CS4100/5100 COMPILER DESIGN PROJECT
Semantics and Code Generation in Part 4
Spring 2022

The final phase of the compiler project is the implementation of code generation. To accomplish this, the syntax analyzer is modified so that, while it is parsing the grammatical structure, it is also creating the correct Quad codes which will carry out the intended actions of the source code. There are a few guiding principles which will make this process straight-forward:

1) Every Quad will be generated sequentially from top to bottom of the Quad Table, with no gaps, so that there will be no complicated moving or rearranging of Quad rows.

2) Quads are added to the table as soon as the operands are passed back from the recursive calls, so no additional record-keeping is needed.

3) The only values returned from the recursive calls are symbol table integer indices.

4) Only a few patterns of Quad generation are needed, and they can be reused in multiple places in the CFG.

5) Because of the CFG structure and the recursive nature of the non-terminal functions, the mathematical precedence is taken care of automatically.

6) Only a few address targets will be back-filled later for jump instructions, as when implementing selection and iteration statements.

Example 1: Simple Assignment Statement

A simple example comes from the assignment of one variable into another:

```
a := b;
```

The goal here is to generate a Quad which does the MOVE instruction. Assuming that variable ‘a’ is located in Symbol Table slot 1 and ‘b’ is in slot 2, the resulting Quad (reference the ‘Foundations’ handout) should be:

```
MOV, 2, 0, 1
```

The pseudocode for the Assignment Statement (derived from the CFG) would look like:

```
int statement;
{ int left, right;
  if (tokenCode==IDENT_) //ident, 50, is present; call <variable>
     {    left = variable; //returns var index
          if (tokenCode==ASGN_) //assignment op
`              {
                GNT; //move ahead
                right = simpleexpression; //get result index
                Quads.AddQuad(MovCode,right,0,left);
               }
          else
               errorexpect(tokenCode,ASGN_); //display error
     }
  else
     ... //do other statement possibilities checks

}
```

The objective of the *variable* non-terminal is to return the index of that variable, so it would look like:

```
     int variable;
     { int result= -1;
       result = identifier; //NOTE: does not move past the token, so
                            // that it can be inspected here!
       if (tokenCode==IDENT_) //Do type check as needed- not label,
                              // and not program name, string?
          {dotypecheck;
           result = Location; //index of the var from last GNT call
           GNT; //move ahead
          }
       return(result);
     }
```

Simpleexpression needs to return the symbol table index of the result of the sub-expressions which accumulated below it. GenSymbol adds a specially-named temp variable to the symbol table and returns its index. *Minus1Index and Plus1Index are preloaded for the locations of symbols -1 and 1 in S.T. at program initialization... they provide the implied FOR loop increment and decrement values later.*

```
int simpleexpression;
     {int left, right, signval, temp, opcode;
      signval = sign; //returns -1 if neg, otherwise 1
      left = term;
      if (signval == -1) //Add a negation quad
        Quads.addQuad(MULT,left,Minus1Index,left);
      //This is adding terms together as they are found, adding Quads
      while (plusorminus(tokenCode) //is + or -
          { if (tokenCode==PLUS) opcode = ADDOP;
            else opcode = SUBOP;
            GNT; //move ahead
            right = term; // index of term result
            temp = GenSymbol; //index of new temp variable
            Quads.addQuad(opcode,left,right,temp);
            left = temp; //new leftmost term is last result
          }
      return(left);
     }

int sign;
     {int result=1; //only move ahead if + or - found; optional sign
      if (tokenCode==MINUS)
          {result = -1;
           GNT;
          }
      else if (tokenCode==PLUS)
           GNT;
      return(result);
     }
```