Scope & Testing Modules

CSCE 110

Influenced by material developed by James Tam & Jennifer Welch
The parts of a program where an item (constant, variable, function, procedure) is declared for use.

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

```plaintext
begin
  var num: integer;
  num := 10;
  :
  :
end.
```
The parts of a program where an item (constant, variable, function, procedure) ________________ for use.

  e.g., variables or constants must first be _______ before they can be ________________.

begin
  var num: integer;
  num := 10;
  ::
end.
The parts of a program where an item (constant, variable, function, procedure) \( \text{________________________} \) for use.

e.g., variables or constants must first be \( \text{_______} \) before they can be \( \text{________________________} \).

begin

var num: integer;
num := 10;
:
:

end.
Scope

Scope: After declaration, the item (constant, variable, function or procedure) can be accessed ______________ in the program.

program exampleProgram;

begin
end.

Declarations here have __________ scope

procedure proc;
var
    \textbf{Declarations with __________ scope}

begin
    \textbf{Declarations with __________ scope}
end;

begin
    \textbf{Declarations with __________ scope}
end.

J. Michael Moore
_________ Scope

When an identifier (constant, variable, function or procedure) is encountered the compiler will:

1. Check in the _______ scope
2. Check the _______ scope if no matches can be found _________

For example:

program exampleProgram;
var
  num : integer;

procedure proc;
var
  num : integer;
begin
  num := 1;
end;

begin
  :
end.
First ______________ Example

scope1.pas

program scope1 (output);
const
  SIZE = 10;
var
  num1 : integer;
  ch : char;
procedure proc1;
var
  num2 : real;
  num3 : real;
begin
  writeln('In proc1');
  writeln('In proc1');
end;
begin
end.

J. Michael Moore
The Use Of _______ Variables

- Remember _______ variables can be _______ or _______ _______ in the program after their declaration.
- This results in:
  - _______ modules – changes in one module may effect other modules
  - Programs that are more difficult to _______ and _______
- Unless there is a _______ reason, variables should be declared _______ and passed as a _______ where ever it is needed.
Second Scoping Example

scope2.pas

program scope2 (output);
var
  num : integer;
  ch  : char;
procedure proc1;
var
  ch : char;
begin
  ch := 'b';
  writeln('In proc1');
  writeln ('num=', num, ' ch=', ch);
  writeln;
end;
procedure proc2(numProc2: integer);
var
  num : integer;
begin
  writeln(‘In proc2’);
  num := 2;
  numProc2 := 20;
  writeln (‘num=’, num, ‘ ch=’, ch, ‘ numProc2=’, numProc2);
  writeln;
  proc1;
end;
begin
    var numLocal : integer;
    num := 1;
    ch := 'a';
    numLocal := 10;
    writeln;
    proc2(numLocal);
    writeln('In main program');
    writeln('num=', num, ' ch=', ch, ' numLocal=', numLocal);
end.
Going back a bit…Testing Decision Making Constructs

• Make sure that the body of each decision making construct executes when it should.

• Test:
  1) Obviously true cases
  2) Obviously false cases
  3) Boundary cases
Developing Modules

• This is an integral part of the top down approach to designing programs.

• Recall with the top down approach:
  1. Start with _____ level idea of solution, and __________________________ it.
     (i.e., specify what modules that the program must consist of but don’t write the code for them yet).

```
wisdom (main)
```

```
getAge
calculateAgeModifier
```
• It’s an outline of a module with the bare ___________ of code needed to __________
  • required __________
  • module _______
  • a ______________ to define the body
  • __________________ and ______________ may or may not be included in the ______________
Develop & Test Modules

2. Implement the body of each module, _________________________.

---

```
# Get age
procedure getAge (var age : integer);
begin
  write('How old are you (1-113 years)? ');
  readln(age);
end;
```
program wisdom (input, output);

procedure getAge (var age : integer);
begin
end;

function calculateWisdomNumber (age : integer):
    integer;
begin
    calculateWisdomNumber := 0;
end;

begin
    var age : integer;
    var wisdomNumber : integer;
    getAge (age);
    wisdomNumber := calculateWisdomNumber(age);
end.
Implementation Of Procedure "getAge"

procedure getAge (var age : integer);
begin
    write('How old are you (1-113 years)? ');
    readln(age);
end;
Testing Procedure “getAge”

