Fortan Debugging Introduction

Andrés Pérez Hortal

April 6, 2018

What is this presentation about?

- Review the **most common errors** in (Fortran) computer programs
- Present **simple techniques** to **find, correct and avoid these errors** in codes.

What is a computer error or "bug"?

- **Software bug** : An error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

Why "bug"?

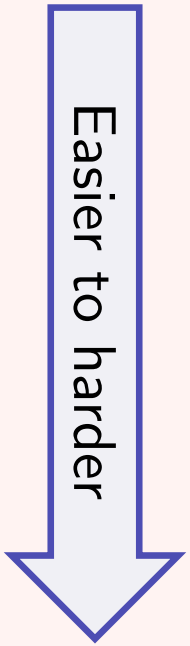
- **Popular attribution**: In 1946, Grace Hopper traced an error in the Mark II to a moth trapped in a relay, a "bug".
- **First use** dates back to Thomas Edison in 1876 to refer to the failures in his experiments.

What is debugging?

- The process of **finding and resolving** defects or problems within the program that prevent correct operation of computer software.
- Debugging is not only about finding and solving problems with your code. It is also increasing your understanding of how the code works.

Types of errors

- **Compilation**
- **Runtime**
- **Logic**



Compilation errors

- Errors that appears during the compilation process (translating code to machine language)
- Detected and pointed out by the compiler.
- The line number in the source code where the problem is located is typically indicated by the compiler.

Runtime errors

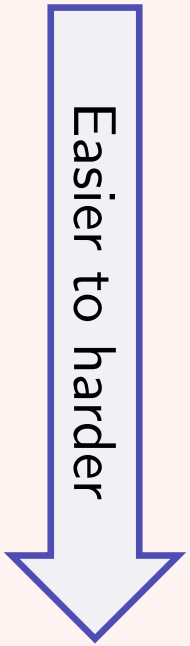
- Error that occurs when you run your program
- The failure is a result of an error that propagates and produce a failure in the program.
- Some of them can be prevent by the compiler! More on compiler warnings coming next.
- Difficult to solve
- E.g.: Memory leaks, segmentation fault, infinity loops, more segmentation faults...

Logical errors

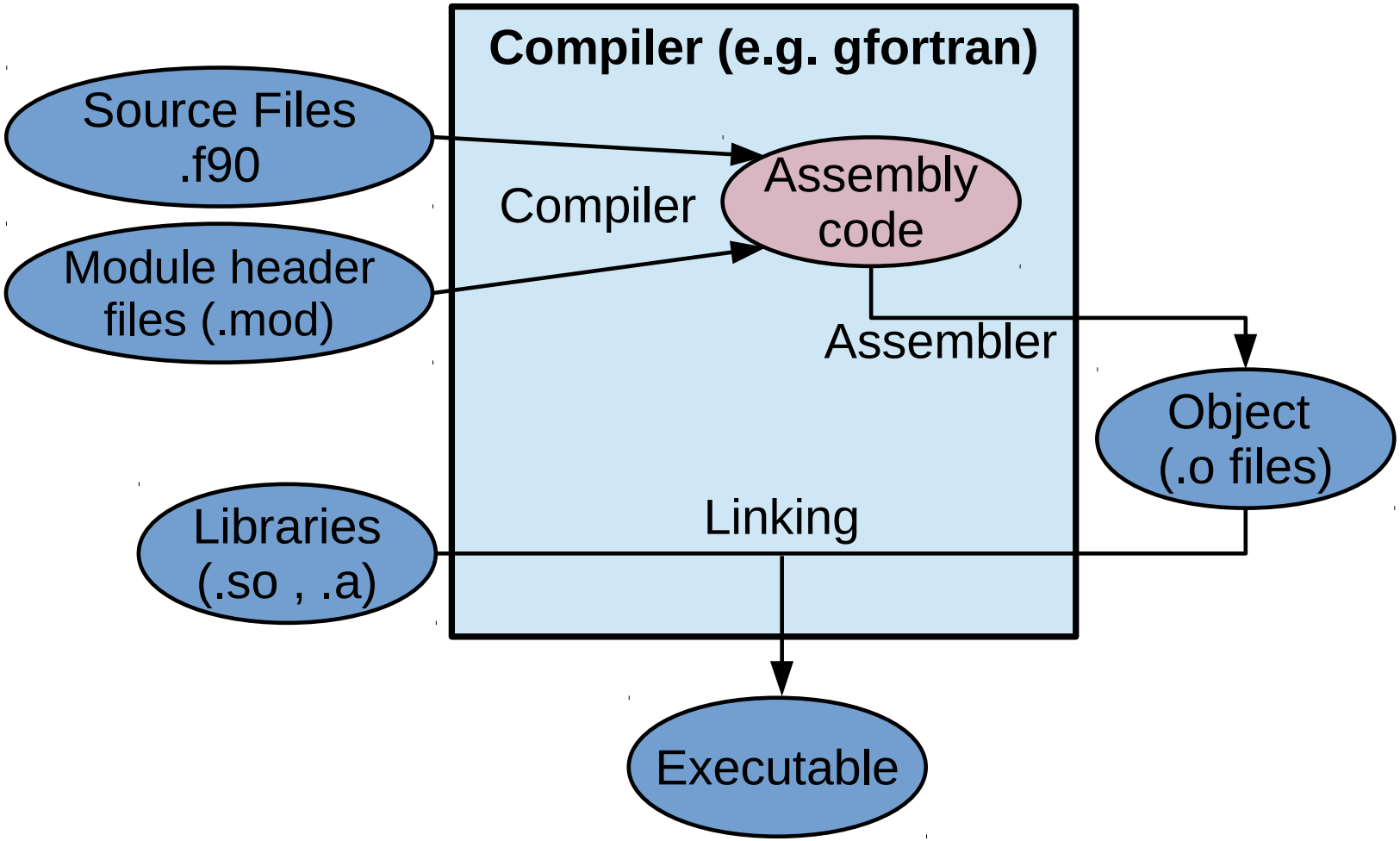
- The program appears to run correctly, by it gives wrong or unexpected results
- Most difficult to correct
- One approach to solve them is to test different parts of the program with inputs with known outputs.

