Breaking Problems Down

This section of notes shows you how to break down a large programming problem into Pascal functions and procedures.

James Tam

Developing An Algorithm: Top-Down Approach

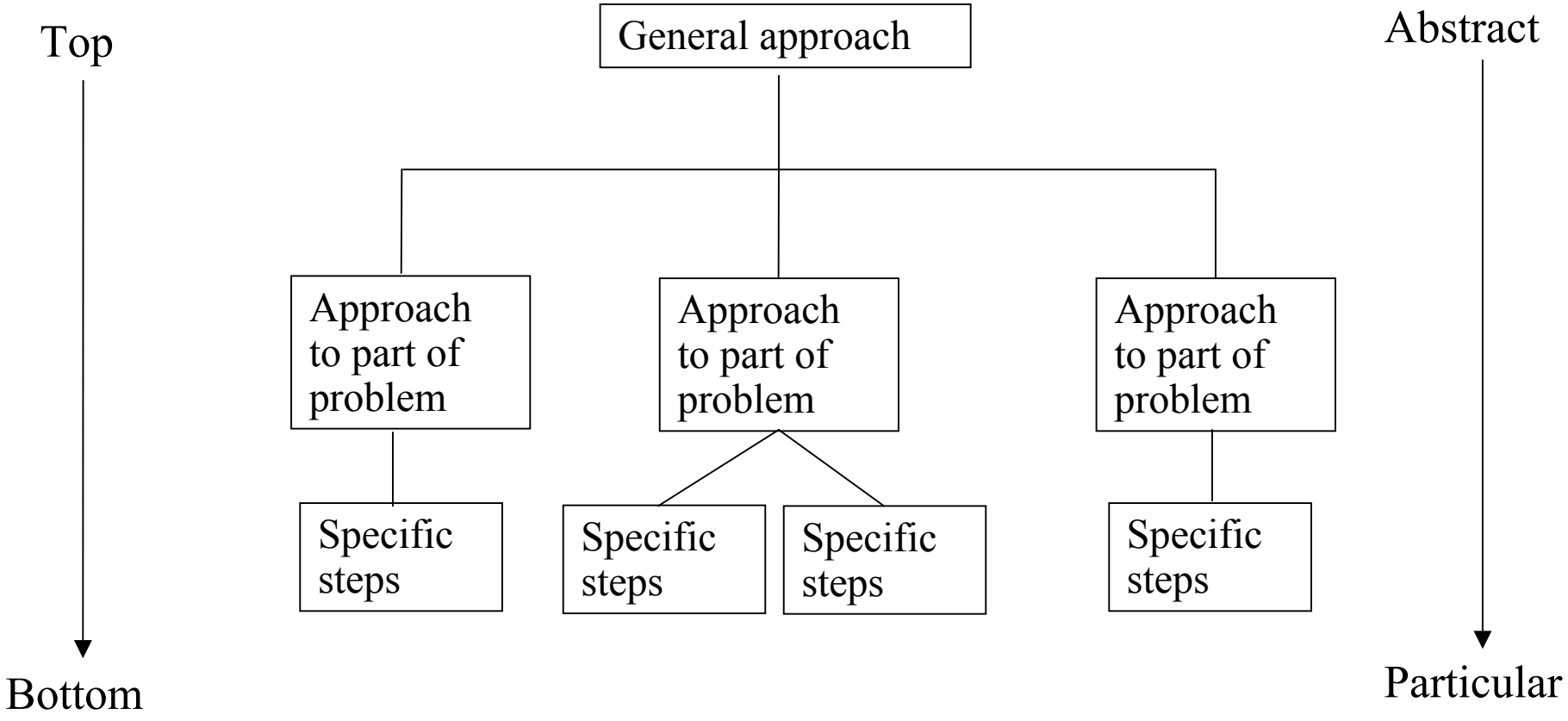
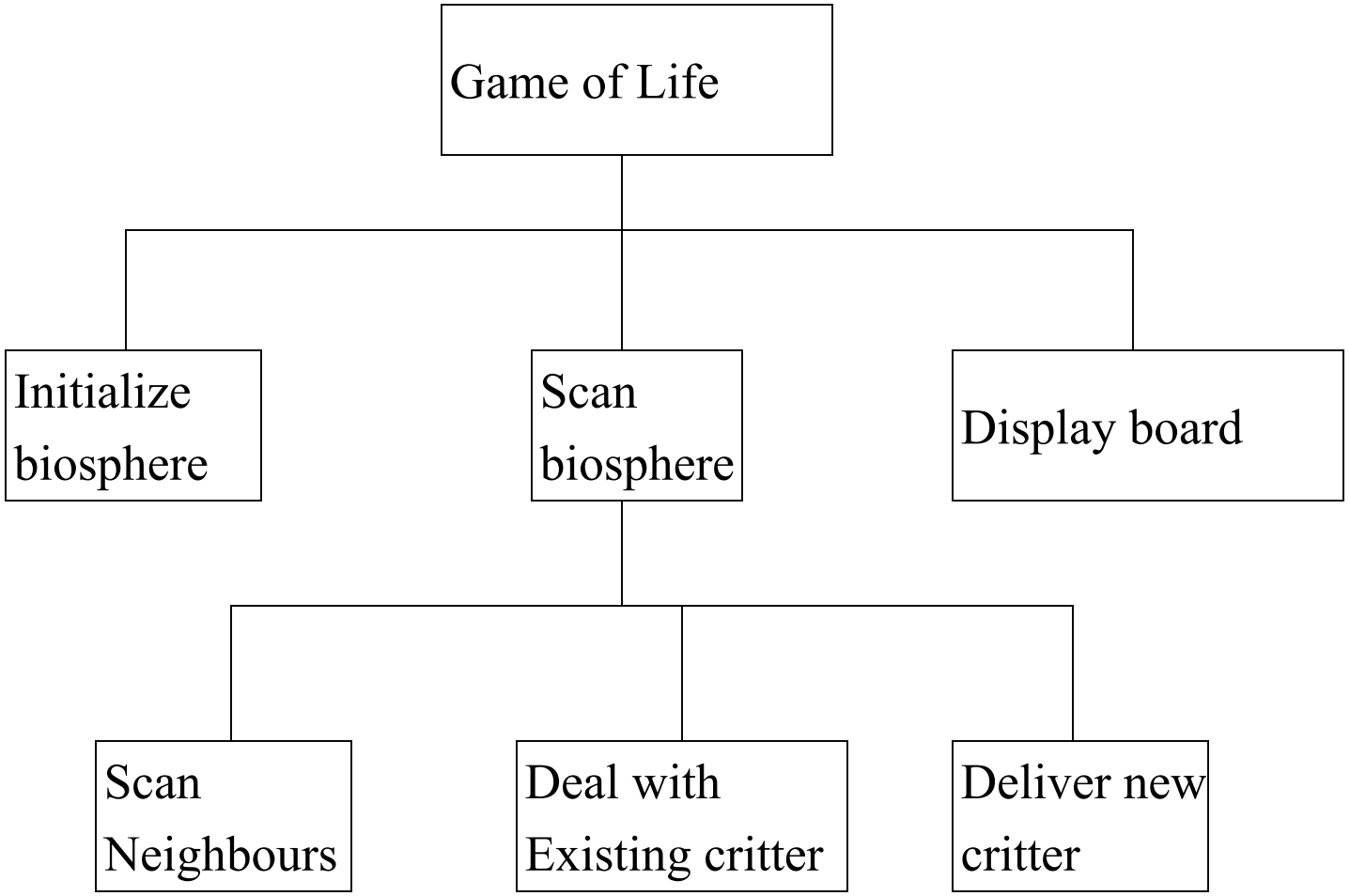


