EVERYTHING YOU NEED TO KNOW ABOUT CODE COVERAGE IN C++

XAVIER BONAVENTURA – JORGE PINTO SOUSA

using std::cpp 2024

ABOUT US

- Where to find Xavier:
  - @limdor
  - @xbonaventurab
- Working in C++
  - since 2011
  - in safety critical systems since 2018
- Represents BMW at the MISRA C++ working group and at the C++ committee meetings

- Where to find Jorge:
  - jorge.pinto.sousa@proton.me
- Working in C++
  - since 2015
  - in safety critical systems since 2020
- Quality toolchain lead for an autonomous driving project for the past few years

ABOUT US

- **Disclaimer:**
  - We are not compiler engineers
  - We are not coverage experts

  - Jorge, just the (ex) quality manager and Xavier the technical lead of a SW development area in a project,
    - that has to deal with functional safety,
    - has high standard of line and branch coverage.

IS TESTING IMPORTANT?



- Ariane 5 Flight 501 - June 4th 1996
  - Rocket disintegration H_0 +39
  - Due to wrong angle of attach
  - Due to inertial reference system sending wrong data
  - Due to not properly handling an operator error
  - Due to conversion error from double 64 bit to integer 16 bits
  - Due to a too high value
  - Coming from a sensor

- https://www.esa.int/Newsroom/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_report
- https://www.esa.int/esatv/Videos/1996/06/Ariane_501_recordings_telecine_4_3

IS TESTING IMPORTANT?



- Ariane 5 Flight 501 - June 4th 1996
  - Code was taken from Ariane 4
  - The code sending the wrong signal was not needed during flight
  - It was never tested with the Ariane 5 planned trajectory
  - Estimated cost of $370 million

- https://www.esa.int/Newsroom/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_report
- https://www.esa.int/esatv/Videos/1996/06/Ariane_501_recordings_telecine_4_3

ABOUT YOU

- How many of you measure code coverage?

- How many of you have a target of 85% or higher?
  - 95% or higher?
  - 100%?

MOTIVATION

Someone said: “You must achieve 100% code coverage with unit tests”

What does it mean 100% code coverage?

WHAT IS CODE COVERAGE?

Percentage of code executed by test suites

EXAMPLE

```
consteval std::string_view day_of_the_week(int index){
    switch(index){
        case 0:
            return "Monday";
        case 1:
            return "Tuesday";
        case 2:
            return "Wednesday";
        case 3:
            return "Thursday";
        case 4:
            return "Friday";
        case 5:
            return "Saturday";
        case 6:
            return "Sunday";
        default:
            return "Unknown";
    }
}
```

https://godbolt.org/z/qjGPcrf5d

EXAMPLE

```
consteval std::string_view day_of_the_week(int index){
    switch(index){
        case 0:
            return "Monday";
        case 1:
            return "Tuesday";
        //...more code....
        case 6:
            return "Sunday";
        default:
            return "Unknown";
    }
}
```

https://godbolt.org/z/qjGPcrf5d

EXAMPLE

```
void test(){
    expect_eq(day_of_the_week(0), "Monday"sv);
    expect_eq(day_of_the_week(1), "Tuesday"sv);
    expect_eq(day_of_the_week(2), "Wednesday"sv);
    expect_eq(day_of_the_week(3), "Thursday"sv);
    expect_eq(day_of_the_week(4), "Friday"sv);
    expect_eq(day_of_the_week(5), "Saturday"sv);
    expect_eq(day_of_the_week(6), "Sunday"sv);
    expect_eq(day_of_the_week(7), "Unknown"sv);
}
```

100% code coverage 

https://godbolt.org/z/qjGPcrf5d

EXAMPLE

```
void test(){
    expect_neq(day_of_the_week(0), "Unknown"sv);
    expect_neq(day_of_the_week(1), "Unknown"sv);
    expect_neq(day_of_the_week(2), "Unknown"sv);
    expect_neq(day_of_the_week(3), "Unknown"sv);
    expect_neq(day_of_the_week(4), "Unknown"sv);
    expect_neq(day_of_the_week(5), "Unknown"sv);
    expect_neq(day_of_the_week(6), "Unknown"sv);
    expect_eq(day_of_the_week(7), "Unknown"sv);
}
```

100% code coverage 

https://godbolt.org/z/qjGPcrf5d

WHAT DOES IT MEAN TO HAVE 100% CODE COVERAGE?

Low code coverage means you can do better, high code coverage does not mean anything

EXAMPLE

```
void test(){
    expect_eq(day_of_the_week(0), "Monday"sv);
    expect_eq(day_of_the_week(1), "Tuesday"sv);
    expect_eq(day_of_the_week(2), "Wednesday"sv);
    expect_eq(day_of_the_week(3), "Thursday"sv);
    expect_eq(day_of_the_week(4), "Friday"sv);
    expect_eq(day_of_the_week(5), "Saturday"sv);
    expect_eq(day_of_the_week(6), "Sunday"sv);
    expect_eq(day_of_the_week(7), "Unknown"sv);
}
```

Requirement: Weekday number is represented by a a digit from 1 through 7, beginning with Monday and ending with Sunday

https://godbolt.org/z/qjGPcrf5d

100% code coverage

0% requirement coverage

WHAT DOES IT MEAN TO HAVE 100% CODE COVERAGE?

Requirement coverage is more important than code coverage,

...,but still, you should have high code coverage.

WHY?

Someone said: “You must achieve 100% code coverage with unit tests”

ISO 26262 says:

9.4.4 To evaluate the completeness of verification and to provide evidence that the objectives for unit testing are adequately achieved, the coverage of requirements at the software unit level shall be determined and the structural coverage shall be measured in accordance with the metrics as listed in Table 9. If the achieved structural coverage is considered insufficient, either additional test cases shall be specified or a rationale based on other methods shall be provided.

Table 9 — Structural coverage metrics at the software unit level

Methods		ASIL			
		A	B	C	D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

STATEMENT COVERAGE

```
int foo_bar(bool foobar, int foo, int bar){
    if(foobar){
       return foo + bar;
    }
    return bar;
}
```

- Statements in the function body:
  - 1 statement for the if
  - 1 statement for each return
  - 3 statements in total

```
expect_eq(foo_bar(true, 32, 10), 42);
```

- Statements executed:
  - if statement
  - return foo + bar;
  - 2 statements in total

- 66.6% statement coverage

BRANCH COVERAGE

```
int foo_bar(bool foobar, int foo, int bar){
    if(foobar){
       return foo + bar;
    }
    return bar;
}
```

```
expect_eq(foo_bar(true, 32, 10), 42);
```

- Branches in the function body:
  - 1 branch when the if condition evaluates to true
  - 1 branch when the if condition evaluates to false
  - 2 branches in total

- Brances executed:
  - The branch where the body of the if is executed
  - 1 branch in total

- 50% branch coverage

MC/DC (MODIFIED CONDITION/DECISION COVERAGE)

```
bool got_to_work(bool weekend, bool national_holiday, bool feeling_sick){
    if(feeling_sick || national_holiday || weekend){
       return false;
    }
    return true;
}
```

Weekend	National_holiday	Feeling_sick
False	False	False
False	False	True
False	True	False
False	True	True
True	False	False
True	True	False
True	True	True

MC/DC (MODIFIED CONDITION/DECISION COVERAGE)

```
bool got_to_work(bool weekend, bool national_holiday, bool feeling_sick){
    if(feeling_sick || national_holiday || weekend){
       return false;
    }
    return true;
}
```

Weekend	National_holiday	Feeling_sick
False	False	False
False	False	True
False	True	False
False	True	True
True	False	False
True	True	False
True	True	True

MC/DC (MODIFIED CONDITION/DECISION COVERAGE)

```
bool got_to_work(bool weekend, bool national_holiday, bool feeling_sick){
    if(feeling_sick || national_holiday || weekend){
       return false;
    }
    return true;
}
```

