Programming Language Design 2024
Scopes, functions and parameter passing

Hans Hüttel

19 February 2024

1 Problems that we will talk about in class

1. In the podcast we saw the following example.

```
begin
      var x:= 0;
      var y:= 42

      proc p is x:= x+3;
      proc q is call p;

      begin
          var x:= 9;
          proc p is x:= x+1;

          call q;
          y:=x
      end
end
```

   Now imagine that we have *mixed scope rules*. First consider the case where variables follow static scope rules whereas procedures follow dynamic scope rules. What is the final value of `y` then? And if we instead had dynamic scopes for variables but static scope rules for procedures?

2. A book on program design claims that in the language Pascal the construct `while E do C` is equivalent to calling the procedure

```
procedure whiledo(e : Boolean; procedure c)
   begin
      if e then
         begin c;
               whiledo(e,c)
         end
   end;
```

   where the first parameter is a call-by-value parameter and the second parameter is a procedure parameter, with the call `whiledo(E,PC)`, where `PC` is a parameterless procedure whose body is `C`.

   Why is this incorrect? Suggest a suitable correction.

3. A famous influencer on Instagram proposes to extend his favourite programming language with formal parameter list definitions of the form

```
parlist mypars = (T1 X1,...,Tn Xn)
```

where the `T1,...,Tn` are types.

This could save a lot of space. Instead of writing e.g.

```
function f1(T1 X1,...,Tn Xn)
... { body of f1 appears here }

function f2(T1 X1,...,Tn Xn)
... { body of f2 appears here }
```

we could simply write

```
function f1(mypars)
... { body of f1 appears here }

function f2(mypars)
... { body of f2 appears here }
```

Why should the influencer be discouraged?

4. Here is an ALGOL60 procedure that returns a real number. Remember that the only parameter passing mechanism in this language is call-by-name.

```
real procedure bingo(k, l, u, ak)
      value l, u;
      integer k, l, u;
      real ak;
      comment k and ak are call-by-name formal parameters;
   begin
      real s;
      s := 0;
      for k := l step 1 until u do
         s := s + ak;
      bingo := s
   end;
```

Now let `V` be a real-valued array with 100 entries.

What happens if we call `bingo(i, 1, 100, V[i])`?

2 Additional problems

a. In a language with dynamic scope rules, where is a free identifier in a function definition bound if it does not have a binding occurrence in the context in which it is called?

b. Can you find an example which shows that call-by-name and call-by-reference are not equivalent?