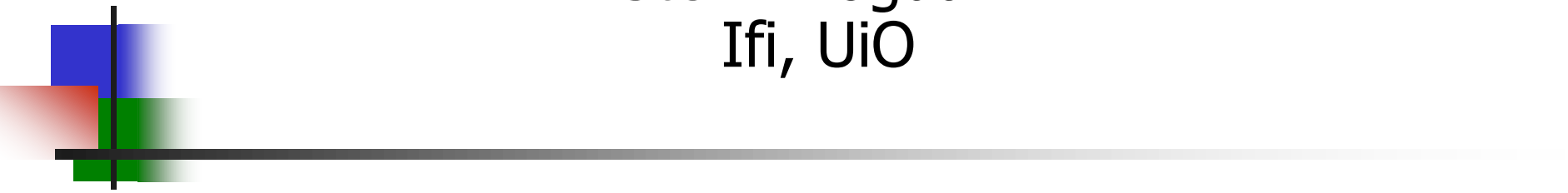
INF5110 – 25. april, 2014

Stein Krogdahl
Ifi, UiO



Svar på oppgaver til kap. 8

som ble lagt ut 24. april

Feil bes rapportert til:

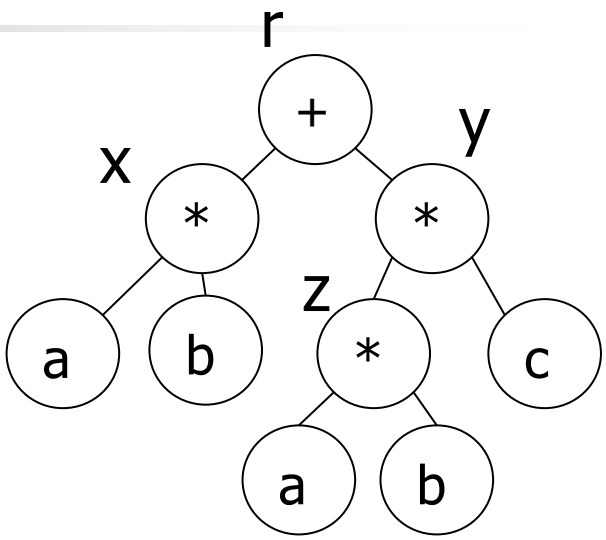
«steinkr@ifi.uio.no»

SVAR: 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. :



t1 = a * b Lages av node x
t2 = a * b Lages av node z
t3 = t2 * c Lages av node y
t4 = t1 + t3 Lages av node r

Det lages altså ikke kode i blad-nodene!

Og man skal melde «opp» at svaret ligger i t4

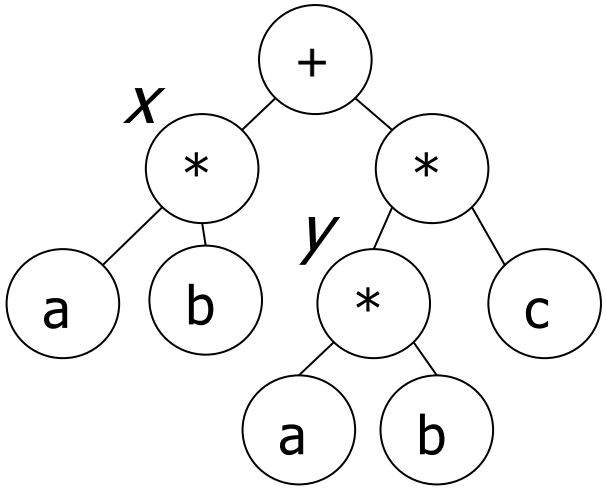
Her er et ADT for uttrykket, og det er en metode i hver node. Til venstre er også angitt i hvilken node de forskjellige deler av TA-koden blir produsert.

