Testing Techniques

Classical Test Design Techniques

Jan Tretmans

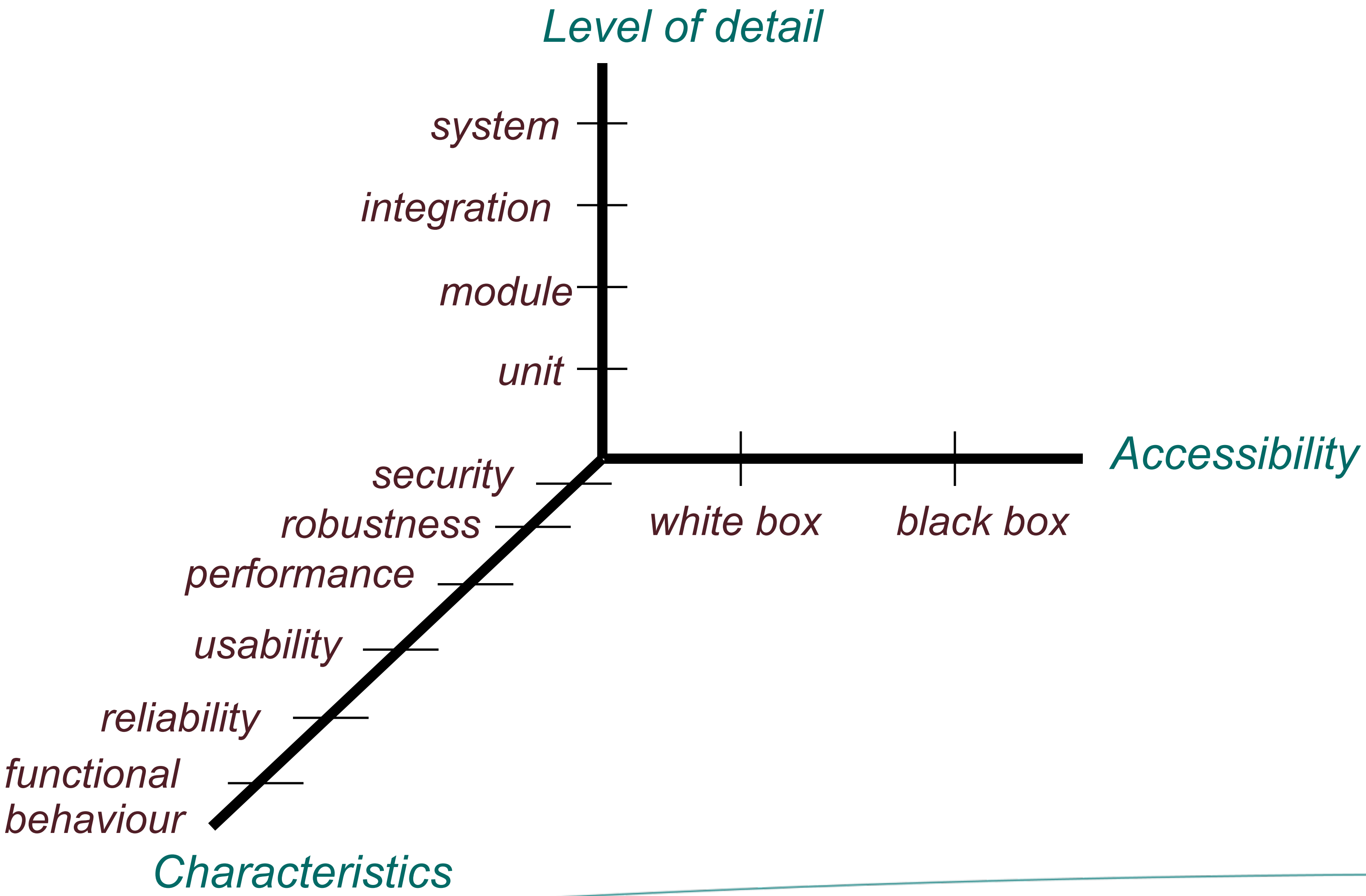
TNO – ESI

Radboud University Nijmegen

Some ‘Classical’ Test Design Techniques

- Black-box testing (*functional testing*)
  - Equivalence partitioning
  - Boundary value analysis
  - Error guessing
- White-box testing (*structural testing*)
  - Statement coverage
  - Decision / branch coverage
- Black-box and white-box test case design in combination
- Basics : heuristics and experience

Types of Testing



Development of Test Cases

Complete testing is in general impossible

$\Downarrow$

Testing cannot guarantee the absence of faults

$\Downarrow$

How to select subset of test cases from all possible test cases with a high chance of detecting most faults ?

