

6. előadás

Hatókör, láthatóság, élettartam. Változók leképzése a memóriára. Blokkszerkezetes nyelvek. Kivételkezelés.

Néhány alapfogalom

- Definíció
- Deklaráció
- Hatókör, láthatóság
- Progamegység
- Blokkszerkezet

Definíció

- ▣ Egy programentitás megadása, egy új entitás bevezetése
- ▣ Változó, típus, programegység (alprogram, csomag, osztály stb.)
- ▣ Például egy programegység definíciója:
specifikáció és törzs
 - specifikáció: hogyan használhatom
 - törzs: hogyan működik

Deklaráció

- „Bejelentés”
- Egy entitáshoz egy nevet rendel
- Ezzel a névvel lehet használni az entitást
- A névvel jelölt entitás tulajdonságainak megadása (hogyan lehet használni)
- Nem feltétlenül definiálja az entitást
 - Előre vetett deklaráció

Változódeklaráció

```
int x;           X: Integer;
```

- A változóhoz az **x/X** nevet rendeli
- Gyakran definiálja is a változót (memóriaterületet rendel hozzá)
- Nem mindig definiál:

```
extern int x;
```

- az x-et int típusú változóként lehet használni

Típusdefiníció és típusdeklaráció

- A típusdefiníció új típust vezet be

```
type Int is new Integer;
```

```
Tömb: array (1..0) of Float;
```

- A típusdeklaráció egy nevet vezet be egy típushoz

```
typedef int integer; // szinoníma
```

- Név szerinti vs. szerkezeti típusequivivalencia

Alprogram definíciója

```
void swap( int& x, int& y )  
{  
    int z = x; x = y; y = z;  
}
```

specifikáció

```
procedure Swap ( X, Y: in out Integer ) is  
    Z: Integer := X;  
begin  
    X := Y; Y := Z;  
end Swap;
```

törzs

deklarációk is

Alprogram deklarációja

- ▣ Nem kell a teljes definíció
- ▣ Csak az, hogy hogyan kell használni
 - specifikáció
- ▣ `void swap(int& x, int& y);`
- ▣ `procedure Swap (X, Y: in out Integer);`

Deklaráció definíció nélkül (1)

- ▣ Interfészek definiálásánál (osztály, csomag, modul)

```
package Ada.Text_IO is
    ...
    procedure Put(Item : in String);
    ...
end Ada.Text_IO;
```

Csak deklaráljuk.

A definíció a
csomag törzsben.

Hasonló a C/C++ nyelvekben

```
#ifndef PROBA_H
#define PROBA_H
extern int x;
int f ( int p );
#endif /* PROBA_H */
```

```
#include "proba.h"
int x;
int f ( int p ) { return x+p; }
```

```
#include "proba.h"
int g( int t ) {
    x = 1; return f(t);
}
```

Deklaráció definíció nélkül (2)

□ Kölcsönös rekurzió feloldása

```
procedure B ( ... );
```

```
procedure A ( ... ) is ... begin ... B(...); ... end A;
```

```
procedure B ( ... ) is ... begin ... A(...); ... end B;
```

Deklaráció definíció nélkül (3)

- ▣ Ada: a renames utasítás. Szinoníma.
 - szöveg rövidítésére
 - nevek közötti konfliktusok feloldására
 - a végrehajtási idő csökkentésére
- ▣ **function Gcd (A, B: Positive) return Positive
renames Lnko;**
- ▣ **AI: Float renames A(I);**
...
AI := AI + 1;

Definíció deklaráció nélkül

- Ha nem adunk nevet a létrehozott entitásnak
 - funkcionális nyelvek: lambda-függvények
 - Java: névtelen osztályok

- Ada: új típust definiálunk név nélkül

```
type T is array (1..0) of Integer;
```

```
V: T;
```

```
X: array (1..10) of Integer;
```

```
Y: array (1..10) of Integer;
```

