# Path Testing

Dr. John H Robb, PMP, SEMC
UTA Computer Science and Engineering

# Program Graphs

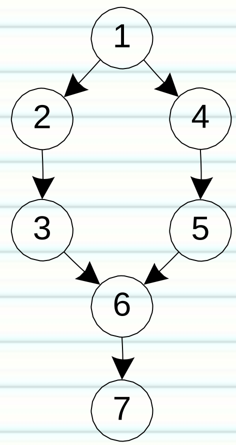
Definition: Given a program written in an imperative programming language, its *program graph* is a directed graph in which nodes are statement fragments, and edges represent flow of control. (A complete statement is a “default” statement fragment.)

We will refer to this as a <u>Control Flow Graph</u>

# Control Flow Graphs of Structured Programming Constructs

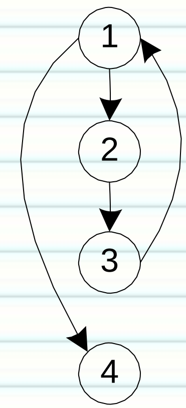
If-Then-Else

```
1    If <condition>
2        Then
3            <then statements>
4        Else
5            <else statements>
6    End If
7    <next statement>
```



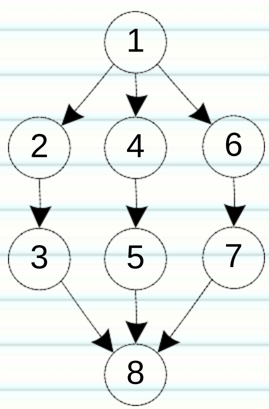

Pre-test Loop

```
1    While <condition>
2        <repeated body>
3    End While
4    <next statement>
```



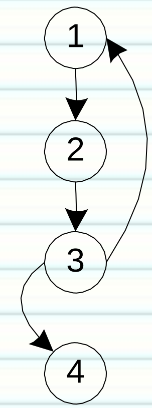

Case/Switch

```
1    switch (n)
2        n=1:
3            <case 1  statements>
4        n=2:
5            <case 2  statements>
6        n=3:
7            <case 3  statements>
8    End Case
```



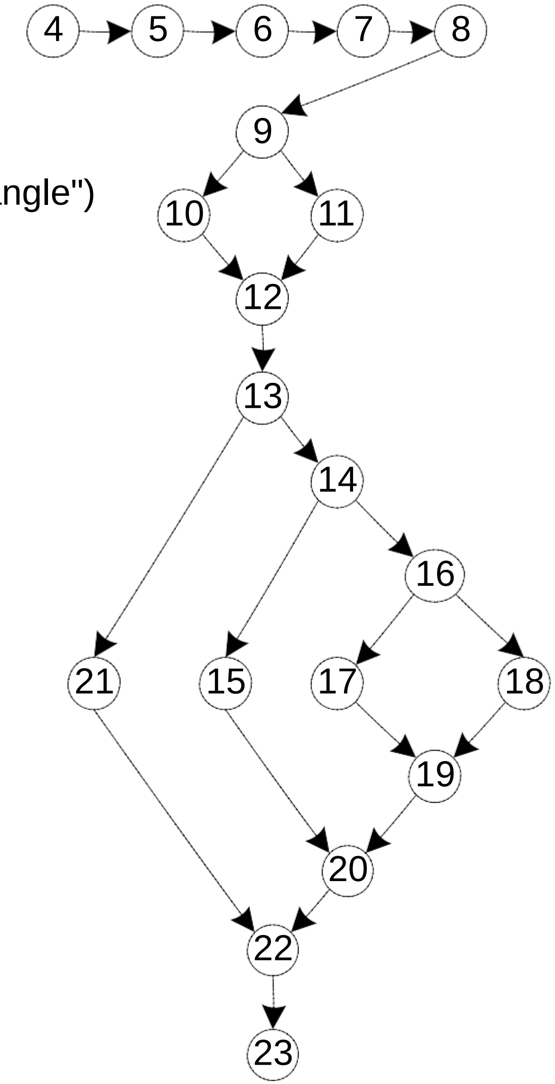

Post-test Loop

```
1    Do
2        <repeated body>
3    Until <condition>
4    <next statement>
```



# Sample Control Flow Graph

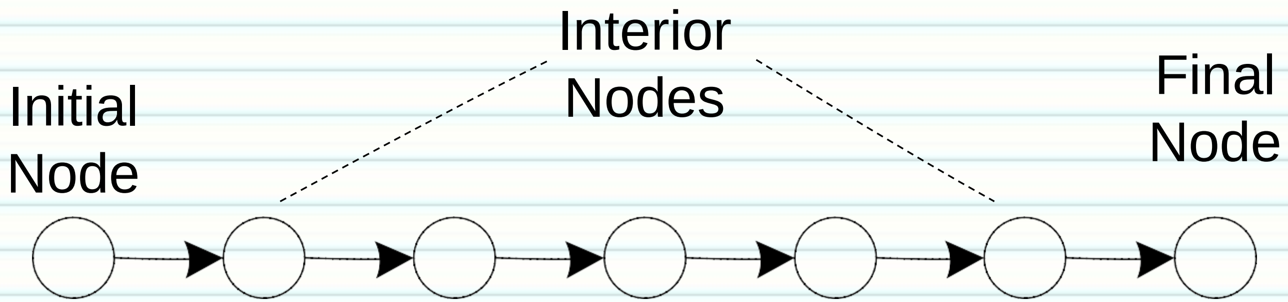
```
1  Program triangle2
2  Dim a,b,c As Integer
3  Dim IsATriangle As Boolean
4  Output("Enter 3 integers which are sides of a triangle")
5  Input(a,b,c)
6  Output("Side A is ",a)
7  Output("Side B is ",b)
8  Output("Side C is ",c)
9  If (a < b + c) AND (b < a + c) AND (c < a + b)
10    Then IsATriangle = True
11    Else IsATriangle = False
12  EndIf
13  If IsATriangle
14Then  If (a = b) AND (b = c)
15Then Output ("Equilateral") 16
17          Else    If (a≠b) AND (a≠c) AND (b≠c)
18                       Then  Output ("Scalene")
19                       Else  Output ("Isosceles")
20                  EndIf
21          EndIf
       Else    Output("Not a Triangle")
22  EndIf
23  End triangle2
```



# DD-Path Notation

# DD-Paths

- Originally defined by E. F. Miller (1977?)
- “DD” is short for “decision to decision”
- Original definition was for early (second generation) programming languages
- Similar to a “chain” in a directed graph
- Bases of interesting test coverage metrics




**Key point: a DD-path graph is a reduced Control Flow Graph!**

# DD-Paths

A *DD-Path* (decision-to-decision) is a chain in a program graph such that

Case 1: it consists of a single node with indeg = 0, (initial node)
Case 2: it consists of a single node with outdeg = 0, (terminal node)
Case 3: it consists of a single node with indeg ≥ 2 or outdeg ≥ 2, (decisional node)
Case 4: it consists of a single node with indeg = 1 and outdeg = 1, (non-decisional node)
Case 5: it is a maximal chain of length ≥ 1.

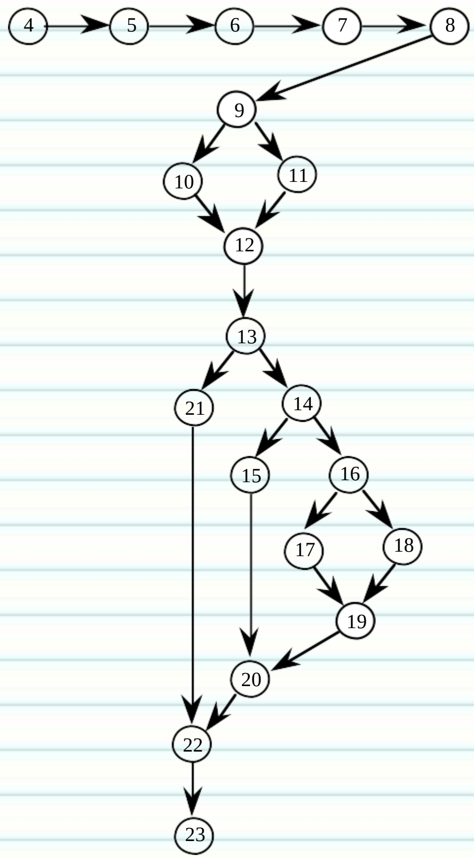
# DD-Path Graph

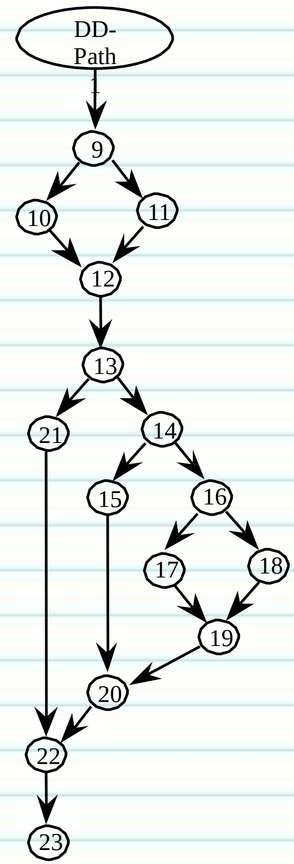
Given a program written in an imperative language, its *DD-Path graph* is the directed graph in which nodes are DD-Paths of its program graph, and edges represent control flow between successor DD-Paths.

- a form of condensation graph
- 2-connected components are collapsed into an individual node
- single node DD-Paths (corresponding to Cases 1 - 4 ) preserve the convention that a statement fragment is in exactly one DD-Path