Figure extracted from Computer Science Illuminated by Dale N. and Lewis J.

Example Of Top Down Approach



Decomposing Problems (Top Down Approach)

Characteristics

- Breaking problem into smaller, well defined modules
- Making modules as independent as possible (loose coupling)

Benefit

- Solution is easier to visualize
- Easier to maintain (if modules are independent)

Drawback

- Complexity – understanding and setting up inter module communication may appear daunting at first

Pascal implementation

- Procedures
- Functions

Referring To Functions And Procedures In Pascal

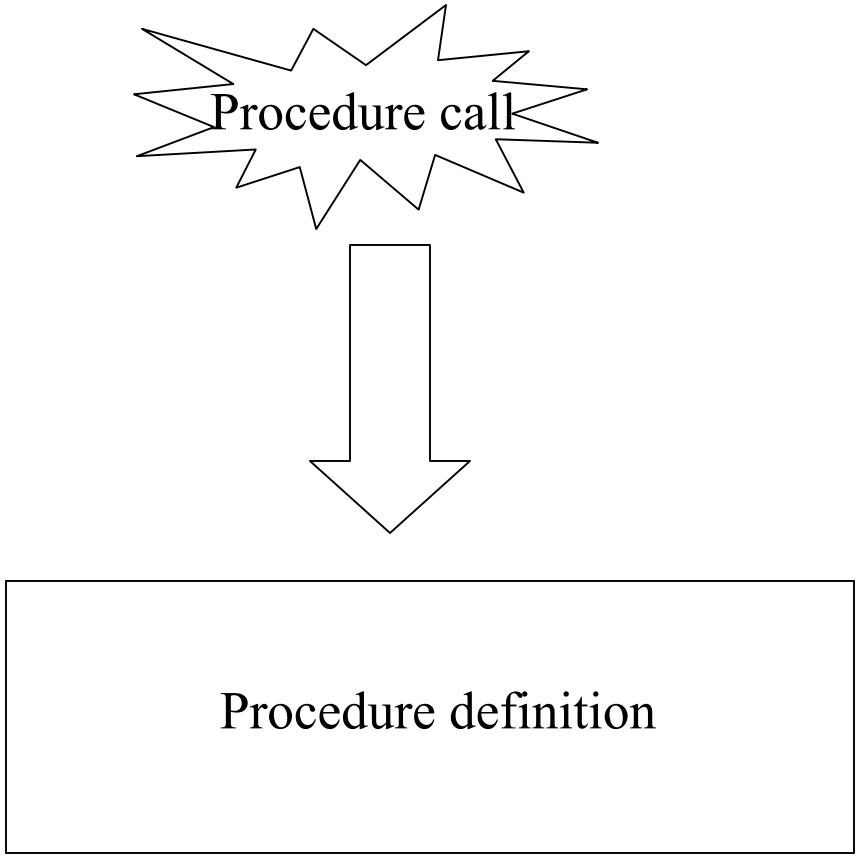
Definition

- Indicating what the function or procedure will do

Call

- Invoking the function or procedure

Procedures (Basic Case – No Parameters)



Defining Procedures

Syntax (Basic case – no parameters):

```
procedure name;
begin
        (* Statements of the function go here *)
end;  (* End of procedure name *)
```

Example (Basic case – no parameters):

```
procedure displayInstructions;
begin
        writeln ('These statements will typically give a high level');
        writeln('overview of what the program as a whole does');
end;  (* End of procedure displayInstructions *)
```

Calling A Procedure

Syntax (Basic case – no parameters):

name;

Example (Basic case – no parameters):

displayInstructions;

Procedures: Putting Together The Basic Case

A compilable version of this example can be found in Unix under
/home/231/examples/functions/firstExampleProcedure.p

```
procedure displayInstructions;
begin
   writeln ('These statements will typically give a high level');
   writeln('overview of what the program as a whole does');
end; (*Procedure displayInstructions *)

begin
    displayInstructions;
     writeln('Thank you, come again!');
end. (* Program *)
```