Weekend	National_holiday	Feeling_sick
False	False	False
False	False	True
False	True	False
False	True	True
True	False	False
True	True	False
True	True	True

MC/DC (MODIFIED CONDITION/DECISION COVERAGE)

```
bool got_to_work(bool weekend, bool national_holiday, bool feeling_sick){
    if(feeling_sick || national_holiday || weekend){
       return false;
    }
    return true;
}
```

Weekend	National_holiday	Feeling_sick
False	False	False
False	False	True
False	True	False
False	True	True
True	False	False
True	True	False
True	True	True

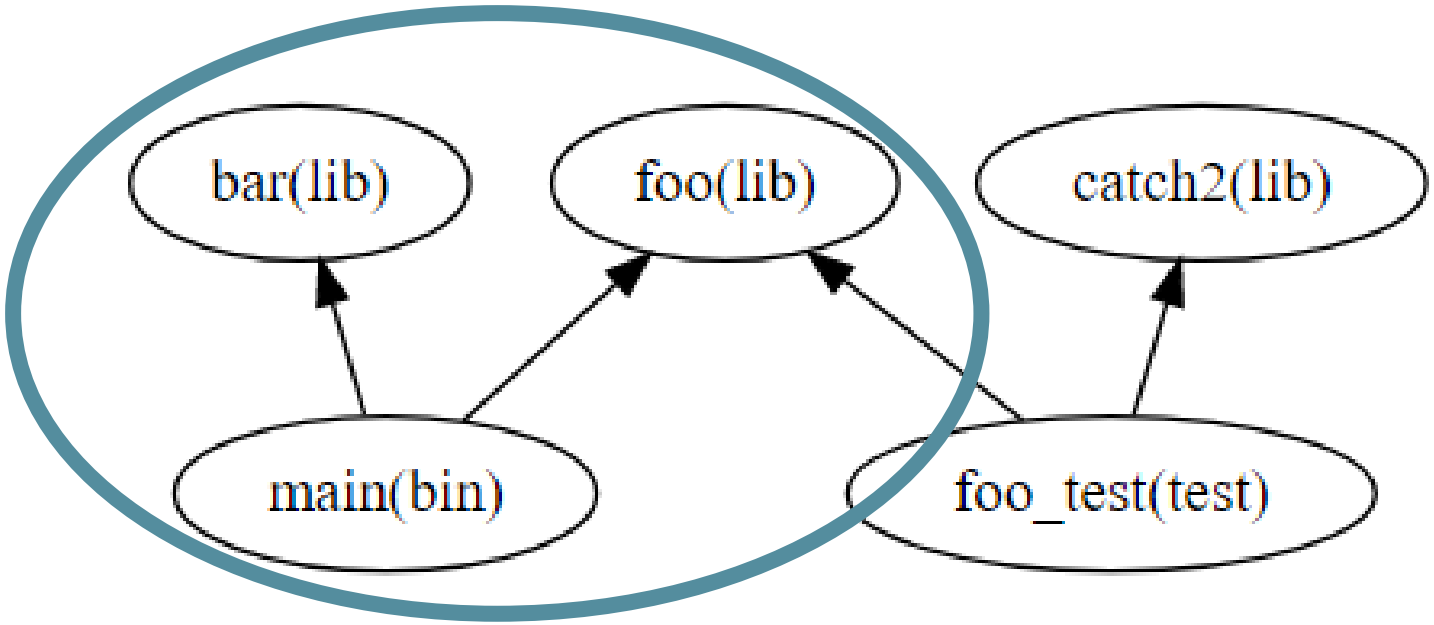
MC/DC (MODIFIED CONDITION/DECISION COVERAGE)

```
bool got_to_work(bool weekend, bool national_holiday, bool feeling_sick){
    if(feeling_sick || national_holiday || weekend){
       return false;
    }
    return true;
}
```

Weekend	National_holiday	Feeling_sick
False	False	False
False	False	True
False	True	False
False	True	True
True	False	False
True	True	False
True	True	True

BASELINE COVERAGE

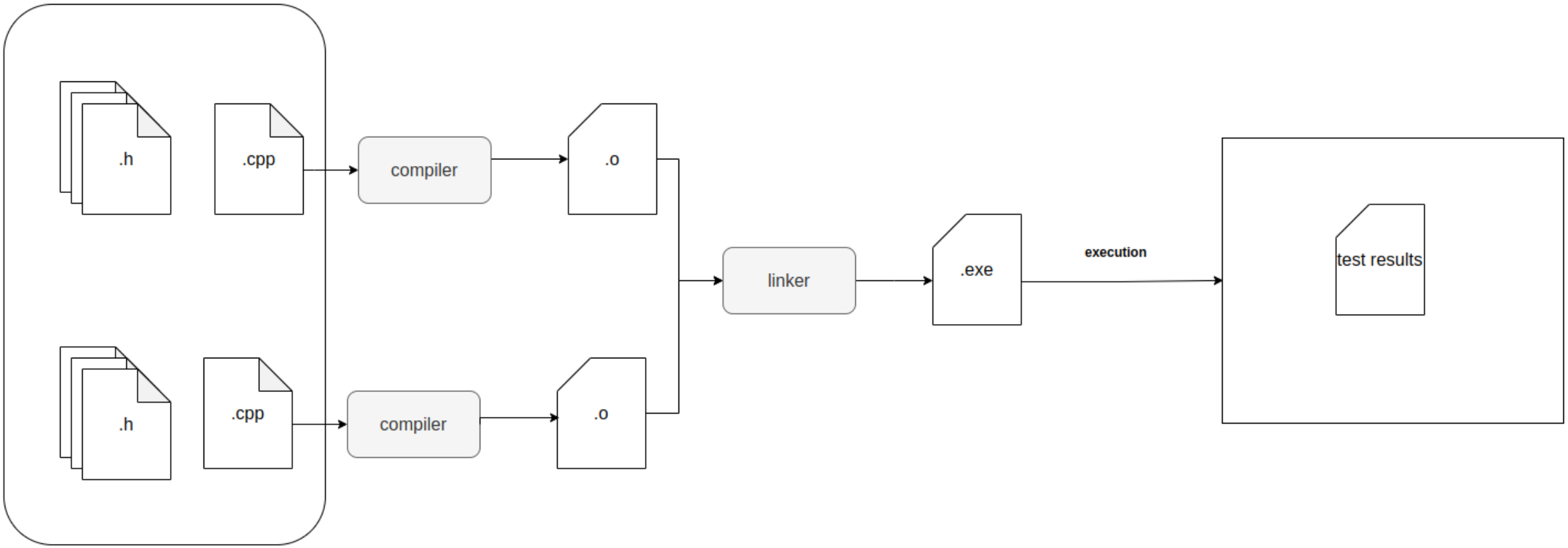
$$\frac{\text{Nº of executed statements}}{\text{Nº of statements to be tested}} \cdot 100$$