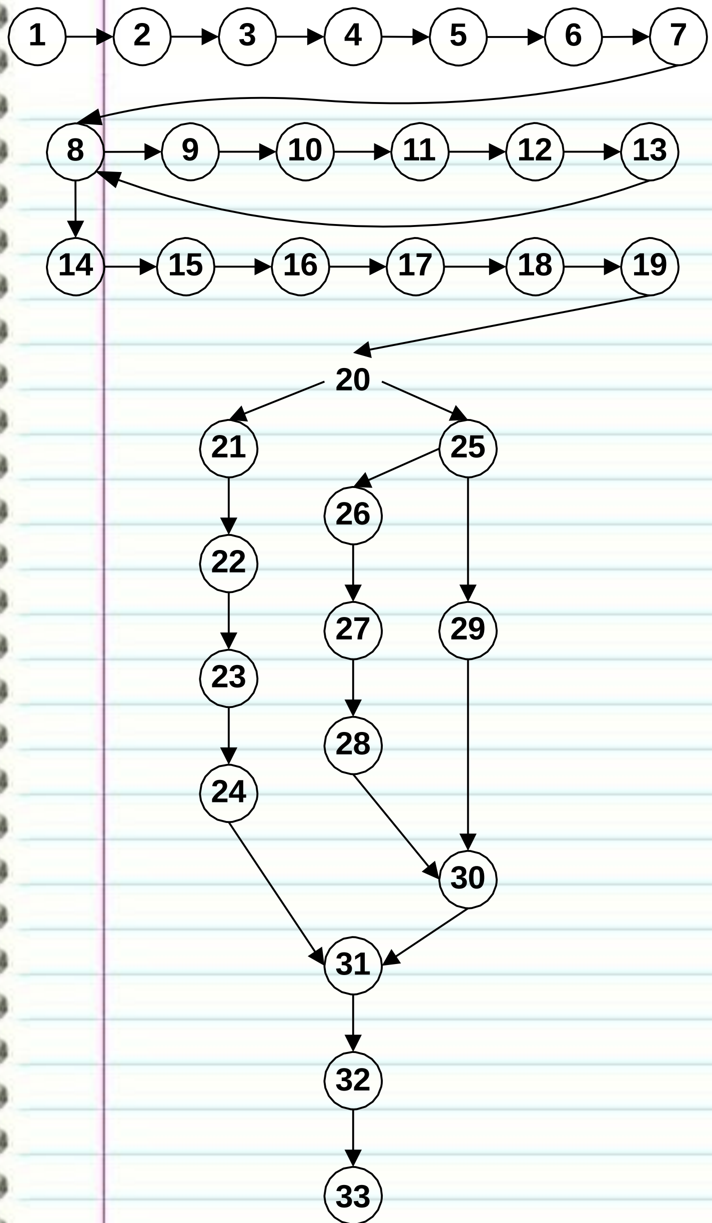
# DD-Path Graph of the Triangle Program

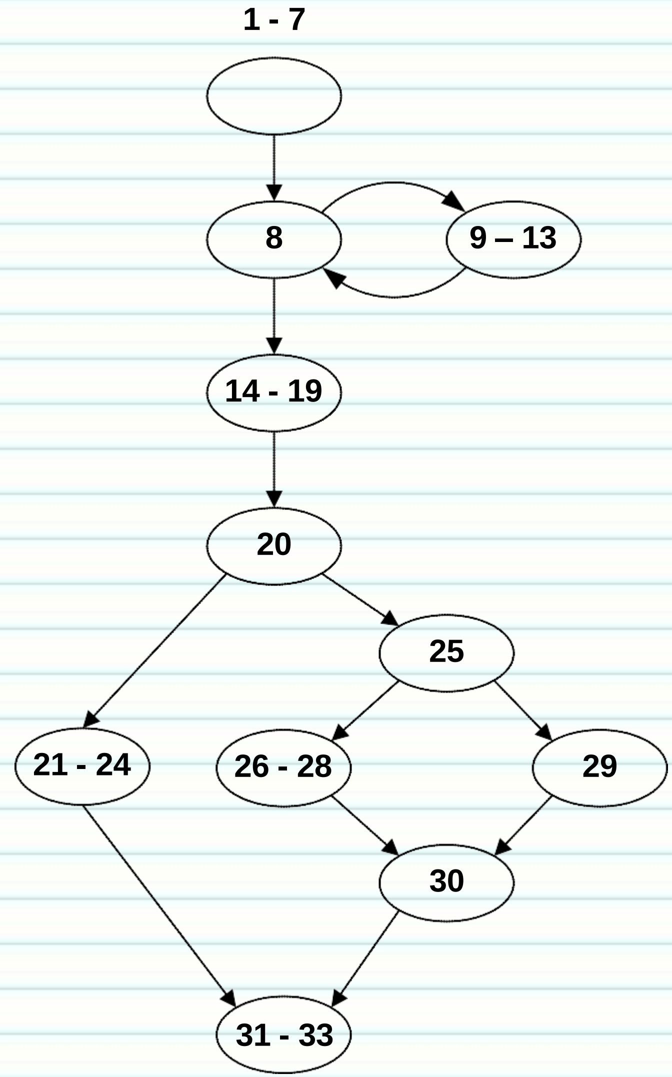
## (not much compression because this example is control intensive, with little sequential code.)



# DD-Path Graph



# Basis Path Testing

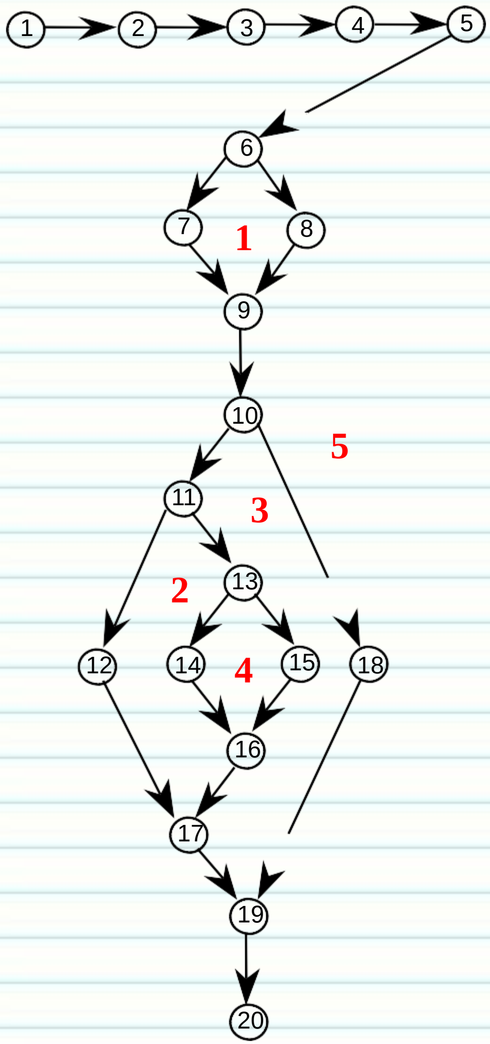
# Cyclomatic Complexity

- In a strongly connected directed graph G, its *cyclomatic complexity*, denoted by V(G), is given by $V(G) = e – n + p$, where
  - e is the number of edges in G
  - n is the number of nodes in G
  - p is the number of connected regions in G
- In code that conforms to structured programming (single entry, single exit), $p = 1$.
- In a directed graph that is not strongly connected, $V(G) = e – n + 2p$.

# Cattle Pens and Cyclomatic Complexity

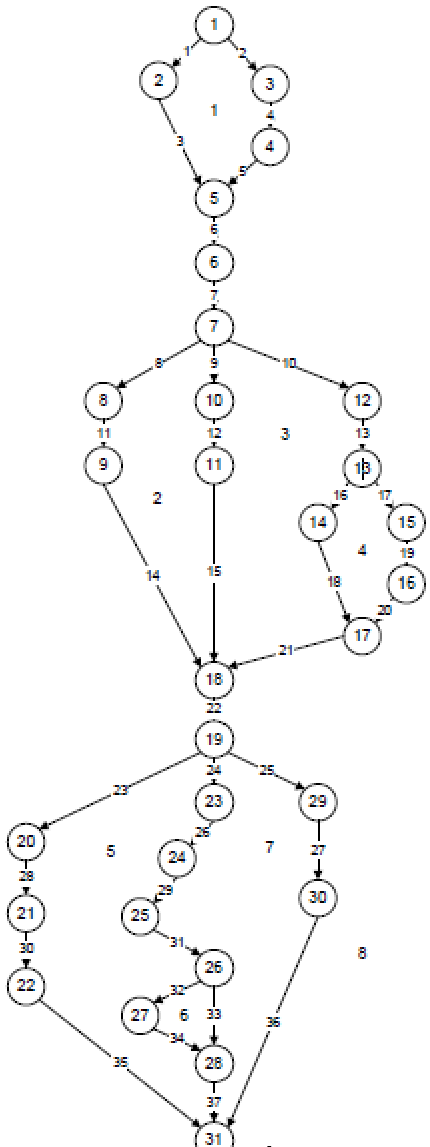
- Cyclomatic complexity describes the number of "enclosed regions" in a directed graph.
- Given a directed graph drawing, to compute V(G) without counting nodes, edges, and connected regions, consider the nodes to be fence posts, and edges are 4-wire fences. Then the number of cattle pens is V(G). (The outside is also a cattle pen.)

# Cattle Pens and Cyclomatic Complexity




$$
\begin{aligned}
V(G) &= e - n + 2p \\
&= 23 - 20 + 2 \\
&= 5
\end{aligned}
$$

# Cattle Pens and Cyclomatic Complexity




Another example…

V(G) = e – n + 2p

= 37 – 31 + 2

= 8

# Another Example—NextDate

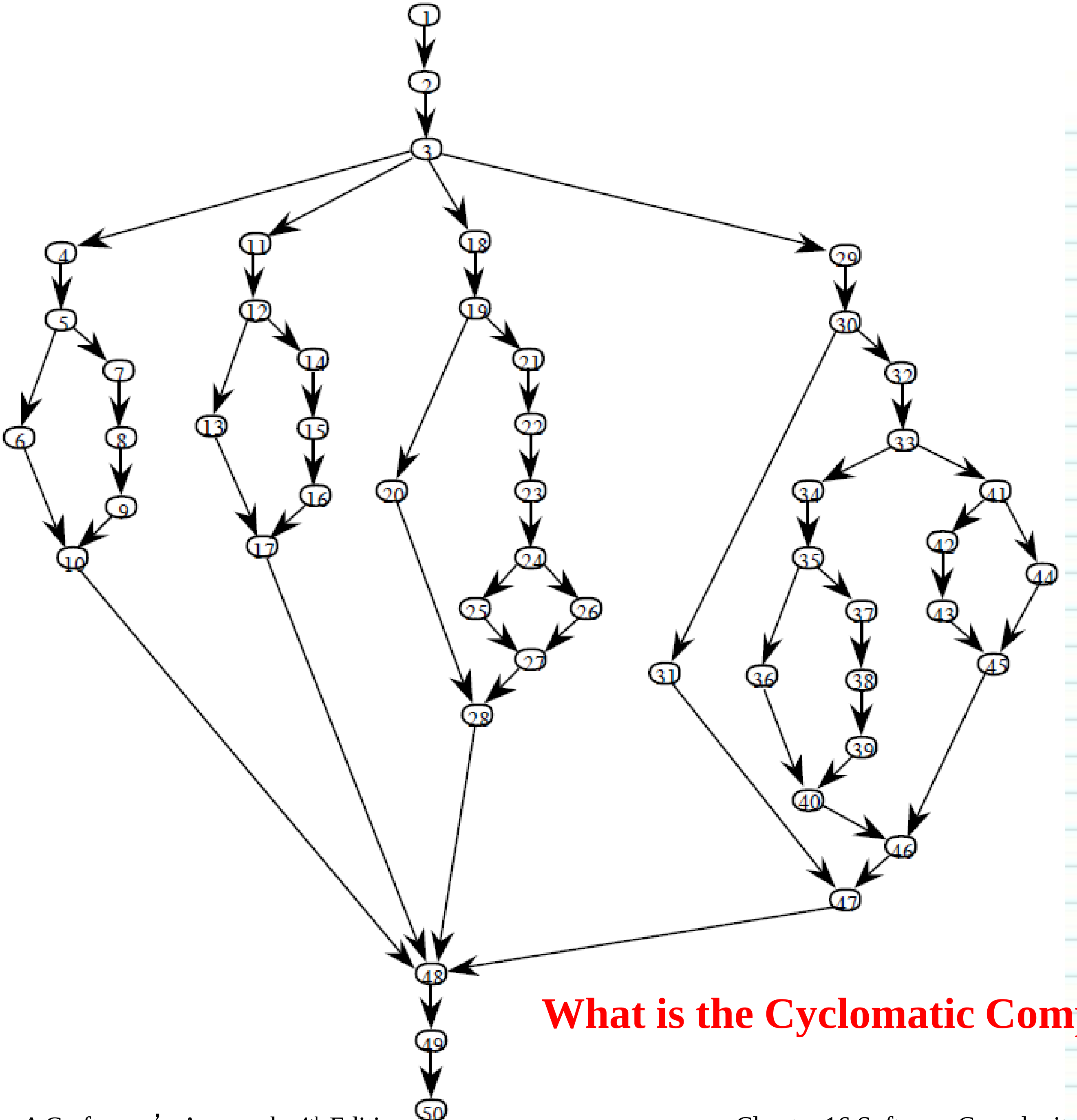
- Pseudo-code on the next four slides
- Program graph after pseudo-code

```
Program NextDate
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
	EndIf
```

