A DETAILED DESCRIPTION OF CODE GENERATION (C-F07)
Robert Heckendorn
Dec 3, 2007

==========
Initial program fed to compiler:

```
int g;                  // a global array

int dog(int x, y)       // two arguments
{
        int z;          // local variable

        z = x + y;      // expression using local vars

        return g*z;     // expression using local and global vars
}

int h[10];              // global array

void cat(int x[], y)     // two arguments one of which is an array
{
        int z[10];      // a local array

        z[8] = z[9];    // regular local array stuff
        h[8] = h[9];    // regular global array stuff
        z[7] = x[y];    // parm array on right hand side
        x[7] = z[y];    // parm array on left hand side
}                       // use default return

void main()
{
        int a[10];      // local array
        int b;          // local var

        dog(b, 999);      // call dog
        cat(a, 6);        // call cat
        b = dog(777,888);  // get return value from call to dog
}
```

==========
Semantic Analysis

Besides doing the type checking of the tree during traversal this
will annotate the nodes in the tree with the data necessary for
code generation phase. FOFFSET is used to hold the local offset
GOFFSET is used to hold the global offset.

```
                        // init goffset=0

int g;                  // decl node is annotated with
                        //   location: 0, size: 1, referenceType: global
                        // update goffset=1
```

```
                        // enter scope of dog in symbol table
                        // foffset=-2

int dog(int x, y)       // decl node for x is annotated with
                        //   location: -2, size: 1, referenceType: parameter
                        // update local offset
                        // decl node for y is annotated with
                        //   location: -3, size: 1, referenceType: parameter
                        // update local offset
{
        int z;          // decl node for z is annotated with
                        //   location: -4, size: 1, referenceType: local
                        // update local offset
        z = x + y;      // annotate the id nodes with location/size/referenceType
via
                        //   lookup in symbol table
        return g*z;     // annotate the id nodes with location/size/referenceType
via
                        //   lookup in symbol table
}
                        // save size of frame (5) for dog in global symbol dog.
                        //     This is in foffset.
                        // exit scope of dog in symbol table

int h[10];              // decl node is for global variable annotated with
                        //   location: -2, size: 10+1 for array size,
referenceType: global
                        //   allocate 1 for size, then 10 for the array it self.
Location
                        //   is array itself
                        // update goffset=-12

                        // enter scope of cat in symbol table
                        // foffset=-2

void cat(int x[], y)    // decl node for x is annotated with
                        //   location: -2, size: 1, referenceType: parameter
                        //   (arrays are passed by reference and so size is 1.
                        //      see code generation section.)
                        // update local offset foffset=-3
                        // decl node for y is annotated with
                        //   location: -3, size: 1, referenceType: parameter
                        // update local offset foffset=-4
{
        int z[10];      // decl node for z is annotated with
                        //   location: -5, size: 10+1 for size, referenceType:
local
                        // update local offset foffset=-15

        z[8] = z[9];    // annotate the id nodes with location/size/referenceType
via
                        //   lookup in symbol table
        h[8] = h[9];    // annotate the id nodes with location/size/referenceType
via
                        //   lookup in symbol table
        z[7] = x[y];    // annotate the id nodes with location/size/referenceType
via
```

```
                        //   lookup in symbol table
        x[7] = z[y];    // annotate the id nodes with location/size/referenceType
via
                        //   lookup in symbol table
}

                        // save size of frame (15) for cat in global symbol cat.
                        //    This is the value of the foffset.
                        // exit scope of cat in symbol table

                        // enter scope of main in symbol table
                        // foffset=-2
void main()
{
        int a[10];      // decl node for a is annotated with
                        //   location: -3, size: 10+1, referenceType: local
                        // update local offset  foffset=-13
        int b;          // decl node for b is annotated with
                        //   location: -13, size: 1, referenceType: local
                        // update local offset

        dog(b, 999);    // annotate the id node with location/size/referenceType
via
                        //   lookup in symbol table
                        // annotate the func node with size via symbol table
        cat(a, 6);      // annotate the id node with location/size/referenceType
via
                        //   lookup in symbol table
                        // annotate the func node with size via symbol table
        b = dog(777,888);  // annotate id nodes with location/size/referenceType via
                        //   lookup in symbol table
                        // annotate the func node with size via symbol table
}

                        // save size of frame (14) for main in global
                        // symbol dog.  This is foffset.  exit scope
                        // of cat in symbol table
```

==========

Code Generation

Now we traverse the tree again using the information at each node to tell us what instructions to generate. The information we need will be:

1. the size of the global space so you know where to put the frame for main. Get this from goffset.