$\Downarrow$

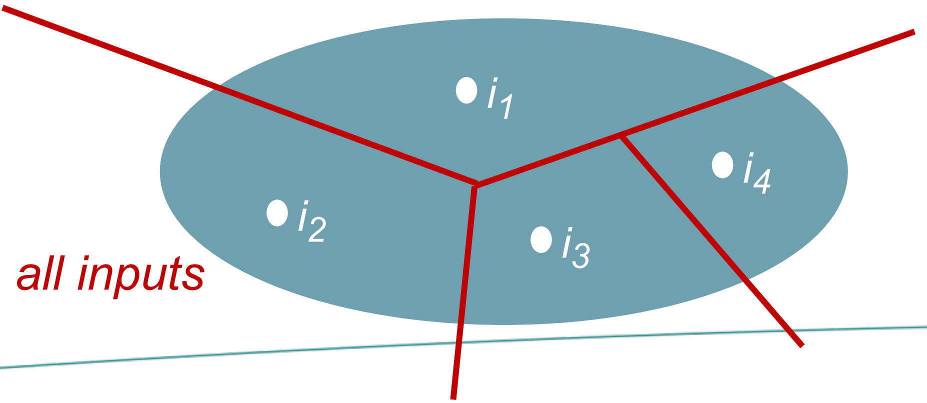
Test Case Design Strategies

Because : *if we have good test suite then we can have more confidence in the product that passes that test suite*

Black-Box Testing : Equivalence Partitioning (EP)

Divide all possible inputs into classes (partitions) such that :

- There is a finite number of input equivalence classes
- You may reasonably assume that
  - the program behaves analogously for inputs in the same class
  - one test with a representative value from a class is sufficient
  - if the representative detects a fault
    then other class members would detect the same fault



Black-Box Testing : Equivalence Partitioning

Strategy :

- Identify input equivalence classes
  - Based on conditions on inputs/outputs in specification/description
  - Both valid and invalid input equivalence classes
  - Based on heuristics and experience :
    - “input x in [1..10]” $\rightarrow$ classes : $x < 1$, $1 \leq x \leq 10$, $x > 10$
    - “enumeration A, B, C" $\rightarrow$ classes : A, B, C, not{A,B,C,}
    - "input integer n" $\rightarrow$ classes : n not an integer, n<min, min≤n<0, 0≤n≤max, n>max
    - ……
- Define one/couple of test cases for each class
  - Test cases that cover valid classes
  - Test cases that cover at most one invalid class

Example : Equivalence Partitioning

Test a function for calculation of the absolute value of an integer x

- Equivalence classes :

Condition	*Valid eq. classes*	*Invalid eq. Classes*
nr of inputs	*1* (1)	*0* (2), *> 1* (3)
input type	*integer* (4)	*non-integer* (5)
particular abs	*< 0* (6), *>= 0* (7)	

a type system can prevent these values

- Test cases :

x = -10 (1,4,6)	*x = -* (2)	*x = 10 20* (3)
x = 100 (1,4,7)	*x = "XYZ"* (5)	

Triangle Program [Myers]

"A program reads three integer values. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral."

Write a set of test cases to test this program.

A Self-Assessment Test [Myers]

Test cases for:

1. valid scalene triangle ?
2. valid equilateral triangle ?
3. valid isosceles triangle ?
4. 3 permutations of previous ?
5. side = 0 ?
6. negative side ?
7. one side is sum of others ?
8. 3 permutations of previous ?
9. one side larger than sum of others ?
10. 3 permutations of previous ?
11. all sides = 0 ?
12. non-integer input ?
13. wrong number of values ?
14. for each test case: is expected output specified ?
15. check behaviour after output was produced ?

Triangle Program [Myers]

Test cases for:

Valid cases:

1. valid scalene triangle ?
2. valid equilateral triangle ?
3. valid isosceles triangle ?

Invalid cases:

4. negative side ?
5. one side larger than sum of others ?
6. non-integer input ?
7. wrong number of values ?

Example : Equivalence Partitioning

- Test a program that computes the sum of the first *N* integers as long as this sum is less than *maxint.* Otherwise an *error* should be reported. If *N* is negative, then it takes the absolute value *N*.

- *Formally*:

  Given integer inputs *N* and *maxint* compute *result* :

$$result = \sum_{K=0}^{|N|} k \quad \text{if this} <= maxint, \quad error \text{ otherwise}$$

Example : Equivalence Partitioning

- Equivalence classes :

Condition	*Valid eq. classes*	*Invalid eq. classes*
nr of inputs	2	< 2, > 2
type of input	int int	int no-int, no-int int
abs(*N*)	$N < 0$, $N \geq 0$	
maxint	$\sum k \leq maxint$ $\sum k > maxint$	

- Test Cases :

	maxint	*N*	*result*
Valid	100	10	55
	100	-10	55
	10	10	error
Invalid	10	-	error
	10 20	30	error
	"XYZ"	10	error
	100	9.1E4	error

Black-Box Testing : Boundary Value Analysis (BVA)

Based on experience / heuristics :

- Testing boundary conditions of eq. classes is more effective i.e. values directly on, above, and beneath edges of classes
- Choose input boundary values as tests in input classes instead of, or additional to arbitrary values
- Choose also inputs that invoke output boundary values (values on the boundary of output classes)
- Example strategy as extension of equivalence partitioning:
  - choose one (n) arbitrary value(s) in each eq. class
  - choose values exactly on lower and upper boundaries of eq. class
  - choose values immediately below and above each boundary (if applicable)

Example : Boundary Value Analysis

Test a function for calculation of the absolute value of an integer

- Valid equivalence classes :

Condition	*Valid eq. classes*	*Invalid eq. Classes*
particular abs	< 0, >= 0	

- Test cases :
  - class x < 0, arbitrary value: x = -10
  - class x >= 0, arbitrary value x = 100
  - classes x < 0, x >= 0, on boundary : x = 0
  - classes x < 0, x >= 0, below and above: x = -1, x = 1

A Self-Assessment Test [Myers]

Test cases for:

1. valid scalene triangle ?
2. valid equilateral triangle ?
3. valid isosceles triangle ?
4. 3 permutations of previous ?
5. side = 0 ?
6. negative side ?
7. one side is sum of others ?
8. 3 permutations of previous ?
9. one side larger than sum of others ?
10. 3 permutations of previous ?
11. all sides = 0 ?
12. non-integer input ?
13. wrong number of values ?
14. for each test case: is expected output specified ?
15. check behaviour after output was produced ?

