Imperativ programmering

1DL126 3p

Föreläsning 3

Imperativa paradigmer

- Ostrukturerad programmering
- Strukturerad programmering
  - Procedurell programmering
  - Objektorienterad programmering
    - Klassbaserad programmering
    - Prototypbaserad programmering
    - Konceptorienterad programmering
  - Aspektorienterad programmering
  - Attributorienterad programmering
  - ... och så vidare ...

Procedurell programmering

Bryter upp problemen i mindre delar och löser var del för sig med hjälp av procedurer.

- Funktioner, delprogram (sub program), rutiner, metoder.

Procedurell programmering

Procedurer

- Kan anropas när som helst under en programkörning.
- Kan anropas av vilken kod som helst - även av andra procedurer eller av proceuren själv (rekursion).

Procedurell programmering

Fördelar:

- Återanvända kod.
- Lättare att hålla ordning på kontrollflödet än med goto.
- Möjligheten att skriva modulär kod.
- Proceduren blir en fristående kodbit med ett enkelt gränssnitt.

Procedurell programmering

För att betraktas som ett procedurellt programmeringsspråk ska språket ha stöd för konceptet procedur och en syntax för att definiera den.

- **Ada**, ALGOL, BASIC, **C**, C++, ColdFusion, COBOL, Component Pascal, D, Delphi, ECMAScript (ActionScript, DMDScript, **JavaScript**, JScript), Forth, **Fortran**, Lasso, Linoleum, **Maple**, Mathematica, **MATLAB**, Modula-2, Oberon, Occam, M, **Pascal**, Perl, **PHP**, PL/C, Python, PL/I, Rapira, VBScript

FORTRAN

FORTRAN - FORmula TRANslation

- John Backus (IBM) 1952 - 54
- Första högrenivåspråket som blev allmänt använt.
- Mycket kontroversiellt!
  - "*Kompilerande språk är ineffektiva!*"

FORTRAN

- Används fortfarande för högprestandaberäkningar.

1966	**FORTRAN 66**
1978	**FORTRAN 77**
1992	**Fortran 90**
1997	Fortran 95
Nov 2004	Fortran2003

FORTRAN

- Fortran77 är i storlek som C
  - Endast statisk minnesallokering.
  - Inga pekare.
  - Endast fördefinierade typer.
- Implicit typning (arv från F66)
  - Variabelnamn som inleds med bokstäverna i,j..n är heltal, annars flyttal.
- Fast kodform (hålkort!)
  - Alla rader är maximalt 72 tecken.
  - Positionerna 1 till 5 är reserverade för rubriker.
  - Position 6 markerar "continuation".
  - Positionerna 7 till 72 FORTRAN-kod.

FORTRAN

- F90, F95 - modernt programmeringsspråk.
  - Egna typdefinitioner.
  - Dynamisk minneshantering.
  - Pekare.
  - Överlagring av operatorer.
- Fortran95 “bugfix” för Fortran90.
- Mest kommersiella kompilatorer men ett par open source-varianter finns.
  - http://gfortran.org
  - http://www.g95.org

FORTRAN

Tidiga versioner låg nära hårdvaran.

- Endast register - ingen stack.
- Minnet betraktas som en array.

Datatyper:

- Integer, Real, Complex, Double, Bool
- Arrayer, Strängar
- Filer

FORTRAN

- Ett program består av ett huvudprogram och ett antal underprogram. De kompileras var för sig och länkas samman när programmet körs - Länkningsfel!
- F66 bygger mest på goto
- F77 Erbjuder mer instruktioner för kontrollflöde, tex "conditionals".

FORTRAN

- DO motsvarar "for", dvs den räknar upp ett steg per varv i loopen.
- Ingen "while".
- Underprogrammets namn betraktas som en variabel för returvärde.

