COP-3402 Systems Software
10.29 Tue

Grammar for Project#3

```
program
  = declaration* statement*

declaration
  = INT IDENTIFIER SEMI

statement
  = PRINT expression SEMI
  | READ IDENTIFIER SEMI
  | IDENTIFIER ASSIGN expression SEMI
  | IF LPAREN expression RPAREN statement
  | IF LPAREN expression RPAREN statement ELSE statement
  | WHILE LPAREN expression RPAREN statement
  | LCURLY statement* RCURLY

expression
  = expression PLUS expression
  | expression MINUS expression
  | expression TIMES expression
  | expression DIVIDE expression
  | expression MOD expression
  | expression EQUALS expression
  | expression NEQUALS expression
  | expression LT expression
  | expression GT expression
  | expression AND expression
  | expression OR expression
  | NOT expression
  | LPAREN expression RPAREN
  | INTEGER
  | IDENTIFIER
```

Operators have precedence, highest to lowest. Same line is equal precedence

```
 !
  * / %
  + -
  > <
  == !=
  &&
  ||
```

Parenthesized expressions have the highest precedence.

Please refer to the grammar for project#3:
https://github.com/cop3402fall19/syllabus/blob/master/projects/project3.md

Ex:

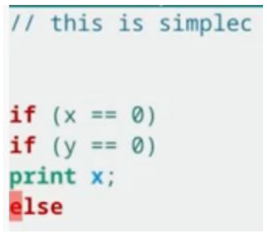
```
int main(int argc) {
        if (argc > 2) {
                return 1;
}
return 0;
}
```

OR

```
int main(int argc) {
        if (argc > 2)  return 1;
return 0;
}
```

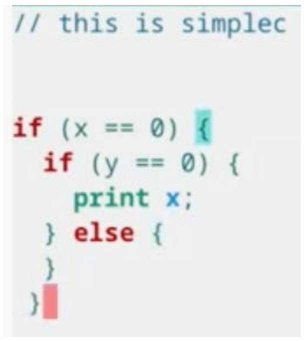
// both valid in our language.

To resolve the dangling else ambiguity, the ‘else’ belongs to the inner if.

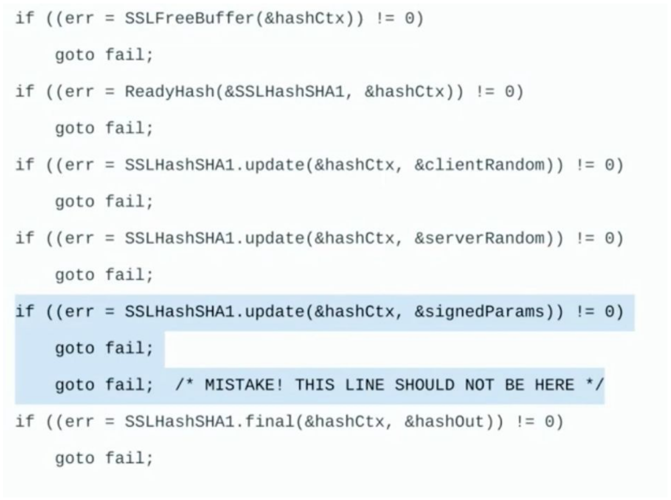


```
// this is simplec


if (x == 0)
if (y == 0)
print x;
else
```

Here, the second if-statement must be a nested if-statement of some kind.