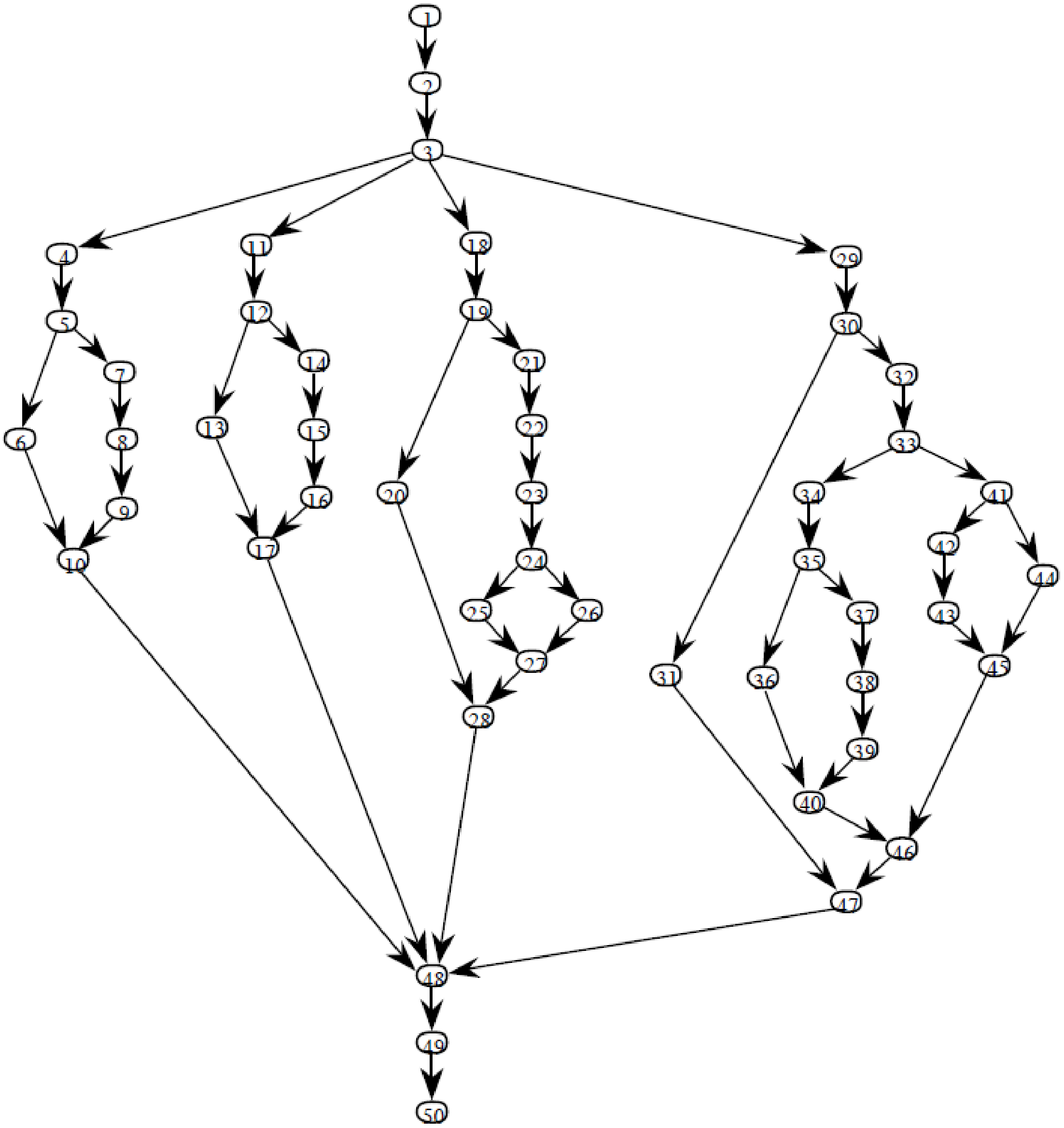
```
  Case 4: month is 2: 'February
If day < 28
          Then tomorrowDay = day + 1
          Else
If day = 28
            Then   If (isLeap)
                     Then tomorrowDay = 29 'leap year
                     Else      'not a leap year
                               tomorrowDay = 1
        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

```
        Else    If day = 29
                     Then  If (isLeap)
                                 Then  tomorrowDay = 1
                                       tomorrowMonth = 3
                                 Else  'not a leap year
                                       Output("Cannot have Feb.", day)
                                EndIf
                        EndIf
        EndIf
   EndIf
EndCase
Output ("Tomorrow's date is", tomorrowMonth, tomorrowDay, tomorrowYear)
End NextDate
```




**What is the Cyclomatic Complexity?**

# Find V(G) for the Program Graph of NextDate

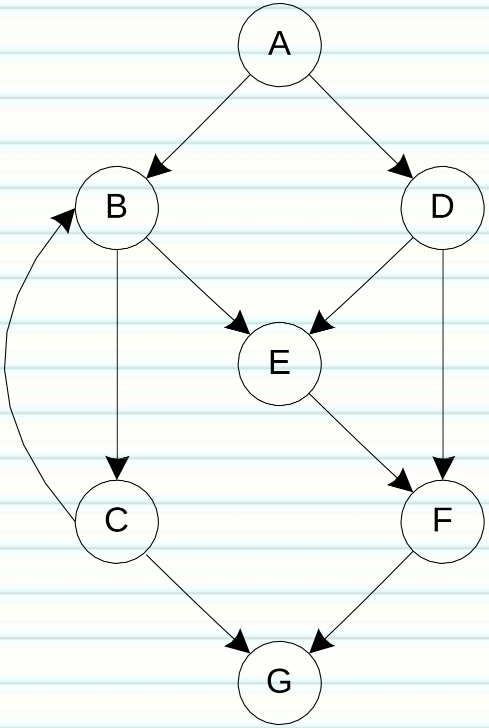



e = 60
n = 50
p = 1

V(G) = 60 – 50 + 2
= 12

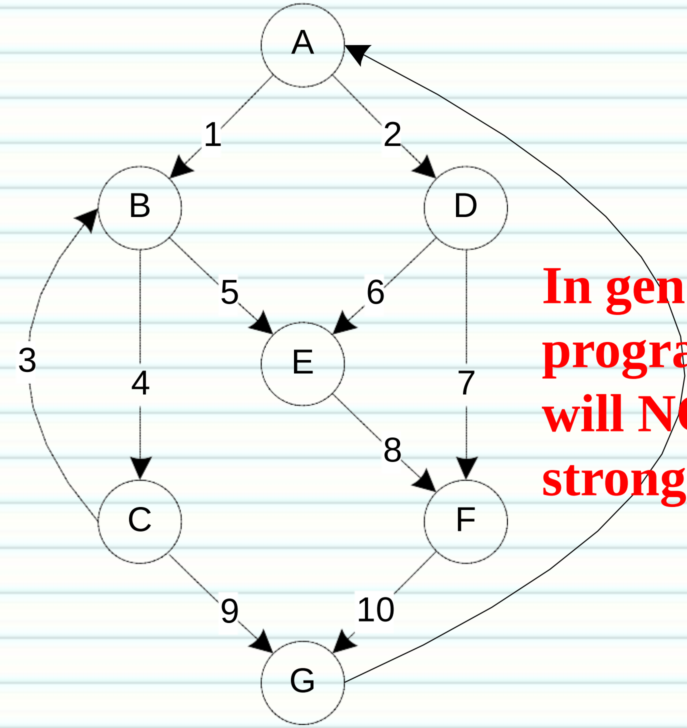
# McCabe's Example

## McCabe's Original Graph




$V(G) = 10 - 7 + 2(1)$
$= 5$

## Derived, Strongly Connected Graph




$V(G) = 11 - 7 + 1$
$= 5$

**In general, program graphs will NOT be strongly connected**

# Basis Path Method

- Perform the following steps:

1. Draw the reduced CFG
2. Determine the Cyclomatic Complexity (from the reduced CFG)
3. Develop the basis paths
   a. Define an initial basis path (pick any path) - identify the basis path
   b. Then "flip" each decision starting with the first - one at a time - capture each derived path from the basis path
   c. Stop when you have reached the Cyclomatic Complexity v(g)
4. Determine the inputs required to traverse each basis path and the expected outputs for each. You should have v(g) test cases.

# Simple Boundary Value Problem

```
public static double guessWhat(double input) {

      double result;

      if (input<0.0)
            result = -input;
      else
            result = input;

      return result;
}
```

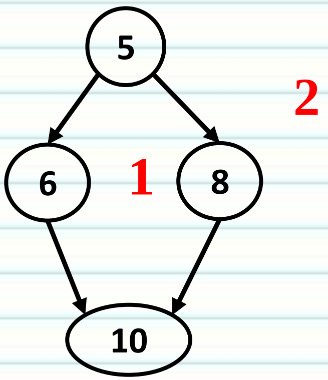
**Student exercise**

1. **Describe what the code seems to be doing**
2. **Draw the reduced CFG (DD-path)**
3. **Determine the Cyclomatic complexity**
4. **Develop the equivalence partitions**
5. **Develop the basis paths using the Cyclomatic complexity**
6. **Develop the test cases using required input values to achieve boundary value coverage**
7. **Do the test cases and outputs refute or support the code functional description?**

(c) JRCS 2016

# Answer

1. The code returns the absolute value of the input
2. The DD-path graph is shown on the right
3. Cyclomatic complexity is 2. Verified with code
4. Basis paths are
   a. 5,6,10
   b. 5,8,10




5. Equivalence partitions are (ignore robustness values)
   a. Note that the significance is not specified – assume 0.01
   b. -min <= input <0.0 (use -0.01)
   c. 0.0 <= input <= max (use 0.0)
6. Test cases (using basis paths and boundary values)
   a. Test case 1: input=-0.01, guessWhat=0.01
   b. Test case 2: input=0.0, guessWhat=0.0
7. Test cases support program description

Note that we do not need to create 100's of test cases incrementing input by 0.01 – we have done this with two test cases

# Another Basis Path Example

```
public static boolean getres(boolean reservation,boolean tableavailable) {

    boolean result;

    if (reservation)
        result = true;
    else
        if (tableavailable)
            result = true;
        else
            result = false;

    return result;

