Recitation 10: Extra Credit Overview

COP3402 FALL 2015 – ARYA POURTABATABAIE

FROM EURIPIDES MONTAGNE, FALL 2014

PL/0 Program

```
var f, n;
procedure fact;
 var ans1;
 begin
      ans1:=n;
      n:= n-1;
      if n = 0 then f := 1;
      if n > 0 then call fact;
      f:=f*ans1;
 end;
begin
 n:=3;
 call fact;
 write f;
end.
```

Global variables used to send and receive data from procedure.

Local variables needed to keep temporal calculations and values.

Extended PL/0 Program

```
var num;
procedure factorial(x);
 begin
       if x = 0 then return := 1
       else return := x * call factorial(x-1);
 end;
begin
 num := call factorial(3);
 write num;
end.
```

Parameter Passing

Functional Value return

Parameter Passing

Functional Value return

Functional Value Return

Decouples the output of a procedure.

Uses the Functional Value (FV) slot in the activation record to return a value.

When parsing the procedure declaration, we insert a variable called “return” with level+1 and address=0.

FV is accessed through “return”.

- return := 5;

Functional Value Return

```
var x,y;
procedure foo;
 var w;
 begin
         w := x;
         w := w*w*w;
         return := w;
 end;
begin
 x := 5;
 y := 25 + call foo / 10;
 write y;
end.
```

```
...
05 LOD 0 4
06 LOD 0 4
07 OPR 0 MUL
08 LOD 0 4
09 OPR 0 MUL
10 STO 0 4
11 LOD 0 4
12 STO 0 0
13 OPR 0 RET
...
```

Functional Value Return

```
var x,y;
procedure foo;
 var w;
 begin
         w := x;
         w := w*w*w;
         return := w;
 end;
begin
 x := 5;
 y := 25 + call foo / 10;
 write y;
end.
```

```
  ...
  05 LOD 0 4
  06 LOD 0 4
  07 OPR 0 MUL
  08 LOD 0 4
➡ 09 OPR 0 MUL
  10 STO 0 4
  11 LOD 0 4
  12 STO 0 0
  13 OPR 0 RET
  ...
```

stack

0	FV
1	SL
2	DL
3	RA
4	x
5	y
6	25
7	FV
8	SL
9	DL
10	RA
11	w
12	125

Functional Value Return

```
var x,y;
procedure foo;
 var w;
 begin
        w := x;
        w := w*w*w;
        return := w;
 end;
begin
 x := 5;
 y := 25 + call foo / 10;
 write y;
end.
```

```
...
05 LOD 0 4
06 LOD 0 4
07 OPR 0 MUL
08 LOD 0 4
09 OPR 0 MUL
10 STO 0 4
11 LOD 0 4
12 STO 0 0
13 OPR 0 RET
...
```

stack

0	FV
1	SL
2	DL
3	RA
4	x
5	y
6	25
7	FV
8	SL
9	DL
10	RA
11	w=125

Functional Value Return

```
var x,y;
procedure foo;
 var w;
 begin
         w := x;
         w := w*w*w;
         return := w;
 end;
begin
 x := 5;
 y := 25 + call foo / 10;
 write y;
end.
```

```
 ...
 05 LOD 0 4
 06 LOD 0 4
 07 OPR 0 MUL
 08 LOD 0 4
 09 OPR 0 MUL
 10 STO 0 4
➡11 LOD 0 4
 12 STO 0 0
 13 OPR 0 RET
 ...
```

stack

0	FV
1	SL
2	DL
3	RA
4	x
5	y
6	25
7	FV
8	SL
9	DL
10	RA
11	w=125
12	125

Functional Value Return

```
var x,y;
procedure foo;
 var w;
 begin
        w := x;
        w := w*w*w;
        return := w;
 end;
begin
 x := 5;
 y := 25 + call foo / 10;
 write y;
end.
```

```
...
05 LOD 0 4
06 LOD 0 4
07 OPR 0 MUL
08 LOD 0 4
09 OPR 0 MUL
10 STO 0 4
11 LOD 0 4
➡ 12 STO 0 0
13 OPR 0 RET
...
```

stack

0	FV
1	SL
2	DL
3	RA
4	x
5	y
6	25
7	FV=125
8	SL
9	DL
10	RA
11	w=125

Functional Value Return

```
var x,y;
procedure foo;
 var w;
 begin
        w := x;
        w := w*w*w;
        return := w;
 end;
begin
 x := 5;
 y := 25 + call foo / 10;
 write y;
end.
```

```
...
05 LOD 0 4
06 LOD 0 4
07 OPR 0 MUL
08 LOD 0 4
09 OPR 0 MUL
10 STO 0 4
11 LOD 0 4
12 STO 0 0
13 OPR 0 RET
...
```

stack

0	FV
1	SL
2	DL
3	RA
4	x
5	y
6	25

Functional Value Return

The FV is still in the stack, but we must recover it before using it.