Procedures: Putting Together The Basic Case

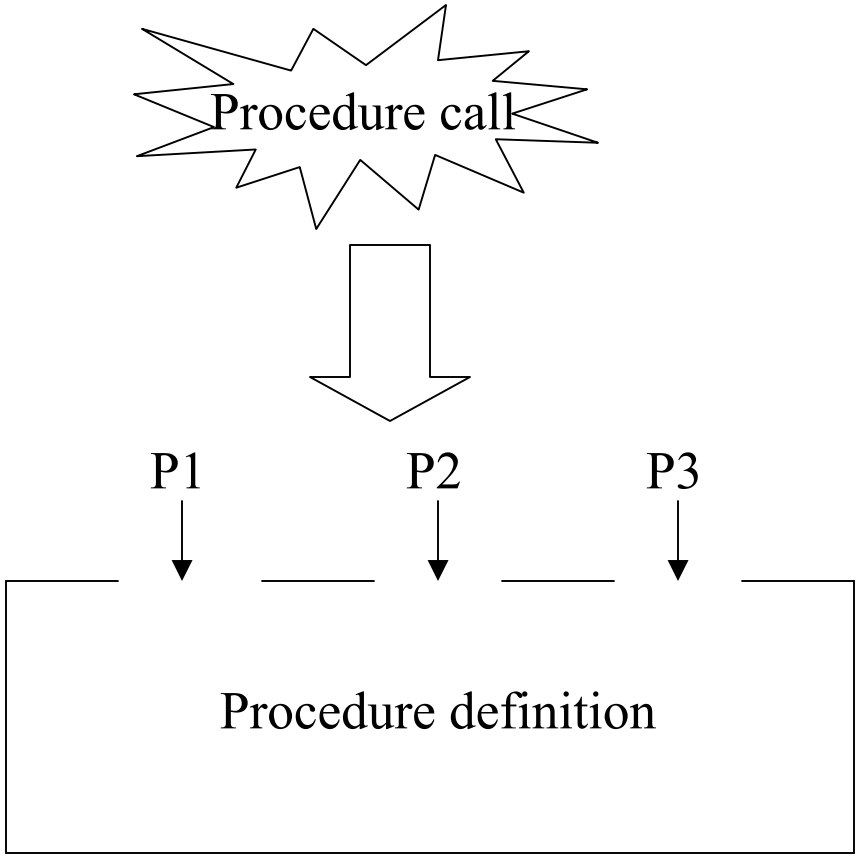
A compilable version of this example can be found in Unix under
/home/231/examples/functions/firstExampleProcedure.p

```
procedure displayInstructions;
begin
   writeln ('These statements will typically give a high level');
   writeln('overview of what the program as a whole does');
end; (*Procedure displayInstructions *)

begin
    displayInstructions;
    writeln('Thank you, come again!');
end. (* Program *)
```

Procedure definition

Procedure call

Procedures With Parameters



Defining Procedures With Parameters

Syntax:

```
procedure name (Name of parameter 1 : type of parameter 1;
                Name of parameter 2 : type of parameter 2;
                        :                   :
                Name of parameter n : type of parameter n);
  begin
      (* Statements of the function go here *)
  end;
```

Example:

```
procedure celciusToFahrenheit (celciusValue :  real);
var
   fahrenheitValue : real;
begin
    fahrenheitValue := 9 / 5 * celciusValue + 32;
    writeln(temperature in Celsius: ', celciusValue:0:2);
    writeln(temperature in Fahrenheit: ', fahrenheitValue:0:2);
end; (* Procedure celciusToFahrenheit *)
```

Calling Procedures With Parameters

Syntax:

name (*Name of parameter 1*, *Name of parameter 2…Name of parameter n*);

Example:

celciusToFahrenheit(celciusValue);

Procedures: Putting Together The Case With Parameters

A compilable version of this example can be found in Unix under
/home/231/examples/functions/temperatureConverter..p

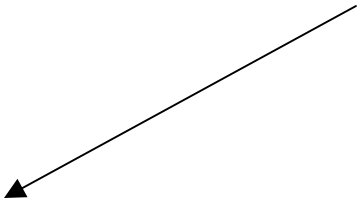
```
program temperatureConverter (input, output);
var
   celciusValue :  real;

procedure celciusToFahrenheit (celciusValue :  real);
var
   fahrenheitValue : real;
begin
   fahrenheitValue := 9 / 5 * celciusValue + 32;
   writeln('Temperature in Celsius: ', celciusValue:0:2);
   writeln('Temperature in Fahrenheit: ', fahrenheitValue:0:2);
end; (* Procedure celciusToFahrenheit *)
```

Procedures: Putting Together The Case With Parameters

A compilable version of this example can be found in Unix under
/home/231/examples/functions/temperatureConverter.p

```
program temperatureConverter (input, output);
var
   celciusValue :  real;
```

Procedure definition



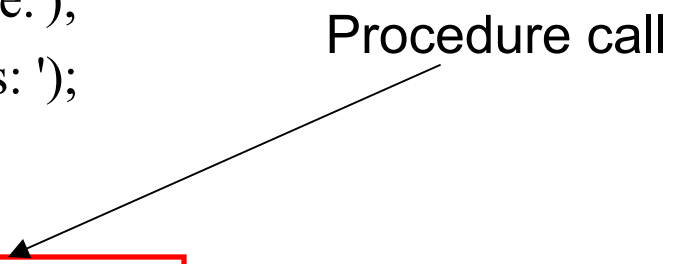
```
procedure celciusToFahrenheit (celciusValue :  real);
var
   fahrenheitValue : real;
begin
   fahrenheitValue := 9 / 5 * celciusValue + 32;
   writeln('Temperature in Celsius: ', celciusValue:0:2);
   writeln('Temperature in Fahrenheit: ', fahrenheitValue:0:2);
end; (* Procedure celciusToFahrenheit *)
```

Procedures: Putting Together The Case With Parameters (2)

```
begin
   writeln;
   writeln('This program will convert a given temperature from a
     Celsius's);
   writeln('value to a Fahrenheit value.');
   write('Input temperature in Celsius: ');
   readln(celciusValue);
   writeln;
   celciusToFahrenheit(celciusValue);
   writeln('Thank you and come again.');
end. (* Program *)
```

Procedures: Putting Together The Case With Parameters (2)

