor_Sablon_Unit`, amelyből „minden” kiderül.

További feladatok

6. feladat

Készítse el a **szöveg-sor kiíró** művelettel bővített unitját! Az eljárásban ne használjon sorműveleteket, hanem közvetlenül az ábrázolást vegye figyelembe. Így jól használható lesz „önellenőrzésre”, azaz a többi művelet tesztelésére!

```
Procedure Kiir(Const cim:String)
{Uf: Ki:cim [középre igazítva, címszerűen]
     Ki:eleje,vege,hossza
     Ki:se[1..MaxHossz] [soronként]
     Ki:hiba}
```

Erre a műveletre is legyen igaz, hogy a **többitől teljesen független** legyen!
A kész unit fájljának a neve legyen: `StrSor_Unit_2.pas`.

7. feladat

Foglalkozzunk egy kicsit a hiba-állapottal! Ha a sor típusát jól valósítottuk meg, akkor garantáltan csak akkor jut a sor hibás állapotba, amikor valami inkorrekt dolgot hajtattak végre vele. Ezzel azonban csak a **lehetőség**ét teremtettük meg annak, hogy a típusunkat használó programozó **figyelhessen az elkövetett hibákra** (l. a `HibasE` függvényt). Ha nem él a lehetőséggel, akkor a hibás állapotban lévő sort tovább használhatja, aminek az a következménye, hogy a programja „végzetesen” rossz állapotba juthat. Nagyon nagy baj, hogy már pusztán a sor aktuális állapotából nem is lehet következtetni arra a pillanatra, amikor hibás állapotba jutott.

Ennek a problémának az elkerülésére két módszer is kínálkozik a Pascalban.

1. A sorműveleteket csak akkor hagyjuk „érdemben dolgozni”, ha a sor hibátlan. Ezt mondja ki –különben– az ún. **implicit hiba-axióma** is.

   Például:

   ```
   Procedure OSor.Sorba(Const e:TElem);
   Begin
     If not hiba then
     Begin
       If hossza<MaxHossz then
       Begin
         se[vege]:=e; cInc(vege); Inc(hossza);
       End
         else
       Begin
         hiba:=True;
       End;
     End;
   End;
   ```

   Pirossal jelöltük az újdonságot.

2. Amikor hibára fut a sorművelet rögvest „generáljon” fatális futási hibát, amivel rákényszeríti a programozót a hibafelderítésére, és egyben támogatja is ebben azzal, hogy a lehető legkorábban figyelmezteti a hibára.

Például:

```
Procedure OSor.Sorba(Const e:TElem);
Begin
  If hossza<MaxHossz then
  Begin
    se[vege]:=e; cInc(vege); Inc(hossza);
  End
    else
  Begin
    hiba:=True;
  End;
  If hiba then Begin runerror; End;
End;
```

Pirossal jelöltük az újdonságot. Itt a `runerror` eljárást használtuk föl a hiba kikényszerítésére.

Világos, hogy ezek a megoldások egy már „belőtt” programban fölöslegesen növelik a kód hosszát, illetve fölöslegesen rabolják a végrehajtási időt. Ezért csak a fejlesztési stádiumban érdemes használni.

Kérdés: hogyan lehetne úgy beletenni e biztonságot nyújtó kóddarabokat, hogy a végleges változatból ne kelljen utólag egyenként kiszedegetni a kódból?

Megoldás:

A **feltételes fordítás** felhasználásával.

Legyen két, az előfordítónak szóló szimbólumunk a kétféle „szigorúságú” biztonsági kód beillesztésének megszervezéséhez: a `SZIGORU1` és a `SZIGORU2`. Ezeket a unit (vagy az inklúd-állomány) elején definiáljuk, ha a kóddarabokat befordítani szeretnénk, illetve kommentbe tesszük, ha már nincs szükség valamelyikre:

```
{$DEFINE HIBA_ESETEN_URES}//hiba állapotban üresek a műveletek
{$DEFINE HIBA_ESETEN_MEGALL}//hiba állapotba jutva megállás
```

