

Breaking Problems Down

This section of notes shows you how to break down a large programming problem into smaller modules that are easier to manage.

James Tam

Designing A Program: Top-Down Approach

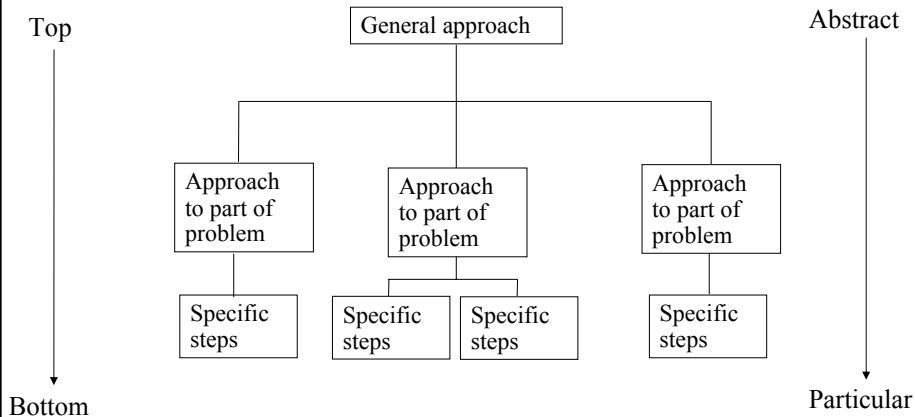
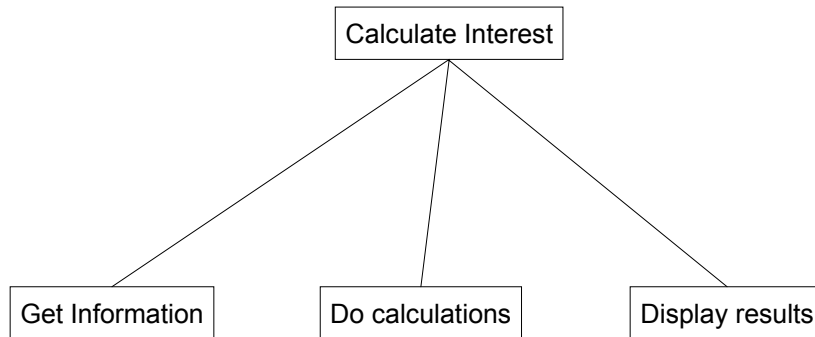


Figure extracted from Computer Science Illuminated by Dale N. and Lewis J.

James Tam

Top Down Approach: Programming



James Tam

Decomposing Problems Via The 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 of program modules

- Procedures
- Functions

James Tam

Using Functions And Procedures In Pascal

Definition

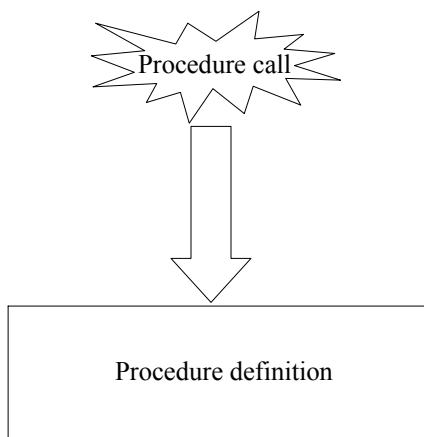
- Indicating what the function or procedure will do

Call

- Executing the code in the function or procedure

James Tam

Procedures (Basic Case – No Parameters)



James Tam

Defining Procedures (Basic Case – No Parameters)

Format:

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

Example:

```
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 *)
```

James Tam

Where To Define Modules (Procedures)

Header

Declarations

```
const  
Procedure and function definitions  
:
```

Statements

```
begin  
  
end.
```

James Tam

Defining Local Variables

Exist only for the life the module

Format:

```
procedure name;  
var  
  (* Local variable declarations go here *)  
begin  
  :  
end;
```

Example:

```
procedure proc;  
var  
  num : integer;  
  ch  : char;  
begin  
  :  
end; (* Procedure celciusToFahrenheit *)
```

James Tam

Calling A Procedure (Basic Case – No Parameters)