2. the symbol table, which at the end of the semantic phase contains all the global symbols. This is used to look up a function location at call time.

3. the annotated tree from the semantic phase. This has all of the sizes and locations of the data elements attached to each ID

You will need to keep track of a compile-time variable called toffset

which is the temporary offset during compilation.

First let's look at the memory layout by looking at the example code below and the configuration of memory using tm. We will stop the code at four spots and look at memory. This can be done using the breakpoint command in tm.

Consider this code:

```
int g;

int dog(int x, y)
{
        int z;
        z = 300;
                        <- stop point D at SECOND call to dog inside expression
                        <- stop point B in first call to dog
        z = x + y;
        return g*z;
}

int h[10];

void cat(int x[], y)
{
        int z[10];

        z[0] = 500;
        z[9] = 509;
                        <- stop point C
        z[8] = z[9];
        h[8] = h[9];
        z[7] = x[y];
        x[7] = z[y];
}

void main()
{
        int a[10];
        int b;

        a[0] = 100;
        a[9] = 109;
        b = 200;
        g = 300;
        h[0] = 400;
        h[9] = 409;
                      <-  stop point A
        dog(b, 999);
        cat(a, 6);
        b = 555 * (666 + dog(777,888));
}
```

At stop point A:

```
 Enter command: r
 r[0]: 9999   r[1]: 9987   r[2]: 0      r[3]: 409
 r[4]: 9      r[5]: 9988   r[6]: 0      r[7]: 148


 Enter command: d 9999 -40
  addr: value    instr that last assigned this loc
  9999:   300    133         <- g        <- reg 0 (globals)  <
  9998:    10    190         <- *h                           |
  9997:   400    140         <- h[0]                         |
  9996:     0    unused                                      |
  9995:     0    unused                                      |
  9994:     0    unused                                      |
  9993:     0    unused                                   globals
  9992:     0    unused                                      |
  9991:     0    unused                                      |
  9990:     0    unused                                      |
  9989:     0    unused                                      |
  9988:   409    147         <- h[9]                         <
  9987:  9987    192         <- old frame pointer  <- reg 1  <
  9986:   195    113         <- return address               |
  9985:    10    115         <- *a                           |
  9984:   100    122         <- a[0]                         |
  9983:     0    unused                                      |
  9982:     0    unused                                      |
  9981:     0    unused                                      |
  9980:     0    unused                                   frame for
  9979:     0    unused                                    main
  9978:     0    unused                                      |
  9977:     0    unused                                      |
  9976:     0    unused                                      |
  9975:   109    129         <- a[9]                         |
  9974:   200    131         <- b                            <
  9973:     9    142         <- was temp used in computing subscript 9 in previous
 expression
  9972:     0    unused
  9971:     0    unused
  9970:     0    unused
  9969:     0    unused
  9968:     0    unused
  9967:     0    unused
  9966:     0    unused
  9965:     0    unused
  9964:     0    unused


 --------------------
 At stop point B:

 Enter command: d 9999 -30
 addr: value    instr that last assigned this loc
  9999:   300    133     <- g        <- reg 0 (globals)  <
  9998:    10    190     <- *h                           |
  9997:   400    140     <- h[0]                         |
  9996:     0    unused                                  |
  9995:     0    unused                                  |
  9994:     0    unused                                  |
  9993:     0    unused                               globals
```

```
 9992:     0    unused                                    |
 9991:     0    unused                                    |
