

Var

a : Procedure; // error

b : ^Procedure;

j, k, l : Integer;

Project II:

{ expressions
assignment statements
procedure/function
definitions & calls
compound statements
(straight-line code)

compound_statement:

begin

{ list of statements
separated by semicolons

end

statement:
label: unlabeled_stmt ^{not used}
{ unlabeled_stmt }

unlabeled_stmt:
structured_stmt
| simple_stmt
;

simple_statement:

empty_stmt ^{never used}

| goto_stmt

| assignment_or_call_stmt

| standard_procedure_statement

| statement_block ^{not used}

;

exprs vs statements
things that have values (they are evaluated or forced) vs don't have values (just executed)

C vs pascal

$x = 6$ ^{assignment expression}

$z = y = (x = 6)$

$(x = 6) + y$

Pascal: assignments are statements

$x := 6$;

$y := x$;

$z := 6$

$y := x$

In C:

$x = 6$;

6 ;

x ;

$(x = 6) + y$;

In Pascal:

$x := 6$;

begin

$x := 6$;

$y := 4$;

$z := 3$;

end

begin

$x := 6$;

$z := x$;

assignment_or_call_stmt:
(variable, out-parameters, in-parameters)
return rest_of_stmt

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

rest_of_stmt:

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

procedure_call

call (x, y, z)

```

Function Foo : Integer;
}

Procedure Bar;

In Pascal, procedure calls are
statements;
function calls are expressions

X := Foo // ok
Bar      // ok
X := Bar // semantic error

```

```

Function Bar : Integer;

```

```

(Bar ^) := 6 // okay!

```

```

vofani:
  identifier
  | address operator variable or function access
  | variable or function access handle
  | "vofani"
  ;

```

```

vofani:
  ; ('expression')
  | var-or-fun-access pointer-char
  | var-or-fun-access '[' index-exp-list ']'
  | var-or-fun-access has standard fun ('actual-
  | param-list')
  | pNEW ('var-or-access-type-name')
  ;

```

```

var-or-fun-access:
  var-or-fun-access has
  | var-or-fun-access LEXAS type-name
  ;

```

```

var-or-fun-access has:
  var-or-fun-access handle fun
  | standard functions
  ;
var-or-fun-access has standard fun:
  identifier
  | vofani
  ;

```

```

type
  ListNode = record
    data : Integer;
    next : ^ListNode;
  end;

```

```

var
  list : ^ListNode;
  p : ^ListNode;
  ;
begin
  new(list);
  list^.data := 6;
  list^.next := nil;
  new(p);
  p^.data := 4;
  p^.next := list;
  list := p;

```