Two types of calls:

- Do not use FV (discard it).
- Use FV (recover it).

Recover FV

This happens when we call the function inside an expression.

The FV is one position over the SP.

Just increment the SP by 1 (INC 0 1).

Functional Value Return

```
var x,y;
procedure foo;
 var w;
 begin
         w := x;
         w := w*w*w;
         return := w;
 end;
begin
 x := 5;
 y := 25 + call foo / 10;
 write y;
end.
```

```
   ...
   12 STO 0 0
➡ 13 OPR 0 RET
   ...
   16 LIT 0 25
   17 CAL 0 1
   18 INC 0 1
   19 OPR 0 SUM
   20 LIT 0 10
   21 OPR 0 DIV
   22 STO 0 5
   23 LOD 0 5
   24 SIO 0 1
   25 OPR 0 RET
```

stack

0	FV
1	SL
2	DL
3	RA
4	x
5	y
6	25

Functional Value Return

```
var x,y;
procedure foo;
 var w;
 begin
         w := x;
         w := w*w*w;
         return := w;
 end;
begin
 x := 5;
 y := 25 + call foo / 10;
 write y;
end.
```

```
...
12 STO 0 0
13 OPR 0 RET
...
16 LIT 0 25
17 CAL 0 1
18 INC 0 1
19 OPR 0 SUM
20 LIT 0 10
21 OPR 0 DIV
22 STO 0 5
23 LOD 0 5
24 SIO 0 1
25 OPR 0 RET
```

stack

0	FV
1	SL
2	DL
3	RA
4	x
5	y
6	25

Functional Value Return

```
var x,y;
procedure foo;
 var w;
 begin
         w := x;
         w := w*w*w;
         return := w;
 end;
begin
 x := 5;
 y := 25 + call foo / 10;
 write y;
end.
```

```
...
12 STO 0 0
13 OPR 0 RET
...
16 LIT 0 25
17 CAL 0 1
→ 18 INC 0 1
19 OPR 0 SUM
20 LIT 0 10
21 OPR 0 DIV
22 STO 0 5
23 LOD 0 5
24 SIO 0 1
25 OPR 0 RET
```

stack

0	FV
1	SL
2	DL
3	RA
4	x
5	y
6	25
7	FV=125

Functional Value Return

```
var x,y;
procedure foo;
 var w;
 begin
         w := x;
         w := w*w*w;
         return := w;
 end;
begin
 x := 5;
 y := 25 + call foo / 10;
 write y;
end.
```

```
  ...
  12 STO 0 0
  13 OPR 0 RET
  ...
  16 LIT 0 25
  17 CAL 0 1
  18 INC 0 1
➡ 19 OPR 0 SUM
  20 LIT 0 10
  21 OPR 0 DIV
  22 STO 0 5
  23 LOD 0 5
  24 SIO 0 1
  25 OPR 0 RET
```

stack

0	FV
1	SL
2	DL
3	RA
4	x
5	y
6	150

Recover FV

To use function calls inside expressions, we must treat them as factors.

factor ::= ident | number | "**(**" expression "**)**" | "**call**" ident**.**

Differentiate call types

When used as factor, we must generate INC.

factor ::= ident | number | "**(**" expression "**)**" | "**call**" ident**.**

When used as statement, we do nothing else.

statement ::= […

| "**call**" ident parameter-list

…

Parameter Passing

Decouples the input of a procedure.

We treat parameter exactly as variables, with one distinction:

- The value of a parameter is set before we call the procedure.

We must add parameter slots in the AR, before the variable slots.

0	FV
1	SL
2	DL
3	RA
4	params
X	…
X	vars
X	…

How Parameter Passing works?

Before the call, we must set the value of the parameters.

Each parameter is an expression, so first solve all expressions.

We must predict the position of parameters.