Deklaráció hatóköre

- Hatókör, hatáskör: scope
- A deklaráció összekapcsol egy entitást egy névvel
- Hatókör:
 - a programszöveg egy szakasza
 - amíg az összekapcsolás érvényben van
- Ezen belül érhetjük el az entitást a névvel
 - Nem biztos, hogy elérjük, lásd láthatóság

Deklaráció láthatósága

- ▣ Visibility
- ▣ A hatókör része
- ▣ Ahol a név az entitást azonosítja
- ▣ Nem feltétlenül az egész hatókör
- ▣ Elfedés (hiding) lehetséges

Elfedés

```
{  
  int x = 1;  
  {  
    int x = 2;  
    cout << x << endl;  
  }  
}
```

- Két változó ugyanazzal a névvel

```
declare  
  X: Integer := 1;  
begin  
  declare  
    X: Integer := 2;  
  begin  
    Put(X);  
  end;  
end;
```


Kiterjesztett láthatóság

```
procedure A is
  X: Integer := 1;
begin
  declare
    X: Integer := 2;
  begin
    Put(X); Put(A.X);
  end;
end;
```

```
K: declare
  X: Integer := 1;
begin
  B: declare
    X: Integer := 2;
  begin
    Put(X); Put(K.X);
  end B;
end K;
```

Minősített név



Hierarchikus felépítés

- Egymásba ágyazunk programrészeket
 - programegységeket, utasításokat
- Blokk
 - blokk utasítás
 - alprogram
- A hatóköri, láthatósági szabályok alapja

Beágyazás

- Progamegységet egy blokk deklarációs részébe
- Blokk utasítást egy blokk utasítássorozat-részébe

```
procedure A is
  procedure B is ... begin ... end;
begin
  declare
    procedure C is ... begin ... end;
  begin
    declare ... begin ... end;
  end;
end;
```

Blokkszerkezetes nyelvek

- Alprogramot beágyazhatunk alprogramba
 - Ilyen: Algol 60, Pascal, Ada, Haskell
 - Nem ilyen: C, C++, Java(!)...
- A nyelv megvalósítása szempontjából fontos kérdés
 - Később még visszatérünk rá...

Blokkok és hatókör

- Egy blokk deklarációinak hatóköre a deklarációtól a blokk végéig tart
- Beleértve a beágyazott blokkokat is

```
procedure A is
    procedure B is ... begin ... end;
begin
    declare
        procedure C is ... begin ... end;
    begin
        declare ... begin ... end;
    end;
end;
```

Blokkok és láthatóság (1)

```
procedure A is
  procedure B is ... begin ... end;
  ...
begin
  ...
  declare
    ...
    procedure B is ... begin ... end;
    ...
  begin
    ...
  end;
  ...
end;
```

Blokkok és láthatóság (2)

```
procedure A is
  procedure B is ... begin ... B; ...end;
  ...
begin
  B;
  declare
    ...
    procedure B is ... begin ... B; ... end;
    ...
  begin
    B;
  end;
  B;
end;
```

Lokális és nonlokális deklaráció

- ▣ Egy blokkban elhelyezett deklaráció lokális (local) a blokkra nézve
 - A blokkon kívülről nem hivatkozható
- ▣ Egy külső blokkban elhelyezett deklaráció nonlokális (non-local) a befoglalt blokkok számára
 - Alternatív elnevezés: globális a befoglalt blokkra nézve

```
declare X: Float; begin declare ... begin ... end; end;
```


Globális deklaráció

- Eddig: lokális / nonlokális (globális) egy deklaráció egy blokkra nézve
 - relatív
- Globális deklaráció: az egész programra vonatkozik
 - A hatóköre az egész program
 - abszolút

```
int glo; void f ( int par ) { int loc = par; glo = loc; }
```

Lokális/nonlokális/globális változó

- ▣ Az X egy
 - lokális változó,
 - nonlokális változó,
 - globális változó
- ▣ Az X deklarációja olyan
- ▣ Ugyanez más entitásokra is (alprogram, típus...)