FORTRAN

do <variable> = <start>,<stop>,<step>

Ett typo...

do 10 i = 1,100 do 10 i = 1.100

... sägs ha varit orsak till att en rymdfärja exploderat vid start.

FORTRAN

```
      PROGRAM MAIN
        PARAMETER (maxSIZ=99)
        REAL A(MAXsiz)
 10   READ (5,100,END=999) K
100   FORMAT(I5)
            IF (K.LE.0 .OR. K.GT.MAXSIZ) STOP
            READ *,(A(I),I=1,K)
            PRINT *,(A(I),I=1,K)
            PRINT *,'SUM=',SUM(A,K)
            GO TO 10
999   PRINT *, "All Done"
        STOP
        END
```

FORTRAN

```
C UNDERPROGRAM SUMMA
      FUNCTION SUM(V,N)
        REAL V(N)
        SUM = 0.0
        DO 20 I = 1,N
          SUM = SUM + V(I)
 20       CONTINUE
        RETURN
        END
```

FORTRAN / Pascal

```
C UNDERPROGRAM SUMMA
      FUNCTION SUM(V,N)
         REAL V(N)
         SUM = 0.0
         DO 20 I = 1,N
            SUM=SUM+V(I)
 20      CONTINUE
         RETURN
         END
```

```
(* UNDERPROGRAM SUMMA *)
FUNCTION SUM(V:ARRAY OF
    REAL; N:INTEGER):REAL;
VAR I : INTEGER;
VAR S : REAL;
BEGIN
   S := 0;
   FOR I := 1 TO N DO
      S := S + V[I];
   SUM := S;
END;
```

Parameteröverföring

Skilj på två viktiga koncept:

- Formella parametrar - Parameternamnen som deklareras i proceduren.
  (V och N - L-value)
- Faktiska parametrar - Värdena som skickas till proceduren.
  (Arrayen i A och storleken i K - R-value)

Parameteröverföring

Vid anropet listas de faktiska parametrarna i samma ordning som de formella parametrarna står i procedurdeklarationen:

```
procedure foo(ett : Integer,
              tva : Character)

foo(48,'Z');
```

Parameteröverföring

Ada: Namngiven parameteröverföring.

```
procedure foo(ett : Integer;
              tva : Character)
foo(tva => 'Z', ett => 48);
```

Standardargument ("default-värden")

```
procedure foo(ett : Integer := 0;
              tva : Character := 'A')
foo(tva => 'Z');
```