Copy expressions results to parameter slots.

Then make the call.

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8   INC 0 6
9   LIT 0 5
10  STO 0 4
11  LOD 0 4
12  LIT 0 2
13  LOD 0 4
14  OPR 0 4
15  STO 0 11
16  STO 0 10
17  CAL 0 1
18  INC 0 1
19  STO 0 5
```

stack

0	FV
1	SL
2	DL
3	RA
4	x
5	y

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
        return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
  ...
  8  INC 0 6
→ 9  LIT 0 5
  10 STO 0 4
  11 LOD 0 4
  12 LIT 0 2
  13 LOD 0 4
  14 OPR 0 4
  15 STO 0 11
  16 STO 0 10
  17 CAL 0 1
  18 INC 0 1
  19 STO 0 5
```

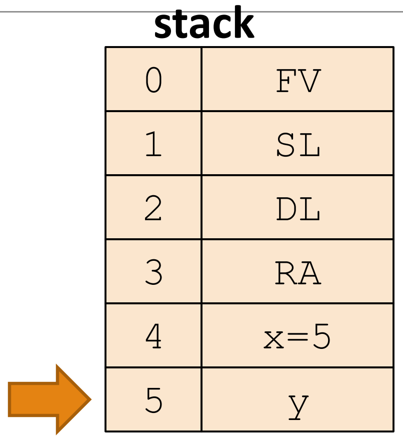
stack

	0	FV
	1	SL
	2	DL
	3	RA
	4	x
	5	y
→	6	5

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
        return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8  INC 0 6
9  LIT 0 5
10 STO 0 4
11 LOD 0 4
12 LIT 0 2
13 LOD 0 4
14 OPR 0 4
15 STO 0 11
16 STO 0 10
17 CAL 0 1
18 INC 0 1
19 STO 0 5
```



stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
     return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
 ...
 8  INC 0 6
 9  LIT 0 5
 10 STO 0 4
➡11 LOD 0 4
 12 LIT 0 2
 13 LOD 0 4
 14 OPR 0 4
 15 STO 0 11
 16 STO 0 10
 17 CAL 0 1
 18 INC 0 1
 19 STO 0 5
```

stack

	0	FV
	1	SL
	2	DL
	3	RA
	4	x=5
	5	y
➡	6	5

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
      return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
   ...
   8  INC 0 6
   9  LIT 0 5
   10 STO 0 4
   11 LOD 0 4
➡ 12 LIT 0 2
   13 LOD 0 4
   14 OPR 0 4
   15 STO 0 11
   16 STO 0 10
   17 CAL 0 1
   18 INC 0 1
   19 STO 0 5
```

stack

	0	FV
	1	SL
	2	DL
	3	RA
	4	x=5
	5	y
	6	5
➡	7	2

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8  INC 0 6
9  LIT 0 5
10 STO 0 4
11 LOD 0 4
12 LIT 0 2
13 LOD 0 4
14 OPR 0 4
15 STO 0 11
16 STO 0 10
17 CAL 0 1
18 INC 0 1
19 STO 0 5
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	5
7	2
8	5

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
        return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8  INC 0 6
9  LIT 0 5
10 STO 0 4
11 LOD 0 4
12 LIT 0 2
13 LOD 0 4
14 OPR 0 4
15 STO 0 11
16 STO 0 10
17 CAL 0 1
18 INC 0 1
19 STO 0 5
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	5
7	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
        return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8  INC 0 6
9  LIT 0 5
10 STO 0 4
11 LOD 0 4
12 LIT 0 2
13 LOD 0 4
14 OPR 0 4
15 STO 0 11
16 STO 0 10
17 CAL 0 1
18 INC 0 1
19 STO 0 5
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	5
7	10
8	--
9	--
10	--
11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
        return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8  INC 0 6
9  LIT 0 5
10 STO 0 4
11 LOD 0 4
12 LIT 0 2
13 LOD 0 4
14 OPR 0 4
15 STO 0 11
16 STO 0 10
17 CAL 0 1
18 INC 0 1
19 STO 0 5
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	5
7	10
8	--
9	--
10	5
11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
        return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8  INC 0 6
9  LIT 0 5
10 STO 0 4
11 LOD 0 4
12 LIT 0 2
13 LOD 0 4
14 OPR 0 4
15 STO 0 11
16 STO 0 10
17 CAL 0 1
18 INC 0 1
19 STO 0 5
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	FV
7	SL
8	DL
9	RA
10	5
11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
1 JMP 0 2
2 INC 0 6
3 LOD 0 4
4 LOD 0 5
5 OPR 0 2
6 STO 0 0
7 OPR 0 0
...
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	FV
7	SL
8	DL
9	RA
10	a=5
11	b=10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
  ...
  1 JMP 0 2
→ 2 INC 0 6
  3 LOD 0 4
  4 LOD 0 5
  5 OPR 0 2
  6 STO 0 0
  7 OPR 0 0
  ...
```