Statikus és dinamikus hatókör

- A programozási nyelvek többségében: statikus
- Dinamikus: egy alprogramban a hivatkozásokat a hívás környezetében értelmezzük

```
procedure A is
    procedure B is ... begin ... end B;
    procedure C is ... begin B; end C;
    procedure D is
        procedure B is ... begin ... end B;
    begin
        C;
    end D;
begin D; end A;
```

Változók élettartama

- ▣ A program végrehajtási idejének egy szakasza
- ▣ Amíg a változó számára lefoglalt tárhely a változóé
- ▣ Kapcsolódó fogalmak
 - Hatókör
 - Memóriára való leképezés

Hatókör és élettartam

- Sok esetben az élettartam az az idő, amíg a változó hatókörében vagyunk
 - Globális változó: az egész program végrehajtása alatt létezik
 - Lokális változó: csak a definiáló blokk végrehajtása alatt létezik
- Nem mindig így van
 - „Dinamikusan” lefoglalt változók
 - C/C++ static változók, Java zárványok (inner)

Dinamikusan lefoglalt változók

- ▣ Allokátorral lefoglalt tárterület
- ▣ Mutatók, referenciák (egy későbbi előadás)
- ▣ Lásd még: memóriára való leképzés

Ha a hatókör kisebb az élettartamnál

```
int sum ( int p )
```

```
{
```

```
    static int s = 0;
```

```
    s += p;
```

```
    return s;
```

```
}
```

```
void f ( void )
```

```
{
```

```
    cout << sum(10) << endl;
```

```
    cout << sum(3) << endl;
```

```
}
```

Deklaráció kiértékelése

▣ Statikus (fordítás közben)

- Rugalmatlan
- C, C++

```
int t[10];
```

▣ Dinamikus (futás közben)

- pl. Ada
- A blokk utasítás szerepe

A blokk utasítás egyik haszna

```
procedure A is
    N: Integer;
begin
    Put("Hány adat lesz?"); Get(N);
    declare
        T: array (1..N) of Integer;
    begin
        -- beolvasás és feldolgozás
    end;
end;
```

Egy másik haszon

- Ha egy nagy tárigényű változót csak rövid ideig akarok használni
- Egy blokk utasítás lokális változója
- Ada: kivételkezelés

Változók leképzése a memóriára

▣ Statikus

- A fordító a tárgykódban lefoglal neki helyet

▣ Automatikus

- Futás közben a végrehajtási vermen jön létre és szűnik meg

▣ Dinamikus

- Allokátorral foglaljuk le, és pl. deallokátorral szabadítjuk fel (vagy a szemétgyűjtés...)

Statikus változók

- ▣ Az élettartamuk a teljes program végrehajtása
- ▣ Fordítási időben tárterület rendelhető hozzájuk
- ▣ Tipikusan a globális változók
 - A hatókörhöz igazodó élettartam
- ▣ De ilyenek a C/C++ static változók is
- ▣ Egy futtatható program: kód + adat

Egy program a memóriában

- ▣ Futtatás közben a program által használt tár felépítése:
 - kód
 - (statikus) adatok
 - végrehajtási verem
 - dinamikus tárterület (heap)

Dinamikus változók

- Dinamikus tárterület
 - Ahonnan a programozó allokátorral tud memóriát foglalni
 - Explicit felszabadítás vagy szemétgyűjtés
- Mutatók és referenciák
- Utasítás hatására jön létre (és esetleg szabadul fel) a változó
 - Statikus és automatikus: deklaráció hatására

