Recitation 6: Code Generation

COP3402 FALL 2015 – ARYA POURTABATABAIE

FROM EURIPIDES MONTAGNE, FALL 2014

Code Generation

The code generation takes the parse tree returned by the parser and creates machine code from it.

Since the parse tree is implicit in the recursion stack of our recursive descending parser, we will *interleave* the code generation into the parsing process.

Our parser uses:

TOKEN –a global variable that stores the current token to analyze.

GET_TOKEN() – a procedure that takes the next token in the string and stores it in TOKEN.

ENTER(*type, name, params*) – a procedure that stores a new symbol into the Symbol Table.

ERROR() – a procedure that stops parsing, and shows an error message.

Additional procedures

gen(int, int, int) – Inserts a new instruction into the code list.

find(ident) – Returns the position of a symbol in the Symbol Table, or 0 if not found.

symboltype(int) – Returns the type of a symbol (constant, variable or procedure).

symbollevel(int) – Returns the level of a symbol.

symboladdress(int) – Returns the address of a symbol.

PL/0 Grammar

```
<program> ::= <block> .
    <block>                        ::= <const-decl> <var-decl> <proc-decl> <statement>
    <const-decl>                   ::= const <const-assignment-list> ; | e
    <const-assignment-list>        ::= <ident> = <number>
                                   | <const-assignment-list> , <ident> = <number>
    <var-decl>                     ::= var <ident-list> ; | e
    <ident-list>                   ::= <ident> | <ident-list> , <ident>
    <proc-decl>                    ::= <proc-decl> procedure <ident> ; <block> ; | e
    <statement>                    ::= <ident> := <expression>       | call <ident>
                                   | begin <statement-list> end      | if <condition> then <statement>
                                   | while <condition> do <statement>    | e
    <statement-list>               ::= <statement> | <statement-list> ; <statement>
    <condition>                    ::= odd <expression> | <expression> <relation> <expression>
    <relation>                     ::= = | <> | < | > | <= | >=
    <expression>                   ::= <term> | <adding-operator> <term>
                                   | <expression> <adding-operator> <term>
    <adding-operator>              ::= + | -
    <term>                         ::= <factor> | <term> <multiplying-operator> <factor>
    <multiplying-operator>         ::= * | /
    <factor>                       ::= <ident> | <number> | ( <expression> )
```

- Again with the slightly variant grammar
- And again, it shouldn’t be hard at all for you to use the lessons from this lab anyway

PL/0 Code Generation

<statement> ::= <ident> := <expression>

For this example, we’ll only focus on the code generation for the assignment statement.

- x := a;
- x := y + b;

<statement> Procedure

<statement> ::= <ident> := <expression>

```
procedure STATEMENT;
begin
 if TOKEN = IDENT then begin
          GET_TOKEN();
          If TOKEN <> ":=" then ERROR (:= missing in statement);
          GET_TOKEN();
          EXPRESSION();
 end
…
```

- We start with the parsing function for statement, and add code generation on it.

<statement> Procedure

<statement> ::= <ident> := <expression>

- First, let’s check that we have a valid variable.

```
 procedure STATEMENT;
begin
 if TOKEN = IDENT then begin
         i = find(ident);
         if i == 0 then ERROR (“Undeclared identifier”);
         if symboltype(i) != variable then ERROR
                  (“Assignment to constant or procedure is not allowed”);
         GET_TOKEN();
         if TOKEN <> ":=" then ERROR (:= missing in statement);
         GET_TOKEN();
         EXPRESSION();
 end
…
```

<statement> Procedure

<statement> ::= <ident> := <expression>

- Now, create some code.

```
 procedure STATEMENT;
begin
  if TOKEN = IDENT then begin
            i = find(ident);
            if i == 0 then ERROR ("Undeclared identifier");
            if symboltype(i) <> variable then ERROR
                        ("Assignment to constant or procedure is not allowed");
            GET_TOKEN();
            if TOKEN <> ":=" then ERROR (:= missing in statement);
            GET_TOKEN();
            EXPRESSION();
            gen(STO, symbollevel(i), symboladdress(i));
  end
…
```

STO = 4 in our project.

