CSE 3302 Notes 10: Implementing Subprograms

(Last updated 4/16/14 3:25 PM)

10.1. Semantics of Calls and Returns

Pascal-S (and PL/0) reflect typical implementations (without optimization in a one-pass compiler)

10.2. Implementing "Simple" Subprograms

Historical FORTRAN and COBOL

Simple =

no recursion
static allocation
minimal call-by-value
call-by-reference for anything non-trivial
If program starts, it will have enough memory

Sebesta, figure 10.2 is *not* a stack - one activation record per subprogram

Layout of activation record (a.r., stack frame), register conventions mandated by vendor

Who saves registers? - caller or callee?

10.3. Implementing Subprograms with Stack-Dynamic Local Variables

PL/0

Calling Sequence:

Caller

```
cal l,a:  call procedure a (absolute address) at level l

    cal: begin {generate new block mark}
           s[t + 1] := base(l);        { static link for nested procs }
           s[t + 2] := b;              { dynamic link to caller proc }
           s[t + 3] := p;              { return address }
           b := t + 1;                 { new base of stack frame
           p := a                      { new program counter }
         end;
```

Called

`jmp` over any instructions for any nested procedures

```
int 0,a  :  increment t-register by a  { a includes 3 slots above }

      int: t := t + a;   { allocate stack frame including sl, dl, ra }
```

Return Sequence

Called

```
opr 0,a  :  execute operation a (=0 here)

      opr: case a of {operator}
           0: begin {return}
                 t := b - 1;              { discard stack frame }
                 p := s[t + 3];           { return to caller }
                 b := s[t + 2];           { old base address }
              end;
```

Caller does nothing special

Pascal-S

Interpreter addressing is done using *display* array to avoid cost of `base` function.

Recursion causes no difficulty as long as subprograms are not nested (e.g. C)

10.4. NESTED SUBPROGRAMS

A nested subprogram may potentially reference local variables for subprograms that contain it. Two difficulties (that also interact):

1. Calls among nested subprograms at same level.
2. Recursion

For a given variable (under static/lexical scoping), the *most recent* invocation of each containing subprograms is the one whose activation record is needed (diagram in notes 05.3).

(The outermost scope is level 0. Nested scopes have increasing level numbers.)

There is exactly one activation record *per level* in the (ascending) *static chain*.

PL/0

Assumption - non-local data is rarely accessed, simple solution is sufficient

Referencing is based on:

Compiler computes level *difference* to put into `lod`, `sto`, and `cal` instructions

`cal` instruction sets saved static link (`s[t + 1]`) to point at next level a.r.

Level difference is used with loop to follow static links:

```
function base(l: integer): integer;
  var b1: integer;
  begin
    b1 := b; {find base l levels down}
    while l > 0 do
      begin
        b1 := s[b1];
        l := l - 1
      end;
    base := b1
  end {base};
```

Pascal-S (section 4 of report)

Makes no assumption regarding locality of references

Instructions include *absolute* level and offset

No static chain traversals. . . since number of scope levels is small, each access references the *display*.

Trivial, but expensive to create a new display for each call (just use the `base` loop).

Instead, maintain one display incrementally:

Updating display for a call is simple. Replace display entry for the level of the *caller*.

Updating display for a return is messier. When the caller is at a deeper level than the called subprogram, then the same number of display entries recovered from the static chain. (This assumes that subsequent calls had occured.)

(Caches on modern machines lessen the performance advantage of displays)