 9990:     0    unused                                    |
 9989:     0    unused                                    |
 9988:   409    147     <- h[9]                           <
 9987:  9987    192     <- old frame pointer              <
 9986:   195    113     <- return address                 |
 9985:    10    115     <- *a                             |
 9984:   100    122     <- a[0]                           |
 9983:     0    unused                                    |
 9982:     0    unused                                    |
 9981:     0    unused                                    |
 9980:     0    unused                                 frame for
 9979:     0    unused                                  main
 9978:     0    unused                                    |
 9977:     0    unused                                    |
 9976:     0    unused                                    |
 9975:   109    129     <- a[9]                           |
 9974:   200    131     <- b                              <
 9973:  9987    148     <- old frame pointer  <- reg 1    <
 9972:   156    30      <- return address                 |
 9971:   200    150     <- parm 1                      dog frame
 9970:   999    152     <- parm 2                         |
 9969:   300    32      <- z                              <
 9968:     0    unused
 9967:     0    unused
 9966:     0    unused
 9965:     0    unused
 9964:     0    unused
 9963:     0    unused


 --------------------
 At stop point C:

 Enter command: r
 r[0]: 9999   r[1]: 9973   r[2]: 359700   r[3]: 509
 r[4]: 9      r[5]: 9959   r[6]: 0      r[7]: 69

 Enter command: d 9999 -50
  addr: value    instr that last assigned this loc
  9999:   300    133       <- g        <- reg 0 (globals)  <
  9998:    10    190       <- *h
  9997:   400    140       <- h[0]                           |
  9996:     0    unused                                      |
  9995:     0    unused                                      |
  9994:     0    unused                                      |
  9993:     0    unused                                   globals
  9992:     0    unused                                      |
  9991:     0    unused                                      |
  9990:     0    unused                                      |
  9989:     0    unused                                      |
  9988:   409    147       <- h[9]                           <
  9987:  9987    192       <- old frame pointer              <
  9986:   195    113       <- return address                 |
  9985:    10    115       <- *a
  9984:   100    122       <- a[0]                           |
  9983:     0    unused                                      |
```

```
 9982:     0    unused                                        |
 9981:     0    unused                                        |
 9980:     0    unused                                    frame for
 9979:     0    unused                                     main
 9978:     0    unused                                        |
 9977:     0    unused                                        |
 9976:     0    unused                                        |
 9975:   109    129       <- a[9]                             |
 9974:   200    131       <- b                                <
 9973:  9987    157       <- old frame pointer  <- reg 1      <
 9972:   165    52        <- return address                   |
 9971:  9984    159       <- parm 1 ptr to array x            |
 9970:     6    161       <- parm 2 y                         |
 9969:    10    54        <- *z                               |
 9968:   500    61        <- z[0]                             |
 9967:     0    unused                                        |
 9966:     0    unused                                   frame for
 9965:     0    unused                                      cat
 9964:     0    unused                                        |
 9963:     0    unused                                        |
 9962:     0    unused                                        |
 9961:     0    unused                                        |
 9960:     0    unused                                        |