}
```

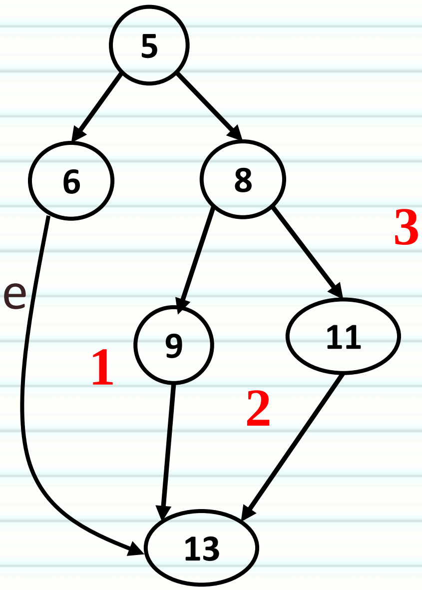
**Student exercise**

1. **Describe what the code seems to be doing**
2. **Draw the reduced CFG (DD-path)**
3. **Determine the Cyclomatic complexity**
4. **Develop the basis paths using the Cyclomatic complexity. Use MC/DC to select basis paths**
5. **Develop the test cases using required input values to achieve boundary value coverage**
6. **Do the test cases and outputs refute or support the code functional description?**

(c) JRCS 2016

# Answers

1. The method is providing the following result
   - getres = reservation || tableavailable
2. The DD-path graph is shown to the right
3. Cyclomatic complexity is 3 from the graph
4. Since this is an || relationship we need to use
   - FF,FT,TF as the inputs
   - TF gives us 5,6,13
   - FT gives us 5,8,9,13
   - FF gives us 5,8,11,13
5. Test cases are
   - TC 1: reservation = true, tableavailable = false, getres = true
   - TC 2: reservation = false, tableavailable = true, getres = true
   - TC 3: reservation = false, tableavailable = false, getres = false
6. Yes the test cases and expected results support the description



# Working Examples (1)

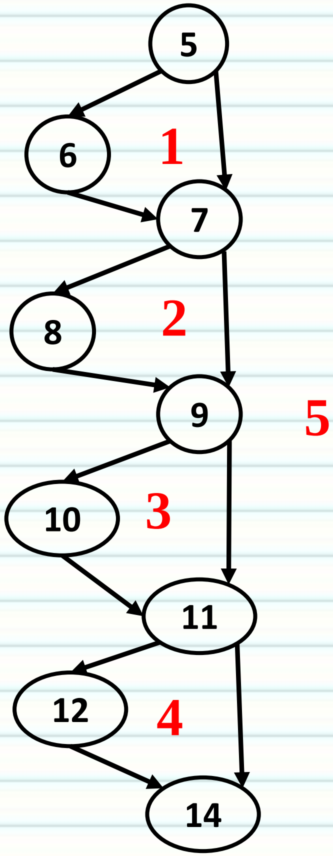
```
public int countNeg(int i,int j,int k,int l) {

int s=0;

if (i < 0)
    s++;
if (j < 0)
    s++;
if (k < 0)
    s++;
if (l < 0)
    s++;

return s;
}
```

**Student exercise**

1. **Describe what the code seems to be doing**
2. **Draw the reduced CFG**
3. **Determine the Cyclomatic complexity**
4. **Develop the basis paths using the Cyclomatic complexity**
5. **Develop the test cases using required input values to achieve boundary value coverage**
6. **Determine coverage achieved**
7. **Do the test cases and outputs refute or support the code functional description?**

(c) JRCS 2016

# Working Examples (1) - Answer




1. The software attempts to count the number of parameters that have a negative value.
2. The reduced CFG is shown to the left with the corresponding line numbers shown in each node.
3. The cyclomatic complexity is counted to the left in red – it is 5 – requiring 5 tests cases for basis path coverage.

| Test | Inputs | | | | Exp Out | |
|---|---|---|---|---|---|---|
| Case | i | j | k | l | Return | Basis Path |
| 1 | -1 | -1 | -1 | -1 | 4 | 5-6-7-8-9-10-11-12-14 |
| 2 | 0 | -1 | -1 | -1 | 3 | 5-7-8-9-10-11-12-14 |
| 3 | 0 | 0 | -1 | -1 | 2 | 5-7-9-10-11-12-14 |
| 4 | 0 | 0 | 0 | -1 | 1 | 5-7-9-11-12-14 |
| 5 | 0 | 0 | 0 | 0 | 0 | 5-7-9-11-14 |

6. Full statement coverage is achieved with the basis paths selected. WHY?
7. The test cases support the functional description.

(c) JRCS 2016

# More Basis Path Examples

```
public static double calculatePostageDue (int weight) {
	double postage_due;

	if (weight<=1)
		postage_due = 0.49;
	else
		if (weight<=2)
			postage_due = 0.70;
		else
			postage_due = 0.99;

	return postage_due;

	}
```

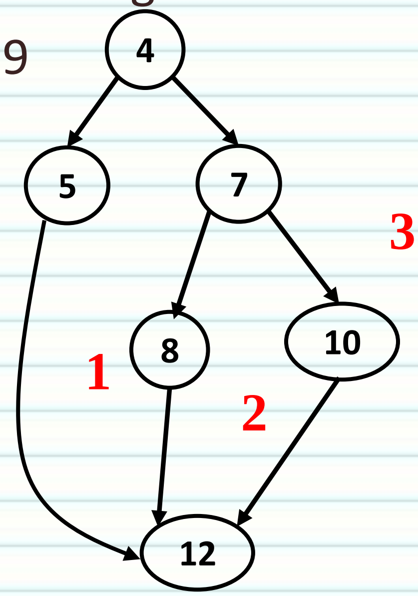
**Student exercise**

1. **Describe what the code seems to be doing**
2. **Draw the reduced CFG (DD-path)**
3. **Determine the Cyclomatic complexity**
4. **Develop the basis paths using the Cyclomatic complexity**
5. **Develop the equivalence partitions**
6. **Develop the test cases using required input values to achieve boundary value coverage**
7. **Do the test cases and outputs refute or support the code functional description?**

(c) JRCS 2016

# Answer

1. The program calculates postage according to the following rates:
   <= 1 (Oz presumably) = $0.49, 2 = $0.70, >=3 = $0.99
2. See DD-path graph on right
3. Cyclomatic complexity = 3, double check confirms
4. Basis paths
   a. 4, 5, 12
   b. 4, 7, 8, 12
   c. 4, 7, 10, 12
5. Partitions (ignore robustness test values)
   a. -min <= weight <= 1
   b. weight = 2
   c. 3 <= weight <= max
6. Test cases
   1. Weight = 1, postage=0.49
   2. Weight = 2, postage=0.70
   3. Weight = 3, postage=0.99
7. Test cases and expected outputs support description above




(c) JRCS 2016

# Extend the Previous Example

**Use your results from the previous answer to develop this**

```
public static double calculatePostageDue (int weight) {
      double postage_due;

      if (weight<=1)
            postage_due = 0.49;
      else
            if (weight<=2)
                  postage_due = 0.70;
            else
                  if (weight <=3)
                        postage_due = 0.91;
                  else
                        postage_due = 1.12;
      return postage_due;

      }
```

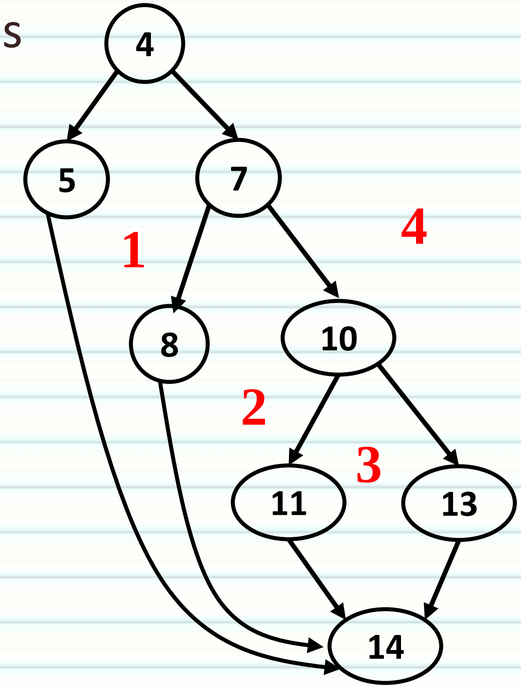
**Student exercise**

1. **Describe what the code seems to be doing**
2. **Draw the reduced CFG (DD-path)**
3. **Determine the Cyclomatic complexity**
4. **Develop the basis paths using the Cyclomatic complexity**
5. **Develop the equivalence partitions**
6. **Develop the test cases using required input values to achieve boundary value coverage**
7. **Do the test cases and outputs refute or support the code functional description?**

(c) JRCS 2016

# Answer

1. The program calculates postage according to the following rates:
   <= 1 (Oz presumably) = $0.49, 2 = $0.70, 3 = $0.91, >=4 = $1.12
2. See DD-path graph on right
3. Cyclomatic complexity = 4, double check confirms
4. Basis paths
   a. 4, 5, 14
   b. 4, 7, 8, 14
   c. 4, 7, 10, 11, 14
   d. 4, 7, 10, 13, 14
5. Partitions (ignore robustness test values)
   a. -min <= weight <= 1
   b. weight = 2
   c. weight = 3
   d. 4 <= weight <= max



# Answer (cont.)

6. Test cases
    1. Weight = 1, postage=0.49
    2. Weight = 2, postage=0.70
    3. Weight = 3, postage=0.91
    4. Weight = 4, postage=1.12

7. Test cases and expected outputs support description above
8. Developing test cases is where we find the most defects, typically not executing the tests.
    a. In this example we detect some strangeness in the code, that we would fix by replacing some of the <= with =

# A Little More Complicated Example

- This example is from the book "How We Test Software at Microsoft"
- From the Gregorian calendar example, the dates of 10/5/-10/14/1582 were excluded from the calendar
- The book writes this as:

```
if (year == 1582 && month ==10 && !(day<5) && !(day > 14))
    return true;
else
    return false;
```

**Is there a better way to write the last two conditions?**

```
if (year == 1582 && month ==10 && day>=5 && day <= 14)
```

# A Little More Complicated Example (cont.)

- This is the solution from the book - what is wrong with this solution?

*Table 6-1. Truth Table for IsInvalidGregorianCalendarDate Function*

| Tests | Parameters | | | Conditional clauses | | | | Expected result |
|---|---|---|---|---|---|---|---|---|
| | Month | Day | Year | Year | Month | !(day < 5) | !(day > 14) | |
| 1 | 10 | 11 | 1582 | True | True | True | True | True |
| 2 | 10 | 21 | 1582 | True | True | True | False | False |
| 3 | 10 | 3 | 1582 | True | True | False | | False |
| 4 | 5 | 7 | 1582 | True | False | | | False |
| 5 | 10 | 5 | 1994 | False | | | | False |

- This turns out to be a really good exercise. If we use MC/DC how many test cases is this - how many conditions are in the expression?

  if (year == 1582 && month ==10 && day>=5 && day <= 14)

# A Little More Complicated Example (cont.)

- For Modules M09-M10 we will work with Single Condition Multiple Decision statements - we’re not ready for Multiple Condition Decisions yet