Example : Boundary Value Analysis

- Given inputs *maxint* and *N* compute *result* :

$$result = \sum_{K=0}^{|N|} k \quad \text{if this} <= maxint, \quad error \text{ otherwise}$$

- Valid equivalence classes :

condition	*valid eq. classes* .
abs(*N*)	$N < 0, \quad N \geq 0$
maxint	$\sum k \leq maxint, \quad \sum k > maxint$

- Can be extended with *maxint*<0, *maxint* >= 0, max integer, ……

Example : Boundary Value Analysis

- Valid equivalence classes :

condition	*valid eq. classes*
abs(N)	$N < 0$, $N \geq 0$
maxint	$\sum k \leq maxint$, $\sum k > maxint$

- Test Cases :

maxint	*N*	*result*
55	10	55
54	10	error
56	10	55
0	0	0

maxint	*N*	*result*
100	0	0
100	-1	1
100	1	1
…	…	…

- How to combine the boundary conditions of different inputs ?
  Take all possible boundary combinations ? This may blow up ……

Black-Box Testing : Error Guessing

now also known as
exploratory testing

- Just ‘guess’ where the errors are ……
- Intuition and experience of tester
- Ad hoc, not really a technique
- But can be quite effective
- Strategy:
  - Make a list of possible errors or error-prone situations

    (may be related to boundary conditions)
  - Write test cases based on this list

Black-Box Testing : Error Guessing

- More sophisticated ‘error guessing’ : Risk Analysis
- Product risk analysis
  - functional : critical functionality or use of the product (e.g. safety)
  - structural : critical parts of the code (high risk code sections)
    - parts with unclear specifications, ...
    - complex algorithms or complex code

      measure code complexity - tools available (McGabe, Logiscope,…)
- Process risk analysis
  - which phases of development were critical

    e.g., requirements capturing, unexperienced development teams, ...
- High-risk code will be more thoroughly tested, or rewritten

Black-Box Testing : Which One ?

- Black-box testing techniques :
  - Equivalence partitioning
  - Boundary value analysis
  - Cause-effect graphing
  - Decision tables
  - State transition testing
  - Error guessing
  - ………
- Which one to use ?
  - None of them are complete
  - All are based on some kind of heuristics
  - They are complementary

Black-Box Testing : Which One ?

- Always use a combination of techniques
  - When a formal specification is available try to use it
  - Identify valid and invalid input equivalence classes
  - Identify output equivalence classes
  - Apply boundary value analysis on valid equivalence classes
  - Guess about possible errors
  - Cause-effect graphing for linking inputs and outputs

White-Box Testing

- Testing based on the (internal) *structure* of the system under test
- For programs: testing based on program code

  hence, programming language dependent
- Extent to which (source) code is executed, i.e. covered
- Different kind of coverage :
  - path coverage
  - statement coverage
  - (multiple-) condition coverage
  - decision / branch coverage
  - . . . . .

White-Box Testing : Path Testing

- Execute every possible path of a program,

  i.e., every possible sequence of statements
- Strongest white-box criterion
- Usually impossible: infinitely many paths (in case of loops)
- So: not a realistic option
- But note: enormous reduction w.r.t. all possible test cases

  (each sequence of statements executed for only one value)

Example Program

- Test a program that computes the sum of the first *N* integers as long as this sum is less than *maxint*. Otherwise an *error* should be reported. If *N* is negative, then it takes the absolute value *N*.
- Formally:

