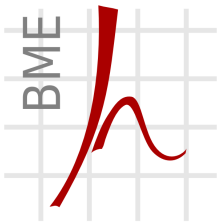
Fordító –Optimalizálás

Kód visszafejtés.



Híradástechnikai Tanszék

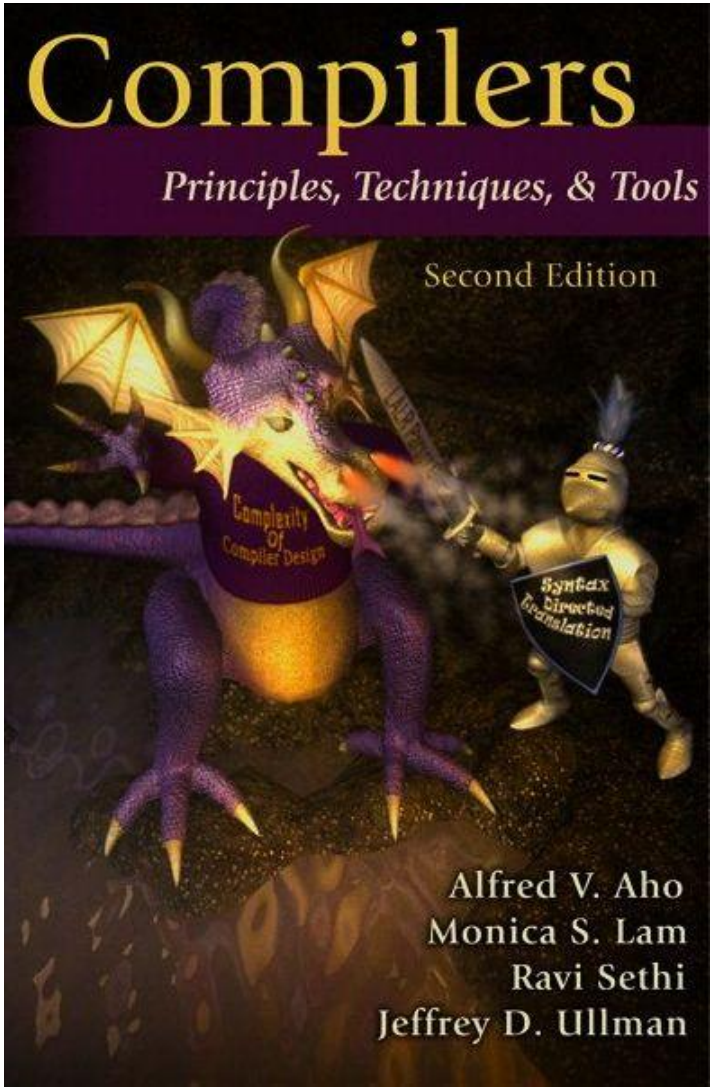
Izsó Tamás

2012. szeptember 27.

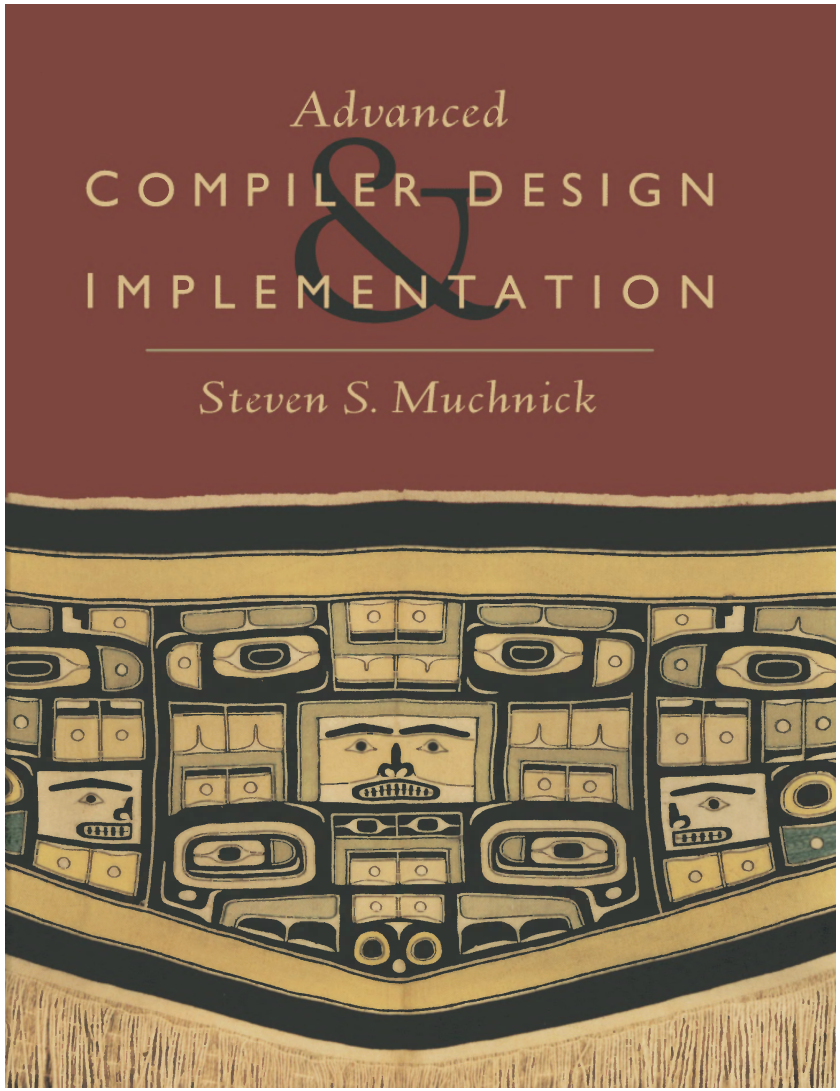
Section 1

Fordító részei

Irodalom


Compilers
Principles, Techniques, & Tools
Second Edition
Complexity Of Compiler Design
Syntax Directed Translation
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman

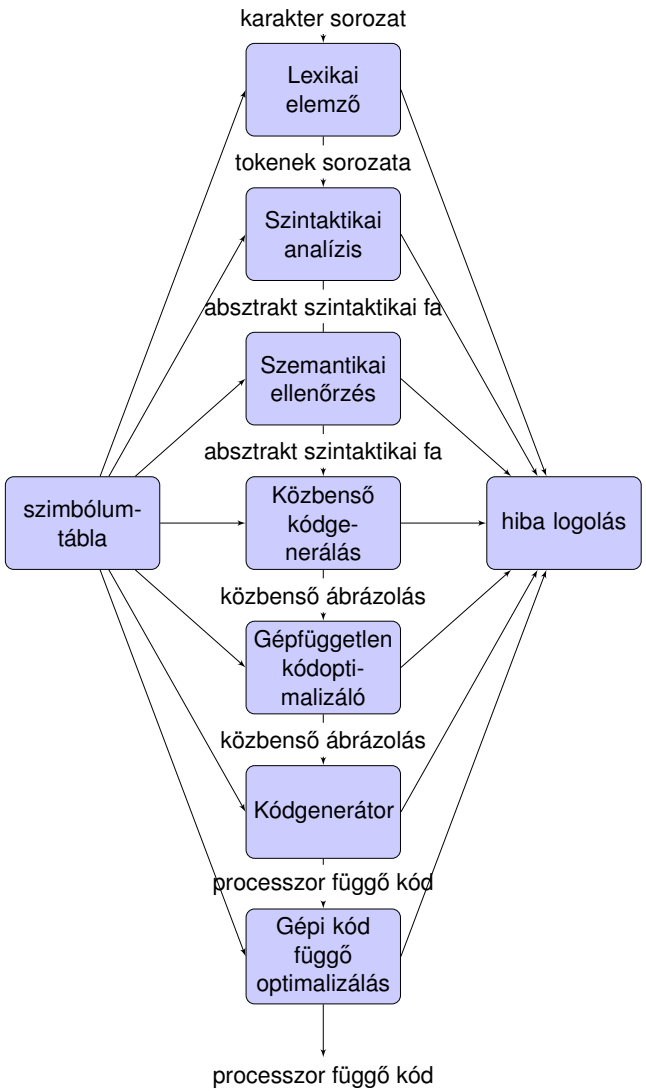
Irodalom


Advanced
COMPILER DESIGN
&
IMPLEMENTATION
Steven S. Muchnick

Miért kell ismerni a fordítót?

