Software Testing
Unit 1

MR. C. R. BELAVI
DEPT. OF CSE, HSIT, NIDASOSHI

Content

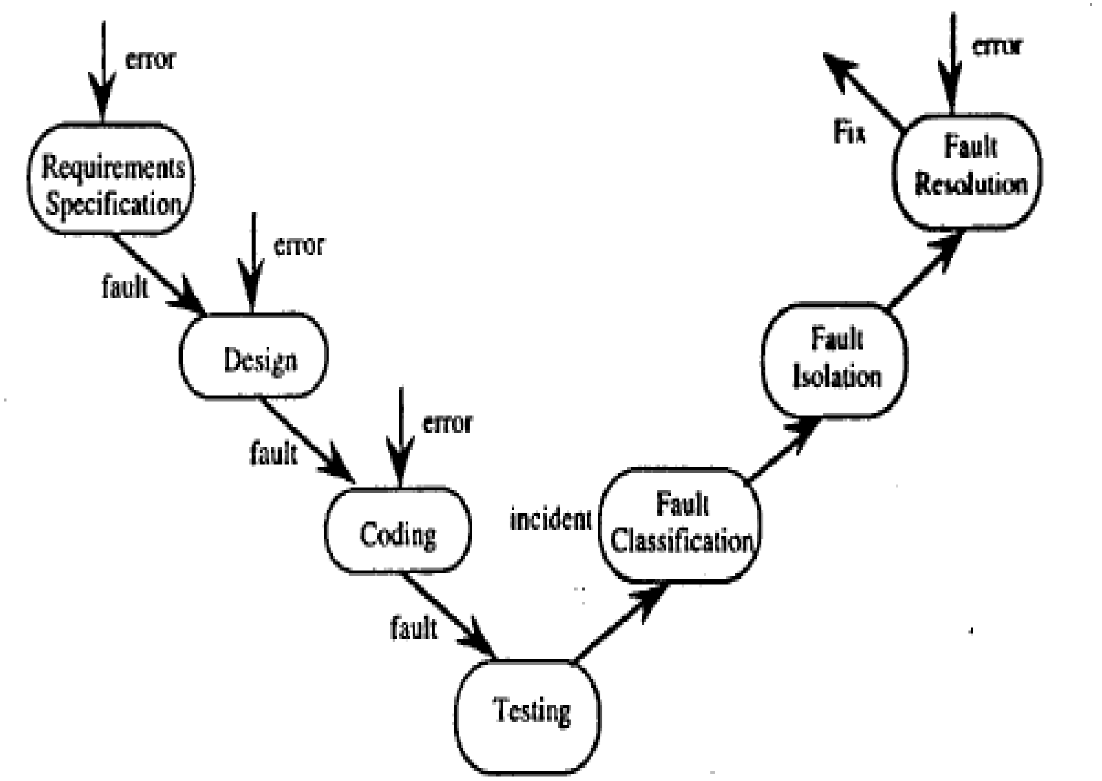
- **A perspective on Testing**
  - Basic Definition
  - Test Cases
  - Insights from a Venn Diagram
  - Identifying Test Cases
    - *Functional Testing*
    - *Structural Testing*
  - Error & Fault Taxonomies
  - Level of Testing

Why Do we Test.?

- To make a judgment about quality or acceptability.
- Discover Problems

Basic Definitions

- Error(mistake): mistake while coding-bug
- Fault(defect): Result of an error
  - Fault of omission
  - Fault of commission
- Failure: A failure occurs when a Fault executes.
- Incident: Alerts user occurrence of a Failure
- Test: concerned with errors, faults, failures, incident
- Test Case: have identity & is associated with a program behavior. Has i/p & o/p

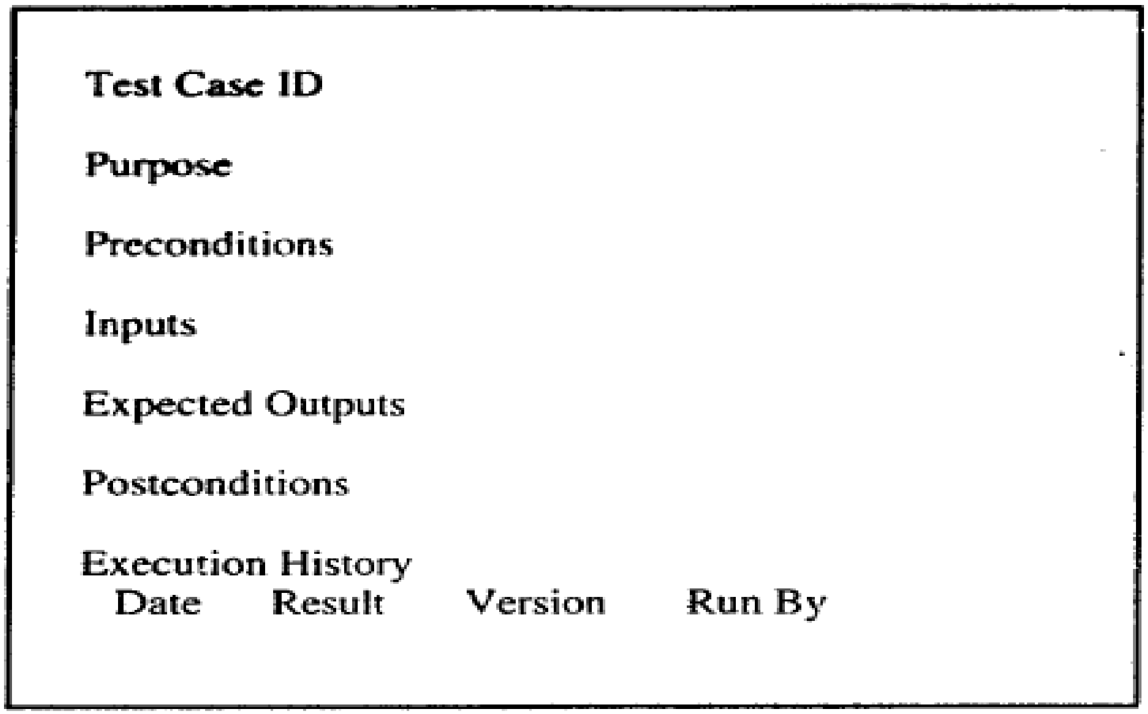


A testing life cycle.

Process of testing

- Test planning
- Test case development
- Running test cases
- Evaluating test results

Test Cases



Test Case ID

Purpose

Preconditions

Inputs

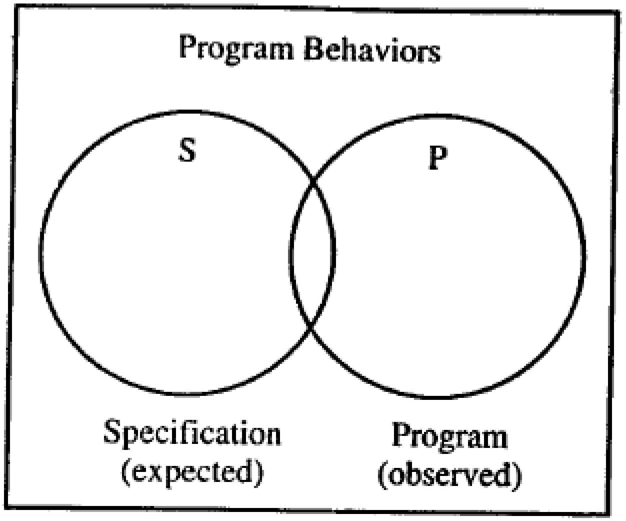
Expected Outputs

Postconditions

Execution History
Date Result Version Run By

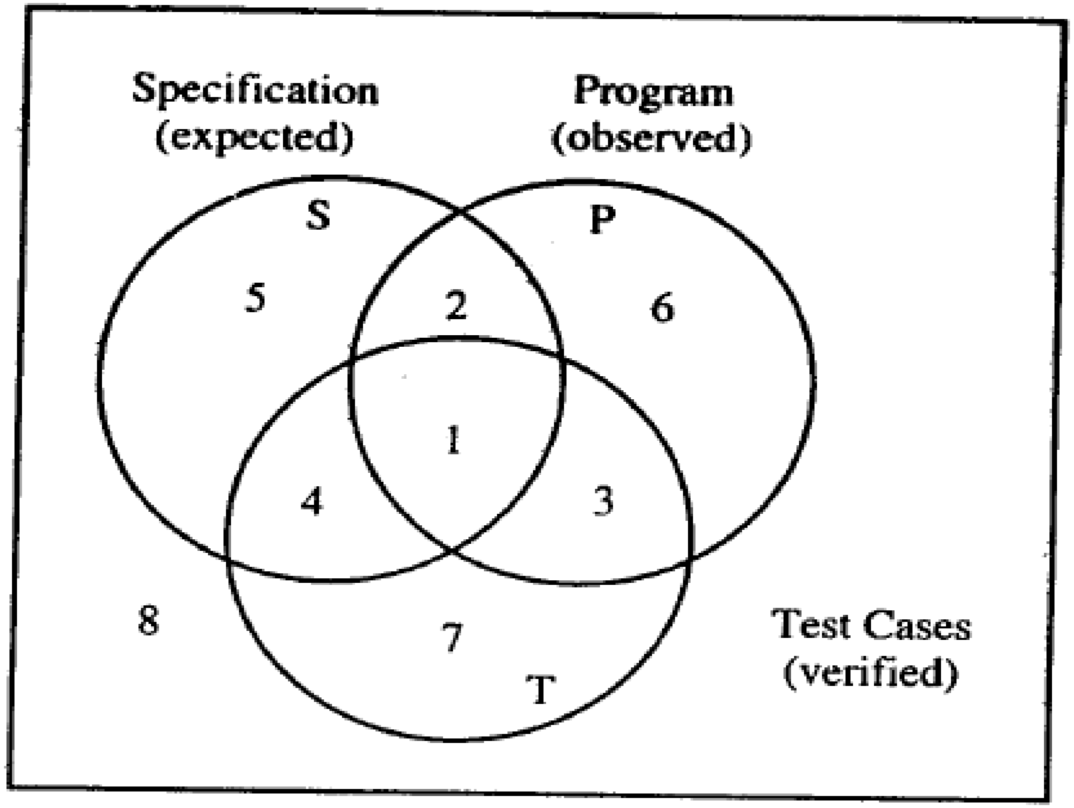
Typical test case information.

Insights from a Venn Diagram



Specified and implemented program behaviors.

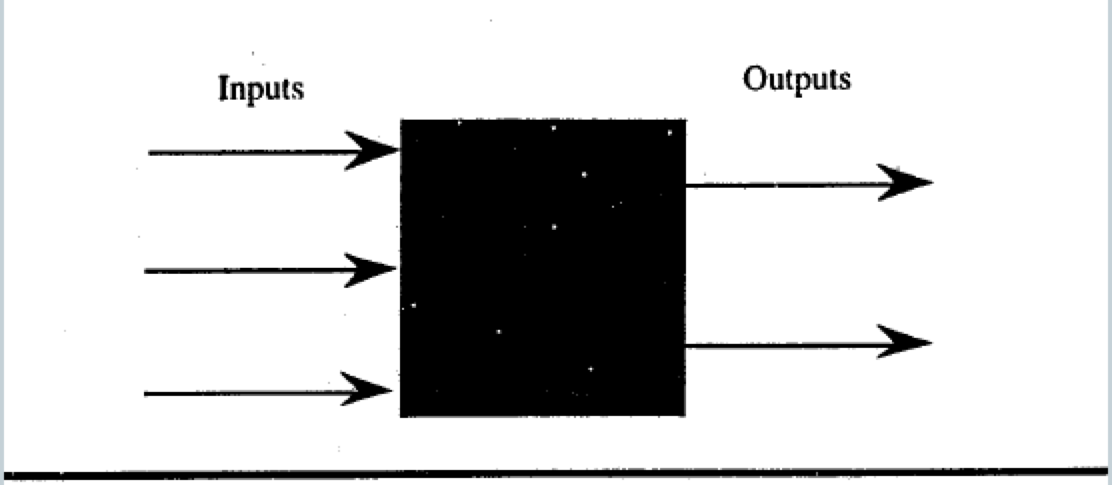
Cont.,



Specified, implemented, and tested behaviors.

Identifying Test Cases

- **Functional Testing(Black Box Testing):** implementation of Black box is not known.
- Function of black box is understood by i/p & o/p.

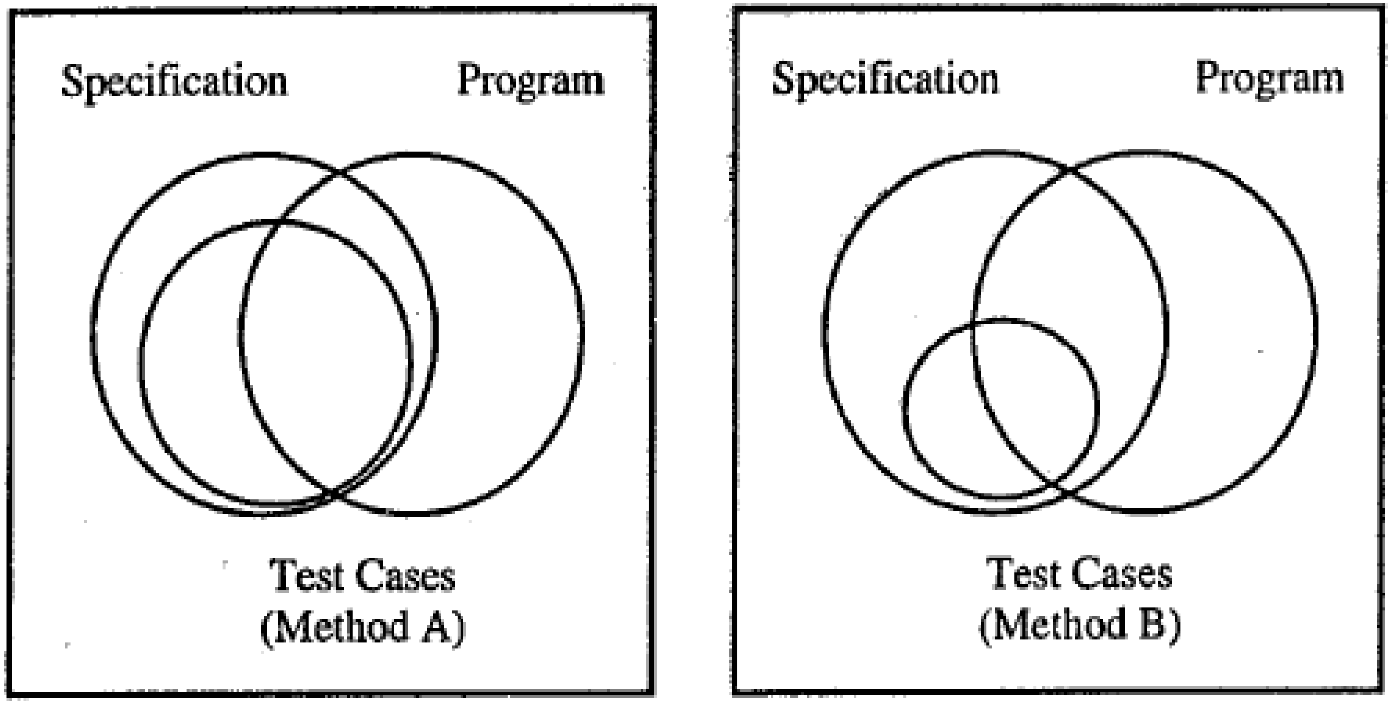


An engineer's black box.

Functional Testing

- Advantages
  - Independent of how the software is implemented.
  - If implementation change test cases are still useful
  - Test case development can occur in parallel with the implementation.
- Disadvantage:
  - Redundancies may exist among test cases
  - Possibility of gaps of untested software.

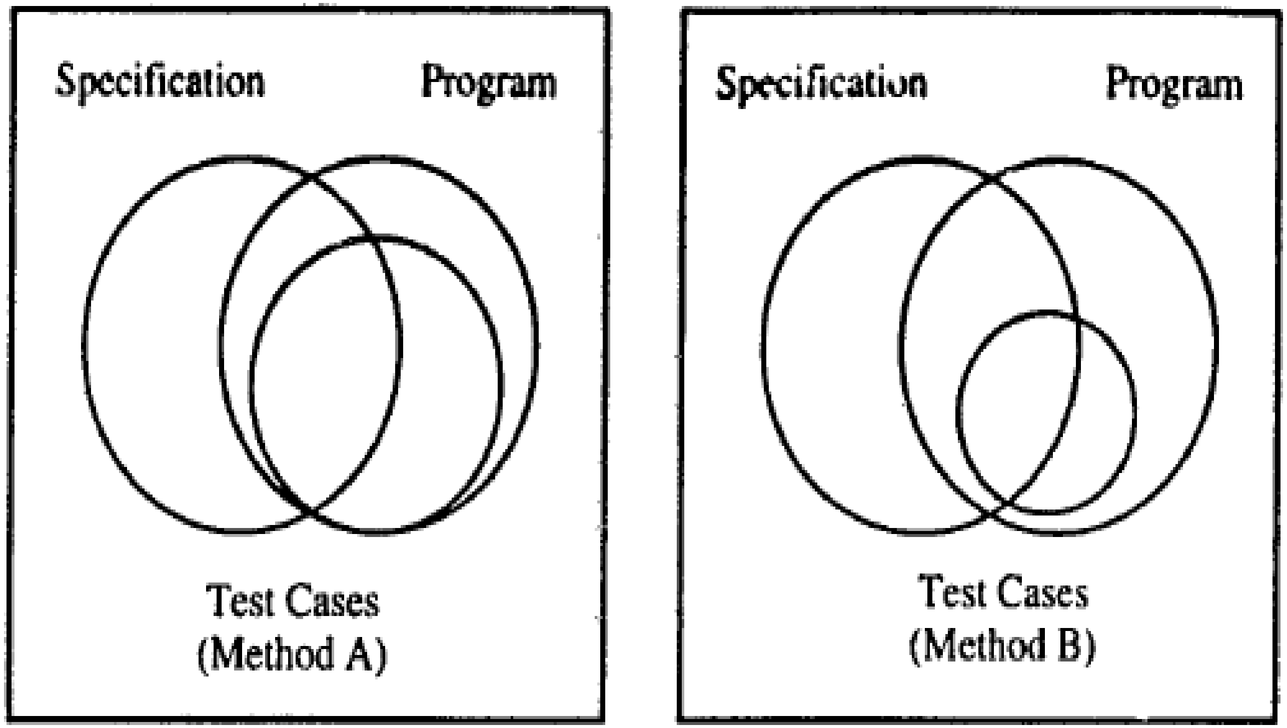
Conti.,



Comparing functional test case identification methods.

Structural Testing

- Also called white box testing(even clear box Testing)
- Implementation (of the Black box) is known & used to identify test cases.