```
result=false;
if (year == 1582)
    if (month==10)
        if (day>=5)
            if (day <= 14)
                result=true;
```

**Student exercise**

1. **Draw the reduced CFG**
2. **Determine the Cyclomatic complexity**
3. **Develop the basis paths using the Cyclomatic complexity**
4. **Develop the test cases using required input values to achieve boundary value coverage**
5. **What coverage have we achieved?**

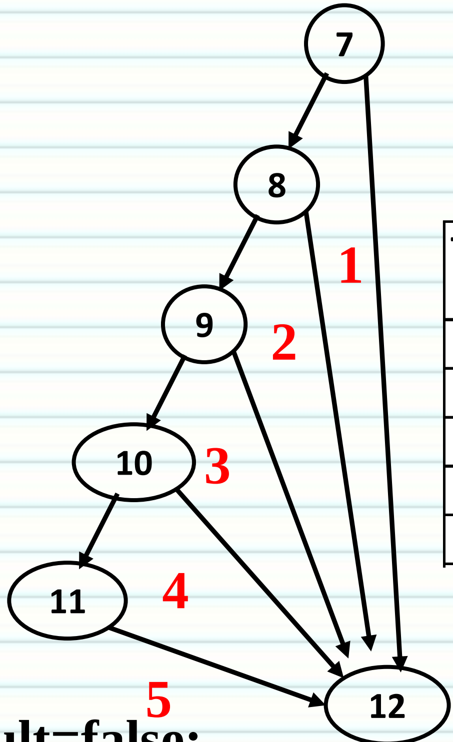
Use this guide:

| | day >= 5 | day <= 14 |
|---|---|---|
| T | 5 | 14 |
| F | 4 | 15 |

(c) JRCS 2016

# A Little More Complicated Example (cont.)

- CFG




| | day >= 5 | day <= 14 |
|---|---|---|
| T | 5 | 14 |
| F | 4 | 15 |

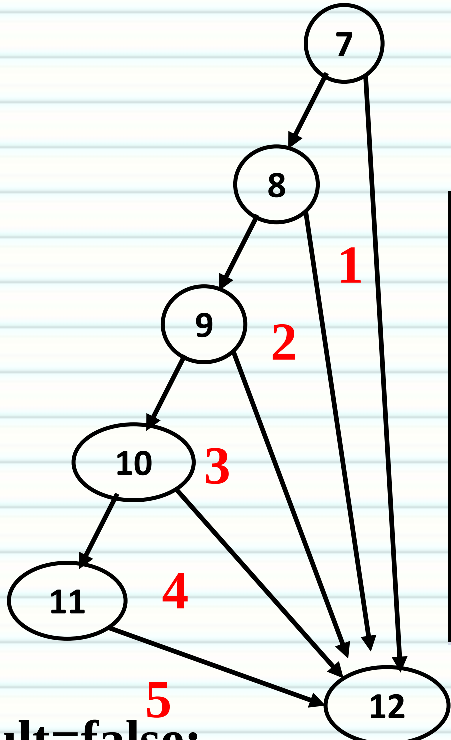
| Test Case Number | Inputs | | | Exp Out | Basis Path |
|---|---|---|---|---|---|
| | Year | Month | Day | Return | |
| 1 | 1582 | 10 | 5 | TRUE | 7-8-9-10-11-12 |
| 2 | 1581 | 10 | 5 | FALSE | 7-12 |
| 3 | 1582 | 11 | 5 | FALSE | 7-8-12 |
| 4 | 1582 | 10 | 4 | FALSE | 7-8-9-12 |
| 5 | 1582 | 10 | 15 | FALSE | 7-8-9-10-12 |

```
6     result=false;
7     if (year == 1582)
8         if (month==10)
9             if (day>=5)
10                if (day <= 14)
11                    result=true;
12    return result;
```

**Have we achieved coverage?**

# A Little More Complicated Example (cont.)

- CFG




| | day >= 5 | day <= 14 |
|---|---|---|
| T | 5 | 14 |
| F | 4 | 15 |

| Test Case Number | Inputs | | | Exp Out | Basis Path |
|---|---|---|---|---|---|
| | Year | Month | Day | Return | |
| 1 | 1582 | 10 | 5 | TRUE | 7-8-9-10-11-12 |
| 2 | 1581 | 10 | 5 | FALSE | 7-12 |
| 3 | 1582 | 11 | 5 | FALSE | 7-8-12 |
| 4 | 1582 | 10 | 4 | FALSE | 7-8-9-12 |
| 5 | 1582 | 10 | 15 | FALSE | 7-8-9-10-12 |
| 6 | 1583 | 10 | 5 | FALSE | - |
| 7 | 1582 | 9 | 5 | FALSE | - |
| 8 | 1582 | 10 | 14 | TRUE | - |

```
result=false;
if (year == 1582)
    if (month==10)
        if (day>=5)
            if (day <= 14)
                result=true;
```

**We have achieved Decision, Statement, BV and Extreme Range coverage**

**Notice how we're changing only one input at a time from the all true path**

# A Simpler Homework Problem

```
public void setAlerts (double fuel_level) {

        setChime(false); setRedLight(false); setYellowLight(false); setGreenLight(false);
        if (fuel_level <= 15.0)
            setChime(true);
        else
            if (fuel_level<30.0)
                setRedLight(true);
            else
                if (fuel_level<=40.0)
                    setYellowLight(true);
                else
                    if (fuel_level<50.0)
                            setGreenLight(true);
        }
```

Assume fuel_level is significant to 0.1 and ranges from 0.0 to 100.0 gallons

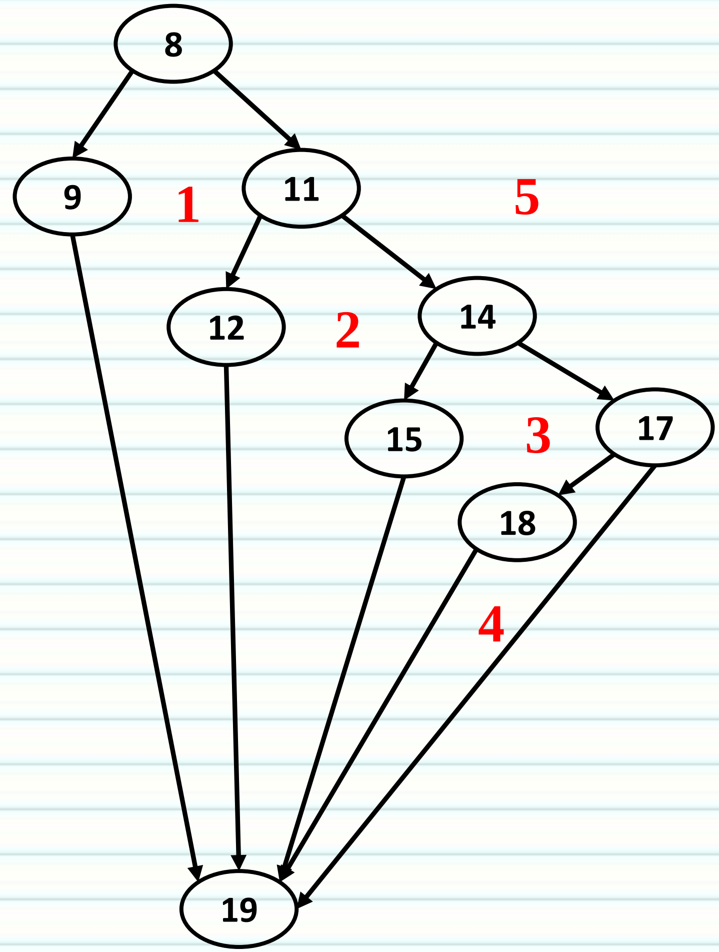
# A Simpler Homework Problem (cont.)

**Corresponding Decision table "code description"**

| fuel_level in gallons | Rule 1 | Rule 2 | Rule 3 | Rule 4 | Rule 5 |
|---|---|---|---|---|---|
| **CONDITIONS** | | | | | |
| 0.0 <= fuel_level <= 15.0 | Y | | | | |
| 15.1 <= fuel_level <= 29.9 | | Y | | | |
| 30.0 <= fuel_level <= 40.0 | | | Y | | |
| 40.1 <= fuel_level <= 49.9 | | | | Y | |
| 50.0 <= fuel_level <= 100.0 | | | | | Y |
| **ACTIONS** | | | | | |
| chime | **TRUE** | FALSE | FALSE | FALSE | FALSE |
| redLight | FALSE | **TRUE** | FALSE | FALSE | FALSE |
| yellowLight | FALSE | FALSE | **TRUE** | FALSE | FALSE |
| greenLight | FALSE | FALSE | FALSE | **TRUE** | FALSE |
| | | | | | |
| Table implements a "first-of" rule | | | | | |

# A Simpler Homework Problem (cont.)

**Corresponding CFG and ECP/BV**




**fuel_level (all values are gallons):**

| | | | | |
|---|---|---|---|---|
| 0.0 15.0 | 15.1 29.9 | 30.0 40.0 | 40.1 49.9 | 50.0 100.0 |

# A Simpler Homework Problem (cont.)

| Test case number | Inputs | Expected Outputs | | | | Basis Path |
|---|---|---|---|---|---|---|
| | fuel_level (gallons) | chime | redLight | yellowLight | greenLight | |
| 1 | 15.0 | **TRUE** | FALSE | FALSE | FALSE | 8-9-19 |
| 2 | 29.9 | FALSE | **TRUE** | FALSE | FALSE | 8-11-12-19 |
| 3 | 40.0 | FALSE | FALSE | **TRUE** | FALSE | 8-11-14-15-19 |
| 4 | 49.9 | FALSE | FALSE | FALSE | **TRUE** | 8-11-14-17-18-19 |
| 5 | 50.0 | FALSE | FALSE | FALSE | FALSE | 8-11-14-17-19 |
| 6 | 15.1 | FALSE | **TRUE** | FALSE | FALSE | - |
| 7 | 30.0 | FALSE | FALSE | **TRUE** | FALSE | - |
| 8 | 40.1 | FALSE | FALSE | FALSE | **TRUE** | - |
| 9 | 0.0 | **TRUE** | FALSE | FALSE | FALSE | - |
| 10 | 100.0 | FALSE | FALSE | FALSE | FALSE | - |

**Notice that the first 5 test cases must be in this order. Test cases after this can be in any order.**

**Notice that we have to add 3 test cases to the basis path to test all BVs and we add the 2 extreme range tests to get a total of 10 test cases.**

**We have designed our test cases to give us decision and statement coverage (guaranteed with basis path), full BV coverage and extreme range coverage.**

# Drawing CFGs on more complicated problems

```
public boolean codeDrawing (int points) {
    boolean result;
    if (points > 40) {
        result=true;
        count++;}
    else {
        result=false;
        if (points < 25) {
            transactions++;
            if (count > 1) {
                result = true;
                transactions=0; }
        }
}
```

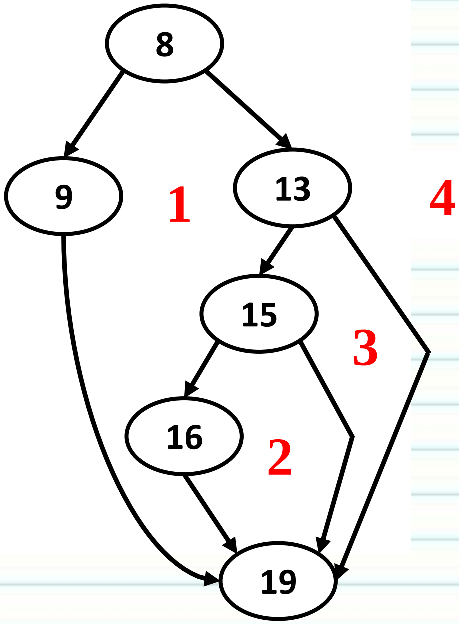
**Assume points ranges from 0 to 100 (both inclusive)**

**Do not worry about extreme ranges for count or transactions**

# Drawing CFGs on more complicated problems (cont.)

```
6  public boolean codeDrawing (int points) {
7      boolean result;
8      if (points > 40) {
9          result=true;
10         count++;}
11     else {
12         result=false;
13         if (points < 25) {
14             transactions++;
15             if (count > 1) {
16                 result = true;
17                 transactions=0; }
18         }
19     }
```

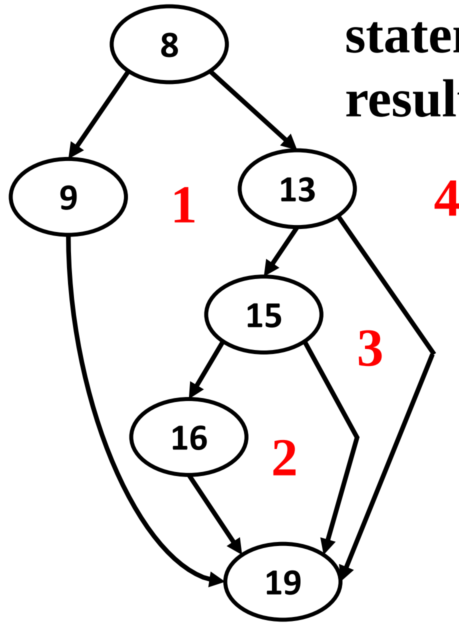
1. The chain at 9-10 is absorbed into 9
2. The assignment at 12 is absorbed into 13
3. The assignment at 14 is absorbed into 15
4. The chain at 16-17 is absorbed into 16




**Nodes are either decisions, merges, or assignments. Nodes are only assignments when they cannot be absorbed into a decision**

# Drawing CFGs on more complicated problems (cont.)