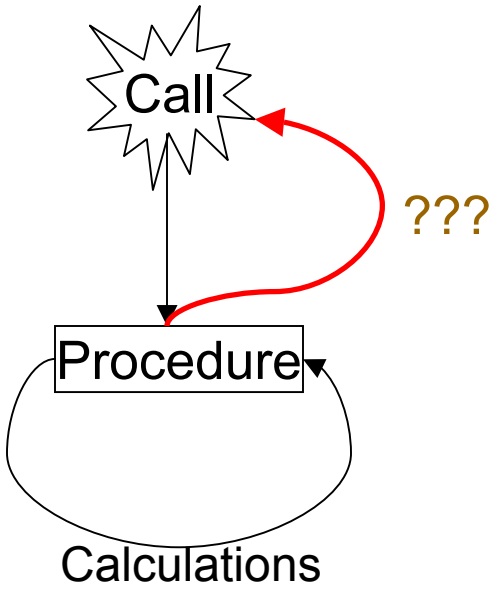
```
begin
   writeln;
   writeln('This program will convert a given temperature from a
     Celsius's);
   writeln('value to a Fahrenheit value.');
   write('Input temperature in Celsius: ');
   readln(celciusValue);
   writeln;
   celciusToFahrenheit(celciusValue);
   writeln('Thank you and come again.');
end. (* Program *)
```



How Retain Information From A Module After The Module (Function Or Procedure) Has Ended

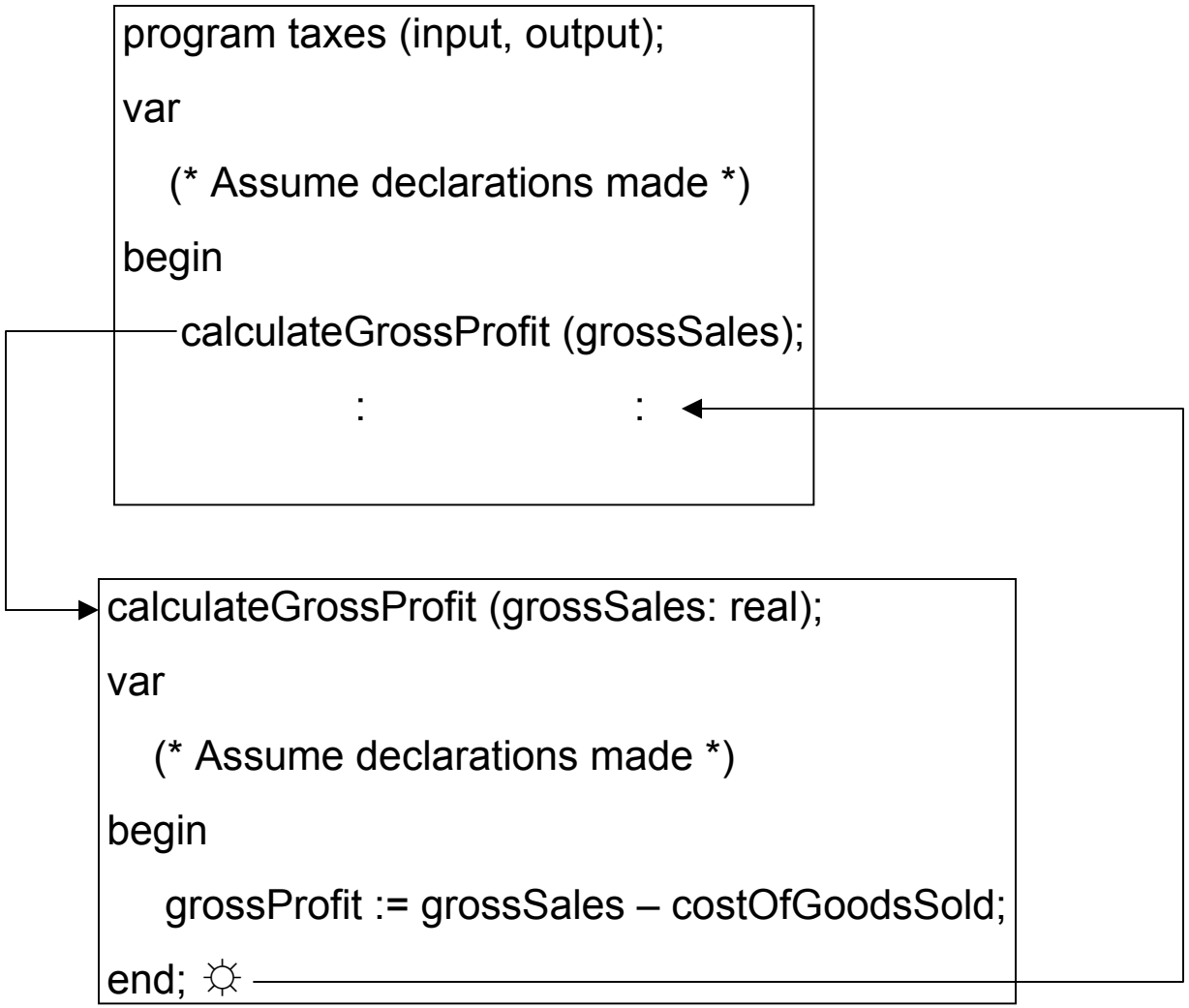
Information needs to be returned from the module

i.e.,



How Retain Information From A Module After The Module (Function Or Procedure) Has Ended (2)