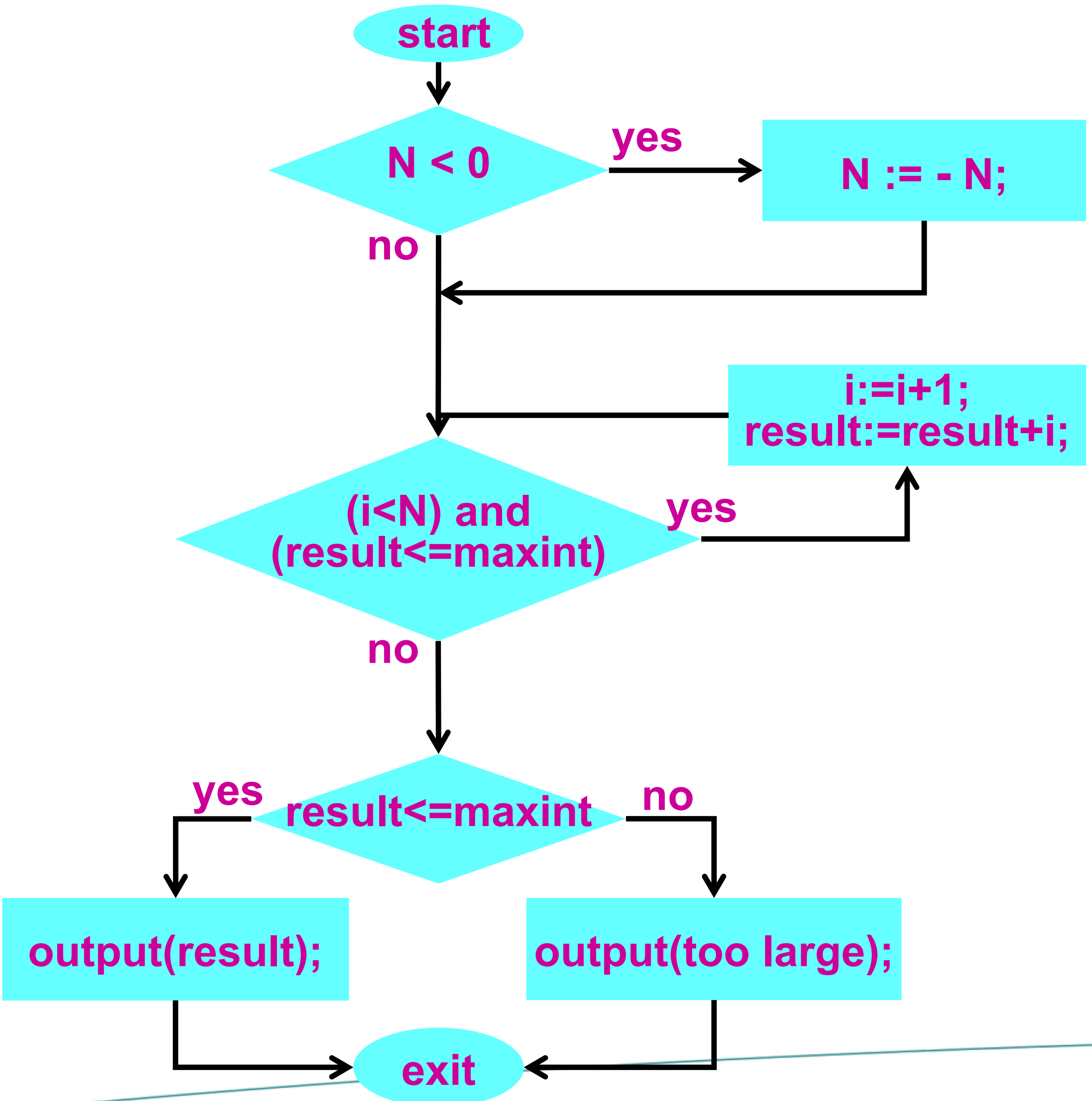
  Given integer inputs *N* and maxint compute *result* :

$$result = \sum_{K=0}^{|N|} k \qquad \text{if this} <= maxint, \quad error \text{ otherwise}$$

Example Program

```
PROGRAM  som ( maxint, N : INT )
        INT    result := 0 ;   i := 0 ;
        IF   N < 0
        THEN   N  :=  - N ;
        WHILE   ( i < N )  AND  ( result <= maxint )
        DO        i  :=  i + 1 ;
                  result  :=  result + i ;
        OD;
        IF   result <= maxint
                  THEN   OUTPUT ( result )
                  ELSE   OUTPUT ( “too large” )
        END.
```

Example : Path Testing



Path:

start

i:=i+1;

result:=result+i;

i:=i+1;

result:=result+i;

….

….

i:=i+1;

result:=result+i;

