

Project 3:
 loops — while, for, repeat
 case statements

while loop:
 while $\langle \text{bool exp} \rangle$ do
 $i := i - 1$ body
 end while

repeat loop:
 repeat
 [statement]
 list
 until $\langle \text{bool exp} \rangle$

For-loops in Pascal
 for $i := 3$ to 8 do
 [statement]
 end for

for $\langle \text{exp} \rangle$ to $\langle \text{exp} \rangle$ do
 [statement]
 end for

$\langle \text{exp} \rangle$ is a variable of some ordinal type
 and $\langle \text{exp} \rangle$ is of the same type

Apr 11-11:04 AM

Now what!
 initial exp & final exp
 evaluated once at beginning
 (or) initialization (assignment)
 to initial exp

but we need final exp
 if $\langle \text{exp} \rangle$, then post
 else, full enough to body

encode body
 increment var $\{ \text{var} := \text{var} + 1 \}$
 (line $\langle \text{exp} \rangle$) (loop)
 original to do $\langle \text{exp} \rangle$ inc
 jump to beginning; after
 initialization and before test

for $\langle \text{exp} \rangle$ to $\langle \text{exp} \rangle$ do
 [statement]
 end for

(same except decrement var at
 end instead of increment,
 and test is put on <)

To use final value several times:
 before test and before the
 repeat block, eval final exp
 repeat:
 call body duplicate just before
 the test
 when for-loop is finished,
 pop that value off.

case (exp) of
 const1: [statement]
 const2: [statement]
 ...
 else: [statement] end

Apr 11-11:22 AM

No full-through with a
 Pascal case statement!
 At most one alternative
 statement is executed.

eval exp
 use long integers for multiple
 comparisons
 optional: use longint !

Array references
 var
 a: array [3..7] of char;
 :

$a[x] := a[y]$
 array references (L-values)
 of array element type
 (char in this case)

as if
 [array diagram]
 L-value: global var ("a")
 evaluate index exp
 convert to offset bit if not done
 subtract initial index
 of the array (L-value only)

array diagram:
 [array diagram]
 L-value: global var ("a")
 evaluate index exp
 convert to offset bit if not done
 subtract initial index
 of the array (L-value only)

L-value: global var ("a")
 evaluate index exp
 convert to offset bit if not done
 subtract initial index
 of the array (L-value only)

Apr 11-11:39 AM

Multiple arrays:
 var
 a: array [3..7, 4..6] of char;
 :

$a[6,5] := 'A'$
 semantic equivalent of
 var
 a: array [3..7] of array [4..6] of char;
 ...
 $a[6][5] := 'A'$

Memory layout is the same
 $a[6]$
 ~~$a[6,5]$~~
 In former case,
 $a[6]$ is illegal (semantic)

In second case:
 $a[6] := a[5]$
 is legal in standard Pascal.
 Array assignment
 $a := b$ where
 a, b are arrays of the
 same type is legal
 entire
 contents of b copied into a
 [note: size of array considered
 part of the type]

We will not support this
 (no assignment of array types
 and never need to use something
 of array type as an L-value.)

Apr 11-11:57 AM

LR parsing (now)
 Code gen and optimization
 (later)

SDTS $\xrightarrow{\text{bison}}$ LR parser

States of the pushdown automaton
 (PDA): sets of LR(0) items

3 types of LR parser
 SLR — LR(0) items
 LR(1) — LR(1) items —
 canonical LR
 $\rightarrow$ LALR — LR(1) items } bison

Let G be a grammar
 and let
 $A \rightarrow X_1 X_2 \dots X_k$
 be some production.

An LR(0)-item is
 an expression of the form
 $A \rightarrow X_1 \dots X_i \cdot X_{i+1} \dots X_k$
 you are

Apr 11-12:09 PM