e.g., producing an income statement

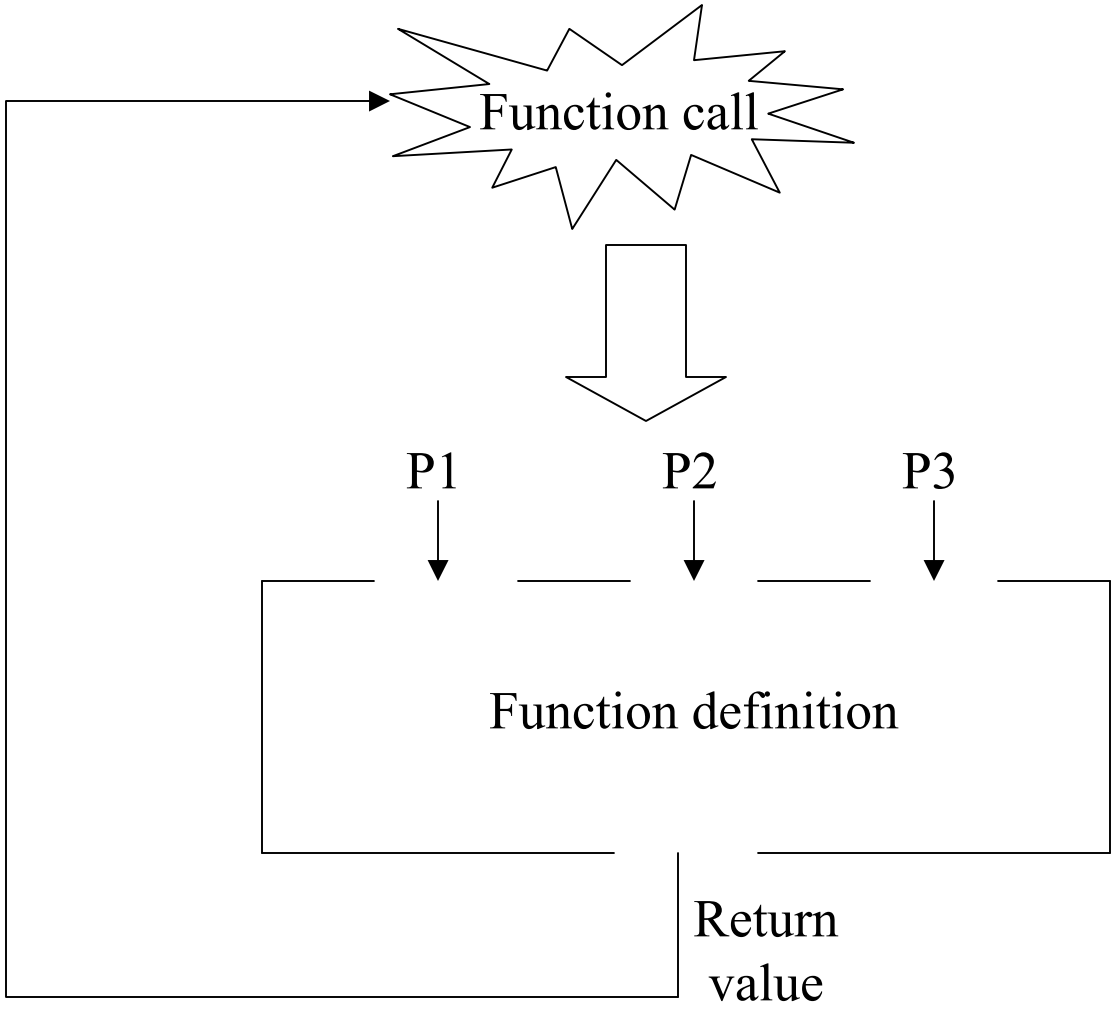


How Retain Information From A Module After The Module (Function Or Procedure) Has Ended (3)

Methods:

- Return a value with a function
- Pass parameters into the procedure as variable parameters (rather than as value parameters)

Functions



Defining Functions

Syntax:

```
function name (Name of parameter 1 : type of parameter 1;
               Name of parameter 2 : type of parameter 2;
                        :                   :
               Name of parameter n : type of parameter n):
                 return type;
 begin
     (* Statements of the function go here *)
              :                :
     name := expression; (* Return value *)
 end;
```

Should be the last statement in the function

Example:

```
function calculateGrossIncome (grossSales, costOfGoodsSold : real) : real;
begin
  calculateGrossIncome := grossSales - costOfGoodsSold;
end;
```

Calling Functions

Syntax:

name;

name (*name of parameter 1*, *name of parameter 2…name of parameter n*);

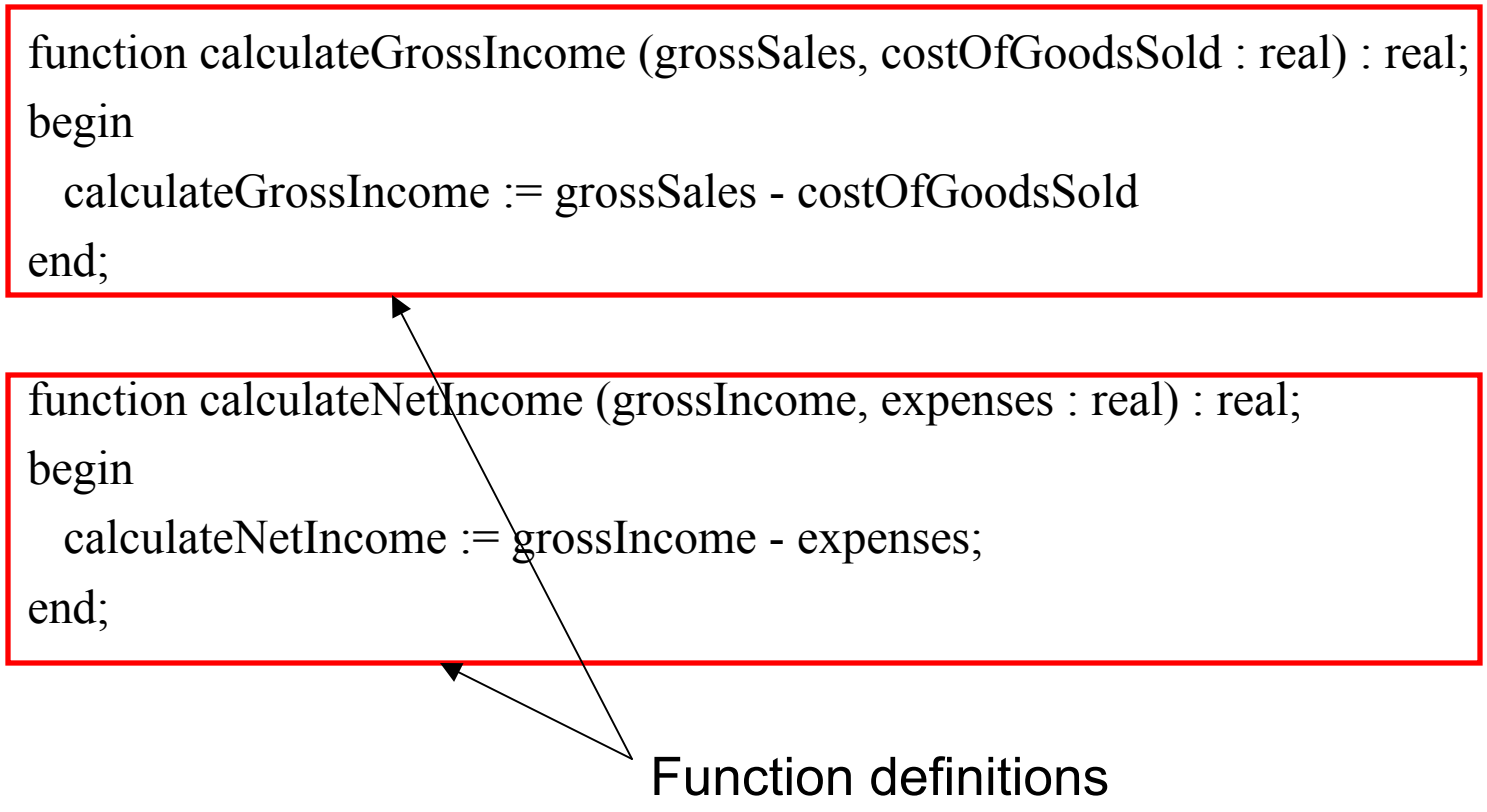
Example:

```
grossIncome := calculateGrossIncome (grossSales, costOfGoodsSold);
```