```
6  public boolean codeDrawing (int points) {
7      boolean result;
8      if (points > 40) {
9          result=true;
10         count++;}
11     else {
12         result=false;
13         if (points < 25) {
14             transactions++;
15             if (count > 1) {
16                 result = true;
17                 transactions=0; }
18         }
19     }
```




**statement 15 ECP/BV result (all values are points):**

| T | F | T |
|---|---|---|
| 0 24 | 25 40 | 41 100 |

| Test case number | Inputs | | | Expected Outputs | | | Basis Path |
|---|---|---|---|---|---|---|---|
| | points | count | transactions | result | count | transactions | |
| 1 | 41 | 0 | 0 | **TRUE** | **1** | 0 | 8-9-19 |
| 2 | 24 | 2 | 0 | **TRUE** | 2 | 0 | 8-13-15-16-19 |
| 3 | 25 | 2 | 0 | FALSE | 2 | 0 | 8-13-19 |
| 4 | 24 | 1 | 0 | FALSE | 1 | **1** | 8-13-15-19 |
| 5 | 40 | 1 | 0 | FALSE | 1 | 0 | - |
| 6 | 0 | 1 | 0 | FALSE | 1 | **1** | - |
| 7 | 100 | 1 | 0 | **TRUE** | **2** | 0 | - |

# Drawing CFGs on more complicated problems (cont.)

**Strictly speaking, this is the corresponding decision table. Note that I have allowed you to simplify this on the homework.**

| | Rule 1 | Rule 2 | Rule 3 | Rule 4 | Rule 5 | Rule 6 | Rule 7 | Rule 8 | Rule 9 |
|---|---|---|---|---|---|---|---|---|---|
| **CONDITIONS** | | | | | | | | | |
| 0 <= points <= 24 | Y | Y | Y | | | | | | |
| 25 <= points <= 40 | | | | Y | Y | Y | | | |
| 41 <= points <= 100 | | | | | | | Y | Y | Y |
| count = 0 | Y | | | Y | | | Y | | |
| count = 1 | | Y | | | Y | | | Y | |
| count >= 2 | | | Y | | | Y | | | Y |
| | | | | | | | | | |
| **ACTIONS** | | | | | | | | | |
| result | FALSE | FALSE | **TRUE** | FALSE | FALSE | FALSE | **TRUE** | **TRUE** | **TRUE** |
| count | count | count | count | count | count | count | count++ | count++ | count++ |
| transactions | transactions++ | transactions++ | 0 | transactions | transactions | transactions | transactions | transactions | transactions |

Table implements a "first-of" rule

**Also note that there is no way to represent this code as a series of multiple condition single decision statements because of the variable transactions - it has three possible values.**

# A Typical Homework Problem

- We are not yet ready to address multiple condition decision statements yet
- Many of the homework problems provide a multiple decision single condition snippets of code instead (like the previous Gregorian snippet)
- For these multiple decision, single condition homework problems follow the following steps
  1. Develop the CFG and Cyclomatic complexity
  2. Develop the basis path test set
  3. Mentally convert the multiple decision statements into a multiple condition single decision statement
  4. Solve for MCDC
  5. Test cases are
     a. Basis path set
     b. MCDC solutions (map these into the BP set)
     c. Untested BVs, Extreme range values, and missing MCDC tests

# A Typical Homework Problem (cont.)

```
x=false
…
if (a)
  x=true;
else
  if (b)
    x=true;
```

converts to → x = a || b;

```
x=false
…
if (a)
 if (b)
   x = true
```

converts to → x = a && b;

# A Typical Homework Problem (cont.)

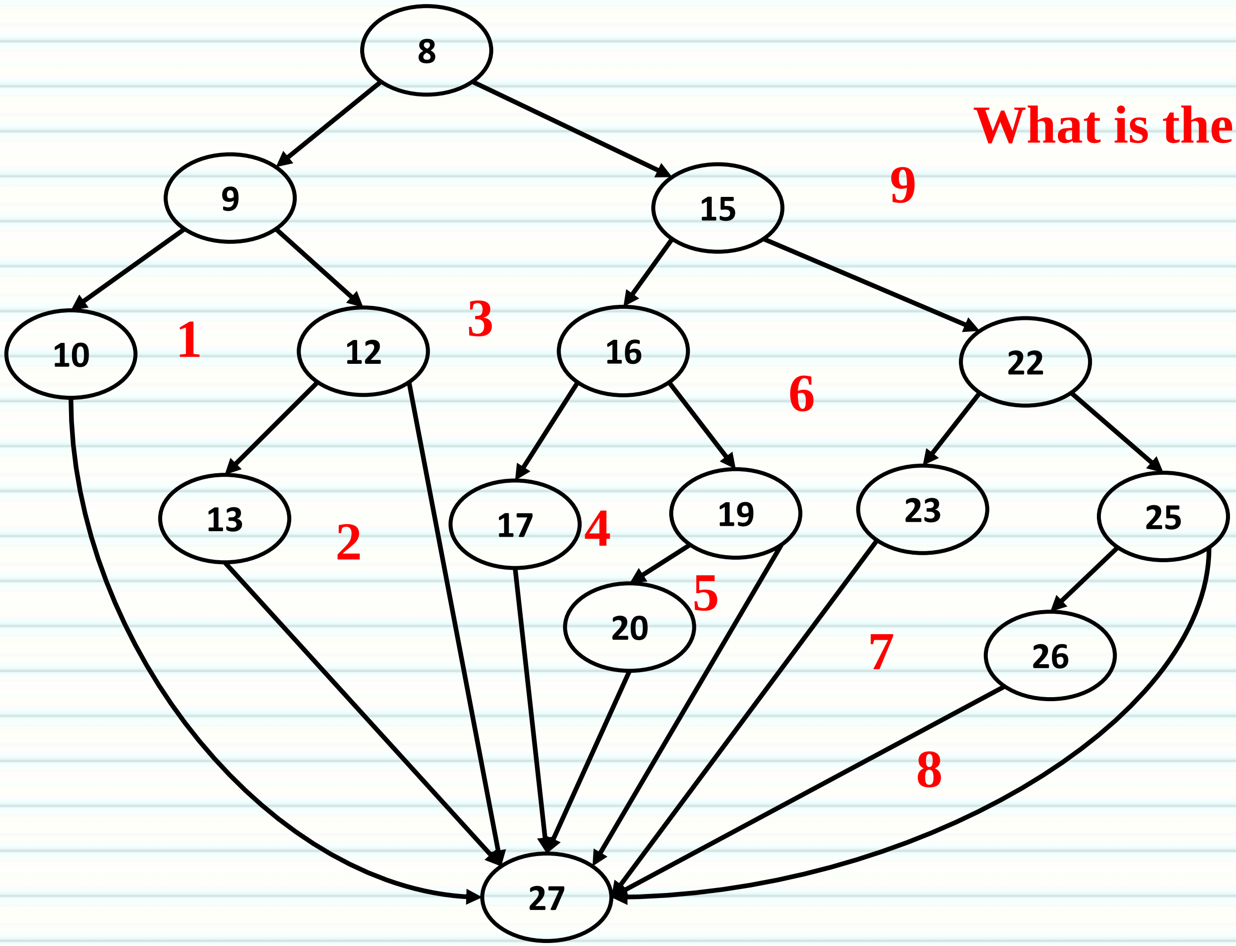
```
public boolean checkOut (double cart, int creditRating, statusClass.Status status) {
        boolean approved=false;