SVAR: Oppgave 8.1.c, med optimalisering

Lag for hånd TA-kode for følgende uttrykk:

a * b + a * b * c

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




```
t1 = a * b
t2 = t1 * c
t3 = t1 + t3
```

For å kunne lage slik kode må kompilatoren:

1. Oppdage når det *er* subtrær som er like. Dette må gjøres med en spesiell type søk, som vi ikke går inn på her.
2. At trær er like kan her markeres med en peker fra node y til node x i treet over (og å fjerne bladnodene under y ?).
3. Så må kodegeneratoren ta hensyn til denne nye pekeren

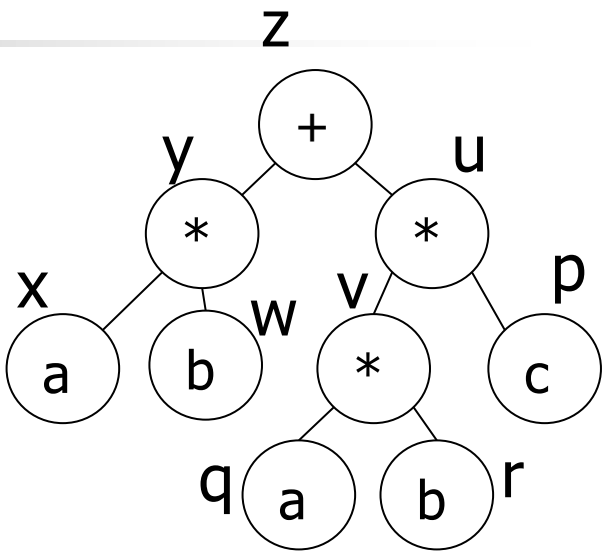
Her er et ADT for uttrykket, og det er én kodegen.-metode i hver node. Tenk på hvordan en kompilator skulle kunne gjøre slike optimaliseringer.

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.



lod a	lages av x
lod b	lages av w
muli	lages av y
lod a	lages av q
lod b	lages av r
muli	lages av u
addi	lages av z

Her er et ADT for uttrykket, og det er én metode i hver node. Angi i hvilken node de forskjellige deler av P-koden blir produsert.

SVAR: 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.

Vi foreslår en «dup»-instruksjon som dupliserer toppen av stakken.
Den gamle koden til venstre kan da skrives som til høyre:

```
lod a
lod b
muli
lod a
lod b
muli
lod c
muli
addi
```

Erstattes av *dup*

```
lod a
lod b
muli
dup
lod c
muli
addi
```

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 skal bli kortsluttet og slik at den blir beregnet til en logisk verdi (true eller false):

```
lod a
lod b
and
lod d
or
jfalse L1
lda a          Load adresse!
lod x
sto            Begge forsvinner
jmp L2
label L1
lda x          Load adresse!
lod a
sto
label L2
```

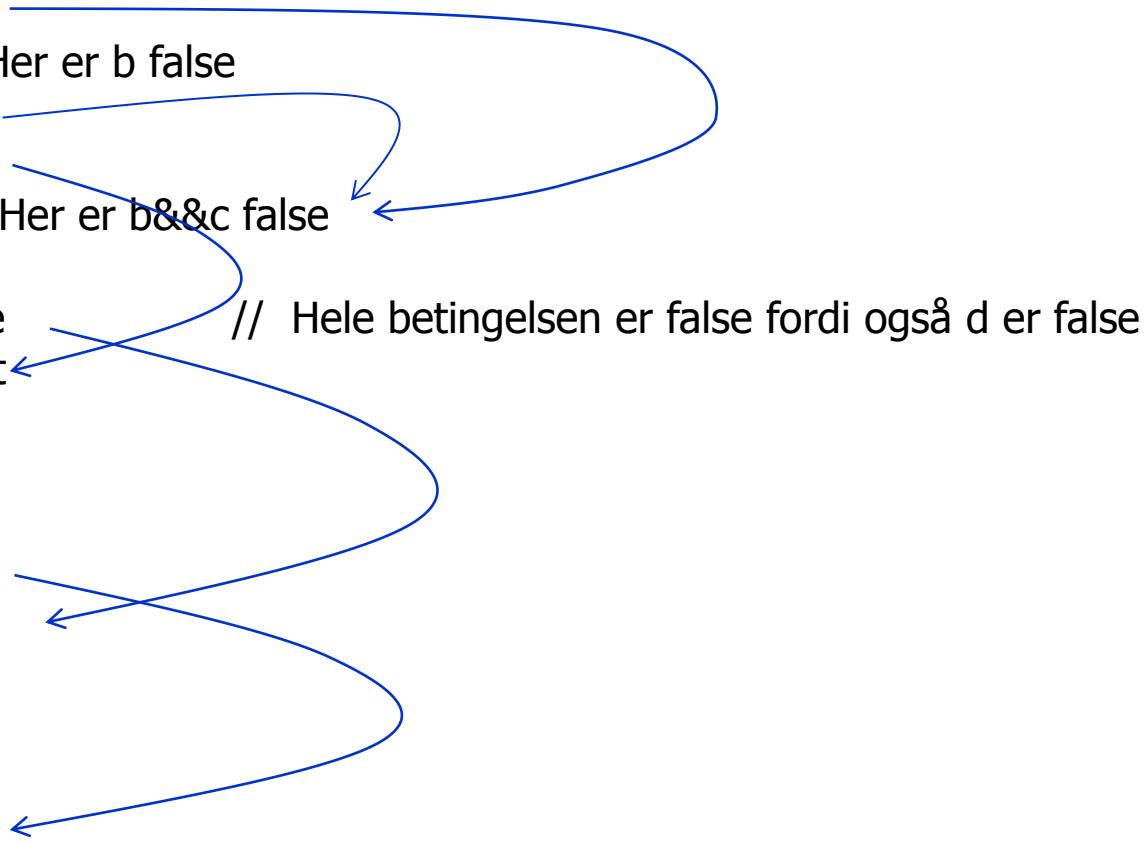
Oppgave: If-setninger oppg. 2. Med kortslutning