A fenti példában szereplő művelet átírása a feltételesen fordítandó kóddarabokkal:

```
Procedure OSor.Sorba(Const e:TElem);
Begin
  {$IFDEF HIBA_ESETEN_URES}If not hiba then Begin{$ENDIF}
  If hossza<MaxHossz then
  Begin
    se[vege]:=e; cInc(vege); Inc(hossza);
  End
    else
  Begin
    hiba:=True;
  End;
  {$IFDEF HIBA_ESETEN_URES}End;{$ENDIF}
  {$IFDEF HIBA_ESETEN_MEGALL}If hiba then Begin runerror; End;{$ENDIF}
End;
```

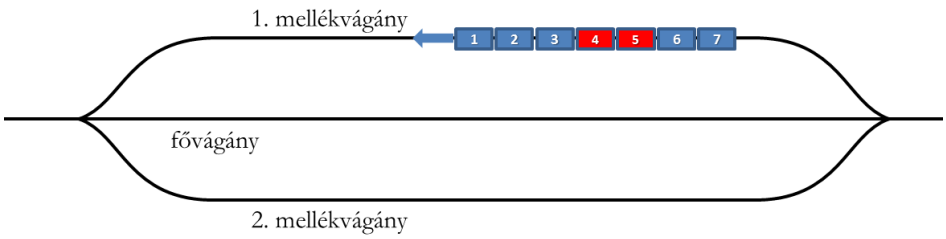
8. feladat

Egy rendező pályaudvaron egy **mellékvágány**ra rátoltak egy szerelvényt, amelyen néhány városba szállítandó árukat pakoltak. Egy-egy kocsijában csak egyfélét, és egy adott címre továbbítandó árut. A mellékvágány eleje –egy váltóval– a fővágányra fut, a vége –egy váltóval– a fővágányról ágazik le. A mellékvágány elejéről szabad egyenként kocsikat levontatni, ill. a mellékvágány végére szabad kocsikat rátolni. (Tehát **sor**-szerűen működik.)

A szerelvényben az egy városba szállítandók egymást követő kocsikon helyezkednek el. Minden kocsiról tudjuk, hogy mi a célállomás irányítószáma (4-jegyű egész szám), valamint ismerjük az áru kódját (5 karakteres szöveges kód).

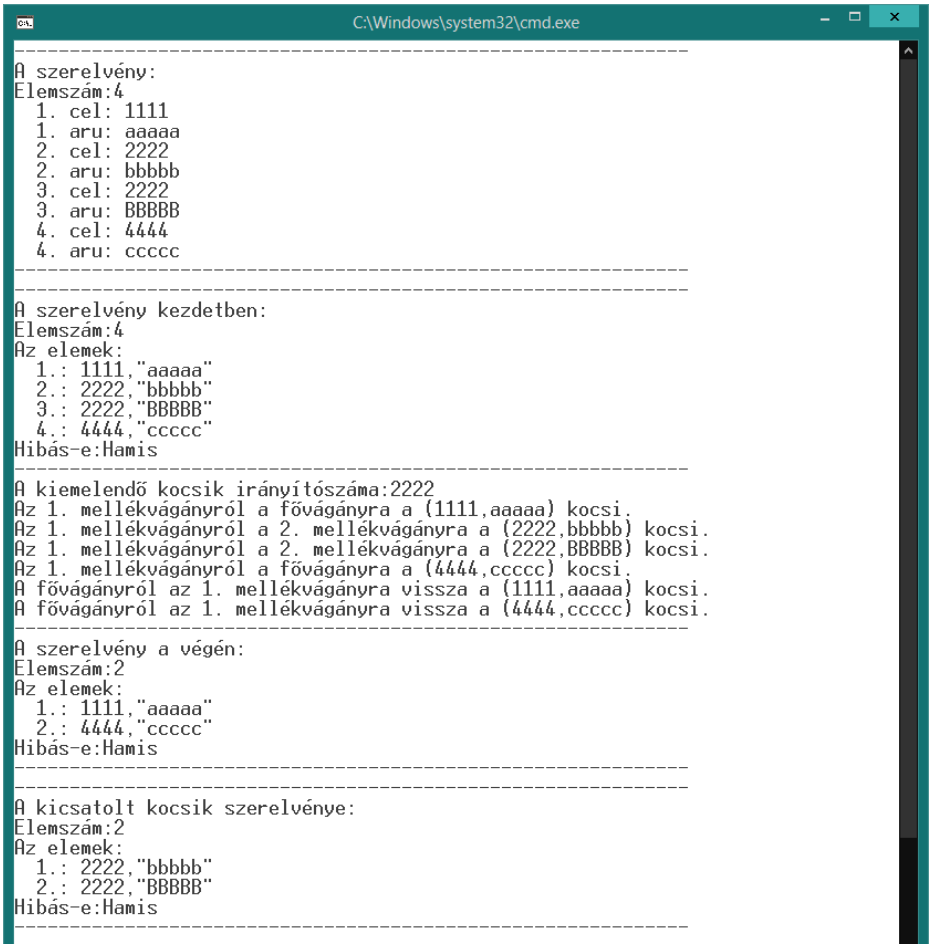
Kiderült, hogy az egyik városba szállítandó kocsikat vissza kell tartani. Szedje ki a szerelvényből a még nem továbbítható kocsikat úgy, hogy a többiek sorrendje nem változhat, és csak a fővágányt használhatja erre! A visszatartandók egy másik mellékvágányon landoljanak!