 9959:   509    68        <- z[9]                             <
 9958:     9    63        <- temp
 9957:     0    unused


 --------------------
 At stop point D

 Enter command: r
 r[0]: 9999   r[1]: 9971   r[2]: 0      r[3]: 300
 r[4]: 7      r[5]: 9977   r[6]: 0      r[7]: 33

 Enter command: d 9999 -60
 addr: value    instr that last assigned this loc
  9999:   300    133       <- g        <- reg 0 (globals)  <
  9998:    10    190       <- *h
  9997:   400    140       <- h[0]                             |
  9996:     0    unused                                        |
  9995:     0    unused                                        |
  9994:     0    unused                                        |
  9993:     0    unused                                    globals
  9992:     0    unused                                        |
  9991:     0    unused                                        |
  9990:     0    unused                                        |
  9989:   409    88                                            |
  9988:   409    147                                           <
  9987:  9987    192       <- old frame pointer                <
  9986:   195    113       <- return address                   |
  9985:    10    115       <- *a
  9984:   100    122       <- a[0]                             |
  9983:     0    unused                                        |
  9982:     0    unused                                        |
  9981:     0    unused                                        |
  9980:     0    unused                                    frame for
  9979:     0    unused                                     main
```

```
 9978:     0    unused                                        |
 9977:     0    108                                           |
 9976:     0    unused                                        |
 9975:   109    129       <- a[9]                             |
 9974:   200    131       <- b                                <
 9973:   555    167       <- temp!
 9972:   666    169       <- temp!
 9971:  9987    170       <- old frame pointer  <- reg 1  <
 9970:   178    30        <- return address (see below)   |
 9969:   777    172       <- parm 1                    dog frame
 9968:   888    174       <- parm 2                       |
 9967:   300    32        <- z                            <
 9966:     0    unused    <- temp!
 9965:     0    unused
 9964:     0    unused
 9963:     0    unused
 9962:     0    unused
 9961:     0    98        <- junk left from previous call to cat
 9960:   509    78        <- junk left from previous call to cat
 9959:   509    68        <- junk left from previous call to cat
 9958:     7    100       <- junk left from previous call to cat
 9957:     0    unused
 9956:     0    unused


 IMPORTANT: notice that pending temporaries for the expression
        555 * (666 + dog(777,888))
 are stored at 9972 and 9973!  This is what the toffset is for.

 the temporary at 9966 was generated in the expression
        z = x + y
 in dog where x was 777 and y was 888.



 As you go through the tree generating code you need to
 keep track of toffset.

 1. It is set to the end of the current frame when you enter a
 function.

 2. As expressions push and pop stuff off the temporary stack after the
 current frame the toffset gets incremented when you save a value there
 and decremented when that value is no longer needed.  Remember the
 stack you are managing is a RUNTIME stack not a compile time stack.  You
 are using toffset to predict where the temporaries go!  First study
 how a simple expression is computed and then just use toffset in your
 emit commands as an offset from register 1.  Here is the code for x =
 y + z from my compiler:

  38:     LD  3,-2(1)    Load variable x                   <- LHS code
  39:     ST  3,-5(1)    Save left side in temporary       <- after LHS code
  40:     LD  3,-3(1)    Load variable y                   <- RHS code
  41:     LD  4,-5(1)    Load left into ac1 from temporary <- after RHS code
  42:    ADD  3,4,3      Op +                              <- do op
  43:     ST  3,-4(1)    Store variable z                  <- assignment

 the size of the dog frame was 5.  So -5(1) means the first spot after
```