- általában a fordító által generált kódot kell visszafejteni;
- ha ismerjük a fordító által alkalmazott kódtranszformációt, hamarabb rájöhetünk az eredeti tartalomra;
- tesztelni kell a fordítót;
- **mert a fordító is azokat az algoritmusokat használja a kód elemzéshez, melyeket a modern decompiler-ek.** (Miért, amikor a fordító rendelkezésére áll az eredeti forrás?)

Fordító részei

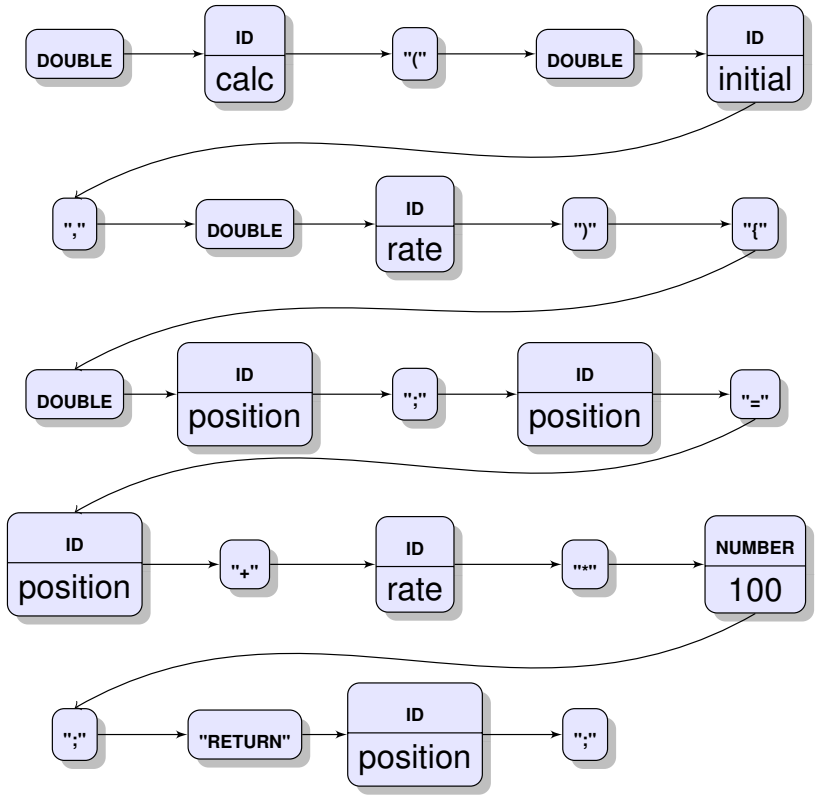


Lexikai elemző

```
1  double calc( double initial, double rate )
2  {
3     double position;
4     position = initial+rate*100;
5     return position;
6  }
```

Lexikai elemző

```
double calc( double initial, double rate ) \n
{ \n double position; \n position = initial+
rate*100; \n return position; \n } \n
```



Token

- token – szintaktikai kategória, csoport;
  - élő nyelvben: ige, főnév, melléknév, tárgy;
  - programozási nyelvekben: egész konstans, valós konstans, string, azonosító, operátor, kulcsszó, írásjelek;
- lexéma – konkrét megvalósulása egy tokennek;
- attribútum – tokenhez rendelt tulajdonság, érték
  - elmaradhat (pl. T_FOR)
  - típus
  - egész konstans , valós konstans , sztring értéke;
  - méret;
  - élettartam;
  - láthatóság
  - stb.
- nem token: szóköz, megjegyzés

Blokk struktúra

```
int main()
{
    int a=1;
    int b=1;
    {
        int a=3;
        printf("%d %d\n",a,b);
    }

    {
        int b=4;
        printf("%d %d\n",a,b);
    }
    printf("%d %d\n",a,b);
    return 0;
}
```

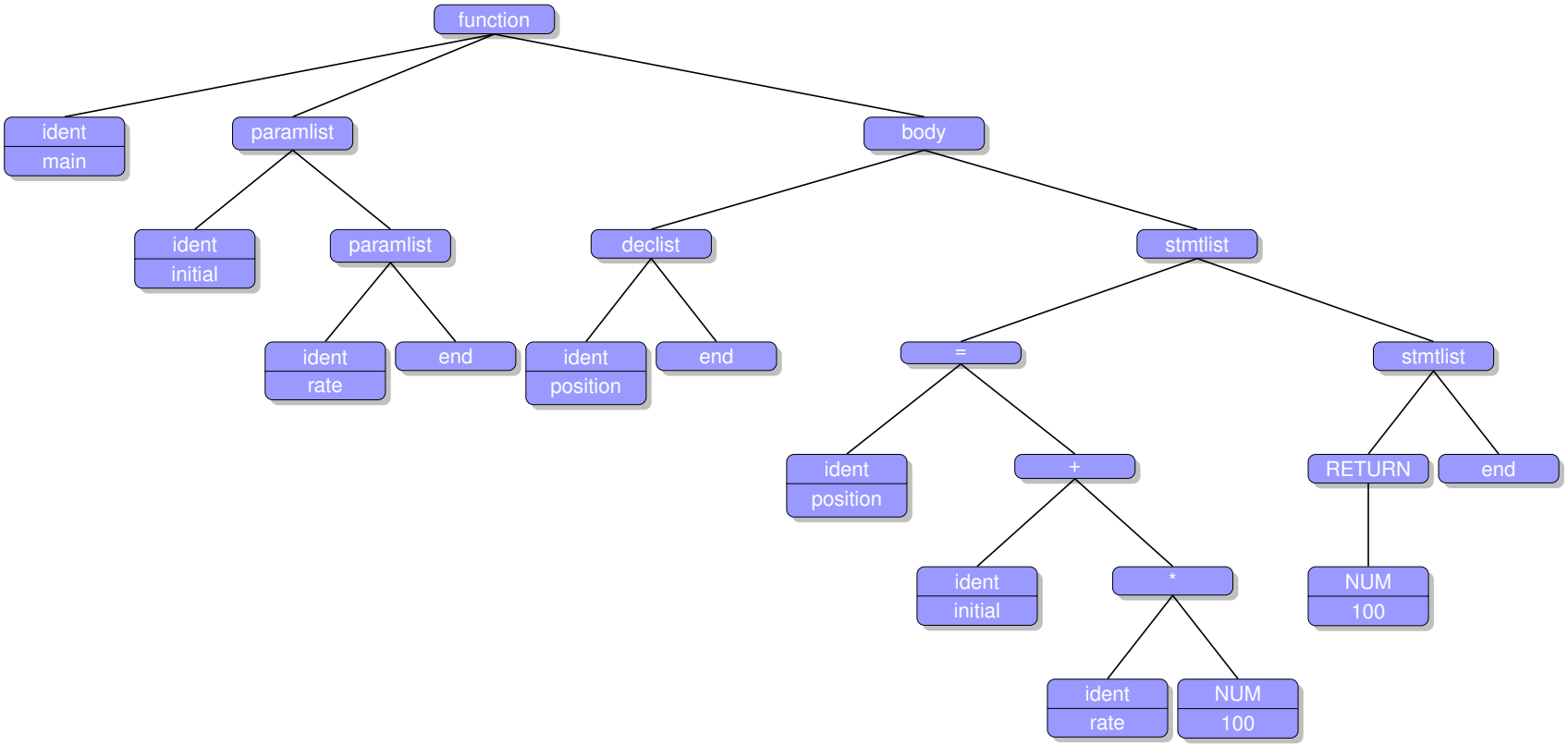
Szimbólum tábla

- a tokenizálás során keletkezett azonosítókat, és az azokhoz kapcsolódó attribútumokat tárolja;
- megőrzi a blokkstruktúrát;
- OOP esetén a származtatást;
- gyorsan lehet benne keresni;
  - sokszor a kulcsszavakat is tartalmazza (pl. for, while).