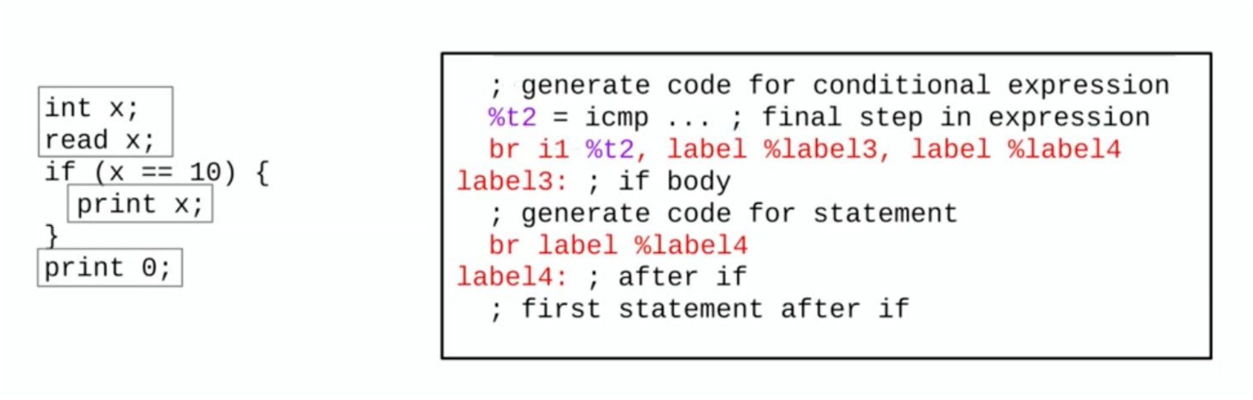
```
// this is simplec


if (x == 0) {
  if (y == 0) {
    print x;
  } else {
  }
 }
```



```
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

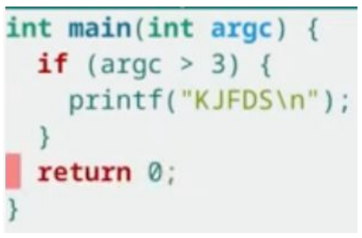
In the highlighted, second goto fail will always run. It does not belong to the if-statement. Meaning the first goto fail is the statement of the if-production. The second goto fail is a sibling of the if-statement.

Template for if-statements:

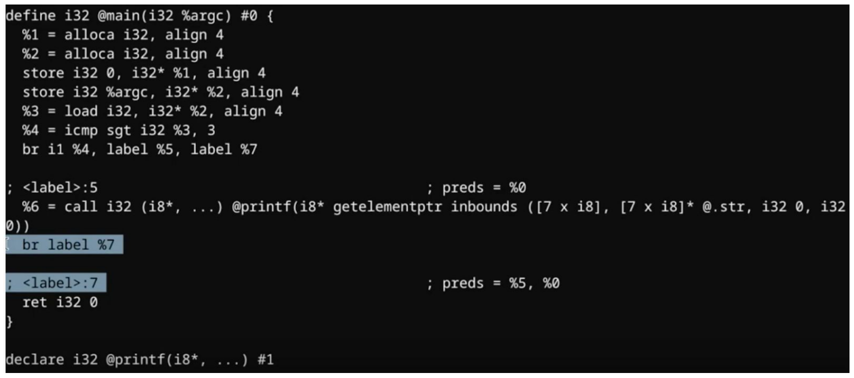


```
int x;
read x;
if (x == 10) {
  print x;
}
print 0;
```

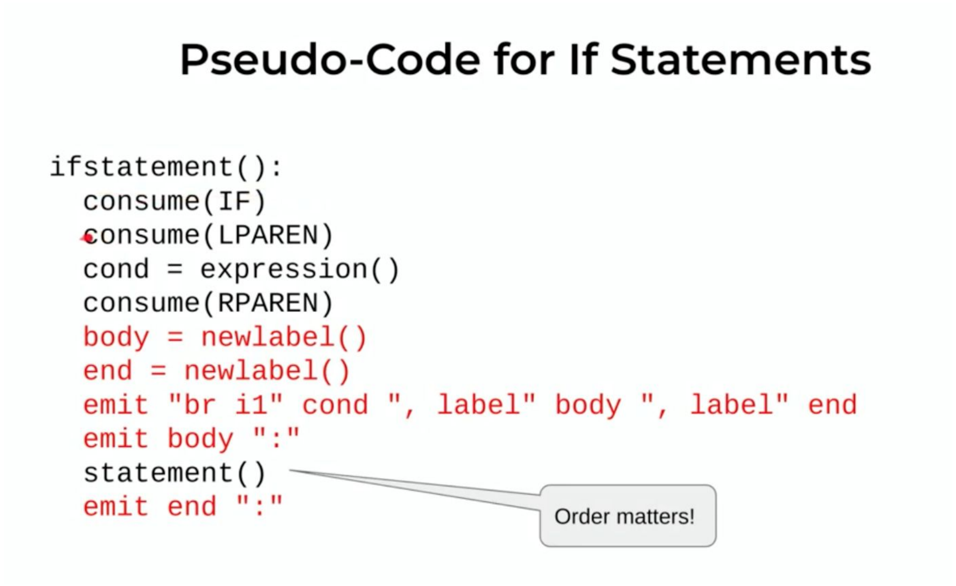
```
  ; generate code for conditional expression
  %t2 = icmp ... ; final step in expression
  br i1 %t2, label %label3, label %label4
label3: ; if body
  ; generate code for statement
  br label %label4
label4: ; after if
  ; first statement after if
```