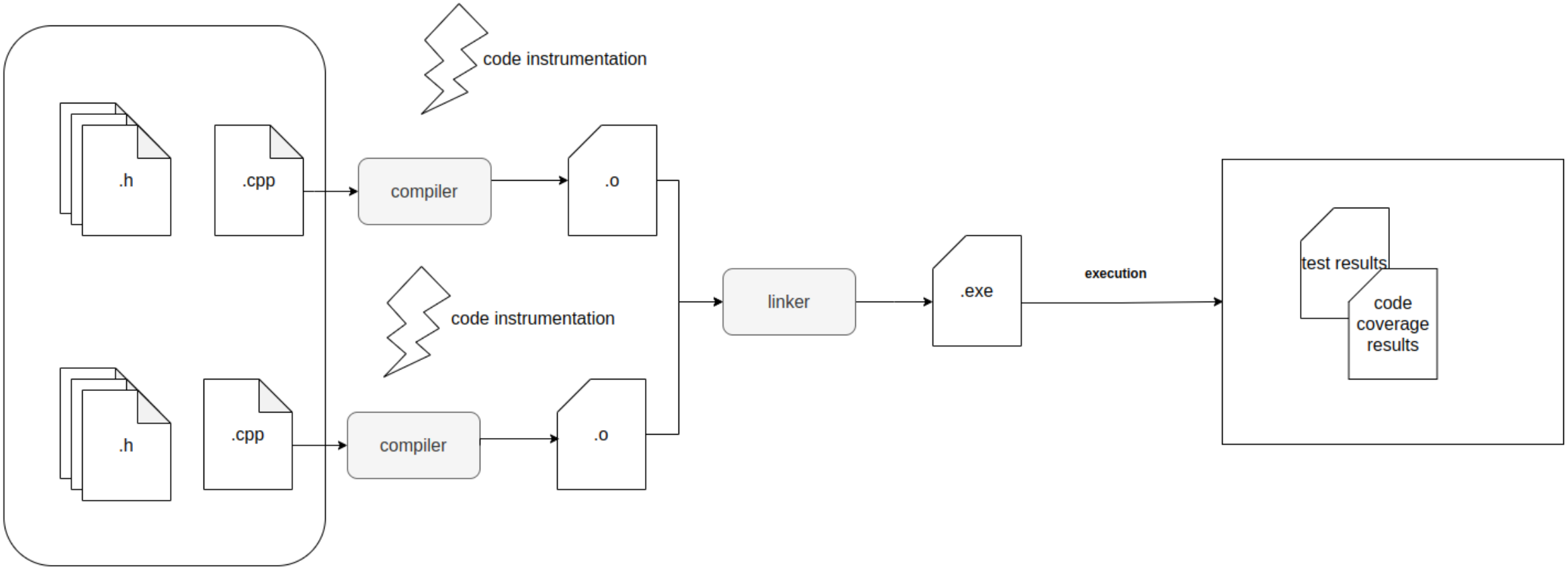
CODE COVERAGE PIPELINE

So, what really enables us to check our code coverage?

CODE COVERAGE PIPELINE (BACK TO THE BASICS - COMPILATION)



CODE COVERAGE PIPELINE (BACK TO THE BASICS)



CODE INSTRUMENTATION

- Code instrumentation adds some extra code to the code a compiler normally generates, for various purposes:
  - Perf statistics
  - Profiling
  - Run-time checks (sanitizers – eg: check for invalid pointer dereferences or out-of-bounds array accesses)
  - And also, code coverage…

```
mov     rax, QWORD PTR __gcov0.main[rip]
add     rax, 1
mov     QWORD PTR __gcov0.main[rip], rax
```

DIFFERENT TOOLS

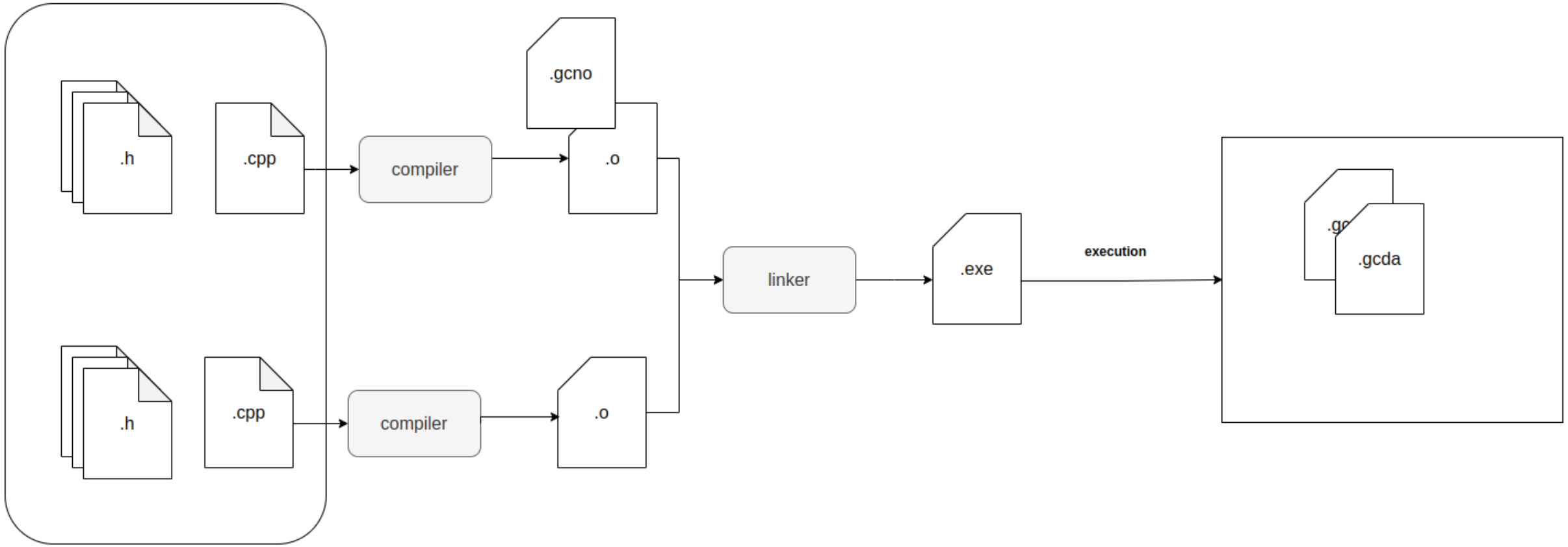
- In general terms, a code coverage tool will help collect information about a running program and will hopefully combine it with the program source information to generate a user-friendly report.
  - There are lots of different tools:
    - gcov (a lot of times used with lcov)
    - llvm-cov
    - gcovr
    - CTC++ (commercial)
    - Bullseye Coverage (commercial)

ABOUT YOU

- How many of you people know/use gcov?
- How many of you people know/use llvm-cov?
- How many of you people know/use gcovr?
- How many of you people use other tools?

GCOV

For gcov based coverage, the reports are generated based on `-fprofile-arcs` (and `-ftest-coverage`).



https://gcc.gnu.org/onlinedocs/gcc/Gcov.html

CONTROL FLOW GRAPH

- Is a representation, that models possible execution paths through a program.