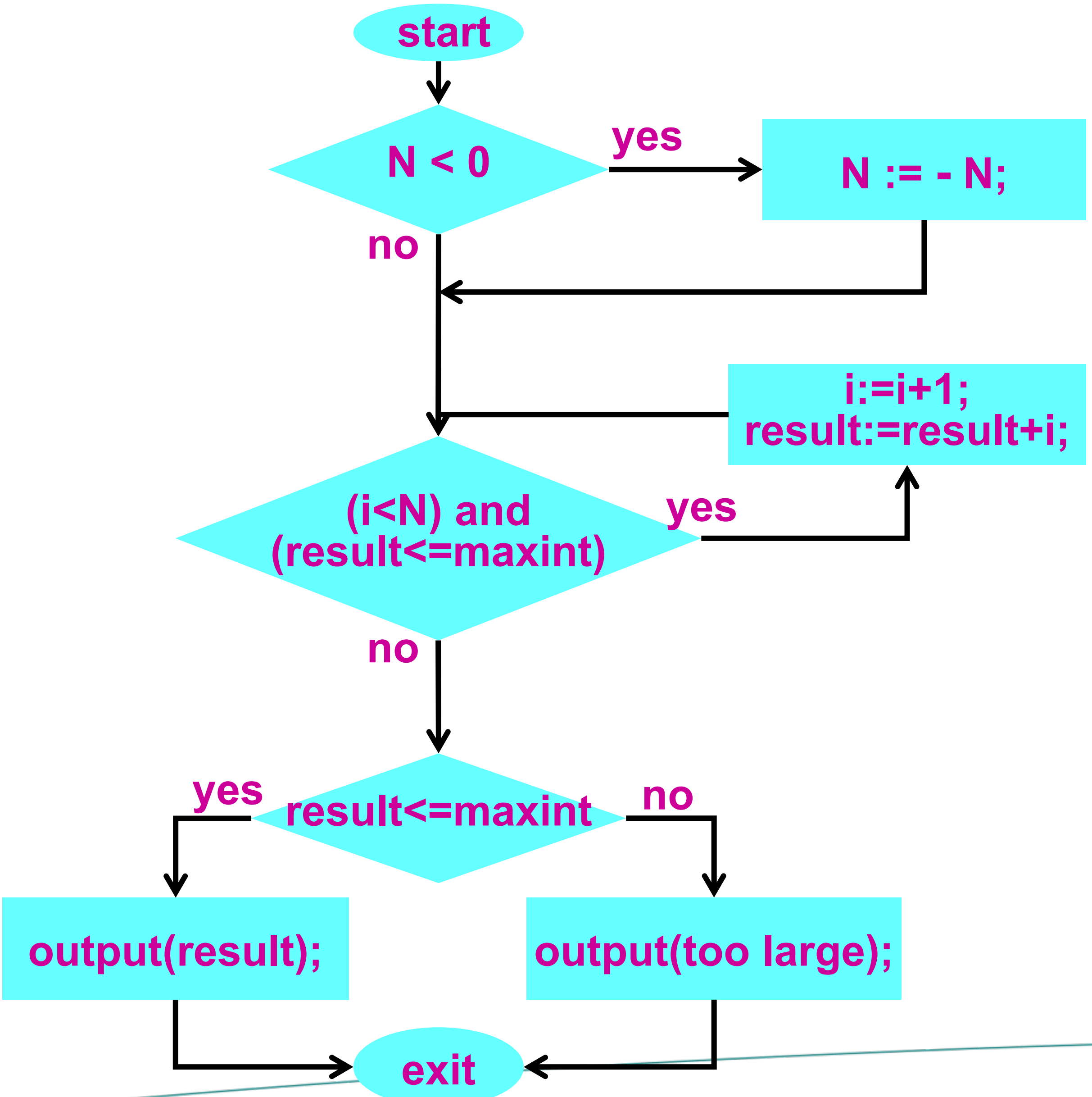
output(result);

exit

White-Box Testing : Statement Coverage

- Execute every statement of a program
- Relatively weak criterion
- Weakest white-box criterion

Example : Statement Coverage



Tests for complete statement coverage:

maxint	*N*
10	- 1
0	- 1

White-Box Testing : Branch Coverage

- Branch coverage == decision coverage
- Execute every branch of a program :

  each possible outcome of each decision occurs at least once

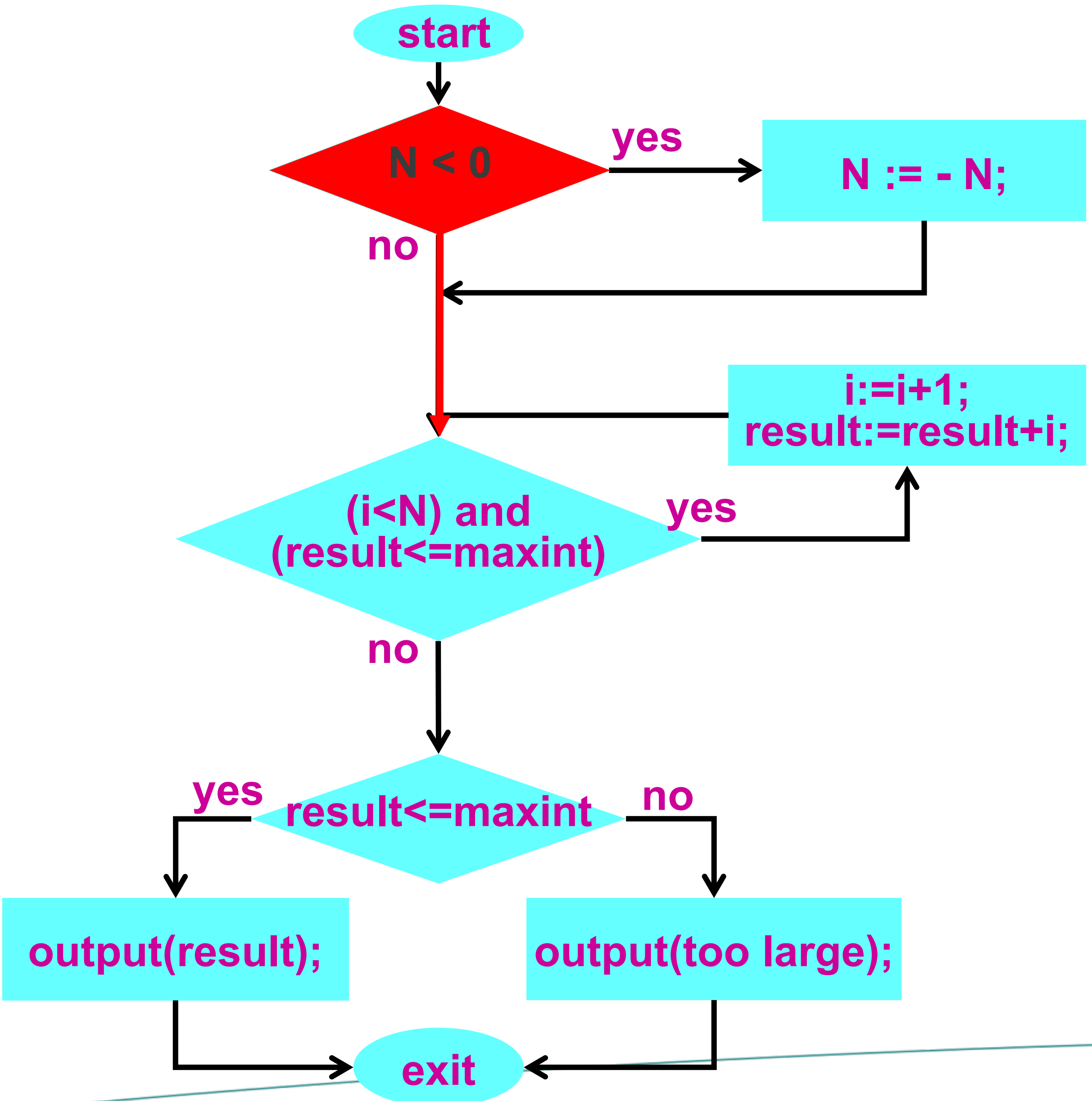
- Example:
  - IF b THEN s1 ELSE s2
  - CASE x OF