Szintaktikia analízis

- lexikai elemzés után tokenek sorozatát kapjuk;
- cél a program szerkezetének az értelmezése a környezetfüggetlen nyelvtani szabályok (context-free grammar CFG) alapján (műveletek, precedencia);
- cél a hibák felderítése;
- absztrakt szintaxisfa (abstract syntax tree – AST) elkészítése.

Szintaktikia analízisre példa



Szemantikai analízis célja

- Olyan hibák felderítése, amit a CFG-ben nehéz, vagy nem lehet megadni.
- További információk gyűjtése.

```
1  extern void f(int n);
2  double calc( )
3  { int *p1, *p2;
4     int i,j;
5     ....
6     p1 = p1 + i;     /* OK */
7     p1 = p1 + p2; /* Hiba*/
8     if( f(*p1) )        /* Hiba f() void */
9     {
10        i = j[2];      /* Hiba */
11        i = j[p1];      /* OK */
12    }
13    /* Nincs visszatérési érték */
14 }
```

Miért használunk közbenső kódot (Intermediate reprezentáció)

AST

- előny – gépfüggetlen;
- hátrány – túl magaszintű.

Assembly

- előny – alkalmas az optimalizálásra;
- hátrány – gépfüggő;
- hátrány – minden egyes processzorra újra kellene írni az optimalizálót.

IR

- előny – alkalmas az optimalizálásra;
- előny – gépfüggetlen.

Intermediate Language (IL)

IL a közbenső ábrázolás (IR) nyelvtani szabályait tartalmazza

- minden fordító saját IL-t használ;
- IL = magasszintű assembly nyelv
  - felhasználható regiszterek száma nincs korlátozva;
  - assembly nyelv szintű vezérlési szerkezetek
  - gépi utasításokat használnak, de egyes utasítások lehetnek magasszintűek is
    - pl. call több assembly utasításra fordul
    - legtöbb IL utasítás egy gépi utasításra fordítható

Intermediate Reprezentáció típusai

- egy operandusú (stack szervezésű gépek pl. JVM);
- két operandusú $x \leftarrow x < op > y$;
- három operandusú $z \leftarrow x < op > y$;

push 2	t1 = 2	t1 = 2
push y	t2 = y	t2 = y
mul	t1 = t1 $*$ t2	t3 = t1 $*$ t2
push x	t3 = x	t4 = x
sub	t3 = t3 $-$ t1	t5 = t4 $-$ t1
egy operandusú	két operandusú	három operandusú

$x - 2 * y$

Példa 3 operandusú IL-re

```
int a;                    |  t0 = b + c;
int b;                    |  a = t0 + d;
int c;                    |  t1 = a * a;
int d;                    |  t2 = b * b;
a = b + c + d;            |  b = t1 + t2;
b = a * a + b * b         |
```

temporális változók: t0, t1, t2

3 operandusú Intermediate Language

- egy utasításnak maximum 3 operandusa lehet;
- összetett kifejezésnél temporális változókat kell bevezetni;
- van két operandusú utasítás, pl. t1 = 5;

Intermediate Language utasításkészlete

Értékadás:

- var = constant;
- var = string;
- var1 = var2;
- var = label;

Intermediate Language utasításkészlete

Aritmetikai operátorok:

- var1 = var2 op var3;
- var1 = constant op var2;
- var1 = var2 op constant;
- var = constant1 op constant2;
- operátorok +.−.∗./,%,|,& stb;

Intermediate Language utasításkészlete

Logikai értékek

- 0 – hamis;
- nem 0 – igaz.

Logikai műveletek:

- var3 = var2 == var1;
- var3 = var2 < var1;
- var3 = var2 || var1;
- var3 = var2 && var1;
- a többi logikai kifejezést ezek segítségével kell előállítani.

Intermediate Language

Vezérlésátadó utasítások:

- névvel rendelkező cimke L1:
- Goto label;
- IfZ value Goto label;
- IfZ csak a Goto utasítással együtt fordulhat elő;

Intermediate Language

Függvényhívó utasítás:

- LCall L1 – esetén a hívott függvény címe fordítási időben ismert.
- ACall t1 – t1 függvény címe futáskor áll elő (pl. virtuális függvénytábla).

- LCall L1;
- t1 = LCall L1;
- ACall t1;
- t0 = ACall t1;

Intermediate Language

Függvény definiálás:

- BeginFunc n; – Függvény törzsének a kezdete, ahol n a lokális adatok számára szükséges hely bájtokan.
- EndFunc – függvény vége;
- Return t1 – visszatérés visszaadott értékkel;
- Return – visszatérés a függvényből.

Intermediate Language

Memória címzés:

- t1 = ∗t2
- t1 = ∗(t2+offset)
- ∗t1 = ∗t2
- ∗(t1 + offset) = ∗t2
- offset – negatív vagy pozitív egész érték.

Tömb:

- t1[t2] = t3
- t3 = t1[t2]
- az index operátor a C-ben megismert módon működik.

Példa 3 operandusú IL-re

```
int x;
int y;



while(x<=y) {
        x = x + 2;
}


y = x;
```

```
L0:
    t0 = x < y;
    t1 = x == y;
    t2 = t0 || t1;
    IfZ t2 Goto L1;
    x = x + 2;
    Goto L0
L1:
    y = x;
```

Section 2

Optimalizálás

Miért van szükség optimalizálásra

- AST-ből IR-re való egyszerű áttérés redundanciát okoz;
- egyes részszámításokat
  - fel lehet gyorsítani;
  - közös részeket össze lehet vonni;
  - felesleges részeket ki lehet hagyni
- mert a programozók lusták
  - for ciklusba, vagy a feltétel részbe írnak a ciklus végrehajtása során nem változó kifejezéseket;

A kód optimalizálása kihívást jelent

- Cél:
  - az eredményeket helyességét ne befolyásolja ;
  - a lehető leghatékonyabb IR-t állítsa elő;
  - ne tartson sok ideig.
- Valóság:
  - néha bibát okoz a kód futásánál;
  - sokáig tart a kódgenerálás.
- Legtöbb optimalizálási algoritmusok NP-teljes, ha egyáltalán létezik megoldás.

A program szemamtikáját nem befolyásoló optimalizálás

- felesleges temporális változók megszüntetése;
- fordítási időben ismert konstans kifejezések kiszámítása;
- ciklusban lévő invariáns részek kiemelése;
- ellenpéldaként meg lehet említeni (szemantikát befolyásoló optimalizálás) a buborékos rendezés kicserélése gyorsrendezésre.

Példa IR optimalizálására

```
int x;
int y;
bool b1;
bool b2;
bool b3;

b1 = x + x < y;
b2 = x + x == y;
b3 = x + x > y;
```

```
t0 = x + x;
t1 = y;
b1 = t0 < t1;

t2 = x + x;
t3 = y;
b2 = t2 == t3;

t4 = x + x;
t5 = y;
b3 = t5 < t4;
```

Példa IR optimalizálására

```
int x;
int y;
bool b1;
bool b2;
bool b3;

b1 = x + x < y;
b2 = x + x == y;
b3 = x + x > y;
```

```
t0 = x + x;
t1 = y;
b1 = t0 < t1;

t2 = x + x;
t3 = y;
b2 = t2 == t3;

t4 = x + x;
t5 = y;
b3 = t5 < t4;
```

Példa IR optimalizálására

```
int x;
int y;
bool b1;
bool b2;
bool b3;

b1 = x + x < y;
b2 = x + x == y;
b3 = x + x > y;
```

```
t0 = x + x;
t1 = y;
b1 = t0 < t1;

t2 = x + x;
t3 = y;
b2 = t2 == t3;

t4 = x + x;
t5 = y;
b3 = t5 < t4;
```

Példa IR optimalizálására

```
int x;
int y;
bool b1;
bool b2;
bool b3;

b1 = x + x < y;
b2 = x + x == y;
b3 = x + x > y;
```

```
t0 = x + x;
t1 = y;
b1 = t0 < t1;

t2 = x + x;
t3 = y;
b2 = t0 == t1;

t4 = x + x;
t5 = y;
b3 = t1 < t0;
```

Példa IR optimalizálására

```
int x;
int y;
bool b1;
bool b2;
bool b3;

b1 = x + x < y;
b2 = x + x == y;
b3 = x + x > y;
```

```
t0 = x + x;
t1 = y;
b1 = t0 < t1;

b2 = t0 == t1;

b3 = t1 < t0;
```

