CS39003 Compilers Laboratory
Autumn 2025
Assignment 7
Date of posting: 27-Oct-2025

Three-address code generation for simple statements

This assignment is an extension of the last assignment (LA6). Your input program starts with a set of variable declarations and structure definitions. For simplicity, we restrict the basic types to int, long int, float, and double only. Moreover, the types of the variables can only be these basic types, (one- and multi-dimensional) arrays, and structures. Arrays of structures are allowed, so also are arrays and structures within structures. We remove pointers from the grammar (because of simplicity, and also because memory allocations to pointers call for dynamic memory management, a topic not covered in this course).

These declarations are followed by a sequence of assignment statements of the following form.

```
l-value = r-value ;
```

Here, both the `l-value` and the `r-value` must be of the basic data types (numeric only). But we should allow them to have different types. The `r-value` is an expression involving numeric (integer and floating-point) arithmetic only. Automatic type conversion is to be made in arithmetic operations, and also during assignments. We use the following widening conventions among the basic types. Each arrow indicates widening.

$$
\begin{array}{ccc}
\text{long} & \longrightarrow & \text{double} \\
\uparrow & & \uparrow \\
\text{int} & \longrightarrow & \text{float}
\end{array}
$$

The grammar that your program for this assignment deals with is given below. The terminal symbols are in red. The new/updated productions (in reference to the grammar of LA6) are highlighted in yellow.

```
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
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 ]
```

Your code will print the type table and the symbols tables (in the same format as in LA6), followed by a sequence of 3-address instructions generated from the assignment statements. There is no need to store the 3-address instructions in quads or triples.

A memory location is accessed in (written to) an `l-value`. The `r-value`, on the other hand, may also contain one or more numeric operands stored in the memory. We assume that there is a single memory segment for all the variables. A numeric operand is specified as `MEM(offset, width)`. Here, `offset` is the offset (in bytes) of the numeric variable in the data segment, and `width` is the size of the numeric variable. We assume that int and float are of size 4 bytes, and long and double are of size 8 bytes. Consider the following declarations.

```
long a, b;
int c, A[5][5];

struct mystruct { double x; float A[10]; } S, SA[4][4];
```

These variables are stored in the memory segment as follows.

`a`		0 – 7	width = 8
`b`		8 – 15	width = 8
`c`		16 – 19	width = 4
`A`		20 – 119	width = 100
`S`		120 – 167	width = 8 + 40 = 48
	`S.x`	120 – 127	width = 8
	`S.A`	128 – 167	width = 40
`SA`		168 – 935	width = 48 × 16 = 768

Therefore, `a` is referred to as `MEM(0,8)`, `b` as `MEM(8,8)`, `c` as `MEM(16,4)`, `A[2][3]` as `MEM(20 + 20 × 2 + 4 × 3 = 72,4)`, `S.x` as `MEM(120,8)`, and `SA[1][2].A[3]` as `MEM(168 + (1 × 4 + 2) × 48 + 8 + 4 × 3 = 476,4)`. Non-numeric types like `A[2]`, `SA[1][2]`, and `SA[1][2].A` are never to be used as an `l-value`, or as an `r-value`, or as an operand of an arithmetic operation. All offsets in the memory segment are assumed to be int-valued.

Against all intermediate grammar symbols in an expression or `l-value`, we maintain the type of that variable. This type is not necessarily basic. For example, `SA[1][2].A[3]` is parsed successively as `SA[1]`, `SA[1][2]`, and `SA[1][2].A[3]`. During an arithmetic operation, we use the widening conventions of basic data types. For example, the addition of an int and a long will first typecast the int to a long, and then the addition is carried out, and the multiplication of a long and a float will typecast both to double. Moreover, during an assignment, `l-value = r-value;` the `r-value` is converted to the type of `l-value` (if these two types are different); this conversion may be a case of widening or one of narrowing. We use the operator `(src2dst)` to convert from the source type `src` to the destination type `dst`. For example, `t56 = (lng2dbl)t48` widens the temporary `t48` of type long to the temporary `t56` of type double. If `t56` is is to be stored in an int variable in memory, then we need a narrowing: `t57 = (dbl2int)t56`. This will be followed by the store instruction `MEM(1234,4) = t57`.

