BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF
ENERGY
Office of Science

Fortran 2023 for you: Features and tools

Katherine Rasmussen, Damian Rouson

NHR PerfLab Seminar
March 10, 2025

Table of Contents

Features

Fortran 2023/ Modern Fortran

- Subset of features in the Fortran language
- New features from Fortran standards 2003, 2008, 2018, 2023, ~2028
- Features often considered Modern Fortran include
    - Loop level parallelism: `do-concurrent`
    - Multi-image parallelism: coarrays and related routines (`this_image`, `ucobound`, etc)
    - Collective and atomic subroutines (`co_sum`, `atomic_fetch_add`, etc)
    - Teams, locks, and events (`form-team-stmt`, `lock-stmt`, `event-post-stmt`, etc)
    - Polymorphism
    - C-interoperability

Fortran - Natively parallel language (since F2008)

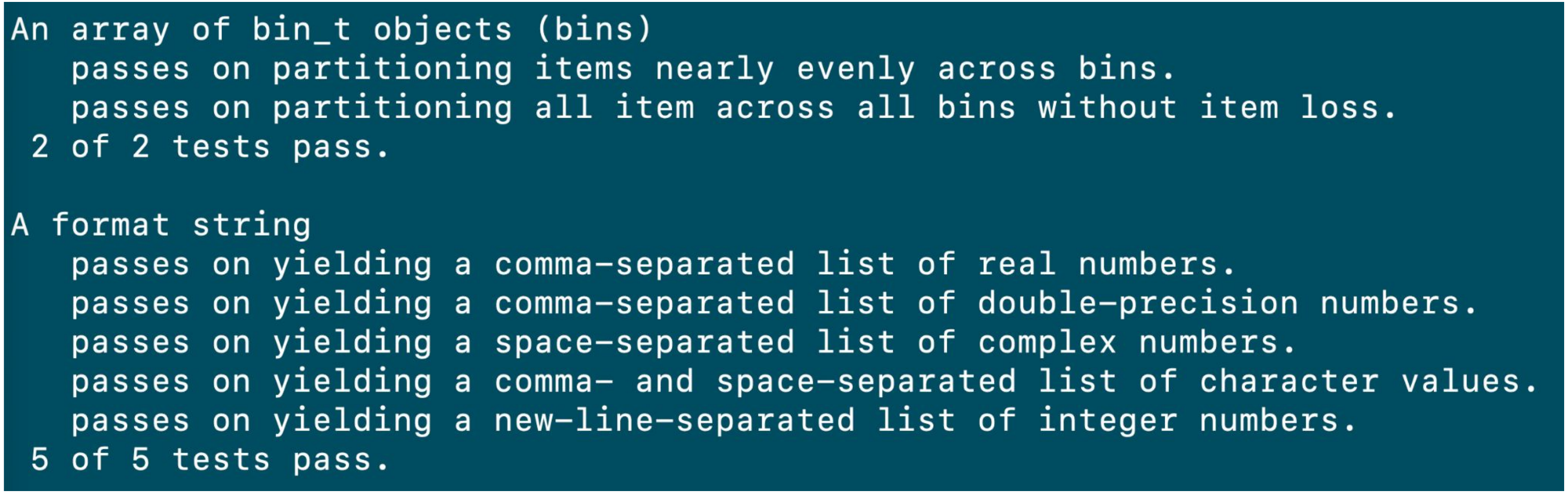
- Coarrays and more (collectives, atomics, etc)
  - SPMD (single program multiple data) parallelism
  - Uses a PGAS (partitioned global address space) shared memory abstraction
  - PRIF and Caffeine - runtime library
    - targeting LLVM Flang and LFortran
- `do-concurrent`
  - Loop-level parallelism
  - Currently 3 compilers (NVIDIA, Intel, HPE) support automatic offloading to GPUs
- Benefits include easier to write and potentially faster to run

Useful Intrinsic Functions/Features

- `findloc()` - Find location of a specified value in an array (optional `mask` arg)
- `pack()` - Pack an array into an array of rank one after `mask` is applied
- `count()` - Count true values in an array
- `merge()` - Merge variables based on the logic of the `mask` arg
- `index()` - Find position of a substring within a string
- Note: `mask` argument (boolean logic) of intrinsic functions
- Note: Combining intrinsic function calls
- More intrinsics with explanations here

BERKELEY LAB
Bringing Science Solutions to the World

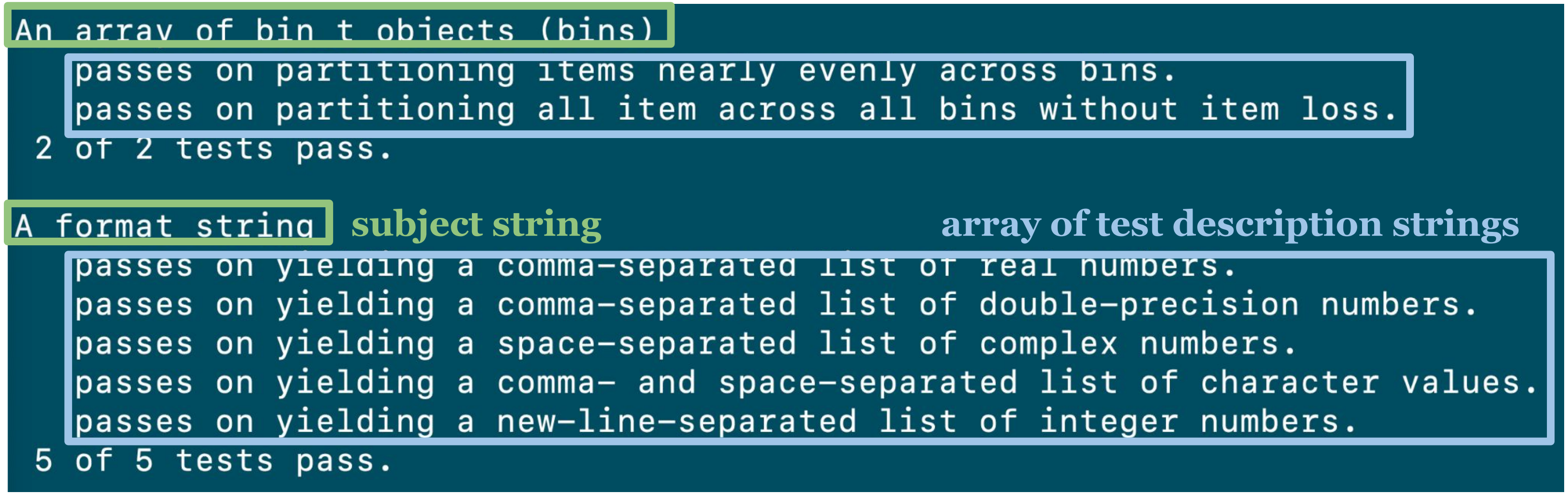
Useful Intrinsic Functions - Example - Use Case



```
An array of bin_t objects (bins)
   passes on partitioning items nearly evenly across bins.
   passes on partitioning all item across all bins without item loss.
 2 of 2 tests pass.

A format string
   passes on yielding a comma-separated list of real numbers.
   passes on yielding a comma-separated list of double-precision numbers.
   passes on yielding a space-separated list of complex numbers.
   passes on yielding a comma- and space-separated list of character values.
   passes on yielding a new-line-separated list of integer numbers.
 5 of 5 tests pass.
```

- Example of Julienne tests (more later)
- Feature: when running the tests, can pass a flag with a string that selects which tests are run: `-- --contains <string>`