Types of errors

- **Compilation**
- Runtime
- Logic



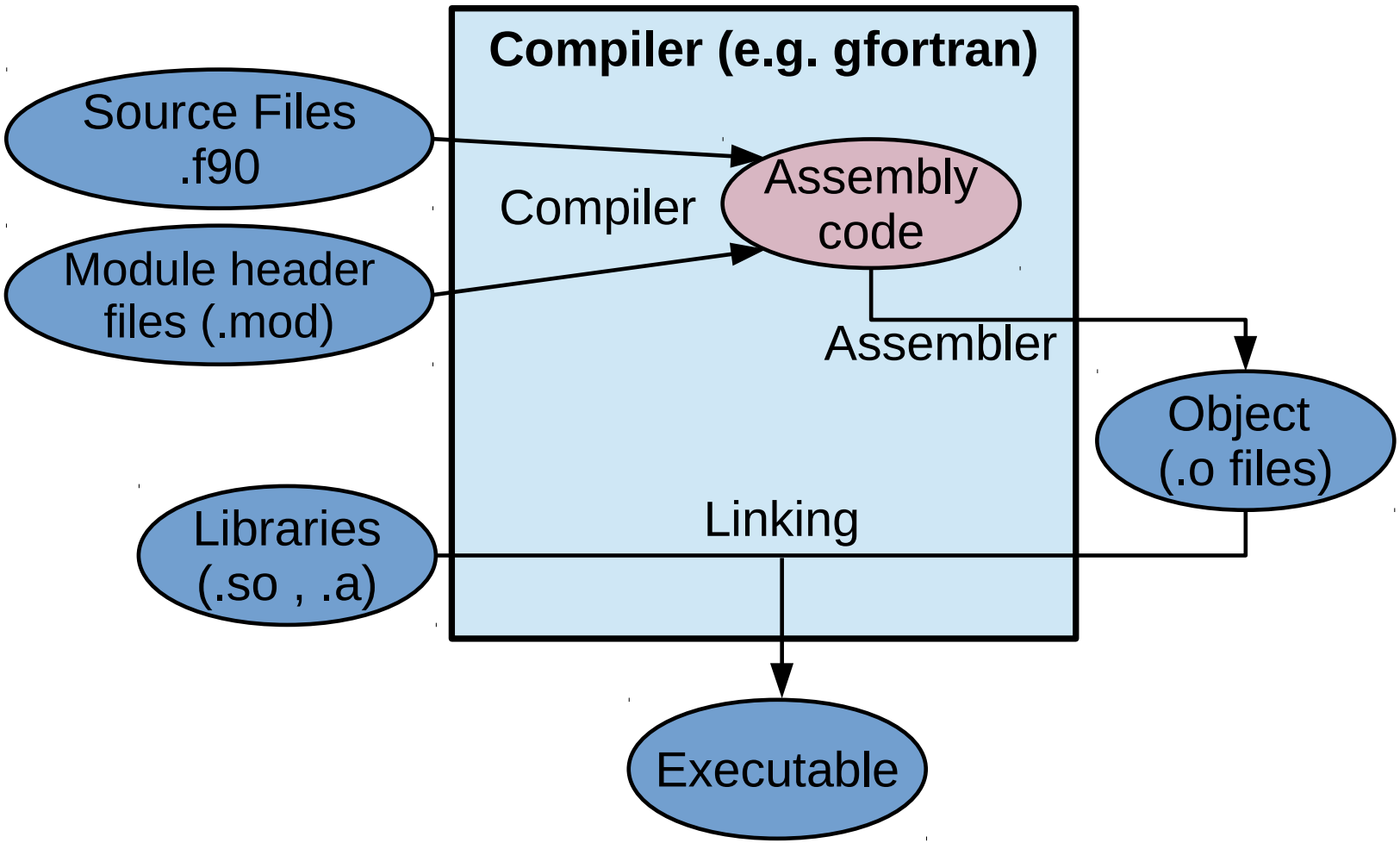
Understanding the compilation process



Differences between header and sources

- The dependency structure in Fortran 90 is somewhat complex and compiler-dependent.
- The basic model is like C with source (.f90) and header (.mod) files.
- The **source** codes defines the **functions, subroutine or the main program**.
- The **header** (.mod) files specify **interfaces, derived types, and similar information**.
- **The header is the interface to whatever is implemented in the corresponding sources (.f90) files.**

What is an object?



What is an object?

- Each source file is translated (compiled) separately by the compiler into an object (.o) file.
- Provides external symbols that can be used by other objects and libraries (exports).
- List of symbols expected from other objects and libraries (imports) .

Source example:simple_module.90

```
MODULE simple_module
  IMPLICIT NONE
    PRIVATE multiply
  CONTAINS

    REAL FUNCTION multiply(a, b) !Visible inside the module (PRIVATE)
      REAL, INTENT(IN) :: a, b
      multiply = a*b
    END FUNCTION
    !volume is visible from outside the module (PUBLIC)
    REAL FUNCTION volume(width, height, depth)
      REAL :: rectagle_area !external function!
      REAL, INTENT(IN) :: width, height, depth
      REAL :: my_area
      my_area = rectagle_area(width ,height )
      volume = multiply(my_area, depth)
    END FUNCTION
END MODULE simple_module
```

Object example: simple_module.o

```
$ gfortran -c simple_module.f90 # Compile source
$ nm simple_module.o # Object viewer
                 U rectagle_area_                    #"U" undefined symbol
0000000000000062 t __simple_module_MOD_multiply #"t" internal symbol
0000000000000000 T __simple_module_MOD_volume #"T" external symbol
```

Symbols: variables, functions, or subroutines

Important: Remember that the objects may have undefined symbols!!