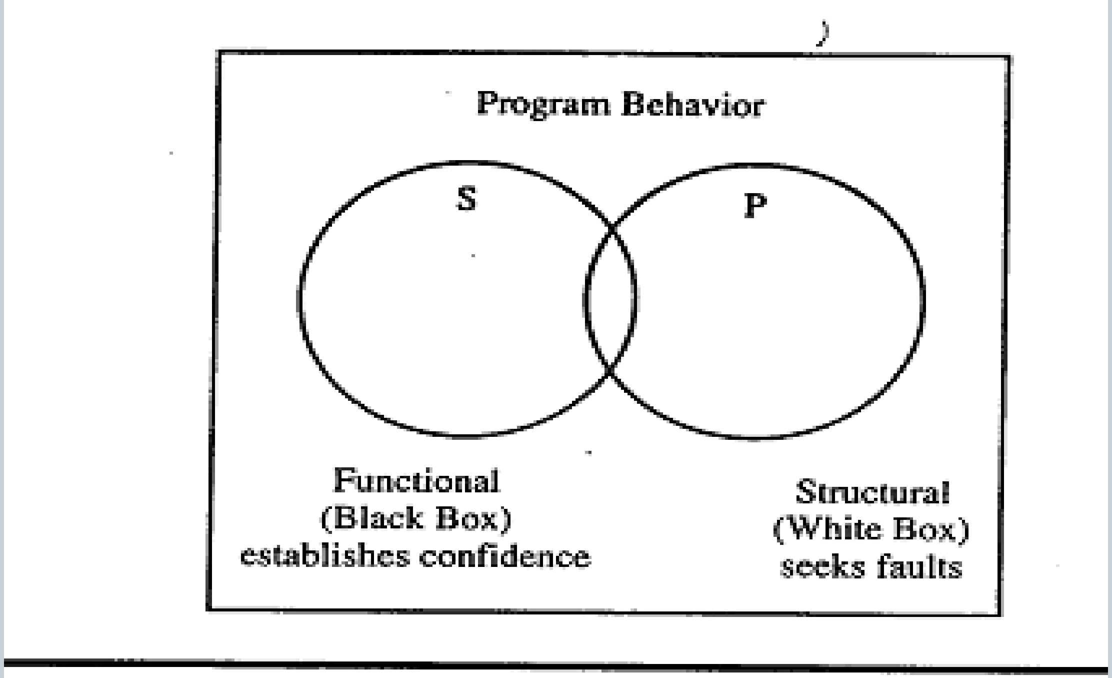
Comparing structural test case identification methods.

The functional VS Structural Debate

- Goals of both approach is to *identify test cases.*
- Functional testing uses only the specification to identify test cases.
- Structural testing uses the programs source code(implementation) as the basis of test case identification.

Cont.,

- When functional test cases are executed in combination with structural test coverage metrics twin problems redundancies & gaps faced by functional testing can be recognized & resolved.



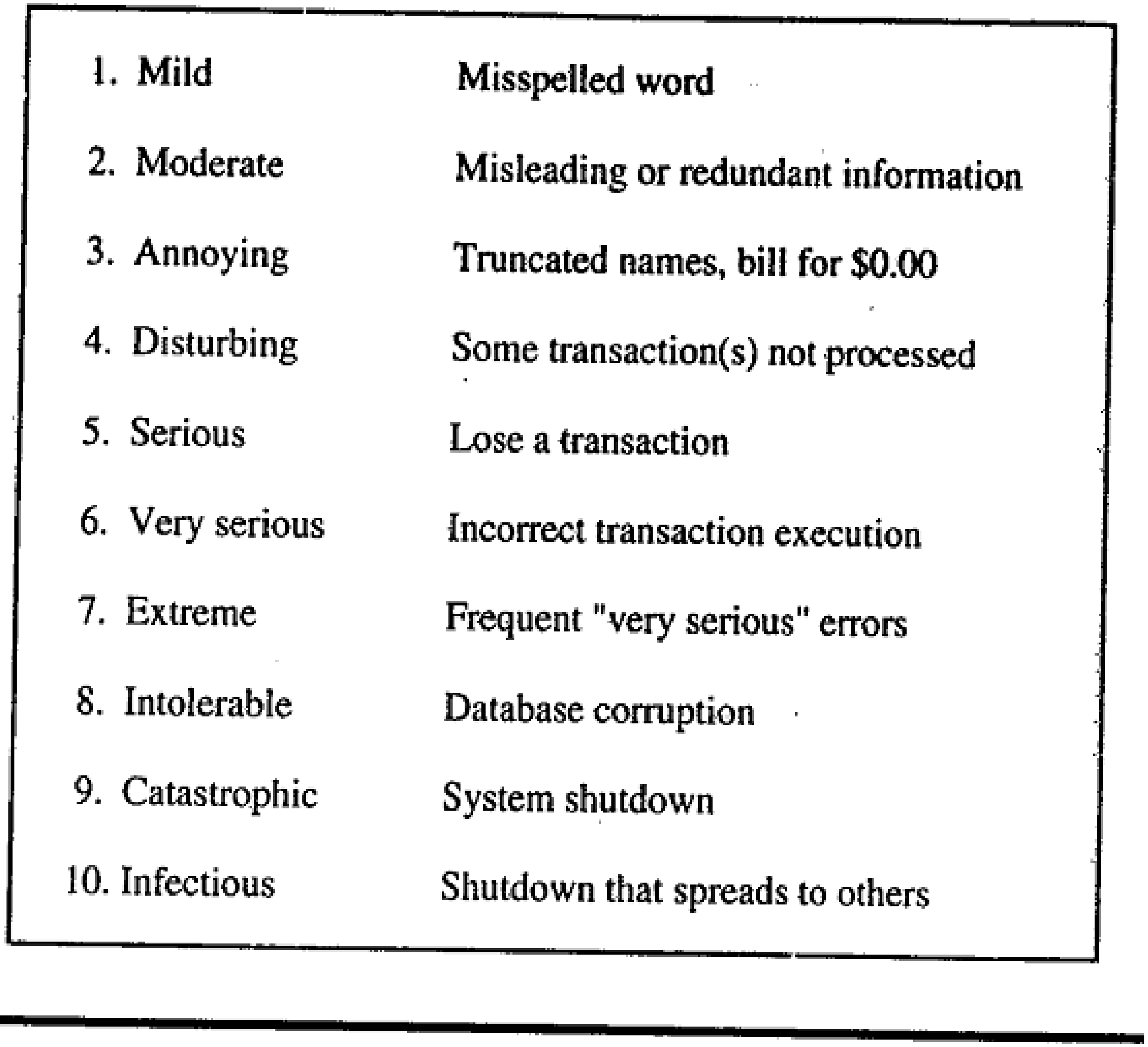
Sources of test cases.

Testing as a craft

- When we know what kind of error we are prone to make
- If we know what kind of faults are likely to reside in software to be tested.
- We can use this to employ more appropriate ***test case identification methods.***
- At this point testing really becomes a craft.

Error & Fault Taxonomies

- Definition of error & fault hinge on the distinction between process & product
- **Process**-refer to how we do something.
- **Product**-end result of a process.
- SQA- tries to improve the product by improving the process.
- Testing is clearly more product oriented.
- Faults can be classified in several ways



1. Mild	Misspelled word
2. Moderate	Misleading or redundant information
3. Annoying	Truncated names, bill for $0.00
4. Disturbing	Some transaction(s) not processed
5. Serious	Lose a transaction
6. Very serious	Incorrect transaction execution
7. Extreme	Frequent "very serious" errors
8. Intolerable	Database corruption
9. Catastrophic	System shutdown
10. Infectious	Shutdown that spreads to others

Faults classified by severity.

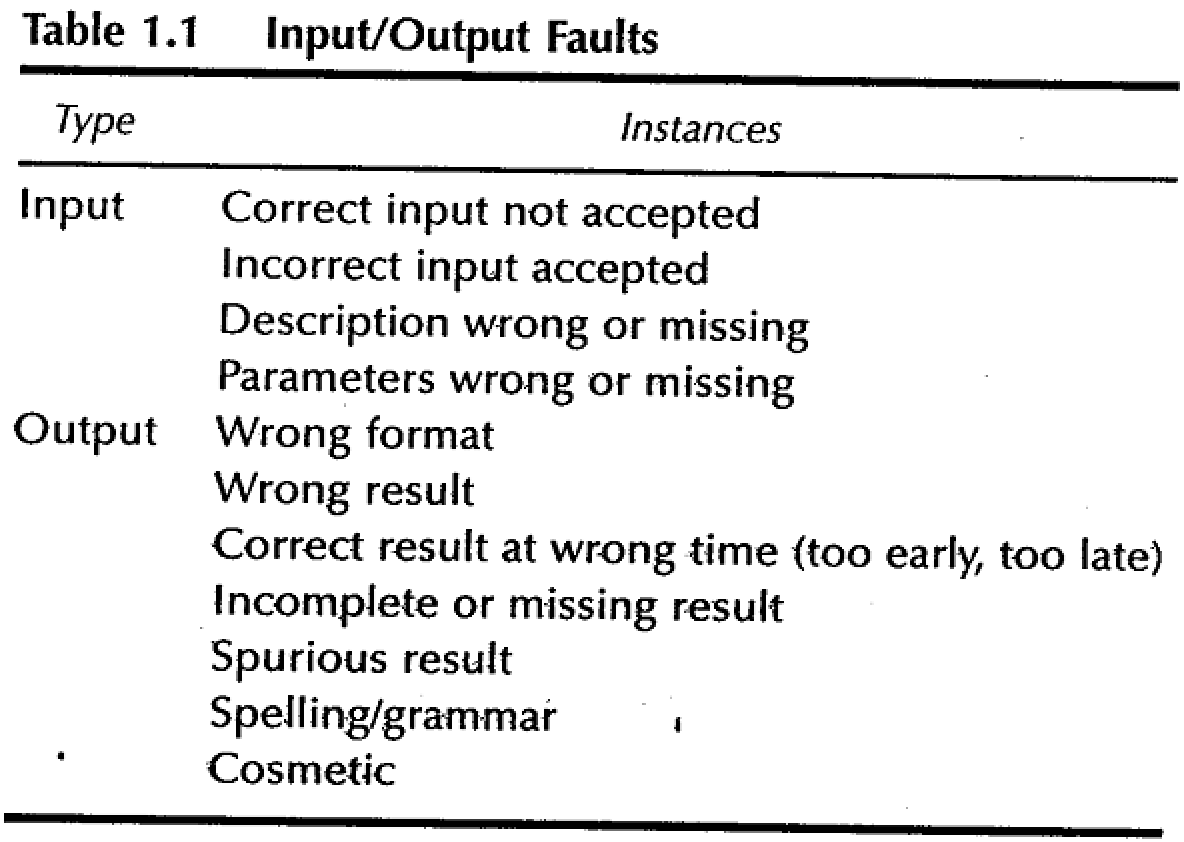


Table 1.1 Input/Output Faults

Type	*Instances*
Input	Correct input not accepted
	Incorrect input accepted
	Description wrong or missing
	Parameters wrong or missing
Output	Wrong format
	Wrong result
	Correct result at wrong time (too early, too late)
	Incomplete or missing result
	Spurious result
	Spelling/grammar
	Cosmetic

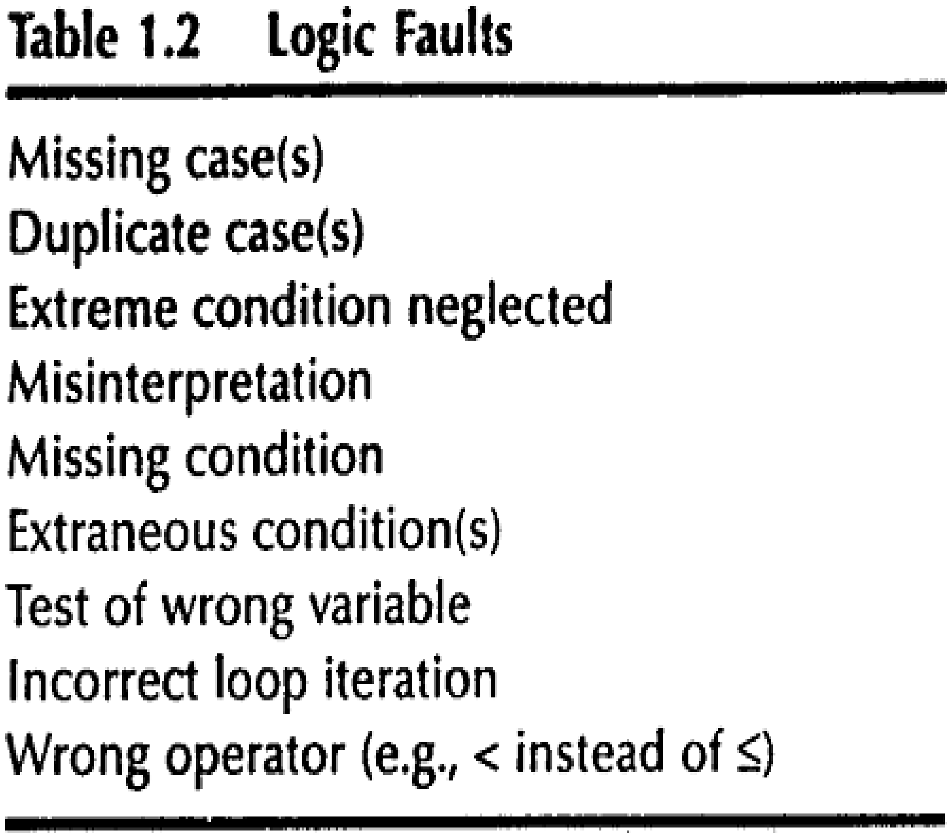


Table 1.2 Logic Faults

Missing case(s)
Duplicate case(s)
Extreme condition neglected
Misinterpretation
Missing condition
Extraneous condition(s)
Test of wrong variable
Incorrect loop iteration
Wrong operator (e.g., < instead of ≤)

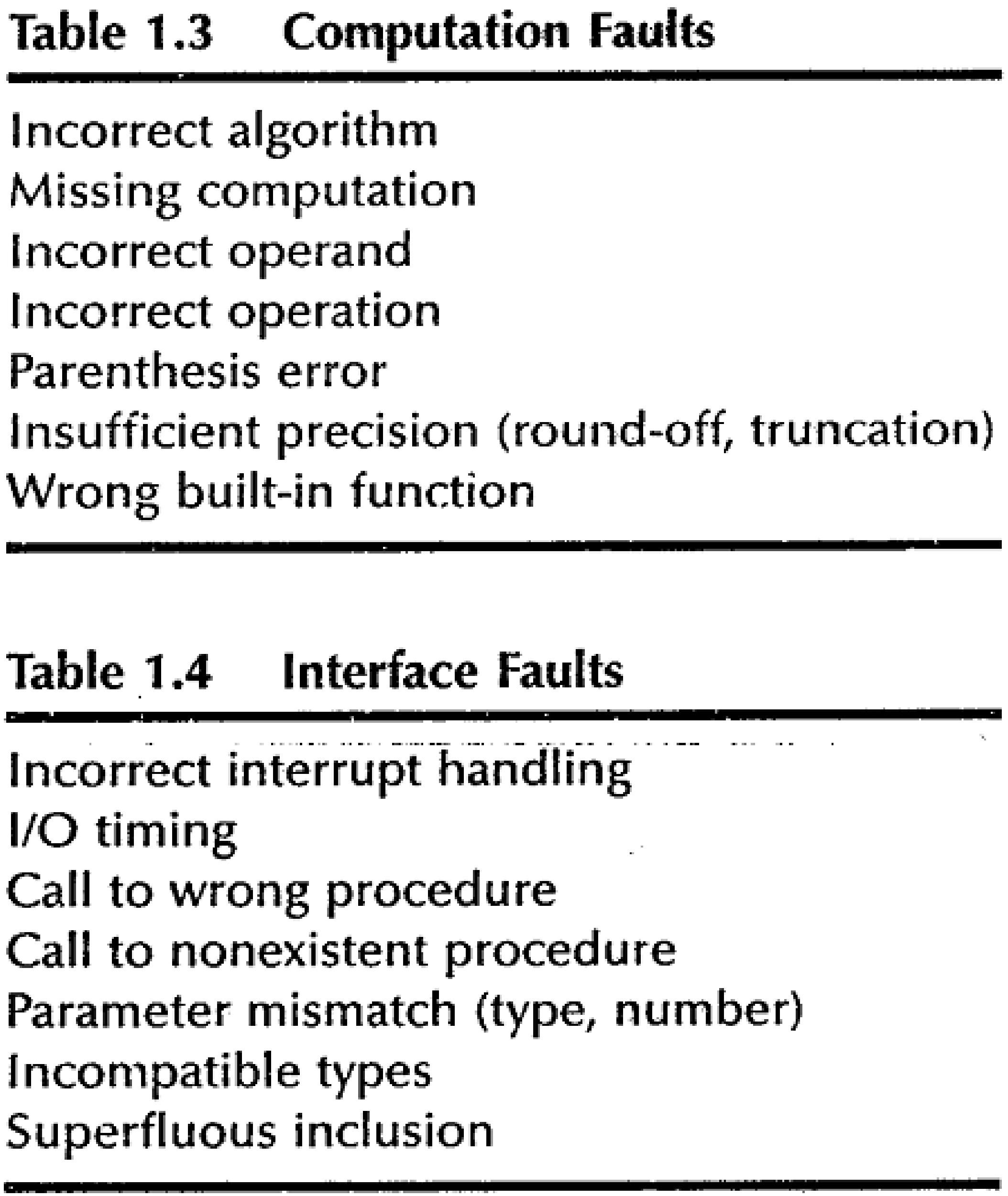


Table 1.3 Computation Faults

Incorrect algorithm
Missing computation
Incorrect operand
Incorrect operation
Parenthesis error
Insufficient precision (round-off, truncation)
Wrong built-in function

Table 1.4 Interface Faults

Incorrect interrupt handling
I/O timing
Call to wrong procedure
Call to nonexistent procedure
Parameter mismatch (type, number)
Incompatible types
Superfluous inclusion

