Programozási nyelvek I.
2. gyakorlat

Balogh Ádám
bas@elte.hu
Eötvös Loránd Tudományegyetem
Informatikai Kar

1. házi feladat megoldása (1)

log_ker.adb:

```
...
   P : Natural := S'First;
   Q : Natural := S'Last;
   K : Natural := ( P + Q ) / 2;
   V, L : Boolean := False;
begin
   while ( not V ) and then ( not L ) loop
      V := ( P > Q );
      if S ( K ) < C then
(folyt. köv.)
```


1. házi feladat megoldása (2)

log_ker.adb:

```
(folyt. köv.)
      P := K + 1;
   elsif S ( K ) > C then
      Q := K – 1;
   else
      L := True;
   end if;
   K := ( P + Q ) / 2;
end loop;
...
```


2. házi feladat megoldása

lin_ker2.adb:

```
...
      V : Boolean := ( P + 1 >= S'Last );
      L : Boolean := False;
...
```


Segítség az 1. feladathoz (1)

- Pozitív egészek típusa: `Positive`
- Karakterlánc változók deklarálása:
  Név `: String (` *Eleje* `..` *Vége* `);`
- Kiírás soremelés nélkül: `Text_IO.Put`
- Beolvasás billentyűzetről:
  `Text_IO.Get_Line (` *Karakterlánc*`,` *Hossz* `)`
- Karakterlánc részlánca:
  Karakterlánc `(` *Eleje* `..` *Vége* `)`

Segítség az 1. feladathoz (2)

- Karakterlánc konvertálása pozitív egésszé:
  `Positive'Value (` *Karakterlánc* `)`
- Elágazás diszkrét kifejezés értéke alapján:

```
case Kifejezés is
   when Értékhalmaz =>
      Legalább egy utasítás
   when Értékhalmaz =>
      Legalább egy utasítás
   ...
end case;
```

Segítség az 1. feladathoz (3)

- Értékhalmazok: *Érték*, *Érték* `..` *Érték*, *Értékhalmaz* `|` *Értékhalmaz* és `others`
- Az egyes ágak értékhalmazainak diszjunktaknak kell lenni
- Az összes ág értékhalmazai uniójának le kell fednie a kifejezés típusának típusértékhalmazazát
- Az `others` csak a végén szerepelhet, és lefedi a maradék ágakat

1. feladat

Írj Ada programot, ami egy adott sorszámú napról megmondja, hogy az a hét első, utolsó vagy középső munkanapja, egyéb hétköznap vagy hétvége. Ha a nap sorszáma nagyobb, mint 7, a program adjon hibaüzenetet. Használd a `case` utasítást! A program neve legyen `Naptar`!

1. feladat megoldása (1)

naptar.adb:

```
...
procedure Naptar is
   Nap : Positive;
   Szam : String ( 1 .. 10 );
   Szam_Hossza : Natural;
begin
   Put ( "Nap: " );
   Get_Line ( Szam, Szam_Hossza );
   Nap := Positive'Value
            ( Szam ( 1 .. Szam_Hossza ));
(folyt. köv.)
```


1. feladat megoldása (2)

naptar.adb:

```
(folyt.)
case Nap is
      when 1 => Put_Line ( "Első m.nap" );
      when 3 => Put_Line ( "Köz. m.nap" );
      when 5 => Put_Line ( "Ut. m.nap." );
      when 2 | 4 => Put_Line ( "M.nap." );
      when 6 .. 7 => Put_Line ( "7vége." );
      when others => Put_Line ( "Rossz!" );
   end case;
end Naptar;
```


Segítség az 2. feladathoz (1)

- Függvény definiálása:

```
function Név ( Paraméterek ) return Típus is
   Változódeklarációk
begin
   Utasítások
end Név;
```

- Paraméterek alakja:

```
Név, Név, … : Típus; Név, Név, … : Típus; …
```

- Külön fordítási egységben: *név*`.adb`
- Paraméterek értéke nem változtatható
- Érték visszaadása: `return` *Kifejezés*

Segítség a 2. feladathoz (2)

- Függvény használata a főprogramban: mint egy csomagot: `with` *Név*, de nem kell (és nem is szabad) utána `use` *Név*!
- Függvény meghívása: tetszőleges (függvény visszatérési típusának megfelelő kifejezés helyén): *Név* `(` *Paraméterek* `)`
- Paraméterek itt: *Kifejezés*, *Kifejezés*, …, amiknek típusa meg kell, hogy feleljen a függvény definíciójában meghatározottnak

2. feladat

Módosítsuk a (javított) lineáris keresést úgy, hogy a találatot egy adott pozíción a `Talalt` függvény jelezze egy logikai típusú érték visszaadásával, melynek három paramétere a szöveg, amelyben keresünk, a karakter, amit keresünk és a pozíció, amelyik után az egyezést szeretnénk vizsgálni. A program neve legyen `Lin_Ker3`!