Végrehajtási verem

- ▢ execution stack
- ▢ Az alprogramhívások tárolására
- ▢ Az éppen végrehajtás alatt álló alprogramokról aktivációs rekordok
- ▢ A verem teteje: melyik alprogramban van az aktuálisan végrehajtott utasítás
- ▢ A verem alja: a főprogram
- ▢ Egy alprogram nem érhet véget, amíg az általa hívott alprogramok véget nem értek
- ▢ Dinamikus (hívási) lánc

Aktivációs rekord

- Activation record, stack frame
- Egy alprogram meghívásakor bekerül egy aktivációs rekord a verembe
- Az alprogram befejeződésekor kikerül az aktivációs rekord a veremből
- Rekurzív alprogram: több aktivációs rekord
- Az aktivációs rekord tartalma:
paraméterek, lokális változók, egyebek
 - Blokkszerkezetes statikus hatókörű nyelvek esetén:
tartalmazó alprogram

Automatikus változók

- ▣ A végrehajtási veremben
- ▣ A blokkok (alprogramok, blokk utasítások) lokális változói
 - ha nem static...
- ▣ Automatikusan jönnek létre és szűnnek meg a blokk végrehajtásakor
 - A hatókörhöz igazodó élettartam
- ▣ Rekurzió: több példány is lehet belőlük

Kivételek

- A végrehajtási verem kiürítése
 - stack trace
- Vezérlésátadás kivételes esetek kezelésénél
- Kivétel: eltérés a megszokottól, az átlagostól
 - Programhiba (dinamikus szemantikai hiba)
pl. tömb túlindexelése
 - Speciális eset jelzése
- Kiváltódás, terjedés, lekezelés, definiálás, kiváltás

Nyelvi eszköz kivételkezelésre

- ▣ A modern nyelvekben gyakori
 - Már a COBOL-ban és a PL/I-ben is ('60)
 - Ada, C++, Java, Delphi, Visual Basic...
 - Sok „apró” különbség
- ▣ Előtte (pl. Pascal, C)
 - Globális változók, speciális visszatérési értékek jelezték a kivételt
 - A kivétel észlelése: elágazás
 - Kusza kód

Szétválasztás

- ▣ Az átlagos és a kivételes szétválasztandó
- ▣ A kivételes események kezelését külön adjuk meg
- ▣ Elkerülhető a kusza kód
- ▣ Megbízható, mégis olvasható, karbantartható
- ▣ Aspektuselvű programozás

Előre definiált kivételek az Adában

- ▣ Constraint_Error (és Numeric_Error)
- ▣ Program_Error
- ▣ Storage_Error
- ▣ Tasking_Error

- ▣ A szabványos könyvtárak által definiált kivételek
(pl. Ada.IO_Exceptions.End_Error)

Constraint_Error

▣ Megszorítás megsértése

```
procedure CE is
    I: Natural := 100;
begin
    loop
        I := I-1;
    end loop;
end CE;
```

Program_Error

▣ Hibás programszerkezet

```
procedure PE is
    generic procedure G;
    procedure P is new G;
    procedure G is begin null; end;
begin
    P;
end PE;
```

Storage_Error

□ Nincs elég memória

```
procedure SE is
  type T( N: Positive := 256) is record
                                     H: Positive;
                                     Str: String(1..N);
  end record;
  V: T;
begin
  ...
end SE;
```


Kivételek definiálása

- ▣ A C++ esetében bármi lehet kivétel
- ▣ A Java esetében egy speciális osztály és leszármazottai
- ▣ Az Ada esetében spec. nyelvi konstrukció
Hibás_Fájlnév: exception;
Üres_A_Verem, Tele_A_Verem: exception;

Kivétel kiváltódása és terjedése

- Egy utasítás végrehajtása közben váltódhat ki („fellép”)
- A hívottból a hívóba terjed
 - A hívottban is fellép a hívás helyén
- A végrehajtási verem mentén
- Dobáljuk ki az aktivációs rekordokat
- Ha a verem kiürül, leáll a program