stack

	0	FV
	1	SL
	2	DL
	3	RA
	4	x=5
	5	y
	6	FV
	7	SL
	8	DL
	9	RA
	10	a=5
→	11	b=10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
1 JMP 0 2
2 INC 0 6
3 LOD 0 4
4 LOD 0 5
5 OPR 0 2
6 STO 0 0
7 OPR 0 0
...
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	FV
7	SL
8	DL
9	RA
10	a=5
11	b=10
10	5

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
       return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
1 JMP 0 2
2 INC 0 6
3 LOD 0 4
4 LOD 0 5
5 OPR 0 2
6 STO 0 0
7 OPR 0 0
...
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	FV
7	SL
8	DL
9	RA
10	a=5
11	b=10
10	5
11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
  ...
  1 JMP 0 2
  2 INC 0 6
  3 LOD 0 4
  4 LOD 0 5
→ 5 OPR 0 2
  6 STO 0 0
  7 OPR 0 0
  ...
```

stack

	0	FV
	1	SL
	2	DL
	3	RA
	4	x=5
	5	y
	6	FV
	7	SL
	8	DL
	9	RA
	10	a=5
	11	b=10
→	10	15
	11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
1 JMP 0 2
2 INC 0 6
3 LOD 0 4
4 LOD 0 5
5 OPR 0 2
6 STO 0 0
7 OPR 0 0
...
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	FV=15
7	SL
8	DL
9	RA
10	a=5
11	b=10
10	15
11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
  ...
  1 JMP 0 2
  2 INC 0 6
  3 LOD 0 4
  4 LOD 0 5
  5 OPR 0 2
  6 STO 0 0
➡ 7 OPR 0 0
  ...
```

stack

	0	FV
	1	SL
	2	DL
	3	RA
	4	x=5
➡	5	y
	6	FV=15
	7	SL
	8	DL
	9	RA
	10	a=5
	11	b=10
	10	15
	11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
        return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8   INC 0 6
9   LIT 0 5
10  STO 0 4
11  LOD 0 4
12  LIT 0 2
13  LOD 0 4
14  OPR 0 4
15  STO 0 11
16  STO 0 10
17  CAL 0 1
18  INC 0 1
19  STO 0 5
20  LOD 0 5
21  SIO 0 1
22  OPR 0 0
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	FV=15
7	SL
8	DL
9	RA
10	a=5
11	b=10
10	15
11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
        return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8   INC 0 6
9   LIT 0 5
10  STO 0 4
11  LOD 0 4
12  LIT 0 2
13  LOD 0 4
14  OPR 0 4
15  STO 0 11
16  STO 0 10
17  CAL 0 1
18  INC 0 1
19  STO 0 5
20  LOD 0 5
21  SIO 0 1
22  OPR 0 0
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y
6	FV=15
7	SL
8	DL
9	RA
10	a=5
11	b=10
10	15
11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
      return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8  INC 0 6
9  LIT 0 5
10 STO 0 4
11 LOD 0 4
12 LIT 0 2
13 LOD 0 4
14 OPR 0 4
15 STO 0 11
16 STO 0 10
17 CAL 0 1
18 INC 0 1
19 STO 0 5
20 LOD 0 5
21 SIO 0 1
22 OPR 0 0
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y=15
6	FV=15
7	SL
8	DL
9	RA
10	a=5
11	b=10
10	15
11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8  INC 0 6
9  LIT 0 5
10 STO 0 4
11 LOD 0 4
12 LIT 0 2
13 LOD 0 4
14 OPR 0 4
15 STO 0 11
16 STO 0 10
17 CAL 0 1
18 INC 0 1
19 STO 0 5
20 LOD 0 5
21 SIO 0 1
22 OPR 0 0
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y=15
6	15
7	SL
8	DL
9	RA
10	a=5
11	b=10
10	15
11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8  INC 0 6
9  LIT 0 5
10 STO 0 4
11 LOD 0 4
12 LIT 0 2
13 LOD 0 4
14 OPR 0 4
15 STO 0 11
16 STO 0 10
17 CAL 0 1
18 INC 0 1
19 STO 0 5
20 LOD 0 5
21 SIO 0 1
22 OPR 0 0
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y=15
6	15
7	SL
8	DL
9	RA
10	a=5
11	b=10
10	15
11	10

