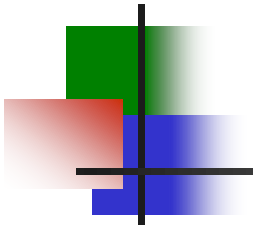
INF5110 – 2014
Noen oppgaver til kap. 8
Utvidet utgave lagt ut 24. april

Gjennomgås 25. april, 2014

Stein Krogdahl



Oppgave 8.1.c (fra boka)

Lag for hånd TA-kode for følgende uttrykk:

a * b + a * b * c

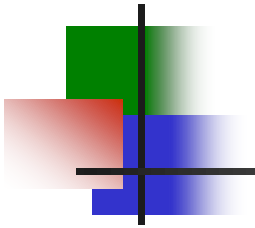
Du skal ikke prøve å optimalisere koden. Velg variabelnavn på temporære som t1, t2, osv.

Oppgave 8.1.c, med optimalisering

Lag igjen for hånd TA-kode for følgende uttrykk:

a * b + a * b * c

Nå skal du optimalisere koden, ved *ikke* å beregne samme subuttrykk om igjen.



Oppgave 8.2.c (fra boka)

Lag for hånd P-kode for følgende uttrykk:

a * b + a * b * c

Du skal her ikke prøve å optimalisere koden.

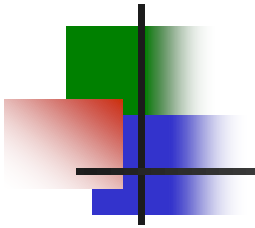
Oppgave 8.2.c, med optimalisering

Lag for hånd P-kode for følgende uttrykk:

a * b + a * b * c

Du skal nå prøve å optimalisere koden, ved ikke å beregne samme uttrykket om igjen.

Foreslå en ekstra P-instruksjon slik at dette går an.



Oppgave 1: If-setninger, P-kode

I den følgende if-setning er b, c og d boolske variable

if (b and c or d) a = x else x = a ;

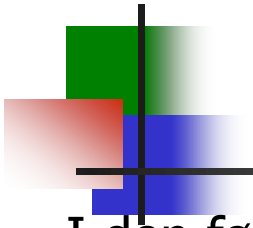
Lag TA-kode for denne der betingelsen ikke er kortsluttet og slik at den blir beregnet til en logisk verdi:

Oppgave 2: If-setninger, med kortslutning

I den følgende if-setning er b, c og d boolske variable

if (b && c || d) a = x else x = a ;

Lag TA-kode for denne der betingelsen blir kortsluttet og slik at alle hopp går så direkte som mulig



Oppgave 3: If-setninger, TA-kode

I den følgende if-setning er b, c og d boolske variable

if (b and c or d) a = x else x = a ;

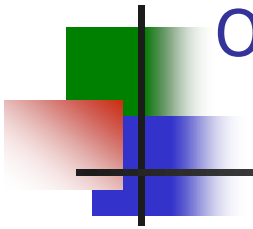
Lag TA-kode for denne der betingelsen ikke er kortsluttet og slik at den blir beregnet til en logisk verdi

Oppgave 4: If-setninger, med kortslutning

I den følgende if-setning er b, c og d boolske variable

if (b && c || d) a = x else x = a ;

Lag TA-kode for denne der betingelsen blir kortsluttet og slik at alle hopp går så direkte som mulig. Kodegenereings-metode diskuteres i pensum-foilene



Oppgave 5: Lag OO-versjon av program som genererer TA-kode med kortslutning for betingelser

Et program fra pensumfoilene som gjør dette er beskrevet på neste foil. Oppgaven er å skrive det samme programmet i en objektorientert stil.

Oppsett for svar ligger på foilen deretter.

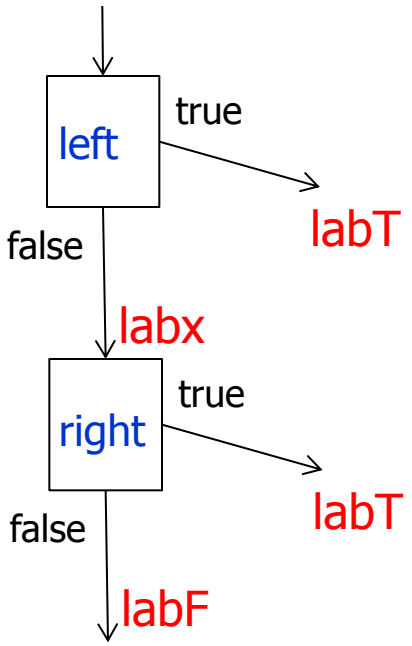
Til oppgave 5:
Genere TA-kode for boolske uttrykk

Men heller ikke *denne* vil lage helt god kode!
Hvorfor?? (se helt nederst)