Useful Intrinsic Functions - Example - Use Case

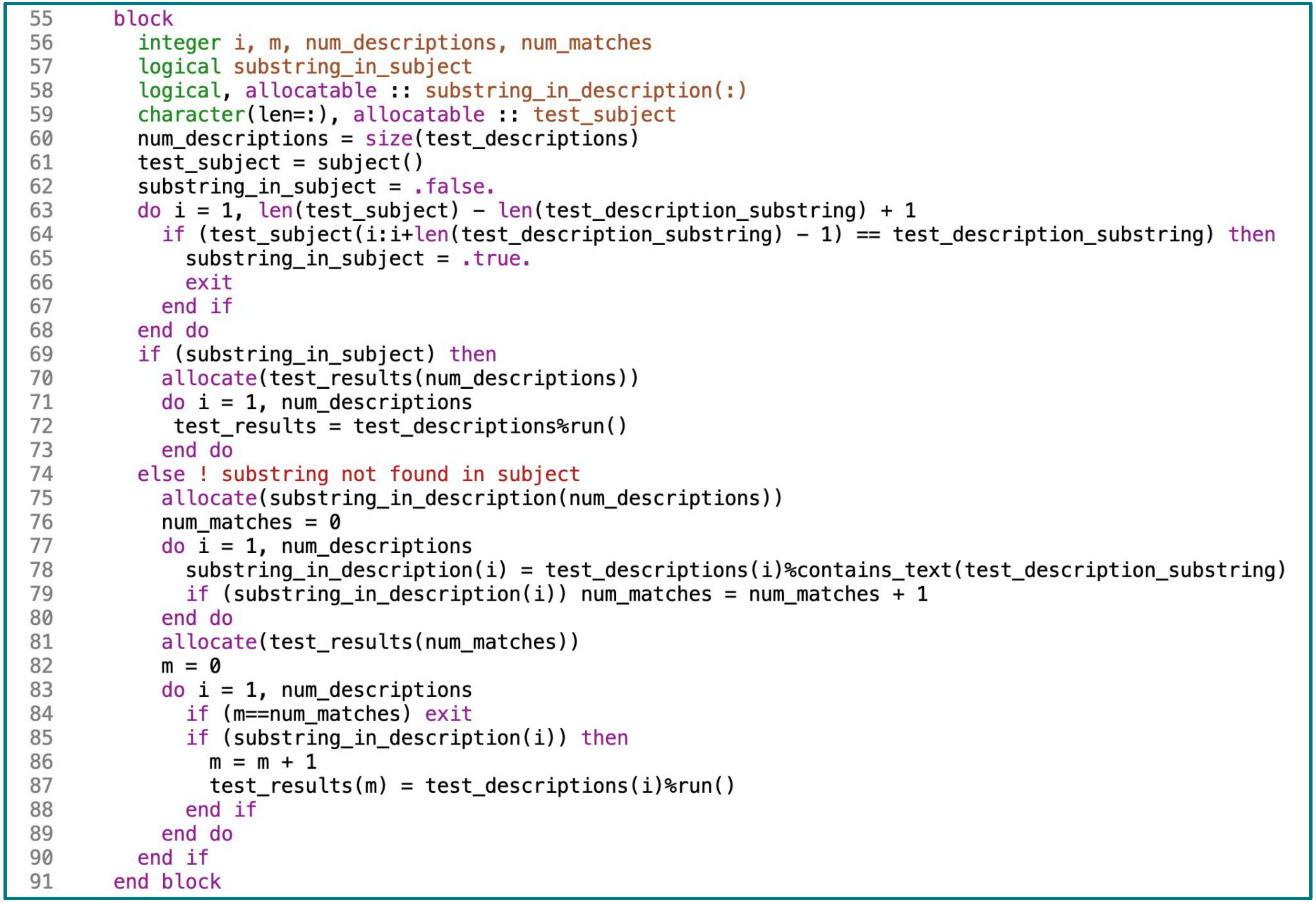


```
An array of bin t objects (bins)
   passes on partitioning items nearly evenly across bins.
   passes on partitioning all item across all bins without item loss.
 2 of 2 tests pass.

A format string
   passes on yielding a comma-separated list of real numbers.
   passes on yielding a comma-separated list of double-precision numbers.
   passes on yielding a space-separated list of complex numbers.
   passes on yielding a comma- and space-separated list of character values.
   passes on yielding a new-line-separated list of integer numbers.
 5 of 5 tests pass.
```

- Need to search the subject string for a substring
- If found, run all of the tests
- If not, search the array of test description strings for the substring and run only the tests where it appears

Starting version

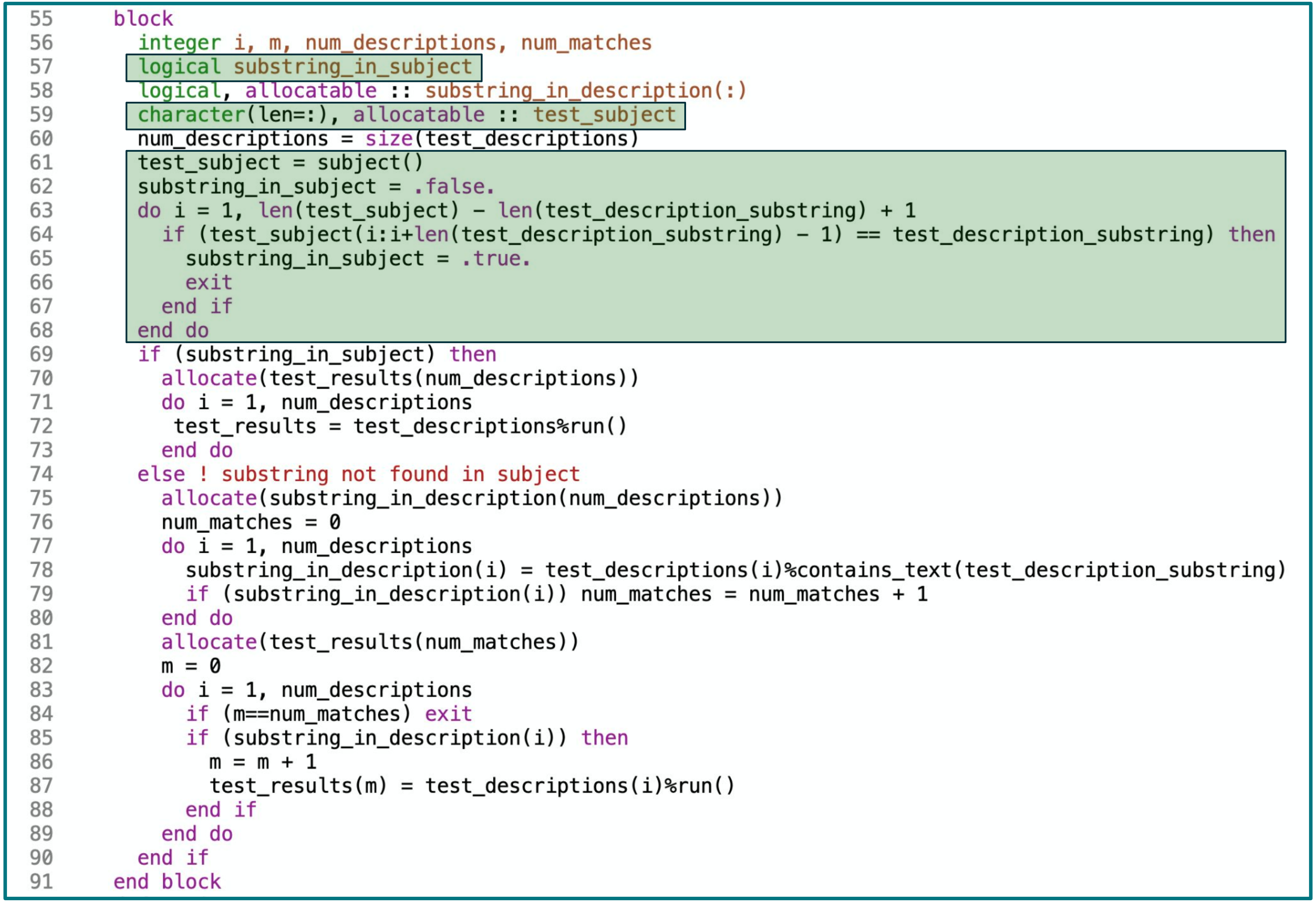
37 lines



```
    block
      integer i, m, num_descriptions, num_matches
      logical substring_in_subject
      logical, allocatable :: substring_in_description(:)
      character(len=:), allocatable :: test_subject
      num_descriptions = size(test_descriptions)
      test_subject = subject()
      substring_in_subject = .false.
      do i = 1, len(test_subject) - len(test_description_substring) + 1
        if (test_subject(i:i+len(test_description_substring) - 1) == test_description_substring) then
          substring_in_subject = .true.
          exit
        end if
      end do
      if (substring_in_subject) then
        allocate(test_results(num_descriptions))
        do i = 1, num_descriptions
         test_results = test_descriptions%run()
        end do
      else ! substring not found in subject
        allocate(substring_in_description(num_descriptions))
        num_matches = 0
        do i = 1, num_descriptions
          substring_in_description(i) = test_descriptions(i)%contains_text(test_description_substring)
          if (substring_in_description(i)) num_matches = num_matches + 1
        end do
        allocate(test_results(num_matches))
        m = 0
        do i = 1, num_descriptions
          if (m==num_matches) exit
          if (substring_in_description(i)) then
            m = m + 1
            test_results(m) = test_descriptions(i)%run()
          end if
        end do
      end if
    end block
```