    1 : ….

    2 : ….

    3 : ….

Example : Branch Coverage



Tests for complete *statement coverage*:

maxint	*N*
10	**- 1**
0	**- 1**

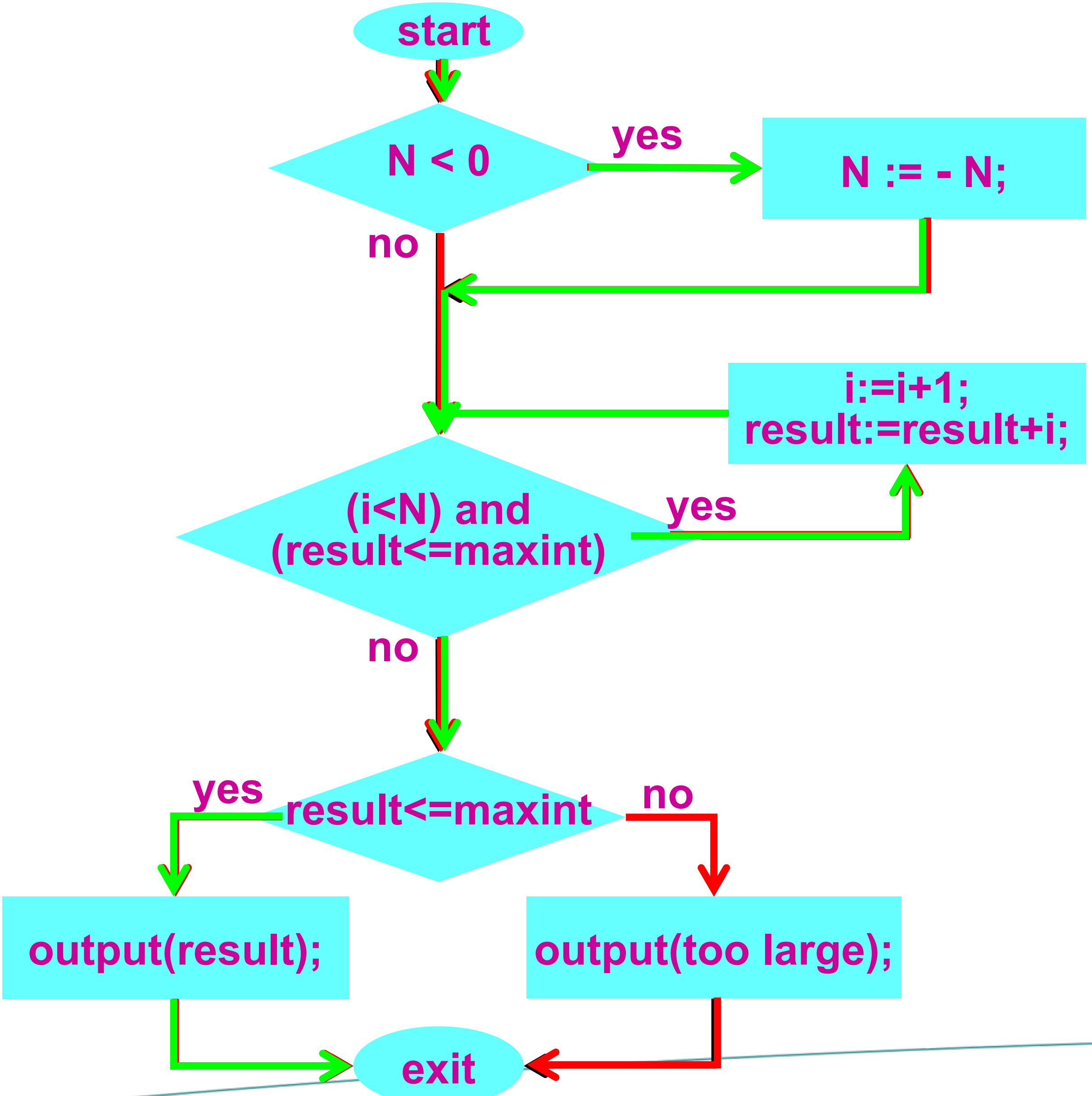
are not sufficient for *branch coverage*;