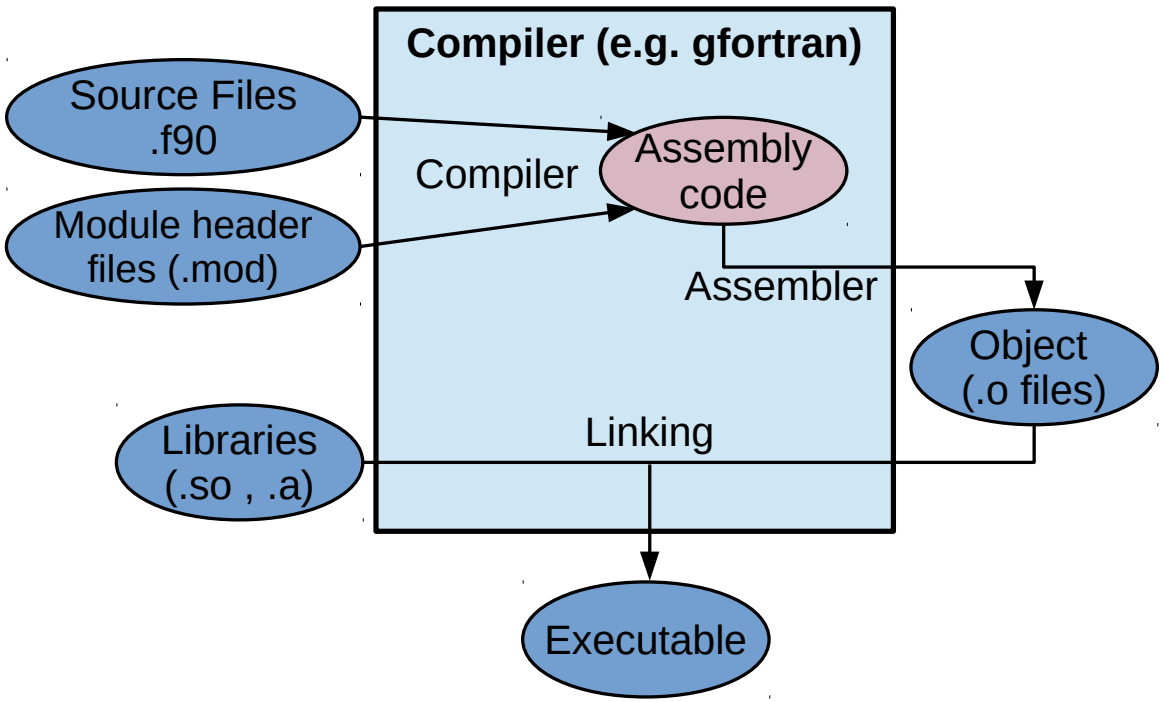
We are going to come back to this later.

What is a library?

- A collection of object files. Just a bunch of object files glued together.
- Files are built from compiled object files (.o files)
- **Static libraries** (.a files): Archives of objects. Objects are copied to the executable during linking.
- **Dynamic libraries** (.so files): The suffix stands for "shared object". All the applications that are linked with the library use the same file rather than making a copy in the resulting executable.

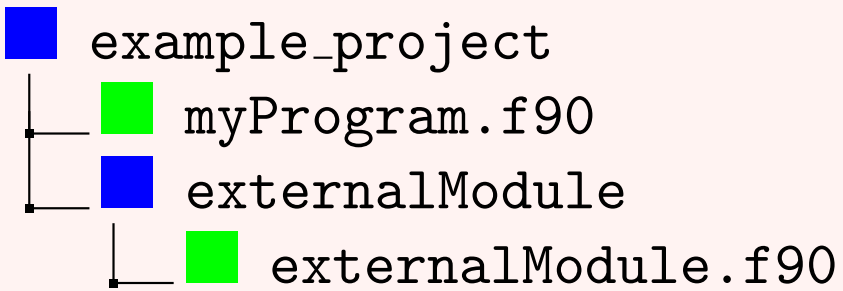
What is linking process?

- Combines one or more object files into a single executable file, library file, or another 'object' file.



Program compilation example 1

Program that use an external module



myProgram calls a function in **externalModule**

Compilation steps

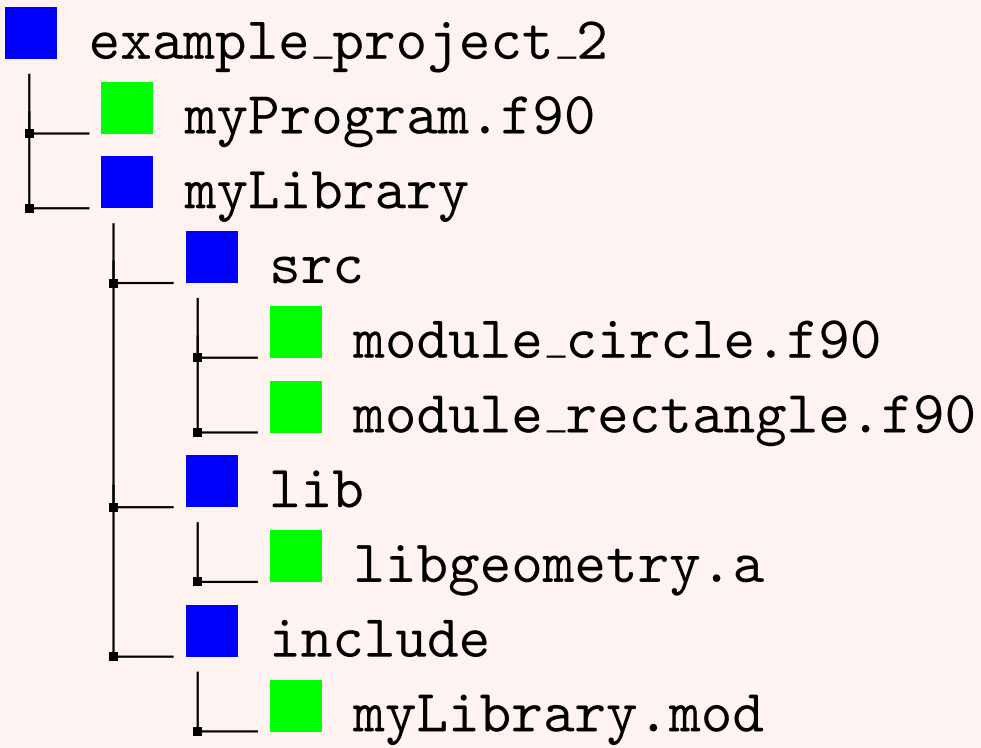
- Compile **externalModule.f90** (.o and .mod files)
- Compile **myProgram.f90** (.o)
- Link **myProgram** with **externalModule**