Vi bryr oss ikke med retur-navnet, siden de alltid vil hoppe ut

```
void genBoolCode(String labT, labF) {
        …
        case "||": {
            String labx = genLabel();
            left.genBoolCode(labT, labx);
            emit2("label", labx);
            right.genBoolCode(labT, labF);
        }
        case "&&": {
            String labx = genLabel();
            left.genBoolCode(labx, labF);  // som over
            emit2("label", labx);
            right.genBoolCode(labT, labF);  // som over
        }
        case "not": { // Har bare "left"-subtre
            left.genBoolCode(labF, labT);  // Ingen kode lages!!!
        }
        case "<": {
            String temp1, temp2, temp3; // temp3 skal holde den boolsk verdi for relasjonen
            temp1 = left.genIntCode(); temp2 = right.genIntCode();      temp3 = genLabel();
            emit4(temp3, temp1, «lt», temp2); // temp3 får (det boolske) svaret på relasjonen
            emit3(«jmp-false», temp3, labF);
            emit2(«ujp», labT);  // Denne er unødvendig dersom det som følger etter er labT
                                 // Dette kan vi oppdage med en ekstra parameter som angir labelen bak
                                 // den konstrusksjonen man kaller kodegenererings-metoden for.
}       }
```

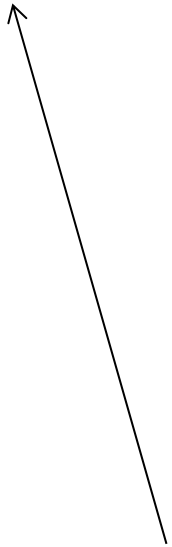
For "||":



Til oppgave 5: Oppsett for kortsluttet kode

Metoden for et slikt uttrykk blir kalt fra while-stm eller if-stm, med labler den vet det skal hoppes til ved true eller false betingelse

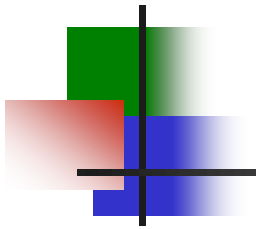
```
abstract  class Expression {
   // Hvordan skal «String codeGen(…)» se ut her?
}

class OrOp extends Expression {
   String codeGen (String LabT, LabF){ ... }
}

class AndOp extends Expression {
   String codeGen (…){ … }
}

class NotOp extends Expression {
   String codeGen (…){ … }
}

class LessThanOp extends expression {
   String codeGen (…){ … }
}
```



Merk at disse metodene kalles «langs» pekere typet med Expression.

Spørsmål:

Hva må da stå i Expression-klassen?

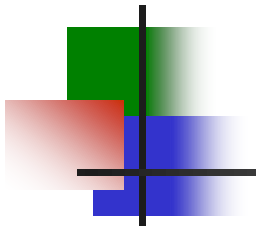


Oppgave 6

Se på kommentaren nederst i programmet angitt i oppgave 5, nemlig:

```
...
emit2(«ujp», labT);  // Denne er unødvendig dersom det
som følger etter er labT. Dette kan vi oppdage med en
ekstra parameter som angir labelen bak den
konstrusksjonen man kaller kodegenererings-metoden for.
```

Gjør om svaret på oppgave 5, slik at man oppnår denne effekten. Da må vi altså ha nok en label-parameter



Oppgave 7

Skriv ferdig delene for WhileKind og BreakKind som ikke er skrevet ut på foil 43 (gjengitt som neste foil her)

For hva som skal gjøres, se foil 42.

Rekursiv prosedyre for P-kodegenerering for setninger. Foil 43 fra forelesninngen

Merk: Stakken antas å være tom før og etter kodegen. for setning, men at stakken øker med én i løpet av kodegen. for uttrykk.

En "break" i kildeprogr. skal bli et hopp til denne labelen. Den vi angi første instruksjon etter nærmest omsluttende while-setning.

```
void genCode(TreeNode t, String label){
  String lab1, lab2;
  if t != null{  // For et tomt tre, ikke gjør noe
    switch t.kind {
      case ExprKind {  // I boka (forrige foil) er det veldig forenklet.
                       // Kan behandles slik uttrykk er behandlet tidligere
      }
      case IfKind {      // If-setning
        genCode(t.child[0], label); // Lag kode for det boolske uttrykket.  Brukers egentlig label her?
        lab1= genLabel();
        emit2("fjp", lab1);  // Hopp til mulig else-gren (eller til slutten om ikke else-gren)
        genCode(t.child[1], label); // kode for then-del, gå helt ut om break opptrer
        if t.child[2] != null {     // Test på om det er else-gren?
          lab2 = genLabel();
          emit2("ujp", lab2);   //  Hopp over else-grenen
        }
        emit2("label", lab1);     // Start på else-grenen, eller slutt på if- setningen
        if t.child[2] != null {     // En gang til: test om det er else-gren? (litt plundrete programmering)
          genCode(t.child[2], label); // Kode for else-gren, gå helt ut om break opptrer
          emit2("lab", lab2);   // Hopp over else-gren går hit
      } }
      case WhileKind {  /* mye som over, men OBS ved indre "break". Se boka og forrige foil */ }
      case BreakKind { emit2("ujp", label); }  // Hopp helt ut av koden som dette genCode-kallet lager
      ...                                      //  (og helt ut av nærmest omsluttende while-setning)
} } }
```