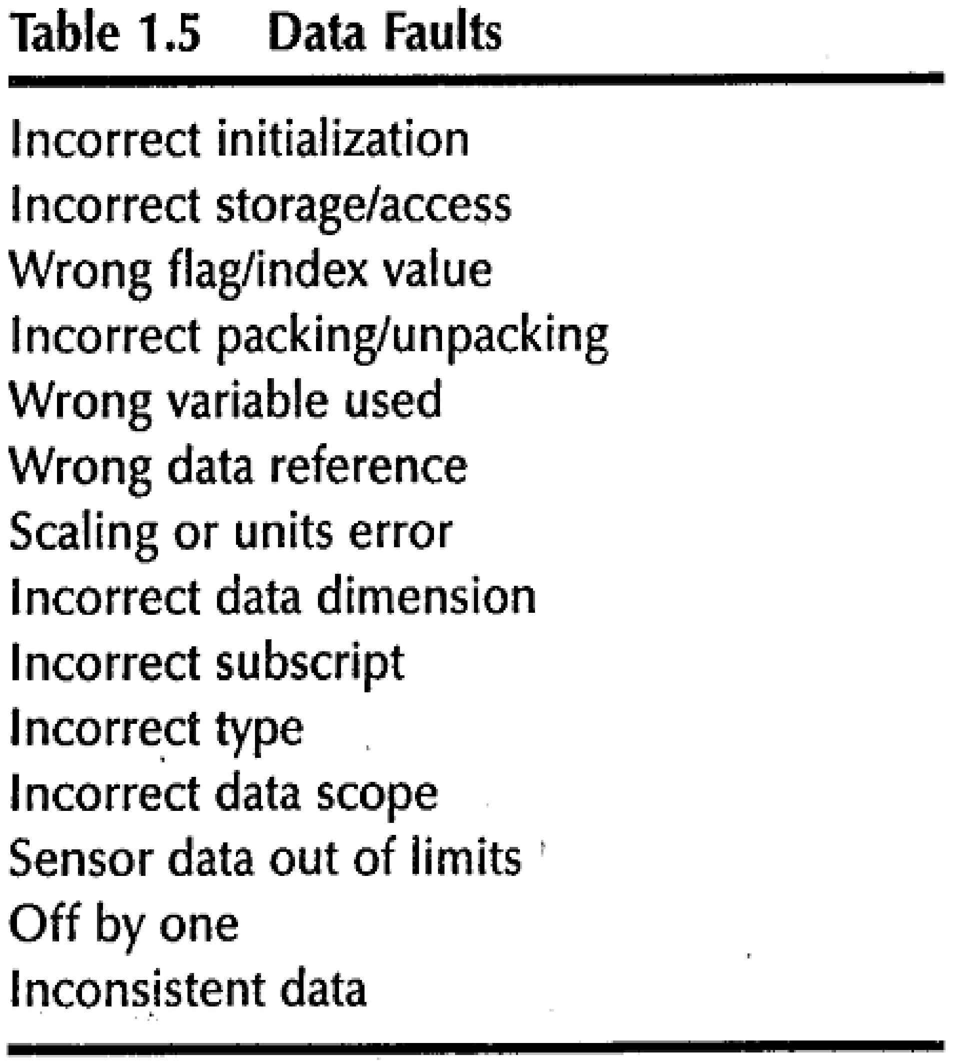
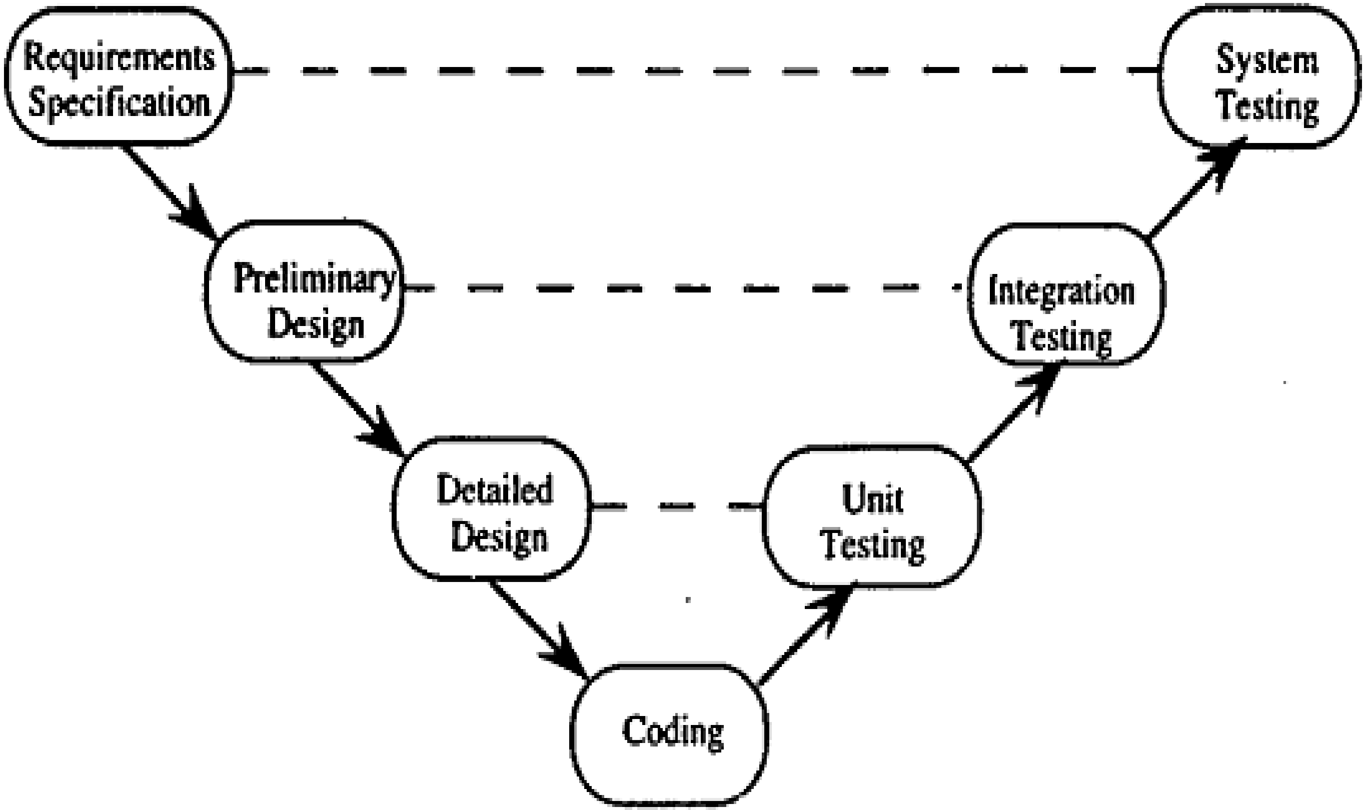


Table 1.5 Data Faults

Incorrect initialization
Incorrect storage/access
Wrong flag/index value
Incorrect packing/unpacking
Wrong variable used
Wrong data reference
Scaling or units error
Incorrect data dimension
Incorrect subscript
Incorrect type
Incorrect data scope
Sensor data out of limits
Off by one
Inconsistent data

Levels of Testing

- Levels of testing echo the levels of abstraction found in the waterfall model of the SDLC.
- In functional testing 3 levels of definition *(specification, preliminary design, detailed design)* correspond directly to 3 levels of testing –*system, integration & unit testing*.



Levels of abstraction and testing in the Waterfall Model.

Examples

- Three examples to illustrate various unit Testing methods.
- *These examples raise most of the issues that testing craftsperson's will encounter at the unit level.*
- For the purpose of structural testing, pseudocode implementation of 3 unit-level eg. are given.
  - The triangle problem
  - NextDate
  - Commission problem

Generalized Psuedocode

- Pseudocode provides a *"language neutral"* way to express program source code.
- Pseudocode given here is based on visual basic.

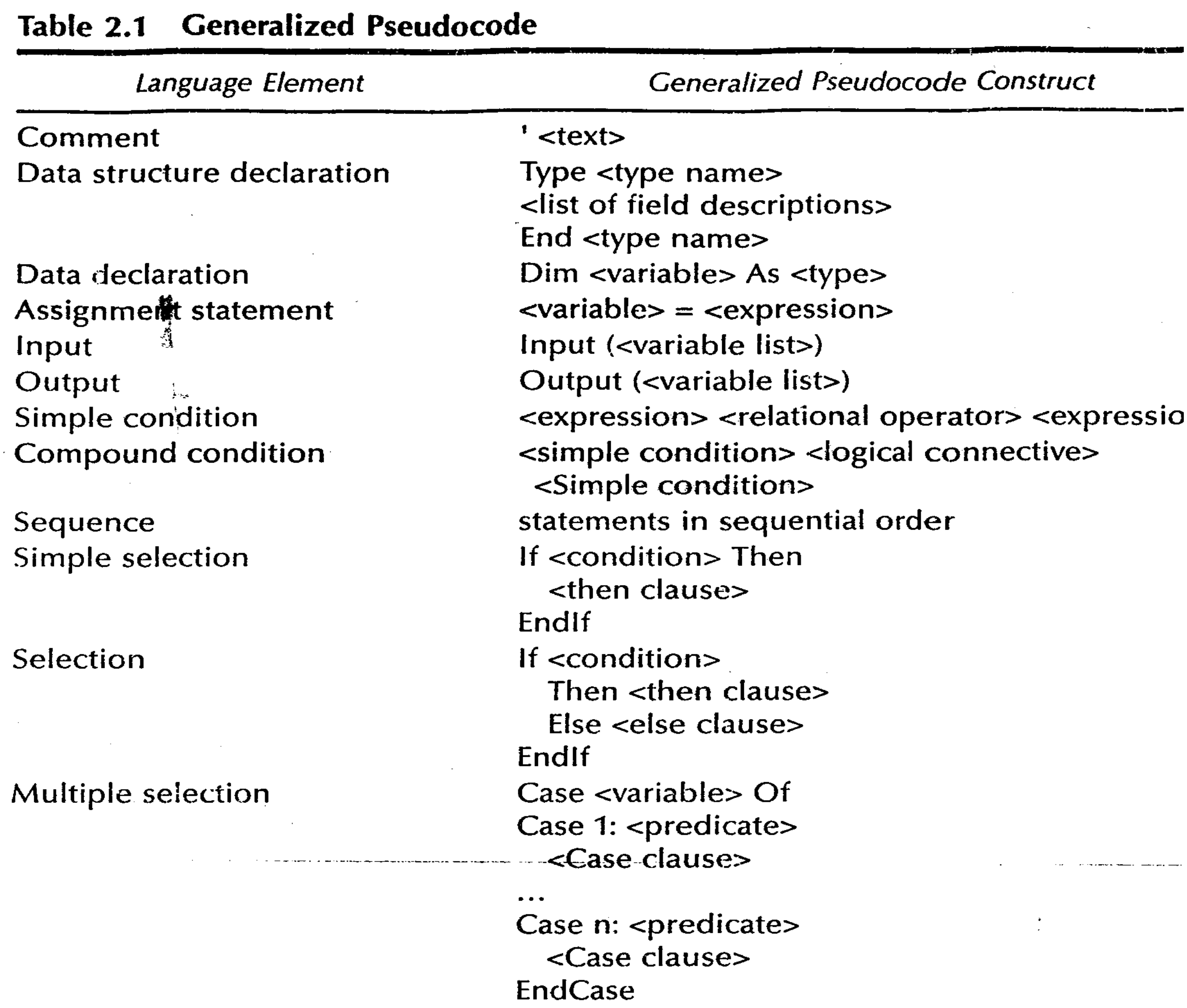
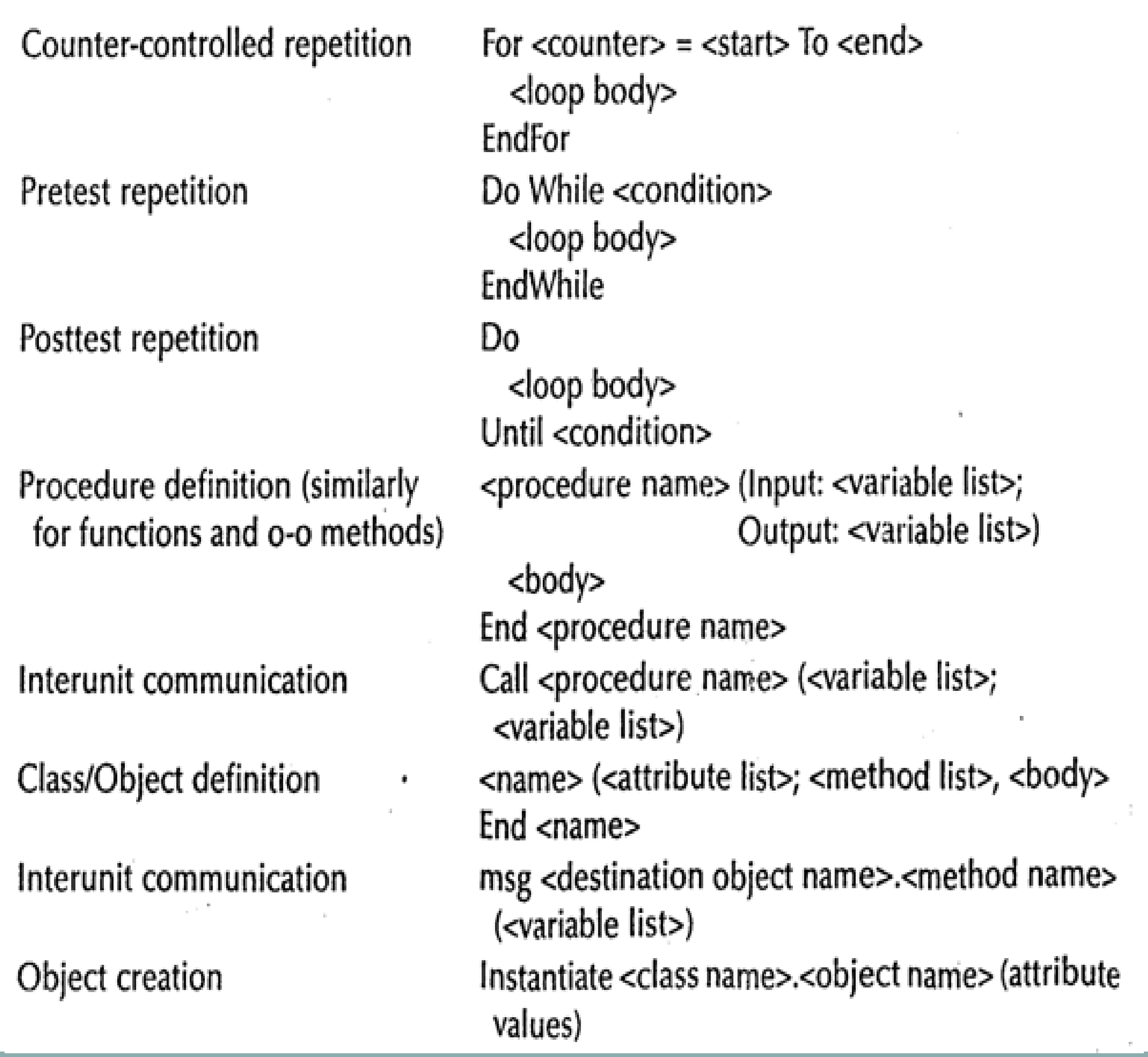


Table 2.1 Generalized Pseudocode

Language Element	*Generalized Pseudocode Construct*
Comment	' <text>
Data structure declaration	Type <type name> <list of field descriptions> End <type name>
Data declaration	Dim <variable> As <type>
Assignment statement	<variable> = <expression>
Input	Input (<variable list>)
Output	Output (<variable list>)
Simple condition	<expression> <relational operator> <expressio
Compound condition	<simple condition> <logical connective> <Simple condition>
Sequence	statements in sequential order
Simple selection	If <condition> Then <then clause> EndIf
Selection	If <condition> Then <then clause> Else <else clause> EndIf
Multiple selection	Case <variable> Of Case 1: <predicate> <Case clause> ... Case n: <predicate> <Case clause> EndCase



Counter-controlled repetition	For <counter> = <start> To <end> <loop body> EndFor
Pretest repetition	Do While <condition> <loop body> EndWhile
Posttest repetition	Do <loop body> Until <condition>
Procedure definition (similarly for functions and o-o methods)	<procedure name> (Input: <variable list>; Output: <variable list>) <body> End <procedure name>
Interunit communication	Call <procedure name> (<variable list>; <variable list>)
Class/Object definition	<name> (<attribute list>; <method list>, <body> End <name>
Interunit communication	msg <destination object name>.<method name> (<variable list>)
Object creation	Instantiate <class name>.<object name> (attribute values)

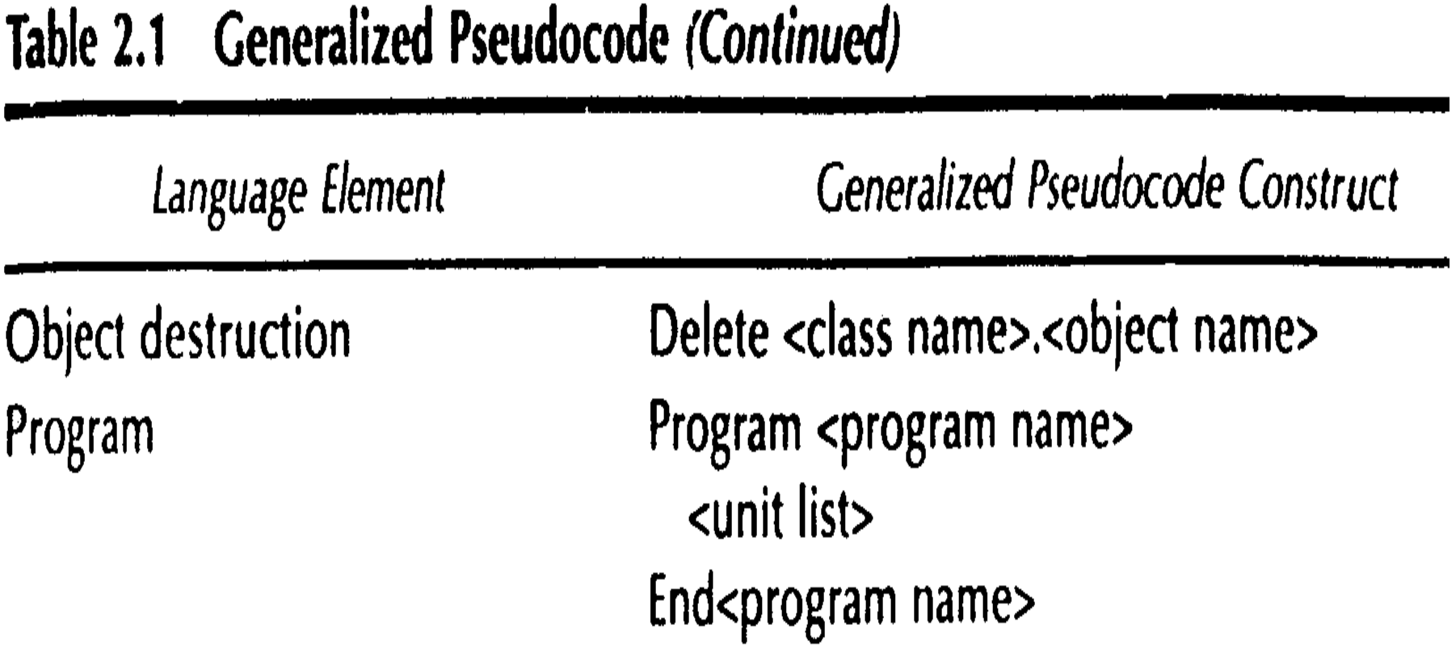


Table 2.1 Generalized Pseudocode *(Continued)*

Language Element	*Generalized Pseudocode Construct*
Object destruction	Delete <class name>.<object name>
Program	Program <program name> <unit list> End<program name>

Triangle Problem

- Problem statement
- Simple version: The triangle program accepts 3 integers a, b, c as input to be sides of a triangle
- o/p is type of triangle determined by 3 sides
- Equilateral, Isosceles, Scalene, Not a triangle.

Improved version

Sides of triangle integer a, b, c must satisfy the following conditions

c1. $1 \leq a \leq 200$

c2. $1 \leq b \leq 200$

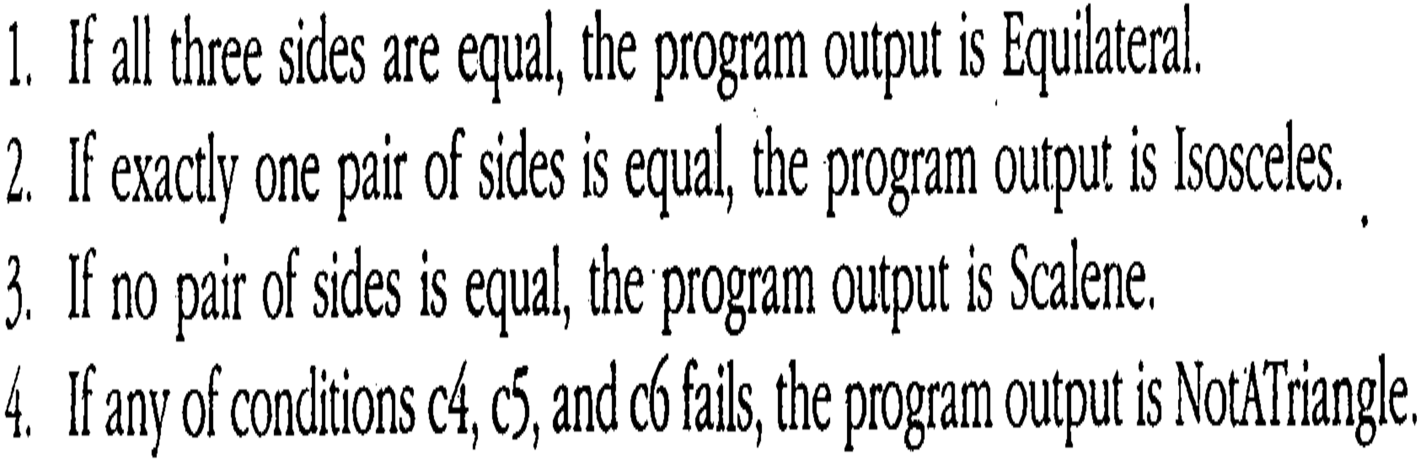
c3. $1 \leq c \leq 200$

c4. $a < b + c$

c5. $b < a + c$

c6. $c < a + b$

One of the 4 mutually exclusive output is given



1. If all three sides are equal, the program output is Equilateral.
2. If exactly one pair of sides is equal, the program output is Isosceles.
3. If no pair of sides is equal, the program output is Scalene.
4. If any of conditions c4, c5, and c6 fails, the program output is NotATriangle.

