Source routines

An introduction to a new approach to source routines

Why user routines?

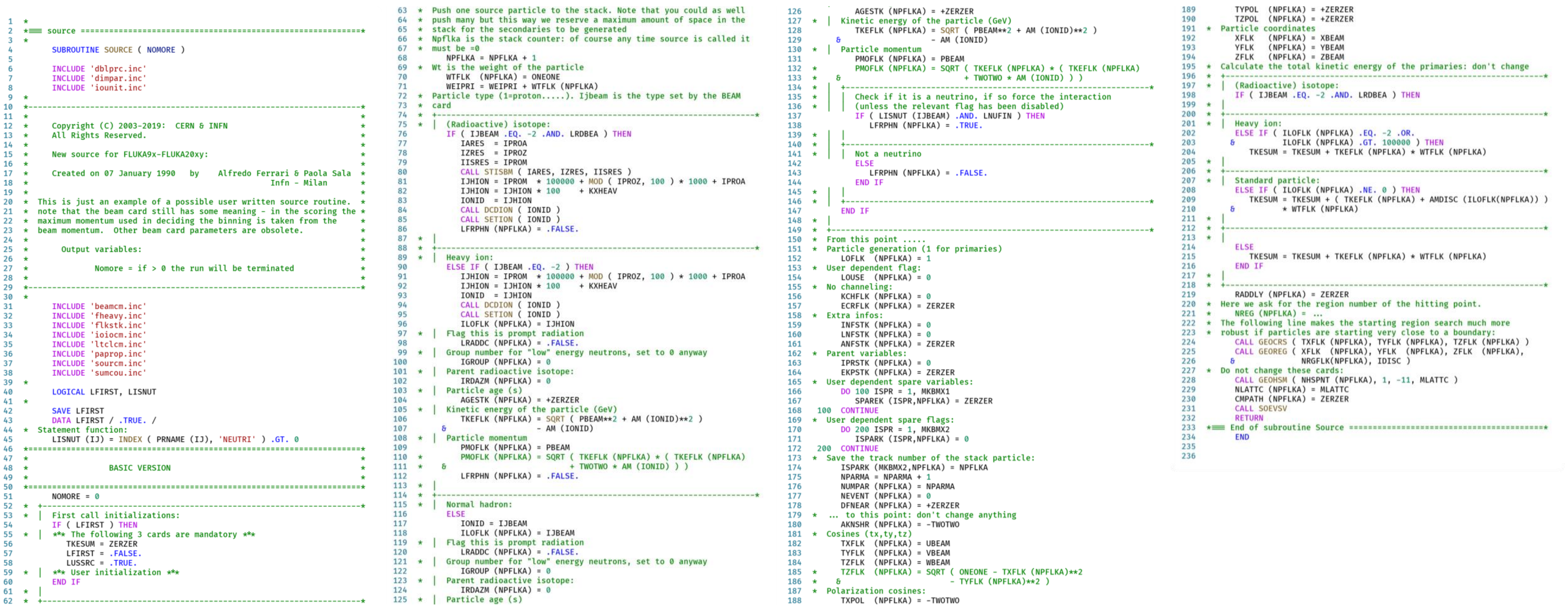
- FLUKA offers plenty of built-in tools to define primary beams and estimate quantities
- Sometime these are not enough
- There is the need to write some dedicated code: a “User Routine”
- URs are beyond the scope of this course because of intrinsic difficulties
- Nevertheless, we have started an effort to make URs more user-friendly
- We want to introduce here the first effort in this direction:

  a modernized version of the **source routine**
- Why the source routine first? Built-in options allow to sample from a limited number

  of distribution and not from histograms. This is an effort to overcome this limitation