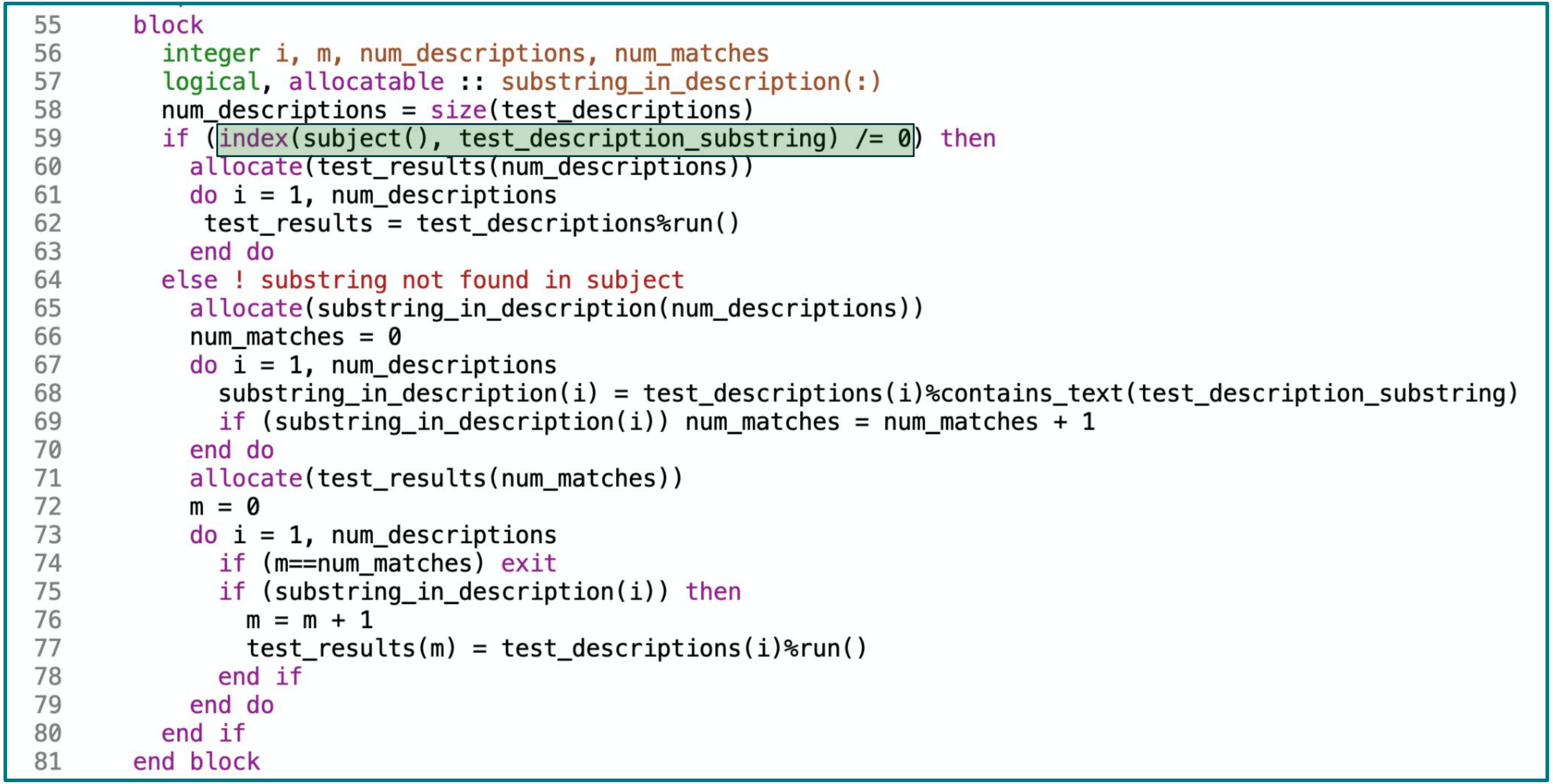
Starting version

37 lines



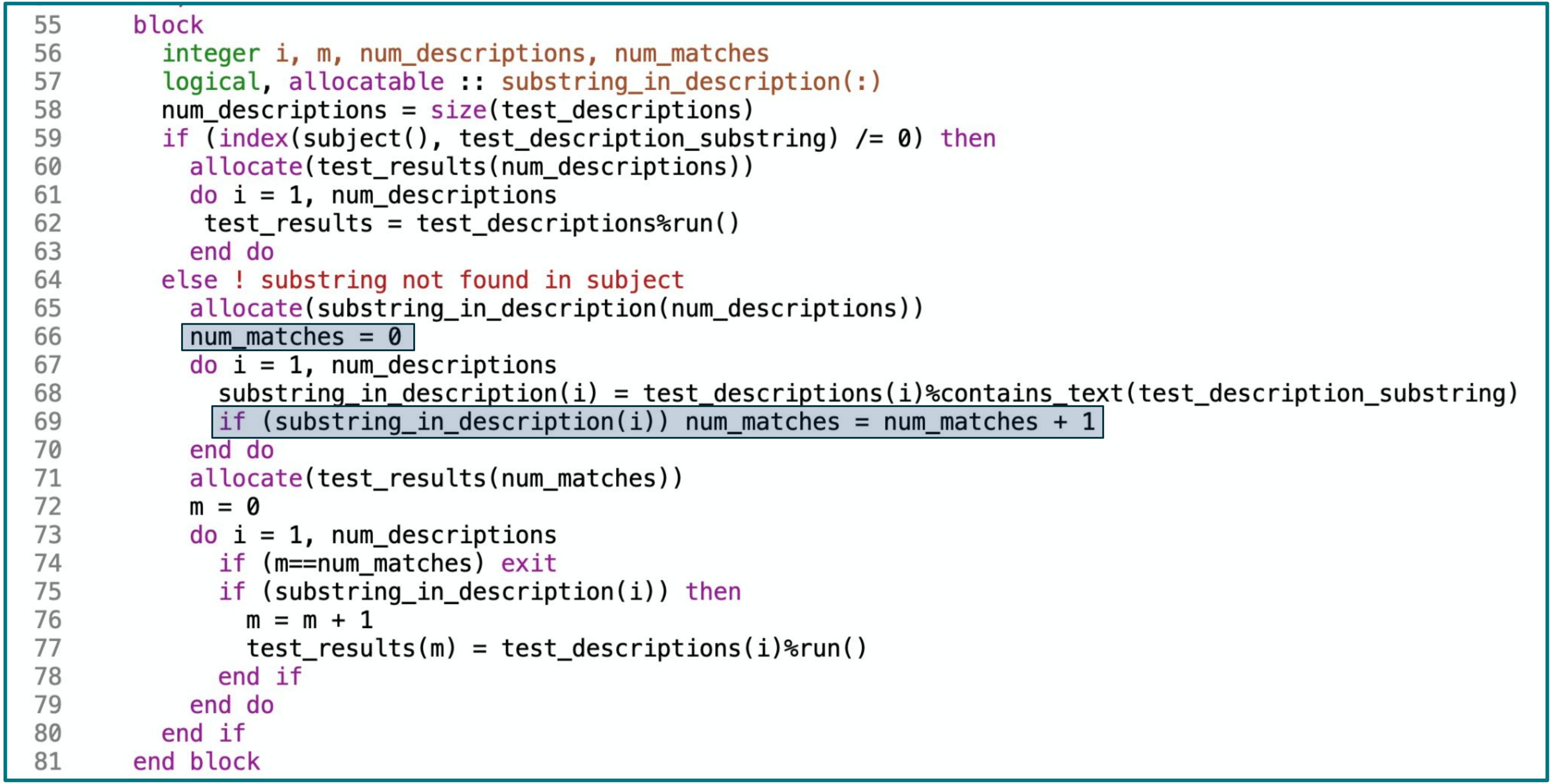
```fortran
    block
      integer i, m, num_descriptions, num_matches
      logical substring_in_subject
      logical, allocatable :: substring_in_description(:)
      character(len=:), allocatable :: test_subject
      num_descriptions = size(test_descriptions)
      test_subject = subject()
      substring_in_subject = .false.
      do i = 1, len(test_subject) - len(test_description_substring) + 1
        if (test_subject(i:i+len(test_description_substring) - 1) == test_description_substring) then
          substring_in_subject = .true.
          exit
        end if
      end do
      if (substring_in_subject) then
        allocate(test_results(num_descriptions))
        do i = 1, num_descriptions
         test_results = test_descriptions%run()
        end do
      else ! substring not found in subject
        allocate(substring_in_description(num_descriptions))
        num_matches = 0
        do i = 1, num_descriptions
          substring_in_description(i) = test_descriptions(i)%contains_text(test_description_substring)
          if (substring_in_description(i)) num_matches = num_matches + 1
        end do
        allocate(test_results(num_matches))
        m = 0
        do i = 1, num_descriptions
