# CS39003 Compilers Laboratory
# Autumn 2025
# Assignment 8
# Date of posting: 03-Nov-2025

---

## Advanced 3-Address Code Generation

This assignment builds upon LA7. Here, you add conditions (Boolean) and if, if-else, and while statements. You are required to do backpatching for generating 3-address instructions in one pass (no need for fall-through optimization). Moreover, your printing of the three-address instructions should show the basic blocks (by single blank lines between consecutive blocks). The modified grammar is given below. The new productions added are highlighted.

```
PROG      →  DECLIST  STMTLIST

DECLIST   →  DECLIST DECL  |  DECL

DECL      →  BASIC  VARLIST  ;  |  struct id { DECLIST }  ;  |  struct id { DECLIST }  VARLIST  ;  |  struct id VARLIST  ;
BASIC     →  int  |  long  |  long int  |  float  |  double
VARLIST   →  VARLIST  ,  VAR  |  VAR
VAR       →  id DIM
DIM       →  [ num ] DIM  |  ε

STMTLIST  →  STMTLIST  STMT  |  ε

STMT      →  ASGN  |
             if  ( BOOL )  { STMTLIST }  |
             if  ( BOOL )  {  STMTLIST  }  else  {  STMTLIST  }  |
             while  ( BOOL )  {  STMTLIST  }

ASGN      →  ITEM  =  EXPR  ;
EXPR      →  EXPR  +  TERM  |  EXPR  –  TERM  |  TERM
TERM      →  TERM  *  FACTOR  |  TERM  /  FACTOR  |  TERM  %  FACTOR  |  FACTOR
FACTOR    →  intconst  |  fltconst  |  ITEM  |  ( EXPR )

ITEM      →  SMPLITEM  |  ITEM  .  SMPLITEM
SMPLITEM  →  id  |  AREF
AREF      →  AREF  [  EXPR  ]  |  id  [  EXPR  ]

BOOL      →  BOOL  ||  BOOL  |  BOOL  &&  BOOL  |  !  BOOL  |  (  BOOL  )  |  EXPR  RELOP  EXPR
RELOP     →  ==  |  !=  |  <  |  <=  |  >  |  >=
```

In order to avoid the dangling-else problem, we mandate the use of braces { … } in if, if-else, and while blocks (even if the block contains a single or no statement). (As an example, Perl enforces this restriction. Perl, however, supports brace-less single-statement blocks as STMT if ( BOOL ) ; )

We allow comparison of numeric operands only. If the two numbers being compared are of different numeric types, then suitable type coercion(s) is/are to be carried out using the widening conventions of LA7.

Modify your code of LA7 in order to incorporate the new additions to the grammar.

### The lex file

The new keywords are if, else, and while. Moreover, lex should now recognize Boolean operators (||, &&, and !) and comparison operators (==, !=, <, <=, >, and >=).

### The yacc file

This file needs substantial changes. In LA7, the 3-address instructions are printed to the terminal as soon as they are generated. Now, we use backpatching. So target-less goto statements cannot be immediately printed, because the jump targets are often available (well) after the generation of the goto statements and are inserted in appropriate instructions using backpatching. This creates a problem because printing cannot go back to earlier lines to complete them.

A compiler usually stores the 3-address instructions in a table of quads or triples, so backpatching can go to the stored instructions generated earlier, and complete them by adding the jump targets left unspecified at the time of the generation of these instructions. Here, we will not go for using quads or triples. Instead we will store the 3-address instructions in strings (one instruction in one string; instruction numbers starting from 0). In a conditional or unconditional jump statement, the jump target is the last element in that instruction. So at the time of backpatching, an earlier incomplete string can be furnished with the additional information by appending the jump target at the end. Use an array of strings (or a two-dimensional character array) for storing all the 3-address instructions. After the entire input is parsed, all these stored instructions are printed in the same sequence as they are generated.

This is the first change you need to make in your (yacc) code for LA7. Maintain a global array of instructions, and a global count (of instructions generated so far). Replace your print-to-terminal calls by appropriate string-library calls (like `sprintf`, `strcat`, and so on).

Another global array is needed for storing the leaders of the blocks (you need to separate consecutive blocks by blank lines). Recall that the first instruction is always a leader. Other leaders are determined by the goto statements. The leader array may be implemented as a list, or as an array of 0's and 1's (1 means leader, 0 means not leader) indexed by instruction numbers. As soon as a jump target is generated (like during explicit goto statements or during backpatching), this array should be updated.