Functions: Putting It All Together

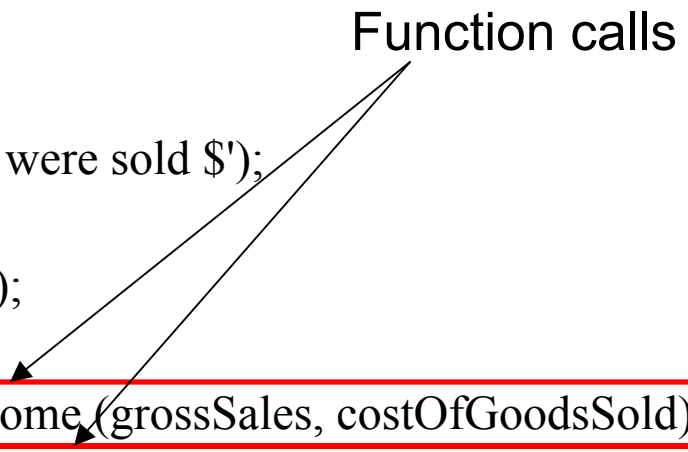
A compilable version of this example can be found in Unix under
/home/231/examples/functions/financialStatements.p

```
program financialStatments (input, output);

function calculateGrossIncome (grossSales, costOfGoodsSold : real) : real;
begin
  calculateGrossIncome := grossSales - costOfGoodsSold
end;

function calculateNetIncome (grossIncome, expenses : real) : real;
begin
  calculateNetIncome := grossIncome - expenses;
end;
```



Functions: Putting It All Together (2)

```
procedure produceIncomeStatement;
var
   grossSales          : real;
   costOfGoodsSold : real;
   grossIncome         : real;
   expenses            : real;
   netIncome           : real;
begin
   write('Input gross sales $');
   readln(grossSales);
   write('Input cost of the goods that were sold $');
   readln(costOfGoodsSold);
   write('Input corporate expenses $');
   readln(expenses);
   grossIncome := calculateGrossIncome (grossSales, costOfGoodsSold);
   netIncome := calculateNetIncome (grossIncome, expenses);
```



Functions: Putting It All Together (3)

```
(* Procedure produceIncomeStatement *)
   writeln;
   writeln('Gross sales $':26, grossSales:0:2);
   writeln('Less: cost of goods sold $':26, costOfGoodsSold:0:2);
   writeln('Gross income $':26, grossIncome:0:2);
   writeln('Less: expenses $':26, expenses:0:2);
   writeln('Net income $':26, netIncome:0:2);
   writeln;
end; (* End of procedure produceIncomeStatement *)
```

Functions: Putting It All Together (3)

```
(* Start of program *)
begin
   writeln;
   writeln('This program will produce an income statement based upon your');
   writeln('gross sales figures, the cost of the goods that you sold and
   writeln('your expenses.');
   writeln;
   produceIncomeStatement;
   writeln('Thank you, come again!');
end.  (* End of entire program. *)
```

How Retain Information From A Module After The Module (Function Or Procedure) Has Ended (3)

Methods:

- Return a value with a function
- Pass parameters into the procedure as variable parameters (rather than as value parameters)