Lex file

Update your lex file of Assignment 6 as follows. Remove many basic data types as keywords. Here, we deal only with the basic types int, long (also called long int), float, and double. Add new punctuation symbols: dot (identifying fields in structures), arithmetic operators, the assignment operator, and parentheses (for grouping in expressions). Finally, identify lexemes that qualify as intconst and fltconst (note that num used in array dimensions is also an intconst).

Yacc file

Update the productions for BASIC (remove the extra basic types). Also remove productions involving pointers. Add the new productions highlighted on the last page.

As in LA6, read the declarations at the beginning of the code, and store all relevant information in a global type table `TT` and multiple symbol tables `ST` (`ST[0]` is for the data segment, `ST[1]`, `ST[2]`, … are for individual structure definitions).

Every non-terminal grammar symbol appearing under EXPR and ITEM has an address `addr` (not a memory address or an offset, but a reference to a three-address instruction). This `addr` can be one of the five categories: (i) an integer constant (`intconst`), (ii) a floating-point constant (`fltconst`), (iii) the number of a temporary storing an intermediate numeric result (`temp`), (iv) an absolute memory offset in the data segment (`offset`), and (v) the number of a temporary storing a calculated offset in the data segment (`toffset`). The value of an `addr` is a union of the data types of the five possibilities. Moreover, we need to maintain the data type of each `addr`, because type checking and type conversion are needed for arithmetic and assignment operations. We summarize the components of `addr` in the table below.

Category	**Data type of value**	**Type of the non-terminal**	**Examples**
`intconst`	int	Index of `INT` in `TT`	`intconst`
`fltconst`	double	Index of `DBL` in `TT`	`fltconst`
`temp`	numeric	Index of `INT`/`LNG`/`FLT`/`DBL` in `TT`	`FACTOR`, `TERM`, `EXPR`
`offset`	int	Index in `TT` of type of data at offset	`ID`
`toffset`	int	Index in `TT` of type of data at offset, stored in a temporary	`AREF`, `ITEM.SMPLITEM`

As an example, consider how ITEM derives `SA[1][2].A[3]` (rightmost derivation).

ITEM → ITEM . SMPLITEM → ITEM . AREF → ITEM . ID[NUM] → AREF . ID[NUM] → AREF[NUM] . ID[NUM] → ID[NUM][NUM] . ID[NUM]

The reduction process generates the following intermediate addresses (calculated in temporaries of category `toffset`).

	Category	Data type of value	Type (index in `TT` of)	Offset in data segment / structure
`SA[1]`	`toffset`	`int`	`array(4, struct mystruct)`	$168 + 192 \times 1 = 360$
`SA[1][2]`	`toffset`	`int`	`struct mystruct`	$360 + 48 \times 2 = 454$
`A[3]`	`toffset`	`int`	`float`	$8 + 4 \times 3 = 20$
`SA[1][2].A[3]`	`toffset`	`int`	`float`	$454 + 20 = 476$

It is to be noted that `A[3]` in the above example illustrates an entry to be found in the symbol table for `struct mystruct`. The global variable `A[3]` (without the structure reference) is also an `addr` of category `toffset`, value `int`, type `array(5, float)`, and offset $20 + 20 \times 3 = 80$ (in data segment). We can use `SA[1][2].A[3]` as an `l-value` or as an operand in an `r-value` (because its type is `float`), but the global `A[3]` cannot be used like that (because it is an array and not of a numeric type).

Since variable names are stored in different symbol tables, it is mandatory to use the correct symbol-table number. By default, a variable name refers to the global symbol table `ST[0]`. As soon as the dot operator is encountered (indicating a reference to a structure), we must have ITEM . at the top of the parse stack. The type of this ITEM must be a reference to a structure type in `TT`. This entry in the `TT` stores the symbol-table number for the structure. Use a marker non-terminal to pass on this symbol-table number as the name-space for the SIMPLITEM that will later be pushed to the parse stack to complete the handle ITEM . SIMPLITEM.