Take:

maxint	*N*
10	**3**
0	**- 1**

for complete *branch coverage*

Example : Branch Coverage



branch coverage guarantees that each decision outcome is taken at least once, but not all combinations of decisions!

maxint	*N*
-1	**-1**
10	**3**

for *branch coverage*

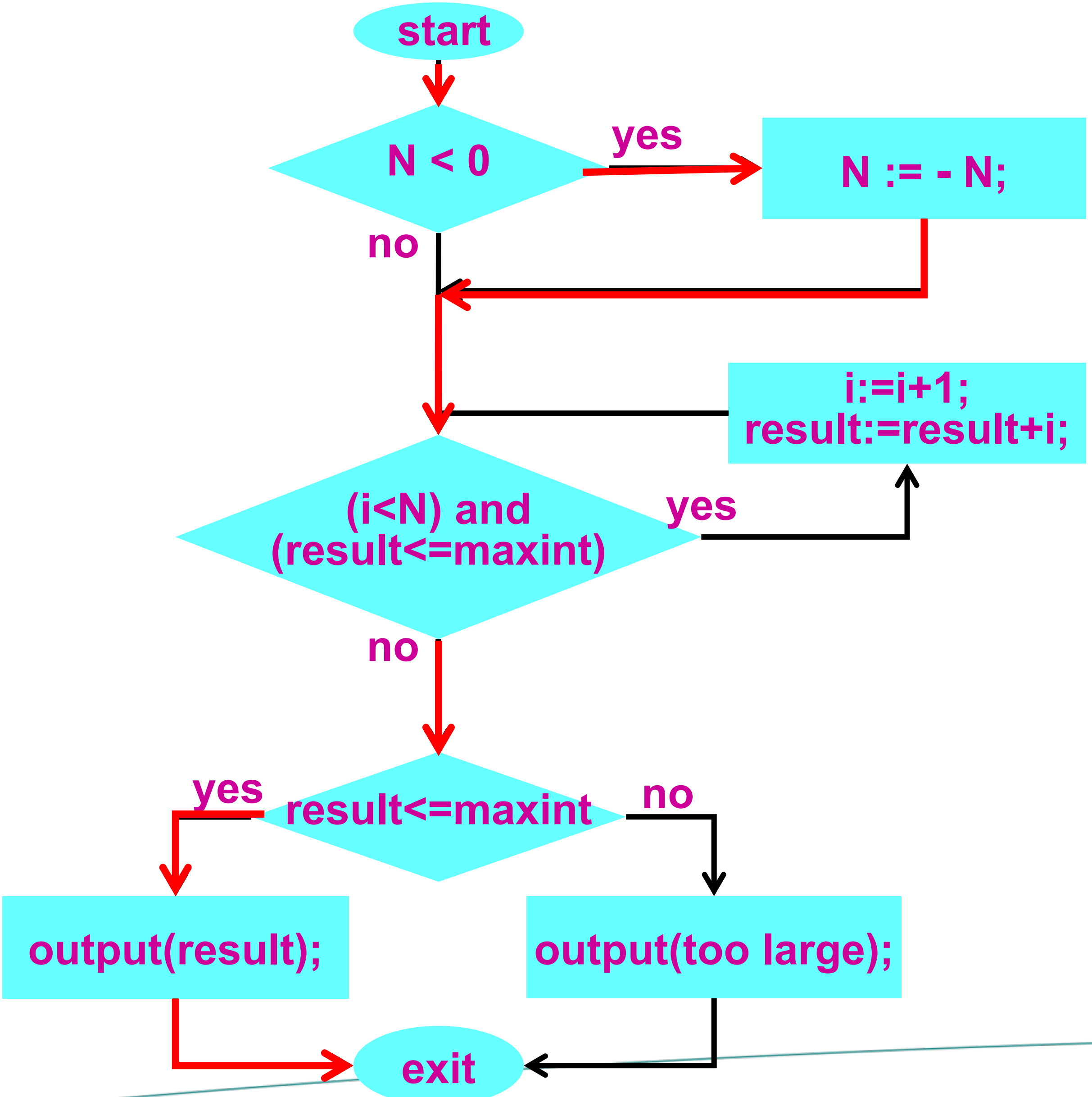
but no **green** path!