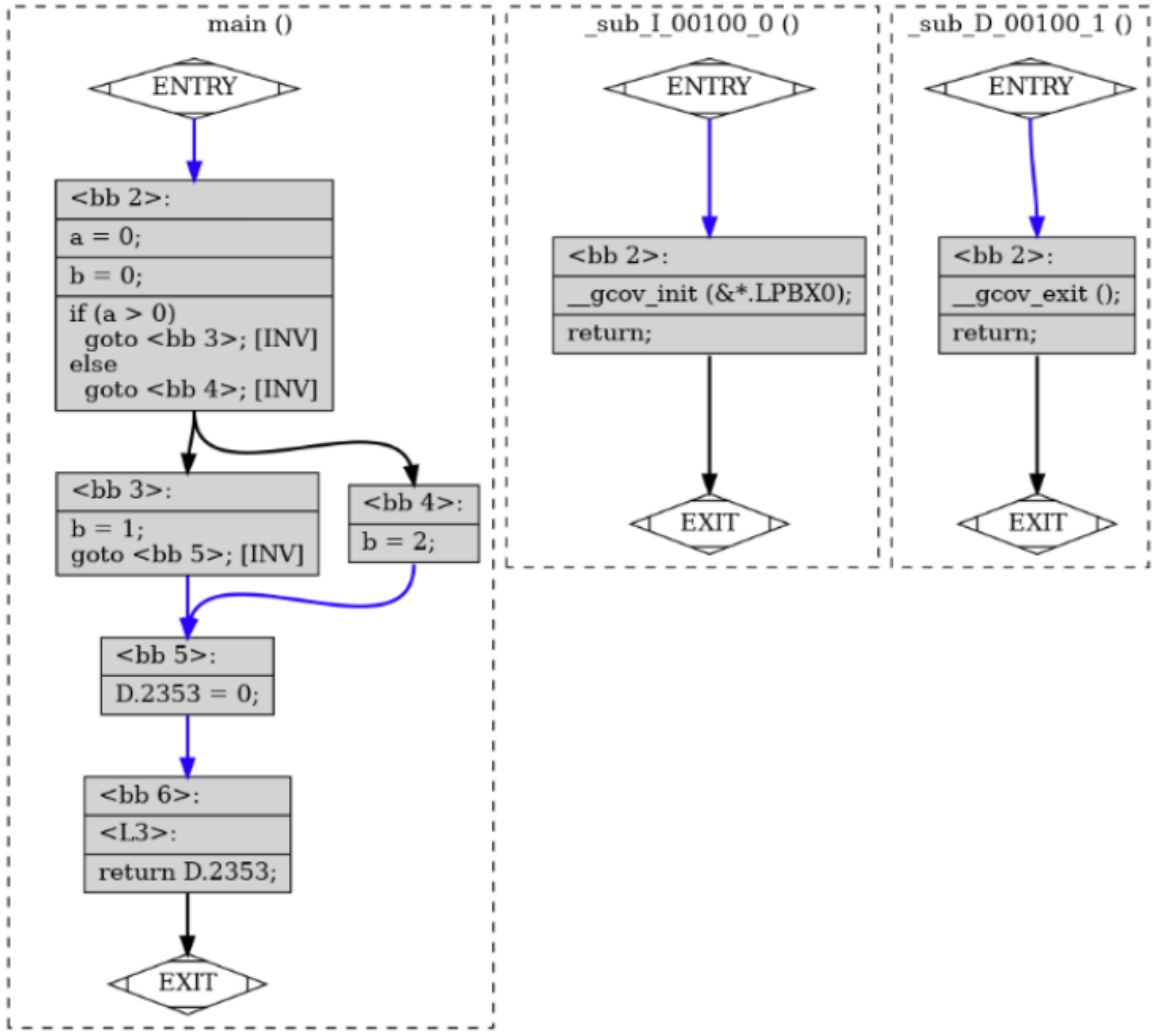
- In this graph:
  - Nodes are basic blocks and an arc connects two nodes that represents a possible transfer of control.
  - A basic block is a linear sequence of instructions, containing no branches except at the end.

- For example, for the following code:

```
// foo.cpp
int main() {
    int a{};
    int b{};
    if(a > 0) {
        b = 1;
    } else {
        b = 2;
    }
    return 0;
}
```

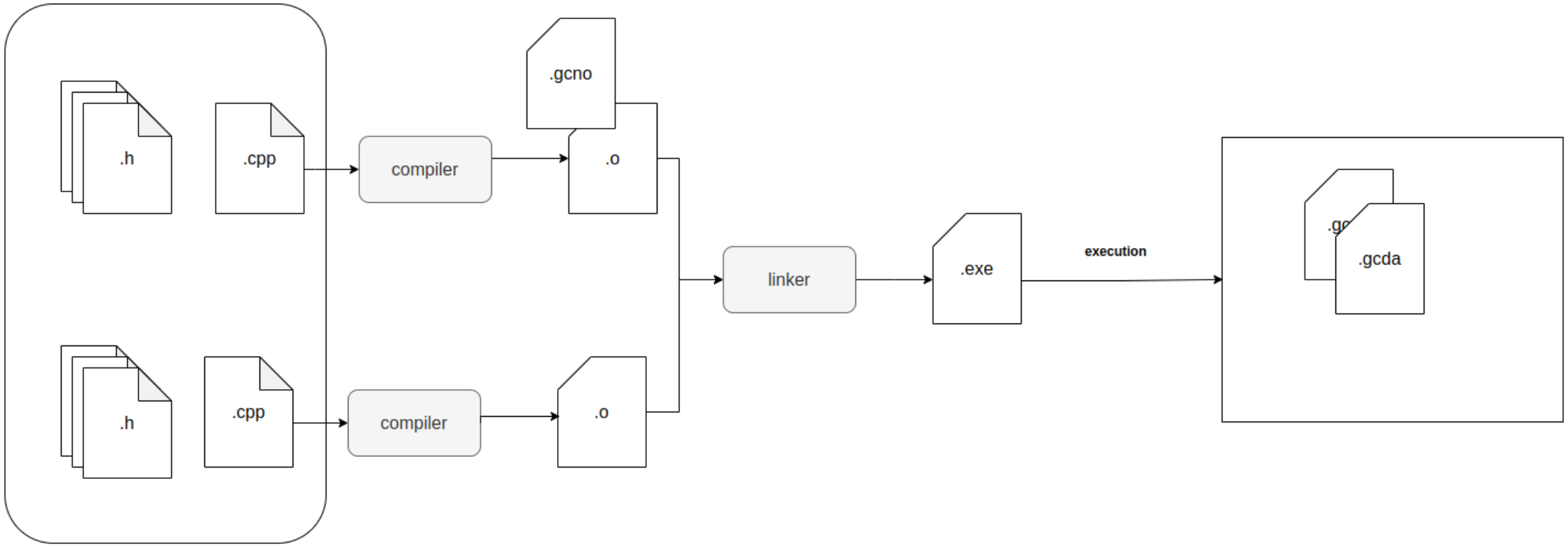
CONTROL FLOW GRAPH

- One can have the following CFG (generated with `-fdump-tree-cfg`):



GCOV

This coverage data will be saved in *.gcda files. The *.gcno files are generated at compile time, and it will help to reconstruct the basic blocks and map them to source code.