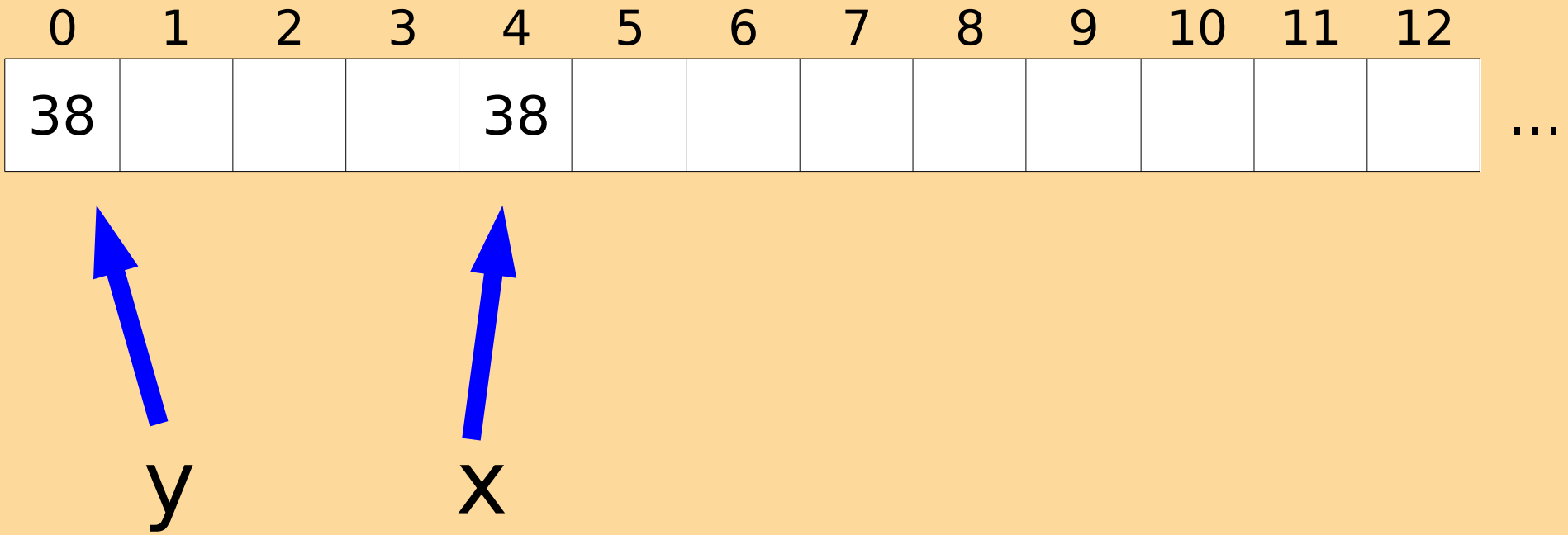
Parameteröverföring

Call-by-value:

- Parametrarna (argumenten) beräknas innan anrop.
- Resultatet kopieras till procedurens egen minnesarea.
- FORTRAN, Pascal, C, Ada, C++, Java (enkla typer)

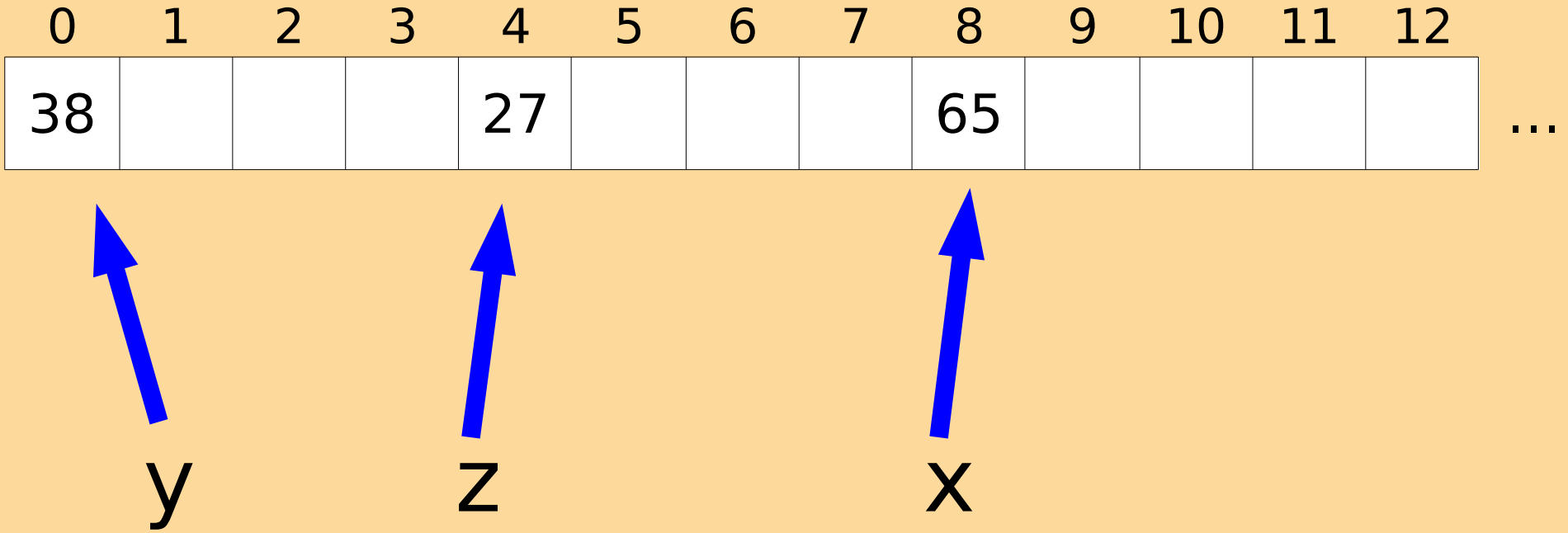
Call-by-value

```
procedure foo(x : Integer)

var y : Integer := 38;
foo(y);
```