Lusta programozó munkájának az optimalizálása

```
while( x < y+ z ) {
    x = x - y;
}
```

```
L0:
        t0 = y + z;
        t1 = x < t0;
        IfZ t1 Goto L1
        x = x - y;
        Goto L0;
 L1:
```

Lusta programozó munkájának az optimalizálása

```
while( x < y+ z ) {
    x = x - y;
}
```

```
L0:
        t0 = y + z;
        t1 = x < t0;
        IfZ t1 Goto L1
        x = x - y;
        Goto L0;
 L1:
```

Lusta programozó munkájának az optimalizálása

```
while( x < y+ z ) {
    x = x - y;
}
```

```
        t0 = y + z;
L0:
        t1 = x < t0;
        IfZ t1 Goto L1
        x = x - y;
        Goto L0;
L1:
```

Lusta programozó munkájának az optimalizálása

```
while( x < y+ z ) {
    x = x - y;
}
```

```
        t0 = y + z;
L0:
        t1 = x < t0;
        IfZ t1 Goto L1
        x = x - y;
        Goto L0;
L1:
```

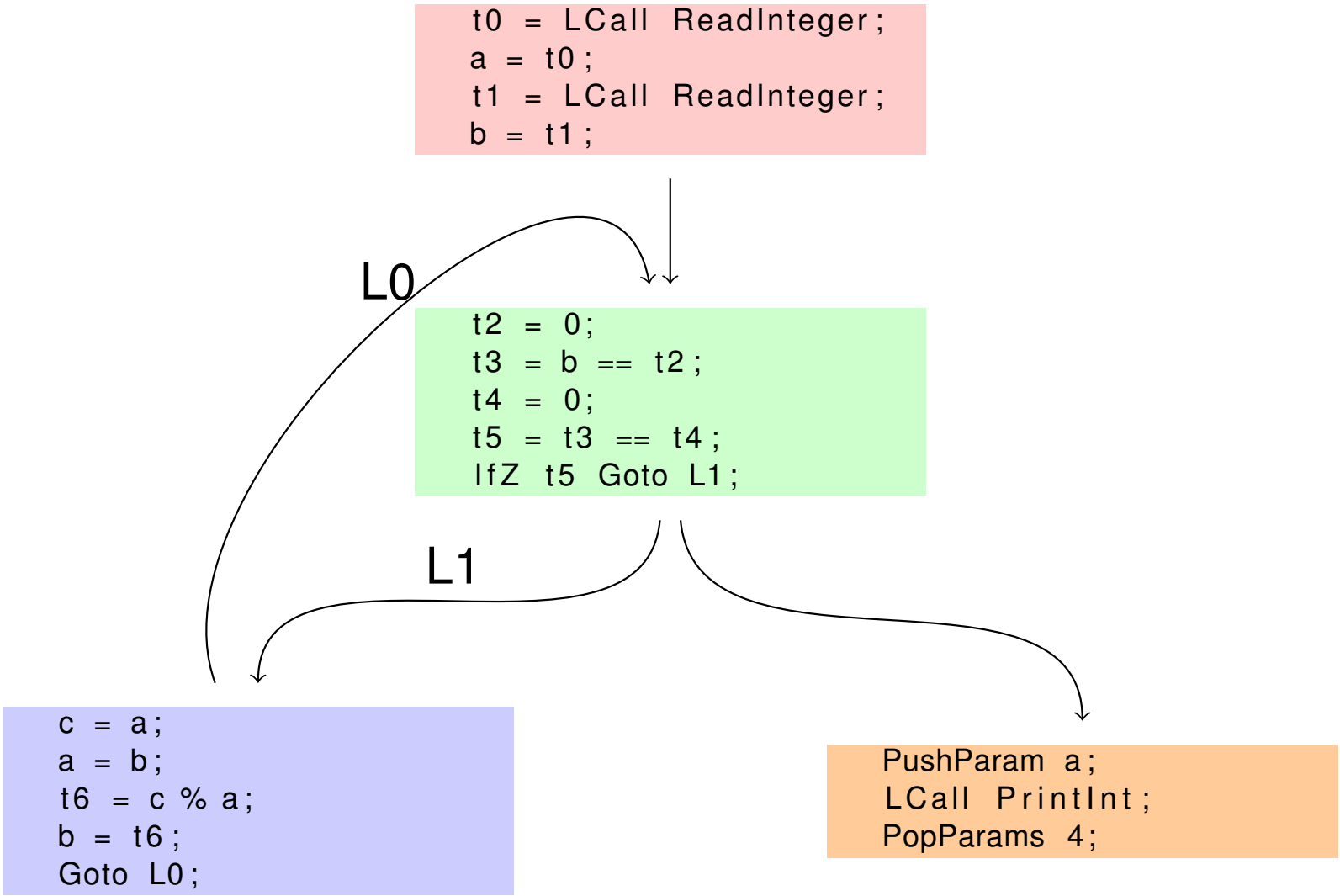
IR megjelenítése

```
BeginFunc 40;
    t0 = LCall ReadInteger;
    a = t0;
    t1 = LCall ReadInteger;
    b = t1;
L0:
    t2 = 0;
    t3 = b == t2;
    t4 = 0;
    t5 = t3 == t4;
    IfZ t5 Goto L1;

    c = a;
    a = b;
    t6 = c % a;
    b = t6;
    Goto L0;
L1:
    PushParam a;
    LCall PrintInt;
    PopParams 4;
EndFunc;
```

IR megjelenítése

```
BeginFunc 40;
    t0 = LCall ReadInteger;
    a = t0;
    t1 = LCall ReadInteger;
    b = t1;
L0:
    t2 = 0;
    t3 = b == t2;
    t4 = 0;
    t5 = t3 == t4;
    IfZ t5 Goto L1;

    c = a;
    a = b;
    t6 = c % a;
    b = t6;
    Goto L0;
L1:
    PushParam a;
    LCall PrintInt;
    PopParams 4;
EndFunc;
```

IR megjelenítése



Alapblokk (Basic Block)

- az IR utasításainak a lineáris sorozata;
- egy belépési ponttal rendelkezik;
- egy kilépési pontja van, és ez a sorozat végén található;
- olyan utasítások sorozata, amelyek a blokk végrehajtása során biztosan végrehajtódnak.
- Alapblokkok kezdete és vége :
  - a program első utasítása az első blokk kezdete;
  - minden olyan hely, amelyre távolról át lehet adni a vezérlést (Goto, LCall, ACall) egy új blokk kezdetét jelenti;
  - ugró utasítás csak a blokk végén lehet;
  - minden blokk vége után egy új blokk kezdődik (kivéve az utolsót).

Vezérlési folyamatgráf – Control Flow Gráf (CFG)

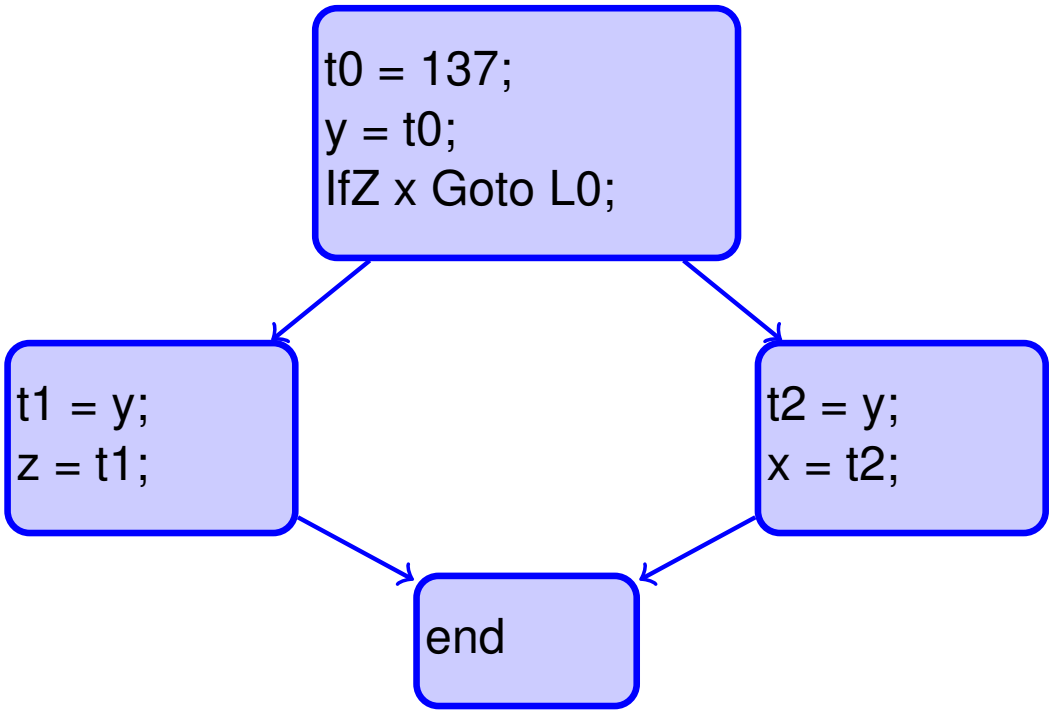
- irányított gráf
- csomópontjai az alapblokkok halmaza;
- az élek a blokkok végpontjából egy másik blokk belépési pontjára mutatnak;
- egy csomópontból több él léphet ki;
- egy csomópontba több alapblokkból is el lehet jutni.