Kivételek terjedése és a blokkok

▣ Alprogramok

- „Dinamikus tartalmazás”
- Ha egy meghívott alprogramban fellép és nem kezeljük le, akkor fellép a hívó blokkban is a hívás helyszínén

▣ Blokk utasítás

- „Statikus tartalmazás”
- Ha egy blokk utasításban fellép és nem kezeljük le, akkor fellép a tartalmazó blokkban is a tartalmazás helyszínén

Kivételek kezelése

- C++ esetén: try-catch összetett utasítás
- Ada: blokk kivételkezelő része
 - alprogram és blokk utasítás is lehet
 - a végén opcionális kivételkezelő rész
 - benne kivételkezelő ágak

Kivételkezelő rész alprogramban

```
procedure A ( S: in out Stack; N: in out Natural ) is
    X: Positive;
begin
    Push(S, N);
    Pop(S, X);
exception
    when Constraint_Error => Push(S,N);
                                N := 1;
    when Tele_A_Verem => null;
end A;
```

Kivételkezelő rész blokk utasításban

```
declare
    N: Positive := Lnko(64*43, 975);
begin
    Put( N );
    Get( N );
    ...
exception
    when Constraint_Error => ...
end;
```

Kivételkezelő ágak

- Egy ággal több, különböző kivételt is lekezelhetünk

```
exception
```

```
    when Name_Error =>
```

```
        Put_Line("Hibás fájlnevet adott meg!");
```

```
    when End_Error | Hibás_Formátum =>
```

```
        Close(Bemenet_fájl);
```

```
        Close(Kimenet_fájl);
```

```
        Put_Line("A fájl szerkezete nem jó.");
```

```
end;
```

when others =>

- ▣ Az összes fellépő kivételt lekezezi
- ▣ Azokat is, amelyek deklarációja nem látható
- ▣ Veszélyes, de néha kell
when others => null;
- ▣ Az utolsó ág kell, hogy legyen

exception

```
when Constraint_Error => null;
```

```
when others => Put_Line("Nem Constraint_Error.");
```

```
end;
```


Nem deklarált kivételek és az others

```
declare
    procedure A is
        X: exception;
    begin ... end;
begin
    A;
exception
    when others => ...
end;
```

Kivételkezelő rész keresése

- A kivétel terjedése mentén
- Alprogram: hívó blokkban,
Blokkt utasítás: tartalmazó blokkban
- A lekezelt kivétel nem terjed tovább
 - Hacsak újra ki nem váltjuk...
- Ha nem találunk megfelelő kivételkezelőt
 - A végrehajtási vermet kiürítjük
 - A program hibaüzenettel végetér

Ha egy blokkban kivétel keletkezik

- Az utasítások végrehajtása közben:
 - A blokkban lekezelhetjük egy kivételkezelő részben (a blokkban „lép fel”)
- A deklarációk kiértékelése, vagy a kivételkezelő rész végrehajtása közben:
 - Csak a hívóban/tartalmazóban kezelhető le (ott „lép fel”)

Kivétel deklarációk kiértékelésekor

```
declare
```

```
    V: T:= F(10); ← a tartalmazó blokkban!
```

```
    ...
```

```
begin
```

```
    ...
```

```
exception
```

```
    ...
```

```
end;
```

Kivételek kiváltása

- ▣ C++, Java: throw utasítás
- ▣ Ada: raise utasítás
- ▣ Előre definiált és programozó által definiált kivétel is kiváltható
- ▣ Kivételes szituáció kialakulásának jelzése

Ha nem tudok elvégezni egy műveletet (1)

```
generic
  type Element is private;
package Stacks is
  ...
  Stack_Empty, Stack_Full: exception;
  ...
  function Top ( S: Stack ) return Element;
    -- Top can raise Stack_Empty
  ...
end Stacks;
```

előfeltétel

Ha nem tudok elvégezni egy műveletet (2)

```
generic
  type Element is private;
package Stacks is
  ...
  Stack_Empty, Stack_Full: exception;
  ...
  function Top ( S: Stack ) return Element;
    -- Top can raise Stack_Empty
  ...
end Stacks;
```

csak megjegyzés

előfeltétel

Ha nem tudok elvégezni egy műveletet (3)

```
package body Stacks is
  ...
  function Top ( S: Stack ) return Element is
  begin
    if Is_Empty(S) then raise Stack_Empty;
    else
      ...
    end if;
  end Top;
  ...
end Stacks;
```

az előfeltétel
ellenőrzése

Mire használom a kivételeket?