That’s it?

- In this case, yes. The assignment statement have to generate the code to do **only** the actual assignment.
- The generated code must store the result of “expression” into the correct variable.
- The code to do whatever is in “expression” (be it another variable, or some calculation) must be created by the <expression> function, not by the <statement> function.

Simple example

:= a;

TOKEN= x	*Symbol Table*

i =	a(t=v, l=1, a=1);
	x(t=v, l=2, a=4);

Recursion stack

…
statement();

```
procedure STATEMENT;
begin
   if TOKEN = IDENT then begin
→   i = find(ident);
    if i == 0 then ERROR ();
    if symboltype(i) <> variable then ERROR ();
    GET_TOKEN();
    if TOKEN <> ":=" then ERROR ();
    GET_TOKEN();
    EXPRESSION();
    gen(STO, symbollevel(i), symboladdress(i));
   end
…
```

Code list

…

Simple example

:= a;

TOKEN= x

i = 2

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();

```
 procedure STATEMENT;
begin
      if TOKEN = IDENT then begin
        i = find(ident);
        if i == 0 then ERROR ();
        if symboltype(i) <> variable then ERROR ();
        GET_TOKEN();
        if TOKEN <> ":=" then ERROR ();
        GET_TOKEN();
        EXPRESSION();
        gen(STO, symbollevel(i), symboladdress(i));
      end
…
```

Code list

…

Simple example

:= a;

TOKEN= x

i = 2

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();

```
procedure STATEMENT;
begin
   if TOKEN = IDENT then begin
      i = find(ident);
      if i == 0 then ERROR ();
 →    if symboltype(i) <> variable then ERROR ();
      GET_TOKEN();
      if TOKEN <> ":=" then ERROR ();
      GET_TOKEN();
      EXPRESSION();
      gen(STO, symbollevel(i), symboladdress(i));
   end
…
```

Code list

…

Simple example

:= a;

TOKEN= x
i = 2

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();

```
 procedure STATEMENT;
begin
      if TOKEN = IDENT then begin
        i = find(ident);
        if i == 0 then ERROR ();
        if symboltype(i) <> variable then ERROR ();
  →     GET_TOKEN();
        if TOKEN <> ":=" then ERROR ();
        GET_TOKEN();
        EXPRESSION();
        gen(STO, symbollevel(i), symboladdress(i));
      end
…
```

Code list

…

Simple example

a;

TOKEN= :=	*Symbol Table*

i = 2	a(t=v, l=1, a=1);
	x(t=v, l=2, a=4);

Recursion stack

…
statement();

```
procedure STATEMENT;
begin
    if TOKEN = IDENT then begin
      i = find(ident);
      if i == 0 then ERROR ();
      if symboltype(i) <> variable then ERROR ();
      GET_TOKEN();
  → if TOKEN <> ":=" then ERROR ();
      GET_TOKEN();
      EXPRESSION();
      gen(STO, symbollevel(i), symboladdress(i));
    end
…
```

Code list

…

Simple example

a;

TOKEN= :=
i = 2

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();

```
 procedure STATEMENT;
begin
      if TOKEN = IDENT then begin
        i = find(ident);
        if i == 0 then ERROR ();
        if symboltype(i) <> variable then ERROR ();
        GET_TOKEN();
        if TOKEN <> ":=" then ERROR ();
   ---> GET_TOKEN();
        EXPRESSION();
        gen(STO, symbollevel(i), symboladdress(i));
      end
…
```

Code list

…

Simple example

;

TOKEN= a	*Symbol Table*

i = 2	a(t=v, l=1, a=1);
	x(t=v, l=2, a=4);