Other than this, all expressions are evaluated in a synthesized manner. An operand in an expression can be either a value stored in a temporary or the memory offset stored in a temporary. A `toffset`-category `addr` is first loaded to a temporary. Then, the operation is carried out, and the result is stored in a `temp`-category `addr`. Before each arithmetic operation, a check for type compatibility and type coercion(s) (if necessary) are to be carried out.

Finally, an assignment `l-value = r-value;` the `l-value` should have a valid memory offset and a numeric type. The `r-value`, on the other hand, can be of different categories (constant, temporary, memory offset, or a memory offset stored in a temporary). All these cases should be handled. Moreover, data stored in `r-value` should be converted to the type of `l-value` (if necessary) before the storage is done.

The `main()` function written in the yacc file or elsewhere should redirect lex's input to a file name. The declarations are first read, and the type table and all the symbol tables are printed in the same format as in Assignment 6. This is followed by a printing of the 3-address instructions generated from the assignment statements, in a format described in the following sample output.

What to submit

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

Sample Output

Consider the following input file.

```
long a, b;
float c;

int A[5][10];

struct coll {
   float f;
   double d;
   long x;
   int A[100];
};

struct coll S, T[10][10];

struct bigcoll {
   long n;
   struct coll C[5][5];
} BC[10];

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

A[1][2] = 2;
A[2][3] = A[1][2] + 5;
A[4][5] = A[1][2] - A[2][3] + c * 123;

T[5][5].x = 100.;
T[5][5].d = T[5][5].x;
S.A[25] = (a + b) * (a - c);
S.x = S.A[25];

BC[5].C[4][3].x = 100;
BC[6].C[5][4].A[50] = BC[5].C[4][3].x;
BC[6].C[5][4].A[80] = BC[5].C[4][3].A[50] + 345.;

BC[A[a/b][c/b]].n = A[a-6][BC[6].C[5][4].A[80]/BC[5].C[4][3].x+1.35792468];

BC[6].C[5][4] = T[7][7];
```