Program compilation example 1

```
### Compilation Procedure ###

# Compile the external module (no linking!)
cd externalModule # Change directory
#   -c : Compile to an object file
#   -o : Specifies the name of the output file.
gfortran -c externalModule.f90 -o externalModule.o

# Compile the myProgram.f90 program (no linking!)
cd .. # Return to the project dir
#  -I : adds include directory of header (.mod) files
gfortran -I./externalModule -c myProgram.f90 -o myProgram.o

# Link myProgram and the external module to create the executable
gfortran myProgram.o externalModule/externalModule.o -o myProgram
```

Program compilation example 2

Program that use an external library



myProgram calls a function in **the library libgeometry.a**

Program compilation example 2

```
### Compilation Procedure ###
# Compile the library first
cd myLibrary # Change directory
#  -J : This option specifies where to put .mod files.
gfortran -Jinclude  -c module_circle.f90 -o module_circle.o
gfortran -Jinclude  -c module_rectangle.f90 -o module_rectangle.o

# Now let's create my static library "libgeometry"
ar -rv lib/libgeometry.a module_circle.o module_rectangle.o

# Compile the myProgram.f90 program (no linking!)
cd .. # Return to the project dir
gfortran  -I./myLibrary/include -c myProgram.f90

# Link myProgram and the external module to create the executable
# -lName : Search the library named "libName.a" or "libName.so"
# -L: Add directories where libraries are search
gfortran myProgram.o -L./myLibrary/lib -lgeometry -o myProgram
```

Back to the compilation errors...

- Syntax
- Type mismatch
- Linking

Compilation errors: Syntax

- Error in the syntax of a sequence of characters or tokens that is intended to be written in a particular programming language.

Compilation errors: Syntax

syntaxError/syntaxError.f90

```
1  !-------------------------------------------------------------------------
2  !
3  !  Example of syntax errors
4  !
5  !-------------------------------------------------------------------------
6
7
8  program syntaxErrors
9
10     IMPLICIT NONE
11
12     INTEGER :: myInput
13
14     ! Ask for a input value
15     write(*, '(A)', ADVANCE = "NO") "Enter a integer value:  "
16     read(*,*) myInput  ! This variable name is not the correct one!
17
18     IF myInput < 0 THEN   ! "IF (myInput < 0 ) THEN" is the correct syntax!
19         write(*, *) "The input value negative"
20     ELSE
21         write(*, *) "The input value positive"
22     END IF
23
24 end program syntaxErrors
25
```

Logs & others

Debugger | Build log | Build messages

```
Checking if target is up-to-date: make -q -f Makefile all
Running command: make -f Makefile all
gfortran   -c syntaxError.f90
syntaxError.f90:18.4:
    IF myInput < 0 THEN   ! "IF (myInput < 0 ) THEN" is the correct syntax!
    1
Error: Unclassifiable statement at (1)
```

Compilation errors: Type mismatch

- Calling a function or procedure with arguments that are not the expected ones.

Compilation errors: Type mismatch

```
 8  PROGRAM typeMismatchError
 9      IMPLICIT NONE
10      REAL(8) :: rectagleArea
11      INTEGER :: side_a_length
12      INTEGER :: side_b_length
13      REAL :: myArea
14
15      side_a_length = 2
16      side_b_length = 4
17      myArea = rectagleArea( side_a_length , side_b_length)
18      PRINT *, myArea
19
20  END PROGRAM
21
22  REAL(8) FUNCTION rectagleArea( side_a_length , side_b_length )
23      IMPLICIT NONE
24      REAL(8), INTENT(IN) :: side_a_length
25      REAL(8), INTENT(IN) :: side_b_length
26
27      rectagleArea=side_a_length*side_b_length
28
29  END FUNCTION
30
```

Logs & others

Build log | Build messages

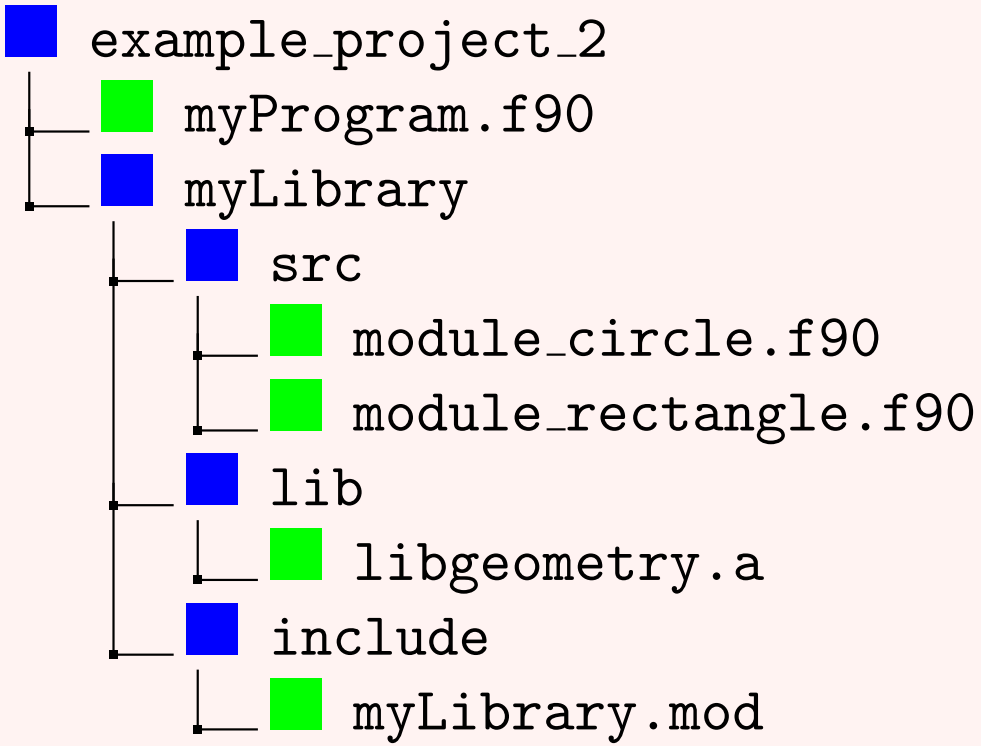
```
Running command: make -f Makefile
gfortran   -c typeMismatchError.f90
typeMismatchError.f90:17.27:
    myArea = rectagleArea( side_a_length , side_b_length)
                           1
Warning: Type mismatch in argument 'side a length' at (1); passed INTEGER(4) to REAL(8)
gfortran typeMismatchError.o -o typeMismatchError
Process terminated with status 0 (0 minute(s), 0 second(s))
0 error(s), 1 warning(s) (0 minute(s), 0 second(s))
```