Recursion stack

…
statement();

```
 procedure STATEMENT;
begin
    if TOKEN = IDENT then begin
      i = find(ident);
      if i == 0 then ERROR ();
      if symboltype(i) <> variable then ERROR ();
      GET_TOKEN();
      if TOKEN <> ":=" then ERROR ();
      GET_TOKEN();
→     EXPRESSION();
      gen(STO, symbollevel(i), symboladdress(i));
    end
…
```

Code list

…

Simple example

;

TOKEN= a	*Symbol Table*
	x(t=v, l=2,a=4);

Recursion stack

…
statement();
expression();

```
procedure EXPRESSION;
begin
      if TOKEN = ADDING_OPERATOR then GET_TOKEN();
      TERM();
      while TOKEN = ADDING_OPERATOR do begin
        GET_TOKEN();
        TERM();
      end
end;
```

Here we should have code to handle the code generation if we find an adding operator. It is not shown in this example.

Simple example

;

TOKEN= a	*Symbol Table*
	x(t=v, l=2,a=4);

Recursion stack

…
statement();
expression();

```
procedure EXPRESSION;
begin
     if TOKEN = ADDING_OPERATOR then GET_TOKEN();
───→ TERM();
     while TOKEN = ADDING_OPERATOR do begin
       GET_TOKEN();
       TERM();
     end
end;
```

Code list

…

Simple example

;

TOKEN= a

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();
term();

```
procedure TERM;
begin
   FACTOR();
   while TOKEN = MULTIPLYING_OPERATOR do begin
     GET_TOKEN();
     FACTOR();
   end
end;
```

Code list

…

Simple example

;

TOKEN= a	*Symbol Table*

	a(t=v, l=1, a=1);
	x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();
term();
factor();

```
procedure FACTOR;
begin
   if TOKEN = IDENTIFIER then
     GET_TOKEN();
   else if TOKEN = NUMBER then
     GET_TOKEN();
   else if TOKEN = "(" then begin
     GET_TOKEN();
     EXPRESSION();
     if TOKEN <> ")" then ERROR );
     GET_TOKEN();
   end
   else ERROR ();
end;
```

We’ll add code here to generate code for our case.

Simple example

;

TOKEN= a

i =

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();
term();
factor();

```
procedure FACTOR;
begin
   if TOKEN = IDENTIFIER then begin
    i = find(ident);
    if i == 0 then ERROR();
    if symboltype(i) == variable then gen(LOD, symbollevel(i), symboladdress(i));
    else if symboltype(i) == constant then gen(LIT, 0, symbolval(i));
    else ERROR();
    GET_TOKEN();
   end;
   else if TOKEN = NUMBER then
    GET_TOKEN();
   else if TOKEN = "(" then begin
    […]
```

Code list

…

Simple example

;

TOKEN= a
i = 1

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();
term();
factor();

```
procedure FACTOR;
begin
   if TOKEN = IDENTIFIER then begin
     i = find(ident);
→    if i == 0 then ERROR();
     if symboltype(i) == variable then gen(LOD, symbollevel(i), symboladdress(i));
     else if symboltype(i) == constant then gen(LIT, 0, symbolval(i));
     else ERROR();
     GET_TOKEN();
   end;
   else if TOKEN = NUMBER then
     GET_TOKEN();
   else if TOKEN = "(" then begin
     […]
```

Code list

…

Simple example

;

TOKEN= a	*Symbol Table*

i = 1	a(t=v, l=1, a=1);
	x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();
term();
factor();

```
procedure FACTOR;
begin
   if TOKEN = IDENTIFIER then begin
    i = find(ident);
    if i == 0 then ERROR();
    if symboltype(i) == variable then gen(LOD, symbollevel(i), symboladdress(i));
    else if symboltype(i) == constant then gen(LIT, 0, symbolval(i));
    else ERROR();
    GET_TOKEN();
   end;
   else if TOKEN = NUMBER then
    GET_TOKEN();
   else if TOKEN = "(" then begin
    […]
```