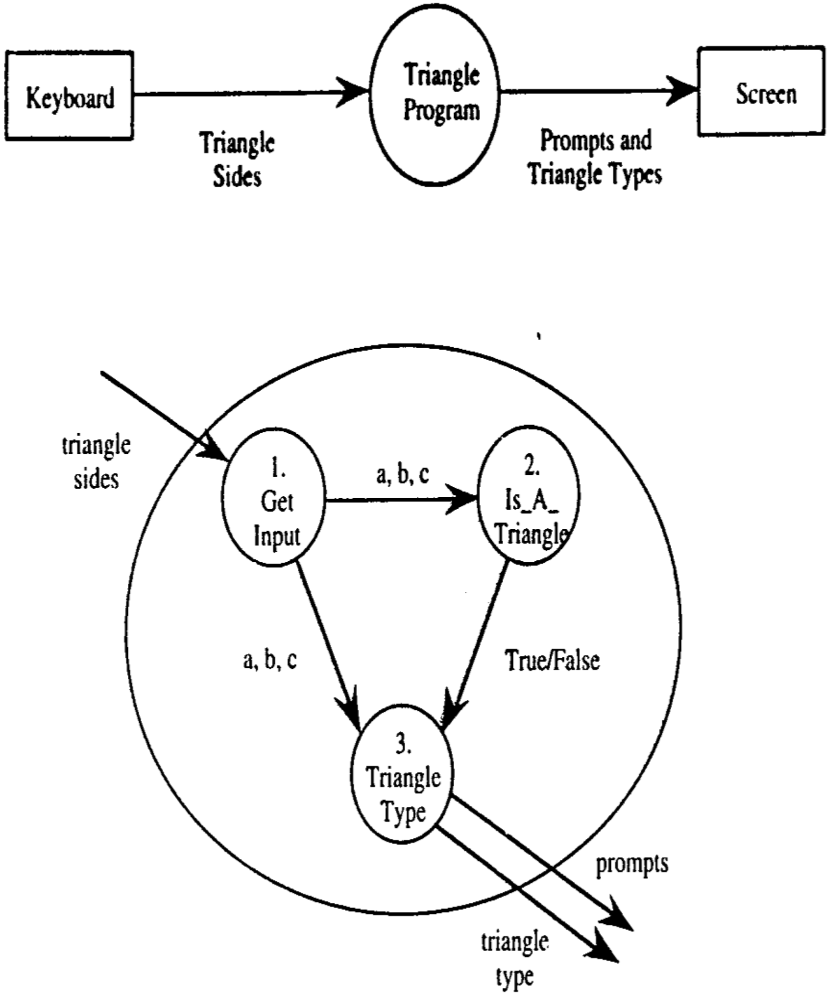
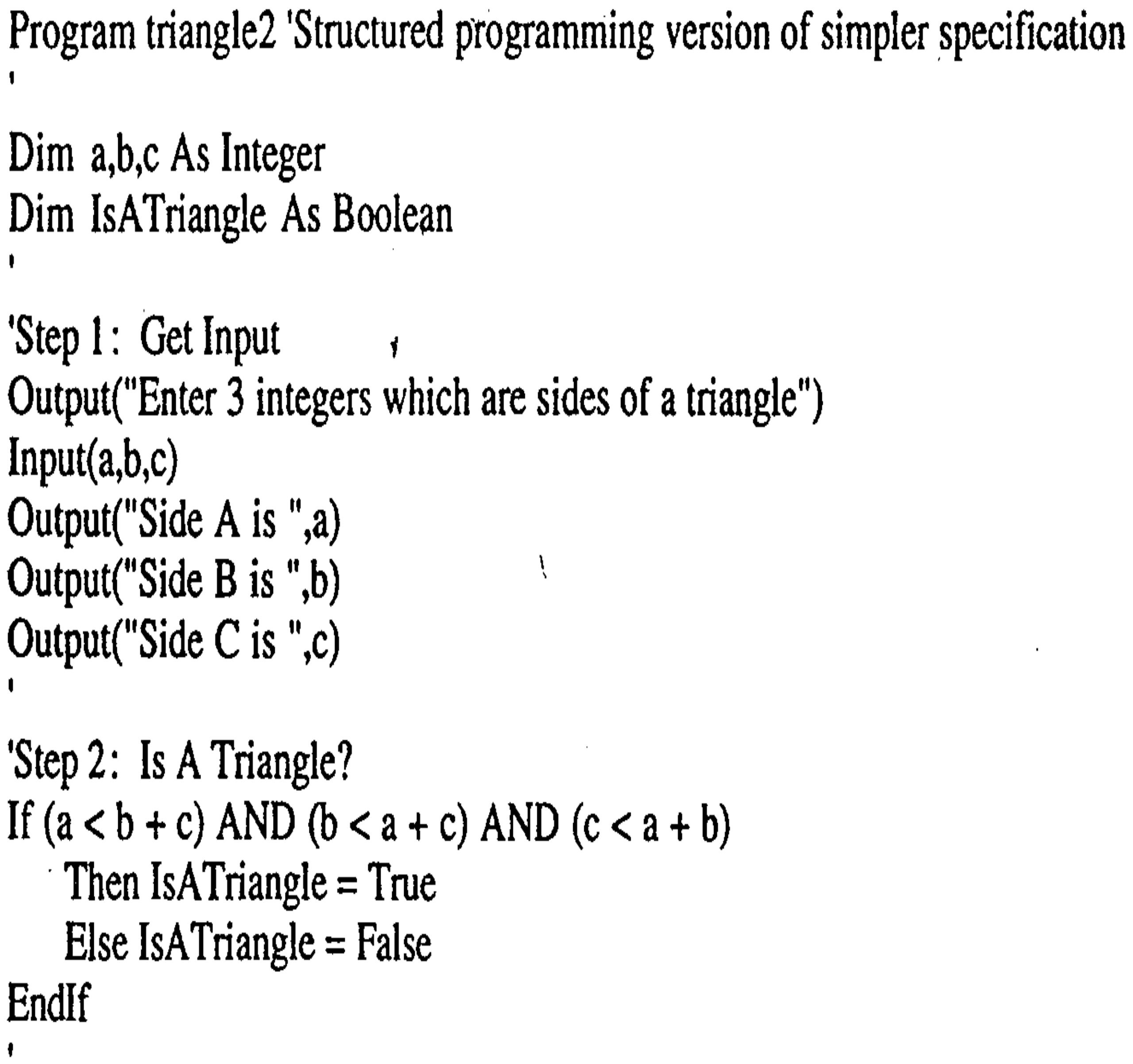
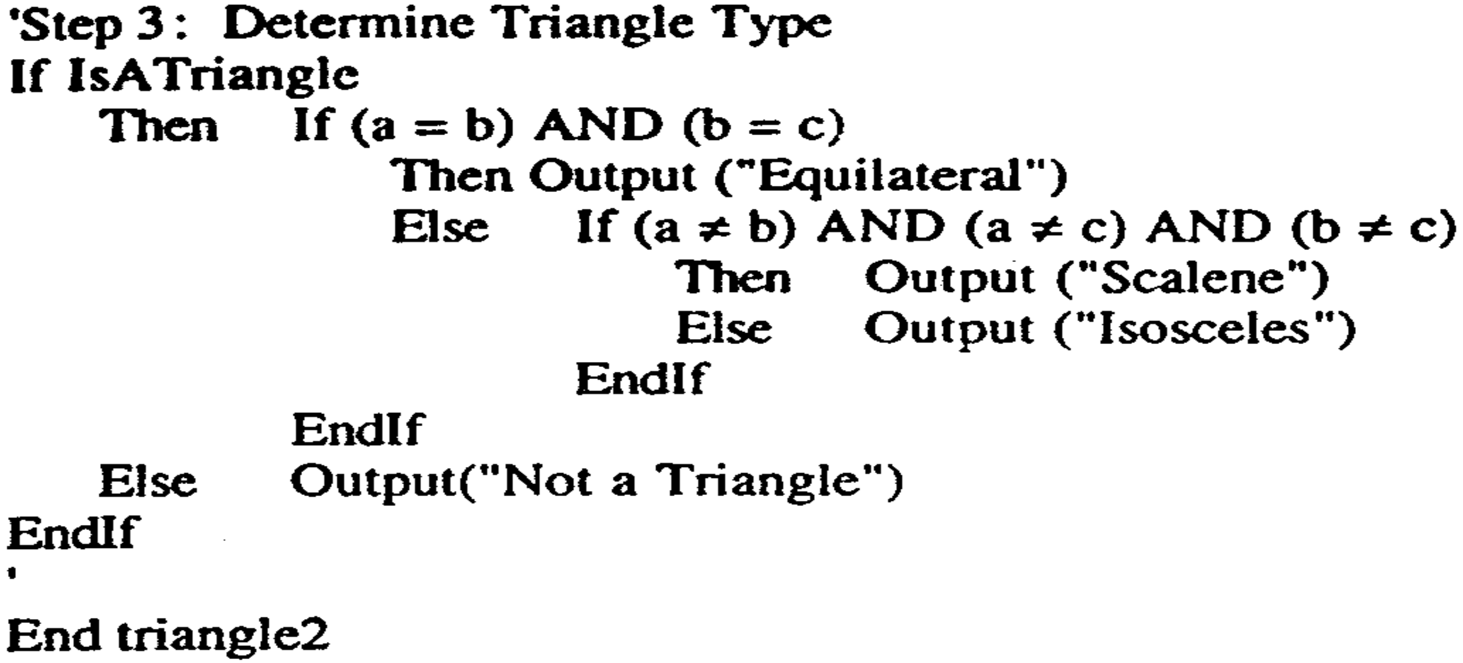


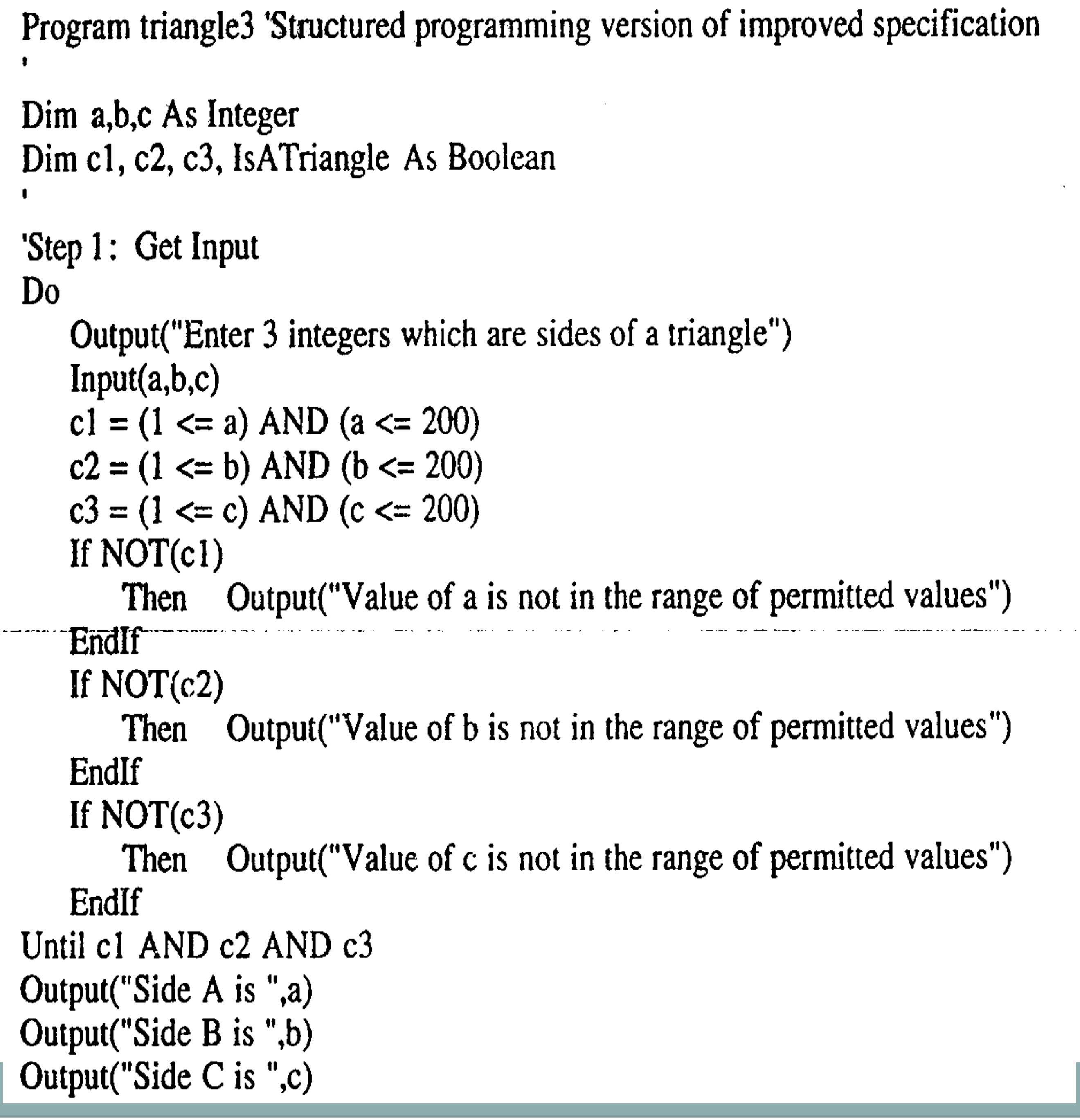
Figure 2.2 Dataflow diagram for a structured triangle program implementation.



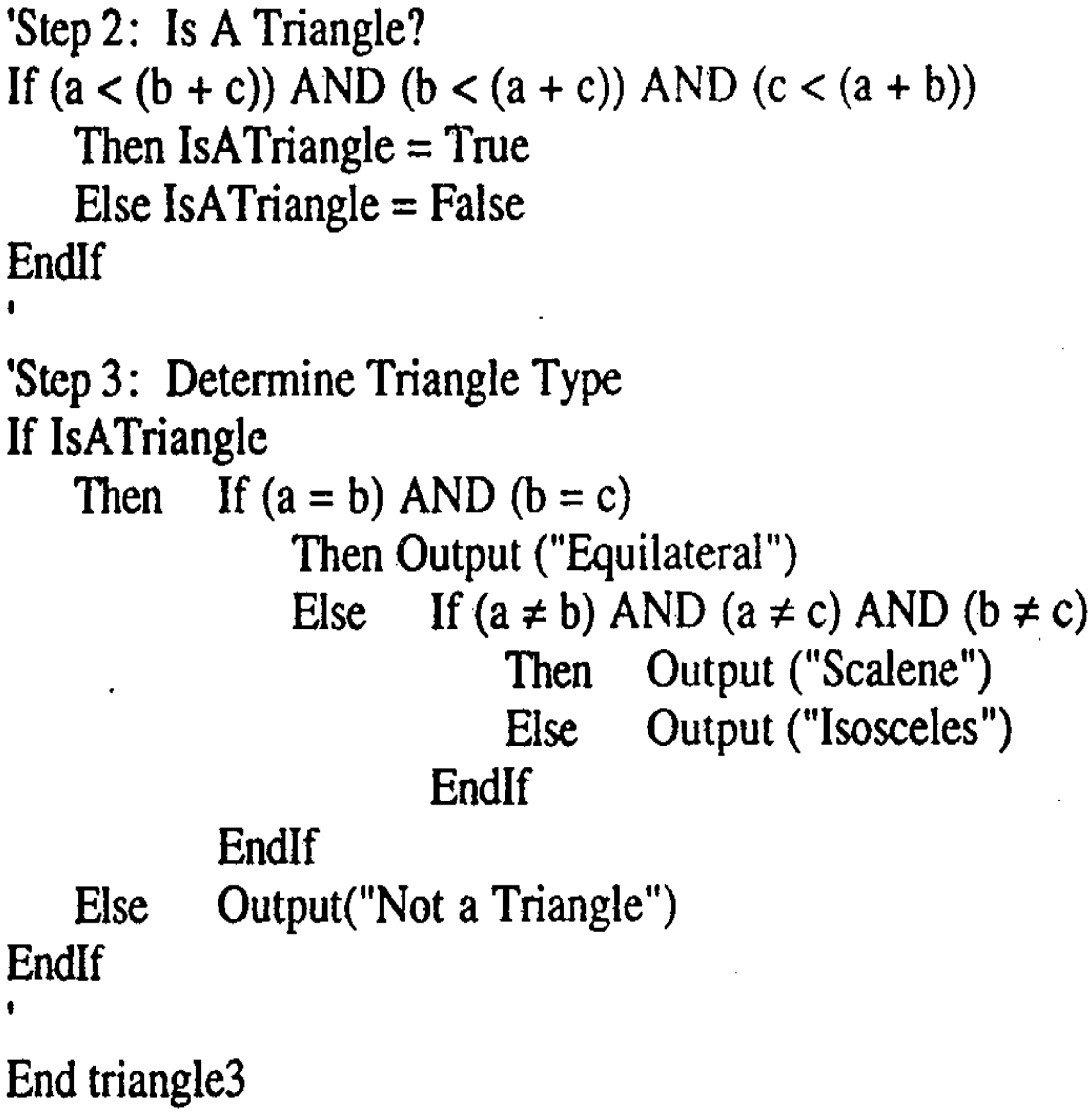
```
Program triangle2 'Structured programming version of simpler specification
'
Dim  a,b,c As Integer
Dim  IsATriangle  As Boolean
'
'Step 1:  Get Input
Output("Enter 3 integers which are sides of a triangle")
Input(a,b,c)
Output("Side A is ",a)
Output("Side B is ",b)
Output("Side C is ",c)
'
'Step 2:  Is A Triangle?
If (a < b + c) AND (b < a + c) AND (c < a + b)
    Then IsATriangle = True
    Else IsATriangle = False
EndIf
'
```



```
'Step 3:  Determine Triangle Type
If IsATriangle
    Then    If (a = b) AND (b = c)
                Then Output ("Equilateral")
                Else    If (a ≠ b) AND (a ≠ c) AND (b ≠ c)
                            Then    Output ("Scalene")
                            Else    Output ("Isosceles")
                        EndIf
            EndIf
    Else    Output("Not a Triangle")
EndIf
'
End triangle2
```