the frame (because 0(1) is the first spot in the frame).

3. When you enter a function, as in the case of entering dog for a second time, as above, note that the we leave some temps between the main frame and the dog frame. This means new frames are always added as if they are on the temp stack of the previous frame! That is, frames are added after the current temp on the stack. This position is toffset. So you should use toffset while creatiing the ghost frame.

4. By carefully incrementing and decrementing toffset you should find that you only need to set toffset to the position right after the frame you just created when you enter a function (as you are generating the code). Expressions should always end with toffset back at the space right after the current frame. But the important idea is that this will happen. So make sure you always pair increment with decrement.

5. toffset is a compile time variable and changes as you pass through the code lexically.

6. Remember as you generate code (compile time) you are traversing the tree which is essentially reading the program from the beginning to the end (lexical order). When you run the code at run time, then you move through the code in execution order.

==========

TM code for this example

```
* C- compiler version C-F07
* Author: Robert B. Heckendorn
* Backend adapted from work by Jorge Williams (2001)
* File compiled:  z.c-
* Nov 29, 2007
* BEGIN function input
  1:     ST  3,-1(1)    Store return address
  2:     IN  2,2,2      Grab int input
  3:     LD  3,-1(1)    Load return address
  4:     LD  1,0(1)     Adjust fp
  5:    LDA  7,0(3)     Return
* END of function input
* BEGIN function output
  6:     ST  3,-1(1)    Store return address
  7:     LD  3,-2(1)    Load parameter
  8:    OUT  3,3,3      Output integer
  9:    LDC  2,0(6)     Set return to 0
 10:     LD  3,-1(1)    Load return address
 11:     LD  1,0(1)     Adjust fp
 12:    LDA  7,0(3)     Return
* END of function output
* BEGIN function inputb
 13:     ST  3,-1(1)    Store return address
 14:    INB  2,2,2      Grab bool input
 15:     LD  3,-1(1)    Load return address
 16:     LD  1,0(1)     Adjust fp
 17:    LDA  7,0(3)     Return
* END of function inputb
```

```
* BEGIN function outputb
 18:     ST  3,-1(1)    Store return address
 19:     LD  3,-2(1)    Load parameter
 20:   OUTB  3,3,3      Output bool
 21:    LDC  2,0(6)     Set return to 0
 22:     LD  3,-1(1)    Load return address
 23:     LD  1,0(1)     Adjust fp
 24:    LDA  7,0(3)     Return
* END of function outputb
* BEGIN function outnl
 25:     ST  3,-1(1)    Store return address
 26:  OUTNL  3,3,3      Output a newline
 27:     LD  3,-1(1)    Load return address
 28:     LD  1,0(1)     Adjust fp
 29:    LDA  7,0(3)     Return
* END of function outnl
* BEGIN function dog
 30:     ST  3,-1(1)    Store return address.
* BEGIN compound statement
* EXPRESSION STMT
 31:     LD  3,-2(1)    Load variable x
 32:     ST  3,-5(1)    Save left side
 33:     LD  3,-3(1)    Load variable y
 34:     LD  4,-5(1)    Load left into ac1
 35:    ADD  3,4,3      Op +
 36:     ST  3,-4(1)    Store variable z
* RETURN
 37:     LD  3,0(0)     Load variable g
 38:     ST  3,-5(1)    Save left side
 39:     LD  3,-4(1)    Load variable z
 40:     LD  4,-5(1)    Load left into ac1
 41:    MUL  3,4,3      Op *
 42:    LDA  2,0(3)     Copy result to rt register
 43:     LD  3,-1(1)    Load return address
 44:     LD  1,0(1)     Adjust fp
 45:    LDA  7,0(3)     Return
* END compound statement
* Add standard closing in case there is no return statement
 46:    LDC  2,0(6)     Set return value to 0
 47:     LD  3,-1(1)    Load return address
 48:     LD  1,0(1)     Adjust fp
 49:    LDA  7,0(3)     Return
* END of function dog
* BEGIN function cat
 50:     ST  3,-1(1)    Store return address.
* BEGIN compound statement
 51:    LDC  3,10(6)    load size of array z
 52:     ST  3,-4(1)    save size of array z
* EXPRESSION STMT
 53:    LDC  3,8(6)     Load constant
 54:     ST  3,-15(1)   Save index
 55:    LDC  3,9(6)     Load constant
 56:    LDA  4,-5(1)    Load address of base of array z