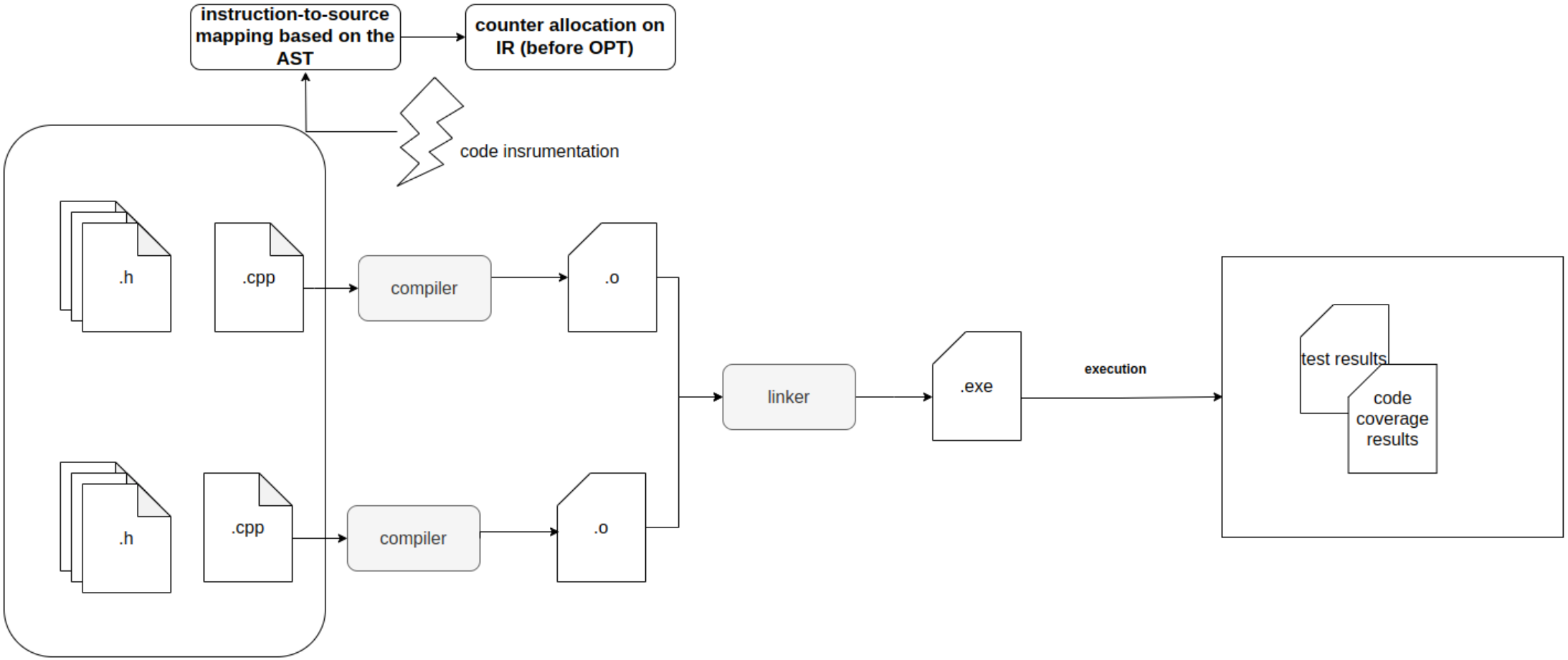


https://gcc.gnu.org/onlinedocs/gcc/Gcov.html

LLVM-COV

- https://www.youtube.com/watch?v=H1hvtJPGWNQ
- llvm-cov docs: https://clang.llvm.org/docs/SourceBasedCodeCoverage.html

LLVM-COV



- llvm-cov docs: https://clang.llvm.org/docs/SourceBasedCodeCoverage.html

GCOVR

- "gcovr provides a utility for managing the use of the GNU gcov utility and generating summarized code coverage results."
- This can be seen as an alternative for lcov and has very interesting features but here one will focus more on the --decisions feature that citing its documentation:
- "With the gcovr --decisions option, gcovr parses the source code to extract a ISO 26262 compliant metric for decision coverage. This metric can be interpreted as the branch coverage on C/C++-Level. While the feature is not always able to detect the decisions reliably when the code is written very compact (uncheckable decisions will be marked), it provides a reliable tool for (i.e. MISRA-compliant) code in security-relevant situations."

- https://gcovr.com/en/stable/
- https://github.com/linux-test-project/lcov

CODE EXAMPLES

Here we try to provide you with some examples, that behave differently between tools, it is not complete for sure, and it represents mostly our experience:

- if with various conditions
- templates
- Static initialization
- constexpr (consteval)
- Exception branches

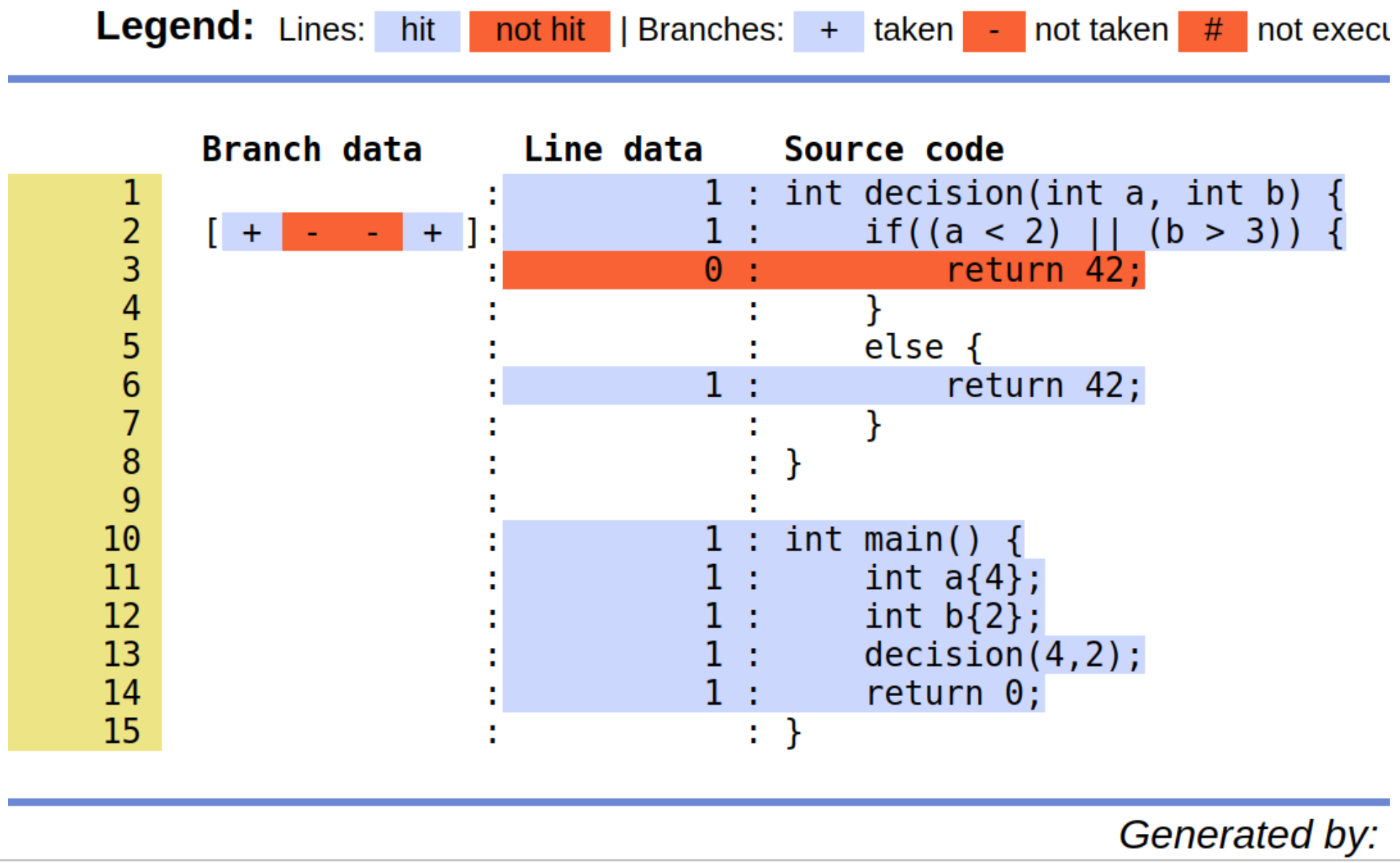
For each of these we will briefly show how gcov, llvm-cov, and gcovr, behave and what info they provide.

IF WITH VARIOUS CONDITIONS

```
unsigned decision(unsigned a, unsigned b) {
    if((a < 2) || (b > 3)) {
        return 42;
    }
    else {
        return 43;
    }
}

int main() {
    decision(4,2);
    return 0;
}
```

