A Guide to the C Version of the PL/O Compiler

There are four versions in which enhancements are incremental:

Version 1: Plain translation of the original PL/0 compiler written in Pascal
Version 2: Enhanced with write statement
Version 3: Enhanced with parameters of call-by-value and call-by-address.
Version 4. Enhanced with interface with X-window functions.

The nested structures of blocks in the PL/0 compiler are translated by factoring out inner procedures to the forefront and bringing the local variables to parameters through pointer parameters. The following example explains the method.

Example.

A Pascal program fragment:

```
procedure A;
var y: integer;
  procedure B;
      var x: integer;
      procedure C;
      begin
         x:=y
      end;
    begin
       x:=1;  y:=3;  C;
    end
begin
  y:=2;  B;
end.
```

The C equivalent:

```
C(px, py)
  int *px, *py;
{
    (*px)=(*py);
}
B(py)
  int *py;
{
    x=1;  (*py)=3; C(&x, py);
}
A( ) {
  int y;
  y=2;  B(&y);
}
```

Details of enhancement in version 2.

Line 20. norw=11 ==> norw=12 (number of reserved words changed form 11 to 12)
Line 35. int writesym=31 (31st sym was added)
Line 40. int wrt=9; int stp=10 at line
Line 350-356. Write statement implemented.
Line 445-446. Output at "store" no longer exists.
Line 454. PL0 instruction "wrt" implemented..
Line 468. wsym[12]=writesym; (added)
Line 492. "wrt" print out.

Example. write(x+y); If sym is writesym, jump to the write statement and getsym. If the sym is "(", getsym and go to expression. After getsym, if sym is not ")", error. Otherwise generate "wrt". The machine instruction "wrt" write the top of the stack, and decrease the stack pointer by one.

Some descriptions of Version 3.

The functions fparam and aparam handles formal parameters and actual parameters respectively. Function fparam treats formal parameters like local variables. It supports call-by-value and call-by-address with "var" attached in front. In the latter case, flag is set to 1, and call vardeclaration. A record in the table is enhanced with another field called "indirect". If a formal parameter is of var type, this "indirect" field is set to 1. Thus if a factor is flagged with 1, we need to access the value in indirect mode, which is done by a new PL0 instruction "lid", reading "load indirect". For an assignment statement for a parameter of var type, we need to store in indirect mode.

Function aparam is to generate necessary code to load the contents of the parameters onto the stack. Call-by-value parameters can be loaded in a usual manner, whereas for parameters of var type, their addresses will be loaded onto the stack.

From the above design, we need to prepare three PL0 instructions "lid", "lda", and "sid", reading "load indirect", "load address", and "store indirect".

Example

```
procedure p(a, var b)
begin
  b:=a+1;
end;
begin
  x:=1;
  call p(x,&y);
  write(y);
end.
```

```
 0    jmp  0 8      10    sto   0 3
 1    jmp  0 2      11    lod  0 3
 2    inc   0 5     12    lda  0 4
 3    lod   0 3     13    stp  0 3
 4    lit    0 1    14    stp  0 3
 5    opr  0 2      15    cal  0 2
 6    sid   0 4     16     lod  0 4
 7    opr  0 2      17    wrt  0 0
 8    inc  0 5      18    opr  0 0
 9    lit   0 1

start  PL/0
 2
END PL/0
```

When an actual parameter is of var type, we use the address operator “&”, similar to C. Also before jumping to procedure p, we store parameters by “stp” instructions. When we store parameters, the stack is popped up by the number of parameters.

The snapshots of the stack is given. The leftmost column corresponds to the instruction addresses.

```
8     0 0 0 0 0
9     0 0 0 0 0 1
10    0 0 0 1 0
11    0 0 0 1 0 1
12    0 0 0 1 0 1 5              two parameters are on the top part of the stack .
13    0 0 0 1 0 1
14    0 0 0 1 0
15    0 0 0 1 0
2     0 0 0 1 0 1 1 16 1 5       16 is the return address. 5 is the address of y
3     0 0 0 1 0 1 1 16 1 5 1
4     0 0 0 1 0 1 1 16 1 5 1 1
5     0 0 0 1 0 1 1 16 1 5 2
6     0 0 0 1 2 1 1 16 1 5
7     0 0 0 1 2
16    0 0 0 1 2 2
17    0 0 0 1 2
18
```

Some descriptions of Version 4, designed for graphics manipulations.

This version supports the following three functions.

color(k): This function set the color to k, where k=1,2,3, and 4 corresponds to black, red, green, and blue. 0 corresponds to white.

draw(x, y): Draw a point of the above color at point (x, y).

erase(x, y): Erases a point at (x, y).

As the X-window library is included in the compiler/interpreter, type

```
gcc -lX11 file.c
```

where file.c is your compiler file.

Some implementation notes:

The “;” after a procedure heading is not allowed.
After each line, space is not allowed.

To obtain the source programs, visit Prof. Takaoka’s home page.