Format:

```
name;
```

Example:

```
displayInstructions;
```

James Tam

Where To Call Modules (Procedures

Most anywhere in the program

Header

Declarations

```
const  
Procedure and function definitions  
:
```

Statements

```
begin  
Call module: This example  
end.
```

James Tam

Procedures: Putting Together The Basic Case

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

```
program firstExampleProcedure (output);  
  
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 *)
```

James Tam

Procedures: Putting Together The Basic Case

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

```
program firstExampleProcedure (output);
```

```
  procedure displayInstructions;
```

```
  begin
```

```
    writeln('These statements will typically give a high level');
```

```
    writeln('overview of what the program as a whole does');
```

```
  end; (*Procedure display Instructions *)
```

```
  begin
```

```
    displayInstructions;
```

```
    writeln('Thank you, come again!');
```

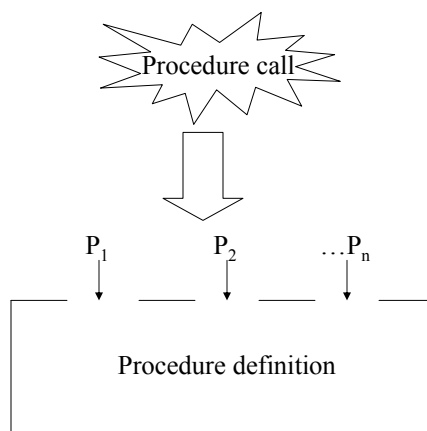
```
  end. (* Program *)
```

Procedure definition

Procedure call

James Tam

Procedures With Parameters



James Tam

Defining Modules (Procedures) With Parameters

Format:

```
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 procedure 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 *)
```

James Tam

Calling Modules (Procedures) With Parameters

Format:

```
name (Name of parameter 1, Name of parameter 2...Name of  
      parameter n);
```

Example:

```
celciusToFahrenheit (celciusValue);
```

James Tam

Procedures: Putting Together The Case Of Procedures With Parameters

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

```
program temperatureConverter (input, output);

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 *)
```

James Tam

Procedures: Putting Together The Case Of Procedures With Parameters

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

```
program temperatureConverter (input, output);
```

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 *)
```

James Tam

Procedures: Putting Together The Case Of Procedures With Parameters (2)

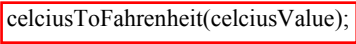
```
begin
  var celciusValue : real;
  writeln;
  writeln('This program will convert a given temperature from a Celsius');
  writeln('value to a Fahrenheit value. ');
  write('Enter a temperature in Celsius: ');
  readln(celciusValue);
  writeln;
  celciusToFahrenheit(celciusValue);
  writeln('Thank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

James Tam

Procedures: Putting Together The Case Of Procedures With Parameters (2)

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

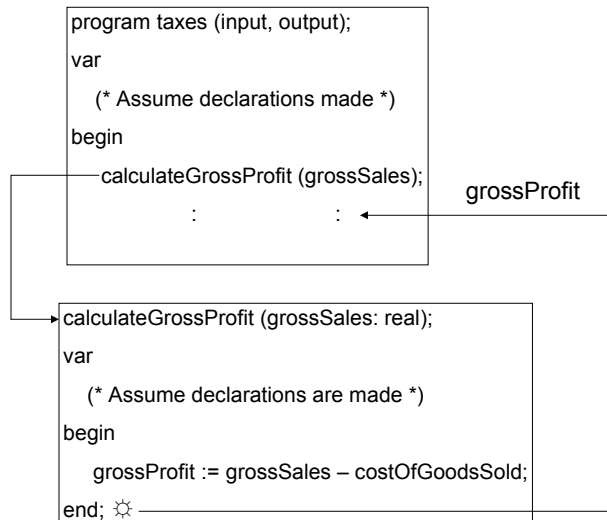
Procedure call



James Tam

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

e.g., producing an income statement



James Tam

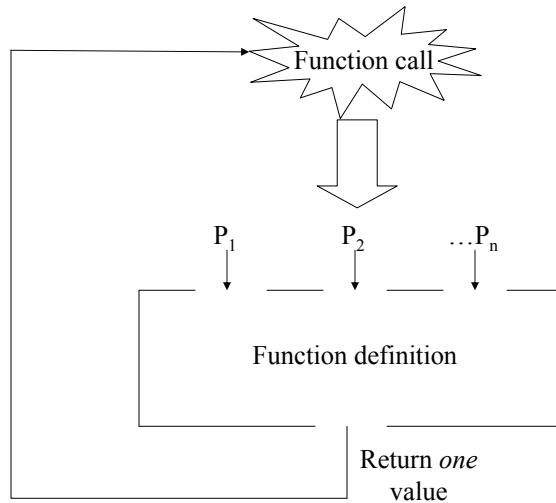
Retaining Information From A Module (Function Or Procedure) After The Module Has Ended (2)