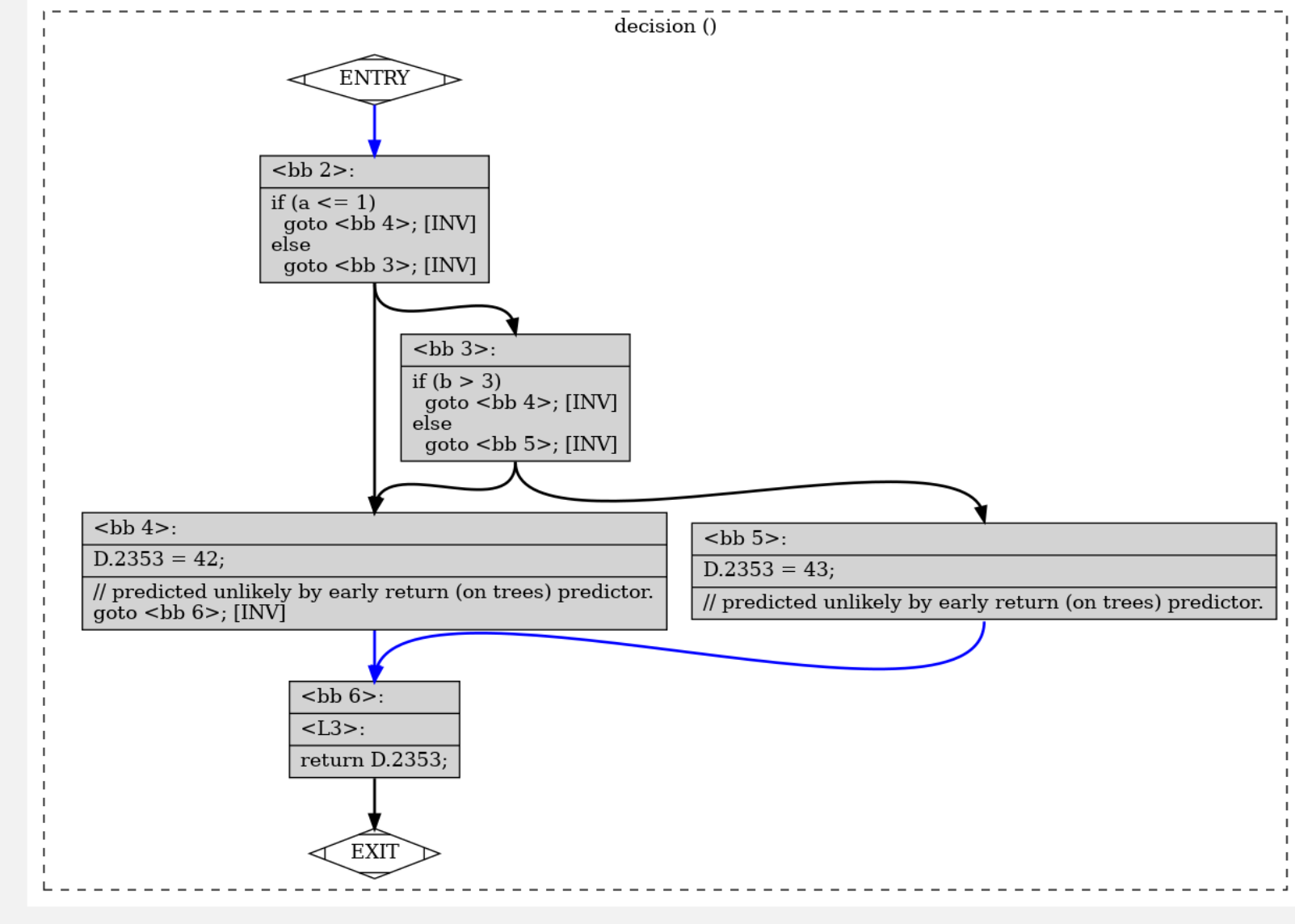
IF WITH VARIOUS CONDITIONS - GCOV



IF WITH VARIOUS CONDITIONS - GCOV

```
15  int decision (int a, int b)
16  {
17    int D.2357;
18
19    <bb 2> :
20    if (a <= 1)
21      goto <bb 4>; [INV]
22    else
23      goto <bb 3>; [INV]
24
25    <bb 3> :
26    if (b > 3)
27      goto <bb 4>; [INV]
28    else
29      goto <bb 5>; [INV]
30
31    <bb 4> :
32    D.2357 = 42;
```

IF WITH VARIOUS CONDITIONS - GCOV



IF WITH VARIOUS CONDITIONS - LLVM-COV

Line	Count	Source (jump to first uncovered line)
1	1	`int decision(int a, int b) {`
2	1	`    if((a < 2) || (b > 3)) {`
		Branch (2:8): [True: 0, False: 1] Branch (2:19): [True: 0, False: 1]
3	0	`        return 42;`
4	0	`    }`
5	1	`    else {`
6	1	`        return 42;`
7	1	`    }`
8	1	`}`
9		
10	1	`int main() {`
11	1	`    int a{4};`
12	1	`    int b{2};`
13	1	`    decision(4,2);`
14	1	`    return 0;`
15	1	`}`

IF WITH VARIOUS CONDITIONS - GCOVR

▸ List of functions

Line	Branch	Decision	Exec	Source
1			1	`int decision(int a, int b) {`
2	▶ 2/4	▶ 1/2	1	`    if((a < 2) || (b > 3)) {`
3			✗	`        return 42;`
4				`    }`
5				`    else {`
6			1	`        return 42;`
7				`    }`
8				`}`
9				
10			1	`int main() {`
11			1	`    int a{4};`
12			1	`    int b{2};`
13			1	`    decision(4,2);`
14			1	`    return 0;`
15				`}`
16				

TEMPLATES

```
template <typename T>
T Template(T t)
{
    if (t > 0)
        return t * t;
    else
        return t;
}

int main()
{
    Template(-1);
    Template(1.0);
    return 0;
}
```

TEMPLATES – INSTANTIATION WITH GCC

```
// g++ templates.cpp -O0 -fdump-tree-cfg -fdump-tree-all-graph
;; Function int main() (null)
;; enabled by -tree-original


<<cleanup_point <<< Unknown tree: expr_stmt
  (void) Template<int> (-1) >>>>>;
<<cleanup_point <<< Unknown tree: expr_stmt
  (void) Template<double> (1.0e+0) >>>>>;
return <retval> = 0;
return <retval> = 0;
```

```
;; Function T Template(T) [with T = int] (null)
;; enabled by -tree-original


if (t > 0)
  {
    return <retval> = t * t;
  }
else
  {
    return <retval> = t;
  }
```

```
;; Function T Template(T) [with T = double] (null)
;; enabled by -tree-original


if (t > 0.0)
  {
    return <retval> = t * t;
  }
else
  {
    return <retval> = t;
  }
```

TEMPLATES – INSTANTIATION WITH CLANG

```
// clang++ -Xclang -ast-print -fsyntax-only templates.cpp
template <typename T> T Template(T t) {
    if (t > 0)
        return t * t;
    else
        return t;
}
template<> int Template<int>(int t) {
    if (t > 0)
        return t * t;
    else
        return t;
}
template<> double Template<double>(double t) {
    if (t > 0)
        return t * t;
    else
        return t;
}
int main() {
    Template(-1);
    Template(1.);
    return 0;
}
```

TEMPLATES – GCOV

Current view: top level - root - templates.cpp (source / functions)
Test: commit SHA1
Date: 2024-04-22 08:54:25
Legend: Lines: hit | not hit | Branches: + taken | - not taken | # not executed

Branch data	Line data	Source code
	1	template <typename T>
	2	T Template(T t)
	3	{
[+ +]	2	if (t > 0)
	1	return t * t;
	4	else
	1	return t;
	5	}
	6	
	1	int main()
	7	{
	1	Template(-1);
	1	Template(1.0);
	1	return 0;
	8	}

Genei

TEMPLATES – LLVM-COV

Line	Count	Source
1		template <typename T>
2		T Template(T t)
3	2	{
4	2	if (t > 0)
		Branch (4:9): [True: 0, False: 1] Branch (4:9): [True: 1, False: 0]
5	1	return t * t;
6	1	else
7	1	return t;
8	2	}
9		
10		int main()
11	1	{
12	1	Template(-1);
13	1	Template(1.0);
14	1	return 0;
15	1	}

int Template<int>(int)

Line	Count	Source
3	1	{
4	1	if (t > 0)
		Branch (4:9): [True: 0, False: 1]
5	0	return t * t;
6	1	else
7	1	return t;
8	1	}

double Template<double>(double)

Line	Count	Source
3	1	{
4	1	if (t > 0)
		Branch (4:9): [True: 1, False: 0]
5	1	return t * t;
6	0	else
7	0	return t;
8	1	}

TEMPLATES – GCOVR

▶ List of functions

Line	Branch	Decision	Exec	Source
1				`template <typename T>`
2			4	`T Template(T t)`
3				`{`
4	▶ 2/2	▶ 2/2	4	`    if (t > 0)`
5			2	`        return t * t;`
6				`    else`
7			2	`        return t;`
8				`}`
9				
10			1	`int main()`
11				`{`
12			1	`    Template(-1);`
13			1	`    Template(1.0);`
14			1	`    return 0;`
15				`}`
16				