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

```
lod b
jfalse Ld
lod c       Her er b false
jfalse Ld
jmp Lstart
label Ld    Her er b&&c false
lod d
jfalse Lelse            //  Hele betingelsen er false fordi også d er false
label Lstart
lda a
lod x
sto
jmp Lslutt
label Lelse
lda x
lod a
sto
label slutt
```



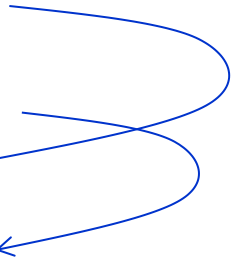
Oppgave 3: If-setninger, TA-kode

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

```
t1 = b and c
t2 = t1 or d
jfalse Lelse
a = x
jmp Lslutt
lab Lelse
x = a
label  Lslutt
```

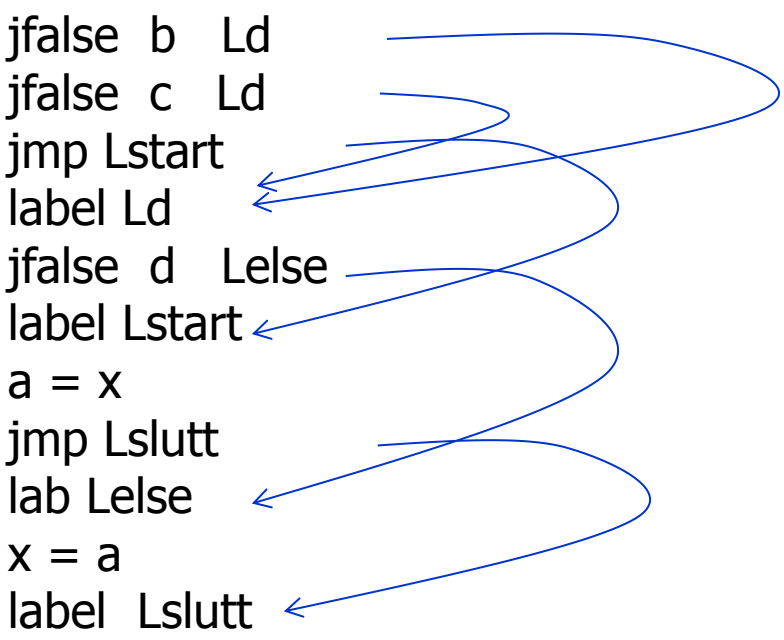


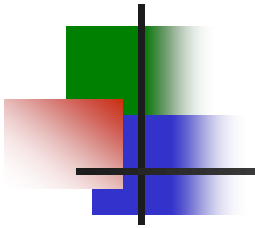
Oppgave 4: If-setninger med kortslutning

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: Generere kode med kortslutning