          if (m==num_matches) exit
          if (substring_in_description(i)) then
            m = m + 1
            test_results(m) = test_descriptions(i)%run()
          end if
        end do
      end if
    end block
```

Add `index`() - 27 lines



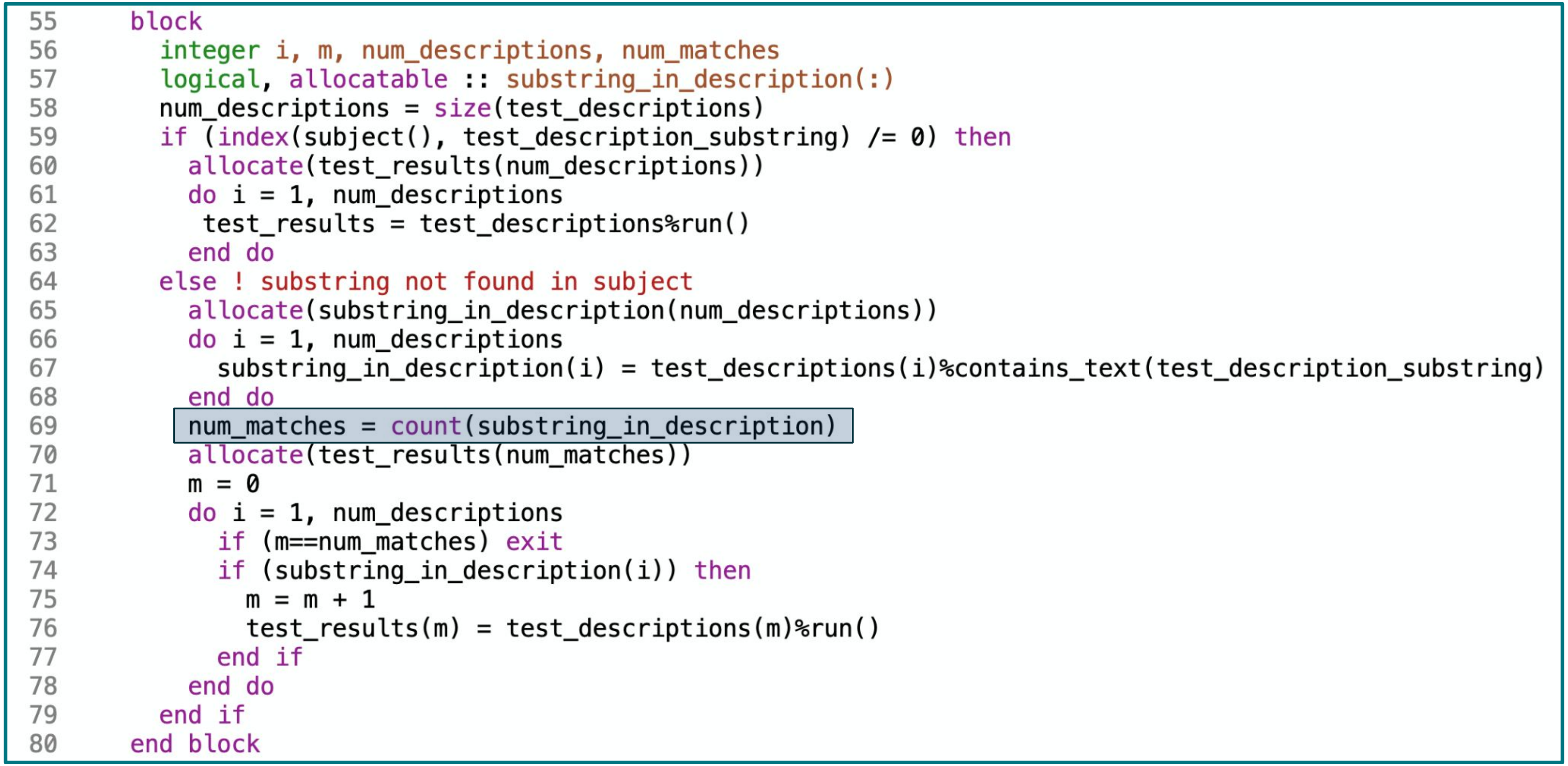
```
    block
      integer i, m, num_descriptions, num_matches
      logical, allocatable :: substring_in_description(:)
      num_descriptions = size(test_descriptions)
      if (index(subject(), test_description_substring) /= 0) then
        allocate(test_results(num_descriptions))
        do i = 1, num_descriptions
         test_results = test_descriptions%run()
        end do
      else ! substring not found in subject
        allocate(substring_in_description(num_descriptions))
        num_matches = 0
        do i = 1, num_descriptions
          substring_in_description(i) = test_descriptions(i)%contains_text(test_description_substring)
          if (substring_in_description(i)) num_matches = num_matches + 1
        end do
        allocate(test_results(num_matches))
        m = 0
        do i = 1, num_descriptions
          if (m==num_matches) exit
          if (substring_in_description(i)) then
            m = m + 1
            test_results(m) = test_descriptions(i)%run()
          end if
        end do
      end if
    end block
```