Param Passing Example

```
var x,y;
procedure sum(a,b);
 begin
         return := a+b;
 end;
begin
 x := 5;
 y := call sum(x,2*x);
 write y;
end.
```

```
...
8   INC 0 6
9   LIT 0 5
10  STO 0 4
11  LOD 0 4
12  LIT 0 2
13  LOD 0 4
14  OPR 0 4
15  STO 0 11
16  STO 0 10
17  CAL 0 1
18  INC 0 1
19  STO 0 5
20  LOD 0 5
21  SIO 0 1
22  OPR 0 0
```

stack

0	FV
1	SL
2	DL
3	RA
4	x=5
5	y=15
6	15
7	SL
8	DL
9	RA
10	a=5
11	b=10
10	15
11	10

PL/0 Extended grammar

(Not what you'll actually use, but close)

procedure-declaration ::= { "**procedure**" ident parameter-block "**;**" block "**;**" }

parameter-block ::= "(" [ident { "," ident }] ")".

parameter-list ::= " (" [expression { "," expression }] ")"**.**

statement   ::= […| "**call**" ident parameter-list|…].

factor ::= ident | number | "**(**" expression "**)**" | "**call**" ident parameter-list**.**

Parsing the parameter block

In the procedure declaration, we found the parameter block.

For each ident found, insert a new symbol in the symbol table with level+1 and corresponding address.

The address of the first parameter is 4 (size of basic AR).

Each subsequent parameter gets the next address (5, 6, etc).

parameter-block()

parameter-block ::= "(" [ident { "," ident }] ")".

```
procedure parameter-block();
begin
 addr := 4;
 if(token <> “(”) then ERROR(“Procedure must have parameters”);
 get_token();
 if(token = ident) then begin
          enter(ident, level+1, addr); addr++;
          get_token();
          while (token = “,”) begin
                    get_token();
                    if(token <> ident) then ERROR();
                    enter(ident, level+1, addr); addr++;
                    get_token();
          end;
 end;
 if token <> “)” then ERROR(“Bad procedure declaration”);
 get_token();
end;
```

Address of variables

The address of the first variable is address of last parameter plus 1.

Each subsequent variable gets the next address.

We must somehow pass the address of the last parameter to function var-decl().

Don’t forget to insert variable “return”.

Parsing the parameter list

Each time we have a call, we found the parameter list.

We must generate code for solving each expression.

Then generate code to store results in parameter slots.

parameter-list()

parameter-list ::= " (" [expression { "," expression }] ")".

```
procedure parameter-list();
begin
 params = 0;
 if(token <> “(”) then ERROR(“Missing parameter list at call”);
 get_token();
 if(token <> “)”) then begin
           expression();
           params++;
 end;
 while (token = “,”) begin
           get_token();
           expression();
           param++;
 end;
 while(param > 0){ // Save results into param slots
           gen(STO, 0, stack_size+4-1);
           param--;
 }
 if (token <> “)”) then ERROR(“Bad calling formating.”);
 get_token();
end;
```

What is stack_size?

In order to predict the position of the parameter slots, we need to know the size of the stack at the moment we do the call.

The stack only grows or shrinks for certain instructions.

We detect when those instructions are issued, and increment or decrement the stack size accordingly.

updateStackSize()

```
int gen(int op, int l, int m){
        code[code_size].op = op;
        code[code_size].l = l;
        code[code_size].m = m;
        updateStackSize(op,l,m);
        return code_size++;
}
void updateStackSize(int op, int l, int m){
 if(op == LIT, LOD, READ){
        stack_size++;
 }else if (op == STO, JPC, WRITE){
        stack_size--;
 }else if (op == INC){
        stack_size += m;
 }else if (op == OPR){
        if(m == RET) {stack_size = 0;}
        else if (m != NEG, ODD) {stack_size--;}
 }
}
```

Questions?