```
int main(int argc) {
  if (argc > 3) {
    printf("KJFDS\n");
  }
  return 0;
}
```



```
define i32 @main(i32 %argc) #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 %argc, i32* %2, align 4
  %3 = load i32, i32* %2, align 4
  %4 = icmp sgt i32 %3, 3
  br i1 %4, label %5, label %7

; <label>:5                                       ; preds = %0
  %6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([7 x i8], [7 x i8]* @.str, i32 0, i32
0))
  br label %7

; <label>:7                                       ; preds = %5, %0
  ret i32 0
}

declare i32 @printf(i8*, ...) #1
```

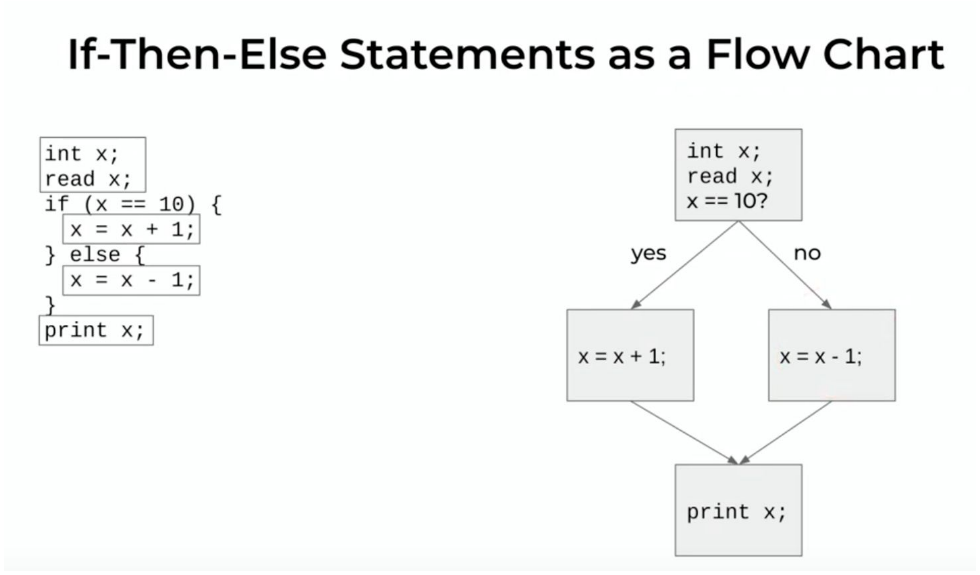


Pseudo-Code for If Statements