Methods:

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

James Tam

Functions



James Tam

Defining Functions

Format:

```
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;
```

Example:

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

James Tam

Defining Functions

Format:

```
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;
```

Often the last statement in the function

Example:

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

James Tam

Calling Functions

Format:

```
name;
```

```
name (name of parameter 1, name of parameter 2...name of  
parameter n);
```

Example:

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

James Tam

Functions: Putting It All Together

A compilable version of this example can be found in Unix under
/home/231/examples/modules/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;
```

James Tam

Functions: Putting It All Together

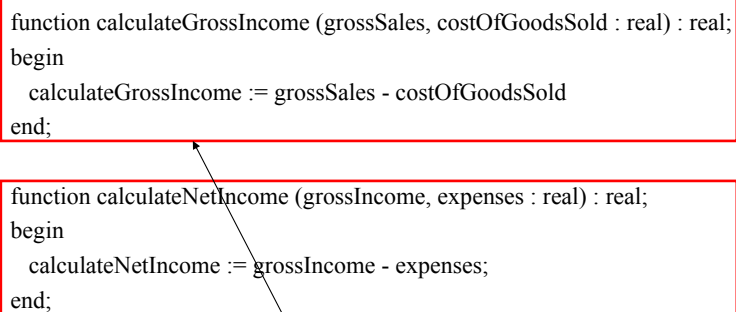
A compilable version of this example can be found in Unix under
/home/231/examples/modules/financialStatements.p

```
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;
```

Function definitions



James Tam

Functions: Putting It All Together (2)

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

James Tam

Functions: Putting It All Together (2)

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

Function calls

James Tam

Functions: Putting It All Together (3)

```
(* Procedure produceIncomeStatement continued *)  
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 *)
```

James Tam

Functions: Putting It All Together (4)

```
(* Start of 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 entire program. *)
```

James Tam

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

Methods:

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

James Tam

Passing Parameters As Value Parameters

Previous examples

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

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

James Tam

Passing Parameters As Value Parameters

Previous examples

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

Pass a copy

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

James Tam

Passing Parameters As Value Parameters

Previous examples

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

Pass a copy

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

James Tam

Passing Parameters As Variable Parameters

Example coming up

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

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

James Tam

Passing Parameters As Variable Parameters

Example coming up

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

Pass variable

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

James Tam

Passing Parameters As Variable Parameters

Example coming up

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

Pass variable

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

James Tam

Procedure Definitions When Passing Parameters As Variable Parameters

Format:

```
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;
```

James Tam

Calling Procedures With Variable Parameters

Same as calling procedures with value parameters!

Format:

name (name of parameter 1, name of parameter 2...name of parameter n);

Example:

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

James Tam

Passing Variable Parameters: Putting It All Together

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

```
program financialStatments (input, output);

procedure getIncomeInformation (var grossSales      : real;
                                var costOfGoodsSold : real;
                                var expenses        : real);

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

James Tam

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 *)
```

James Tam

Passing Variable Parameters: Putting It All Together (3)

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

James Tam

Passing Variable Parameters: Putting It All Together (4)

```
procedure produceIncomeStatement;
var
    grossSales, grossIncome, costOfGoodsSold, expenses, netIncome :real;
begin
    getIncomeInformation(grossSales, costOfGoodsSold, expenses);
    tabulateIncome(grossSales, costOfGoodsSold, grossIncome, expenses, netIncome);
    displayIncomeStatement
        (grossSales, costOfGoodsSold, grossIncome, expenses, netIncome);
end; (* End of procedure produceIncomeStatement *)
```

James Tam

Passing Variable Parameters: Putting It All Together (5)

```
(* 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 *)
```

James Tam

Scope

It determines when a part of a program (Constant, variable, function, procedure) is available for use in that program.

e.g., variables or constants must first be declared before they can be referred to or used.