Your next task is to implement the productions for conditional statements and loops, and for Boolean expressions. Use the semantic actions as explained in Section 6.7 of the Dragon Book. You do not have to implement the fall-through optimization technique for minimizing the number of goto statements. Each STMT or STMTLIST now has a synthesized attribute called `nextlist`, whereas each BOOL has two synthesized attributes `truelist` and `falselist`. Define lists of instructions appropriately (like singly linked lists). You need to use C constructs only, because yacc will use these types as the parse-stack types for the above non-terminals. Use the marker non-terminals M and N as explained in the book. The next instruction number is available from the global count of instructions. Also implement the functions `makelist()`, `merge()`, and `backpatch()` (Section 6.7.1 of the Dragon Book).

Your `main()` function will print all the stored instructions **after** `yyparse()` returns. Print a blank line before each leader (so blocks are now visibly separated). Also print an empty instruction at the end (this may be a jump target).

## What to submit

Submit an archive (tar/tgz/zip) consisting of the following files: (i) the lex file *advcodegen.l*, (ii) the yacc file *advcodegen.y*, (iii) any other source/header file(s) that you write, and (iv) a *makefile* with compile, run, and clean targets.

## Sample Input

```
long a, b;
float c;
long Catalan[50];
int i, j, n;

struct matrix {
   int row, col;
   double A[100][100];
} M;

a = 10;
b = a;
c = a + b;

if (c > a) { a = a + 5; b = b + 5; }

if (c > -100 && c < 100 || a != b) { a = a + 5; b = b + 5; } else { a = a - 5; b = b - 5; }

while (c > -100 && c < 100 || a != b && a != c) { a = a + 5; b = b - 5; c = a - b; }

n = 25;
Catalan[0] = 1;
i = 1;
while (i<=n) {
   Catalan[i] = 0;
   j = 0;
   while (j < i) { Catalan[i] = Catalan[i] + Catalan[j] * Catalan[i-j-1]; j = j + 1; }
   i = i + 1;
}

M.row = 10; M.col = 20; i = 0;
while (i < M.row) {
   j = 0;
   while (j < M.col) {
      if (i == j) {
         M.A[i][j] = 0;
      } else {
         if (i > j && i<j+3) {
            M.A[i][j] = i - j;
         } else {
            if (i<j && i>j-3) { M.A[i][j] = j - i; } else { M.A[i][j] = -1; }
         }
      }
      j = j + 1;
   }
   i = i + 1;
}
```