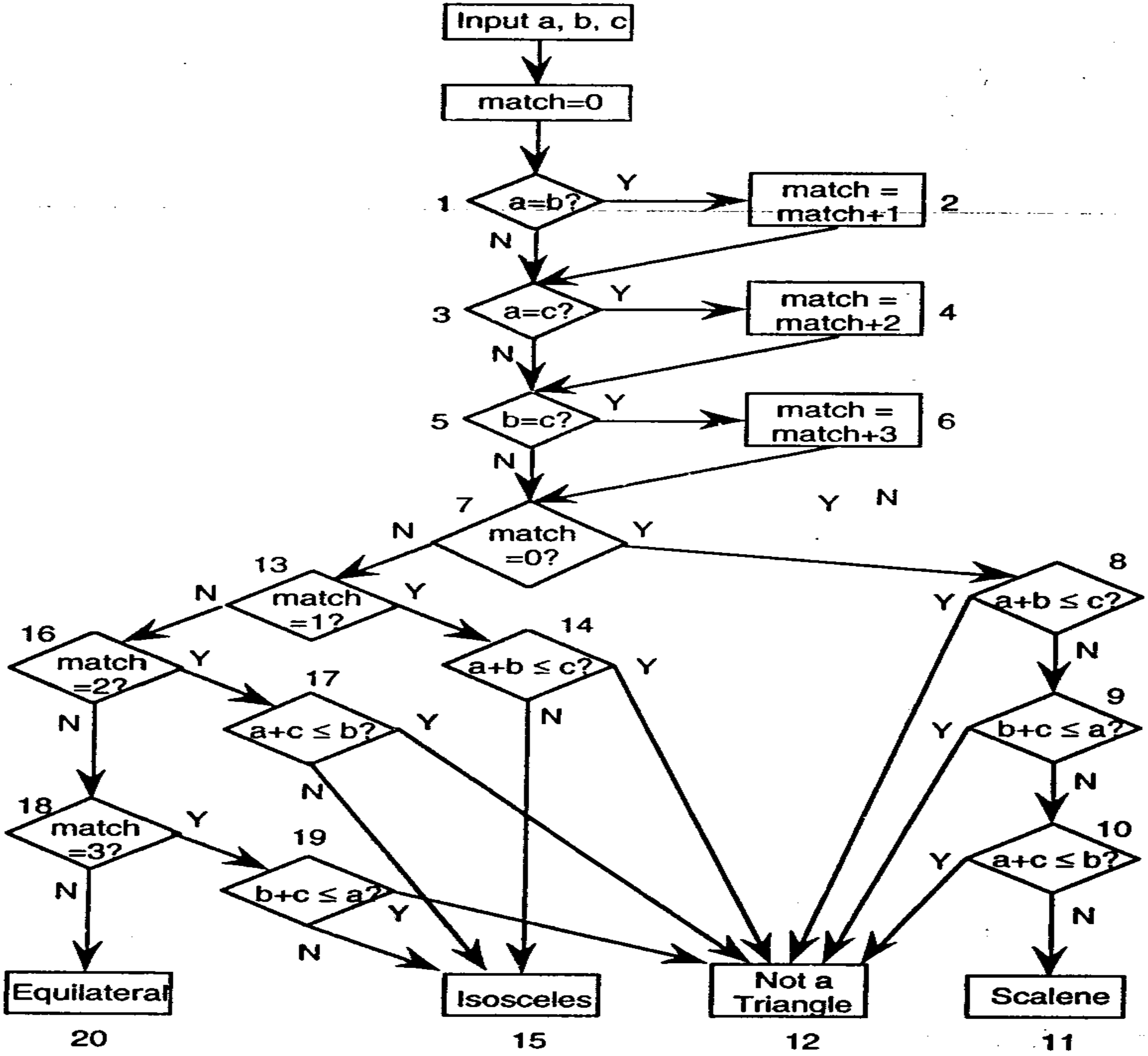


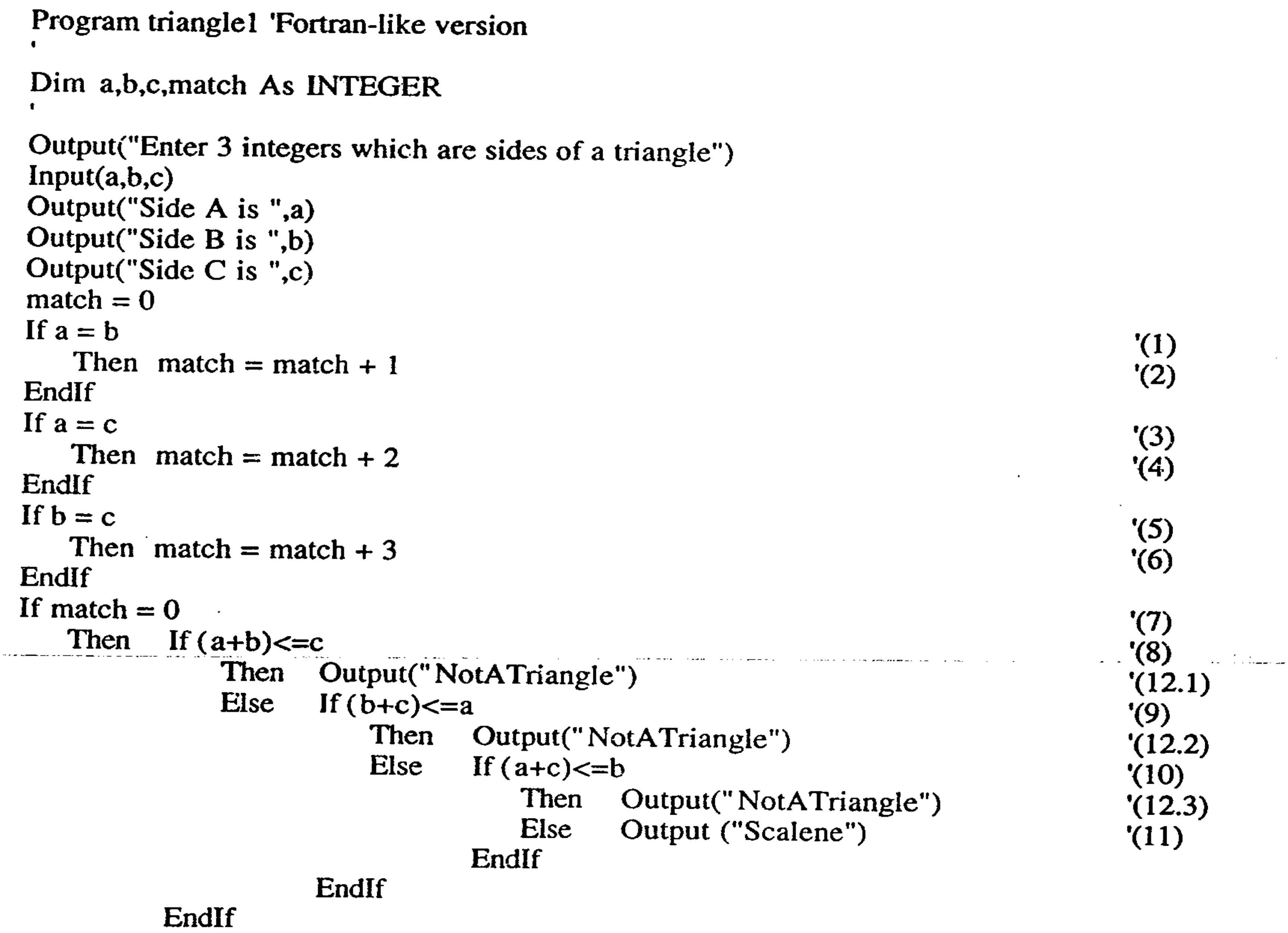
```
Program triangle3 'Structured programming version of improved specification
'
Dim a,b,c As Integer
Dim c1, c2, c3, IsATriangle As Boolean
'
'Step 1: Get Input
Do
    Output("Enter 3 integers which are sides of a triangle")
    Input(a,b,c)
    c1 = (1 <= a) AND (a <= 200)
    c2 = (1 <= b) AND (b <= 200)
    c3 = (1 <= c) AND (c <= 200)
    If NOT(c1)
        Then    Output("Value of a is not in the range of permitted values")
    EndIf
    If NOT(c2)
        Then    Output("Value of b is not in the range of permitted values")
    EndIf
    If NOT(c3)
        Then    Output("Value of c is not in the range of permitted values")
    EndIf
Until c1 AND c2 AND c3
Output("Side A is ",a)
Output("Side B is ",b)
Output("Side C is ",c)
```



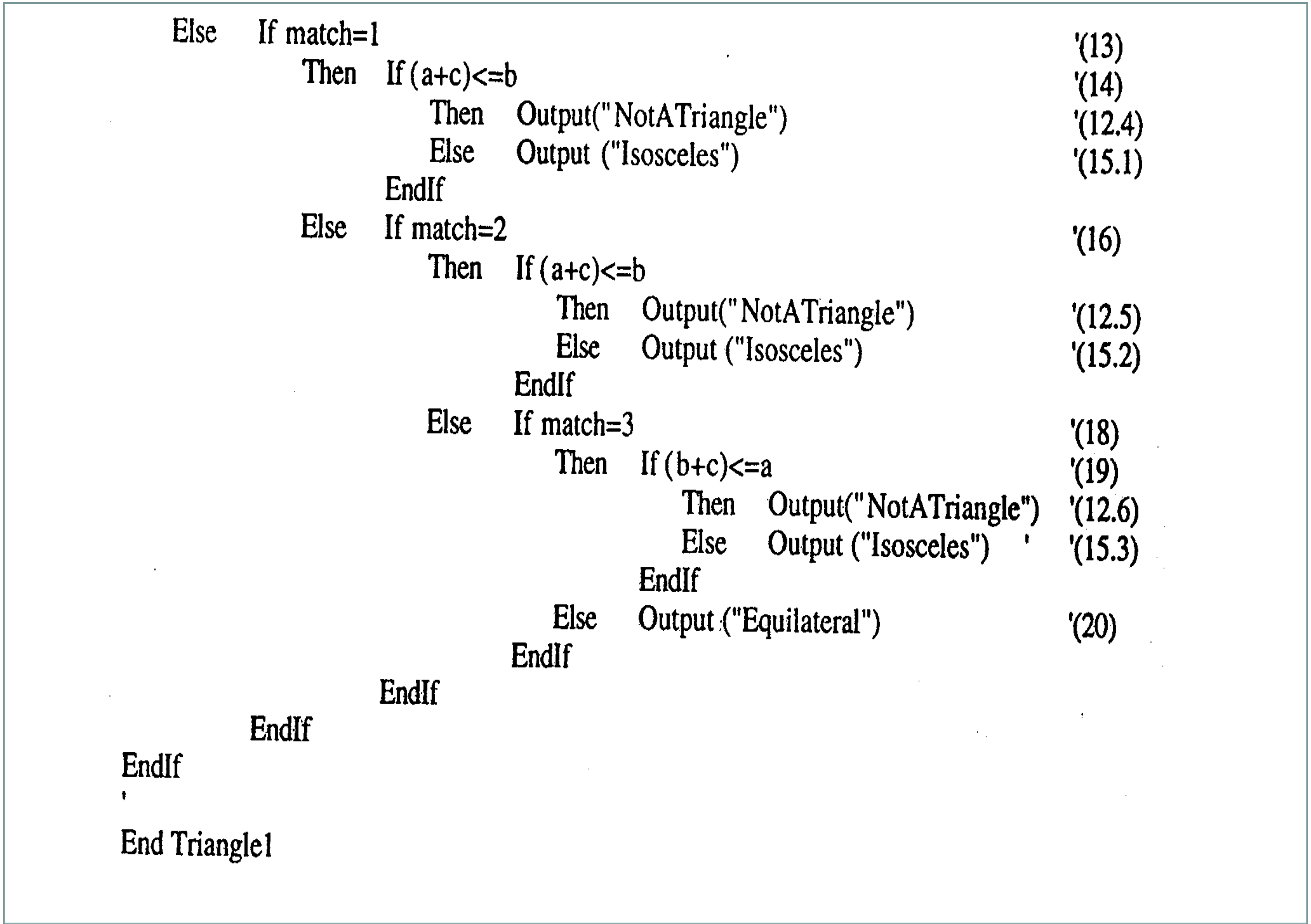
```
'Step 2:  Is A Triangle?
If (a < (b + c)) AND (b < (a + c)) AND (c < (a + b))
     Then IsATriangle = True
     Else IsATriangle = False
EndIf
'
'Step 3:  Determine Triangle Type
If IsATriangle
     Then     If (a = b) AND (b = c)
                   Then Output ("Equilateral")
                   Else     If (a ≠ b) AND (a ≠ c) AND (b ≠ c)
                                 Then    Output ("Scalene")
                                 Else    Output ("Isosceles")
                            EndIf
              EndIf
     Else     Output("Not a Triangle")
EndIf
'
End triangle3
```

Traditional Implementation





```
Program triangle1 'Fortran-like version
'
Dim a,b,c,match As INTEGER
'
Output("Enter 3 integers which are sides of a triangle")
Input(a,b,c)
Output("Side A is ",a)
Output("Side B is ",b)
Output("Side C is ",c)
match = 0
If a = b                                                                  '(1)
    Then  match = match + 1                                               '(2)
EndIf
If a = c                                                                  '(3)
    Then  match = match + 2                                               '(4)
EndIf
If b = c                                                                  '(5)
    Then  match = match + 3                                               '(6)
EndIf
If match = 0                                                              '(7)
    Then   If (a+b)<=c                                                    '(8)
                Then    Output("NotATriangle")                            '(12.1)
                Else    If (b+c)<=a                                       '(9)
                             Then    Output("NotATriangle")               '(12.2)
                             Else    If (a+c)<=b                          '(10)
                                          Then    Output("NotATriangle")  '(12.3)
                                          Else    Output ("Scalene")      '(11)
                                     EndIf
                        EndIf
           EndIf
```



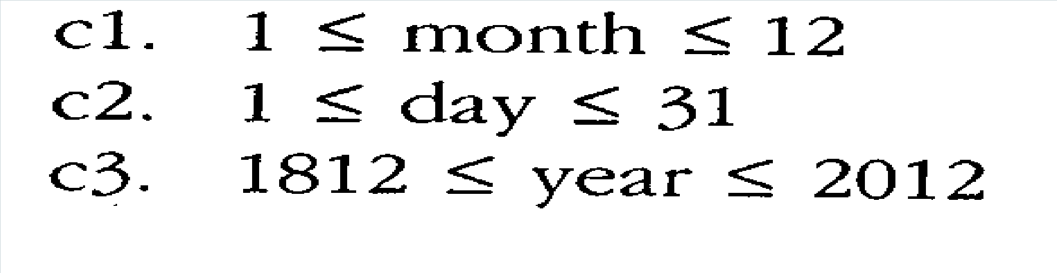
```
  Else  If match=1                                               '(13)
          Then  If (a+c)<=b                                      '(14)
                  Then  Output("NotATriangle")                   '(12.4)
                  Else  Output ("Isosceles")                     '(15.1)
                EndIf
          Else  If match=2                                       '(16)
                  Then  If (a+c)<=b
                          Then  Output("NotATriangle")           '(12.5)
                          Else  Output ("Isosceles")             '(15.2)
                        EndIf
                  Else  If match=3                               '(18)
                          Then  If (b+c)<=a                      '(19)
                                  Then  Output("NotATriangle")   '(12.6)
                                  Else  Output ("Isosceles")  '  '(15.3)
                                EndIf
                          Else  Output ("Equilateral")           '(20)
                        EndIf
                EndIf
      EndIf
EndIf
'
End Triangle1
```

The NextDate Function

- Illustrate complexity
- Logical relationship among the i/p variables

Problem statement:

- NextDate is a function of 3 variables Month, Day, Year.
- It returns the date of the day after the i/p date.
- condition



c1. $1 \leq \text{month} \leq 12$

c2. $1 \leq \text{day} \leq 31$

c3. $1812 \leq \text{year} \leq 2012$

Problem statement

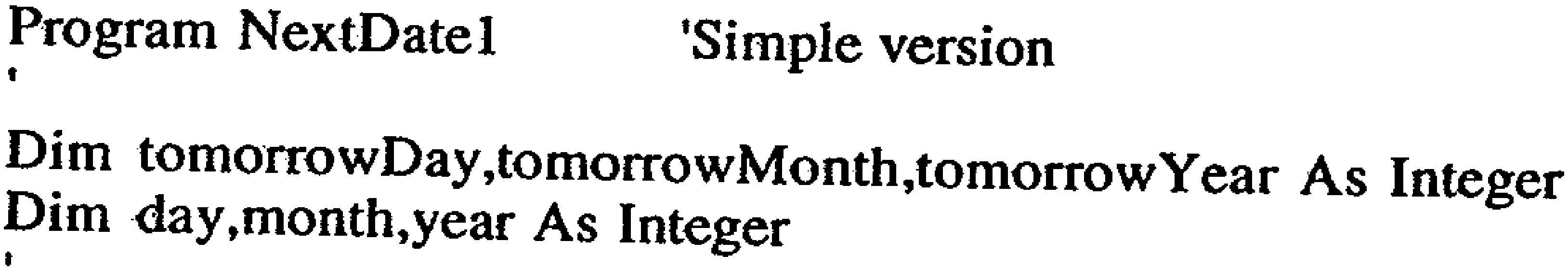
- Responses for invalid values of i/p values for day, month, year.
- Responses for invalid combination of i/p june 31 any year.
- If any of the conditions C1, C2, or C3 fails
  - Corresponding variables has out-of-range values.
  - Eg. "Value of month not in range 1…12"
- If invalid day-month- year combination exist NextDate collapses these into one message

  "Invalid input date"

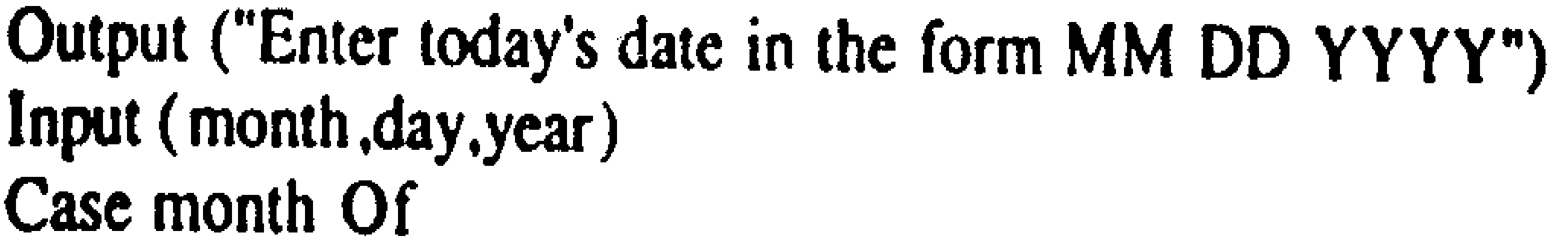
Discussion

- Two source of complexity
  - Complexity of input domain
  - Rule that determine when a year is leap year.
- A year is 365.2422 days long
- Leap years are used for the “extra day” problem.
- According to Gregorian calendar
  - A year is a leap year if it is divisible by 4, unless it is a century year.
  - Century years are leap years only if they are multiples of 400
  - So 1992, 1996, 2000 are leap years… 1900 is not