Call-by-value

```
procedure foo(x : Integer)

var y : Integer := 38;
var z : Integer := 27;
foo(y + z);
```



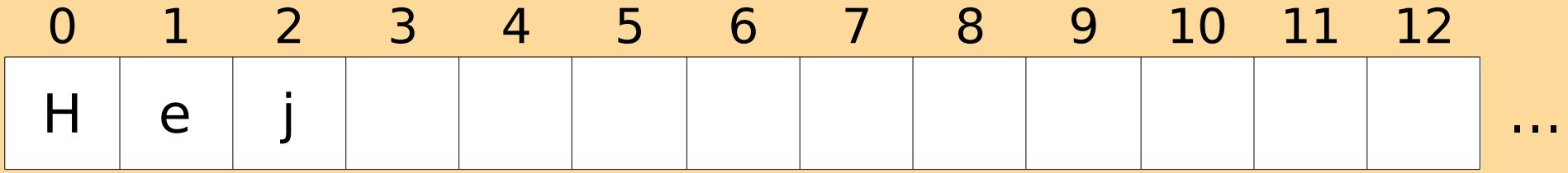
Parameteröverföring

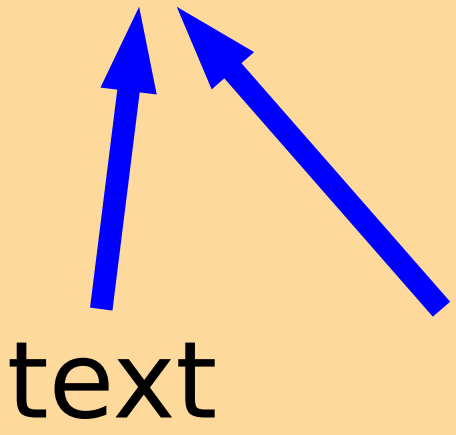
Call-by-reference:

- Den formella parametern binds till samma minnesposition som den faktiska parametern använder. (Aliasing)
- Den faktiska parametern måste ha ett L-värde - Det måste finnas en plats i minnet att referera till.