The “old” source routine

- Scary for beginners, limited documentation
- Use of `IMPLICIT` and **`FORTRAN77`** naming convention



```
*
*=== source ==========================================================*
*
      SUBROUTINE SOURCE ( NOMORE )

      INCLUDE 'dblprc.inc'
      INCLUDE 'dimpar.inc'
      INCLUDE 'iounit.inc'
*
*----------------------------------------------------------------------*
*                                                                      *
*     Copyright (C) 2003-2019:  CERN & INFN                            *
*     All Rights Reserved.                                             *
*                                                                      *
*     New source for FLUKA9x-FLUKA20xy:                                *
*                                                                      *
*     Created on 07 January 1990   by    Alfredo Ferrari & Paola Sala  *
*                                                   Infn - Milan       *
*                                                                      *
*  This is just an example of a possible user written source routine.  *
*  note that the beam card still has some meaning - in the scoring the *
*  maximum momentum used in deciding the binning is taken from the     *
*  beam momentum.  Other beam card parameters are obsolete.            *
*                                                                      *
*       Output variables:                                              *
*                                                                      *
*              Nomore = if > 0 the run will be terminated              *
*                                                                      *
*----------------------------------------------------------------------*
*
      INCLUDE 'beamcm.inc'
      INCLUDE 'fheavy.inc'
      INCLUDE 'flkstk.inc'
      INCLUDE 'ioiocm.inc'
      INCLUDE 'ltclcm.inc'
      INCLUDE 'paprop.inc'
      INCLUDE 'sourcm.inc'
      INCLUDE 'sumcou.inc'
*
      LOGICAL LFIRST, LISNUT
*
      SAVE LFIRST
      DATA LFIRST / .TRUE. /
*  Statement function:
      LISNUT (IJ) = INDEX ( PRNAME (IJ), 'NEUTRI' ) .GT. 0
*======================================================================*
*                                                                      *
*                 BASIC VERSION                                        *
*                                                                      *
*======================================================================*
      NOMORE = 0
*  +-------------------------------------------------------------------*
*  |  First call initializations:
      IF ( LFIRST ) THEN
*  |  *** The following 3 cards are mandatory ***
         TKESUM = ZERZER
         LFIRST = .FALSE.
         LUSSRC = .TRUE.
*  |  *** User initialization ***
      END IF
*  |
*  +-------------------------------------------------------------------*
*  Push one source particle to the stack. Note that you could as well
*  push many but this way we reserve a maximum amount of space in the
*  stack for the secondaries to be generated
*  Npflka is the stack counter: of course any time source is called it
*  must be =0
      NPFLKA = NPFLKA + 1
*  Wt is the weight of the particle
      WTFLK  (NPFLKA) = ONEONE
      WEIPRI = WEIPRI + WTFLK (NPFLKA)
*  Particle type (1=proton.....). Ijbeam is the type set by the BEAM
*  card
*  +-------------------------------------------------------------------*
*  |  (Radioactive) isotope:
      IF ( IJBEAM .EQ. -2 .AND. LRDBEA ) THEN
         IARES  = IPROA
         IZRES  = IPROZ
         IISRES = IPROM
         CALL STISBM ( IARES, IZRES, IISRES )
         IJHION = IPROM  * 100000 + MOD ( IPROZ, 100 ) * 1000 + IPROA
         IJHION = IJHION * 100    + KXHEAV
         IONID  = IJHION
         CALL DCDION ( IONID )
         CALL SETION ( IONID )
         LFRPHN (NPFLKA) = .FALSE.
*  |
*  +-------------------------------------------------------------------*
*  |  Heavy ion:
      ELSE IF ( IJBEAM .EQ. -2 ) THEN
         IJHION = IPROM  * 100000 + MOD ( IPROZ, 100 ) * 1000 + IPROA
         IJHION = IJHION * 100    + KXHEAV
         IONID  = IJHION
         CALL DCDION ( IONID )
         CALL SETION ( IONID )
         ILOFLK (NPFLKA) = IJHION
*  |  Flag this is prompt radiation
         LRADDC (NPFLKA) = .FALSE.
*  |  Group number for "low" energy neutrons, set to 0 anyway
         IGROUP (NPFLKA) = 0
*  |  Parent radioactive isotope:
         IRDAZM (NPFLKA) = 0
*  |  Particle age (s)
         AGESTK (NPFLKA) = +ZERZER
*  |  Kinetic energy of the particle (GeV)
         TKEFLK (NPFLKA) = SQRT ( PBEAM**2 + AM (IONID)**2 )
     &                   - AM (IONID)
*  |  Particle momentum
         PMOFLK (NPFLKA) = PBEAM
*        PMOFLK (NPFLKA) = SQRT ( TKEFLK (NPFLKA) * ( TKEFLK (NPFLKA)
*    &                            + TWOTWO * AM (IONID) ) )
         LFRPHN (NPFLKA) = .FALSE.
*  |
*  +-------------------------------------------------------------------*
*  |  Normal hadron:
      ELSE
         IONID = IJBEAM
         ILOFLK (NPFLKA) = IJBEAM
*  |  Flag this is prompt radiation
         LRADDC (NPFLKA) = .FALSE.
*  |  Group number for "low" energy neutrons, set to 0 anyway
         IGROUP (NPFLKA) = 0
*  |  Parent radioactive isotope:
         IRDAZM (NPFLKA) = 0
*  |  Particle age (s)
         AGESTK (NPFLKA) = +ZERZER
*  |  Kinetic energy of the particle (GeV)
         TKEFLK (NPFLKA) = SQRT ( PBEAM**2 + AM (IONID)**2 )
     &                   - AM (IONID)
*  |  Particle momentum
         PMOFLK (NPFLKA) = PBEAM
*        PMOFLK (NPFLKA) = SQRT ( TKEFLK (NPFLKA) * ( TKEFLK (NPFLKA)
*    &                            + TWOTWO * AM (IONID) ) )
*  |  +----------------------------------------------------------------*
*  |  |  Check if it is a neutrino, if so force the interaction
*  |  |  (unless the relevant flag has been disabled)
         IF ( LISNUT (IJBEAM) .AND. LNUFIN ) THEN
            LFRPHN (NPFLKA) = .TRUE.
*  |  |
*  |  +----------------------------------------------------------------*
*  |  |  Not a neutrino
         ELSE
            LFRPHN (NPFLKA) = .FALSE.
         END IF
*  |  |
*  |  +----------------------------------------------------------------*
      END IF
*  |
*  +-------------------------------------------------------------------*
*  From this point .....
*  Particle generation (1 for primaries)
      LOFLK  (NPFLKA) = 1
*  User dependent flag:
      LOUSE  (NPFLKA) = 0
*  No channeling:
      KCHFLK (NPFLKA) = 0
      ECRFLK (NPFLKA) = ZERZER
*  Extra infos:
      INFSTK (NPFLKA) = 0
      LNFSTK (NPFLKA) = 0
      ANFSTK (NPFLKA) = ZERZER
*  Parent variables:
      IPRSTK (NPFLKA) = 0
      EKPSTK (NPFLKA) = ZERZER
*  User dependent spare variables:
      DO 100 ISPR = 1, MKBMX1
         SPAREK (ISPR,NPFLKA) = ZERZER
 100  CONTINUE
*  User dependent spare flags:
      DO 200 ISPR = 1, MKBMX2
         ISPARK (ISPR,NPFLKA) = 0
 200  CONTINUE
*  Save the track number of the stack particle:
      ISPARK (MKBMX2,NPFLKA) = NPFLKA
      NPARMA = NPARMA + 1
      NUMPAR (NPFLKA) = NPARMA
      NEVENT (NPFLKA) = 0
      DFNEAR (NPFLKA) = +ZERZER
*  ... to this point: don't change anything
      AKNSHR (NPFLKA) = -TWOTWO
*  Cosines (tx,ty,tz)
      TXFLK  (NPFLKA) = UBEAM
      TYFLK  (NPFLKA) = VBEAM
      TZFLK  (NPFLKA) = WBEAM
*     TZFLK  (NPFLKA) = SQRT ( ONEONE - TXFLK (NPFLKA)**2
*    &                       - TYFLK (NPFLKA)**2 )
*  Polarization cosines:
      TXPOL  (NPFLKA) = -TWOTWO
      TYPOL  (NPFLKA) = +ZERZER
      TZPOL  (NPFLKA) = +ZERZER
*  Particle coordinates
      XFLK   (NPFLKA) = XBEAM
      YFLK   (NPFLKA) = YBEAM
      ZFLK   (NPFLKA) = ZBEAM
*  Calculate the total kinetic energy of the primaries: don't change
*  +-------------------------------------------------------------------*
*  |  (Radioactive) isotope:
      IF ( IJBEAM .EQ. -2 .AND. LRDBEA ) THEN
*  |
*  +-------------------------------------------------------------------*
*  |  Heavy ion:
      ELSE IF ( ILOFLK (NPFLKA) .EQ. -2 .OR.
     &          ILOFLK (NPFLKA) .GT. 100000 ) THEN
         TKESUM = TKESUM + TKEFLK (NPFLKA) * WTFLK (NPFLKA)
*  |
*  +-------------------------------------------------------------------*
*  |  Standard particle:
      ELSE IF ( ILOFLK (NPFLKA) .NE. 0 ) THEN
         TKESUM = TKESUM + ( TKEFLK (NPFLKA) + AMDISC (ILOFLK(NPFLKA)) )
     &          * WTFLK (NPFLKA)
*  |
*  +-------------------------------------------------------------------*
*  |
      ELSE
         TKESUM = TKESUM + TKEFLK (NPFLKA) * WTFLK (NPFLKA)
      END IF
*  |
*  +-------------------------------------------------------------------*
      RADDLY (NPFLKA) = ZERZER
*  Here we ask for the region number of the hitting point.
*     NREG (NPFLKA) =  ...
*  The following line makes the starting region search much more
*  robust if particles are starting very close to a boundary:
      CALL GEOCRS ( TXFLK (NPFLKA), TYFLK (NPFLKA), TZFLK (NPFLKA) )
      CALL GEOREG ( XFLK  (NPFLKA), YFLK  (NPFLKA), ZFLK  (NPFLKA),
     &              NRGFLK(NPFLKA), IDISC )
*  Do not change these cards:
      CALL GEOHSM ( NHSPNT (NPFLKA), 1, -11, MLATTC )
      NLATTC (NPFLKA) = MLATTC
      CMPATH (NPFLKA) = ZERZER
      CALL SOEVSV
      RETURN
*=== End of subroutine Source =========================================*
      END
```

The “new” source routine

- Distributed since FLUKA4-1.0 release
- Simplified appearance
- Long & meaningful names for variables and routines
- Use of **`implicit none`** (see later)
- Abundant comments and examples
- Advanced sampling routines

Removed from snapshot

- Variables for user’s usage clearly indicated
- Lines not to be edited are “hidden” in routines