Optimalizálás fajtái

- lokális – egy blokkon belül;
- globális – teljes vezérlési folyamatgráfra.

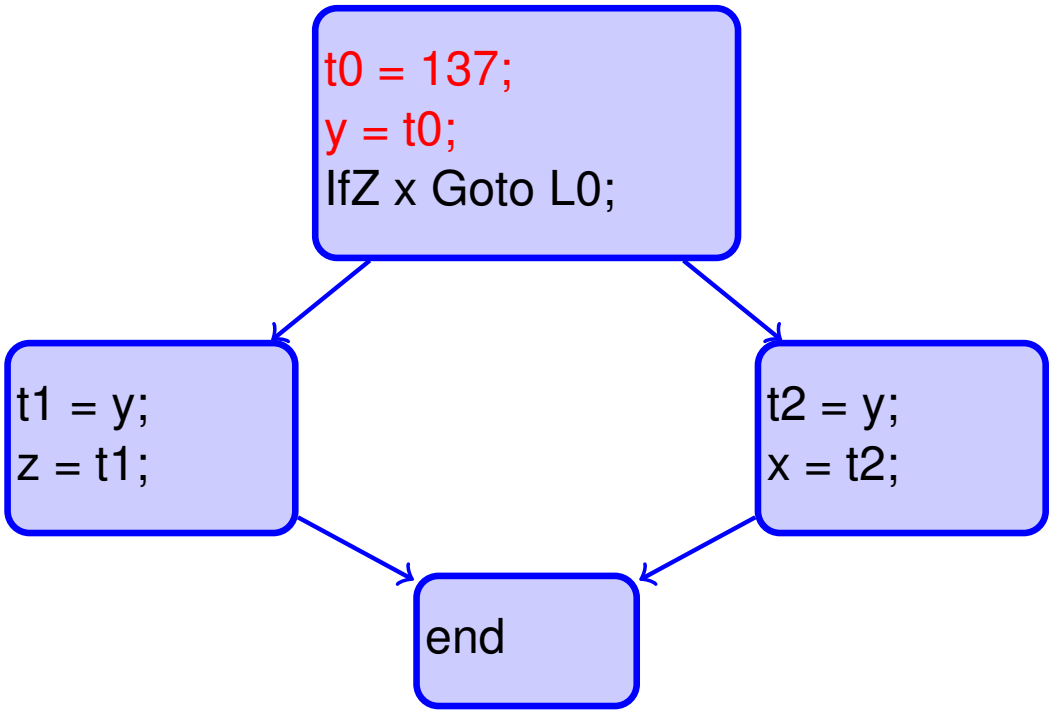
Lokális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



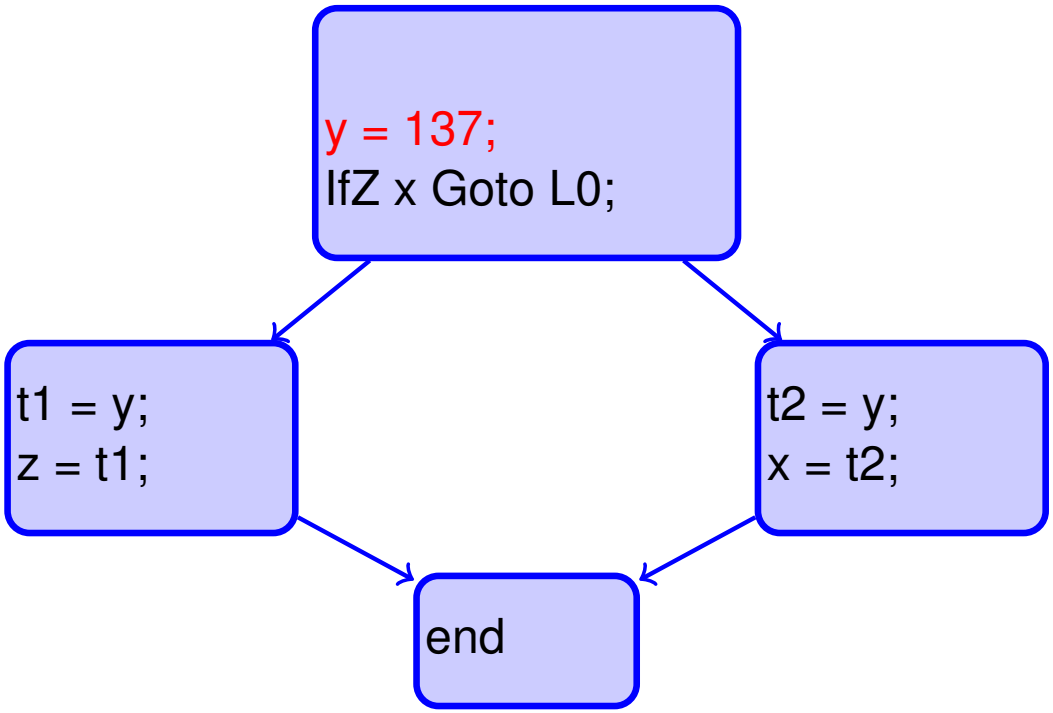
Lokális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



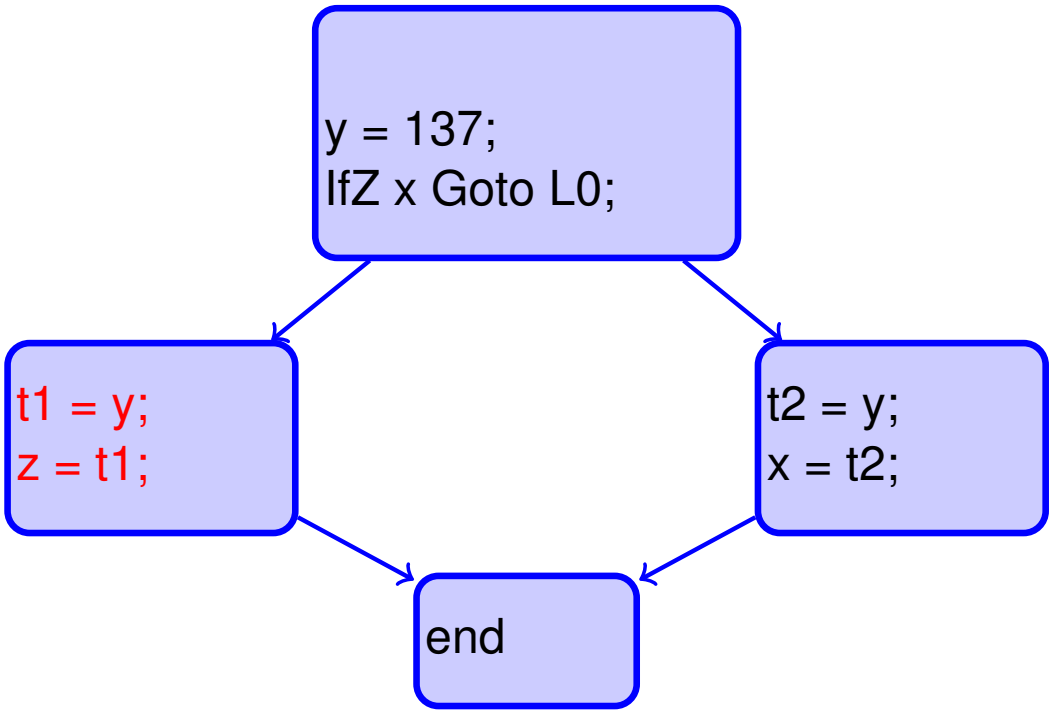
Lokális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