## Output on Sample Input

Present your output in a format similar to what is given below.

```
+++ All declarations read

+++ 8 types
    Type   0:        4    int
    Type   1:        8    long
    Type   2:        4    float
    Type   3:        8    double
    Type   4:      400    array(50,long)
    Type   5:    80008    struct matrix [st = 1]
    Type   6:      800    array(100,double)
    Type   7:    80000    array(100,array(100,double))

+++ Symbol table 0 [main]
    a                     0 - 7          type =    1 = long
    b                     8 - 15         type =    1 = long
    c                    16 - 19         type =    2 = float
    Catalan              20 - 419        type =    4 = array(50,long)
    i                   420 - 423        type =    0 = int
    j                   424 - 427        type =    0 = int
    n                   428 - 431        type =    0 = int
    M                   432 - 80439      type =    5 = struct matrix [st = 1]
    Total width = 80440

+++ Symbol table 1 [struct matrix]
    row                   0 - 3          type =    0 = int
    col                   4 - 7          type =    0 = int
    A                     8 - 80007      type =    7 = array(100,array(100,double))
    Total width = 80008



     0  :  [lng]  t1 = (int2lng)10
     1  :  [lng]  MEM(0,8) = t1
     2  :  [lng]  t2 = MEM(0,8)
     3  :  [lng]  MEM(8,8) = t2
     4  :  [lng]  t3 = MEM(0,8)
     5  :  [lng]  t4 = MEM(8,8)
     6  :  [lng]  t5 = t3 + t4
     7  :  [flt]  t6 = (lng2flt)t5
     8  :  [flt]  MEM(16,4) = t6
     9  :  [flt]  t7 = MEM(16,4)
    10  :  [lng]  t8 = MEM(0,8)
    11  :  [dbl]  t9 = (flt2dbl)t7
    12  :  [dbl]  t10 = (lng2dbl)t8
    13  :         if t9 > t10 goto 15

    14  :         goto 23

    15  :  [lng]  t11 = MEM(0,8)
    16  :  [lng]  t12 = (int2lng)5
    17  :  [lng]  t13 = t11 + t12
    18  :  [lng]  MEM(0,8) = t13
    19  :  [lng]  t14 = MEM(8,8)
    20  :  [lng]  t15 = (int2lng)5
    21  :  [lng]  t16 = t14 + t15
    22  :  [lng]  MEM(8,8) = t16

    23  :  [flt]  t17 = MEM(16,4)
    24  :  [flt]  t18 = (int2flt)-100
    25  :         if t17 > t18 goto 27

    26  :         goto 31

    27  :  [flt]  t19 = MEM(16,4)
    28  :  [flt]  t20 = (int2flt)100
    29  :         if t19 < t20 goto 35

    30  :         goto 31

    31  :  [lng]  t21 = MEM(0,8)
    32  :  [lng]  t22 = MEM(8,8)
    33  :         if t21 != t22 goto 35

    34  :         goto 44

    35  :  [lng]  t23 = MEM(0,8)
    36  :  [lng]  t24 = (int2lng)5
    37  :  [lng]  t25 = t23 + t24
    38  :  [lng]  MEM(0,8) = t25
    39  :  [lng]  t26 = MEM(8,8)
    40  :  [lng]  t27 = (int2lng)5
    41  :  [lng]  t28 = t26 + t27
    42  :  [lng]  MEM(8,8) = t28
    43  :         goto 52

    44  :  [lng]  t29 = MEM(0,8)
    45  :  [lng]  t30 = (int2lng)5
    46  :  [lng]  t31 = t29 - t30
    47  :  [lng]  MEM(0,8) = t31
    48  :  [lng]  t32 = MEM(8,8)
    49  :  [lng]  t33 = (int2lng)5
    50  :  [lng]  t34 = t32 - t33
    51  :  [lng]  MEM(8,8) = t34

    52  :  [flt]  t35 = MEM(16,4)
    53  :  [flt]  t36 = (int2flt)-100
    54  :         if t35 > t36 goto 56

    55  :         goto 60

    56  :  [flt]  t37 = MEM(16,4)
    57  :  [flt]  t38 = (int2flt)100
    58  :         if t37 < t38 goto 70

    59  :         goto 60
```

```
 60  :  [lng]  t39 = MEM(0,8)
 61  :  [lng]  t40 = MEM(8,8)
 62  :         if t39 != t40 goto 64

 63  :         goto 84

 64  :  [lng]  t41 = MEM(0,8)
 65  :  [flt]  t42 = MEM(16,4)
 66  :  [dbl]  t43 = (lng2dbl)t41
 67  :  [dbl]  t44 = (flt2dbl)t42
 68  :         if t43 != t44 goto 70

 69  :         goto 84

 70  :  [lng]  t45 = MEM(0,8)
 71  :  [lng]  t46 = (int2lng)5
 72  :  [lng]  t47 = t45 + t46
 73  :  [lng]  MEM(0,8) = t47
 74  :  [lng]  t48 = MEM(8,8)
 75  :  [lng]  t49 = (int2lng)5
 76  :  [lng]  t50 = t48 - t49
 77  :  [lng]  MEM(8,8) = t50
 78  :  [lng]  t51 = MEM(0,8)
 79  :  [lng]  t52 = MEM(8,8)
 80  :  [lng]  t53 = t51 - t52
 81  :  [flt]  t54 = (lng2flt)t53
 82  :  [flt]  MEM(16,4) = t54
 83  :         goto 52

 84  :  [int]  MEM(428,4) = 25
 85  :  [int]  t55 = 8 * 0
 86  :  [int]  t56 = 20 + t55
 87  :  [lng]  t57 = (int2lng)1
 88  :  [lng]  MEM(t56,8) = t57
 89  :  [int]  MEM(420,4) = 1