2. feladat megoldása (1)

talalt.adb:

```
...
function Talalt ( Szoveg : String;
                  Ch : Character;
                  Poz : Natural )
   return Boolean is
begin
   return ( Szoveg ( Poz + 1 ) = Ch );
end Talalt;
```


2. feladat megoldása (2)

lin_ker3.adb:

```
with Text_IO, Talalt;
...
begin
   while ( not V ) and then ( not L ) loop
      V := ( P + 1 >= S'Last );
      L := Talalt ( S, C, P );
      P := P + 1;
   end loop;
   ...
end Lin_Ker3;
```


Segítség a 3. feladathoz

- Függvények be is ágyazhatók egy másik fordítási egységbe (pl. főprogramba):

```
procedure Fő is
   Függvénydefiníciók
   Változódeklarációk
begin
   Utasítások
end Fő;
```

- Ilyenkor nem szerepel a `with` utasításnál!

3. feladat

Ágyazzuk be a `Talalt` függvényt a főprogramba! Az új program neve legyen `Lin_Ker4`!

3. feladat megoldása

lin_ker4.adb:

```
...
procedure Lin_Ker4 is
   function Talalt ...
      ...
   end Talalt;

   S : constant String ...
   ...
begin
   ...
end Lin_Ker4;
```

Segítség a 4. feladathoz

- Egy változó deklarációja az adott blokkban, és annak a deklarációt követő alblokkjaiban látható
- Adott blokkra nézve globális változó: a blokkot tartalmazó blokkok valamelyikében van deklarálva
- Függvénydefiníciók és változódeklarációk keverhetők tetszőleges sorrendben egy deklarációs részen belül

4. feladat

A begyázott `Talalt` függvény ne kapjon paramétereket, hanem a főprogram változóit, mint globális változókat használja! Az új program neve legyen `Lin_Ker5`!

4. feladat megoldása

lin_ker5.adb:

```
...
procedure Lin_Ker5 is
   S : constant String ...
   ...

   function Talalt return Boolean is
      ...
   end Talalt;
begin
   ...
end Lin_Ker5;
```


Segítség az 5. feladathoz

- Rekurzív függvény: önmagát hívja
- Vigyázni kell, nehogy „végtelenszer” hívja önmagát

5. feladat

Alakítsd át a 10 faktoriálisát számoló programot úgy, hogy a faktoriálist a `Fakt` rekurzív függvény számolja, melynek paramétere az a szám, amelynek faktoriálisára kíváncsiak vagyunk. Ez a függvény lehet külön fordítási egység vagy akár a főprogramba ágyazott függvény is. Az új program neve legyen `Faktor2`!

5. feladat megoldása (1)

fakt.adb:

```
function Fakt ( Szam : Natural )
   return Natural is
begin
   if Szam = 0 then
      return 1;
   else
      return Szam * Fakt ( Szam – 1 );
   end if;
end Fakt;
```

5. feladat megoldása (2)

faktor2.adb:

```
with Text_IO, Fakt;
use Text_IO;

procedure Faktor2 is
begin
   Put_Line ( ”10!=” &
              Natural’Image ( Fakt ( 10 )));
end Faktor2;
```


Segítség a 6. feladathoz (1)

- Eljárás definiálása:

```
function Név ( Paraméterek ) is
   Változódeklarációk
begin
   Utasítások
end Név;
```

- Paraméterek alakja:

```
Név, … : Mód Típus; Név, … : Mód Típus; …
```

- Egy paraméter módja lehet bemenő (`in`), kimenő (`out`) vagy ki- és bemenő (`in out`)

Segítség a 6. feladathoz (2)

- Eljárás ugyanúgy lehet külön fordítási egység, mint egy függvény, és ugyanúgy beágyazható más fordítási egységbe is
- Ugyanúgy with-tel use nélkül használjuk, mint egy függvényt
- Meghívása: utasítás helyén, ugyanúgy, mint egy függvényt
- Az `out` és `in out` paraméterek helyére csak változó írható!