Passing Parameters As Value Parameters

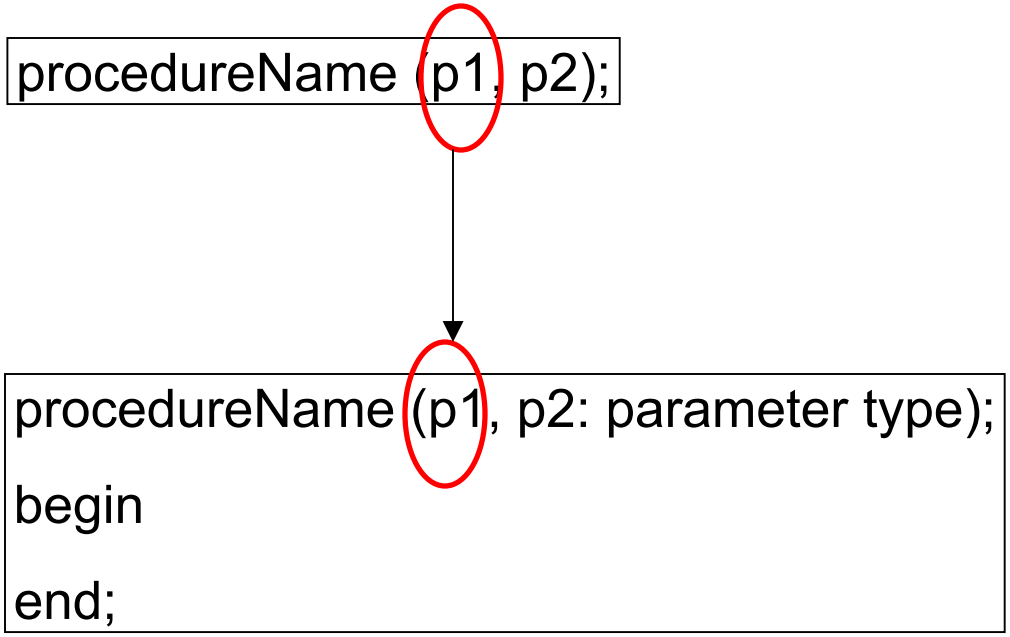
Previous examples

```
procedureName (p1, p2);
```

```
procedureName (p1, p2: parameter type);
begin
end;
```