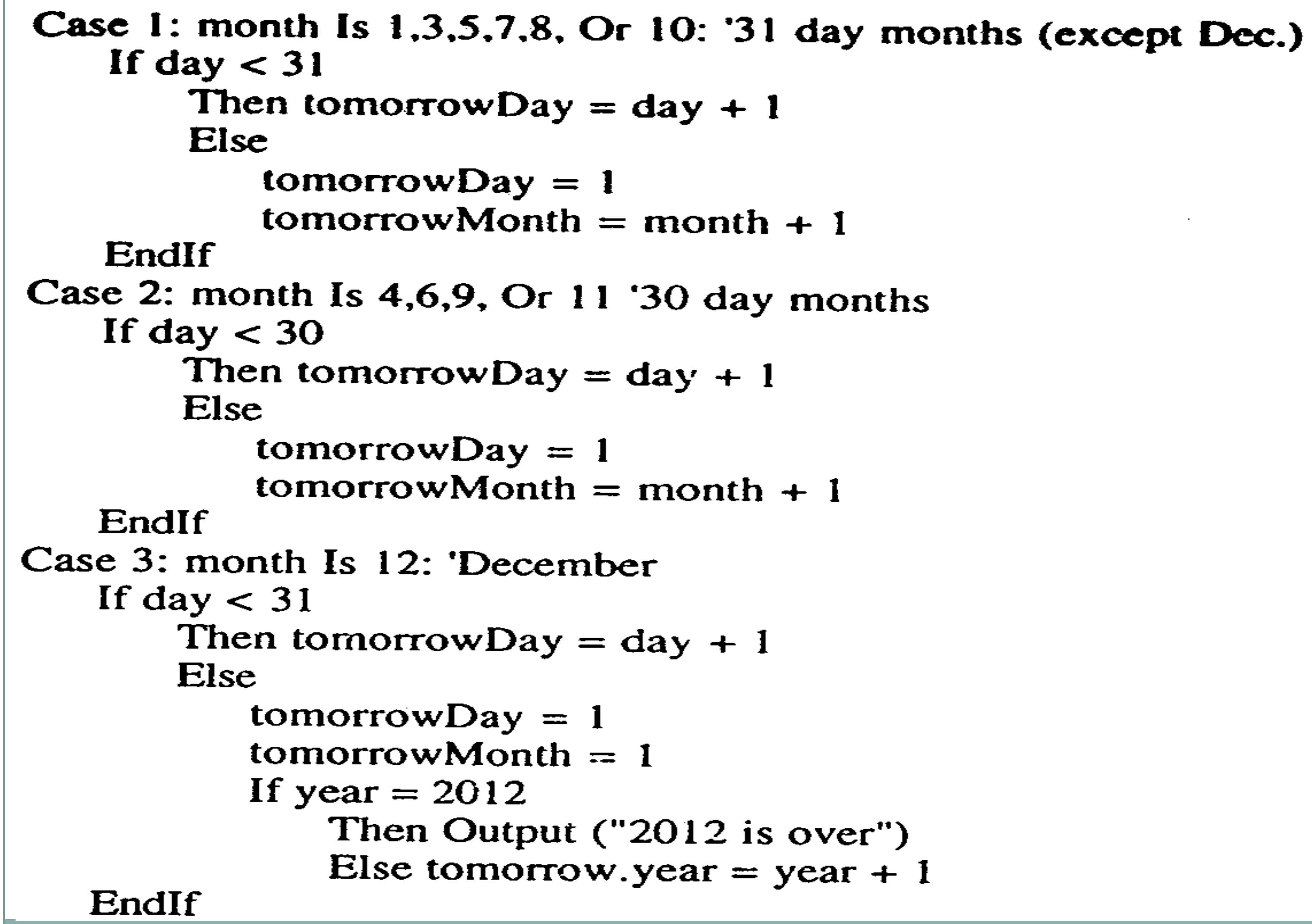
Implementation



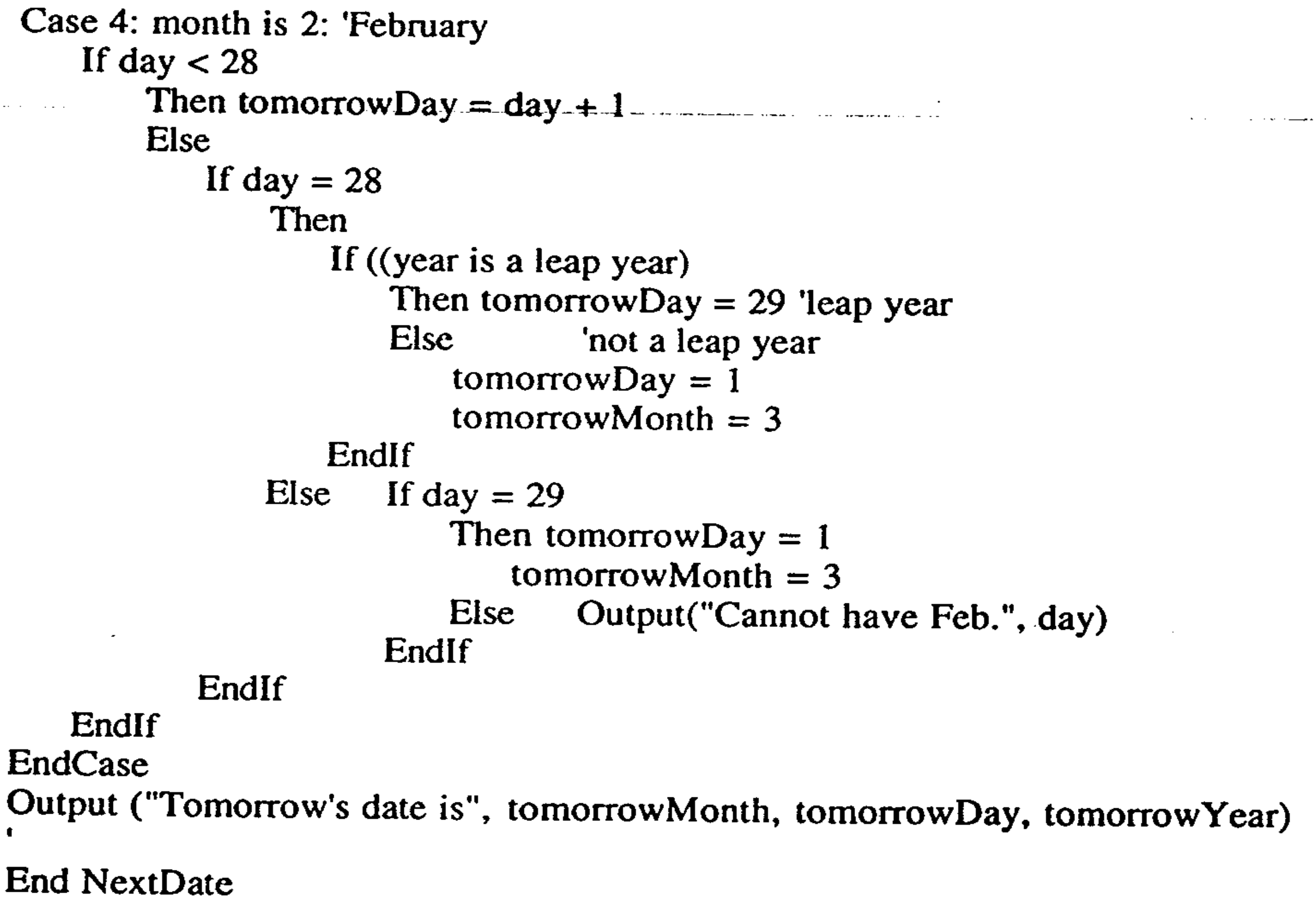
```
Program NextDate1          'Simple version
'
Dim tomorrowDay,tomorrowMonth,tomorrowYear As Integer
Dim day,month,year As Integer
'
```



```
Output ("Enter today's date in the form MM DD YYYY")
Input (month,day,year)
Case month Of
```

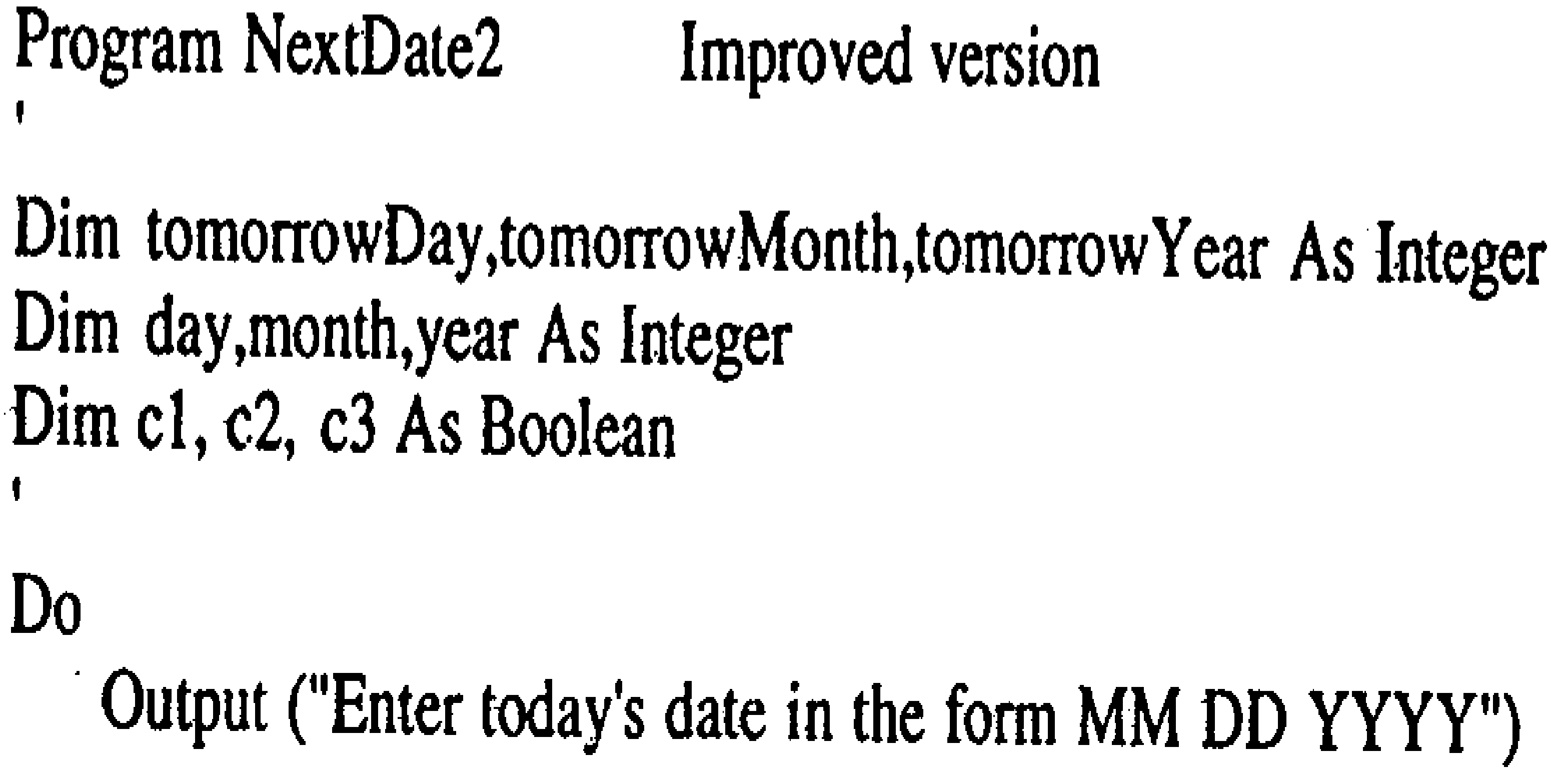


```
Case 1: month Is 1,3,5,7,8, Or 10: '31 day months (except Dec.)
    If day < 31
        Then tomorrowDay = day + 1
        Else
            tomorrowDay = 1
            tomorrowMonth = month + 1
    EndIf
Case 2: month Is 4,6,9, Or 11 '30 day months
    If day < 30
        Then tomorrowDay = day + 1
        Else
            tomorrowDay = 1
            tomorrowMonth = month + 1
    EndIf
Case 3: month Is 12: 'December
    If day < 31
        Then tomorrowDay = day + 1
        Else
            tomorrowDay = 1
            tomorrowMonth = 1
            If year = 2012
                Then Output ("2012 is over")
                Else tomorrow.year = year + 1
    EndIf
```

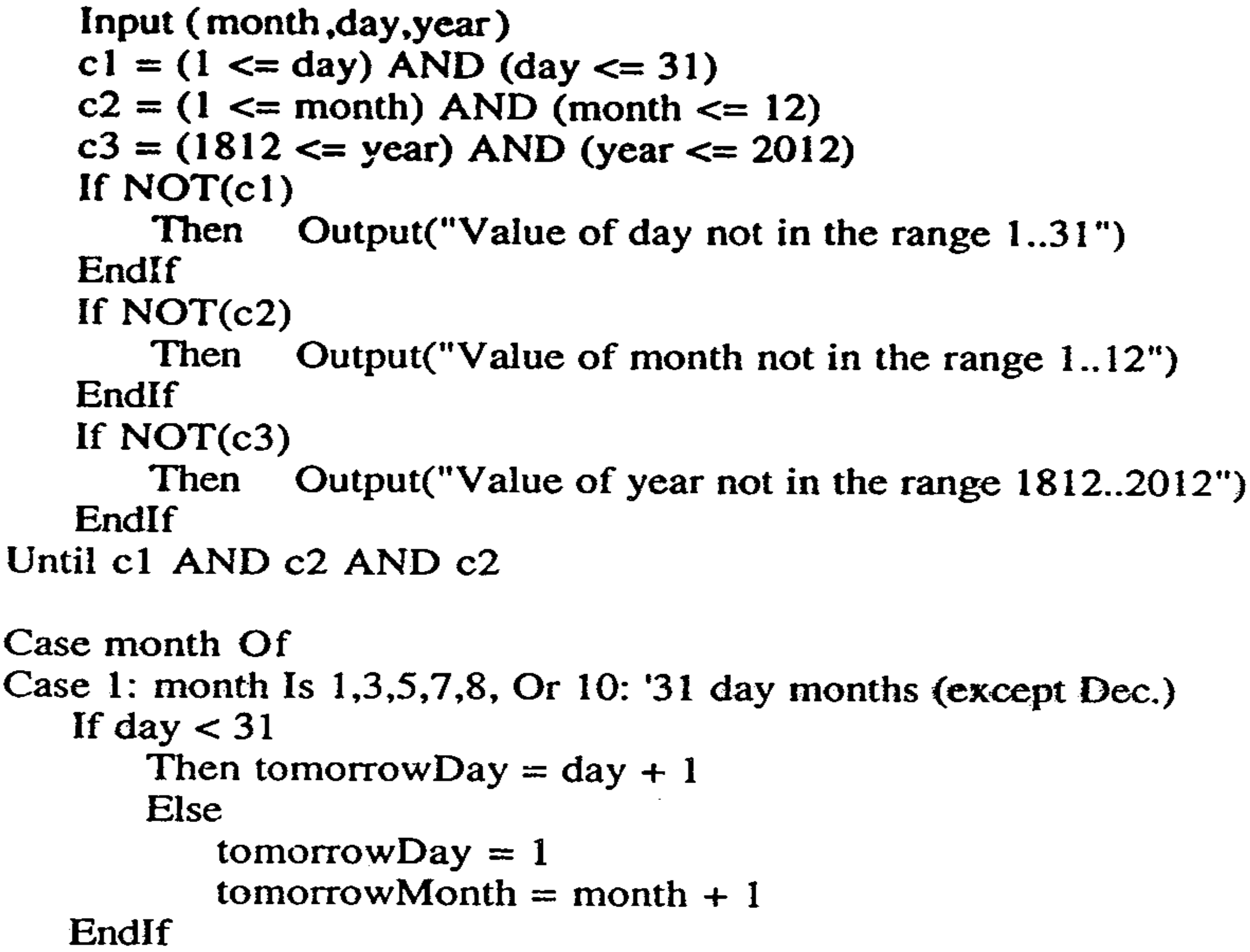


```
Case 4: month is 2: 'February
    If day < 28
        Then tomorrowDay = day + 1
        Else
            If day = 28
                Then
                    If ((year is a leap year)
                        Then tomorrowDay = 29 'leap year
                        Else            'not a leap year
                            tomorrowDay = 1
                            tomorrowMonth = 3
                    EndIf
                Else    If day = 29
                            Then tomorrowDay = 1
                                tomorrowMonth = 3
                            Else    Output("Cannot have Feb.", day)
                        EndIf
            EndIf
    EndIf
EndCase
Output ("Tomorrow's date is", tomorrowMonth, tomorrowDay, tomorrowYear)
'
End NextDate
```