Testing simply involves checking the input:

(* In the main procedure *)
getAge(age);
writeln('After getAge, age=', age);
Implementing Function “calculateWisdomNumber”

function calculateWisdomNumber (age : integer): integer;
begin
  if (age >= 1) AND (age <= 30) then
    calculateWisdomNumber := age * 1
  else if (age >= 31) AND (age <= 65) then
    calculateWisdomNumber := age * 2
  else if (age > 65) then
    calculateWisdomNumber := age * 3;
  else
    calculateWisdomNumber := 0;
end;
Testing Function “calculateWisdomNumber”

(* Testing calculateWisdomNumber in the main procedure *)

wisdomNumber := calculateWisdomNumber(-5);
if (wisdomNumber <> 0) then
    writeln('Error if age < 1');

wisdomNumber := calculateWisdomNumber(1); (* boundary *)
if (wisdomNumber <> 1) then
    writeln('Error if age = 1');

wisdomNumber := calculateWisdomNumber(21);
if (wisdomNumber <> 21) then
    writeln('Error if age = 21');

wisdomNumber := calculateWisdomNumber(30); (* boundary *)
if (wisdomNumber <> 30) then
    writeln('Error if age = 30');
Testing Function “calculateWisdomNumber” (2)

\[
\text{wisdomNumber} := \text{calculateWisdomNumber}(31); (* \text{boundary} *)
\]
if (wisdomNumber <> 62) then
  writeln('Error if age = 31');

\[
\text{wisdomNumber} := \text{calculateWisdomNumber}(55);
\]
if (wisdomNumber <> 110) then
  writeln('Error if age 30 - 65');

\[
\text{wisdomNumber} := \text{calculateWisdomNumber}(65); (* \text{boundary} *)
\]
if (wisdomNumber <> 130) then
  writeln('Error if age = 65');

\[
\text{wisdomNumber} := \text{calculateWisdomNumber}(90);
\]
if (wisdomNumber <> 270) then
  writeln('Error if age > 65');
Recall This Design

Change program (main)

inputAmount

computeChange

outputCoins

computeQuarters

computeDimes

computePennies

amount

quarters

dimes

pennies

amount

amountLeft

amountLeft

amountLeft

amountLeft

amountLeft

amountLeft

amountLeft

amountLeft
Designing & Testing these Modules

1. Change program (main)
   - \( \uparrow \text{pennies} \uparrow \text{dimes} \uparrow \text{quarters} \)
   - \( \downarrow \text{amount} \)

2. inputAmount
   - \( \uparrow \text{amount} \)

3. computeChange
   - \( \downarrow \text{amount} \)
   - \( \uparrow \text{quarters} \uparrow \text{dimes} \uparrow \text{pennies} \)

4. outputCoins
   - \( \downarrow \text{pennies} \downarrow \text{dimes} \downarrow \text{quarters} \)

5. computeQuarters
   - \( \uparrow \text{quarters} \uparrow \text{amountLeft} \)
   - \( \downarrow \text{amount} \)

6. computeDimes
   - \( \uparrow \text{dimes} \uparrow \text{amountLeft} \)
   - \( \downarrow \text{amountLeft} \)

7. computePennies
   - \( \uparrow \text{pennies} \uparrow \text{amountLeft} \)
   - \( \downarrow \text{amountLeft} \)

J. Michael Moore
Designing & Testing these Modules

Alternatively

1. Change program (main)

2. inputAmount
   - amount

3. computeChange
   - amount
   - quarters
   - dimes
   - pennies

4. computeQuarters
   - amount
   - amountLeft

5. computeDimes
   - amountLeft
   - amountLeft

6. computePennies
   - amountLeft
   - amountLeft

7. outputCoins
   - pennies
   - dimes
   - quarters

J. Michael Moore