27 lines version



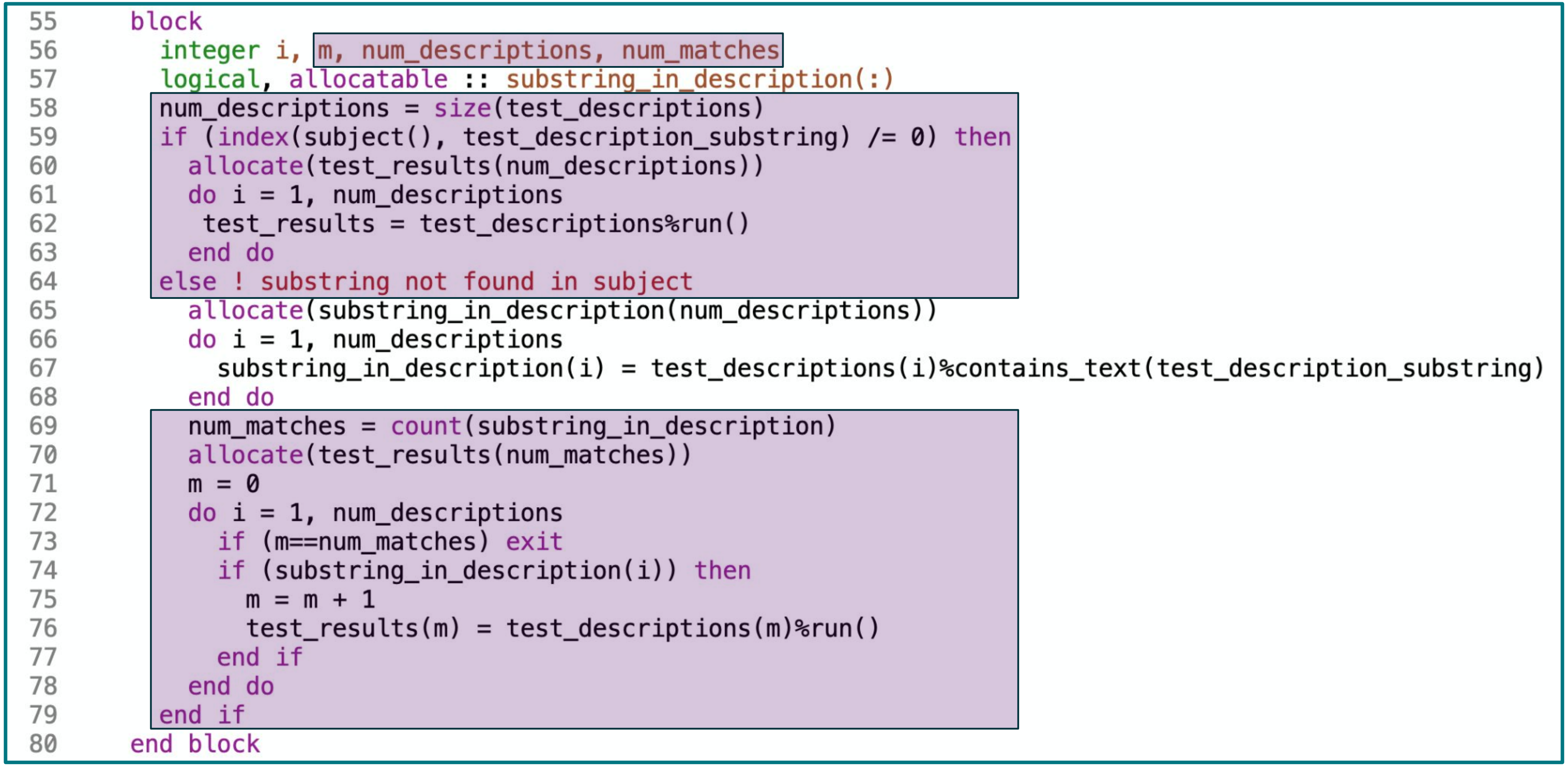
```
    block
      integer i, m, num_descriptions, num_matches
      logical, allocatable :: substring_in_description(:)
      num_descriptions = size(test_descriptions)
      if (index(subject(), test_description_substring) /= 0) then
        allocate(test_results(num_descriptions))
        do i = 1, num_descriptions
         test_results = test_descriptions%run()
        end do
      else ! substring not found in subject
        allocate(substring_in_description(num_descriptions))
        num_matches = 0
        do i = 1, num_descriptions
          substring_in_description(i) = test_descriptions(i)%contains_text(test_description_substring)
          if (substring_in_description(i)) num_matches = num_matches + 1
        end do
        allocate(test_results(num_matches))
        m = 0
        do i = 1, num_descriptions
          if (m==num_matches) exit
          if (substring_in_description(i)) then
            m = m + 1
            test_results(m) = test_descriptions(i)%run()
          end if
        end do
      end if
    end block
```

Add `count()` - 26 lines



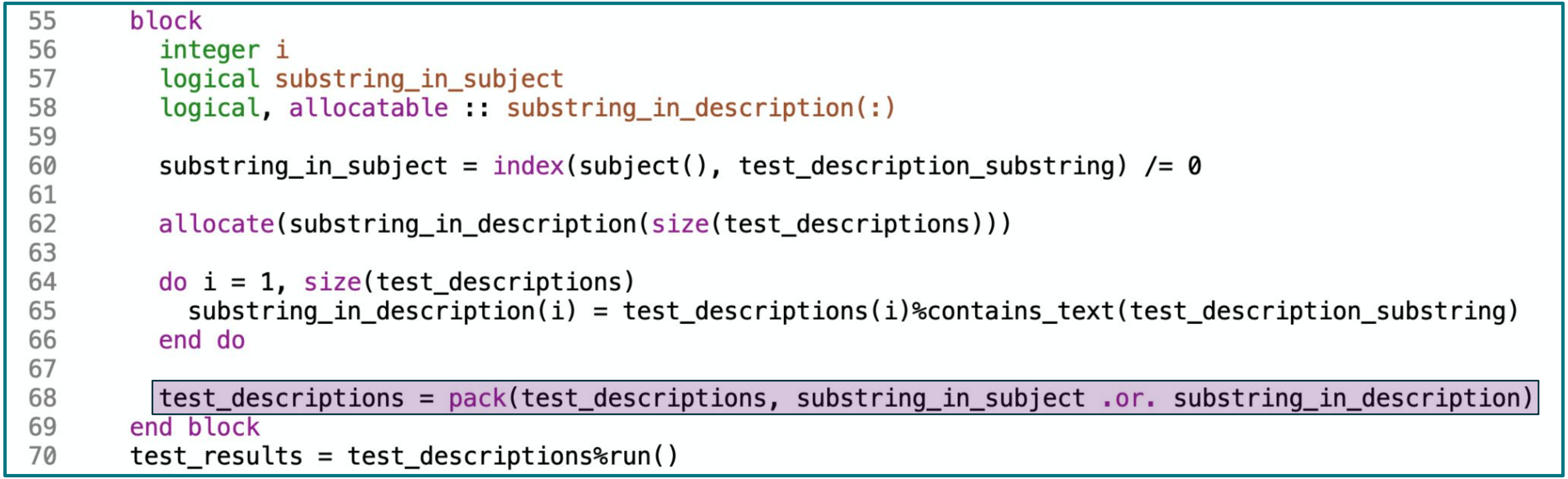
```
block
  integer i, m, num_descriptions, num_matches
  logical, allocatable :: substring_in_description(:)
  num_descriptions = size(test_descriptions)
  if (index(subject(), test_description_substring) /= 0) then
    allocate(test_results(num_descriptions))
    do i = 1, num_descriptions
     test_results = test_descriptions%run()
    end do
  else ! substring not found in subject
    allocate(substring_in_description(num_descriptions))
    do i = 1, num_descriptions
      substring_in_description(i) = test_descriptions(i)%contains_text(test_description_substring)
    end do
    num_matches = count(substring_in_description)
    allocate(test_results(num_matches))
    m = 0
    do i = 1, num_descriptions
      if (m==num_matches) exit
      if (substring_in_description(i)) then
        m = m + 1
        test_results(m) = test_descriptions(m)%run()
      end if
    end do
  end if
end block
```

