CSc 520 — Principles of Programming Languages

27 : Control Structures — Procedures

Christian Collberg
Department of Computer Science
University of Arizona
`collberg+520@gmail.com`

March 31, 2008

1 Procedures as Control Abstractions

- A procedure is a collection of computation (expressions, statements, etc).that we can give a name.
- A *call-site* is a location where a *caller* invokes a procedure, the *callee*.
- The caller waits for the callee to finish executing, at which time controls to the point after the call-site.
- Most procedures are *parameterized*. The values passed to the procedure are called *actual parameters*.
- The actual parameters are mapped to *formal parameters*, which hold the actual values within the procedure.

2 Procedures as Control Abstractions...

- Some procedures (called functions) return a value. In some languages, a function can return multiple values.
- Most languages use a *call-stack* on which actual parameters and local variables are stored.
- Different languages have different rules as to how parameters should be passed to a procedure.

3 Questions

- How do we deal with recursion? Every new recursive call should get its own set of local variables.
- How do we pass parameters to a procedure?
  - Call-by-Value or Call-by-Reference?
  - In registers or on the stack?
- How do we allocate/access local and global variables?
- How do we access non-local variables? (A variable is non-local in a procedure `P` if it is declared in procedure that statically encloses `P`.)

- How do we pass large structured parameters (arrays and records)?

Case Study — Pascal

4 Pascal Procedures

```
PROCEDURE Name (list of formals);
   CONST (* Constant declarations *)
   TYPE (* Type declarations *)
   VAR (* Variable declarations *)
      (* Procedure and function definitions *)
BEGIN
   (* procedure body *)
END;
```

- Note the similarity with the program structure.
- Note that procedures can be nested.
- Note the semicolon after the end.

5 Pascal Procedures...

- Formal parameters look like this:

```
procedure name (formal1:type1; formal2:type2;...);
         or like this
procedure name (formal1,formal2...:type1; ...);
```

- By default, arguments are **passed by value**. **var** indicates that they are **passed by reference**:

```
procedure name (var formal1:type1; ...);
```

6 Pascal Procedures...

- Functions are similar to procedures but return values:

```
function func1 (formals);
begin
   func1 := 99;
end;
```

- To return a value assign it to the function name.

7 Pascal Procedures...

- Procedures can be nested:

```
procedure A ();
   procedure B();
   begin
         ...
   end;
begin
   ...
end;
```

- Names declared in an outer procedure are visible to nested procedures unless the name is redeclared.

8 Pascal Procedures...

- Procedures can be recursive. The **forward** declaration is used to handle mutually recursive procedures:

```
procedure foo (); forward;

procedure bar ();
begin
   foo();
end;

procedure foo();
begin
   bar();
end;
```

Case Study — Ada

9 Ada — Subprogram Declarations

```
procedure Traverse_Tree;
procedure Increment(X : in out Integer);
procedure Right_Indent(Margin : out Line_Size);
procedure Switch(From, To : in out Link);

function Random return Probability;

function Min_Cell(X : Link) return Cell;
function Next_Frame(K : Positive) return Frame;
function Dot_Product(Left, Right : Vector)
                       return Real;
```

10 Ada — Subprogram Declarations

```
function "*"(Left, Right : Matrix) return Matrix;
```

- Examples of in parameters with default expressions:

```
procedure Print_Header(Pages  : in Natural;
          Header : in Line    :=
                   (1 .. Line'Last => ' ');
          Center : in Boolean := True);
```

11 Ada — Subprogram Bodies

```
-- Example of procedure body:

procedure Push(E : in Element_Type;
               S : in out Stack) is
begin
   if S.Index = S.Size then
      raise Stack_Overflow;
   else
      S.Index := S.Index + 1;
      S.Space(S.Index) := E;
   end if;
end Push;
```

12 Ada — Procedure Call

```
Traverse_Tree;
Print_Header(128, Title, True);
```

```
Switch(From => X, To => Next);
Print_Header(128, Header => Title,
             Center => True);
Print_Header(Header=>Title,
             Center=>True, Pages=>128);

--Examples of function calls:
Dot_Product(U, V)
Clock
```

13 Ada — Procedure Call

```
-- Procedures with default expressions:
procedure Activate(
      Process : in Process_Name;
      After   : in Process_Name:=No_Process;
      Wait    : in Duration := 0.0;
      Prior   : in Boolean := False);

procedure Pair(Left, Right :
                  in Person_Name:=new Person);
```

14 Ada — Procedure Call. . .

```
-- Examples of their calls:
Activate(X);
Activate(X, After => Y);
Activate(X, Wait => 60.0, Prior => True);
Activate(X, Y, 10.0, False);
Pair;
Pair(Left => new Person, Right => new Person);
```

15 Ada — Overloaded Calls

```
procedure Put(X : in Integer);
procedure Put(X : in String);

procedure Set(Tint   : in Color);
procedure Set(Signal : in Light);

-- Examples of their calls:
Put(28);
Put("no possible ambiguity here");

Set(Tint=>Red);   --  Set(Red) is ambiguous.
Set(Signal=>Red); --  Red can denote either
Set(Color'(Red)); --  a Color or a Light
```

16 Ada — Userdefined Operators

```
function "+" (Left,Right:Matrix) return Matrix;
function "+" (Left,Right:Vector) return Vector;

-- assuming that A, B, and C are of
-- the type Vector the following two
-- statements are equivalent:

A := B + C;
A := "+"(B, C);
```

17 Readings and References

- **Read Scott, pp. 117–123, 428–433**

18 Summary

- Each procedure call pushes a new activation record on the run-time stack. The AR contains local variables, actual parameters, a static (access) link, a dynamic (control) link, the return address, saved registers, etc.
- The frame pointer (FP) (which is usually kept in a register) points to a fixed place in the topmost activation record. Each local variable and actual parameter is at a fixed offset from FP.

19 Summary...

- The dynamic link is used to restore the FP when a procedure call returns.
- The static link is used to access non-local variables, i.e. local variables which are declared within a procedure which statically encloses the current one.