```
begin
  var num: integer;
  num := 10;
  :       :
end.
```

James Tam

Scope

It determines when a part of a program (Constant, variable, function, procedure) is available for use in that program.

e.g., variables or constants must first be declared before they can be referred to or used.

```
begin
  var num: integer;
  num := 10;
  :       :
end.
```

Declaration

Usage

James Tam

Scope

It determines when a part of a program (Constant, variable, function, procedure) is available for use in that program.

e.g., variables or constants must first be declared before they can be referred

to or used.

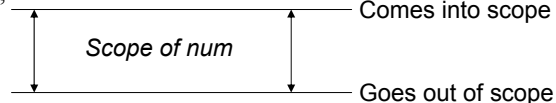
begin

```
var num: integer;
```

```
num := 10;
```

```
  :      :
```

```
end.
```



James Tam

Scope (2)

A complete version of this program can be found in Unix under:
`/home/231/examples/modules/scope1.p`

```
program scope1 (output);
```

```
const
```

```
  SIZE = 10;
```

```
var
```

```
  num : integer;
```

```
  ch  : char;
```

```
procedure proc1;
```

```
var
```

```
  x, y : real;
```

```
begin
```

```
  writeln('In proc1');
```

```
end;
```

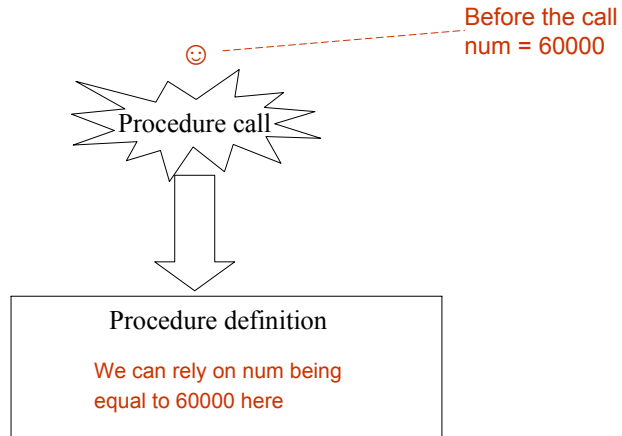
```
begin
```

```
end.
```

James Tam

Preconditions

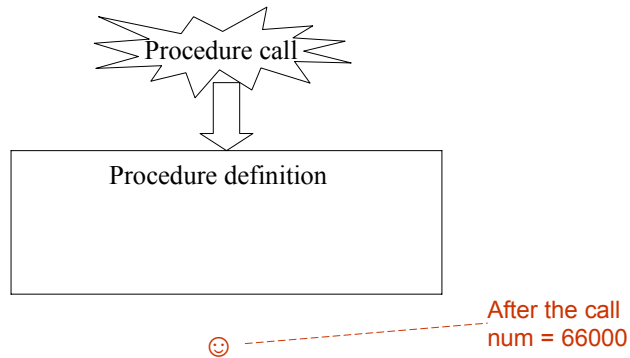
Describe what should be true before a statement is executed
e.g., What will be the value of a variable before a procedure call.



James Tam

Postconditions

Describe what should be true after a statement is executed
e.g., What will be the value of a variable after a procedure call.



James Tam

Preconditions And PostConditions

Relative: One procedure's postcondition can be another procedure's precondition

e.g.,

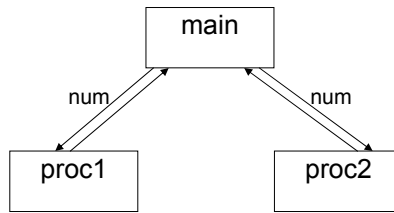
begin

var num: integer;

proc1(num);

proc2(num);

end.



James Tam

Preconditions And PostConditions

Assertions: Both make assumptions about what is the state of (a part of) the program at a certain point.

☺ ----- Before the call
num2 = 2

```
procedure proc1 (num2: integer);  
begin  
  writeln('At start of proc1: num2=', num2);  
  writeln('At end of proc1: num2=', num2);  
end;
```