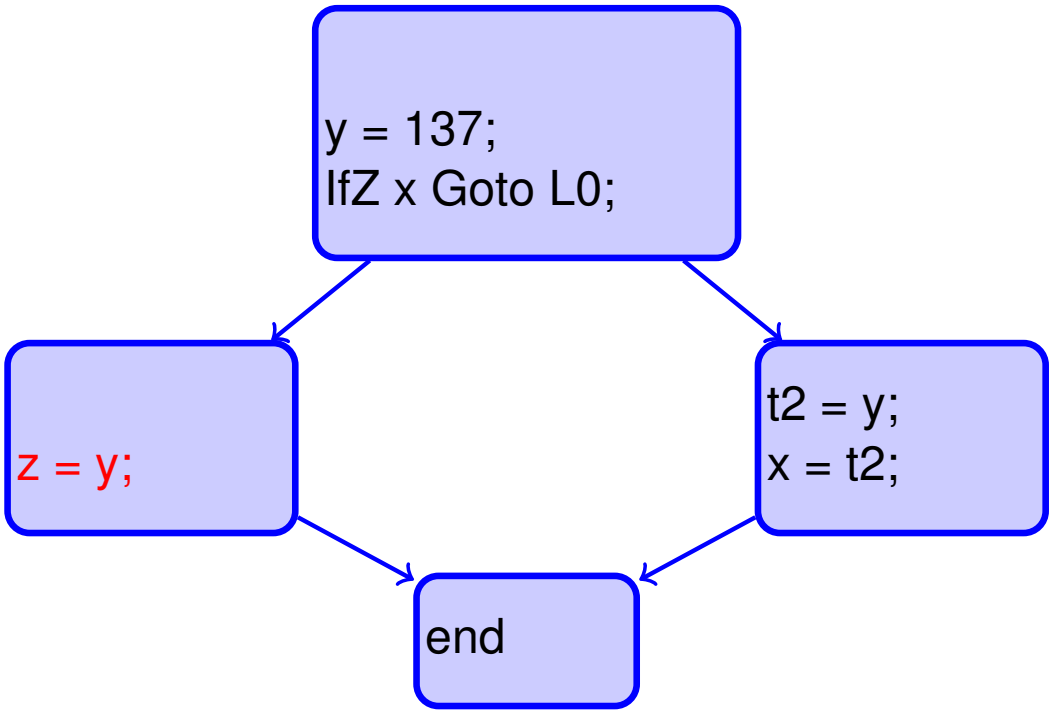
Lokális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



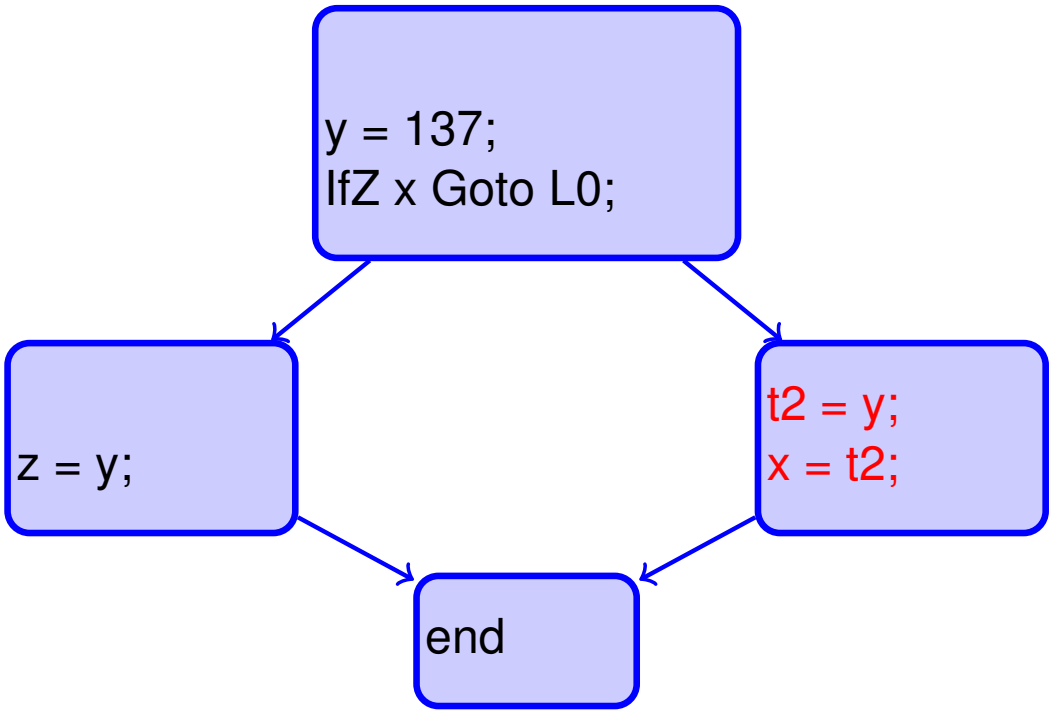
Lokális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



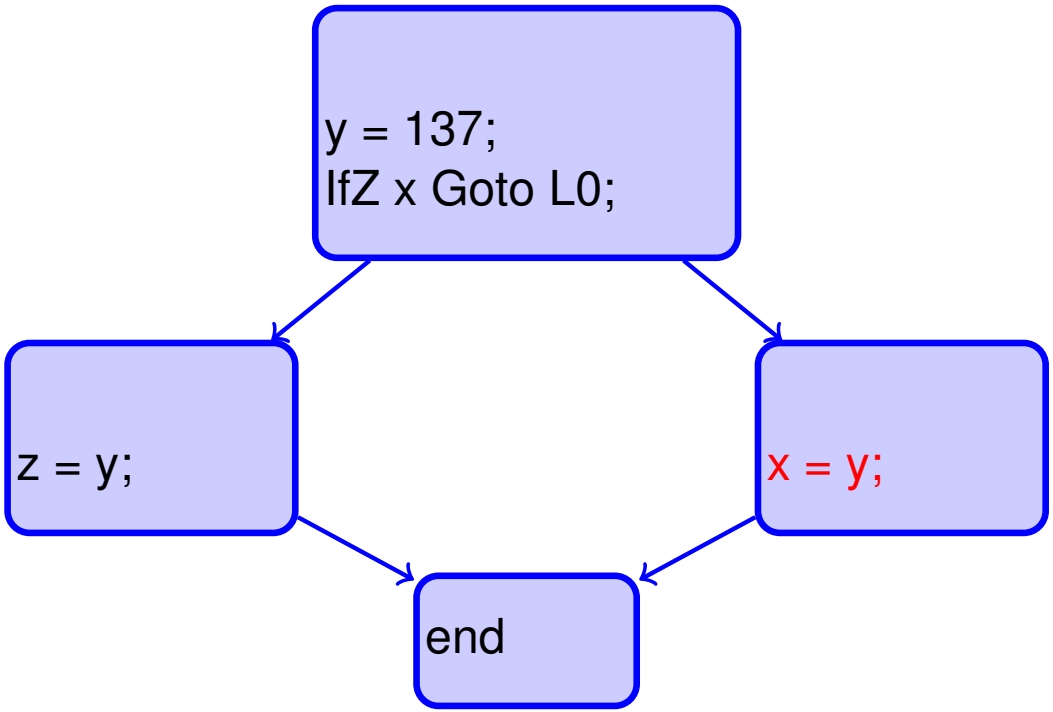
Lokális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