The output on this file may be presented in the following format.

```
+++ All declarations read

+++ 14 types
    Type   0:        4    int
    Type   1:        8    long
    Type   2:        4    float
    Type   3:        8    double
    Type   4:       40    array(10,int)
    Type   5:      200    array(5,array(10,int))
    Type   6:      420    struct coll [st = 1]
    Type   7:      400    array(100,int)
    Type   8:     4200    array(10,struct coll [st = 1])
    Type   9:    42000    array(10,array(10,struct coll [st = 1]))
    Type  10:    10508    struct bigcoll [st = 2]
    Type  11:     2100    array(5,struct coll [st = 1])
    Type  12:    10500    array(5,array(5,struct coll [st = 1]))
    Type  13:   105080    array(10,struct bigcoll [st = 2])

+++ Symbol table 0 [main]
    a                     0 - 7           type =    1 = long
    b                     8 - 15          type =    1 = long
    c                    16 - 19          type =    2 = float
    A                    20 - 219         type =    5 = array(5,array(10,int))
    S                   220 - 639         type =    6 = struct coll [st = 1]
    T                   640 - 42639       type =    9 = array(10,array(10,struct coll [st = 1]))
    BC                42640 - 147719      type =   13 = array(10,struct bigcoll [st = 2])
    Total width = 147720

+++ Symbol table 1 [struct coll]
    f                     0 - 3           type =    2 = float
    d                     4 - 11          type =    3 = double
    x                    12 - 19          type =    1 = long
    A                    20 - 419         type =    7 = array(100,int)
    Total width = 420

+++ Symbol table 2 [struct bigcoll]
    n                     0 - 7           type =    1 = long
    C                     8 - 10507       type =   12 = array(5,array(5,struct coll [st = 1]))
    Total width = 10508


     [lng] t1 = (int2lng)10
     [lng] MEM(0,8) = t1

     [lng] t2 = MEM(0,8)
     [lng] MEM(8,8) = t2

     [lng] t3 = MEM(0,8)
     [lng] t4 = MEM(8,8)
     [lng] t5 = t3 + t4
     [flt] t6 = (lng2flt)t5
     [flt] MEM(16,4) = t6

     [int] t7 = 40 * 1
     [int] t8 = 20 + t7
     [int] t9 = 4 * 2
     [int] t10 = t8 + t9
     [int] MEM(t10,4) = 2

     [int] t11 = 40 * 2
     [int] t12 = 20 + t11
     [int] t13 = 4 * 3
     [int] t14 = t12 + t13
```

```
[int] t15 = 40 * 1
[int] t16 = 20 + t15
[int] t17 = 4 * 2
[int] t18 = t16 + t17
[int] t19 = MEM(t18,4)
[int] t20 = t19 + 5
[int] MEM(t14,4) = t20

[int] t21 = 40 * 4
[int] t22 = 20 + t21
[int] t23 = 4 * 5
[int] t24 = t22 + t23
[int] t25 = 40 * 1
[int] t26 = 20 + t25
[int] t27 = 4 * 2
[int] t28 = t26 + t27
[int] t29 = 40 * 2
[int] t30 = 20 + t29
[int] t31 = 4 * 3
[int] t32 = t30 + t31
[int] t33 = MEM(t28,4)
[int] t34 = MEM(t32,4)
[int] t35 = t33 - t34
[flt] t36 = MEM(16,4)
[flt] t37 = (int2flt)123
[flt] t38 = t36 * t37
[flt] t39 = (int2flt)t35
[flt] t40 = t39 + t38
[int] t41 = (flt2int)t40
[int] MEM(t24,4) = t41

[int] t42 = 4200 * 5
[int] t43 = 640 + t42
[int] t44 = 420 * 5
[int] t45 = t43 + t44
[int] t46 = t45 + 12
[lng] t47 = (dbl2lng)100.0000000000000000
[lng] MEM(t46,8) = t47

[int] t48 = 4200 * 5
[int] t49 = 640 + t48
[int] t50 = 420 * 5
[int] t51 = t49 + t50
[int] t52 = t51 + 4
[int] t53 = 4200 * 5
[int] t54 = 640 + t53
[int] t55 = 420 * 5
[int] t56 = t54 + t55
[int] t57 = t56 + 12
[lng] t58 = MEM(t57,8)
[dbl] t59 = (lng2dbl)t58
[dbl] MEM(t52,8) = t59

[int] t60 = 4 * 25
[int] t61 = 20 + t60
[int] t62 = 220 + t61
[lng] t63 = MEM(0,8)
[lng] t64 = MEM(8,8)
[lng] t65 = t63 + t64
[lng] t66 = MEM(0,8)
[flt] t67 = MEM(16,4)
