2013-12-03

**CASES**

Simple implementation can be a glorified if statement.

Pascal Syntax:

Case E of:

[<taglist> : <statement> ]

End

E can be of type char or integer.

Result on the stack returned type T (int, Boolean, char)

E

Each taglist will be a constant of the same type as the result of E.

Case c of

‘A’ : S //single tag

‘B’ .. ‘D’: S //range

‘N’,’O’,’P’: S //list

end

//for single tag generation

dup //duplicate the top of stack

Pushi t

Op\_eq

Jfalse [next]

Pop //instruction to remove the top element of the stack

S //statement to write.

Jmp [end of case]

//for range t1..t2

//check that type(t1) == type(t2) == T

//t1 < t2

dup

pushi t1

OP\_LESS\_THAN

jtrue [nextcase]

dup

pushi t2

OP\_GREATER\_THAN

jtrue [nextcase]

pop

S //jump ended

Additional (Extra) concepts:

Optimise the case statement parsing

On each statement

Generate:

Jmp [decide]

S1

Jmp [end of case] //also copy generate addr(s1),t into a temporary table.

S2

Jmp [end of case]

decide:

JTAB l0,l1,l2,l3,….

Take the value from the stack and jump to the index in the list.

If index is not in list then show error and stop execution.

To optimise we can sort the list.

Normalise the values if the range does not start at 0.

If the list is not sequential, insert end of case for missing labels.

1,2,3,4,6,7 becomes jtab 1,2,3,4,[end of case],6,7

0,1,2,1000001,1000002

//find out the density of the range

(# of tags) / (hi – lo + 1)

Based on the result above we can decide which solution is best to use.

One is to break the range into numbers between 1-2 and 1000001-1000002.

The condition for where to break is variable. But worst case you end up with a binary search, which is still fairly good.

If there appear to be an arithmetic progression, then that can be a way to optimise the case performance.

4-5 office hours.

We can meet professor to show him the status next week. Room #: NA8-202H.