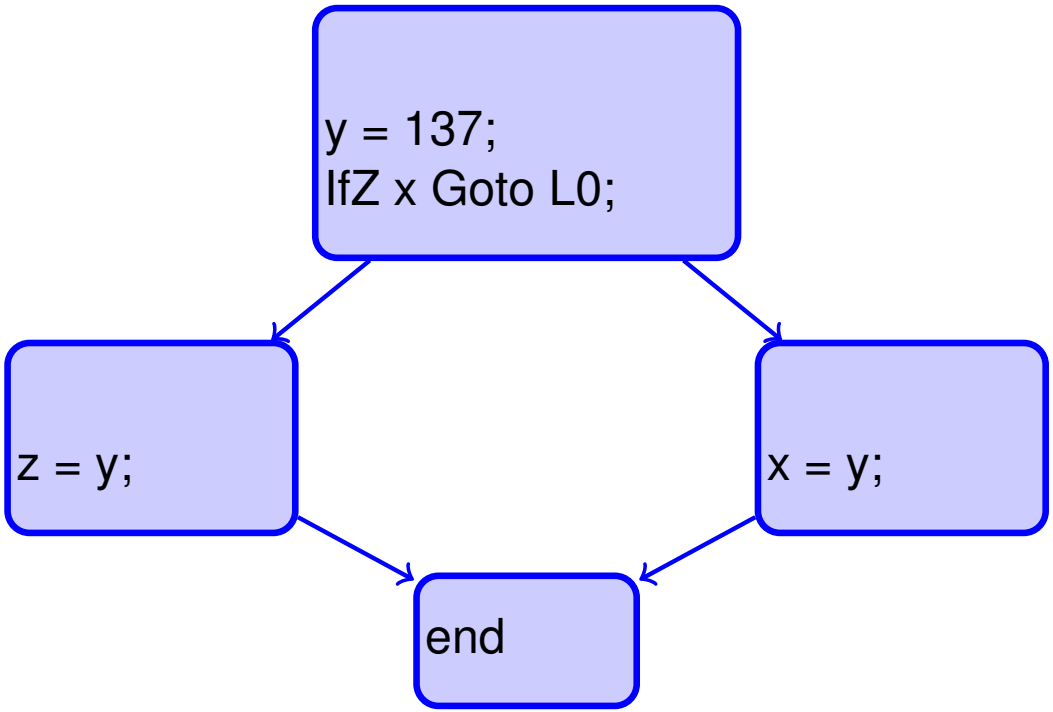
Lokális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



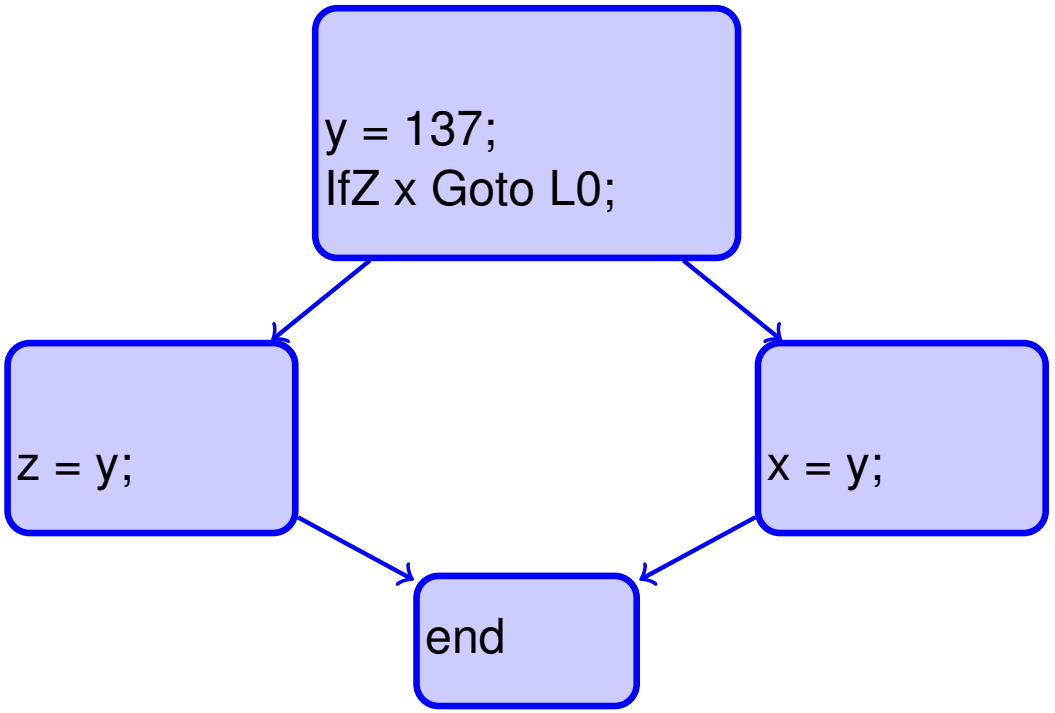
Lokális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



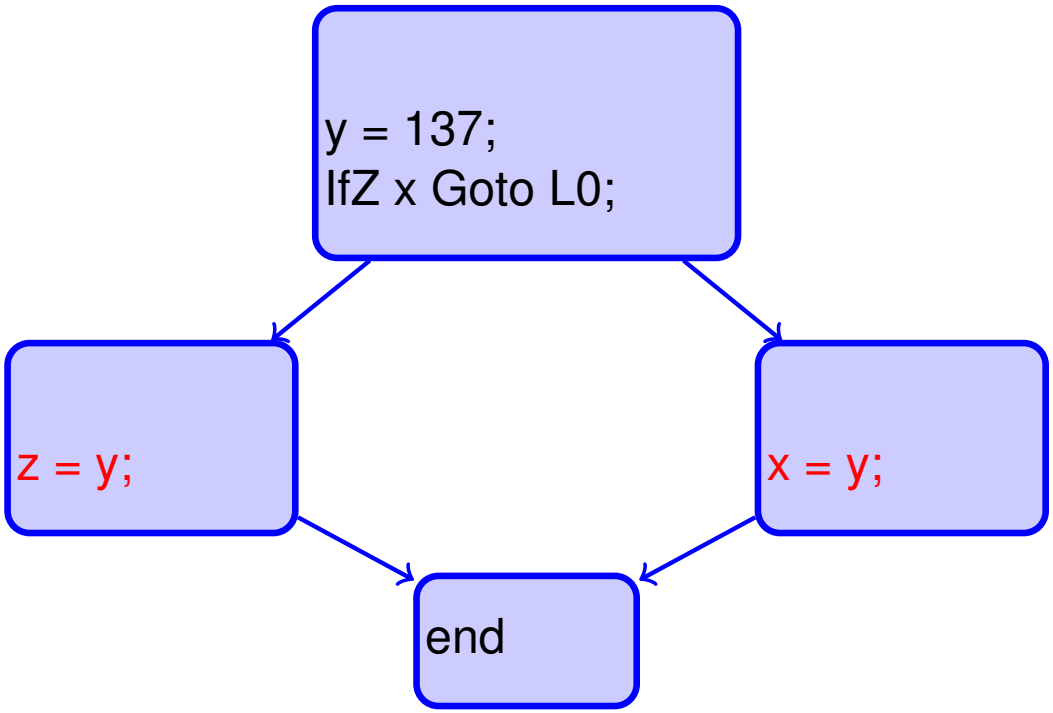
Globális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



