Imperative Lang. features

Sebesta (11) Ed

- Ch 7. Expressions and Assignment Statement
- Ch 8. Statement-level Control Structures. — (Structured prog.)

---

## Ch 7.

- Arith. expressions
  - precedence
  - Associativity

  — ✓ already covered.

§7.3 — overloaded Operators

ex) $\boxed{+}$ — integer add.
- ft add
- string catenation in Java.

ex) C++

$\boxed{\&}$ — bitwise AND
- address (reference op)

— low readability, error prone in coding

- User-defined overloaded operator.
  - C++, C#, F#. (Java-no).
  - Compiler determines the operation based on operand types

ex) Matrix_Add(Matrix_Mult(A,B), Matrix_Mult(C,D));

$\Rightarrow A \circledast B \oplus C \circledast D$ ;

overloaded matrix mult/add.

§7.4 — Type Conversions ✓ already covered

- Coercion in expression
  - auto type conversion between types

saves programmer's time

ex) Java

```
int a;
float b,c,d;
  :
d = b*a;
```

no error detected.

a: int → float

vs. — explicit type conversion
- (Cast  type casting)
- in C-based languages

ex) C: (int) float_var

ex) F#: float (sum)

func. call.

## §7.6 Short-circuit Evaluation

C/C++/Java

$\underbrace{||}_{OR}$, $\underbrace{\&\&}_{AND}$ — ( short-circuit evaluation of Boolean type
⇒ 2nd operand is evaluated only if necessary.

— all of the logical operators of Ruby, perl, ML, F#, python are short-circuit evaluated.

— Multiple Assignment — multiple_target, multiple_source

⇒ Perl, Ruby, Lua — ( some recent lang's support

Perl

```
($first, $second, $third) = (20, 40, 60);

($first, $second) = ($second, $first);
```

— exchange values.

( without using temp var

$$\begin{tikzcd} & \boxed{temp} \arrow[dr, "3"] & \\ val1 \arrow[ur, "1"] & & val2 \arrow[ll, "2"] \end{tikzcd}$$

Ch8. — Syntax-directed control structures

∄ goto → structured programming
(syntax-directed control structures)

## while/for loops

ex)

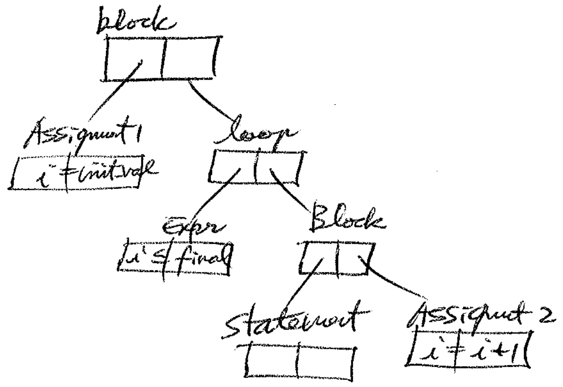
```
        i = init_val ;
block   while (i ≤ final)
        { statement ;
  block   i = i+1 ;
        }
```

= for(i=init_val ; i≤final ; i=i+1)
    statement ;

concrete syntax

↓

Abstract syntax
(by parser)



C/C++/Java syntax

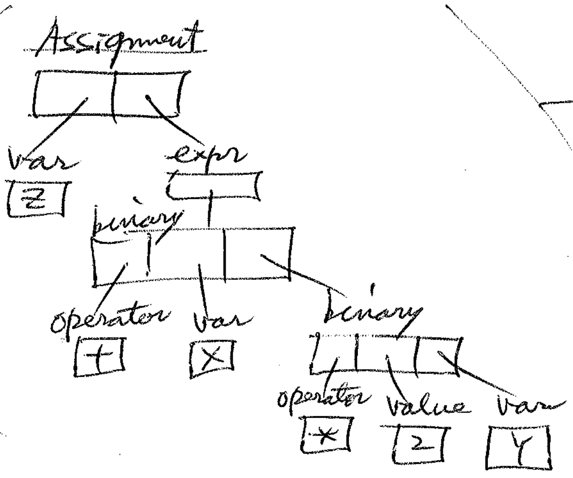
ForStatement → for (Assignmt1 ; expr ; Assignmt2)
    Statement

optional

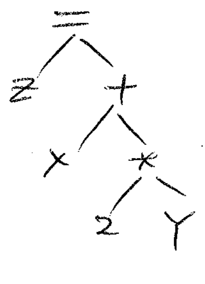
[ without ~~those~~ Assignmt 1, 2
⇒ while statement

ref AST syntax notation

ex) Z = X + 2 * Y ; =

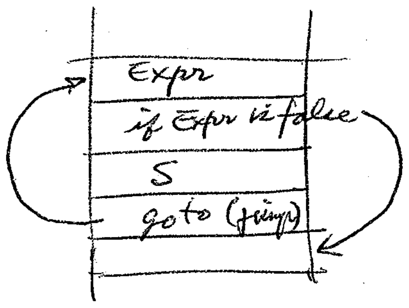


→ Simplified



— implementation of loop construct (in Compiler)

— ex while Expr do S



— ex while (A≥B) do S

⇒ Label1: BLT A,B, Label2 ;
S
JMP Label1 :
Label2: Nop

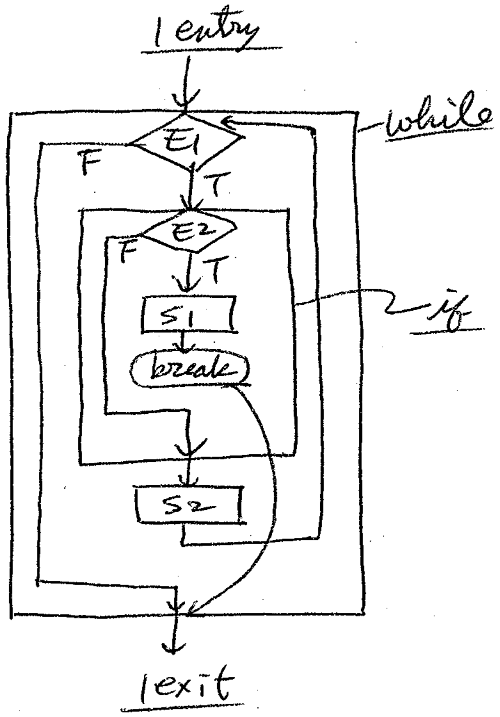
Assemble level

— case statement
switch
→ use jump table — stores jump addresses.

— Handling special cases in the loop

Break — exit from loop

```
ex while (E1)
   { if (E2)
       { S1; — special case
         break; — to out of loop
       }
     S2 ; — normal case
   }
```



— each construct 1 entry/1 exit