A LOT OF BRANCHES

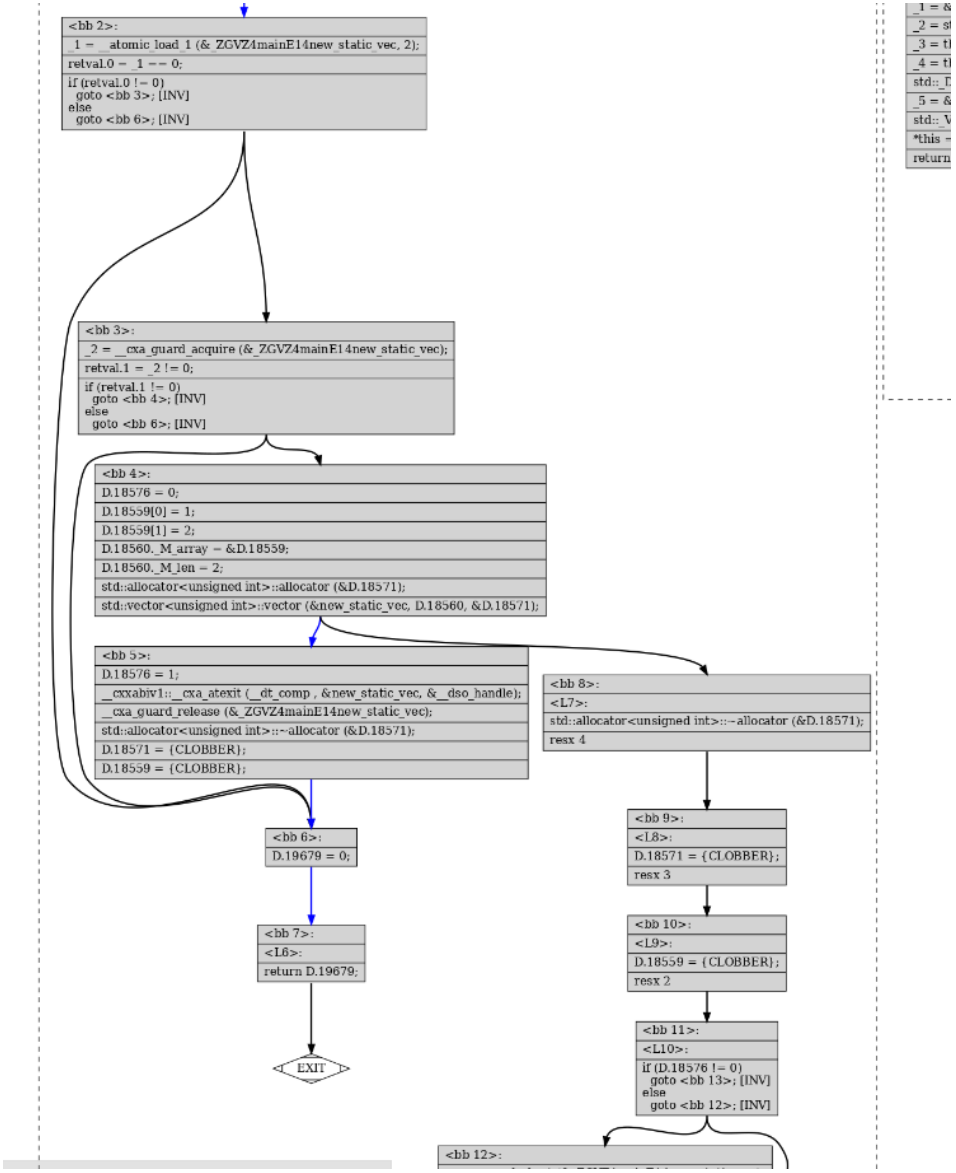
```
#include <vector>

int main() {
    static std::vector<unsigned>new_static_vec{1,2};
}
```

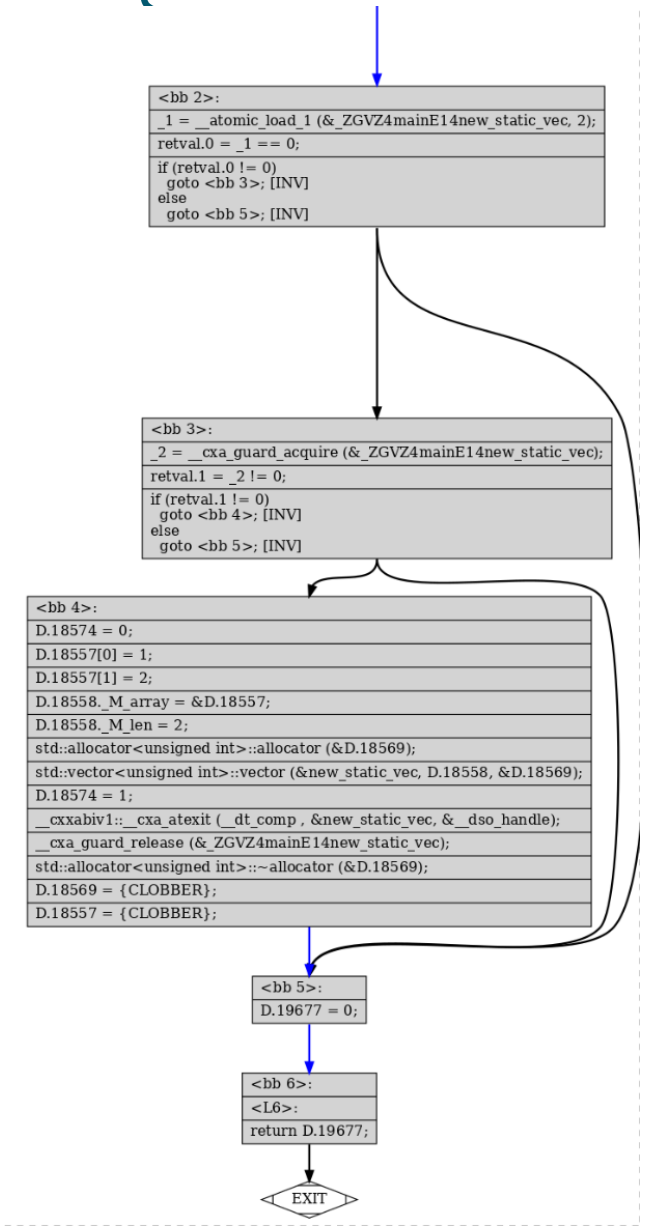
A LOT OF BRANCHES - GCOV

```
        Branch data     Line data    Source code
      1                 :            : #include <vector>
      2                 :            :
      3                 :          1 : int main() {
      4  [ +   -   +   - :          1 :     static std::vector<unsigned>new_static_vec{1,2};
           +   -   -   - ]
      5                 :          1 : }
```