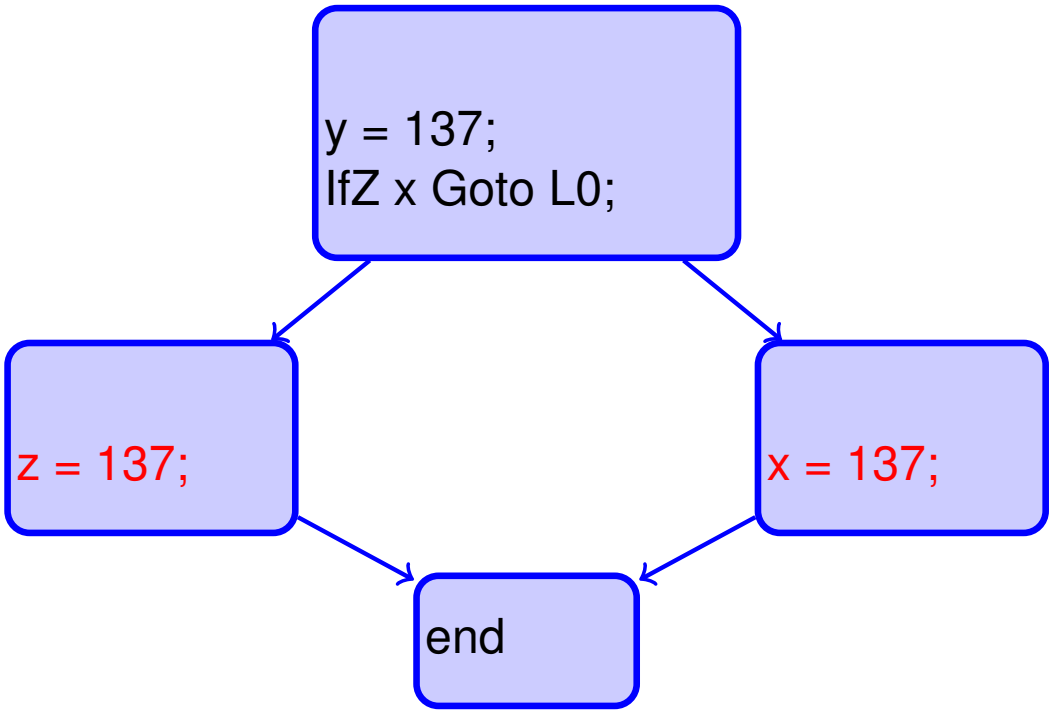
Globális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



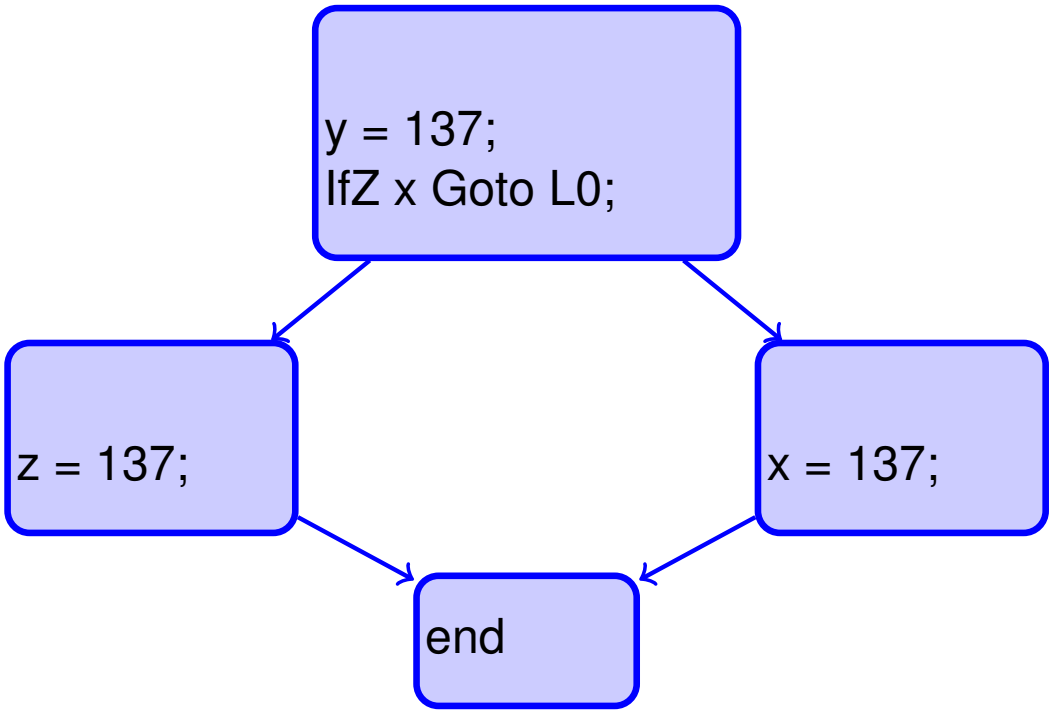
Globális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



Globális optimalizálás

```
int main() {
    int x;
    int y;
    int z;
    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```



Ismétlődő kifejezések kiszűrése

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = Call Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4 ;
a = t3 ;
t4 = a + b ;
c = t4 ;
t5 = a + b ;
PushParam t5 ;
PushParam x ;
Call set ;
PopParams 8 ;
```

Ismétlődő kifejezések kiszűrése

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = Call Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4 ;
a = t3 ;
t4 = a + b ;
c = t4 ;
t5 = a + b ;
PushParam t5 ;
PushParam x ;
Call set ;
PopParams 8 ;
```

Ismétlődő kifejezések kiszűrése

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = Call Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4 ;
a = t3 ;
t4 = a + b ;
c = t4 ;
t5 = t4 ;
PushParam t5 ;
PushParam x ;
Call set ;
PopParams 8 ;
```

Ismétlődő kifejezések kiszűrése

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = Call Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4 ;
a = t3 ;
t4 = a + b ;
c = t4 ;
t5 = t4 ;
PushParam t5 ;
PushParam x ;
Call set ;
PopParams 8 ;
```

Ismétlődő kifejezések kiszűrése

- Ha van két értékadás

$$v1 = a\,op\,b$$
$$\ldots$$
$$v2 = a\,op\,b$$

és az a és b operandusok értéke közben nem változik, akkor a következő transzformációt hajthatjuk végre

$$v1 = a\,op\,b$$
$$\ldots$$
$$v2 = v1$$

- Ezzel elhagyjuk az ismétlődő számításokat.
- lehetőséget teremtünk egy későbbi optimalizáláshoz.

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = t3 ;
t4 = a + b ;
c = t4 ;
t5 = t4;
PushParam t5 ;
PushParam x ;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = t3 ;
t4 = a + b ;
c = t4 ;
t5 = t4;
PushParam t5 ;
PushParam x ;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = t3 ;
t4 = a + b ;
c = t4 ;
t5 = t4;
PushParam t5 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = t3 ;
t4 = a + b ;
c = t4 ;
t5 = t4;
PushParam t5 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = t3 ;
t4 = t3 + b ;
c = t4 ;
t5 = t4;
PushParam t5 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = t3 ;
t4 = t3 + b ;
c = t4 ;
t5 = t4;
PushParam  t5 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = t3 ;
t4 = t3 + b ;
c = t4 ;
t5 = t4;
PushParam  t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = t3 ;
t4 = t3 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = 4 ;
t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = 4 ;
t4 = 4 + b ;
c = t4 ;
t5 = c;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = 4 ;
t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = 4 ;
t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Másolat továbbterjedés

- angol szakirodalomban a Copy Propagation kifejezés él;
- ha lehet, akkor máshol is az eredeti példányra szeretnénk hivatkozni;
- Ha van egy értékadás

$$v1 = v2$$

és a továbbiakban a $v1$ és $v2$ operandusok értéke nem változik, akkor a következő transzformációt hajthatjuk végre

$$a = \cdots v1 \cdots$$

lecseréljük:

$$a = \cdots v2 \cdots$$

- Ez a transzformáció később a segítségünkre lesz.

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = 4 ;
t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;
x = t1 ;
t3 = 4;
a = 4 ;
t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;

t3 = 4;
a = 4 ;
t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;

t3 = 4;
a = 4 ;
t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;

a = 4 ;
t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;

a = 4 ;
t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;

t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;

t4 = 4 + b ;
c = t4 ;
t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;

t4 = 4 + b ;

t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;

t4 = 4 + b ;

t5 = t4;
PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése (Dead Code Elimination)

```
int* x;
int a;
int b;
int c;
x = new int(2);
a = 4;
c = a + b;
set(x,a + b);
```

```
t0 = 4 ;
PushParam t0 ;
t1 = LCall Alloc ;
PopParams 4 ;
t2 = 2 ;
*(t1) = t2 ;

t4 = 4 + b ;

PushParam t4 ;
PushParam t1;
Call set;
PopParams 8 ;
```

Felesleges utasítások törlése

- Értékadás bal oldalán található változó akkor *felesleges*, ha az értékét a későbbiekben nem használjuk fel.
- Ha az értékadás bal oldalán található változó *felesleges*, akkor az egész utasítást el lehet hagyni.