10.5. BLOCKS

==========

```
bash-3.2$ cat recurse.pl0 - |plzero.argstack
    0 var aout,result2;
    1       3    4
    1 procedure a(ain);
    2              -1
    2   var bout;
    2         3
    2   procedure b(bin);
    3               -1
    3     var cout;
    3           3
    3     procedure c(cin);
    4                 -1
    4       begin
    5         cout:=cin+8;
    9         result2:=ain+bin+cin+8
   15       end;
    4  int  0    3           c code
    5  lod  0   -1 cin
    6  lit  0    8
    7  opr  0    2
    8  sto  1    3 cout
    9  lod  2   -1 ain
   10  lod  1   -1 bin
   11  opr  0    2
   12  lod  0   -1 cin
   13  opr  0    2
   14  lit  0    8
   15  opr  0    2
   16  sto  3    4 result2
   17  opr  0    0
   18
   18     begin              b code
   19       call c(4);
   22       bout:=bin+cout
   24     end;
   18  int  0    4
   19  lit  0    4
   20  cal  0    4 c
   21  int  0   -1
   22  lod  0   -1 bin
   23  lod  0    3 cout
   24  opr  0    2
   25  sto  1    3 bout
   26  opr  0    0
   27
   27   begin                a code
   28     call b(2);
   31     aout:=ain+bout
   33   end;
   27  int  0    4
   28  lit  0    2
   29  cal  0   18 b
   30  int  0   -1
   31  lod  0   -1 ain
   32  lod  0    3 bout
   33  opr  0    2
   34  sto  1    3 aout
   35  opr  0    0
   36
   36 begin
   37 call a(1);
   40 out:=aout;
   42 out:=result2
   43 end.
   36  int  0    5
   37  lit  0    1
   38  cal  0   27 a
   39  int  0   -1
   40  lod  0    3 aout
   41  wro  0    0
   42  lod  0    4 result2
   43  wro  0    0
   44  opr  0    0
```

```
start pl/0
t=5 b=1 p=37     initial
s[1]=0   s.l.
s[2]=0   d.l.
s[3]=0   ret adr
s[4]=0   aout
s[5]=0   result2
 t=10 b=7 p=28    after call to a
s[1]=0   s.l.
s[2]=0   d.l.
s[3]=0   ret adr
s[4]=0   aout
s[5]=0   result2
s[6]=1   ain
s[7]=1   s.l.
s[8]=1   d.l.
s[9]=39  ret adr
s[10]=0  bout
t=15 b=12 p=19   after call to b
s[1]=0   s.l.
s[2]=0   d.l.
s[3]=0   ret adr
s[4]=0   aout
s[5]=0   result2
s[6]=1   ain
s[7]=1   s.l.
s[8]=1   d.l.
s[9]=39  ret adr
s[10]=0  bout
s[11]=2  bin
s[12]=7  s.l
s[13]=7  d.l.
s[14]=30 ret adr
s[15]=0  cout
t=19 b=17 p=5    after call to c
s[1]=0   s.l.
s[2]=0   d.l.
s[3]=0   ret adr
s[4]=0   aout
s[5]=0   result2
s[6]=1   ain
s[7]=1   s.l.
s[8]=1   d.l.
s[9]=39  ret adr
s[10]=0  bout
s[11]=2  bin
s[12]=7  s.l.
s[13]=7  d.l.
s[14]=30 ret adr
s[15]=0  cout
s[16]=4  cin
s[17]=12 s.l.
s[18]=12 d.l.
s[19]=21 ret adr
! 15
! 15
 end pl/0
```

```
-bash-3.2$ cat rtsExample.pl0 - |
plzero.argstack
    0 procedure iloop(i);
    2                -1
    2   procedure jloop(j);
    3                  -1
    3     procedure kloop(k);
    4                    -1
    4       begin
    5         k:=k+1;
    9         out:=i+j+k;
   15         if k<3 then
   18           call kloop(k);
   22         if k=3 then
   25           call jloop(j)
   29       end;
    4  int  0    3
    5  lod  0   -1 k
    6  lit  0    1
    7  opr  0    2
    8  sto  0   -1 k
    9  lod  2   -1 i
   10  lod  1   -1 j
   11  opr  0    2
   12  lod  0   -1 k
   13  opr  0    2
   14  wro  0    0
   15  lod  0   -1 k
   16  lit  0    3
   17  opr  0   10
   18  jpc  0   22
   19  lod  0   -1 k
   20  cal  1    4 kloop
   21  int  0   -1
   22  lod  0   -1 k
   23  lit  0    3
   24  opr  0    8
   25  jpc  0   29
   26  lod  1   -1 j
   27  cal  2    2 jloop