- Kivételes eset *jelzésére*
 - Előfeltétel nem teljesült
 - Valami elromlott menet közben
(pl. hálózati kapcsolat megszakadt)
- Kivételes eset *lekezelésére*
 - Tovább futhasson az alkalmazás
 - Kilépés előtt valamit csináljak még
(log-olás, erőforrások elengedése)

Kivétel újrakiváltása

▣ Biztonságos menekülés

```
declare
```

```
    ...
```

```
begin
```

```
    ...    -- adatok kiírása az F fájlba
```

```
exception
```

```
    when others => Close(F); raise;
```

```
end;
```

Kivétel lecserélése

```
package body Stacks is
```

```
...
```

```
function Top ( S: Stack ) return Element is
```

```
begin
```

```
    return S.Data( S.Stack_Pointer );
```

```
exception
```

```
    when Constraint_Error => raise Stack_Empty;
```

```
end Top;
```

```
...
```

```
end Stacks;
```

Szétválasztás

ha üres

Kivétel, mint vezérlési szerkezet

```
type Napok is (Hétfő, Kedd, Szerda, ... Vasárnap);  
  
function Holnap ( Ma: Napok ) return Napok is  
begin  
    return Napok'Succ(Ma);  
exception  
    when Constraint_Error => return Napok'First;  
end;
```

Többletinformáció a kivételes esetről

- C++ és Java esetében a kivétel plussz információkat tárolhat, hordozhat
- Ada 95: kivételpéldányok
 - korlátozott lehetőség
 - csak szöveges adat (hibaüzenet)
 - az Ada.Exceptions csomag

Az Ada.Exceptions csomag

```
with Ada.Exceptions; use Ada.Exceptions;
...
exception
  when E: Stack_Full =>
    Close(F);
    Put_Line( Standard_Error, Exception_Message(E) );
    Raise_Exception(
      Exception_Identity(E),
      Exception_Message(E) & "(logged)"
    );
...
```

Kivétel sablon paramétereiként (1)

```
with Ada.Exceptions; use Ada.Exceptions;
generic
  type Elem is limited private;
  type Index is (<>);
  type Tömb is array (Index range <>) of Elem;
  with function "<" ( A, B: Elem ) return Boolean is <>;
  Üres_Tömb: Exception_Id := Constraint_Error ' Identity;
function Max ( T: Tömb ) return Elem;
```

Kivétel sablon paramétereiként (2)

```
function Max ( T: Tömb ) return Elem is
begin
  if T ' Length = 0 then
    Raise_Exception( Üres_Tömb, "Üres a tömb" );
  else declare
    Mh: Index := T ' First;
    begin
      ...
    end;
  end if;
end Max;
```



opcionális, itt
most nem
szerencsés

Kivétel sablon paramétereként (3)

```
with Max;
procedure Max_Demo is
  type T is array (Integer range <>) of Float;
  Baki: exception;
  function Float_Max is new Max(Float,Integer,T);
  function Float_Min is new Max( Float, Integer, T, ">",
                                Baki ' Identity );

  X: T(1..0);  -- üres tömb
begin
  ...
```

Kivételek elnyomása

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Szoroz is
  pragma Suppress(Range_Check);
  I: Integer range 1..10000 := 1;
begin
  loop
    Put_Line(Integer'Image(I));
    I := I * 10;
  end loop;
end;
```