A pirosak a kicsatolandó kocsik



Sorolja föl (a képernyőn) a szerelvényt tartalmazó mellékvágányon tartózkodó kocsikat kezdetben és a végén, továbbá a másik mellékvágány „tartalmát" is, amelyen a kivezetett kocsik vannak (szintén eredeti sorrendjükben)! A fő- és a mellékvágányokat –természetesen– egy-egy sortípusú adatszerkezettel kell megvalósítani. (A kocsik mozgatását végző mozdony mozgását nem kell programozni.)

A feldolgozás közben jelenjen meg a képernyőn a kocsi-mozgatás művelete. Ilyesféle képernyő kimenetre gondoltam:



A megoldás annál többet ér, minél kevesebb sorral, sorműveléttel dolgozik.

Házi feladatok

Hf0

Készítsen egy sort tesztelő programot! Az `StrSor_Unit_2` unitra építheti, amely tartalmaz egy tesztelés során jól használható sor-állapot kiíró rutint is.

Mindenekelőtt gondolja meg, milyen érdekes sorállapotok lehetnek, amelyeket érdemes vizsgálni! Néhány példa:

1. Az üres sornak nincsen egyetlen eleme sem, és nem is hibás.
2. Üres sorba betétel után pontosan egy elem lesz benne, éppen az, amit betettünk, és állapota: nem hibás.
3. Ha teli sorba elemet akarunk tenni, akkor állapota hibássá válik, de elemei nem változnak.

Hf1

Készítsen az `StrSor_Unit`-ból kiindulva egy **`Integer`**-eket tartalmazó unitot `IntSor_Unit` néven!

Hf2

Készítse el az alábbi egyszerű szimulációs programot!

A boltban `K` db kassza van. A fizetni szándékozók véletlenszerű időpontban érkeznek a pénztárakhoz. Stratégiájuk az, hogy a legrövidebb sor végére állnak. „Mérjük meg”, hogy mennyi az átlagos várakozási idő a pénztáraknál!

A `Penztarak_Keret.pas` a megoldó program kezdeménye.

http://people.inf.elte.hu/szlavi/AlgAdat/1felev/Sor_szekvencialisan/Penztarak_Keret.pas

Bemenete a `K` – a pénztárak száma; és `N` – a vevők száma. Van egy eljárás, amely képes véletlenszerűen feltölteni a `vevok` tömböt.

Erről a következőt elég tudni:

```
Const
  MaxK=10; {pénztárak maximális száma}
  MaxN=100;{vevők maximális száma}

Type
  TVevo=Record
          ki:Integer;    {a vevő rajtszáma}
          odaer:Integer;{a pénztárakhoz érkezés ideje}
        End;
  TVevok:Array [1..MaxN] of TVevo;

Procedure VevokErkezese(Const n:Integer; Var vevok:TVevok);
{Feltölti a vevők tömböt növekvő érkezési idők szerint rendezve
 n darab vevővel}
```

Az aktuális időt a `T:Integer` változó mutatja. (Önnek kell kezelnie!)