Call-by-reference

```
void foo(String str)

String text = "Hej";
foo(text);
```

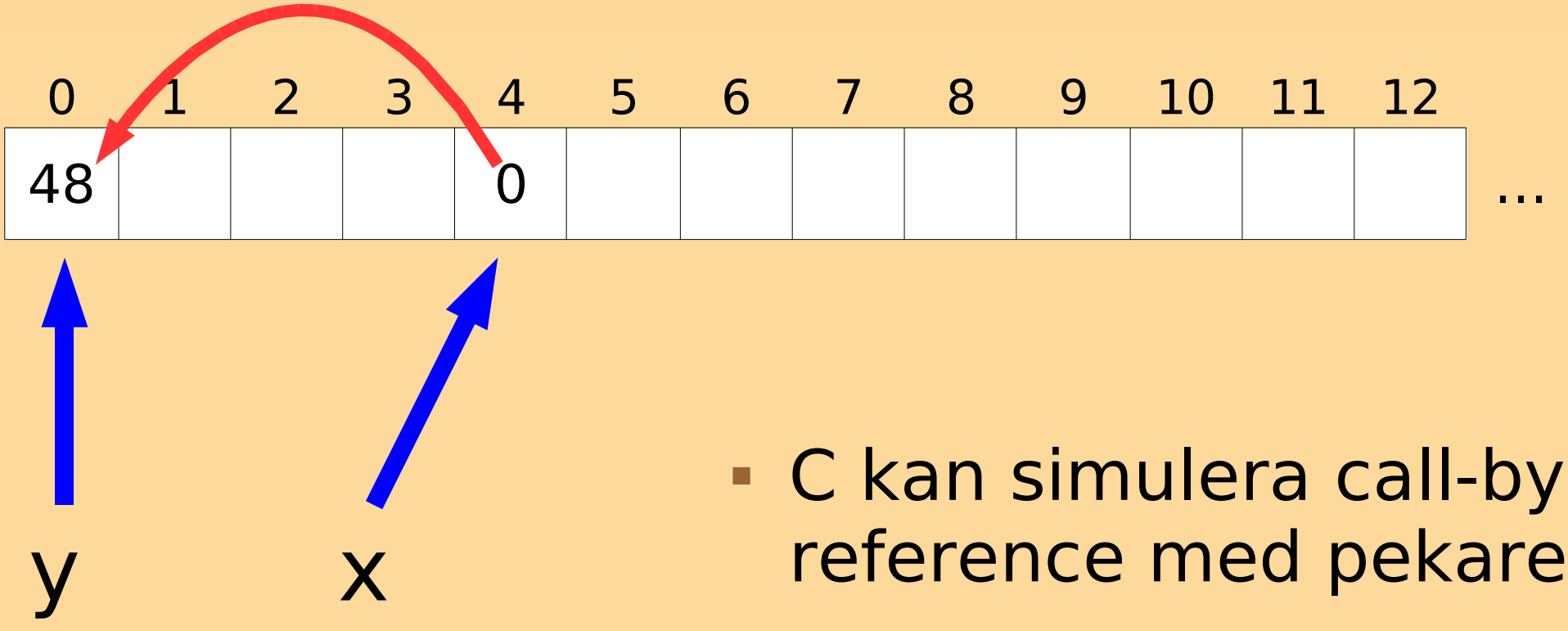




- Java använder alltid call-by-reference för objekt.

Call-by-reference

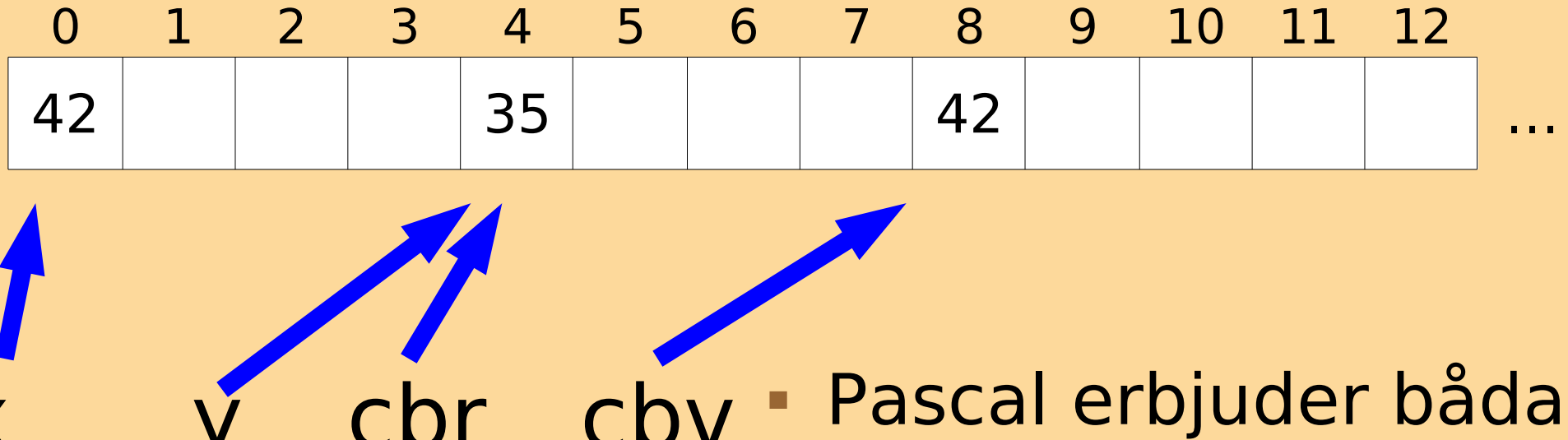
```
void foo(int* x)

int y = 48;
foo(&y);
```