26 lines version



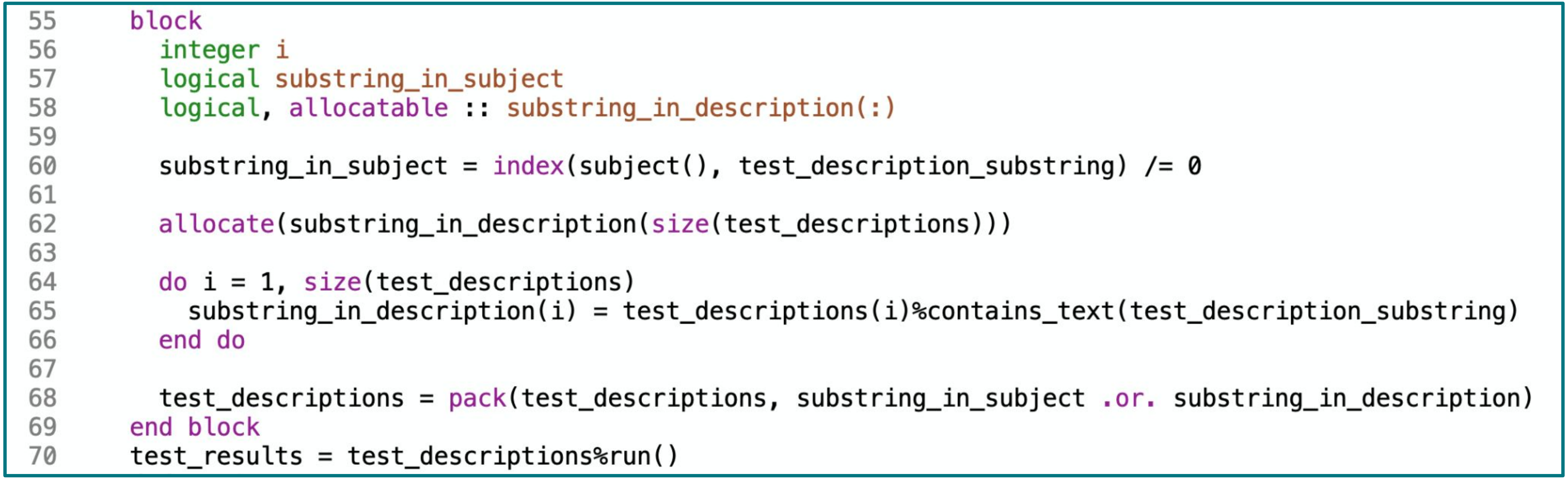
```
block
  integer i, m, num_descriptions, num_matches
  logical, allocatable :: substring_in_description(:)
  num_descriptions = size(test_descriptions)
  if (index(subject(), test_description_substring) /= 0) then
    allocate(test_results(num_descriptions))
    do i = 1, num_descriptions
     test_results = test_descriptions%run()
    end do
  else ! substring not found in subject
    allocate(substring_in_description(num_descriptions))
    do i = 1, num_descriptions
      substring_in_description(i) = test_descriptions(i)%contains_text(test_description_substring)
    end do
    num_matches = count(substring_in_description)
    allocate(test_results(num_matches))
    m = 0
    do i = 1, num_descriptions
      if (m==num_matches) exit
      if (substring_in_description(i)) then
        m = m + 1
        test_results(m) = test_descriptions(m)%run()
      end if
    end do
  end if
end block
```

Add **pack()** - 12 lines (when whitespace removed)



```fortran
    block
      integer i
      logical substring_in_subject
      logical, allocatable :: substring_in_description(:)

      substring_in_subject = index(subject(), test_description_substring) /= 0

      allocate(substring_in_description(size(test_descriptions)))

      do i = 1, size(test_descriptions)
        substring_in_description(i) = test_descriptions(i)%contains_text(test_description_substring)
      end do

      test_descriptions = pack(test_descriptions, substring_in_subject .or. substring_in_description)
    end block
    test_results = test_descriptions%run()
```

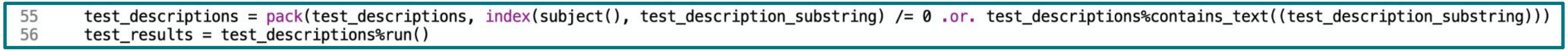
12 lines version (when whitespace removed)



```
    block
      integer i
      logical substring_in_subject
      logical, allocatable :: substring_in_description(:)

      substring_in_subject = index(subject(), test_description_substring) /= 0

      allocate(substring_in_description(size(test_descriptions)))

      do i = 1, size(test_descriptions)
        substring_in_description(i) = test_descriptions(i)%contains_text(test_description_substring)
      end do

      test_descriptions = pack(test_descriptions, substring_in_subject .or. substring_in_description)
    end block
    test_results = test_descriptions%run()
```

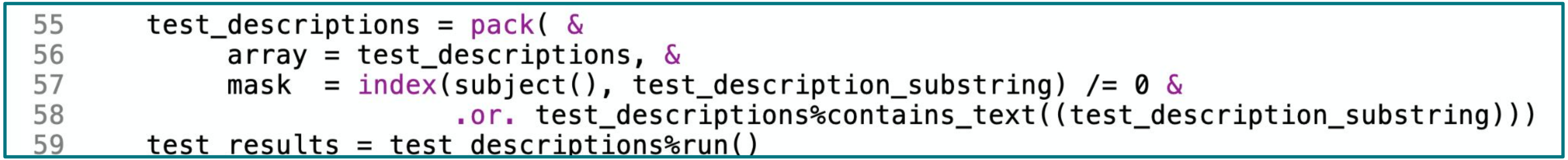
Combine and take advantage of automatic reallocation - ? lines

Combine and take advantage of automatic reallocation - 2 lines



```
    test_descriptions = pack(test_descriptions, index(subject(), test_description_substring) /= 0 .or. test_descriptions%contains_text((test_description_substring)))
    test_results = test_descriptions%run()
```

Combine and take advantage of automatic reallocation



```
test_descriptions = pack( &
     array = test_descriptions, &
     mask  = index(subject(), test_description_substring) /= 0 &
                   .or. test_descriptions%contains_text((test_description_substring)))
test_results = test_descriptions%run()
```

Useful Fortran Features - Example Review

- Fortran 95 features utilized in example:
  - `pack` intrinsic function
  - `index` intrinsic function
  - `count` intrinsic function (in intermediate step)
  - Automatic (re)allocation of arrays (F90-F2003)
  - `elemental` type-bound procedures - `run()`
    - `pure` procedure
- Code from julienne/test/bin_test.F90

Tools

fpm

- fpm (Fortran Package Manager)
  - Package manager and build system
  - Written in Fortran
  - Maintained by Fortran developer community
  - Automatically detects file dependencies
  - Supports both Fortran and C source code
  - fpm itself very easy to install
    - Available through package managers (Homebrew, etc)
    - If using gfortran13, can also compile a one file version of the `fpm` source code to install it