Compilation errors: Linking

- The source code is compiled successfully, but the objects can be linked
- E.g. missing library or function

Program compilation example 2

Program that use an external library



myProgram calls a function in **the library libgeometry.a**

Example project: Compilation commands

```
# ERROR DURING LINKING
# Object that needs the library was placed after the libraries definitions.

# Link myProgram and the external module to create the executable
# -lName : Search the library named "libName.a" or "libName.so"
# -L: Add directories where libraries are search
$ gfortran -L./myLibrary/lib -lgeometry myProgram.o  -o myProgram

myProgram.o: In function `MAIN__':
myProgram.f90:(.text+0x4e): undefined reference to
  `__module_rectangle_MOD_rectagle_area'
collect2: error: ld returned 1 exit status
```

Why did this happen?

Lets review the linking process... [a]

- The linker maintains a symbol table with two lists:
  - A list of **symbols exported** by all the objects and libraries encountered so far.
  - A list of **undefined symbols** that the encountered objects and libraries requested to import and were not found yet.

[a] https://eli.thegreenplace.net/2013/07/09/library-order-in-static-linking

Lets review the linking process... [a]

- When the linker encounters a new object file, it looks at:
  - The **symbols it exports**: these are added to the list of exported symbols. If any symbol is in the undefined list, it's removed from there because it has now been found.
  - The **symbols it imports**: these are added to the list of undefined symbols, unless they can be found in the list of exported symbols.

[a] https://eli.thegreenplace.net/2013/07/09/library-order-in-static-linking

Lets review the linking process... [a]

- When the linker encounters a new library.
  - The linker goes over all the objects in the library.
  - Add **only** the objects that contain symbols that are on the undefined list.
- When the linker finishes
  - If any symbols remain in the undefined list, the linker throw an **"undefined reference" error**.
  - Note that after the linker has looked at a library, it won't look at it again.

[a] https://eli.thegreenplace.net/2013/07/09/library-order-in-static-linking

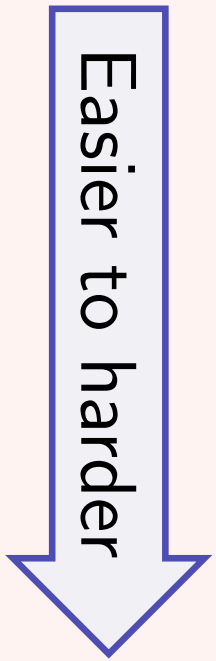
Back to the linking error...

```
# ERROR DURING LINKING
# Object that needs the library was placed after the libraries definitions.

# Link myProgram and the external module to create the executable
# -lName : Search the library named "libName.a" or "libName.so"
# -L: Add directories where libraries are search
$ gfortran -L./myLibrary/lib -lgeometry myProgram.o  -o myProgram

myProgram.o: In function `MAIN__':
myProgram.f90:(.text+0x4e): undefined reference to
  `__module_rectangle_MOD_rectagle_area'
collect2: error: ld returned 1 exit status
```

Why did this happen?

Runtime errors

- Compilation
- **Runtime**
- Logic



Common Runtime errors

- Segmentation fault
- Infinity loops
- Uninitialized variables
- Arithmetic operations errors

Segmentation fault

- Specific kind of error caused by accessing memory that "does not belong to you."
- It is a helper mechanism that keeps you from corrupting the memory and introducing hard-to-debug memory bugs.
- It can happen when:
  - Accessing variable that has already been freed
  - Writing to a read-only portion of the memory
  - Accessing an indexes of an arrays that is beyond its dimension

Segmentation fault

- Contrary to other languages, Fortran is not able to properly handle invalid memory access on runtime.
- If an index is out of scope, Fortran will not see it and a memory corruption might appear.

Segmentation fault example: Code

```
1  program segmentation_fault
2  implicit none
3  integer :: i
4
5  real,dimension(4) :: one_array
6  write(*,*) "one_array elements: From",loc(one_array),"to",loc(one_array)+4*9
7  write(*,*) "i location relative to one_array(0):", loc(i)-loc(one_array)
8  write(*,*)
9
10 write(*,*) '                              index          memory address'
11 do i = 1,10
12    write(*,*) 'Writing "10" for index',i, "   at",loc(one_array(i))-loc(one_array)
13    one_array(i)=10
14 enddo
15 end program
```

Segmentation fault example: Output

```
one_array elements: From      140735594477648 to      140735594477684
i location relative to one_array(0):                  28

                                  index          memory address
Writing "10" for index              1     at                0
Writing "10" for index              2     at                4
Writing "10" for index              3     at                8
Writing "10" for index              4     at               12 #one_array(4)
Writing "10" for index              5     at               16 #???
Writing "10" for index              6     at               20 #???
Writing "10" for index              7     at               24 #???
Writing "10" for index              8     at               28 # "i"
Writing "10" for index     1092616193     at       4370464768
Program received signal SIGSEGV: Segmentation faultinvalid memory
      reference.
```

Segmentation fault example: Memory layout

Memory Address	Element
140728745873376	one_array(1)
140728745873376 + 4	one_array(2)
140728745873376 + 8	one_array(3)
140728745873376 + 12	one_array(4)
140728745873376 + 16	???
140728745873376 + 20	???
140728745873376 + 24	???
140728745873376 + 28	"i"

- Accesing an element outside the array has an undefined behavior!
- Sometimes you don't get a Segmentation Fault and you an modify other variables

Segmentation fault example: Debugging

Easiest way to debug this:

- Add the **-fcheck=bounds** flag to the compilation process

```
$ gfortran -fcheck=bounds -o segmentation_fault segmentation_fault.f90
# Compiling and linking done in one line!
# The -c option is not used. No intermediate object is generated
# -fcheck=bounds: Enable generation of run-time checks for array subscripts
#    and against the declared minimum and maximum values.
#    This is intended for debugging since it will
#    slow down the program execution
```