A lényeget a `VevoSzimulacio` eljárás jelenti. Ennek kell a `vevok` tömbben lévő vevőket a megfelelő (pénztár-)sorba beléptetnie a megfelelő időpontban; és kiléptetni minden időegységben egy-egy embert a sor elejéről (ha van egyáltalán ott); továbbá az egyes emberek össz várakozási idejét figyelembe venni az átlagos várakozási idő kiszámításához. Amikor az utolsó (=`N`.) vevő is kilépett a sorból, akkor ér véget a szimulációs eljárás feladata, pontosabban akkor kell/lehet az átlagot kiszámítani.

A megoldáshoz persze szükség van egy olyan sor unitra, amely vevők adatait képes tárolni. Két út áll Ön előtt:

1. **Módosítja** a **sor unitját** úgy, hogy az elemtípus `TVevo` legyen!
   Ekkor a unit neve legyen `VevoSor_Unit`!
2. **Ír két konverziós rutint**, amelyek a `TVevo` és a `String` között konvertálnak. Így „vezeti vissza" a megoldást a már kész `String` elemű unitjára.
   Ötlet: a vevő két adatát szöveggé alakítja, és pl. vesszővel választja el őket egymástól. Ehhez használhatja az alábbi Pascal műveleteket:
   - `Str(szám,szöveg)`
     `{a szám a szövegbe kerül karakteresen}`
   - `Pos(mi,miben):hol`
     `{a miben szövegben megkeresi a mi szöveg pozícióját; ha nincs benne, akkor 0}`
   - `Copy(szöveg,tól,db):szöveg`
     `{=szöveg[tól..tól+db-1]}`
   - `Val(szöveg,szám,hiba)`
     `{a kerekteres számot a számba teszi, ha nem lehet, akkor a hiba≠0}`

Hf3 (verem+sor)

Egy vállalat 2 emeletes házában egy lift üzemel. A földszinten érkeznek reggel a dolgozók. Készítsünk lift-szimulációs programot, amely az alábbiakat tudja:

1. a bemenet tartalmazza az érkező dolgozók nevét és érkezési idejét (valamilyen választott egységben) egész szám formájában, növekvő idők szerinti sorrendben; a program inputja a programíró ízlése szerint akár a klaviatúráról, akár szövegfájlból jöhet;
2. kezdetben (`T=0`-kor) a lift a földszinten várakozik;
3. a liftbe legfeljebb `N` ember fér be, veremszerűen;
4. ha van várakozó a (`T` időpontban), akkor beszáll annyi ember, amennyi csak befér, érkezési sorrendben;
5. a liftnek `TL1` időegységig tart, amíg beszállnak, felviszi az embereket, majd kiszállnak, és `TL2` ideig, amire visszaér a földszintre;
6. kiírandó, hogy az egyes emberek milyen sorrendben és mennyi várakozás után jutottak fel.

Az algoritmus vázlata:

```
Típus
  TVáró=Rekord(név:Szöveg,érk:Egész,...súly...)

Változó
  várók:Sor(TVáró) [a lift előtt sorakozók sora]
  T:Egész [az aktuális időpont]
  lift:Verem(TVárók)
```

```
Eljárás LiftSzimuláció:
  T:=0; Üres(lift)
  Ciklus amíg nem ÜresE(várók)
    Ciklus amíg Első(várók).érk≤T és Mélység(lift)<N
      Verembe(lift,Sorból(várók))
    Ciklus vége
    Ha nem ÜresE(lift) akkor [a lift elindul]
      T:+TL1
      kiszállás(lift,T) [a liftezők adminisztrálása; Sorürítés]
      T:+TL2
    különben [a lift tovább vár]
      T:+1
    Elágazás vége
  Ciklus vége
Eljárás vége.
```

Alakítsa át figyelembe véve, hogy a be- és a kiszállás emberenként **1 időegység**be telik! Vegyen figyelembe egy **súlykorlátozás**t is: ha a liftbe beszálltak súlyához a következő belépő súlya már meghaladná a korlátot, akkor ő már nem léphet be.

Hf4

A korábbi szimulációs feladatot oldja meg a `class`-alapú sor osztályra építve!

Megjegyzés: a **`cInc`** tanulságát tartsa emlékezetébe! Minden rutin, amelyet egy metódus (sőt akár szimbolikus konstans is, mint a **`CrLf`**) meghív (felhasznál) szerepelnie kell az osztály metódusai (definíciós részben) között. Emiatt fájl-inklúdálás (pl. az **`AltRutinok.inc`**-é) nem használható.