 57:    SUB  3,4,3      Compute offset of value
 58:     LD  3,0(3)     Load the value
 59:     LD  4,-15(1)   Restore index
 60:    LDA  5,-5(1)    Load address of base of array z
 61:    SUB  5,5,4      Compute offset of value
```

```
 62:     ST  3,0(5)     Store variable z
* EXPRESSION STMT
 63:    LDC  3,8(6)     Load constant
 64:     ST  3,-15(1)   Save index
 65:    LDC  3,9(6)     Load constant
 66:    LDA  4,-2(0)    Load address of base of array h
 67:    SUB  3,4,3      Compute offset of value
 68:     LD  3,0(3)     Load the value
 69:     LD  4,-15(1)   Restore index
 70:    LDA  5,-2(0)    Load address of base of array h
 71:    SUB  5,5,4      Compute offset of value
 72:     ST  3,0(5)     Store variable h
* EXPRESSION STMT
 73:    LDC  3,7(6)     Load constant
 74:     ST  3,-15(1)   Save index
 75:     LD  3,-3(1)    Load variable y
 76:     LD  4,-2(1)    Load address of base of array x
 77:    SUB  3,4,3      Compute offset of value
 78:     LD  3,0(3)     Load the value
 79:     LD  4,-15(1)   Restore index
 80:    LDA  5,-5(1)    Load address of base of array z
 81:    SUB  5,5,4      Compute offset of value
 82:     ST  3,0(5)     Store variable z
* EXPRESSION STMT
 83:    LDC  3,7(6)     Load constant
 84:     ST  3,-15(1)   Save index
 85:     LD  3,-3(1)    Load variable y
 86:    LDA  4,-5(1)    Load address of base of array z
 87:    SUB  3,4,3      Compute offset of value
 88:     LD  3,0(3)     Load the value
 89:     LD  4,-15(1)   Restore index
 90:     LD  5,-2(1)    Load address of base of array x
 91:    SUB  5,5,4      Compute offset of value
 92:     ST  3,0(5)     Store variable x
* END compound statement
* Add standard closing in case there is no return statement
 93:    LDC  2,0(6)     Set return value to 0
 94:     LD  3,-1(1)    Load return address
 95:     LD  1,0(1)     Adjust fp
 96:    LDA  7,0(3)     Return
* END of function cat
* BEGIN function main
 97:     ST  3,-1(1)    Store return address.
* BEGIN compound statement
 98:    LDC  3,10(6)    load size of array a
 99:     ST  3,-2(1)    save size of array a
* EXPRESSION STMT
100:     ST  1,-14(1)   Store old fp in ghost frame
101:     LD  3,-13(1)   Load variable b
102:     ST  3,-16(1)   Store parameter
103:    LDC  3,999(6)   Load constant
104:     ST  3,-17(1)   Store parameter
105:    LDA  1,-14(1)   Load address of new frame
106:    LDA  3,1(7)     Return address in ac
107:    LDA  7,-78(7)   call dog
108:    LDA  3,0(2)     Save the result in ac
* EXPRESSION STMT
109:     ST  1,-14(1)   Store old fp in ghost frame
```

```
 110:    LDA  3,-3(1)    Load address of base of array a
 111:     ST  3,-16(1)   Store parameter
 112:    LDC  3,6(6)     Load constant
 113:     ST  3,-17(1)   Store parameter
 114:    LDA  1,-14(1)   Load address of new frame
 115:    LDA  3,1(7)     Return address in ac
 116:    LDA  7,-67(7)   call cat
 117:    LDA  3,0(2)     Save the result in ac
 * EXPRESSION STMT
 118:     ST  1,-14(1)   Store old fp in ghost frame
 119:    LDC  3,777(6)   Load constant
 120:     ST  3,-16(1)   Store parameter
 121:    LDC  3,888(6)   Load constant
 122:     ST  3,-17(1)   Store parameter
 123:    LDA  1,-14(1)   Load address of new frame
 124:    LDA  3,1(7)     Return address in ac
 125:    LDA  7,-96(7)   call dog
 126:    LDA  3,0(2)     Save the result in ac
 127:     ST  3,-13(1)   Store variable b
 * END compound statement
 * Add standard closing in case there is no return statement
 128:    LDC  2,0(6)     Set return value to 0
 129:     LD  3,-1(1)    Load return address
 130:     LD  1,0(1)     Adjust fp
 131:    LDA  7,0(3)     Return
 * END of function main
   0:    LDA  7,131(7)   Jump to init
 * BEGIN Init
 132:     LD  0,0(0)     Set the global pointer
 * BEGIN init of global array sizes
 133:    LDC  3,10(6)    load size of array h
 134:     ST  3,-1(0)    save size of array h
 * END init of global array sizes
 135:    LDA  1,-12(0)   set first frame at end of globals
 136:     ST  1,0(1)     store old fp (point to self)
 137:    LDA  3,1(7)     Return address in ac
 138:    LDA  7,-42(7)   Jump to main
 139:   HALT  0,0,0      DONE!
 * END Init
```