`https://gcc.gnu.org/onlinedocs/gfortran/Code-Gen-Options.html`

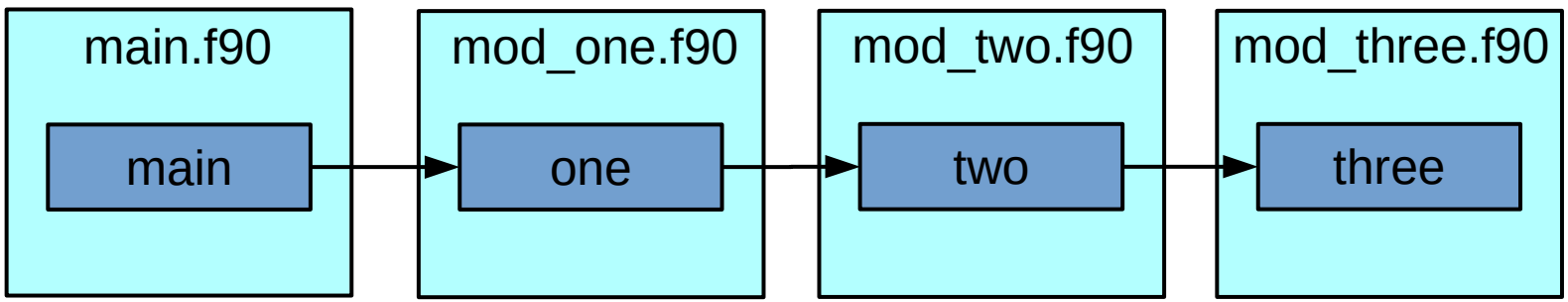
Segmentation fault example: Debug output

```
one_array elements: From       140731043230640 to       140731043230676
i location relative to one_array(0):                   28

                              index          memory address
Writing "10" for index            1   at                    0
Writing "10" for index            2   at                    4
Writing "10" for index            3   at                    8
Writing "10" for index            4   at                   12
At line 13 of file segmentation_fault.f90
Fortran runtime error: Index '5' of dimension 1 of array 'one_array'above upper
      bound of 4
```

Another segmentation fault example

Segmentation fault example

- segmentation_fault_stacktrace
  - main.f90
  - mod_one.f90
  - module_two.f90
  - module_three.f90



Another segmentation fault example (ifort)

In which routine the segmentation fault occurs?

```
$ ./main
forrtl: severe (174): SIGSEGV, segmentation fault occurred
Image              PC                Routine            Line        Source
main               0000000000469BF9  Unknown               Unknown  Unknown
main               0000000000404684CE  Unknown             Unknown  Unknown
main               00000000004379A2  Unknown               Unknown  Unknown
main               000000000041BC98  Unknown               Unknown  Unknown
main               00000000004029FB  Unknown               Unknown  Unknown
libpthread.so.0    00002B86249695E0  Unknown               Unknown  Unknown
main               00000000004026BC  Unknown               Unknown  Unknown
main               0000000000402729  Unknown               Unknown  Unknown
main               000000000040268E  Unknown               Unknown  Unknown
libc.so.6          00002B8624B97C05  Unknown               Unknown  Unknown
main               0000000000402599  Unknown               Unknown  Unknown
```

Another segmentation fault example (ifort)

In which routine the segmentation fault occurs?

```
# Adding "-g -tracback" flags to ifort compiler:
#
$ ./main
forrtl: severe (174): SIGSEGV, segmentation fault occurred
Image              PC                Routine            Line        Source
...
main               0000000000437A32  Unknown               Unknown  Unknown
main               000000000041BD28  Unknown               Unknown  Unknown
main               0000000000402A8B  Unknown               Unknown  Unknown
libpthread.so.0    00002B3F099255E0  Unknown               Unknown  Unknown
main               00000000004026FC  mod_three_mp_thre      10  mod_three.f90
main               00000000004027EB  mod_two_mp_two_         7  mod_two.f90
main               00000000004027FF  mod_one_mp_one_         7  mod_one.f90
main               00000000004027D2  MAIN__                 4      main.f90
....
```

Another segmentation fault example (ifort)

```
1  module mod_three
2      implicit none
3  contains
4      subroutine three()
5      implicit none
6          INTEGER :: J
7          REAL, ALLOCATABLE :: M (:)
8
9          DO J=1, 10
10             M(J)=4
11         END DO
12
13     end subroutine three
14 end module mod_three
```

Uninitialized variables errors

- An uninitialized variable is a variable that is declared but is its value was not set before it is used.
- It will have some value, but not a predictable one.
- This can lead to unexpected results and a program failure.

Uninitialized variable: Example

```
1  program compute_factorial
2    implicit none
3    integer :: number, i
4    write(*,*) 'Type a positive integer number'
5    read (*,*) number
6    write(*,*) factorial(number)
7
8    contains
9    integer function factorial(n)
10         integer, intent(in) :: n
11         integer :: i
12         if (n < 0) error stop 'Only positive integers supported'
13         do i = 2, n
14           factorial = factorial * i ! Factorial never initialized!!
15         enddo
16   end function factorial
17 end program compute_factorial
```

Uninitialized variable example: output

```
$ ./compute_factorial
 Type a positive integer number: 4
 factorial:              0#this is not right...
```

Uninitialized variable example: debug

```
$ gfortran -Wmaybe-uninitialized -Wuninitialized -O compute_factorial.f90
-o compute_factorial
# These warnings are possible only in optimized compilation!
# Hence, at least -O optimization is needed

compute_factorial.f90: In function compute_factorial:
compute_factorial.f90:6:0: warning: __result_factorial may be
 used uninitialized in this function [-Wmaybe-uninitialized]
   write(*,*) "factorial:", factorial(number)
 ^
compute_factorial.f90:9:0: note: __result_factorial was declared here
   integer function factorial(n)
 ^
```

Uninitialized variable example: debug

```
1  program compute_factorial
2    implicit none
3    integer :: number, i
4    write(*,*) 'Type a positive integer number'
5    read (*,*) number
6    write(*,*) factorial(number)
7
8    contains
9    integer function factorial(n)
10         integer, intent(in) :: n
11         integer :: i
12         factorial = 1 ! Initialize
13         if (n < 0) error stop 'Only positive integers supported'
14         do i = 2, n
15           factorial = factorial * i
16         enddo
17   end function factorial
18 end program compute_factorial
```