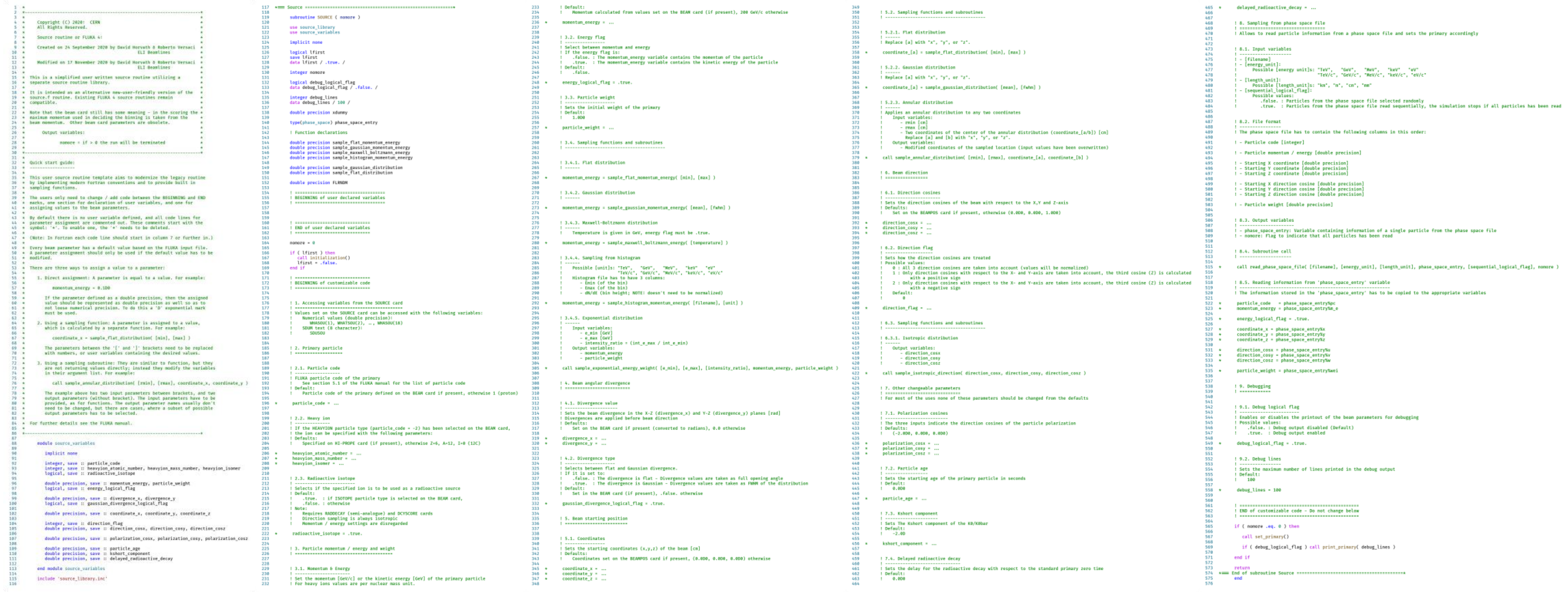
  in the **`source_library.inc`** library file

- **Old source routines can still be used**

```
      ! ==============================
      ! BEGINNING of customizable code
      ! ==============================

*      particle_code = ...

*      heavyion_atomic_number = ...
*      heavyion_mass_number = ...
*      heavyion_isomer = ...

*      radioactive_isotope = .true.

*      momentum_energy = ...

*      energy_logical_flag = .true.

*      particle_weight = ...

*      divergence_x = ...
*      divergence_y = ...

*      gaussian_divergence_logical_flag = .true.

*      coordinate_x = ...
*      coordinate_y = ...
*      coordinate_z = ...

*      direction_cosx = ...
*      direction_cosy = ...
*      direction_cosz = ...

*      direction_flag = ...

*      polarization_cosx = ...
*      polarization_cosy = ...
*      polarization_cosz = ...

*      particle_age = ...
*      kshort_component = ...
*      delayed_radioactive_decay = ...

      ! ===========================================
      ! END of customizable code - Do not change below
      ! ===========================================

      if ( nomore .eq. 0 ) then

         call set_primary()

         if ( debug_logical_flag ) call print_primary( debug_lines )

      end if

      return
*=== End of subroutine Source ======================================*
      end
```