```
ifstatement():
  consume(IF)
  consume(LPAREN)
  cond = expression()
  consume(RPAREN)
  body = newlabel()
  end = newlabel()
  emit "br i1" cond ", label" body ", label" end
  emit body ":"
  statement()
  emit end ":"
```

Order matters!

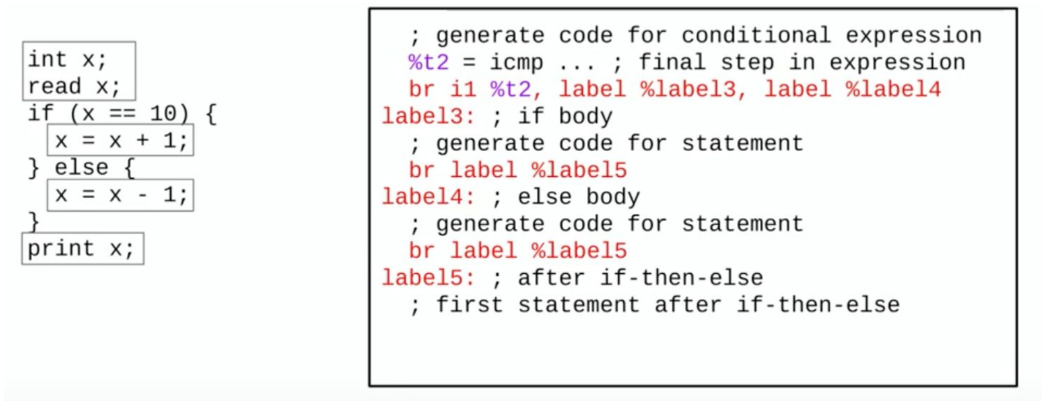
Generating Labels:
Labels can only be defined once, as in variables.
We generate them like temporary registers, such as label1, label2, ifbranch1, elsebranch1, etc.



If-Then-Else Statements as a Flow Chart

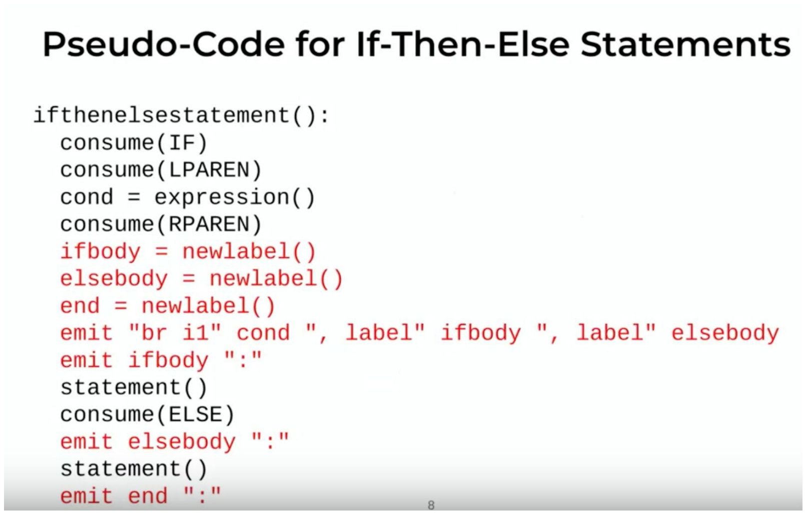
```
int x;
read x;
if (x == 10) {
  x = x + 1;
} else {
  x = x - 1;
}
print x;
```

Template for if-then-else statements:
It works like if-statement; however, instead of jumping immediately to the end-block, we jump to the else-block.



```
int x;
read x;
if (x == 10) {
  x = x + 1;
} else {
  x = x - 1;
}
print x;
```

```
  ; generate code for conditional expression
  %t2 = icmp ... ; final step in expression
  br i1 %t2, label %label3, label %label4
label3: ; if body
  ; generate code for statement
  br label %label5
label4: ; else body
  ; generate code for statement
  br label %label5
label5: ; after if-then-else
  ; first statement after if-then-else
```

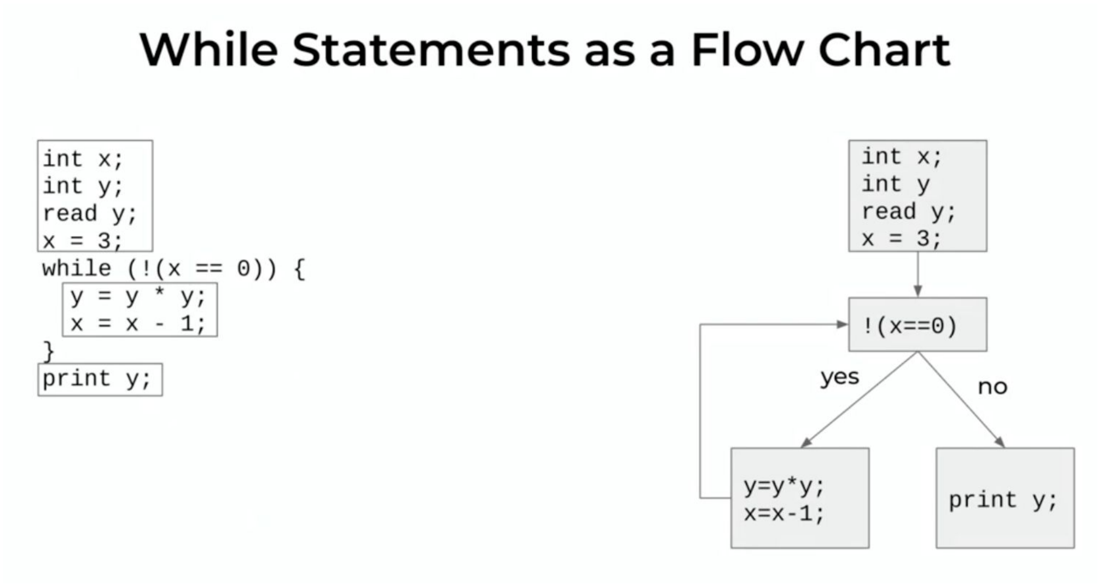


Pseudo-Code for If-Then-Else Statements