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

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

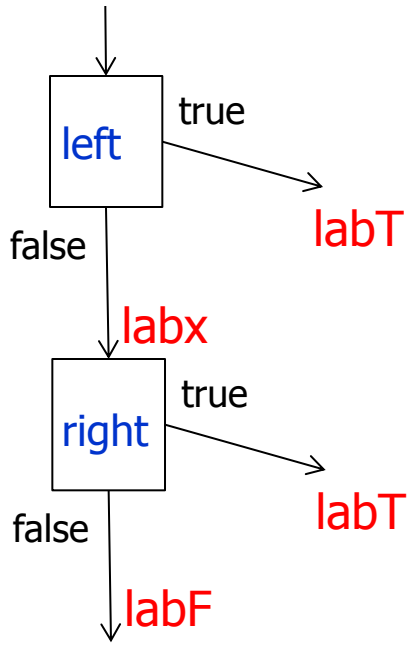
Oppsett for svar ligger på foilen deretter.

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

For "||":




```
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.
```

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

```
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 (…){ … }
}
```

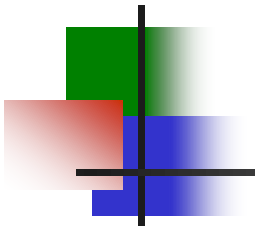
Merk at disse metodene kalles «langs» pekere typet med Expression.

Vi må derfor ha en versjon av metoden også i Expression. Angi hvordan denne ser ut.

```
class OrOp extends Expression {
   String codeGen (String LabT, LabF){
        String Labx = genLabel();
        left.genBoolCode(labT, labx);
        emit2("label", labx);
        right.genBoolCode(labT, labF);
} }
class AndOp extends Expression {
   String codeGen (String LabT, LabF){
        String Labx = genLabel();
        left.genBoolCode(Labx, LabF);
        emit2("label", labx);
        right.genBoolCode(LabT, LabF);
}  }
class NotOp extends Expression {
   String codeGen (String LabT, LabF){
   left.genBoolCode(LabF, LabT);
} }
class LessThanOp extends expression {
    String codeGen (String LabT, LabF){
        String temp1, temp2, temp3;
        temp1 = left.genIntCode(); temp2 = right.genIntCode();  temp3 = genLabel();
        emit4(temp3, temp1, «lt», temp2);
        emit3(«jmp-false», temp3, LabF);
        emit2(«ujp», LabT);
} }
```

Svar på oppgave 5

Se også neste foil



Avslutnings-spørsmål til oppgave 5:

a. Hva må da stå i Expression-klassen?

```
abstract class Expression {
    abstract String codeGen (String LabT, LabF);
    …
}
```

b. Hva skal startkallet være?

Startkallet gjøres typisk for betingelsen i en IF- eller WHILE-setning, og vil typisk se slik ut:

```
betingelse.codeGen(LabX, LabY);
```

Om det er en IF-setning, så må da «Label LabX» stå ved starten av then-grenen, og «label LabY» stå ved starten av else-grenen.

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 «label labT». Dette kan vi oppdage
med en ekstra parameter som angir labelen bak den
konstrusksjonen som man nå kaller kodegenererings-
metoden for.
```

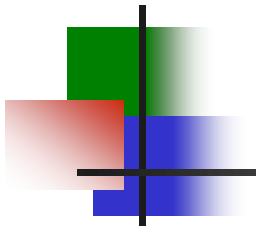
Gjør om svaret på oppgave 5, slik at man oppnår denne effekten. Da må vi altså ha nok en label-parameter.

Svar på oppgave 6

```
class OrOp extends Expression {
   String codeGen (String LabT, LabF, LabA){
        String labx = genLabel();
        left.genBoolCode(LabT, Labx, Labx);
        emit2("label", labx);
        right.genBoolCode(LabT, LabF, LabA);
} }
class AndOp extends Expression {
   String codeGen (String LabT, LabF, LabA){
        String labx = genLabel();
        left.genBoolCode(Labx, LabF, Labx);
        emit2("label", Labx);
        right.genBoolCode(LabT, LabF, LabA);
} }
class NotOp extends Expression {
   String codeGen (String LabT, LabF, LabA){
   left.genBoolCode(LabF, LabT, LabA);
} }
class LessThanOp extends expression {
    String codeGen (String LabT, LabF, LabA){
        String temp1, temp2, temp3;
        temp1 = left.genIntCode(); temp2 = right.genIntCode();  temp3 = genLabel();
        emit4(temp3, temp1, «lt», temp2);
        emit3(«jmp-false», temp3, LabF);
        if (LabT != LabA)  { emit2(«ujp», LabT); }  // Vi genererer ikke «tulle-hopp»
} }
```

Her er altså LabA («Label After») den label som blir stående rett etter det uttrykket vi nå kaller «genCode» for.

Dette vil kalleren av metoden hele tiden ha oversikt over



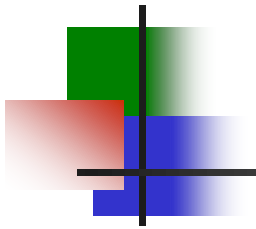
Avslutnings-spørsmål til oppgave 6:

Hva skal startkallet være?

Startkallet gjøres typisk for betingelsen i en IF- eller WHILE-setning:

```
betingelse.codeGen(LabX, LabY, LabX);
```

Om det er en IF-setning, så må da «label LabX» stå ved starten av then-grenen, og «Label LabY» stå ved starten av else-grenen. Den siste parameteren er labelen på det som vil komme umiddelbart etter betingelsen, altså LabX.



Oppgave 7

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

Angående 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 vil 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)
} } }
```

Rekursiv prosedyre for P-kodegenerering for setninger (Foreles. foil 42)

En "break" i kildeprogr. skal bli et hopp til denne labelen

```
void genCode( SyntaxTree t, char * label)
{ char codestr[CODESIZE];
  char * lab1, * lab2;
  if (t != NULL) switch (t->kind)
  { case ExpKind:
      if (t->val==0) emitCode("ldc false")
      else emitCode("ldc true");
      break;
    case IfKind:
      genCode(t->child[0],label);  Rek. kall
      lab1 = genLabel();
      sprintf(codestr,"%s %s","fjp",lab1);
      emitCode(codestr);
      genCode(t->child[1],label);  Rek. kall
      if (t->child[2] != NULL)
      { lab2 = genLabel();
        sprintf(codestr,"%s %s","ujp",lab2);
        emitCode(codestr);}
      sprintf(codestr,"%s %s","lab",lab1);
      emitCode(codestr);
      if (t->child[2] != NULL)
      { genCode(t->child[2],label);  Rek. kall
        sprintf(codestr,"%s %s","lab",lab2);
        emitCode(codestr);}
      break;
    case WhileKind:
      lab1 = genLabel();
      sprintf(codestr,"%s %s","lab",lab1);
      emitCode(codestr);
      genCode(t->child[0],label);  Rek. kall
      lab2 = genLabel();
      sprintf(codestr,"%s %s","fjp",lab2);
      emitCode(codestr);
      genCode(t->child[1],lab2);  Kode for S  Rek. kall
      sprintf(codestr,"%s %s","ujp",lab1);
      emitCode(codestr);
      sprintf(codestr,"%s %s","lab",lab2);
      emitCode(codestr);
      break;
    case BreakKind:
      sprintf(codestr,"%s %s","ujp",label);
      emitCode(codestr);
      break;
    case OtherKind:
      emitCode("Other");
      break;
    default:
      emitCode("Error");
      break;
  }
}
```

Svar, oppgave 7

```
void genCode(TreeNode t, String label){
   String lab1, lab2;
   if (t != null) {  // For et tomt tre, ikke gjør noe
      switch t.kind {
         case ExprKind  { … }
         case IfKind    { … }

         case WhileKind {  // mye som for «if», men spes.beh. av ”break”. Parameter «label» brukes ikke
              String lab1= genLabel(); // Vil angi starten av betingelsen
              String lab2= genLabel(); // Label på setningene etter while-setningene
              emit2(”lab”, lab);   // Hopp hit for neste runde i while-setningen
              genCode(t.child[0], label); // Lag kode for det boolske uttrykket.  (Brukers egentlig label her?)
              emit2(”fjp”, lab2);  // Hopp til setningene etter denne while-setningen
              genCode(t.child[1], lab2); // kode for setningene inne i while-setn. Gå helt ut om break opptrer
              emit2(”ujp”, lab1);   // Hopp tilbake og gjør ny while-test
              emit2(”lab”, lab2);   // Hopp hit for å gå ut av while-setningen
         }

         case BreakKind {
              emit2(”ujp”, label);   // Hopp helt ut av nærmest omsluttende while-setning
         }
      }
   }
}
```

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