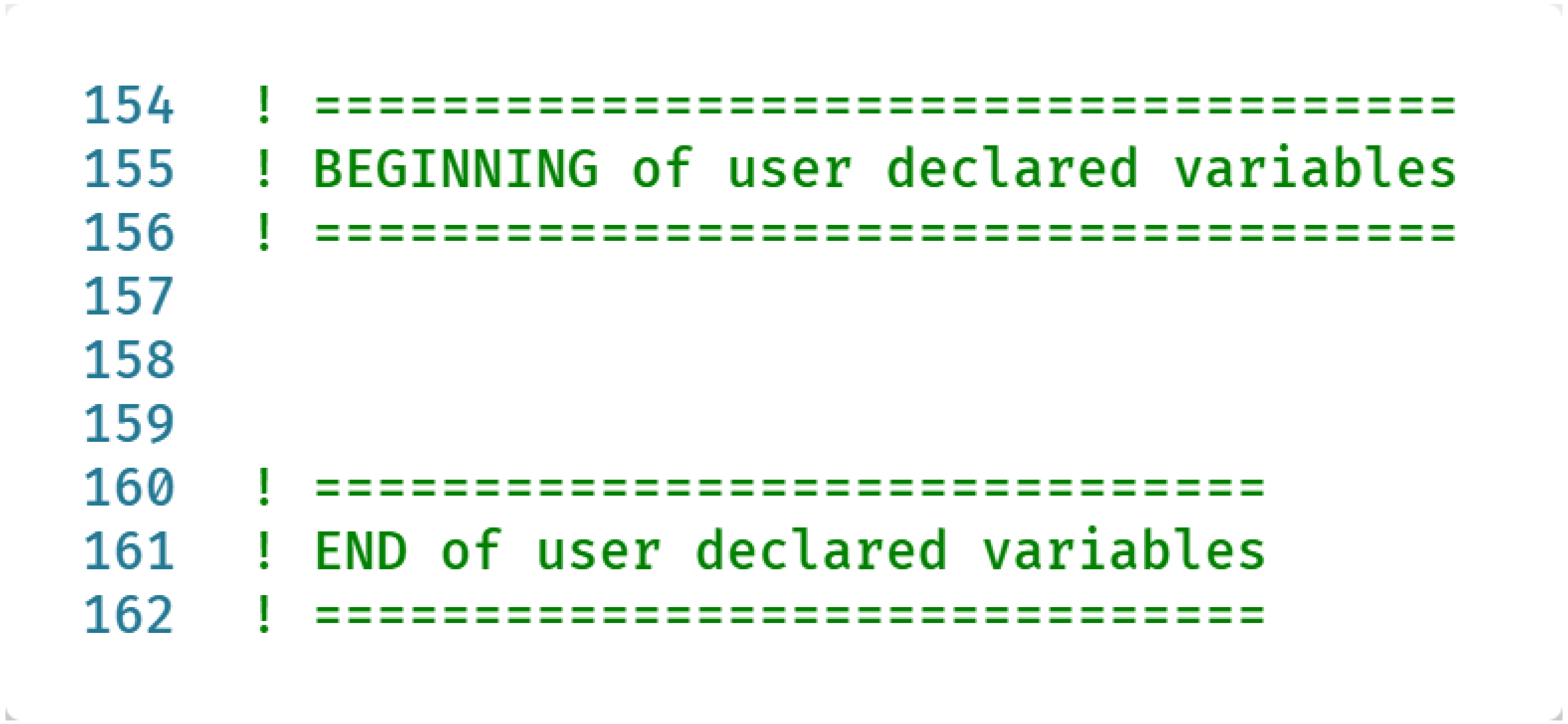
The “new” source routine

- Without removing comments, examples and advanced features (notice the ratio of code and comment lines)
- Note: the snapshot is not meant to be read – Detailed view will follow



```fortran
*
*----------------------------------------------------------------------*
*                                                                      *
*    Copyright (C) 2020:  CERN                                         *
*    All Rights Reserved.                                              *
*                                                                      *
*    Source routine or FLUKA 4:                                        *
*                                                                      *
*    Created on 24 September 2020 by David Horvath & Roberto Versaci   *
*                                          ELI Beamlines               *
*                                                                      *
*    Modified on 17 November 2020 by David Horvath & Roberto Versaci   *
*                                          ELI Beamlines               *
*                                                                      *
*  This is a simplified user written source routine utilizing a        *
*  separate source routine library.                                    *
*                                                                      *
*  It is intended as an alternative new-user-friendly version of the   *
*  source.f routine. Existing FLUKA 4 source routines remain           *
*  compatible.                                                         *
*                                                                      *
*  Note that the beam card still has some meaning - in the scoring the *
*  maximum momentum used in deciding the binning is taken from the     *
*  beam momentum.  Other beam card parameters are obsolete.            *
*                                                                      *
*      Output variables:                                               *
*                                                                      *
*             nomore = if > 0 the run will be terminated               *
*                                                                      *
*----------------------------------------------------------------------*
*
*  Quick start guide:
*  ------------------
*
*  This user source routine template aims to modernize the legacy routine
*  by implementing modern Fortran conventions and to provide built in
*  sampling functions.
*
*  The users only need to change / add code between the BEGINNING and END
*  marks, one section for declaration of user variables, and one for
*  assigning values to the beam parameters.
*
*  By default there is no user variable defined, and all code lines for
*  parameter assignment are commented out. These comments start with the
*  symbol: '*'. To enable one, the '*' needs to be deleted.
*
*  (Note: In Fortran each code line should start in column 7 or further in.)
*
*  Every beam parameter has a default value based on the FLUKA input file.
*  A parameter assignment should only be used if the default value has to be
*  modified.
*
*  There are three ways to assign a value to a parameter:
*
*     1. Direct assignment: A parameter is equal to a value. For example:
*
*           momentum_energy = 0.1D0
*
*        If the parameter defined as a double precision, then the assigned
*        value should be represented as double precision as well so as to
*        not loose numerical precision. To do this a 'D' exponential mark
*        must be used.
*
*     2. Using a sampling function: A parameter is assigned to a value,
*        which is calculated by a separate function. For example:
*
*           coordinate_x = sample_flat_distribution( [min], [max] )
*
*        The parameters between the '[' and ']' brackets need to be replaced
*        with numbers, or user variables containing the desired values.
*
*     3. Using a sampling subroutine: They are similar to function, but they
*        are not returning values directly; instead they modify the variables
*        in their argument list. For example:
*
*           call sample_annular_distribution( [rmin], [rmax], coordinate_x, coordinate_y )
*
*        The example above has two input parameters between brackets, and two
*        output parameters (without bracket). The input parameters have to be
*        provided, as for functions. The output parameter names usually don't
*        need to be changed, but there are cases, where a subset of possible
*        output parameters has to be selected.
*
*  For further details see the FLUKA manual.
*
*----------------------------------------------------------------------*

      module source_variables

         implicit none

         integer, save :: particle_code
         integer, save :: heavyion_atomic_number, heavyion_mass_number, heavyion_isomer
         logical, save :: radioactive_isotope

         double precision, save :: momentum_energy, particle_weight
         logical, save :: energy_logical_flag

         double precision, save :: divergence_x, divergence_y
         logical, save :: gaussian_divergence_logical_flag

         double precision, save :: coordinate_x, coordinate_y, coordinate_z

         integer, save :: direction_flag
         double precision, save :: direction_cosx, direction_cosy, direction_cosz

         double precision, save :: polarization_cosx, polarization_cosy, polarization_cosz

         double precision, save :: particle_age
         double precision, save :: kshort_component
         double precision, save :: delayed_radioactive_decay

      end module source_variables

      include 'source_library.inc'

*=== Source ===========================================================*

      subroutine SOURCE ( nomore )

      use source_library
      use source_variables

      implicit none

      logical lfirst
      save lfirst
      data lfirst / .true. /

      integer nomore

      logical debug_logical_flag
      data debug_logical_flag / .false. /

      integer debug_lines
      data debug_lines / 100 /

      double precision xdummy

      type(phase_space) phase_space_entry

      ! Function declarations

      double precision sample_flat_momentum_energy
      double precision sample_gaussian_momentum_energy
      double precision sample_maxwell_boltzmann_energy
      double precision sample_histogram_momentum_energy

      double precision sample_gaussian_distribution
      double precision sample_flat_distribution

      double precision FLRNDM

      ! ===================================
      ! BEGINNING of user declared variables
      ! ===================================



      ! =============================
      ! END of user declared variables
      ! =============================

      nomore = 0

      if ( lfirst ) then
         call initialization()
         lfirst = .false.
      end if

      ! =============================
      ! BEGINNING of customizable code
      ! =============================


      ! 1. Accessing variables from the SOURCE card
      ! ==========================================
      ! Values set on the SOURCE card can be accessed with the following variables:
      !    Numerical values (double precision):
      !       WHASOU(1), WHATSOU(2), …, WHASOU(18)
      !    SDUM text (8 character):
      !       SDUSOU


      ! 2. Primary particle
      ! ===================


      ! 2.1. Particle code
      ! ------------------
      ! FLUKA particle code of the primary
      !    See section 5.1 of the FLUKA manual for the list of particle code
      ! Default:
      !    Particle code of the primary defined on the BEAM card if present, otherwise 1 (proton)

*      particle_code = …


      ! 2.2. Heavy ion
      ! --------------
      ! If the HEAVYION particle type (particle_code = -2) has been selected on the BEAM card,
      ! the ion can be specified with the following parameters:
      ! Defaults:
      !    Specified on HI-PROPE card (if present), otherwise Z=6, A=12, I=0 (12C)

*      heavyion_atomic_number = …
*      heavyion_mass_number = …
*      heavyion_isomer = …


      ! 2.3. Radioactive isotope
      ! ------------------------
      ! Selects if the specified ion is to be used as a radioactive source
      ! Default:
      !    .true.  : if ISOTOPE particle type is selected on the BEAM card,
      !    .false. : otherwise
      ! Note:
      !    Requires RADDECAY (semi-analogue) and DCYSCORE cards
      !    Direction sampling is always isotropic
      !    Momentum / energy settings are disregarded

*      radioactive_isotope = .true.


      ! 3. Particle momentum / energy and weight
      ! ======================================


      ! 3.1. Momentum & Energy
      ! ----------------------
      ! Set the momentum [GeV/c] or the kinetic energy [GeV] of the primary particle
      ! For heavy ions values are per nuclear mass unit.
      ! Default:
      !    Momentum calculated from values set on the BEAM card (if present), 200 GeV/c otherwise

*      momentum_energy = …


      ! 3.2. Energy flag
      ! ----------------
      ! Select between momentum and energy
      ! If the energy flag is:
      !    .false. : The momentum_energy variable contains the momentum of the particle
      !    .true.  : The momentum_energy variable contains the kinetic energy of the particle
      ! Default:
      !    .false.

*      energy_logical_flag = .true.


      ! 3.3. Particle weight
      ! --------------------
      ! Sets the initial weight of the primary
