Kodegenerasie

- Verfyn sintaksontleder om kode te genereer.
- Genereer LDW instruksies vir verwysings na 'n veranderlikes.
- Genereer STW instruksies om waardes toe te ken aan veranderlikes.
- Genereer ADD instruksie vir "+", SUB vir "–', ens.
- Hou tred van watter registers in gebruik is.

Toekennings-opdragte

- Laai veranderlike waardes en konstantes in ongebruikte register(s).
- Genereer instruksies wat bewerkings weerspieël.
- Stoor resultaat in veranderlike.

Voorbeeld: `x := x + 1`

```
LDW   1,0,x   R1 := x
ADDI  2,0,1   R2 := 1
ADD   1,1,2   R1 := R1 + R2
STW   1,0,x   x := R1
```


Kode is nie optimaal nie

- Dis onnodig om die konstante "1" in 'n register te laai, want konstantes kan *direk* bygetel word by registers.
- Kode kan verbeter word deur die besluit oor watter kode om te genereer so lank moontlik uit te stel.
- Die volgende kode is dan moontlik:

```
LDW   1,0,x   R1 := x
ADDI  1,0,1   R1 := R1 + 1
STW   1,0,x   x := R1
```

- Elke instruksie wat vermy kan word maak die kode *kleiner* en waarskynlik *vinniger*.

Toekennings-opdragte: beter kode

- Beginsel: koppel ekstra inligting aan elke *operand* om kodegenerasie te ondersteun.
- Operande beïnvloed kode: (1) in geheue tydens looptyd (2) konstante of (3) waarde in register.
- Rekord-struktuur is nuttig om ekstra inligting te beskryf:

```
Item = RECORD
  mode: INTEGER; (* Var, Const, Reg *)
  type: Type; (* tipe van entiteit *)
  a, r: LONGINT (* memory or register *)
END
```

Laai van operande in registers

- Soek veranderlikes op in simbooltabel en ken inligting toe aan rekord van tipe `Item`. Gebruik die inligting om kode te genereer:

```
PROCEDURE Statement;
  VAR ent: Entity; x, y: Item;
BEGIN
IF sym = ident THEN
  find(ent); Scanner.Get(sym);
  x.mode := ent.class;
  x.a := ent.adr; x.r := 0;
  IF sym = becomes THEN
    Scanner.Get(sym); expression(y);
    CodeGen.Store(x, y)
  ELSIF ...
  END
ELSIF ...
END Statement;
```


CodeGen: kodegenerator module

Bevat prosedures wat spesifiek is tot die teikenmasjien: GetReg (get a free register), load (load operand into register), Store (store register in memory), Op1 (generate unary operator), Op2 (generate binary operator), ens.

```
PROCEDURE load(VAR x: Item);
BEGIN
(* assumption: x.mode # Reg *)
  IF x.mode = Var THEN
    GetReg(x.r);
    Put(LDW, x.r, 0, x.a)
  ELSIF x.mode = Const THEN
    IF x.a = 0 THEN x.r := 0
    ELSE
      GetReg(x.r);
      Put(ADDI, x.r, 0, x.a)
      END
    END
  END;
  x.mode := Reg
END load;
```


Kode vir bewerkings

Prosedure `expression` roep prosedure `SimpleExpression`.

```
PROCEDURE SimpleExpression(VAR x: Item);
  VAR y: Item; op: INTEGER;
BEGIN
  IF sym = plus THEN Get(sym); term(x)
  ELSIF sym = minus THEN
    Get(sym); term(x);
    CodeGen.Op1(minus, x)
  ELSE term(x)
  END;
  WHILE (sym = plus) OR (sym = minus) DO
    op := sym; Get(sym); term(y);
    CodeGen.Op2(op, x, y)
  END
END SimpleExpression;
```


Bewerkings met een operand

```
PROCEDURE Op1(op: INTEGER; VAR x: Item);
(* replace "x" by "op x" *)
BEGIN
  IF op = minus THEN
    IF x.mode = Const THEN x.a := -x.a
    ELSE
      IF x.mode = Var THEN load(x) END;
      Put(SUB, x.r, 0, x.r)
    END
  ...
  END
END Op1;
```

- Kies instruksie in oorstemming met operand. (Prosedure `Put` genereer die instruksie.)
- Onderskei tussen konstantes en veranderlikes.

Bewerkings met twee operande

```
PROCEDURE Op2(op: INTEGER; VAR x, y: Item);
(* replace "x" by "x op y" *)
BEGIN
  IF (x.mode = Const) & (y.mode = Const) THEN
    IF op = plus THEN x.a := x.a + y.a
    ELSIF op = minus THEN x.a := x.a - y.a
    ...
  ELSE
    IF op = plus THEN PutOp(ADD, x, y)
    ELSIF op = minus THEN PutOp(SUB, x, y)
    ...
    END
  END
END Op2;
```



```
PROCEDURE PutOp(cd: LONGINT; VAR x, y: Item);
  VAR r: LONGINT;
BEGIN
  IF x.mode # Reg THEN load(x) END;
  IF x.r = 0 THEN
    GetReg(x.r); r := 0
  ELSE
    r := x.r
  END;
  IF y.mode = Const THEN
    (* immediate mode is cd+16 *)
    Put(cd+16, r, x.r, y.a)
  ELSE
    IF y.mode # Reg THEN load(y) END;
    Put(cd, x.r, r, y.r);
    ReleaseRegister(y.r)
  END
END PutOp;
```