```
ifthenelsestatement():
  consume(IF)
  consume(LPAREN)
  cond = expression()
  consume(RPAREN)
  ifbody = newlabel()
  elsebody = newlabel()
  end = newlabel()
  emit "br i1" cond ", label" ifbody ", label" elsebody
  emit ifbody ":"
  statement()
  consume(ELSE)
  emit elsebody ":"
  statement()
  emit end ":"
```

8

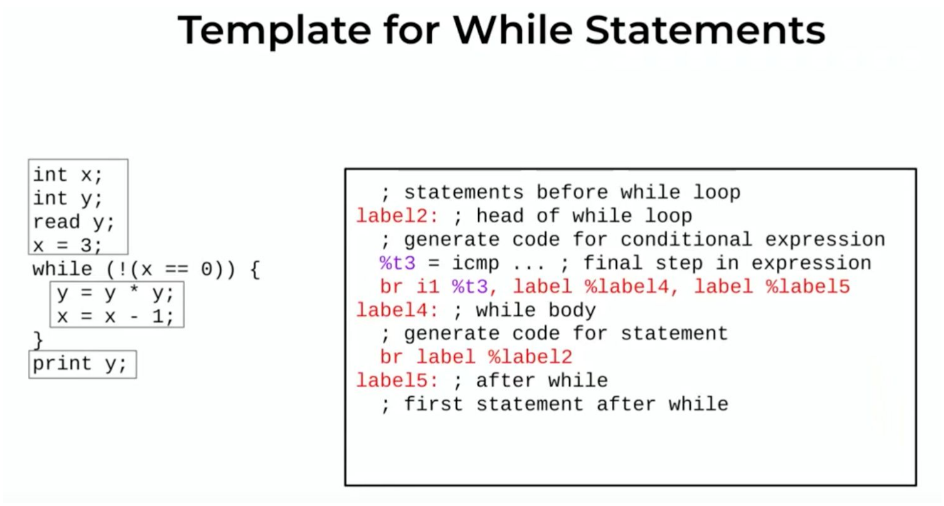
If you always emit a label after an ‘if’ before ‘else’, you can wait until you see an ‘else’ and make a decision about what that label is going to be.



While Statements as a Flow Chart

```
int x;
int y;
read y;
x = 3;
while (!(x == 0)) {
  y = y * y;
  x = x - 1;
}
print y;
```

A while-statement is an if-statement with unconditional branch. The difference is the body has an unconditional branch back to the conditional expression.

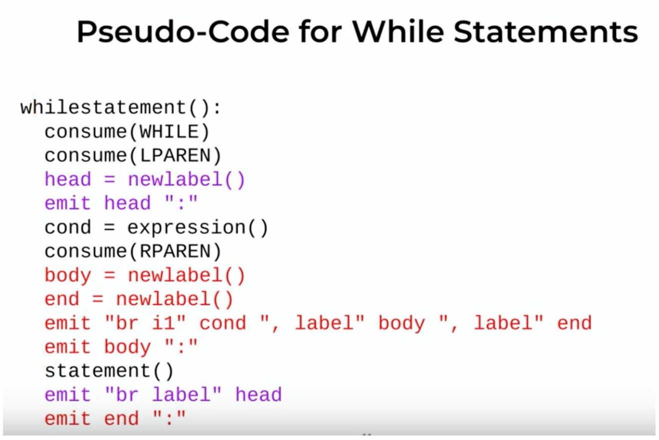


Template for While Statements

```
int x;
int y;
read y;
x = 3;
while (!(x == 0)) {
  y = y * y;
  x = x - 1;
}
print y;
```

```
  ; statements before while loop
label2: ; head of while loop
  ; generate code for conditional expression
  %t3 = icmp ... ; final step in expression
  br i1 %t3, label %label4, label %label5
label4: ; while body
  ; generate code for statement
  br label %label2
label5: ; after while
  ; first statement after while
```

The body always jumps to the head of the while loop(the parentheses of while).



Pseudo-Code for While Statements

```
whilestatement():
  consume(WHILE)
  consume(LPAREN)
  head = newlabel()
  emit head ":"
  cond = expression()
  consume(RPAREN)
  body = newlabel()
  end = newlabel()
  emit "br i1" cond ", label" body ", label" end
  emit body ":"
  statement()
  emit "br label" head
  emit end ":"
```