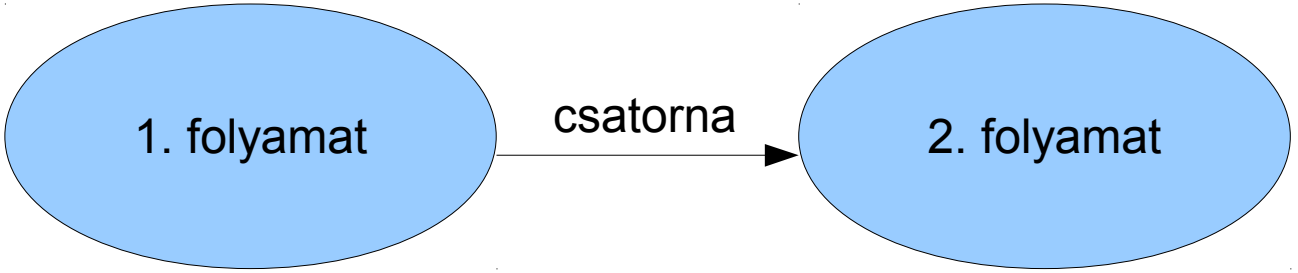
Occam 1.

Készítette: Szabó Éva

Párhuzamos programozás

- Egyes folyamatok (processzek) párhuzamosan futnak.
- Több processzor -> tényleges párhuzamosság
- Egy processzor -> Időosztásos szimuláció
- Folyamatok közötti kommunikáció: csatornákon keresztül



- A csatorna akkor jön létre, mikor mindkét folyamat vezérlése a megfelelő részhez ér!

KroC

- Fordítás: ***kroc -d*** *fájlnév****.occ***

  pl.: *kroc -d pelda.occ*
- A lefordított állomány futtatása: ***./*** *fájlnév*

  pl.: *./ pelda*
- külső libek linkelése: *-lnév*

  pl. kroc -lcourse -d fajl.occ
- Csak hibamentesen fordul le, és keletkezik futtatható állomány!
- Linux!

Hasznos tudnivaló az Occamről

- Minden, a nyelvben **lefoglalt kulcsszó**t nagy betűvel kell írni (***SEQ***, ***PAR***, ***PROC***, stb…)
- A blokkstruktúrát indentációval jelöljük (két szóközzel beljebb kezdjük, **nem** tabulálunk!)
- Minden egyes kifejezés **új sorban** kezdődik (esetlegesen két szóközzel beljebb)
- Egy soros megjegyzés: *-- ez egy megjegyzés*

Occam programok felépítése

<deklarációk>

<folyamatok>

pl.:

```
INT x:
SEQ
  x := 10
  x := x + 1
```

Elemi folyamatok (5 db)

- Értékadás – ***<változó> := <kifejezés>***

  pl.: *k := k + 10*

- Küldés – ***<csatorna> ! <kifejezés>***

  pl.: *C ! k + 5*

- Fogadás – ***<csatorna> ? <változó>***

  pl.: *C ? x*

- SKIP – ***SKIP***

  pl.: *SKIP*

- STOP – ***STOP***

  pl.: *STOP*

- A ***SKIP*** folyamat a legegyszerűbb elemi folyamat, „semmit nem csinál”. Haszontalannak tűnhet, de összetettebb programok esetében (például még nem kifejlesztett programrészek esetében) hasznos lehet.
- Párhuzamos folyamatok esetében **fontos, hogy minden folyamat terminál**jon, ellenkező esetben az egész, folyamatokból álló „rendszer” leáll.
- A ***STOP*** szintén „nem csinál semmit”, de **ez sosem terminál** – ellentétben a SKIP-el. Egy folyamatban a STOP (feltéve hogy a vezérlés odakerül), annak **holtpontba jut**ását eredményezi. Szintén haszontalannak tűnhet, de ezzel egy folyamatot leállíthatunk más folyamatok működésének befolyásolása nélkül, ami **hibakeresésnél hasznos** lehet.

Holtpont

Azt mondjuk, hogy egy folyamat holtpont állapotba került, ha az **már nem képes további működésre (vezérlése leáll)**, és ez a leállás nem a folyamat helyes lefutásának eredménye.