Uninitialized variable example: output

```
$ ./compute_factorial
 Type a positive integer number:
4
 factorial:          24 # Correct!
```

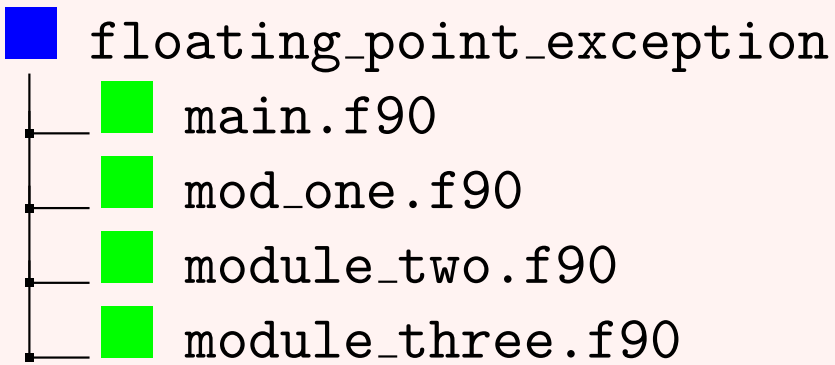
Floating points exceptions as a result of a calculation

- **Invalid**: Invalid operation. E.g. sqrt(-1).
- **Zero division**: Division by zero
- **Overflow**: The result is bigger in absolute value than maximum value that the computer can actually represent.
- **Underflow**: The result is smaller in absolute value than minimum value that the computer can actually represent.

Floating points exception: Example

Segmentation fault example

Takes 3 numbers (a,b, and c) and computes a/(b-c) in one of the subroutines.



Floating points exception: Example

```
$ ./main
 Type 3 numbers: 5 4 4
 a/(b−c) = Infinity
```

- Where did this exception happen?

Debugging

- Add the following compilation flags (gfortran):
- **-g** : Add debug information (important!)
- **-ffpe-trap=underflow,overflow,invalid,zero** : Trap arithmetic errors

Floating points exception: Example

```
$ ./main
 Type 3 numbers: 5 4 4

Program received signal SIGFPE: Floating-point exception
- erroneous arithmetic operation.

Backtrace for this error:
#0  0x2B86BFC1E6F7
#1  0x2B86BFC1ED3E
#2  0x2B86C06B026F
#3  0x4009AD in __mod_three_MOD_three at mod_three.f90:9 #Here!!
#4  0x400BC7 in __mod_two_MOD_two at mod_two.f90:9
#5  0x400BF9 in __mod_one_MOD_one at mod_one.f90:10
#6  0x400AD5 in MAIN__ at main.f90:7
Floating point exception (core dumped)
```

Floating points exception: Example

```
1  module mod_three
2      implicit none
3  contains
4      subroutine three(a,b,c,d)
5      implicit none
6          REAL,INTENT(IN) :: a,b,c
7          REAL,INTENT(OUT) :: d
8
9          d = a/(b-c)
10
11     end subroutine three
12 end module mod_three
```

More on Floating-point arithmetic [b]

- Due to the limited precision available to represent real numbers, things that are true for normal arithmetic no longer hold in floating-point arithmetic.

[b] http://jules-lsm.github.io/coding_standards/guidelines/fp_arithmetic.html

More on Floating-point arithmetic [c]

- Comparing 2 float numbers

```
1  IF ( precipitation(x,y) > 0.0 ) THEN
2    ! Do something
3  ELSE
4    ! Do something else
5  END IF
```

- Due to rounding errors or optimizations, in places were we expect to have zero precipitation a very low value can be encountered.

[c] http://jules-lsm.github.io/coding_standards/guidelines/fp_arithmetic.html

More on Floating-point arithmetic [d]

- Due to rounding errors or optimizations, in places were we expect to have zero precipitation a very low value can be encountered.
- **Solution**: specify a physically realistic tolerance level.

```
1  IF ( precipitation(x,y) > tolerance ) THEN
2    ! Do something
3  ELSE
4    ! Do something else
5  END IF
```

[d] http://jules-lsm.github.io/coding_standards/guidelines/fp_arithmetic.html

More on Floating-point arithmetic [e]

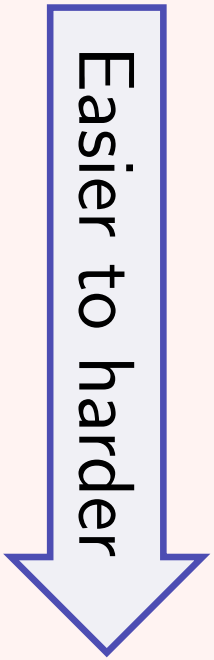
- More about rounding errors

```
1  IF ( a_real_number == other_real_number ) THEN
2    ! Do something
3  ELSE
4    ! Do something else
5  END IF
6
7  !!! Better approach
8  IF ( ABS(a_real_number  other_real_number) < tolerance ) THEN
9    ! Do something
10 ELSE
11   ! Do something else
12 END IF
```