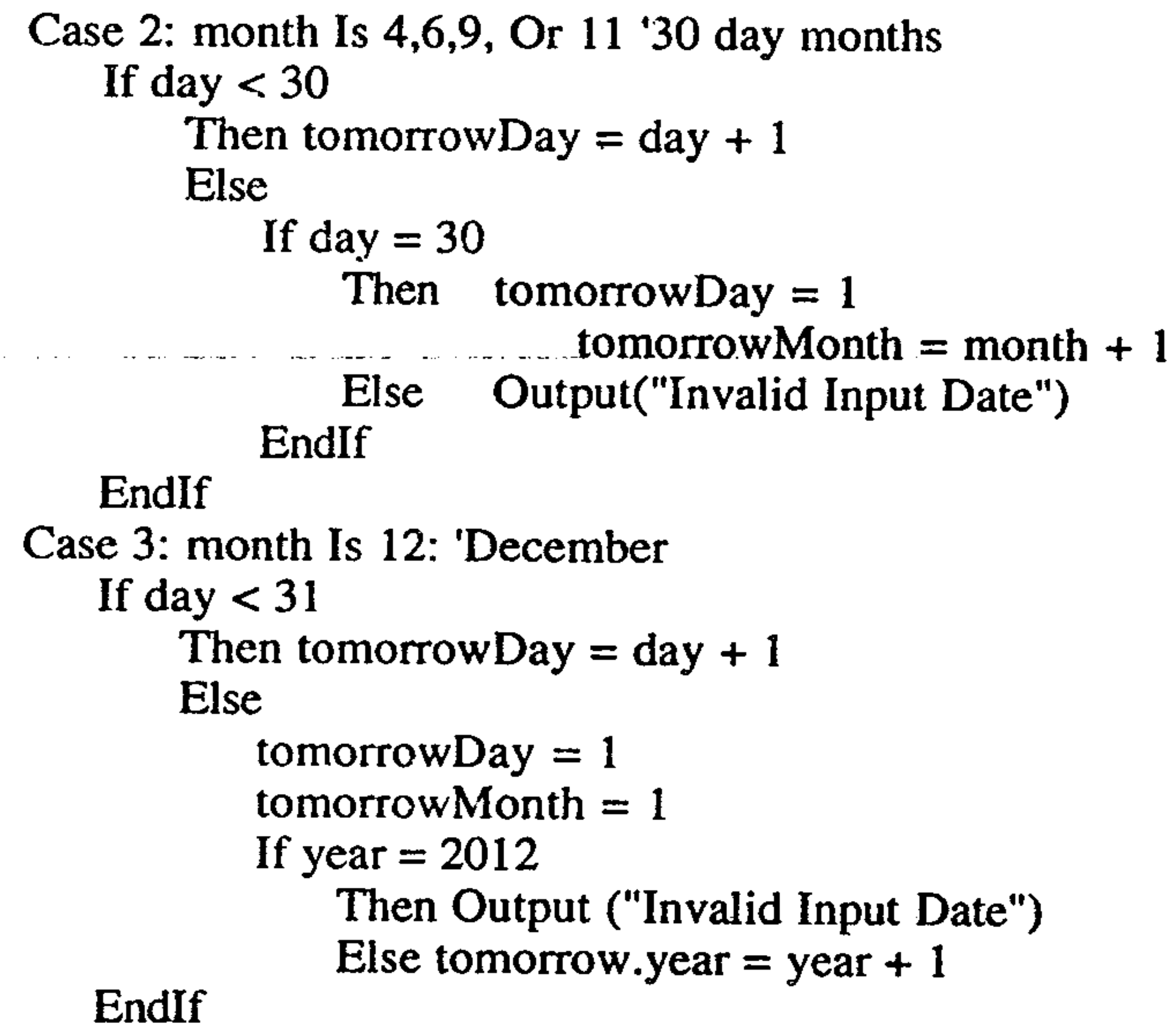
Improved Version



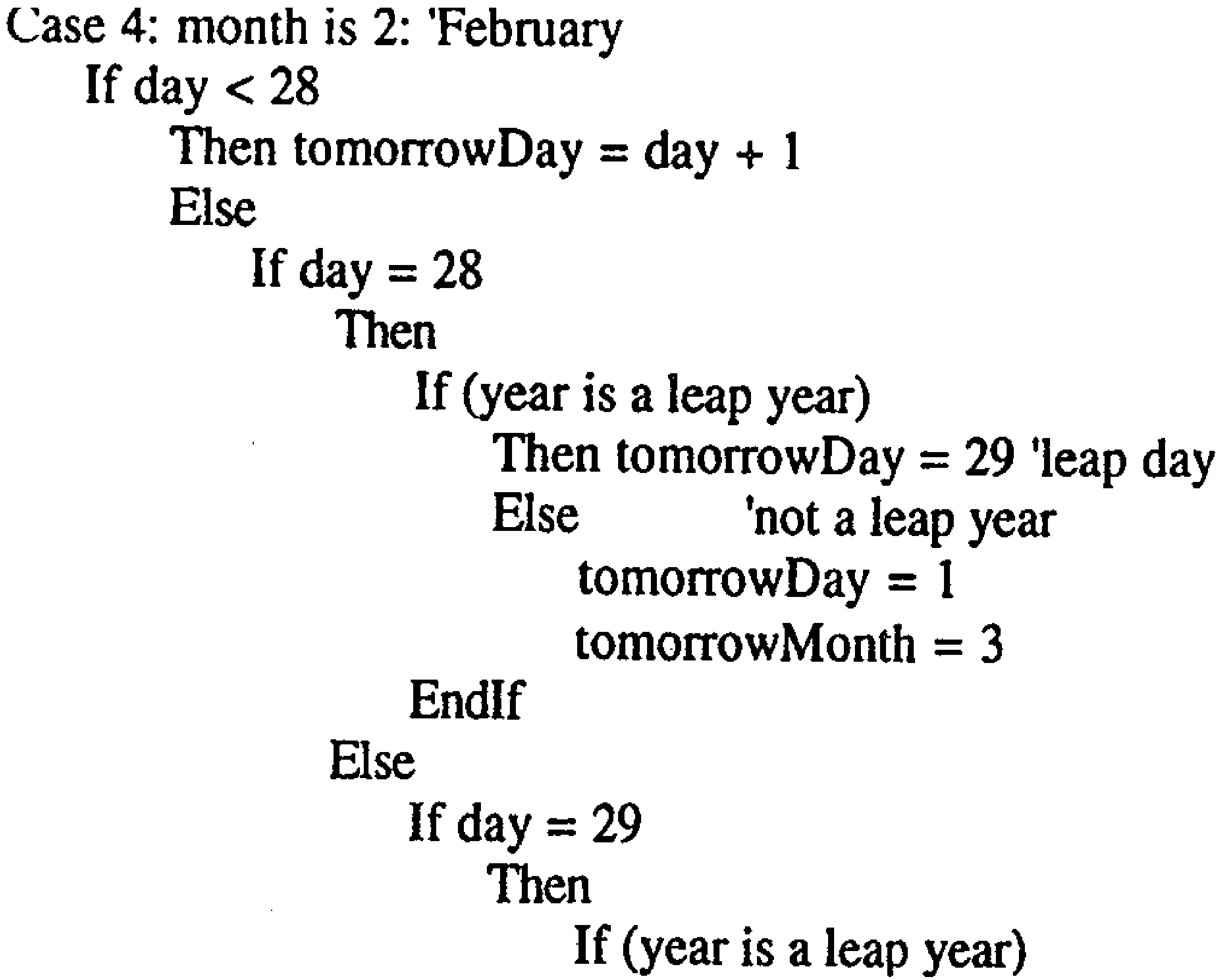
```
Program NextDate2        Improved version
'
Dim tomorrowDay,tomorrowMonth,tomorrowYear As Integer
Dim day,month,year As Integer
Dim c1, c2, c3 As Boolean
'
Do
    Output ("Enter today's date in the form MM DD YYYY")
```



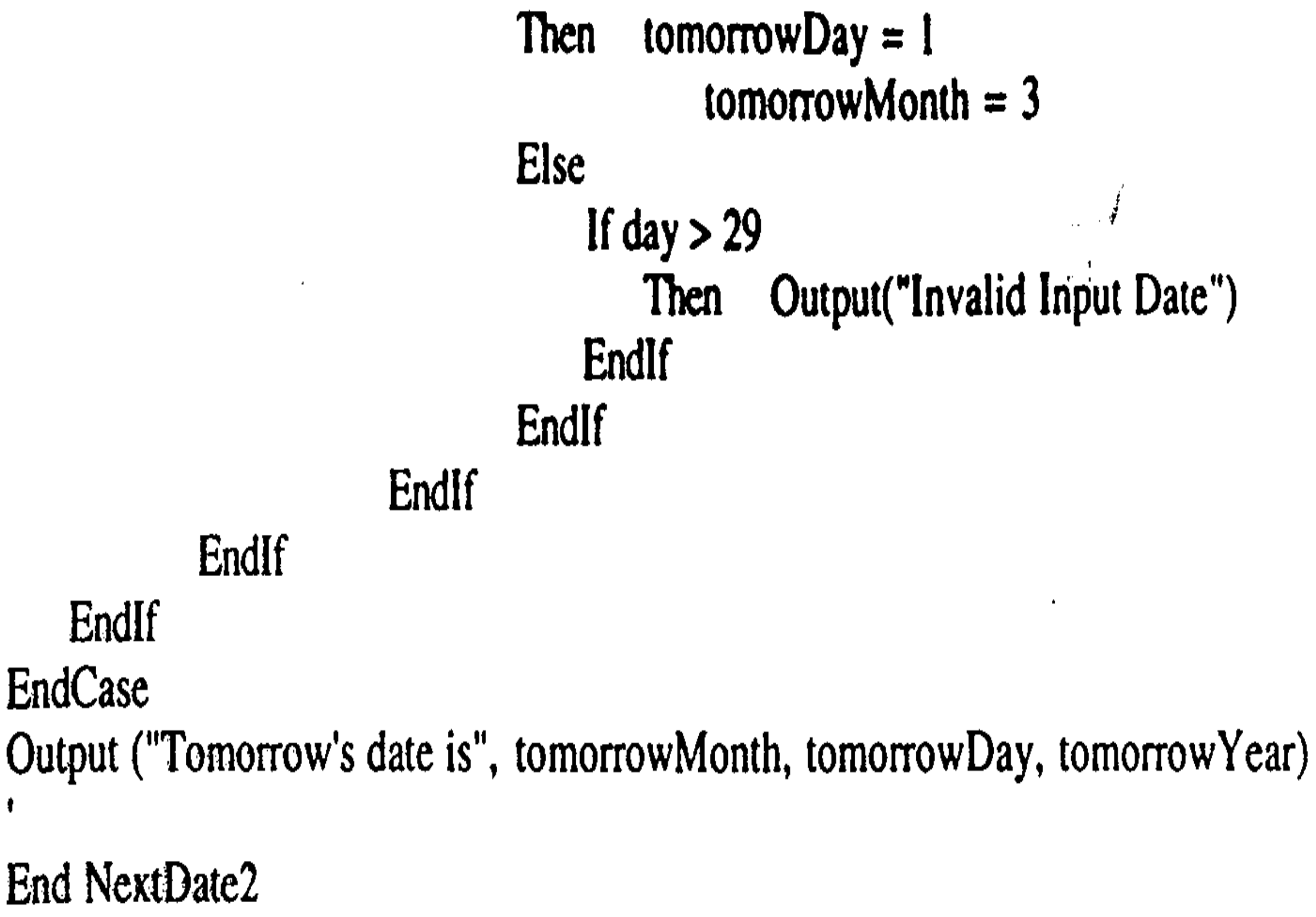
```
    Input (month,day,year)
    c1 = (1 <= day) AND (day <= 31)
    c2 = (1 <= month) AND (month <= 12)
    c3 = (1812 <= year) AND (year <= 2012)
    If NOT(c1)
        Then    Output("Value of day not in the range 1..31")
    EndIf
    If NOT(c2)
        Then    Output("Value of month not in the range 1..12")
    EndIf
    If NOT(c3)
        Then    Output("Value of year not in the range 1812..2012")
    EndIf
Until c1 AND c2 AND c2

Case month Of
Case 1: month Is 1,3,5,7,8, Or 10: '31 day months (except Dec.)
    If day < 31
        Then tomorrowDay = day + 1
        Else
            tomorrowDay = 1
            tomorrowMonth = month + 1
    EndIf
```



```
Case 2: month Is 4,6,9, Or 11 '30 day months
    If day < 30
        Then tomorrowDay = day + 1
        Else
            If day = 30
                Then    tomorrowDay = 1
                        tomorrowMonth = month + 1
                Else    Output("Invalid Input Date")
            EndIf
    EndIf
Case 3: month Is 12: 'December
    If day < 31
        Then tomorrowDay = day + 1
        Else
            tomorrowDay = 1
            tomorrowMonth = 1
            If year = 2012
                Then Output ("Invalid Input Date")
                Else tomorrow.year = year + 1
    EndIf
```



```
Case 4: month is 2: 'February
    If day < 28
        Then tomorrowDay = day + 1
        Else
            If day = 28
                Then
                    If (year is a leap year)
                        Then tomorrowDay = 29 'leap day
                        Else            'not a leap year
                            tomorrowDay = 1
                            tomorrowMonth = 3
                    EndIf
                Else
                    If day = 29
                        Then
                            If (year is a leap year)
```



```
                                  Then    tomorrowDay = 1
                                              tomorrowMonth = 3
                                  Else
                                       If day > 29
                                          Then    Output("Invalid Input Date")
                                      EndIf
                                  EndIf
                         EndIf
             EndIf
    EndIf
EndCase
Output ("Tomorrow's date is", tomorrowMonth, tomorrowDay, tomorrowYear)
'
End NextDate2
```

The commission Problem

- It contains a mix of computation & decision making.
- A rifle salesperson in the former Arizona territory sold rifle lock's, stocks, & barrel's made of a gunsmith in Missouri.
- Locks cost $45, stocks cost $30, Barrel Cost $ 25.
- Sales person has to sell at least 1 complete rifle per month
- Production limitation such that 1 sales man can sell 70 locks, 80 stocks, 90 barrels per month.

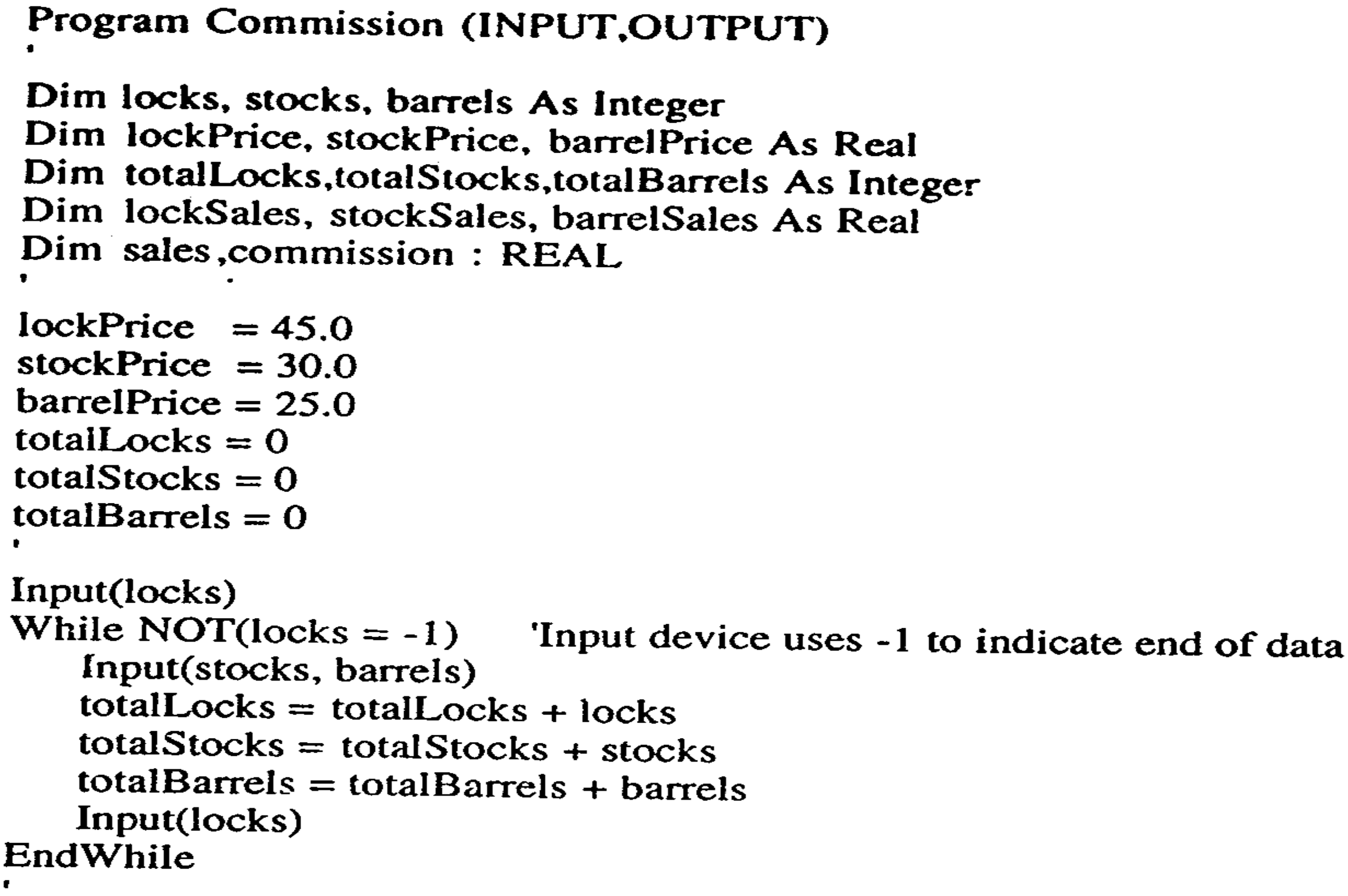
- After each town visit salesperson update sale of no of locks, stocks, barrels through a telegram to gunsmith
- At the end of month salesperson sent a shot telegram showing -1 locks sold.
- Gunman knew sales for month are over & compute the commission of sales person
  - 10% on sales up to $1000
  - 15% on the next $800
  - 20% on any sales in excess of $1800

The commission program produces a monthly sales report that gave total no. of locks, barrels, stocks sold. Sales persons total dollar sale & commission.

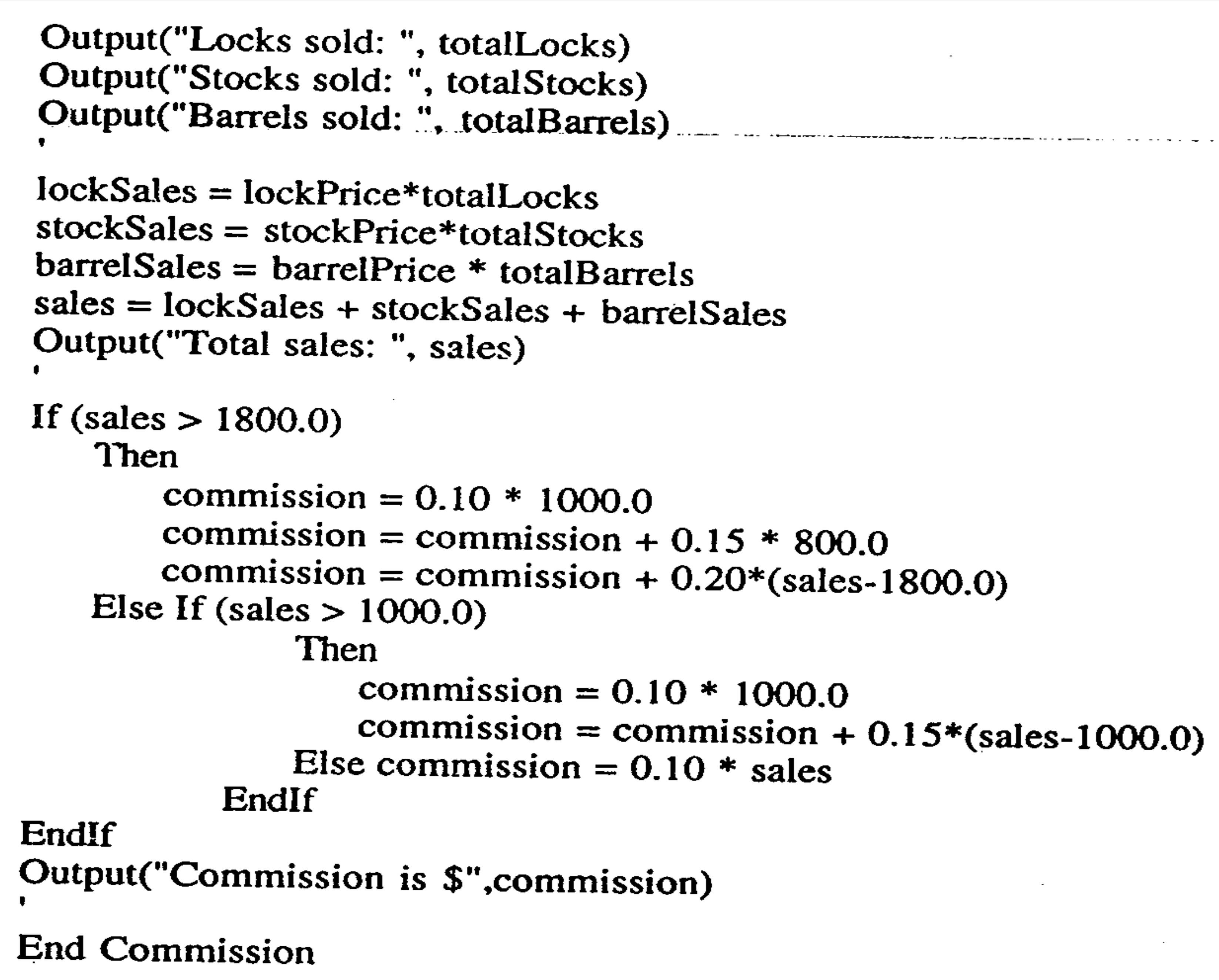
Discussion

- This problem separates into 3 distinct pieces
- The input data portion(data validation) ignore here
- Sales calculation
- Commission calculation problem.