6. feladat

Írj egy `Kozos` eljárást, amely kiszámítja két pozitív egész legkisebb közös osztóját és legnagyobb közös többszörösét. Az eljárás bemenő paramétere legyen a két szám, kimenő paramétere pedig a két eredmény. Az eljárás segítségével írj programot, amely kiszámítja 12 és 18 legnagyobb közös osztóját és legkisebb közös többszörösét. A program neve legyen `LNKO_LKKT`!

6. feladat megoldása (1)

kozos.adb:

```
procedure Kozos ( A, B : in Positive;
                  LNKO, LKKT : out Positive ) is
   X : Positive := A;
   Y : Positive := B;
begin
   ...
   LNKO := A;
   LKKT := B;
end Kozos;
```


6. feladat megoldása (2)

lnko_lkkt.adb:

```
with Text_IO, Kozos;
use Kozos;

procedure LNKO_LKKT is
   P, Q : Positive;
begin
   Kozos ( 12, 18, P, Q );
   Put_Line ( ”(12;18)=” &
                  Positive’Image  ( P ));
   ...
end LNKO_LKKT;
```

7. feladat

Írj egy `Novel` nevű eljárást, amely megnöveli a paraméterben kapott változó értékét eggyel! Írj programot az eljárás segítségével, amely megnöveli egy kezdetben 6-ot tartalmazó változó értékét, és kiírja a képernyőre! A program neve legyen `Noveles`!

7. feladat megoldása (1)

novel.adb:

```
procedure Novel ( Sz : in out Natural ) is
begin
   Sz := Sz + 1;
end Novel;
```


7. feladat megoldása (2)

noveles.adb:

```
with Novel, Text_IO;
use Text_IO;

procedure Noveles is
   N : Natural;
begin
   N := 6;
   Novel ( N );
   Put_Line ( "N = " &
                  Natural'Image ( N ));
end Noveles;
```


Segítség a 8. feladathoz

- Paramétereknek lehet alapértelmezett értéke: *`Név, … : [in] Típus := Érték`*
- Ezek elhagyhatók (de nem középről)
- Hívás névvel jelölt formában: paramétereket *`Név => Kifejezés`* alakban adjuk meg, sorrendtől függetlenül
- Vegyes forma: előbb pozicionális, majd névvel jelölt

8. feladat

Egészítsd ki a `Novel` eljárást `Novel2` néven egy bemenő paraméterrel, amely azt mondja meg, hogy a számot mennyivel kell megnövelni. A paraméter legyen elhagyható, ebben az esetben az eljárás ugyanazt a műveletet végzi, mint a `Novel`. A `Noveles2` programban próbálj ki minél több paraméterátadási formát, amikor a `Novel2`-t hívod!

8. feladat megoldása (1)

novel2.adb:

```
procedure Novel2 ( Sz : in out Natural;
                   Kul : in Natural := 1 ) is
begin
   Sz := Sz + Kul;
end Novel2;
```

8. feladat megoldása (2)

noveles2.adb:

```
...
   Novel2 ( N );
   Novel2 ( Sz => N );
   Novel2 ( N, 2 );
   Novel2 ( Sz => N, Kul => 2 );
   Novel2 ( Kul => 2, Sz => N );
   Novel2 ( N, Kul => 2 );
...
```


Házi feladatok (1)

- Írj egy `Max2` és egy `Max3` nevű függvényt, amely két illetve három szám maximumát számolja ki. A két főprogram, amely ezeket használja legyen `Maximum2` és `Maximum3`, és írja ki a képernyőre 4 és 6 illetve 4, 6 és 11 közül a legnagyobbat!
- Ágyazd be mindkét függvényt azonos `Max` néven egy `Maximum` nevű főprogramba, amely egymás után kiírja 4 és 6, illetve 4, 6 és 11 maximumát!

Házi feladatok (2)

- Írj programot, amely összeszorozza egymással az első két parancssori argumentumában kapott számot! Ha az argumentumok száma kevesebb, mint kettő, írjon ki hibaüzenetet a képernyőre! A program neve legyen `Szoroz`! (Segítség: `Ada.Command_Line`, könyv A.6.)