```
procedure proc2 (var num2: integer);  
begin  
  writeln('At start of proc2: num2=', num2);  
  num2 := 20;  
  writeln('At end of proc2: num2=', num2);  
end;
```

James Tam

Preconditions And Postconditions: An Example

A complete version of this program can be found in Unix under:
/home/231/examples/modules/conditions.p

```
program conditions (output);
var
  num1 : integer;

procedure proc1 (num2: integer);
begin
  writeln('At start of proc1: num2=', num2);
  writeln('At end of proc1: num2=', num2);
end; (* Proc 1 *)

procedure proc2 (var num2: integer);
begin
  writeln('At start of proc2: num2=', num2);
  num2 := 20;
  writeln('At end of proc2: num2=', num2);
end; (* Proc 2 *)
```

James Tam

Preconditions And Postconditions: An Example (2)

```
procedure proc3;
begin
  writeln('At start of proc3: num1=', num1);
  num1 := 10;
  writeln('At end of proc3: num1=', num1);
end; (* Proc 3 *)

begin
  var num2 : integer;
  num1 := 1;
  num2 := 2;
  proc1(num2);
  writeln;
  proc2(num2);
  writeln;
  proc3;
  writeln;
end. (* Main Program *)
```

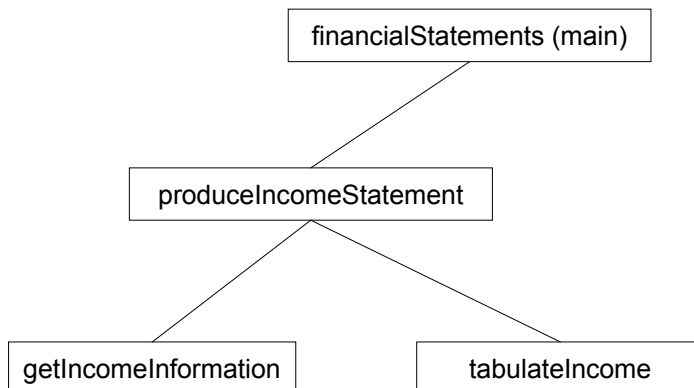
James Tam

Testing Modules

- Making sure the function or procedure does what it is supposed to do e.g., checking if calculations are correct.
- Ties into the top-down approach to design
 - 1) Outline the structure of the program (empty modules)
 - 2) As modules are implemented test each one as appropriate
 - 3) Fix and bugs and add the working module to the program.

James Tam

Outline Of The Financial Program



James Tam

Code Skeleton For The Financial Program

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

begin
end;

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

James Tam

Code Skeleton For The Financial Program (2)

```
begin
  produceIncomeStatement;
end. (* Main program *)
```

James Tam

Implementation Of Procedure “tabulateIncome”

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

James Tam

Testing Procedure “tabulateIncome”

```
procedure produceIncomeStatement;
var
  grossSales      : real;
  grossIncome     : real;
  costOfGoodsSold : real;
  expenses        : real;
  netIncome       : real;
begin
  tabulateIncome(0,0,grossIncome,0,netIncome);
  writeln('Testing "produceIncomeStatment" with all values equal to zero');
  if (grossIncome = 0) AND (netIncome = 0) then
    writeln('Test condition okay')
  else
    begin
      if NOT (grossIncome = 0) then
        writeln('Problem with calculating gross income');
      if NOT (netIncome = 0) then
        writeln('Problem with calculating net income');
    end;
  end;
  writeln;
```

James Tam

Testing Procedure “tabulateIncome” (2)

```
tabulateIncome(100, 75, grossIncome, 10, netIncome);
writeln("Testing "produceIncomeStatment" with non-zero values");
if (grossIncome = 25) AND (netIncome = 15) then
    writeln("Test condition okay")
else
begin
    if NOT (grossIncome = 25) then
        writeln('Problem with calculating gross income');
    if NOT (netIncome = 15) then
        writeln('Problem with calculating net income');
end;
writeln;
end; (* Procedure tabulateIncome *)
```

James Tam

You Should Now Know

How to break a programming problem down into modules

What is the difference between a procedure and a function

What is the difference between a value parameter and variable parameter

How to define and call program modules (procedures and functions)

Variables and scope

- What is a local variable
- What is a global variable
- What is the scope of a procedure or function

What are preconditions and postconditions

How to test functions and procedures

James Tam