[dbl] t68 = (lng2dbl)t66
[dbl] t69 = (flt2dbl)t67
[dbl] t70 = t68 - t69
[dbl] t71 = (lng2dbl)t65
[dbl] t72 = t71 * t70
[int] t73 = (dbl2int)t72
[int] MEM(t62,4) = t73

[int] t74 = 220 + 12
[int] t75 = 4 * 25
[int] t76 = 20 + t75
[int] t77 = 220 + t76
[int] t78 = MEM(t77,4)
[lng] t79 = (int2lng)t78
[lng] MEM(t74,8) = t79

[int] t80 = 10508 * 5
[int] t81 = 42640 + t80
[int] t82 = 2100 * 4
[int] t83 = 8 + t82
[int] t84 = 420 * 3
[int] t85 = t83 + t84
[int] t86 = t81 + t85
[int] t87 = t86 + 12
[lng] t88 = (int2lng)100
[lng] MEM(t87,8) = t88

[int] t89 = 10508 * 6
[int] t90 = 42640 + t89
[int] t91 = 2100 * 5
[int] t92 = 8 + t91
[int] t93 = 420 * 4
[int] t94 = t92 + t93
[int] t95 = t90 + t94
[int] t96 = 4 * 50
[int] t97 = 20 + t96
[int] t98 = t95 + t97
[int] t99 = 10508 * 5
[int] t100 = 42640 + t99
[int] t101 = 2100 * 4
[int] t102 = 8 + t101
[int] t103 = 420 * 3
[int] t104 = t102 + t103
[int] t105 = t100 + t104
[int] t106 = t105 + 12
[lng] t107 = MEM(t106,8)
[int] t108 = (lng2int)t107
[int] MEM(t98,4) = t108

[int] t109 = 10508 * 6
[int] t110 = 42640 + t109
[int] t111 = 2100 * 5
```

```
    [int] t112 = 8 + t111
    [int] t113 = 420 * 4
    [int] t114 = t112 + t113
    [int] t115 = t110 + t114
    [int] t116 = 4 * 80
    [int] t117 = 20 + t116
    [int] t118 = t115 + t117
    [int] t119 = 10508 * 5
    [int] t120 = 42640 + t119
    [int] t121 = 2100 * 4
    [int] t122 = 8 + t121
    [int] t123 = 420 * 3
    [int] t124 = t122 + t123
    [int] t125 = t120 + t124
    [int] t126 = 4 * 50
    [int] t127 = 20 + t126
    [int] t128 = t125 + t127
    [int] t129 = MEM(t128,4)
    [dbl] t130 = (int2dbl)t129
    [dbl] t131 = t130 + 345.0000000000000000
    [int] t132 = (dbl2int)t131
    [int] MEM(t118,4) = t132

    [lng] t133 = MEM(0,8)
    [lng] t134 = MEM(8,8)
    [lng] t135 = t133 / t134
    [int] t136 = (lng2int)t135
    [int] t137 = 40 * t136
    [int] t138 = 20 + t137
    [flt] t139 = MEM(16,4)
    [lng] t140 = MEM(8,8)
    [dbl] t141 = (flt2dbl)t139
    [dbl] t142 = (lng2dbl)t140
    [dbl] t143 = t141 / t142
    [int] t144 = (dbl2int)t143
    [int] t145 = 4 * t144
    [int] t146 = t138 + t145
    [int] t147 = MEM(t146,4)
    [int] t148 = 10508 * t147
    [int] t149 = 42640 + t148
    [int] t150 = t149 + 0
    [lng] t151 = MEM(0,8)
    [lng] t152 = (int2lng)6
    [lng] t153 = t151 - t152
    [int] t154 = (lng2int)t153
    [int] t155 = 40 * t154
    [int] t156 = 20 + t155
    [int] t157 = 10508 * 6
    [int] t158 = 42640 + t157
    [int] t159 = 2100 * 5
    [int] t160 = 8 + t159
    [int] t161 = 420 * 4
    [int] t162 = t160 + t161
    [int] t163 = t158 + t162
    [int] t164 = 4 * 80
    [int] t165 = 20 + t164
    [int] t166 = t163 + t165
    [int] t167 = 10508 * 5
    [int] t168 = 42640 + t167
    [int] t169 = 2100 * 4
    [int] t170 = 8 + t169
    [int] t171 = 420 * 3
    [int] t172 = t170 + t171
    [int] t173 = t168 + t172
    [int] t174 = t173 + 12
    [int] t175 = MEM(t166,4)
    [lng] t176 = MEM(t174,8)
    [lng] t177 = (int2lng)t175
    [lng] t178 = t177 / t176
    [dbl] t179 = (lng2dbl)t178
    [dbl] t180 = t179 + 1.3579246800000000
    [int] t181 = (dbl2int)t180
    [int] t182 = 4 * t181
    [int] t183 = t156 + t182
    [int] t184 = MEM(t183,4)
    [lng] t185 = (int2lng)t184
    [lng] MEM(t150,8) = t185

    [int] t186 = 10508 * 6
    [int] t187 = 42640 + t186
    [int] t188 = 2100 * 5
    [int] t189 = 8 + t188
    [int] t190 = 420 * 4
    [int] t191 = t189 + t190
    [int] t192 = t187 + t191
    [int] t193 = 4200 * 7
    [int] t194 = 640 + t193
    [int] t195 = 420 * 7
    [int] t196 = t194 + t195

*** Error: invalid type of l-value
```