        if (status==statusClass.Status.gold) {
            if (cart < 3_500.00)
                approved = true;
            else
                if (creditRating > 650)
                    approved = true; }
        else {
            if (status==statusClass.Status.silver) {
                if (cart < 2_500.00)
                    approved = true;
                else
                    if (creditRating > 750)
                        approved = true; }
            else {
                if (cart < 1_500.00)
                    approved = true;
                else
                    if (creditRating > 800)
                        approved = true; }}
        return approved; }
```

1. Draw the reduced CFG (DD-path)
2. Determine the Cyclomatic complexity

Assume

1. creditRating ranges from 0 to 850 both inclusive
2. cart ranges from $0.00 to $20,000.00 both inclusive.

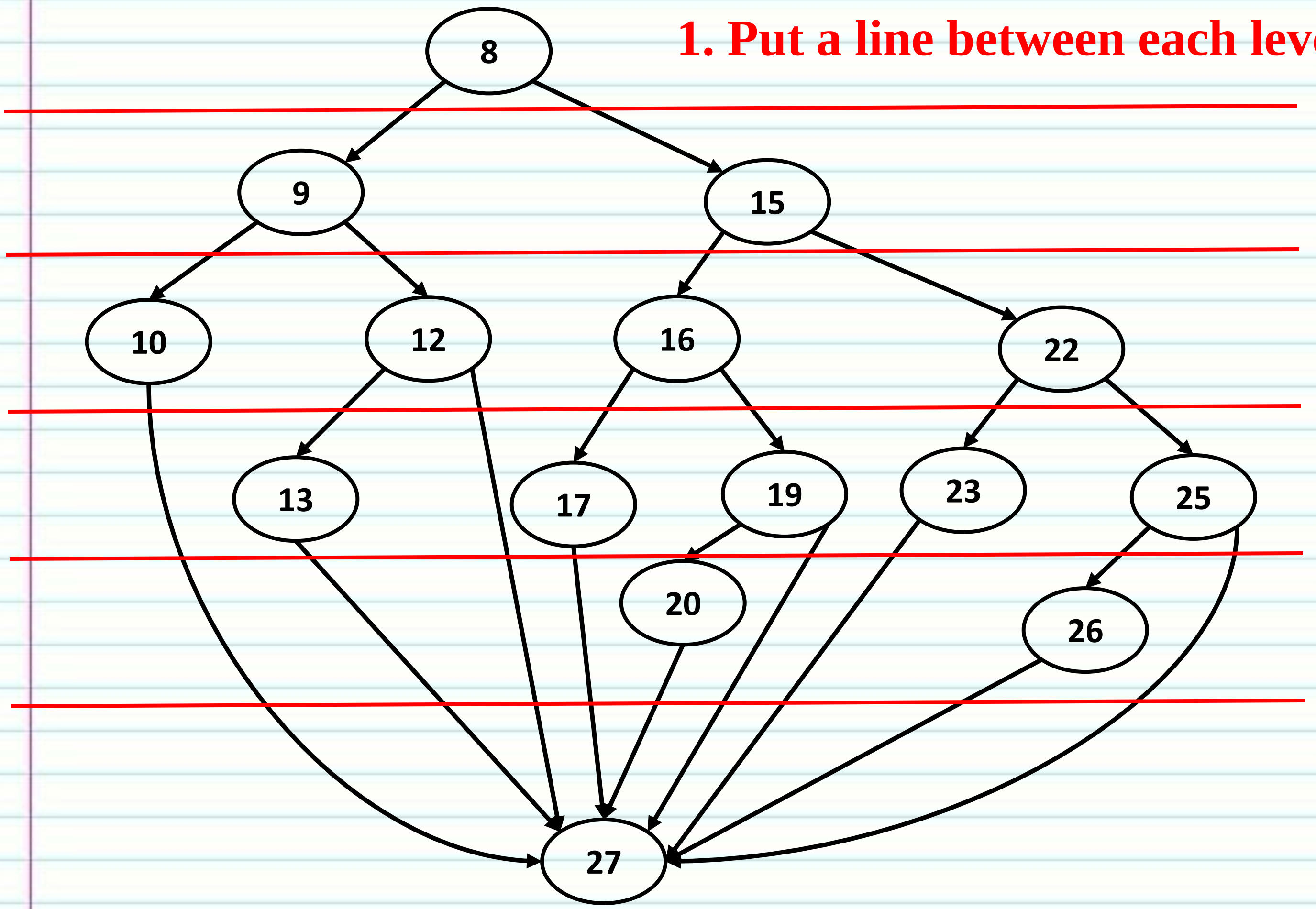
# A Typical Homework Problem (cont.)




What is the correct basis path?

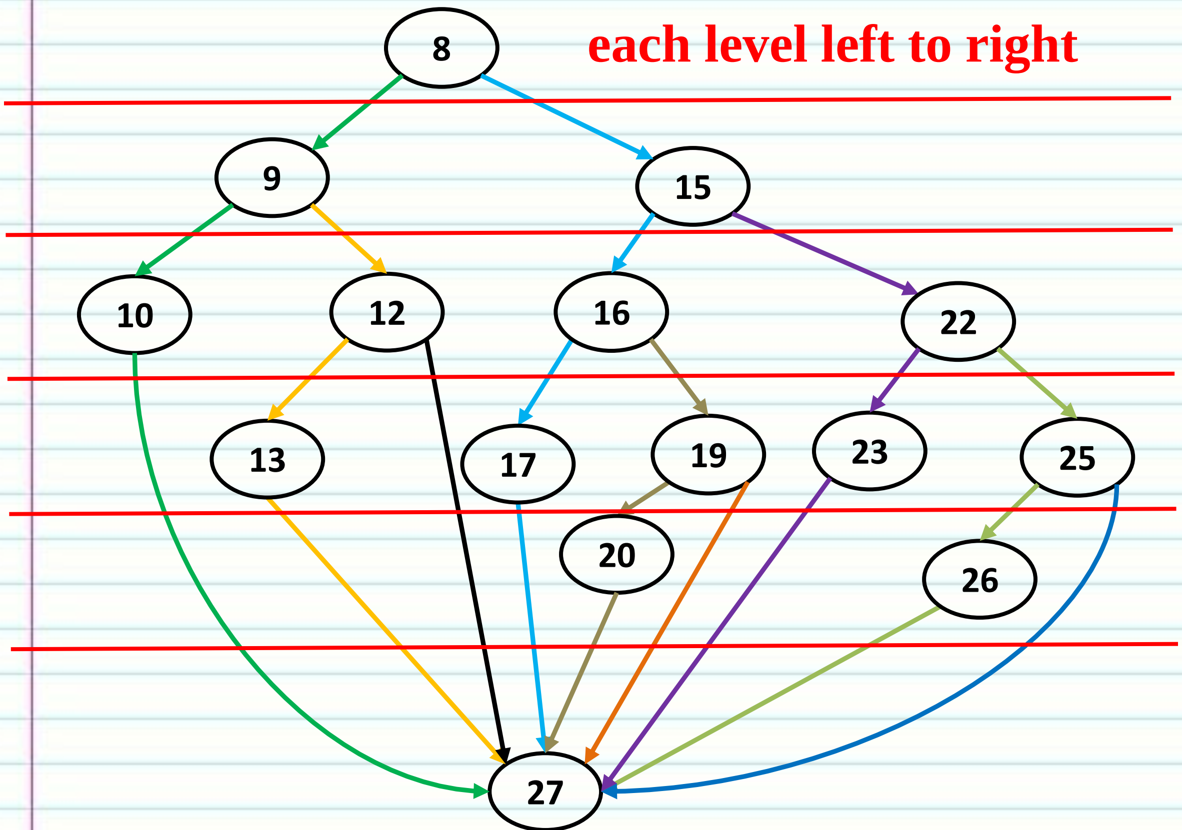
# A Typical Homework Problem (cont.)

1. Put a line between each level




2. Work down from the very top level to the bottom
3. Flip each decision on a level from left to right

# A Typical Homework Problem (cont.)




1. 8-9-10-27
2. 8-15-16-17-27
3. 8-9-12-13-27
4. 8-15-22-23-27
5. 8-9-12-27
6. 8-15-16-19-20-27
7. 8-15-22-25-26-27
8. 8-15-16-19-27
9. 8-15-22-25-27

Now develop the test cases, check all BVs, etc

# A Typical Homework Problem (cont.)

- Initial Test Case Table (from Basis Path only)

| Test Case Number | Inputs | | | Exp Out | |
|---|---|---|---|---|---|
| | cart | creditRating | status | return | Basis Path |
| 1 | $3,499.99 | 650 | gold | TRUE | 8-9-10-27 |
| 2 | $2,499.99 | 750 | silver | TRUE | 8-15-16-17-27 |
| 3 | $3,500.00 | 651 | gold | TRUE | 8-9-12-13-27 |
| 4 | $1,499.99 | 800 | regular | TRUE | 8-15-22-23-27 |
| 5 | $3,500.00 | 650 | gold | FALSE | 8-9-12-27 |
| 6 | $2,500.00 | 751 | silver | TRUE | 8-15-16-19-20-27 |
| 7 | $1,500.00 | 801 | regular | TRUE | 8-15-22-25-26-27 |
| 8 | $2,500.00 | 750 | silver | FALSE | 8-15-16-19-27 |
| 9 | $1,500.00 | 800 | regular | FALSE | 8-15-22-25-27 |

- Now we must add in
  1. MCDC test cases
  2. Missing BVs and Extreme range values

# A Typical Homework Problem

```
public boolean checkOut (double cart, int creditRating, statusClass.Status status) {
    boolean approved=false;

    if (status==statusClass.Status.gold) {
        if (cart < 3_500.00)
            approved = true;
        else
            if (creditRating > 650)
                approved = true; }
    else {
        if (status==statusClass.Status.silver) {
            if (cart < 2_500.00)
                approved = true;
            else
                if (creditRating > 750)
                    approved = true; }
        else {
            if (cart < 1_500.00)
                approved = true;
            else
                if (creditRating > 800)
                    approved = true; }}
    return approved; }
```

**approved = (cart < 3_500.0 || creditRating > 650)**

**Use (MCDC FT, FF, TF)**

**3_500.00, 651 (test case 3)**
**3_500.00, 650 (test case 5)**
**3_499.99, 650 (test case 1)**

**approved = (cart < 2_500.0 || creditRating > 750)**

**2_500.00, 751 (test case 6)**
**2_500.00, 750 (test case 8)**
**2_499.99, 750 (test case 2)**

**approved = (cart < 1_500.0 || creditRating > 800)**

**1_500.00, 801 (test case 7)**
**1_500.00, 800 (test case 9)**
**1_499.99, 800 (test case 4)**

**All of our MCDC test cases are already captured in the BP set - we don’t need to add any**

# A Typical Homework Problem (cont.)

- No extra MCDC tests need to be added
- No extra BVs need to be added
- Complete Test Case Table

| Test Case | Inputs | | | Exp Out | | |
|---|---|---|---|---|---|---|
| Number | cart | creditRating | status | return | Basis Path | MCDC test cases |
| 1 | $3,499.99 | 650 | gold | TRUE | 8-9-10-27 | stmts 9-13 TF |
| 2 | $2,499.99 | 750 | silver | TRUE | 8-15-16-17-27 | stmts 16-20 TF |
| 3 | $3,500.00 | 651 | gold | TRUE | 8-9-12-13-27 | stmts 9-13 FT |
| 4 | $1,499.99 | 800 | regular | TRUE | 8-15-22-23-27 | stmts 22-26 TF |
| 5 | $3,500.00 | 650 | gold | FALSE | 8-9-12-27 | stmts 9-13 FF |
| 6 | $2,500.00 | 751 | silver | TRUE | 8-15-16-19-20-27 | stmts 16-20 FT |
| 7 | $1,500.00 | 801 | regular | TRUE | 8-15-22-25-26-27 | stmts 22-26 FT |
| 8 | $2,500.00 | 750 | silver | FALSE | 8-15-16-19-27 | stmts 16-20 FF |
| 9 | $1,500.00 | 800 | regular | FALSE | 8-15-22-25-27 | stmts 22-26 FF |
| 10 | $20,000.00 | 650 | gold | FALSE | Extreme range cart | |
| 11 | $0.00 | 650 | gold | TRUE | Extreme range cart | |
| 12 | $3,500.00 | 0 | gold | FALSE | Extreme range cR | |
| 13 | $3,500.00 | 850 | gold | TRUE | Extreme range cR | |

- value is a don’t care - we are just testing the extreme ranges

Code coverage achieved is: full boundary coverage, full statement and decision coverage, and extreme range coverage.

# Another Homework Problem

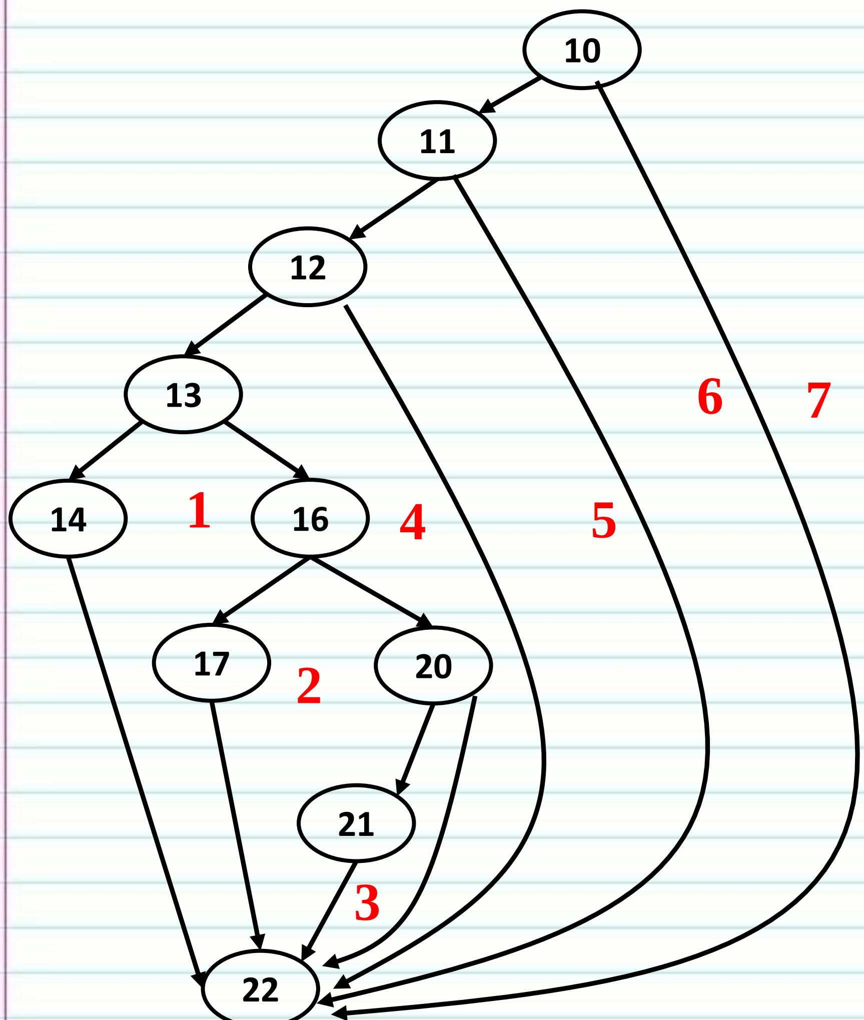
```
public void carCollAlarms (boolean selfDrive, double speed, double distance) {
  carCollAlert=carCollWarn=carCollCaut=emerBrake=false;
   if (selfDrive)
       if (speed > 50.0)
           if (distance<=150.0)
               if (distance>100.0)
                   carCollCaut=true;
               else
                   if (distance>50.0)
                       carCollWarn=true;
                   else {
                       carCollAlert=true;
                       if (distance<=25.0)
                           emerBrake=true;}}
```

1. **Draw the reduced CFG (DD-path)**
2. **Determine the Cyclomatic complexity**

**Assume**

1. **speed ranges from 0.0 to 200.0 both inclusive**
2. **distance ranges from 0.0 to 1,000.0 both inclusive**

# Another Homework Problem (cont.)




| Distance (feet) | | | | | |
|---|---|---|---|---|---|
| Speed (miles/hour) | Rule 1 | Rule 2 | Rule 3 | Rule 4 | Rule 5 |
| **CONDITIONS** | | | | | |
| selfDrive=TRUE | Y | Y | Y | Y | All others |
| speed > 50.0 | Y | Y | Y | Y | |
| 100.1 <= distance <= 150.0 | Y | | | | |
| 50.1 <= distance <= 100.0 | | Y | | | |
| 25.1 <= distance <= 50.0 | | | Y | | |
| 0 <= distance <= 25.0 | | | | Y | |
| **ACTIONS** | | | | | |
| carCollCaut | **TRUE** | FALSE | FALSE | FALSE | FALSE |
| carCollWarn | FALSE | **TRUE** | FALSE | FALSE | FALSE |
| carCollAlert | FALSE | FALSE | **TRUE** | **TRUE** | FALSE |
| emerBrake | FALSE | FALSE | FALSE | **TRUE** | FALSE |

**In the decision table we have to break up each distinct region of distance into a separate ECP**

# Another Homework Problem (cont.)

- Initial test case table (from the BP set)

| Test Case Number | Inputs | | | Exp Out | | | | Basis Path tested |
|---|---|---|---|---|---|---|---|---|
| | selfDrive | speed | distance | carCollCaut | carCollWarn | carCollAlert | emerBrake | |
| 1 | TRUE | 50.1 | 100.1 | **TRUE** | FALSE | FALSE | FALSE | 10-11-12-13-14-22 |
| 2 | FALSE | 50.1 | 100.1 | FALSE | FALSE | FALSE | FALSE | 10-22 |
| 3 | TRUE | 50.0 | 100.1 | FALSE | FALSE | FALSE | FALSE | 10-11-22 |
| 4 | TRUE | 50.1 | 150.1 | FALSE | FALSE | FALSE | FALSE | 10-11-12-22 |
| 5 | TRUE | 50.1 | 50.1 | FALSE | **TRUE** | FALSE | FALSE | 10-11-12-13-16-17-22 |
| 6 | TRUE | 50.1 | 25.0 | FALSE | FALSE | **TRUE** | **TRUE** | 10-11-12-13-16-20-21-22 |
| 7 | TRUE | 50.1 | 25.1 | FALSE | FALSE | **TRUE** | FALSE | 10-11-12-13-16-20-22 |

# Another Homework Problem (cont.)