      ! Default:
      !    1.0D0

*      particle_weight = …


      ! 3.4. Sampling functions and subroutines
      ! ----------------------------------------


      ! 3.4.1. Flat distribution
      ! ------

*      momentum_energy = sample_flat_momentum_energy( [min], [max] )


      ! 3.4.2. Gaussian distribution
      ! ------

*      momentum_energy = sample_gaussian_momentum_energy( [mean], [fwhm] )


      ! 3.4.3. Maxwell-Boltzmann distribution
      ! ------
      !    Temperature is given in GeV, energy flag must be .true.

*      momentum_energy = sample_maxwell_boltzmann_energy( [temperature] )


      ! 3.4.4. Sampling from histogram
      ! ------
      !    Possible [unit]s: "TeV",   "GeV",   "MeV",   "keV"   "eV"
      !                      "TeV/c", "GeV/c", "MeV/c", "keV/c", "eV/c"
      !    Histogram file has to have 3 columns:
      !       - Emin (of the bin)
      !       - Emax (of the bin)
      !       - dN/dE (bin height; NOTE: doesn't need to be normalized)

*      momentum_energy = sample_histogram_momentum_energy( [filename], [unit] )


      ! 3.4.5. Exponential distribution
      ! ------
      !    Input variables:
      !       - e_min [GeV]
      !       - e_max [GeV]
      !       - intensity_ratio = (int_e_max / int_e_min)
      !    Output variables:
      !       - momentum_energy
      !       - particle_weight

*      call sample_exponential_energy_weight( [e_min], [e_max], [intensity_ratio], momentum_energy, particle_weight )


      ! 4. Beam angular divergence
      ! ==========================


      ! 4.1. Divergence value
      ! ---------------------
      ! Sets the beam divergence in the X-Z (divergence_x) and Y-Z (divergence_y) planes [rad]
      ! Divergences are applied before beam direction
      ! Defaults:
      !    Set on the BEAM card if present (converted to radians), 0.0 otherwise

*      divergence_x = …
*      divergence_y = …


      ! 4.2. Divergence type
      ! --------------------
      ! Selects between flat and Gaussian divergence.
      ! If it is set to:
      !    .false. : The divergence is flat - Divergence values are taken as full opening angle
      !    .true.  : The divergence is Gaussian - Divergence values are taken as FWHM of the distribution
      ! Default:
      !    Set in the BEAM card (if present), .false. otherwise

*      gaussian_divergence_logical_flag = .true.


      ! 5. Beam starting position
      ! =========================


      ! 5.1. Coordinates
      ! ----------------
      ! Sets the starting coordinates (x,y,z) of the beam [cm]
      ! Defaults:
      !    Coordinates set on the BEAMPOS card if present, (0.0D0, 0.0D0, 0.0D0) otherwise

*      coordinate_x = …
*      coordinate_y = …
*      coordinate_z = …


      ! 5.2. Sampling functions and subroutines
      ! ----------------------------------------


      ! 5.2.1. Flat distribution
      ! ------
      ! Replace [a] with "x", "y", or "z".

*      coordinate_[a] = sample_flat_distribution( [min], [max] )


      ! 5.2.2. Gaussian distribution
      ! ------
      ! Replace [a] with "x", "y", or "z".

*      coordinate_[a] = sample_gaussian_distribution( [mean], [fwhm] )


      ! 5.2.3. Annular distribution
      ! ------
      ! Applies an annular distribution to any two coordinates
      !    Input variables:
      !       - rmin [cm]
      !       - rmax [cm]
      !       - Two coordinates of the center of the annular distribution (coordinate_[a/b]) [cm]
      !         Replace [a] and [b] with "x", "y", or "z".
      !    Output variables:
      !       - Modified coordinates of the sampled location (input values have been overwritten)

*      call sample_annular_distribution( [rmin], [rmax], coordinate_[a], coordinate_[b] )


      ! 6. Beam direction
      ! =================


      ! 6.1. Direction cosines
      ! ----------------------
      ! Sets the direction cosines of the beam with respect to the X,Y and Z-axis
      ! Defaults:
      !    Set on the BEAMPOS card if present, otherwise (0.0D0, 0.0D0, 1.0D0)

*      direction_cosx = …
*      direction_cosy = …
*      direction_cosz = …


      ! 6.2. Direction flag
      ! -------------------
      ! Sets how the direction cosines are treated
      ! Possible values:
      !    0 : All 3 direction cosines are taken into account (values will be normalized)
      !    1 : Only direction cosines with respect to the X- and Y-axis are taken into account, the third cosine (Z) is calculated
      !          with a positive sign
      !    2 : Only direction cosines with respect to the X- and Y-axis are taken into account, the third cosine (Z) is calculated
      !          with a negative sign
      !    Default:
      !        0

*      direction_flag = …


      ! 6.3. Sampling functions and subroutines
      ! ----------------------------------------

      ! 6.3.1. Isotropic distribution
      ! ------
      !    Output variables:
      !       - direction_cosx
      !       - direction_cosy
      !       - direction_cosz

*      call sample_isotropic_direction( direction_cosx, direction_cosy, direction_cosz )


      ! 7. Other changeable parameters
      ! ==============================
      ! For most of the uses none of these parameters should be changed from the defaults


      ! 7.1. Polarization cosines
      ! -------------------------
      ! The three inputs indicate the direction cosines of the particle polarization
      ! Defaults:
      !    (-2.0D0, 0.0D0, 0.0D0)

*      polarization_cosx = …
*      polarization_cosy = …
*      polarization_cosz = …


      ! 7.2. Particle age
      ! -----------------
      ! Sets the starting age of the primary particle in seconds
      ! Default:
      !    0.0D0

*      particle_age = …


      ! 7.3. Kshort component
      ! ---------------------
      ! Sets The Kshort component of the K0/K0bar
      ! Default:
      !    -2.0D

*      kshort_component = …


      ! 7.4. Delayed radioactive decay
      ! ------------------------------
      ! Sets the delay for the radioactive decay with respect to the standard primary zero time
      ! Default:
      !    0.0D0

*      delayed_radioactive_decay = …


      ! 8. Sampling from phase space file
      ! =================================
      ! Allows to read particle information from a phase space file and sets the primary accordingly


      ! 8.1. Input variables
      ! --------------------
      ! - [filename]
      ! - [energy_unit]:
      !      Possible [energy unit]s: "TeV",   "GeV",   "MeV",   "keV"   "eV"
      !                               "TeV/c", "GeV/c", "MeV/c", "keV/c", "eV/c"
      ! - [length_unit]:
      !      Possible [length_unit]s: "km", "m", "cm", "mm"
      ! - [sequential_logical_flag]:
      !      Possible values:
      !         .false. : Particles from the phase space file selected randomly
      !         .true.  : Particles from the phase space file read sequentially, the simulation stops if all particles has been read


      ! 8.2. File format
      ! ----------------
      ! The phase space file has to contain the following columns in this order:

      ! - Particle code [integer]

      ! - Particle momentum / energy [double precision]

      ! - Starting X coordinate [double precision]
      ! - Starting Y coordinate [double precision]
      ! - Starting Z coordinate [double precision]

      ! - Starting X direction cosine [double precision]
      ! - Starting Y direction cosine [double precision]
      ! - Starting Z direction cosine [double precision]

      ! - Particle weight [double precision]


      ! 8.3. Output variables
      ! ---------------------
      ! - phase_space_entry: Variable containing information of a single particle from the phase space file
      ! - nomore: Flag to indicate that all particles has been read


      ! 8.4. Subroutine call
      ! --------------------

*      call read_phase_space_file( [filename], [energy_unit], [length_unit], phase_space_entry, [sequential_logical_flag], nomore )


      ! 8.5. Reading information from 'phase_space_entry' variable
      ! ----------------------------------------------------------
      ! The information stored in the 'phase_space_entry' has to be copied to the appropriate variables

*      particle_code   = phase_space_entry%pc
*      momentum_energy = phase_space_entry%m_e

*      energy_logical_flag = .true.

*      coordinate_x = phase_space_entry%x
*      coordinate_y = phase_space_entry%y
*      coordinate_z = phase_space_entry%z

*      direction_cosx = phase_space_entry%u
*      direction_cosy = phase_space_entry%v
*      direction_cosz = phase_space_entry%w

*      particle_weight = phase_space_entry%wei


      ! 9. Debugging
      ! ============


      ! 9.1. Debug logical flag
      ! -----------------------
      ! Enables or disables the printout of the beam parameters for debugging
      ! Possible values:
      !    .false. : Debug output disabled (Default)
      !    .true.  : Debug output enabled

*      debug_logical_flag = .true.


      ! 9.2. Debug lines
      ! ----------------
      ! Sets the maximum number of lines printed in the debug output
      ! Default:
      !    100

*      debug_lines = 100


      ! ============================================
      ! END of customizable code - Do not change below
      ! ============================================

      if ( nomore .eq. 0 ) then

         call set_primary()

         if ( debug_logical_flag ) call print_primary( debug_lines )

      end if

      return
*=== End of subroutine Source =========================================*
      end
```