   28  int  0   -1
   29  opr  0    0
   30
   30     begin   jloop code
   31       j:=j+10;
   35       if j<40 then
   38         call kloop(0);
   42       if j=40 then
   45         call iloop(i)
   49     end;
   30  int  0    3
   31  lod  0   -1 j
   32  lit  0   10
   33  opr  0    2
   34  sto  0   -1 j
   35  lod  0   -1 j
   36  lit  0   40
   37  opr  0   10
   38  jpc  0   42
   39  lit  0    0
   40  cal  0    4 kloop
   41  int  0   -1
   42  lod  0   -1 j
   43  lit  0   40
   44  opr  0    8
   45  jpc  0   49
   46  lod  1   -1 i
   47  cal  2    1 iloop
   48  int  0   -1
   49  opr  0    0
   50
   50   begin     iloop code
   51     i:=i+100;
   55     if i<400 then
   58       call jloop(0)
   62   end;
   50  int  0    3
   51  lod  0   -1 i
   52  lit  0  100
   53  opr  0    2
   54  sto  0   -1 i
   55  lod  0   -1 i
   56  lit  0  400
   57  opr  0   10
   58  jpc  0   62
   59  lit  0    0
   60  cal  0   30 jloop
   61  int  0   -1
   62  opr  0    0
   63
   63 begin
   64   call iloop(0)
   67 end.
   63  int  0    3
   64  lit  0    0
   65  cal  0   50 iloop
   66  int  0   -1
   67  opr  0    0
  start pl/0
       ...
 t=167 b=165 p=5 (beg kloop)
 s[1]=0       s.l.
 s[2]=0       d.l.
 s[3]=0
 s[4]=100
 s[5]=1
 s[6]=1       d.l
 s[7]=66
 s[8]=10
 s[9]=5
 s[10]=5      d.l.
 s[11]=61
 s[12]=1
 s[13]=9
 s[14]=9      d.l.
 s[15]=41
 s[16]=2
 s[17]=9
 s[18]=13     d.l.
 s[19]=21
 s[20]=3
 s[21]=9
 s[22]=17     d.l.
 s[23]=21
 s[24]=20
 s[25]=5
 s[26]=21     d.l.
 s[27]=28
 s[28]=1
 s[29]=25
 s[30]=25     d.l.
 s[31]=41
 s[32]=2
 s[33]=25
 s[34]=29     d.l.
 s[35]=21
 s[36]=3
 s[37]=25
 s[38]=33     d.l.
 s[39]=21
 s[40]=30
 s[41]=5
 s[42]=37     d.l.
 s[43]=28
 s[44]=1
 s[45]=41
 s[46]=41     d.l.
 s[47]=41
 s[48]=2
 s[49]=41
 s[50]=45     d.l.
 s[51]=21
 s[52]=3
 s[53]=41
 s[54]=49     d.l.
 s[55]=21
 s[56]=40
 s[57]=5
 s[58]=53     d.l.
 s[59]=28
 s[60]=200
 s[61]=1
 s[62]=57     d.l.
 s[63]=48
 s[64]=10
 s[65]=61
 s[66]=61     d.l.
 s[67]=61
 s[68]=1
 s[69]=65
 s[70]=65     d.l.
 s[71]=41
 s[72]=2
 s[73]=65
 s[74]=69     d.l.
 s[75]=21
 s[76]=3
 s[77]=65
 s[78]=73     d.l.
 s[79]=21
 s[80]=20
 s[81]=61
 s[82]=77     d.l.
 s[83]=28
 s[84]=1
 s[85]=81
 s[86]=81     d.l.
 s[87]=41
 s[88]=2
 s[89]=81
 s[90]=85     d.l.
 s[91]=21
 s[92]=3
 s[93]=81
 s[94]=89     d.l.
 s[95]=21
 s[96]=30
 s[97]=61
 s[98]=93     d.l.
 s[99]=28
 s[100]=1
 s[101]=97
 s[102]=97    d.l.
 s[103]=41
 s[104]=2
 s[105]=97
 s[106]=101   d.l.
 s[107]=21
 s[108]=3
 s[109]=97
 s[110]=105   d.l.
 s[111]=21
 s[112]=40
 s[113]=61
 s[114]=109   d.l.
 s[115]=28
 s[116]=300   i
 s[117]=1     s.l.
 s[118]=113   d.l.
 s[119]=48    ret adr (exit jloop)
 s[120]=10
 s[121]=117
 s[122]=117   d.l.
 s[123]=61
 s[124]=1
 s[125]=121
 s[126]=121   d.l.
 s[127]=41
 s[128]=2
 s[129]=121
 s[130]=125   d.l.
```

```
s[131]=21
s[132]=3
s[133]=121
s[134]=129   d.l.
s[135]=21
s[136]=20
s[137]=117
s[138]=133   d.l.
s[139]=28
s[140]=1
s[141]=137
s[142]=137   d.l.
s[143]=41
s[144]=2
s[145]=137
s[146]=141   d.l.
s[147]=21
s[148]=3
s[149]=137
s[150]=145   d.l.
s[151]=21
s[152]=30    j
s[153]=117   s.l.
s[154]=149   d.l.
s[155]=28    ret adr (exit kloop)
s[156]=1
s[157]=153
s[158]=153   d.l.
s[159]=41
s[160]=2
s[161]=153
s[162]=157   d.l.
s[163]=21
s[164]=2     k
s[165]=153   s.l.
s[166]=161   d.l.
s[167]=21    ret adr (if k=3 then)
! 333
```