Code list

…

Simple example

;

TOKEN= a	*Symbol Table*

i = 1	a(t=v, l=1, a=1);
	x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();
term();
factor();

```
procedure FACTOR;
begin
      if TOKEN = IDENTIFIER then begin
        i = find(ident);
        if i == 0 then ERROR();
        if symboltype(i) == variable then gen(LOD, symbollevel(i), symboladdress(i));
        else if symboltype(i) == constant then gen(LIT, 0, symbolval(i));
        else ERROR();
  →     GET_TOKEN();
      end;
      else if TOKEN = NUMBER then
        GET_TOKEN();
      else if TOKEN = "(" then begin
        […]
```

Code list

…
3 1 1

Simple example

TOKEN= ;
i = 1

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();
term();
factor();

```
procedure FACTOR;
begin
    if TOKEN = IDENTIFIER then begin
      i = find(ident);
      if i == 0 then ERROR();
      if symboltype(i) == variable then gen(LOD, symbollevel(i), symboladdress(i));
      else if symboltype(i) == constant then gen(LIT, 0, symbolval(i));
      else ERROR();
      GET_TOKEN();
  → end;
    else if TOKEN = NUMBER then
      GET_TOKEN();
    else if TOKEN = "(" then begin
      […]
```

Code list

…
3 1 1

Simple example

TOKEN= ;

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();
term();

```
procedure TERM;
begin
   FACTOR();
   while TOKEN = MULTIPLYING_OPERATOR do begin
    GET_TOKEN();
    FACTOR();
   end
end;
```

Code list

…
3 1 1

Simple example

TOKEN= ;

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();
term();

```
procedure TERM;
begin
      FACTOR();
      while TOKEN = MULTIPLYING_OPERATOR do begin
        GET_TOKEN();
        FACTOR();
      end
end;
```

Code list

…
3 1 1

Simple example

TOKEN= ;

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();

```
procedure EXPRESSION;
begin
      if TOKEN = ADDING_OPERATOR then GET_TOKEN();
      TERM();
→     while TOKEN = ADDING_OPERATOR do begin
        GET_TOKEN();
        TERM();
      end
end;
```

Code list

…
3 1 1

Simple example

TOKEN= ;

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();
expression();

```
procedure EXPRESSION;
begin
      if TOKEN = ADDING_OPERATOR then GET_TOKEN();
      TERM();
      while TOKEN = ADDING_OPERATOR do begin
        GET_TOKEN();
        TERM();
      end
end;
```

Code list

…
3 1 1

Simple example

TOKEN= ;	*Symbol Table*

i = 2	a(t=v, l=1, a=1);
	x(t=v, l=2, a=4);

Recursion stack

…
statement();

```
 procedure STATEMENT;
begin
      if TOKEN = IDENT then begin
        i = find(ident);
        if i == 0 then ERROR ();
        if symboltype(i) <> variable then ERROR ();
        GET_TOKEN();
        if TOKEN <> ":=" then ERROR ();
        GET_TOKEN();
        EXPRESSION();
 →      gen(STO, symbollevel(i), symboladdress(i));
      end
…
```

Code list

…
3 1 1

Simple example

TOKEN= ;

i = 2

Symbol Table

a(t=v, l=1, a=1);
x(t=v, l=2, a=4);

Recursion stack

…
statement();

```
 procedure STATEMENT;
begin
      if TOKEN = IDENT then begin
        i = find(ident);
        if i == 0 then ERROR ();
        if symboltype(i) <> variable then ERROR ();
        GET_TOKEN();
        if TOKEN <> ":=" then ERROR ();
        GET_TOKEN();
        EXPRESSION();
        gen(STO, symbollevel(i), symboladdress(i));
→     end
…
```

Code list

…
3 1 1
4 2 4

Questions?