Párhuzamos folyamatok közül akár **egy folyamat holtpont állapotba kerülése az egész program holtpont állapotba kerülését eredményezi**, hiszen az összes többi folyamat várja a holtpontban levő folyamat terminálását, ami sosem fog bekövetkezni

A precedencia nem meghatározott.

- Ezért fontos a **zárójelezés**!
- Helytelen:

  *x := x * 3 + 5*
- Helyesen:

  *x := (x * 3) + 5*
- Küldés/fogadás esetén használhatjuk a **;**-t több kifejezés küldése/fogadása esetén.

  pl.: *C ! x; y; x + y*

Ezesetben a C csatorna protokollja egy (Int; Int; Int) szekvenciális protokoll lesz.

Adattípusok

- Az Occam ERŐSEN típusos nyelv.
- Minden változót deklarálni kell!
- Változók deklarálása:

  <típus> <azonosítók vesszővel elválasztva>:
- Például két egész típusú x és y változó deklarációja így néz ki:

  INT x, y:

BOOL	Logikai típus, értéke lehet: TRUE vagy FALSE
BYTE	8-bites nem előjeles egész érték (mint a C-beli *char*)
INT	Általában 32 vagy 64-bites egész érték (mint a C-beli *int*)
INT16	16-bites előjeles egész
INT32	32-bites előjeles egész
INT64	64-bites előjeles egész
REAL32	32-bites valós érték
REAL64	64-bites valós érték

- Csatorna deklarálása:

  CHAN [OF] <típus> <azonosító>:

- Az *OF* elhagyható. Pl. egy egész értékeket továbbító *c* csatorna deklarációja a következő:

  CHAN OF INT c: **vagy** ***CHAN INT c:***

- Az azonosítók deklarációi bárhol elhelyezhetők a kódban, nem kell hogy a program legelején legyenek.
- Mivel az Occam nyelvet biztonságos párhuzamos programozásra tervezték, ezért a **pointerek használata nem engedélyezett**.

Operátorok

! -- csatorna adatküldés

? -- csatorna adatfogadás

:= -- értékadás

+, -, *, / -- szokásos

**** -- egész osztás maradéka

AND, OR, NOT -- logikai operátorok

=, <> -- egyenlő-e, nem egyenlő

<, >, <=, >= -- szokásos relációk

Tömbök

- csak előre rögzített méretű lehet
- a dimenziót a típus előtt kell megadjuk,
- 0-tól indexelünk

pl.: *[5]INT egeszek:*

5 elemű, egészeket tároló vektor

[10][10]REAL64 matrix:

2 dimenziós, 10x10-es mátrix

SEQ

- SEQuential – szekvenciális
- A *SEQ* blokkjában definiált folyamatok szekvenciálisan kerülnek végrehajtásra. pl.:

```
SEQ
  x := 10
  y := x + 1
  z := x – 1
```

- Vegyük észre, hogy mindhárom folyamat 2 szóközzel beljebb kezdődik, amivel a blokkstruktúrát határozzuk meg. Ez nem egy lehetőség a könnyebb olvashatóságra, ezt kötelezően így kell írjuk!

PAR

- PARallel – párhuzamos
- A PAR (PARallel) blokkjában definiált folyamatok párhuzamosan kerülnek végrehajtásra.
- Definiáljunk két párhuzamosan működő elemi folyamatot:

PAR

INT m:

c1 ? m -- adatfogadás a c1 csatornáról

INT n:

c2 ? n -- adatfogadás a c2 csatornáról

Bármennyi folyamat (független attól, hogy azok elemi vagy nem elemi folyamatok) **futtatható párhuzamosan**. Az egész *PAR* blokk akkor terminál, ha a benne „elindított” folyamatok **mindegyike** terminál.