- C kan simulera call-by-reference med pekare.

Call-by-reference

```
procedure foo(cbv : Integer;
              var cbr : Integer)

var x : Integer := 42;
var y : Integer := 35;
foo(x,y);
```



- Pascal erbjuder båda varianterna (`var`)

Parameteröverföring

- Call-by-reference är att föredra när man skickar stora datastrukturer som argument - det tar tid att kopiera.
- Call-by-value är effektivare för enkla typer och små datastrukturer som används väldigt mycket i proce-duren. Att dereferera en pekare kräver en extra instruktion.

Parameteröverföring

Call-by-reference kan innebära en säkerhetsrisk. Om proceduren ändrar i datat så kommer förändringen att överleva proceduren.

Call-by-name

Call-by-name:

- De formella parametrarna byts textuellt ut mot de faktiska. Textuellt = Ordagrant textutbyte.
- Algol 60, Haskell, Makron i C

Call-by-name

C-makro byter ut texten på riktigt:

```
#define add(x,y) x+y
z = add(45,6);
```

```
z = 45+6;
```

Implementationen under ytan i t.ex. Haskell använder funktionspekare.

Returvärden

- Funktioner returnerar värden.
- Procedurer returnerar INTE värden. (undantaget call-by-reference)
- Pascal erbjuder både procedurer och funktioner.
- C och Java m.fl. har bara funktioner men simulerar procedurer med void.

Omgivning - Scope

- En omgivning är en del av programmet där procedurer och variabler är tillgängliga.
- Omgivningar kan vara nästlade. Det som definieras i den yttre omgivningen är tillgängligt i den inre, men inte tvärt om!

Omgivningar

C++

```
namespace N {                    // namespace scope
   class C {                     // class scope
      void f (bool b) {          // function scope
         if (b) {                // unnamed scope
            ...
         }
      }
   };
}
```

Namnrymd: Grupperar namn

Klass: Medlemsvariabler och funktioner

Funktion: Körbar kod

Anonymt: Kan skapas i körbar kod

Omgivningar

```
class Javaklass {
    private int a;
    public int b;
    void foo(int c) {
        int d;
    }
}
```

`a`: Tillgänglig i hela klassen
`b`: Tillgänglig i hela klassen och i andra klasser
`c` & `d`: Endast lokalt tillgängliga i metoden `foo`

Omgivningar

Skuggning:

```
void foo() {
    int x = 42;

    {
        int x = 38;
        print(x);
    }
    print(x);
}
```

Programmet skriver ut:

38

42

Den lokala deklarationen av x skuggar den tidigare.

Omgivningar

- Statisk omgivning - Definieras av programmkoden. Omsluts normalt med { ... } alt. BEGIN ... END.
- Dynamisk omgivning - Den senast deklarerade versionen gäller.

Omgivningar

```
int x = 0;
int f () { return x; }
int g () { int x = 1; return f(); }
```

Statisk omgivning: g returnerar 0.
Dynamisk omgivning: g returnerar 1.

Omgivningar

- Dynamisk omgivning farligt och ovanligt.
- Valbart i t.ex. Common LISP och Perl.

Obligatorisk uppgift

Skriv ner det viktigaste / det du minns bäst från dagens föreläsning