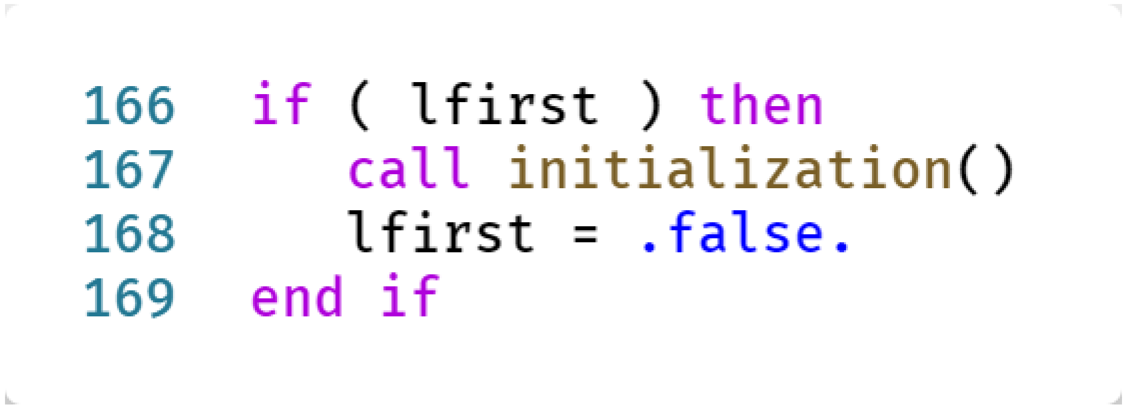
Source routine – Initialization



```
! ====================================
! BEGINNING of user declared variables
! ====================================



! ==============================
! END of user declared variables
! ==============================
```

- Dedicated space for the declaration of user variables (and functions)

Source routine – Initialization

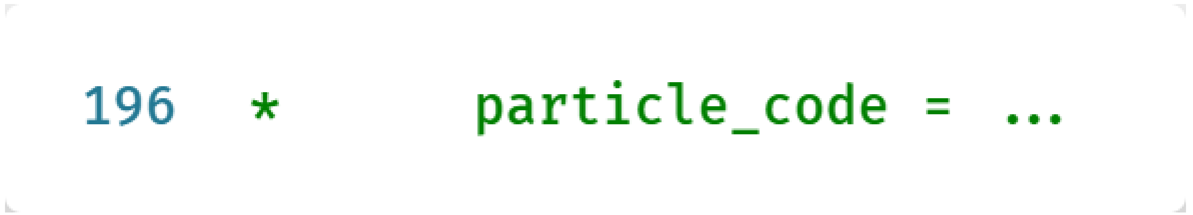


```
if ( lfirst ) then
   call initialization()
   lfirst = .false.
end if
```

- Initialization of internal variables
- Only performed the first time the routine is called

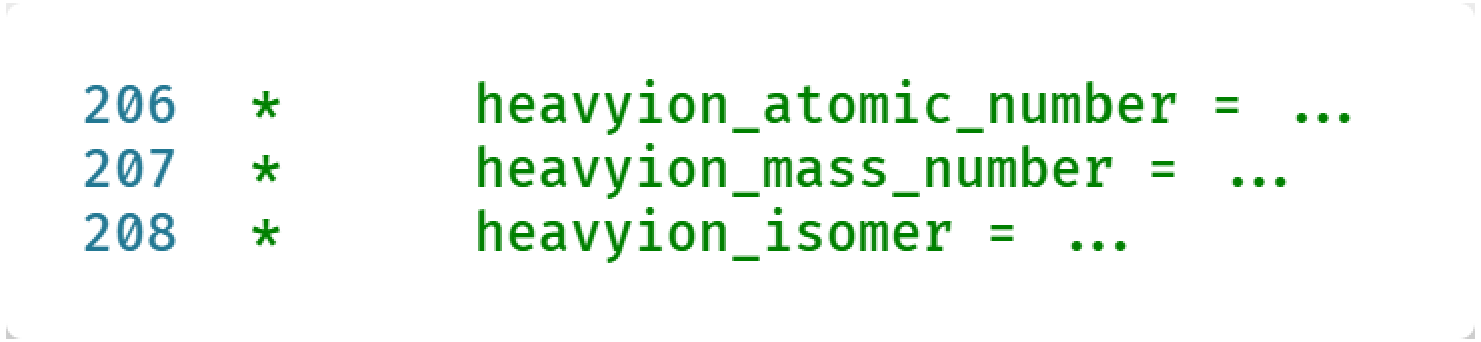
- To overwrite the default values the relevant lines needs to be uncommented, by removing the ‘*’ at the beginning of the line.
  (See next slides)

Source routine – Primary particle



```
*       particle_code = ...
```

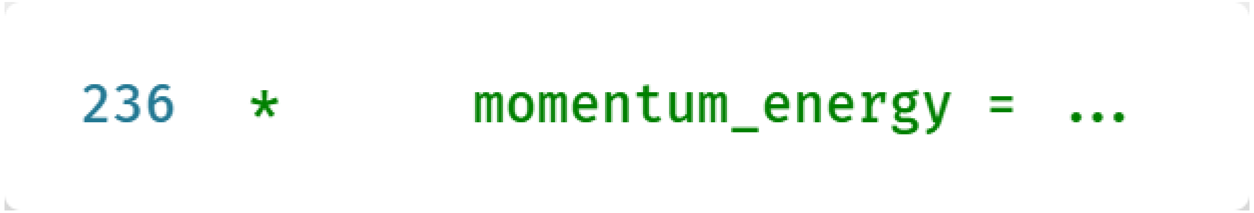
- By default, the particle type given in the **BEAM** card is taken
- Particle codes explained in FLUKA manual section 5.1
- Possible application: beam made of more than one type particles



```
*       heavyion_atomic_number = ...
*       heavyion_mass_number = ...
*       heavyion_isomer = ...
```