Vezérlési szerkezetek

- **IF – Feltételes vezérlés**

  feltételhez köthetjük egy folyamat kiválasztását

IF

<logikai kifejezés 1>

<folyamat 1>

<logikai kifejezés 2>

<folyamat 2>

<logikai kifejezés 3>

<folyamat 3>

Az a folyamat kerül kiválasztásra, mely feletti logikai kifejezés értéke igaz. Ha egyik logikai értéke sem igaz, akkor az *IF* szerkezet ***STOP***-ként funkcionál, ezért ezt mindig kerüljük. A ***STOP*** **elkerülésére két lehetőség** is adódik. Az egyik lehetőség, ha **a logikai vizsgálatok minden lehetséges értéket lefednek**, így valamelyik feltétel mindig teljesül, pl.:

```
IF
  x > y
    a := 1
  x < y
    a := 2
  x = y
    a := 3
```

Nem tudunk úgy értéket adni *x*-nek és *y*-nak, hogy a feltételek közül egyik se teljesüljön.

A másik lehetőség, ha **a feltételek halmazába felvesszük a *TRUE* értéket, amelyhez a *SKIP* folyamat tartozik**. A *SKIP* nem csinál semmit, de hatására a folyamat nem kerül holtpontba, pl.:

```
IF
  x < 10
    a := 1
  TRUE
    SKIP
```

A ***TRUE-SKIP* nélkül** és *x>9* teljesülése esetén az *IF* szerkezet **holtpontba** kerülne.

WHILE

- Elöltesztelő ismétléses vezérlés

WHILE <logikai kifejezés>

<folyamat>

- pl.:

WHILE x<10

x := x+1

Replikáció

- Összetett folyamatok (*SEQ*, *PAR*, *IF*, ...) felírhatók egy ún. replikált alakban. Ehhez egy for-ciklushoz hasonló szerkezetet fogunk használni.

- Tekintsük a SEQ egy replikált változatát:

```
SEQ i = 1 FOR 4
  c[i-1] ! x[i-1]
```

- Ennek a nem replikált megfelelője:

```
SEQ
  c[0] ! x[0]
  c[1] ! x[1]
  c[2] ! x[2]
  c[3] ! x[3]
```

- az IF összetett folyamat egy lehetséges replikált alakja:

```
IF i=0 FOR 4
  x[i] > 5
    z := i
```

- Ennek nem replikált megfelelője:

```
IF
  x[0] > 5
    z := 0
  x[1] > 5
    z := 1
  x[2] > 5
    z := 2
  x[3] > 5
    z := 3
```

PROC

- A PROC egy **előre definiált, névvel ellátott folyamat**. (Tekinthetünk úgy rá, mintha egy eljárást definiálnánk.)
- **paraméterezése tetszőleges lehet, kivéve azt a PROC-ot, mely a program belépési pontja lesz, annak paraméterezése (és a paraméterek sorrendje) ugyanis kötött:**

  (standard input, standard output, standard error)
- A standard input egy BYTE csatorna. Ennek megfelelően a belépési pontot meghatározó PROC paraméterezése így néz ki (ahol a keyboard, screen és error helyett tetszőleges azonosítókat írhatunk):

  (CHAN BYTE keyboard, screen, error)

- Egy PROC deklarálásának általános alakja:

  PROC <azonosító>(paraméterek)

  <folyamatok>

  :

- Ha egy Occam program egyetlen *PROC* deklarációt tartalmaz, akkor azt a KroC a program fő belépési pontjának tekinti.
- Ha több is van, akkor egymásba ágyazás esetén a legkülső, több egyszintű PROC esetén pedig a **legutoljára** deklarált lesz a program fő belépési pontja.

Hello World!

```
PROC hello(CHAN BYTE keyboard, screen, error)
  VAL szoveg IS "Hello*c*n":
  SEQ i=0 FOR (SIZE szoveg)
    screen!szoveg[i]
:
```

- A második sor egy „szoveg” azonosítójú, „Hello” sztringet tartalmazó konstans. A *c*n rendre a C-beli \r\n megfelelői. Ezt követi egy replikált *SEQ*, mely 0-tól a szöveg konstans hosszáig megy. Ez a blokkjában levő folyamatok szekvenciális végrehajtását írja elő. A *SEQ* blokkja a screen csatornára (a képernyőre) a sztringet – mint *BYTE* típusú értékeket tároló tömböt – karakterenként a képernyőre írja. Mint általában minden deklaráció, ez is :-al fejeződik be.