Implementation



```
Program Commission (INPUT,OUTPUT)
'
Dim locks, stocks, barrels As Integer
Dim lockPrice, stockPrice, barrelPrice As Real
Dim totalLocks,totalStocks,totalBarrels As Integer
Dim lockSales, stockSales, barrelSales As Real
Dim sales,commission : REAL
'
lockPrice   = 45.0
stockPrice  = 30.0
barrelPrice = 25.0
totalLocks = 0
totalStocks = 0
totalBarrels = 0
'
Input(locks)
While NOT(locks = -1)     'Input device uses -1 to indicate end of data
    Input(stocks, barrels)
    totalLocks = totalLocks + locks
    totalStocks = totalStocks + stocks
    totalBarrels = totalBarrels + barrels
    Input(locks)
EndWhile
'
```



```
Output("Locks sold: ", totalLocks)
Output("Stocks sold: ", totalStocks)
Output("Barrels sold: ", totalBarrels)
'
lockSales = lockPrice*totalLocks
stockSales = stockPrice*totalStocks
barrelSales = barrelPrice * totalBarrels
sales = lockSales + stockSales + barrelSales
Output("Total sales: ", sales)
'
If (sales > 1800.0)
    Then
        commission = 0.10 * 1000.0
        commission = commission + 0.15 * 800.0
        commission = commission + 0.20*(sales-1800.0)
    Else If (sales > 1000.0)
            Then
                commission = 0.10 * 1000.0
                commission = commission + 0.15*(sales-1000.0)
            Else commission = 0.10 * sales
        EndIf
EndIf
Output("Commission is $",commission)
'
End Commission
```

The SATM System

- To better discuss the issues of integration & system testing

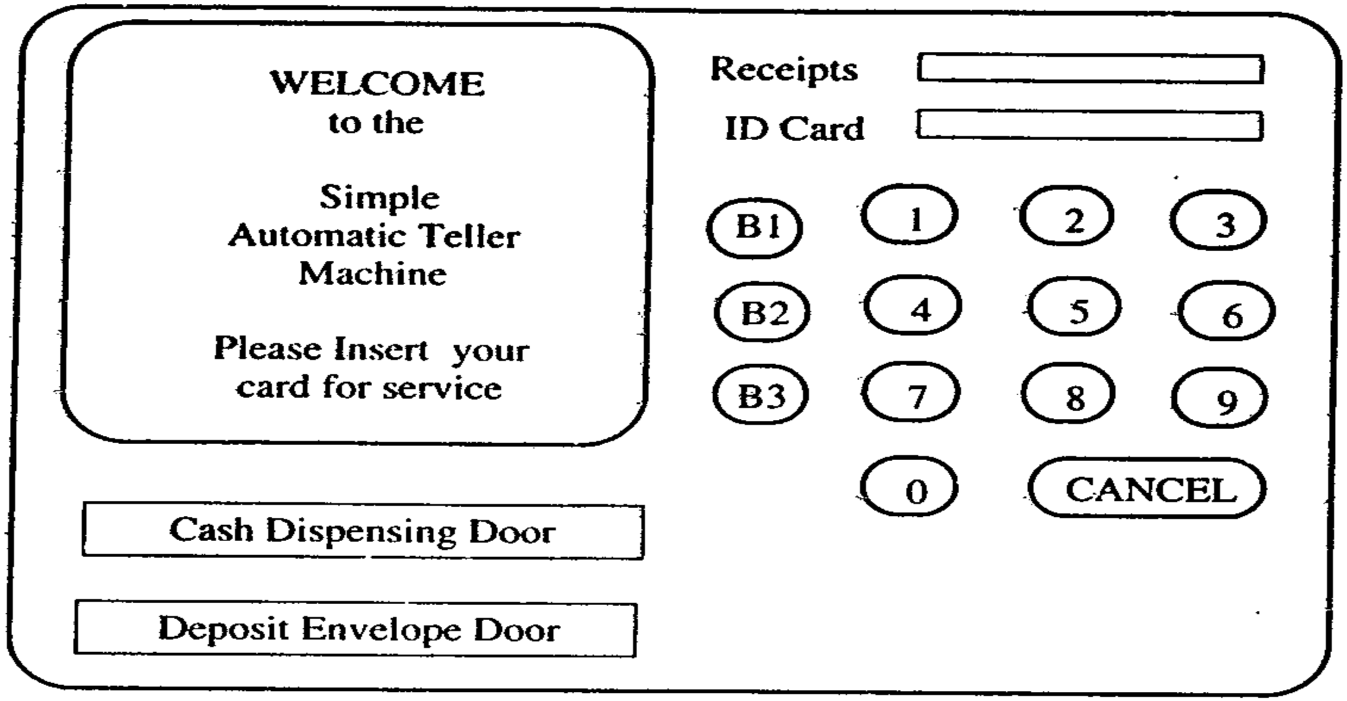
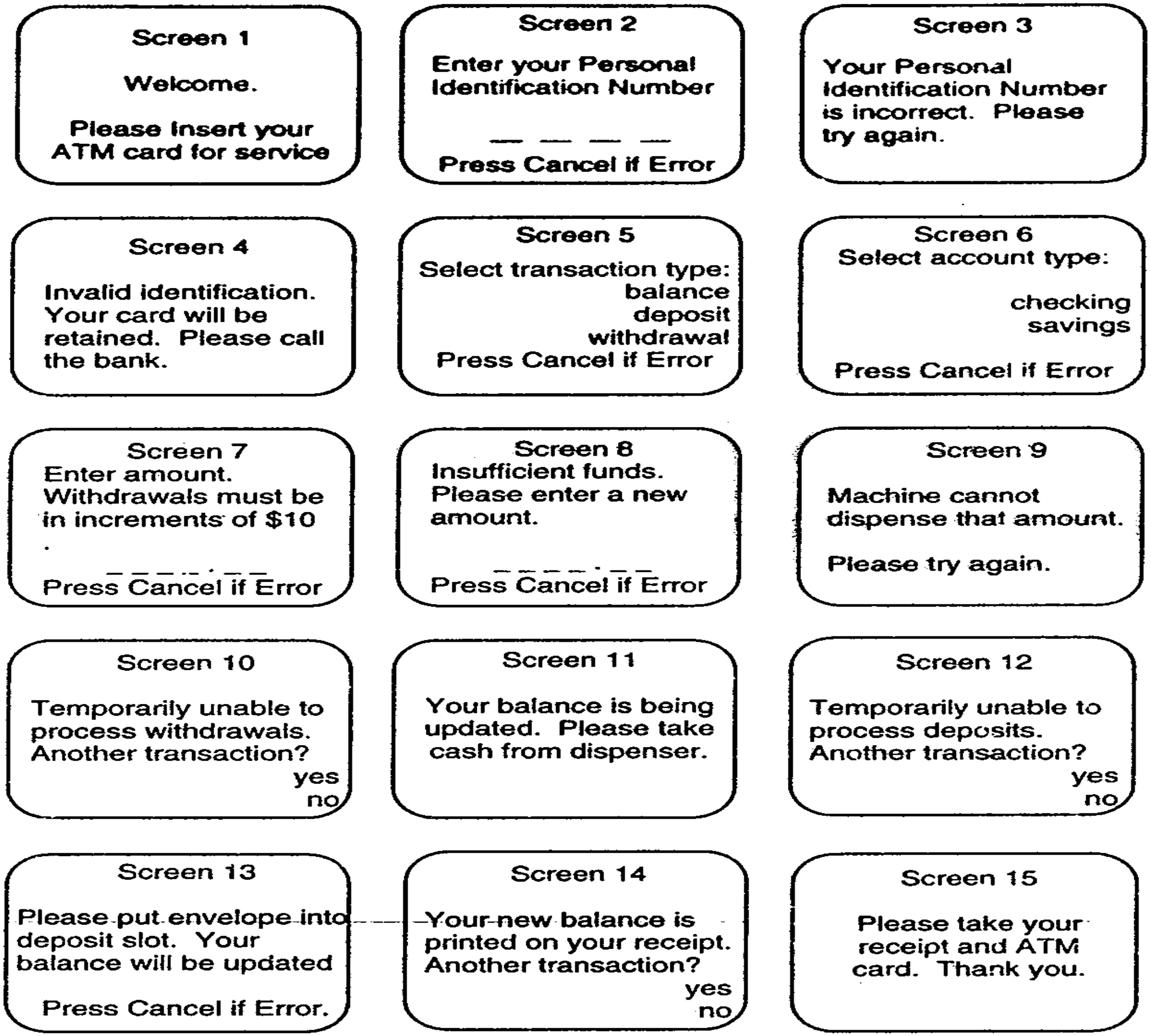


Figure 2.3 **The SATM terminal.**



Screen 1

Welcome.

Please insert your
ATM card for service

Screen 2

Enter your Personal
Identification Number

— — — —

Press Cancel if Error

Screen 3

Your Personal
Identification Number
is incorrect. Please
try again.

Screen 4

Invalid identification.
Your card will be
retained. Please call
the bank.

Screen 5

Select transaction type:
balance
deposit
withdrawal
Press Cancel if Error

Screen 6

Select account type:

checking
savings

Press Cancel if Error

Screen 7

Enter amount.
Withdrawals must be
in increments of $10
.

_ _ _ _ . _ _
Press Cancel if Error

Screen 8

Insufficient funds.
Please enter a new
amount.

_ _ _ _ . _ _
Press Cancel if Error

Screen 9

Machine cannot
dispense that amount.

Please try again.

Screen 10

Temporarily unable to
process withdrawals.
Another transaction?
yes
no

Screen 11

Your balance is being
updated. Please take
cash from dispenser.

Screen 12

Temporarily unable to
process deposits.
Another transaction?
yes
no

Screen 13

Please put envelope into
deposit slot. Your
balance will be updated

Press Cancel if Error.

Screen 14

Your new balance is
printed on your receipt.
Another transaction?
yes
no

Screen 15

Please take your
receipt and ATM
card. Thank you.

Figure 2.4 SATM screens.

The currency converter

- ***Another event driven program that emphasizes code associated with a GUI***
- ***A sample GUI built with visual basic is shown.***

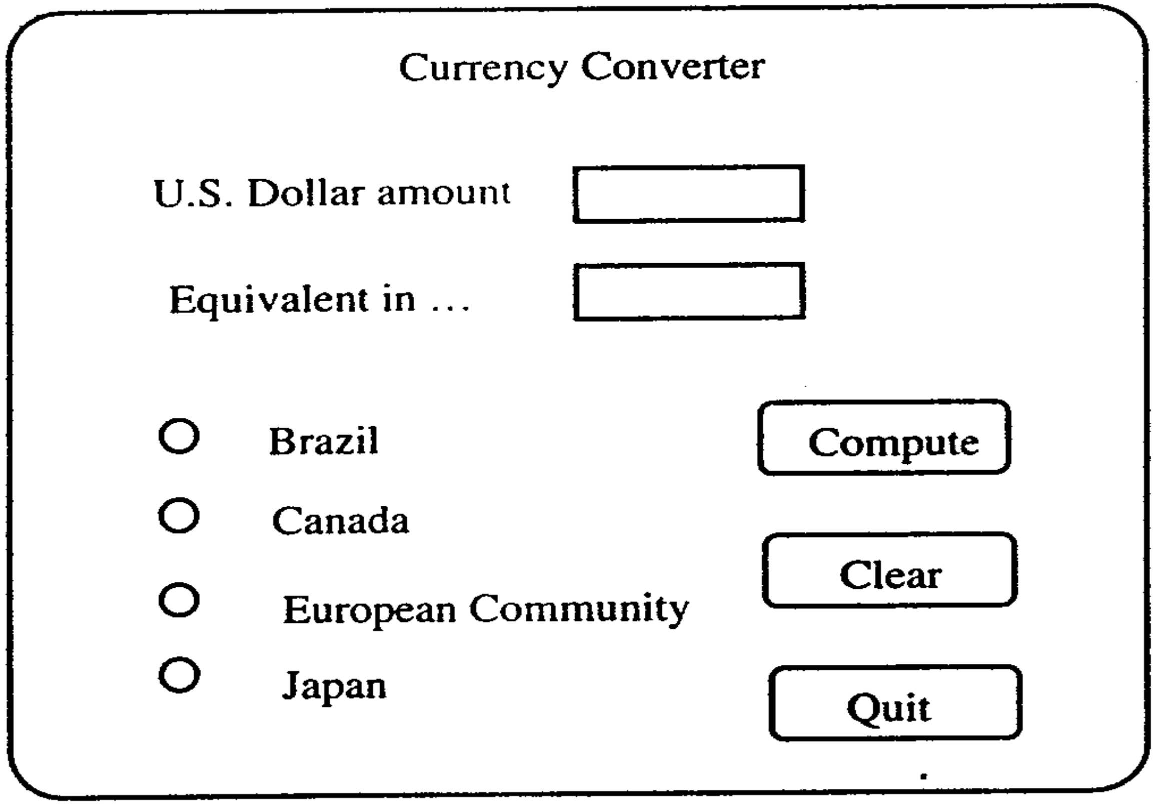
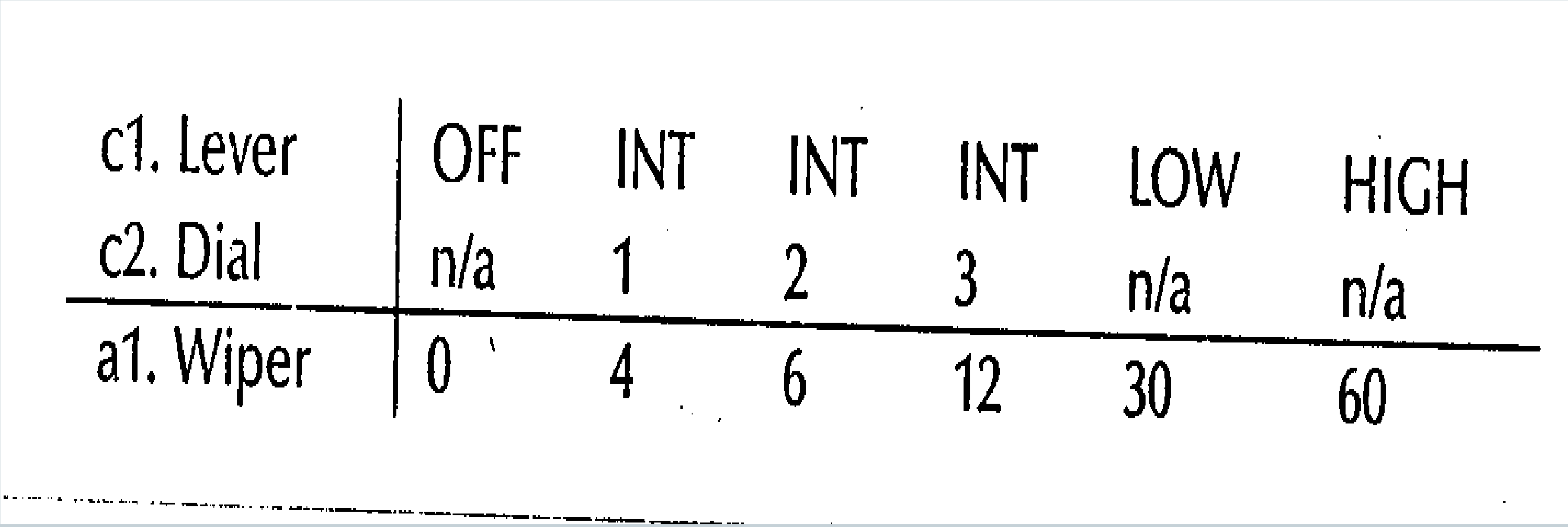
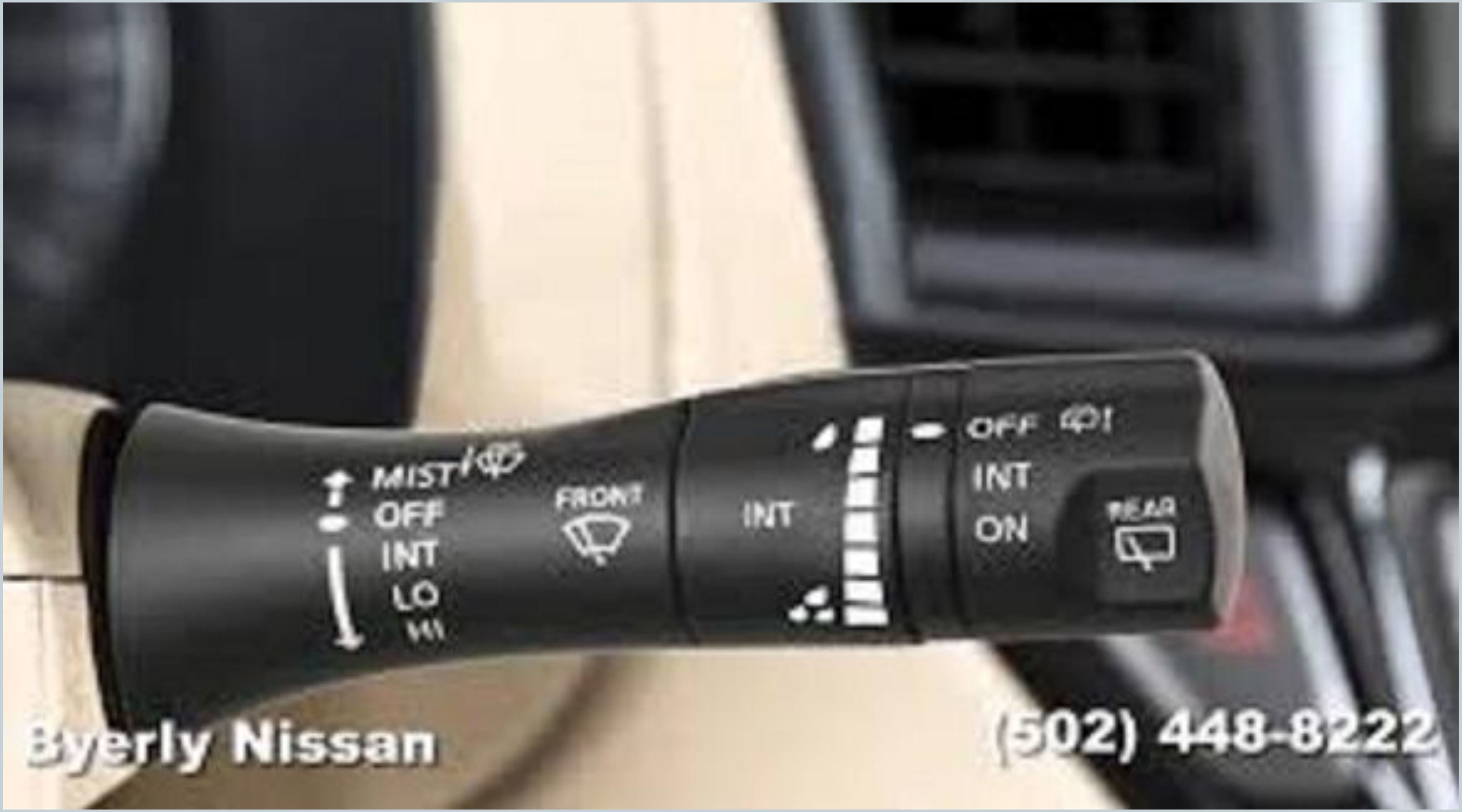


Figure 2.5 **Currency converter GUI.**

Saturn Windshield Wiper Controller



c1. Lever	OFF	INT	INT	INT	LOW	HIGH
c2. Dial	n/a	1	2	3	n/a	n/a
a1. Wiper	0	4	6	12	30	60


MIST
OFF
INT
LO
HI
FRONT
INT
OFF
INT
ON
REAR
Byerly Nissan
(502) 448-8222

Thank you ???

References

- Software Testing Craftsman’s Approach-Paul C Jorgensen.