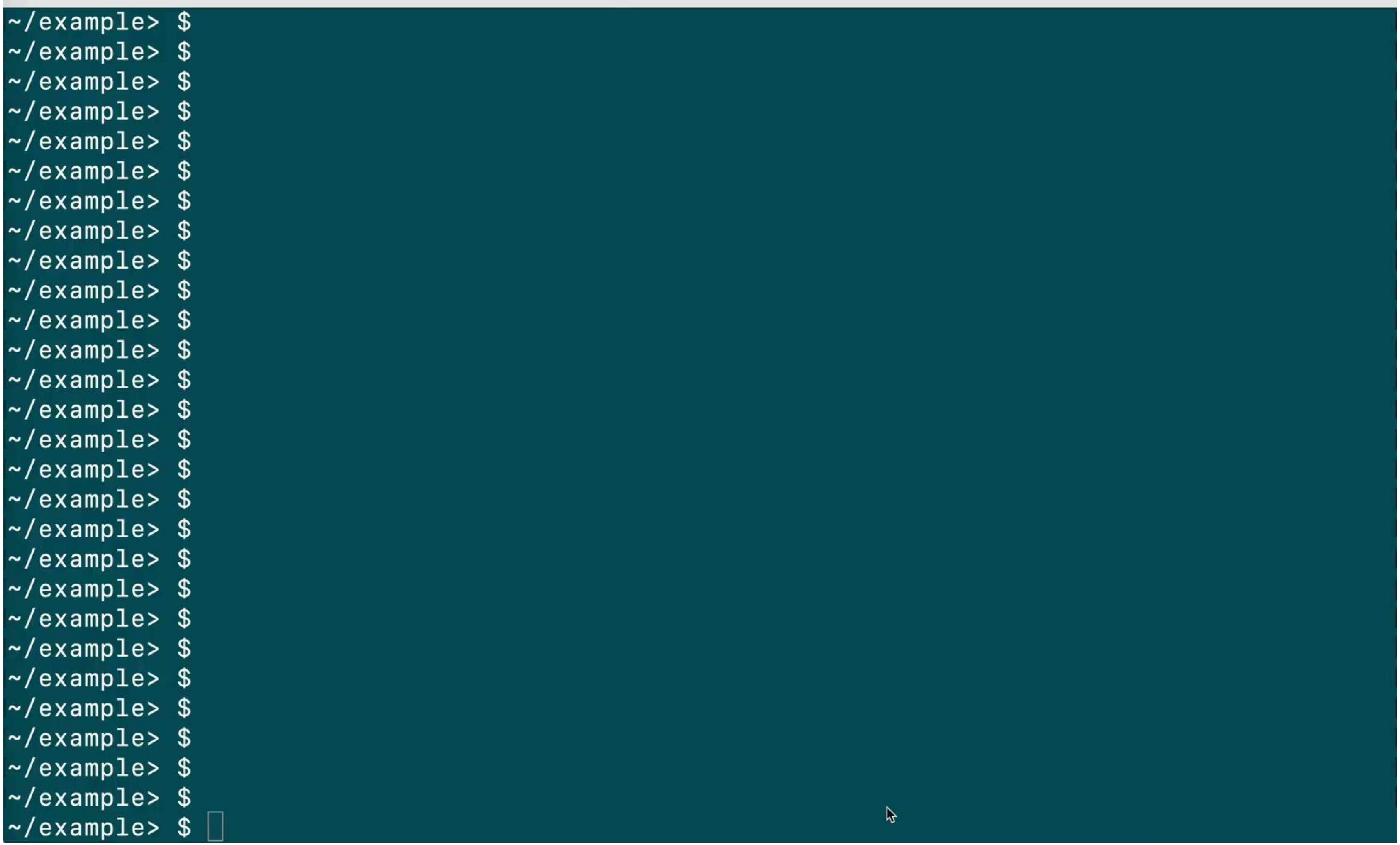
Unit testing

- Test driven development
- Benefits of unit tests
- Julienne
  - Lightweight unit testing framework
  - Test output can read as a specification
  - Provides diagnostic information for failures
- Assert
  - Utility that helps verify constraints
  - Useful diagnostic output from pure procedures

Julienne Example (content of video walkthrough)

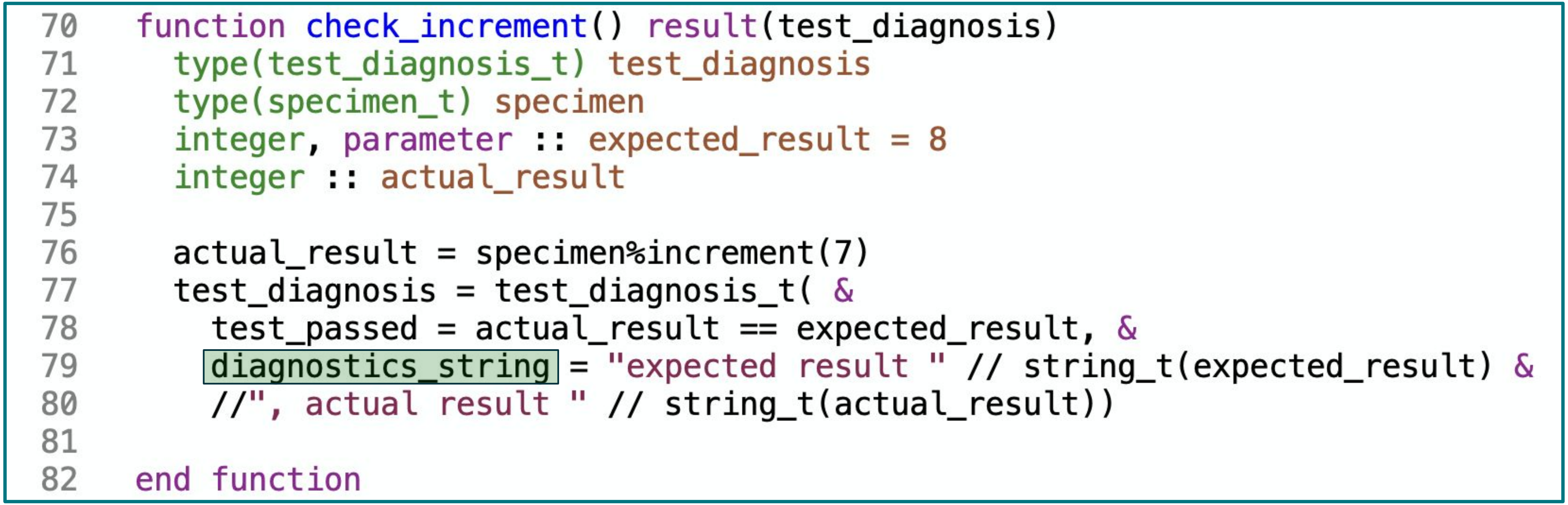
- Create new fpm project with `fpm new name_of_proj`
- Add Julienne dependency to `fpm.toml` file
- Copy Julienne example from `julienne/example/example-project` into new fpm project
- Move `specimen_m.f90` into `src` dir
- Run tests with `fpm test`
- Fix purposeful error in source code so test passes

BERKELEY LAB
Bringing Science Solutions to the World



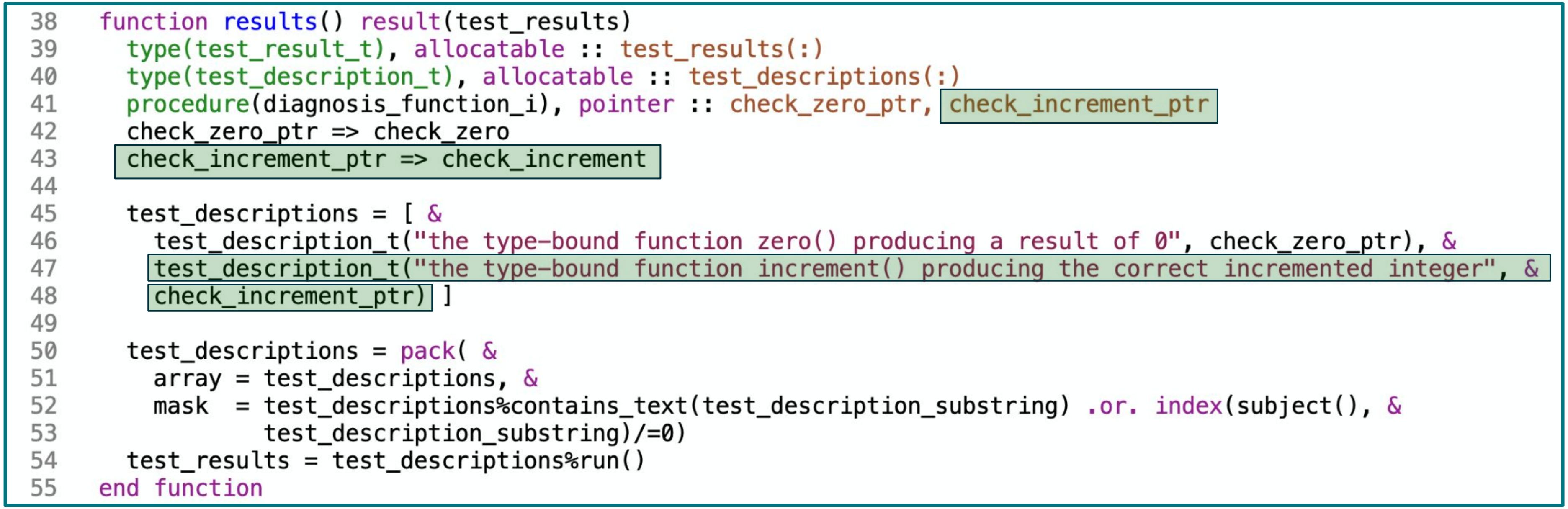
```
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
~/example> $
```