- Only used if primary particle is set to HEAVYION or ISOTOPE
- Default values are set on the **HI-PROPE** card, or for ^{12}C if the card is missing

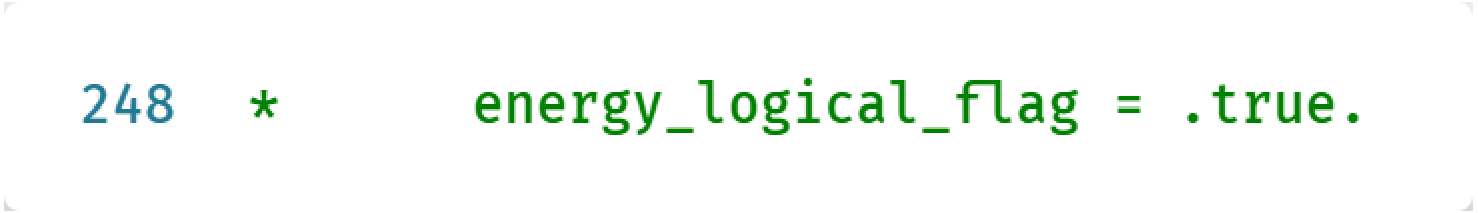
Source routine – Energy / momentum



```
*       momentum_energy = ...
```

- By default, the particle <u>momentum</u> is expected
- The default value is based on the **BEAM** card
  (Automatically converted into momentum if energy is given on the **BEAM** card)

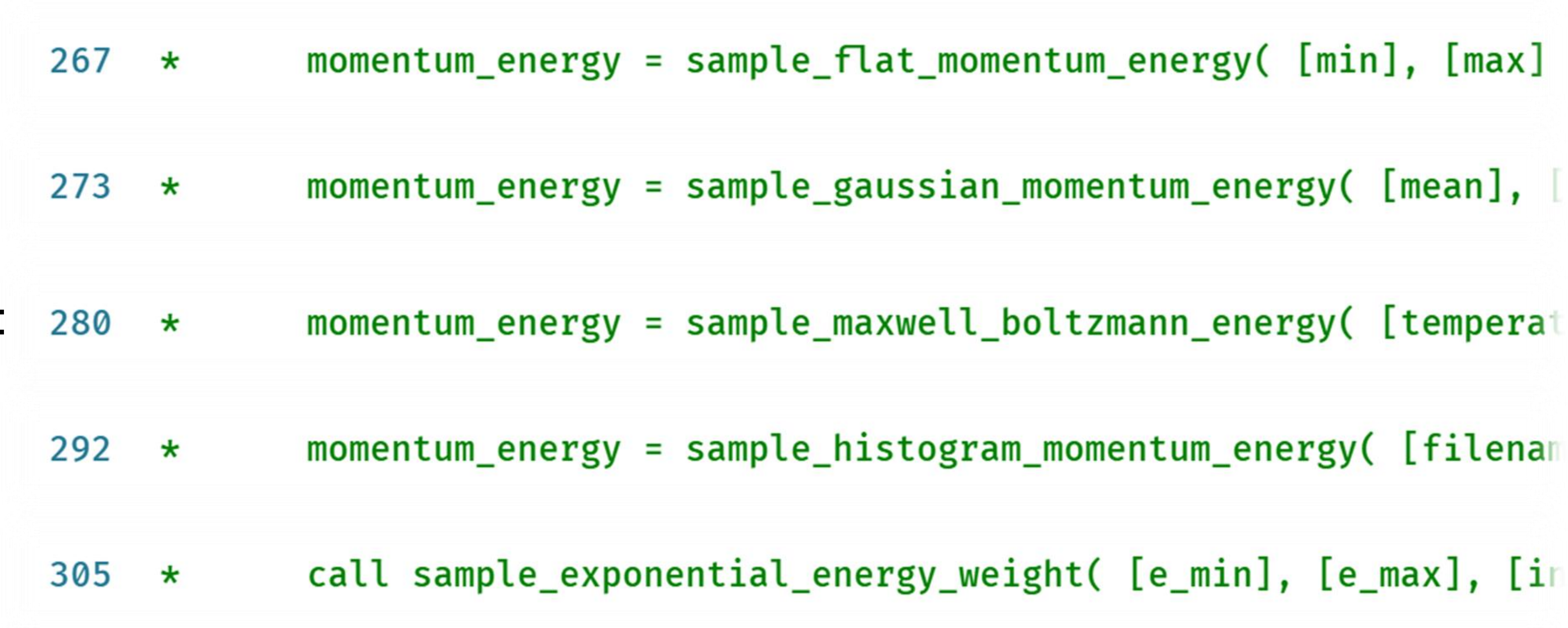
- If energy is specified in the source routine, the following logical value must be set *.true.*



```
*       energy_logical_flag = .true.
```

Source routine – Energy / momentum

- The momentum divergence set on the **BEAM** card is not retained
- It in necessary to specify in the source routine
- It is easy with the supplied functions / subroutine



Flat spectrum:

```
*       momentum_energy = sample_flat_momentum_energy( [min], [max]
```

Gaussian spectrum:

```
*       momentum_energy = sample_gaussian_momentum_energy( [mean], [
```

Maxwell-Boltzmann spectrum:

```
*       momentum_energy = sample_maxwell_boltzmann_energy( [temperat
```

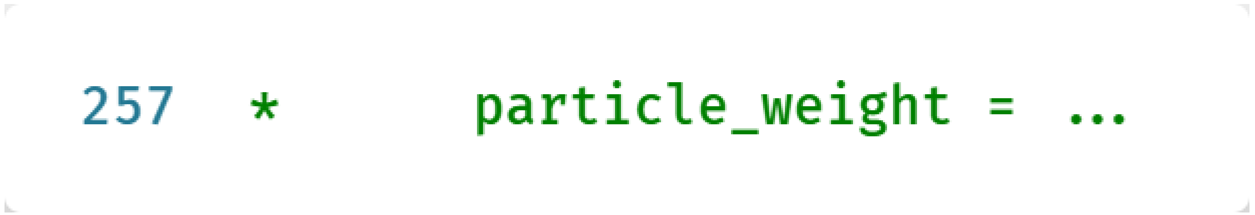
Spectrum from histogram:

```
*       momentum_energy = sample_histogram_momentum_energy( [filenam
```

Exponential spectrum:
(biased sampling)

```
*       call sample_exponential_energy_weight( [e_min], [e_max], [i
```