 90  :  [int]  t58 = MEM(420,4)
 91  :  [int]  t59 = MEM(428,4)
 92  :         if t58 <= t59 goto 94

 93  :         goto 133

 94  :  [int]  t60 = MEM(420,4)
 95  :  [int]  t61 = 8 * t60
 96  :  [int]  t62 = 20 + t61
 97  :  [lng]  t63 = (int2lng)0
 98  :  [lng]  MEM(t62,8) = t63
 99  :  [int]  MEM(424,4) = 0

100  :  [int]  t64 = MEM(424,4)
101  :  [int]  t65 = MEM(420,4)
102  :         if t64 < t65 goto 104

103  :         goto 129

104  :  [int]  t66 = MEM(420,4)
105  :  [int]  t67 = 8 * t66
106  :  [int]  t68 = 20 + t67
107  :  [int]  t69 = MEM(420,4)
108  :  [int]  t70 = 8 * t69
109  :  [int]  t71 = 20 + t70
110  :  [int]  t72 = MEM(424,4)
111  :  [int]  t73 = 8 * t72
112  :  [int]  t74 = 20 + t73
113  :  [int]  t75 = MEM(420,4)
114  :  [int]  t76 = MEM(424,4)
115  :  [int]  t77 = t75 - t76
116  :  [int]  t78 = t77 - 1
117  :  [int]  t79 = 8 * t78
118  :  [int]  t80 = 20 + t79
119  :  [lng]  t81 = MEM(t74,8)
120  :  [lng]  t82 = MEM(t80,8)
121  :  [lng]  t83 = t81 * t82
122  :  [lng]  t84 = MEM(t71,8)
123  :  [lng]  t85 = t84 + t83
124  :  [lng]  MEM(t68,8) = t85
125  :  [int]  t86 = MEM(424,4)
126  :  [int]  t87 = t86 + 1
127  :  [int]  MEM(424,4) = t87
128  :         goto 100

129  :  [int]  t88 = MEM(420,4)
130  :  [int]  t89 = t88 + 1
131  :  [int]  MEM(420,4) = t89
132  :         goto 90

133  :  [int]  t90 = 432 + 0
134  :  [int]  MEM(t90,4) = 10
135  :  [int]  t91 = 432 + 4
136  :  [int]  MEM(t91,4) = 20
137  :  [int]  MEM(420,4) = 0

138  :  [int]  t92 = MEM(420,4)
139  :  [int]  t93 = 432 + 0
140  :  [int]  t94 = MEM(t93,4)
141  :         if t92 < t94 goto 143

142  :         goto 224

143  :  [int]  MEM(424,4) = 0

144  :  [int]  t95 = MEM(424,4)
145  :  [int]  t96 = 432 + 4
146  :  [int]  t97 = MEM(t96,4)
147  :         if t95 < t97 goto 149

148  :         goto 220

149  :  [int]  t98 = MEM(420,4)
150  :  [int]  t99 = MEM(424,4)
151  :         if t98 == t99 goto 153

152  :         goto 163
```

```
153  :  [int]  t100 = MEM(420,4)
154  :  [int]  t101 = 800 * t100
155  :  [int]  t102 = 8 + t101
156  :  [int]  t103 = MEM(424,4)
157  :  [int]  t104 = 8 * t103
158  :  [int]  t105 = t102 + t104
159  :  [int]  t106 = 432 + t105
160  :  [dbl]  t107 = (int2dbl)0
161  :  [dbl]  MEM(t106,8) = t107
162  :         goto 216

163  :  [int]  t108 = MEM(420,4)
164  :  [int]  t109 = MEM(424,4)
165  :         if t108 > t109 goto 167

166  :         goto 185

167  :  [int]  t110 = MEM(420,4)
168  :  [int]  t111 = MEM(424,4)
169  :  [int]  t112 = t111 + 3
170  :         if t110 < t112 goto 172

171  :         goto 185

172  :  [int]  t113 = MEM(420,4)
173  :  [int]  t114 = 800 * t113
174  :  [int]  t115 = 8 + t114
175  :  [int]  t116 = MEM(424,4)
176  :  [int]  t117 = 8 * t116
177  :  [int]  t118 = t115 + t117
178  :  [int]  t119 = 432 + t118
179  :  [int]  t120 = MEM(420,4)
180  :  [int]  t121 = MEM(424,4)
181  :  [int]  t122 = t120 - t121
182  :  [dbl]  t123 = (int2dbl)t122
183  :  [dbl]  MEM(t119,8) = t123
184  :         goto 216

185  :  [int]  t124 = MEM(420,4)
186  :  [int]  t125 = MEM(424,4)
187  :         if t124 < t125 goto 189

188  :         goto 207

189  :  [int]  t126 = MEM(420,4)
190  :  [int]  t127 = MEM(424,4)
191  :  [int]  t128 = t127 - 3
192  :         if t126 > t128 goto 194

193  :         goto 207

194  :  [int]  t129 = MEM(420,4)
195  :  [int]  t130 = 800 * t129
196  :  [int]  t131 = 8 + t130
197  :  [int]  t132 = MEM(424,4)
198  :  [int]  t133 = 8 * t132
199  :  [int]  t134 = t131 + t133
200  :  [int]  t135 = 432 + t134
201  :  [int]  t136 = MEM(424,4)
202  :  [int]  t137 = MEM(420,4)
203  :  [int]  t138 = t136 - t137
204  :  [dbl]  t139 = (int2dbl)t138
205  :  [dbl]  MEM(t135,8) = t139
206  :         goto 216

207  :  [int]  t140 = MEM(420,4)
208  :  [int]  t141 = 800 * t140
209  :  [int]  t142 = 8 + t141
210  :  [int]  t143 = MEM(424,4)
211  :  [int]  t144 = 8 * t143
212  :  [int]  t145 = t142 + t144
213  :  [int]  t146 = 432 + t145
214  :  [dbl]  t147 = (int2dbl)-1
215  :  [dbl]  MEM(t146,8) = t147

216  :  [int]  t148 = MEM(424,4)
217  :  [int]  t149 = t148 + 1
218  :  [int]  MEM(424,4) = t149
219  :         goto 144

220  :  [int]  t150 = MEM(420,4)
221  :  [int]  t151 = t150 + 1
222  :  [int]  MEM(420,4) = t151
223  :         goto 138

224:
```