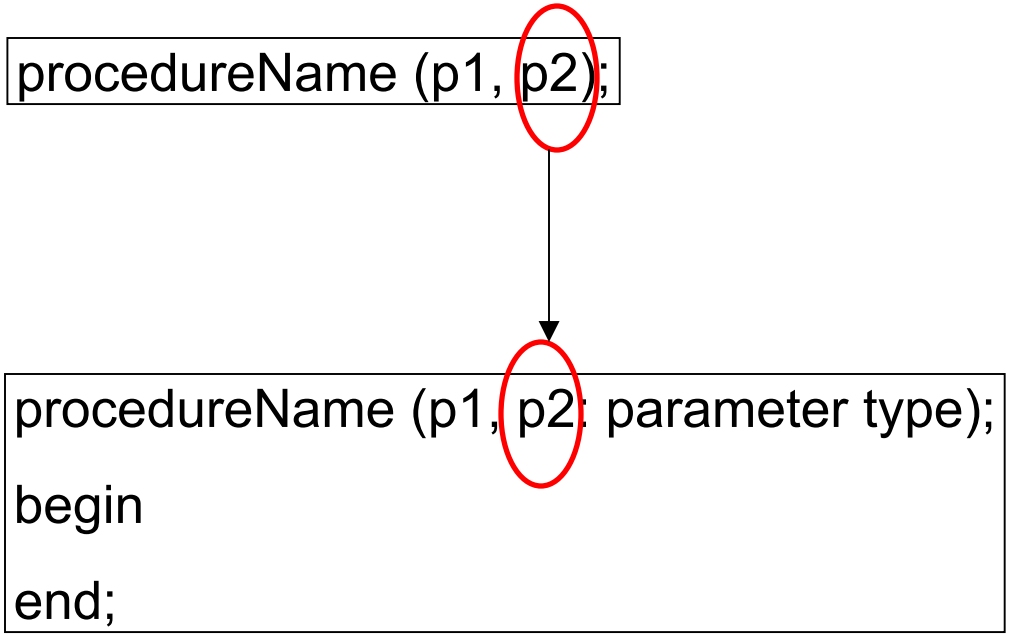
Passing Parameters As Value Parameters

Previous examples



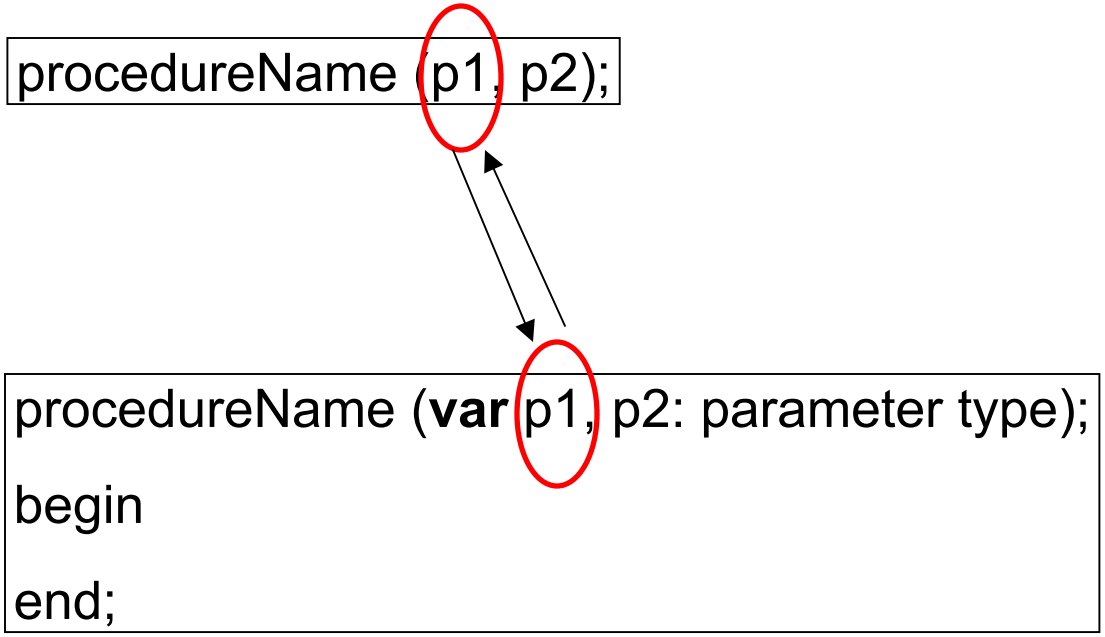
Passing Parameters As Value Parameters

Previous examples



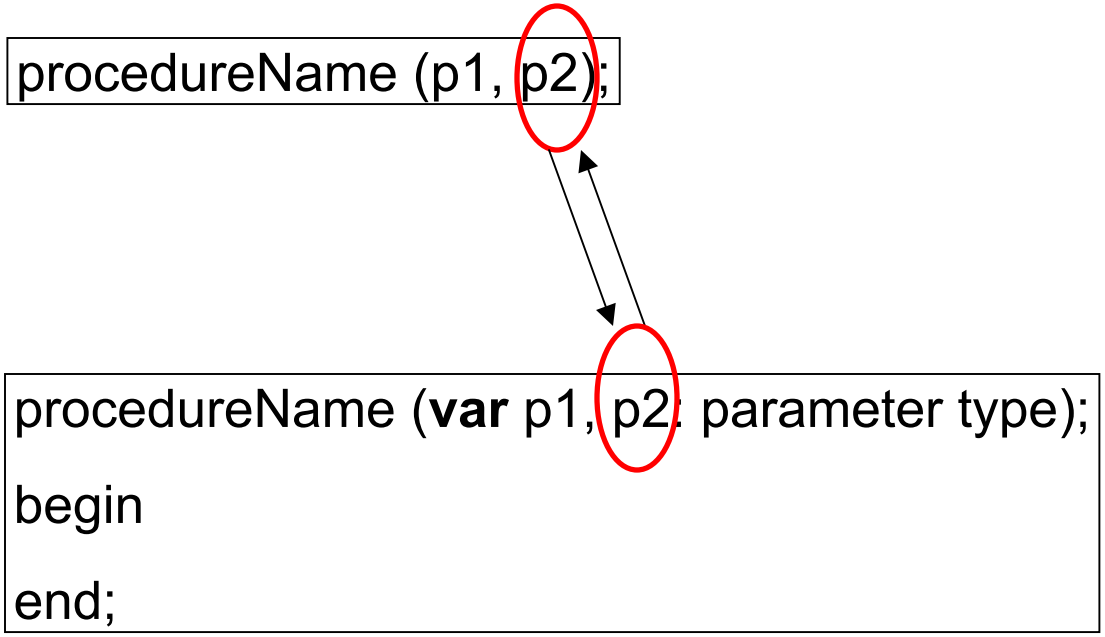
Passing Parameters As Variable Parameters

Example coming up



Passing Parameters As Variable Parameters

Example coming up



Procedure Definitions When Passing Parameters As Variable Parameters

Syntax:

```
procedure name (var Name of parameter 1 : type of parameter 1;
                var Name of parameter 2 : type of parameter 2;
                    :                       :
                var Name of parameter n : type of parameter n);
   begin
      (* Statements of the function go here *)
   end;
```

Example:

```
   procedure tabulateIncome (      grossSales          : real;
                                   costOfGoodsSold     : real;
                               var grossIncome         : real;
                                   expenses            : real;
                                var netIncome          : real);
begin
  grossIncome := grossSales - costOfGoodsSold;
  netIncome := grossIncome - expenses;
end;
```

Calling Procedures With Variable Parameters

Same as calling procedures with value parameters!

Syntax:

name (*name of parameter 1*, *name of parameter 2…name of parameter n*);

Example:

```
tabulateIncome(grossSales,costOfGoodsSold,grossIncome,expenses,
netIncome);
```

Passing Variable Parameters: Putting It All Together

A compilable version of this example can be found in Unix under
/home/231/examples/functions/financialStatements2.p

```
program financialStatments (input, output);

procedure getIncomeInformation (var grossSales          : real;
                                var costOfGoodsSold : real;
                                var expenses            : real);
begin
  write('Input gross sales $');
  readln(grossSales);
  write('Input the cost of the goods that were sold $');
  readln(costOfGoodsSold);
  write('Input business expenses $');
  readln(expenses);
end; (* End of procedure getIncomeInformation *)
```

Passing Variable Parameters: Putting It All Together (2)

```
procedure tabulateIncome (      grossSales          : real;
                                costOfGoodsSold : real;
                           var grossIncome        : real;
                                expenses           : real;
                           var netIncome          : real);
begin
   grossIncome := grossSales - costOfGoodsSold;
   netIncome := grossIncome - expenses;
end; (* End of procedure tabulateIncome *)

procedure displayIncomeStatement (grossSales          : real;
                                  costOfGoodsSold : real;
                                  grossIncome         : real;
                                  expenses            : real;
                                  netIncome           : real);
```

Passing Variable Parameters: Putting It All Together (3)

```
(* Procedure displayIncomeStatement *)
begin
   writeln;
   writeln('INCOME STATEMENT':40);
   writeln('Gross sales $':40, grossSales:0:2);
   writeln('Less: Cost of the goods that were sold $':40,
     costOfGoodsSold:0:2);
   writeln('Equals: Gross Income $':40, grossIncome:0:2);
   writeln('Less: Business Operating Expenses $':40, expenses:0:2);
   writeln('Equals: Net income $':40, netIncome:0:2);
   writeln;
end; (* End of procedure displayIncomeStatement *)

procedure produceIncomeStatement;
var
   grossSales, grossIncome, costOfGoodsSold, expenses, netIncome :real;
```

Passing Variable Parameters: Putting It All Together (3)

```
begin
   getIncomeInformation(grossSales, costOfGoodsSold, expenses);
   tabulateIncome(grossSales,costOfGoodsSold,grossIncome,expenses,netIncome);
   displayIncomeStatement
     (grossSales,costOfGoodsSold,grossIncome,expenses,netIncome);
end;

(* Begin main program *)
begin
   writeln;
   writeln('This program will produce an income statement based upon your');
   writeln('gross sales figures, the cost of the goods that you sold and');
   writeln('your expenses.');
   writeln;
   produceIncomeStatement;
   writeln('Thank you, come again!');
end. (* End of main program *)
```

Summary

How to break up a Pascal program into modules using functions and procedures

What is the difference between a procedure/function definition and a call as well as how to write each one in Pascal with the different cases:

- No parameters
- Parameters passed in as variable parameters
- Parameters passed in as value parameters