Needed :

combination of decisions

10	**- 3**

Example : Statement Coverage



Sometimes there are *infeasible paths* (infeasible combinations of conditions)

White-Box Testing : Condition Coverage

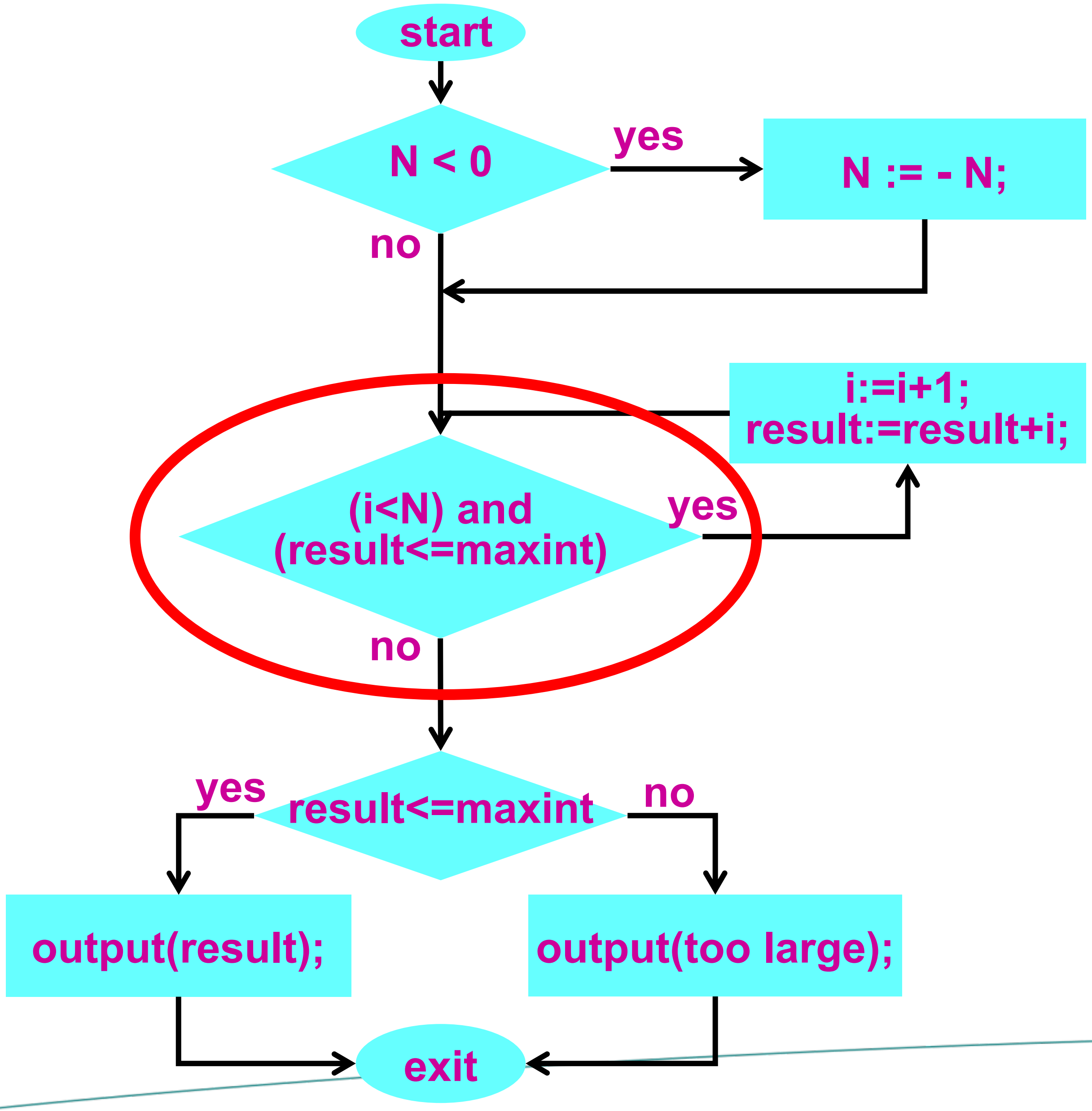
- Design test cases such that each possible outcome of each *condition* in each *decision* occurs at least once

- Example:
  - decision (*i* < *N*) AND (*result* <= *maxint*)

    consists of two conditions : (*i* < *value*) and (*result* <= *maxint*)

    test cases should be designed such that each *condition* gets value *true* and *false* at least once

Example : Statement Coverage



But (*i* = *result* = *0*) :

maxint	*N*	*i*<N	*result*<=*maxint*
-1	1	true	false
1	0	false	true

gives *condition coverage* for all conditions

But it does not preserve *decision coverage*

⇓

always take care that *condition coverage* preserves *decision coverage*

⇓

decision / condition coverage

White-Box testing : Multiple Condition Testing

- Design test cases for each combination of conditions
- Example:
  - (*i* < *N*) (*result* <= *maxint*)

(*i* < *N*)	(*result* <= *maxint*)
false	false
false	true
true	false
true	true

- Implies *decision*-, *condition*-, *decision/condition coverage*
- But : exponential blow-up
- Again : some combinations may be infeasible

White-Box testing : How to Apply ?

- Don’t start with designing white-box test cases !
- Start with black-box test cases

  (equivalence partitioning, boundary value analysis, ...)
- Check white-box coverage

  (statement-, branch-, condition-, ... coverage)
- Use a coverage tool
- Design additional white-box test cases for not covered code

A Coverage Tool

- Many coverage tools : commercial and open source

  tcov gcov Cobertura CodeCover Coverage.py EMMA, Jacoco, Jcov,

  PITest Clover Bullseye Jtest hpc VS Cantata, ...

- Compile your program under test with a special option
- Run a number of test cases
- A listing indicates how often each statement/decision/. . . . . was executed

  and percentage of statements . . . . . executed