```
8  public void carCollAlarms (boolean selfDrive, double speed, double distance) {
9    carCollAlert=carCollWarn=carCollCaut=emerBrake=false;
10     if (selfDrive)
11        if (speed > 50.0)
12           if (distance<=150.0)
13              if (distance>100.0)
14                 carCollCaut=true;
15              else
16                 if (distance>50.0)
17                    carCollWarn=true;
18                 else {
19                    carCollAlert=true;
20                    if (distance<=25.0)
21                       emerBrake=true;}}
```

**If we take statements 10-14 together as a logical expression *abcd* we use the MCDC test cases TTTT, FTTT, TFTT, TTFT,TTTF**

| | selfD | speed | distance | TC# |
|---|---|---|---|---|
| TTTT | true | 50.1 | 100.1 | 1 |
| FTTT | false | 50.1 | 100.1 | 2 |
| TFTT | true | 50.0 | 100.1 | 3 |
| TTFT | true | 50.1 | 150.1 | 4 |
| TTTF | true | 50.1 | 100.0 | 8 |

**ECP/BV analysis of statement 14 - speed MUST be 50.1 mph**

| distance (ft.) | | | |
|---|---|---|---|
| | 0.0 100.0 | 100.1 150.0 | 150.1 200.0 |

**We see that with speed = 50.1 mph we need to test 4 BVs - we need to add an additional test case (TC 9) to the MCDC set for the untested BV.**

**When we have a multiple condition with variable a like a < x && a > y one of the TT BVs will be missing**

# Another Homework Problem (cont.)

```
8  public void carCollAlarms (boolean selfDrive, double speed, double distance) {
9    carCollAlert=carCollWarn=carCollCaut=emerBrake=false;
10    if (selfDrive)
11       if (speed > 50.0)
12          if (distance<=150.0)
13             if (distance>100.0)
14                carCollCaut=true;
15             else
16                if (distance>50.0)
17                   carCollWarn=true;
18                else {
19                   carCollAlert=true;
20                   if (distance<=25.0)
21                      emerBrake=true;}}
```

**ECP/BV analysis of statement 17 - speed MUST be 50.1 mph**

| distance (ft.) | | | |
|---|---|---|---|
| | 0.0   50.0 | 50.1   100.0 | 100.1 200.0 |

**We see that with speed = 50.1 mph we need to test 4 BVs - we need to add an additional test case (TC 10) to test the untested BV of distance=50.0 when speed=50.1**

# Another Homework Problem (cont.)

- Final test case table - we have to add three BVs to the Basis Path set (test cases 8, 9 , and 10)

| Test Case Number | Inputs | | | Exp Out | | | | Basis Path tested |
|---|---|---|---|---|---|---|---|---|
| | selfDrive | speed | distance | carCollCaut | carCollWarn | carCollAlert | emerBrake | |
| 1 | TRUE | 50.1 | 100.1 | **TRUE** | FALSE | FALSE | FALSE | 10-11-12-13-14-22 |
| 2 | FALSE | 50.1 | 100.1 | FALSE | FALSE | FALSE | FALSE | 10-22 |
| 3 | TRUE | 50.0 | 100.1 | FALSE | FALSE | FALSE | FALSE | 10-11-22 |
| 4 | TRUE | 50.1 | 150.1 | FALSE | FALSE | FALSE | FALSE | 10-11-12-22 |
| 5 | TRUE | 50.1 | 50.1 | FALSE | **TRUE** | FALSE | FALSE | 10-11-12-13-16-17-22 |
| 6 | TRUE | 50.1 | 25.0 | FALSE | FALSE | **TRUE** | **TRUE** | 10-11-12-13-16-20-21-22 |
| 7 | TRUE | 50.1 | 25.1 | FALSE | FALSE | **TRUE** | FALSE | 10-11-12-13-16-20-22 |
| 8 | TRUE | 50.1 | 100.0 | FALSE | **TRUE** | FALSE | FALSE | TTTF from slide 61 |
| 9 | TRUE | 50.1 | 150.0 | **TRUE** | FALSE | FALSE | FALSE | Missing MCDC slide 61 |
| 10 | TRUE | 50.1 | 50.0 | FALSE | FALSE | **TRUE** | FALSE | Missing MCDC slide 62 |
| 11 | TRUE | 0.0 | 150.0 | FALSE | FALSE | FALSE | FALSE | extreme range speed |
| 12 | TRUE | 200.0 | 150.0 | **TRUE** | FALSE | FALSE | FALSE | extreme range speed |
| 13 | TRUE | 50.1 | 1,000.0 | FALSE | FALSE | FALSE | FALSE | extreme range distance |
| 14 | TRUE | 50.1 | 0.0 | FALSE | FALSE | **TRUE** | **TRUE** | extreme range distance |

- value is a don’t care - we are just testing the extreme ranges

# Mathematical Tests

- Mathematical tests also require some insight and rules, much like && and || operators are sensitive to F and T respectively

- Sums are sensitive to zero
  - a = b + c,
  - where either b or c =0 is not a good test condition, as it does not strongly demonstrate the use of b, c (many values are initialized to 0)
  - For example, a = b+c when c=0 could not distinguish a=b from a=b+c
  - so it is best to use non-zero and best to use unique values to more strongly assert the correct operands
- Multiplies are sensitive to 0 or 1
  - anything * 0 = 0, so it doesn't demonstrate that the correct operand was used  c=a*b, b=0 doesn't show a was actually used
  - the identity function does not as strongly demonstrate both operands were correctly used either –     c=a*b vs. c=a
  - so it is best to avoid using 0,1 in multiplications and best to use unique values to more strongly assert the correct operands

(c) JRCS 2016

# Mathematical Exercise

```
private static int getNumberofThingAwaMichees(double a, double b, double c)  {

        double d;
        int numroots;
        d=b*b - 4*a*c;
        if (d>0)
                numroots=2;
        else {
                if (d==0)
                        numroots=1;
                else
                        numroots=0;
        }
        return numroots;
}
```

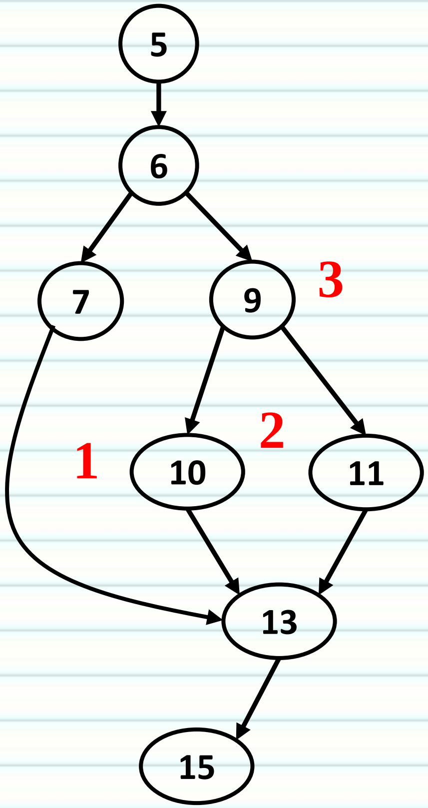
**hint: use a=18.0, c=2.0**

**Student exercise**

1. **Describe what the code seems to be doing**
2. **Draw the reduced CFG (DD-path)**
3. **Determine the Cyclomatic complexity**
4. **Develop the basis paths using the Cyclomatic complexity**
5. **Develop the test cases using required input values to achieve boundary value coverage (0.01)**
6. **Determine coverage achieved**
7. **Do the test cases and outputs refute or support the code functional description?**

(c) JRCS 2016

# Mathematical Exercise - answers




1. The software attempts to count the number of roots for a quadratic equation given a,b,c
2. The reduced CFG is shown to the left with the corresponding line numbers shown in each node.
3. The cyclomatic complexity is counted to the left in red – it is 3 – requiring 3 tests cases for basis path coverage.

| Test | Inputs | | | Output | |
|---|---|---|---|---|---|
| Case | a | b | c | Return Val. | Path coverage |
| 1 | 18.00 | 12.01 | 2.00 | 2 | 5,6,7,13,15 |
| 2 | 18.00 | 12.00 | 2.00 | 1 | 5,6,9,10,13,15 |
| 3 | 18.00 | 11.99 | 2.00 | 0 | 5,6,9,11,13,15 |

Notice that a, b, c were picked to be unique

Note also the 0.01 threshold

6. Full statement coverage is achieved with the basis paths selected. WHY?
7. The test cases support the functional description.

I had to develop a model to identify unique values

(c) JRCS 2016