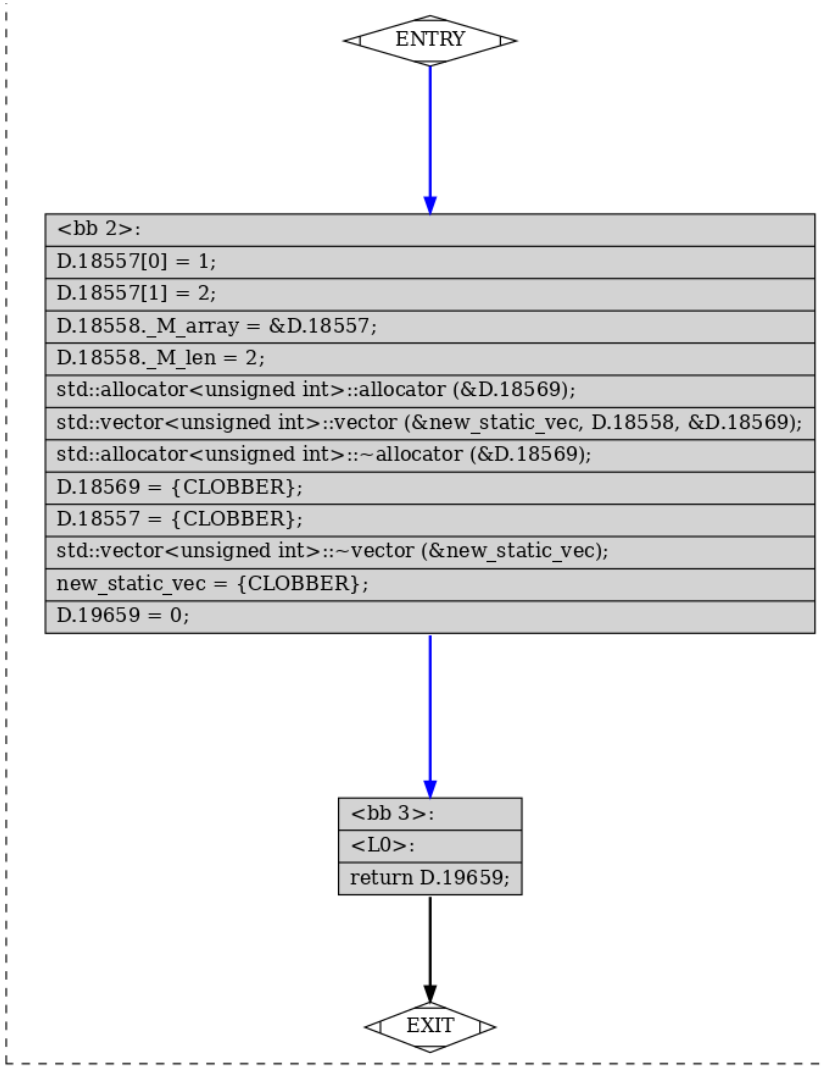
A LOT OF BRANCHES - GCOV



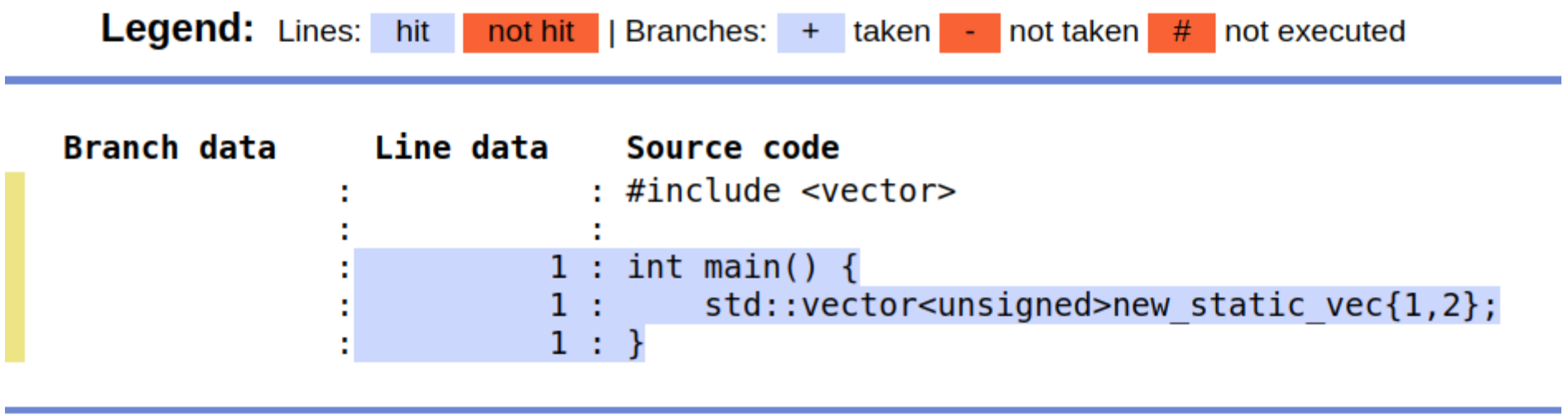
A LOT OF BRANCHES – GCOV (WITH –FNO-EXCEPTIONS)



A LOT OF BRANCHES – GCOV WITH –FNO-EXCEPTIONS NOT STATIC



A LOT OF BRANCHES – GCOV WITH –FNO-EXCEPTIONS NOT STATIC



A LOT OF BRANCHES - LLVM-COV

ne	Count	Source
1		#include <vector>
2		
3	1	int main() {
4	1	std::vector<unsigned>new_static_vec{1,2};
5	1	}

A LOT OF BRANCHES - GCOVR

▶ List of functions

Line	Branch	Decision	Exec	Source
1				`#include <vector>`
2				
3			1	`int main() {`
4	▶ 3/8		1	`    static std::vector<unsigned>new_static_vec{1,2};`
5			1	`}`
6				

CONSTEXPR

```
constexpr int absolute(int x) {
    return (x >= 0) ? x : -x;
}

int main() {
    [[maybe_unused]] int abs_num = absolute(1);
    [[maybe_unused]] int abs_num1 = absolute(-11);
    return 0;
}
```

CONSTEXPR - GCOV

```
    Branch data     Line data    Source code
1                 :              : constexpr int absolute(int x) {
2                 :              :     return (x >= 0) ? x : -x;
3                 :              : }
4                 :              :
5                 :            1 : int main() {
6                 :            1 :     [[maybe unused]] int abs_num = absolute(1);
7                 :            1 :     [[maybe unused]] int abs_num1 = absolute(-11);
8                 :            1 :     return 0;
9                 :              : }
```

CONSTEXPR - GCOV

```
Removing basic block 3
;; 1 loops found
;;
;; Loop 0
;;  header 0, latch 1
;;  depth 0, outer -1
;;  nodes: 0 1 2 3
;; 2 succs { 3 }
;; 3 succs { 1 }
int main ()
{
  int abs_num1;
  int abs_num;
  int D.2355;

  <bb 2> :
  abs_num = 1;
  abs_num1 = 11;
  D.2355 = 0;

  <bb 3> :
<L0>:
  return D.2355;

}
```

CONSTEXPR – LLVM-COV

Line	Count	Source
1	2	constexpr int absolute(int x) {
2	2	return (x >= 0) ? x : -x;
		Branch (2:12): [True: 1, False: 1]
3	2	}
4		
5	1	int main() {
6	1	[[maybe_unused]] int abs_num = absolute(1);
7	1	[[maybe_unused]] int abs_num1 = absolute(-11);
8	1	return 0;
9	1	}

CONSTEXPR – GCOVR

▶ List of functions

Line	Branch	Decision	Exec	Source
1				`constexpr int absolute(int x) {`
2				`    return (x >= 0) ? x : -x;`
3				`}`
4				
5			1	`int main() {`
6			1	`    [[maybe_unused]] int abs_num = absolute(1);`
7			1	`    [[maybe_unused]] int abs_num1 = absolute(-11);`
8			1	`    return 0;`
9				`}`
10				

SUMMARY

- High code coverage does not necessary mean high quality
- Don't forget about requirement coverage
- Always be aware of your baseline coverage
- Know your tool
  - Different tools produce different reports

REFERENCES

- https://testing.googleblog.com/2020/08/code-coverage-best-practices.htm
- https://www.ibm.com/docs/en/hla-and-tf/1.6?topic=areas-control-flow-graph-window
- https://en.wikipedia.org/wiki/Control-flow_graph
- https://gcovr.com/en/stable/index.html
- https://gcc.gnu.org/onlinedocs/gcc-10.1.0/gcc/Instrumentation-Options.html
- https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
- https://man7.org/linux/man-pages/man1/gcov.1.html
- https://llvm.org/devmtg/2020-09/slides/PhippsAlan_BranchCoverage_LLVM_Conf_Talk_final.pdf

EVERYTHING YOU NEED TO KNOW ABOUT CODE COVERAGE IN C++

XAVIER BONAVENTURA – JORGE PINTO SOUSA

using std::cpp 2024