[e] http://jules-lsm.github.io/coding_standards/guidelines/fp_arithmetic.html

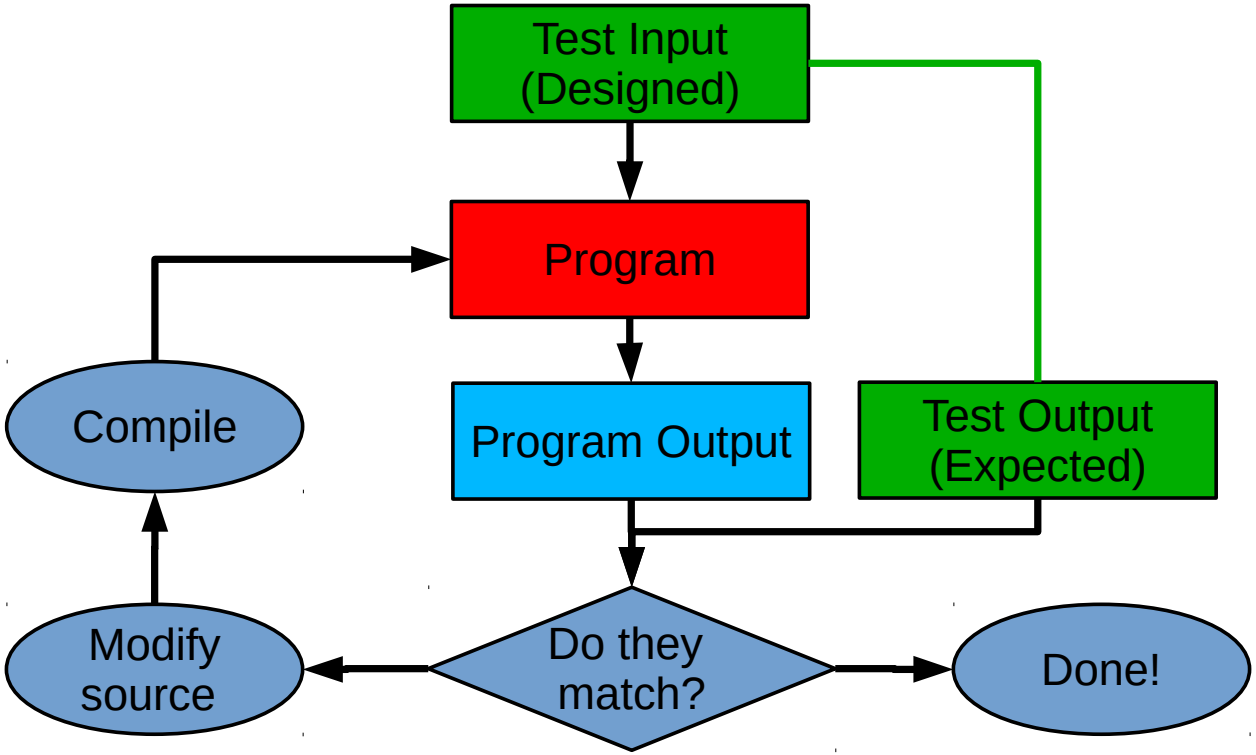
Logical errors

- Compilation
- Runtime
- **Logic**



Logical errors

- The program appears to run correctly, by it gives wrong or unexpected results.
- One approach to solve them is to test different parts of the program with inputs with known outputs.



Final remarks

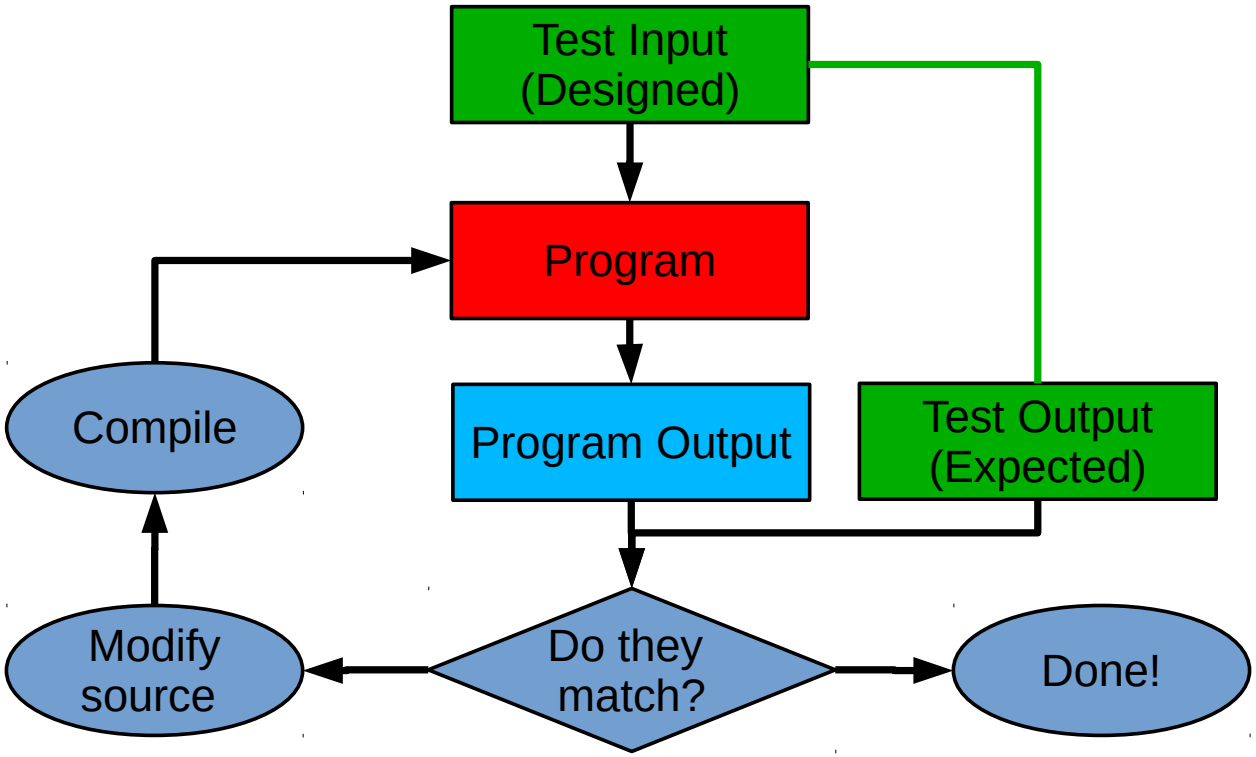
- First fix all the compilation errors and warnings
  - Compile and run the program with the following flags (gfortran):
  - **-fcheck=bounds**
  - **-Wmaybe-uninitialized -Wuninitialized -O**

Final remarks

- If an error is detected on runtime:
  - Recompile and run the program with the following flags (gfortran):
  - **-fcheck=bounds** (check arrays bounds)
  - **-O0** (disable optimizations!)
  - **-g -fbacktrace** (add debugging information!)
  - **-ffpe-trap=underflow,overflow,invalid,zero** (trap floating point exceptions)

Final remarks

- Run test cases to find logical errors



About this presentation

- This presentation was released into the public domain (CC0)
- A copy of this presentation and the error examples can be found in:
  `https://github.com/aperezhortal/fortran_debugging_introduction`

Thanks!