Add a test for a new function `increment()`



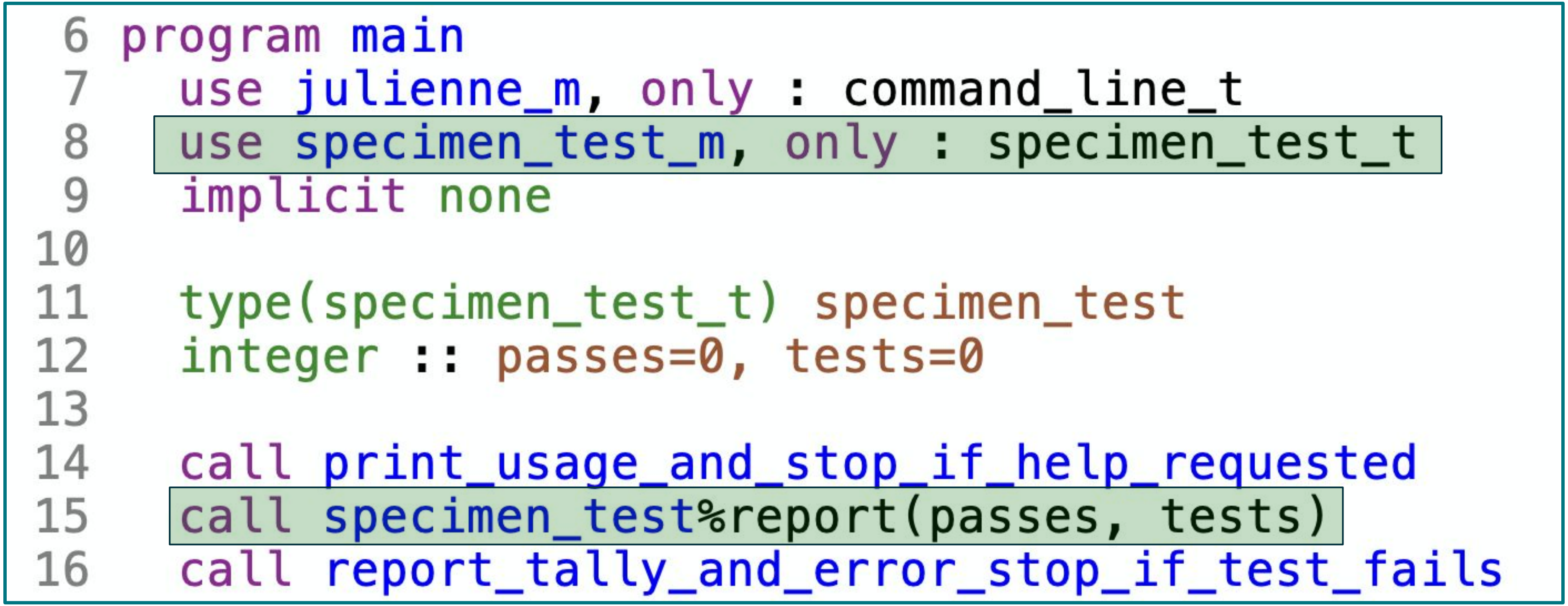
```fortran
function check_increment() result(test_diagnosis)
  type(test_diagnosis_t) test_diagnosis
  type(specimen_t) specimen
  integer, parameter :: expected_result = 8
  integer :: actual_result

  actual_result = specimen%increment(7)
  test_diagnosis = test_diagnosis_t( &
    test_passed = actual_result == expected_result, &
    diagnostics_string = "expected result " // string_t(expected_result) &
    //", actual result " // string_t(actual_result))

end function
```

Update `results()` function to call new test



```
function results() result(test_results)
  type(test_result_t), allocatable :: test_results(:)
  type(test_description_t), allocatable :: test_descriptions(:)
  procedure(diagnosis_function_i), pointer :: check_zero_ptr, check_increment_ptr
  check_zero_ptr => check_zero
  check_increment_ptr => check_increment

  test_descriptions = [ &
    test_description_t("the type-bound function zero() producing a result of 0", check_zero_ptr), &
    test_description_t("the type-bound function increment() producing the correct incremented integer", &
    check_increment_ptr) ]

  test_descriptions = pack( &
    array = test_descriptions, &
    mask  = test_descriptions%contains_text(test_description_substring) .or. index(subject(), &
            test_description_substring)/=0)
  test_results = test_descriptions%run()
end function
```

Julienne `main` program – call each collection of tests



```fortran
program main
  use julienne_m, only : command_line_t
  use specimen_test_m, only : specimen_test_t
  implicit none

  type(specimen_test_t) specimen_test
  integer :: passes=0, tests=0

  call print_usage_and_stop_if_help_requested
  call specimen_test%report(passes, tests)
  call report_tally_and_error_stop_if_test_fails
```

Deep Learning with Fortran

- Machine learning and AI impacts on HPC
- Fiats (Functional inference and training for surrogates)
  - Alternative name: Fortran inference and training for science
  - “training and deployment of neural-network surrogate models for computational science”
  - Automatic parallelization of batch inference

Grabbag of Fortran tools

- Codee - Static analysis tool for Fortran and C/C++
    - Code correctness
    - Modernization
    - Codee Youtube Channel
    - Codee training at NERSC
- fortran-linter
- rojff - Return of JSON for Fortran - Utility to support use of JSON files in Fortran source
- iso_varying_string - An implementation of the ISO_VARYING_STRING module as proposed for the ISO standard

Grabbag of Fortran tools

- FORD - (FORtran Documentation)
  - automatic documentation generator for Fortran projects
  - Can build documentation locally, provides html files
  - Can deploy documentation in Github Actions
  - Example Ford Documentation
- Various tools to convert fixed form code to free form code
- A grabbag of more tools - Beliavsky/Fortran-Tools

Communities & Where to Find More

Fortran Standards Committee

- International body - WG5
- Working group - INCITS US National body (informally known as J3)
- Upcoming features:
    - Generic programming (templates, etc)
        - Type-safe templates: requirements (strong concepts)
            - Must state properties of types
            - Compiler can provide error messages in template source code
    - Asynchrony: tasks, collectives
    - Standardized Fortran preprocessor

Fortran Resources & Communities

- Stereotypes about Fortran include ideas of being antiquated
- Vibrant, engaged developer community
- For more Fortran questions or to engage and **share** with the Fortran community, visit:
  - Fortran Lang: fortran-lang.org
    - Tutorials, links to playgrounds, compilers info
    - Helpful language information and advice
    - Link to LFortran, a Fortran compiler and interpreter
  - Discourse: fortran-lang.discourse.group
  - Fortran Wiki: fortranwiki.org
- Fortran at LBNL

Thank You

Questions?

Email: fortran@lbl.gov