Source routine – Particle weight



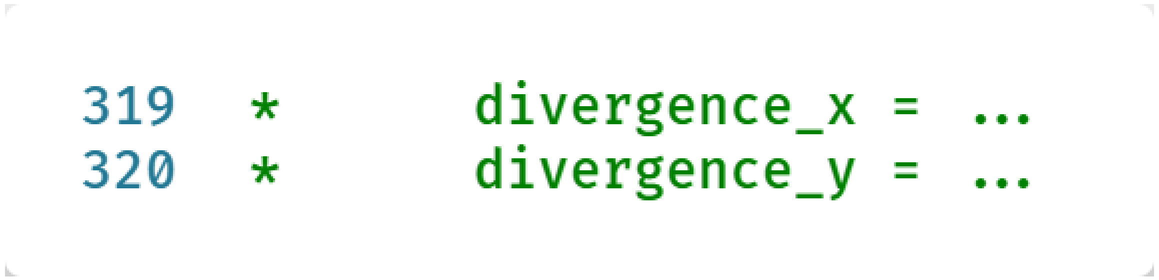
```
*       particle_weight = ...
```

- Monte Carlo concept for biased sources

- The default value (**`particle_weight = 1.0`**) is usually sufficient
- Not for a beginners’ use, mentioned here for completeness

- Note: The exponential spectrum sampling subroutine, uses variable particle weight

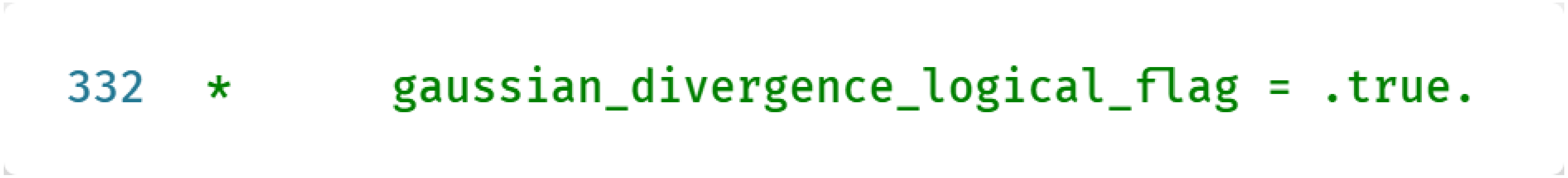
Source routine – Beam divergence



```
*       divergence_x = ...
*       divergence_y = ...
```

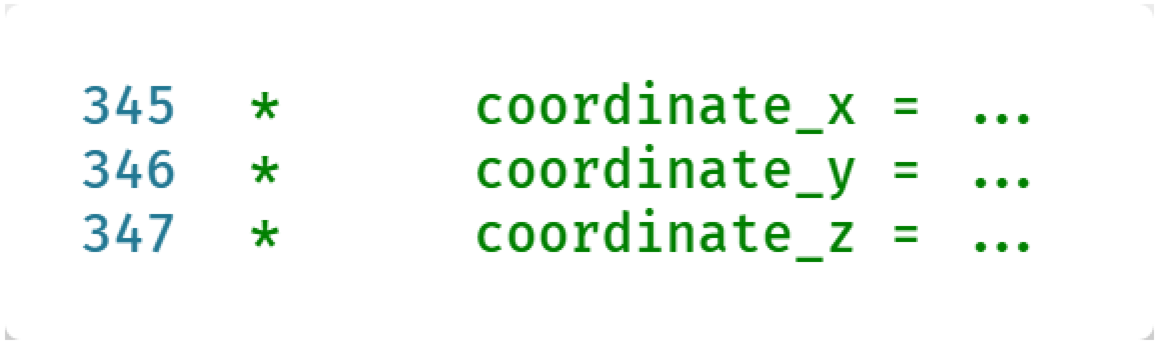
- By default:
  - values are taken from the **BEAM** card
  - It is assumed to be a flat angular distribution

- For Gaussian divergence the following logical value must be set *.true.*



```
*       gaussian_divergence_logical_flag = .true.
```

Source routine – Beam starting position



```
*      coordinate_x = ...
*      coordinate_y = ...
*      coordinate_z = ...
```

- By default, values are taken from the **BEAMPOS** card

- Beam shape set on the **BEAM** card, and
- Extended sources specified on additional **BEAMPOS** cards are not implemented

Source routine – Beam starting position

- Some predefined routines (2 functions and 1 subroutine) are already available:

Flat distribution:

```
*        coordinate_[a] = sample_flat_distribution( [min], [max] )
```

Gaussian distribution:

```
*        coordinate_[a] = sample_gaussian_distribution( [mean], [fwhm] )
```

Annular distribution:

```
*        call sample_annular_distribution( [rmin], [rmax], coordinate_[a],
```

Remember the values must be in double precision (**`1.0D0`**).

Note: If annular sampling is used, the coordinates has to be set manually as well.

Source routine – Beam direction

```
*       direction_cosx =  ...
*       direction_cosy =  ...
*       direction_cosz =  ...
```

- By default, values are taken from the **BEAMPOS** card

- If the **direction_flag** is set to:

```
*       direction_flag =  ...
```

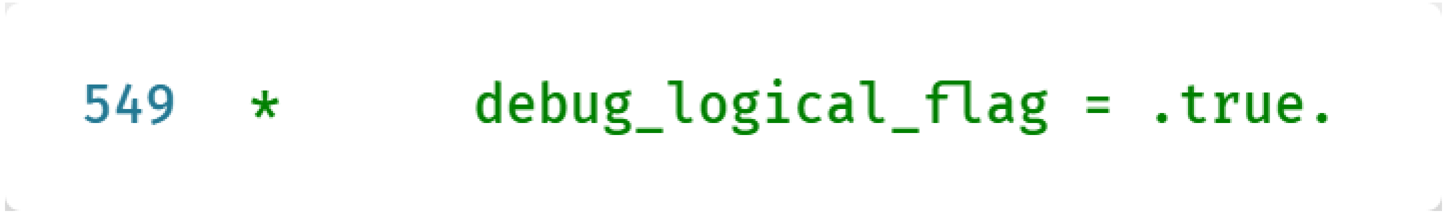
  - 0 : All three values are considered and the they are normalized automatically (Default)
  - 1 : The manually set value of the z direction is disregarded. Instead, it is calculated from the x and y direction cosines with a positive sign.
  - 2 : As with option 1, but negative sign is used.

- A predefined subroutine is are already available for isotropic direction sampling

```
*       call sample_isotropic_direction( direction_cosx, direction_cosy, direction_cosz )
```

Source routine – Debugging

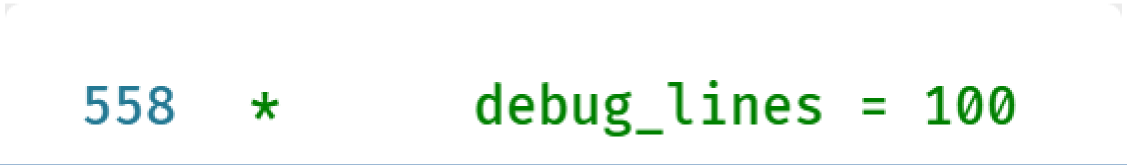
- To help debug the source routine, the major particle parameters can be printed
- To enable this feature, set



```
*       debug_logical_flag = .true.
```

- The printed parameters:
  - Energy / momentum
  - Coordinates
  - Direction
  - Weight

- The number of primaries printed can be set with:



```
*       debug_lines = 100
```

Some predefined FLUKA random sampling routines

- FLUKA offers some useful, predefined routines for random sampling

- `my_variable = FLRNDM(XDUMMY)`

  Assigns a 64-bit random number in [0,1)
- `call FLNRRN(gauss1)`

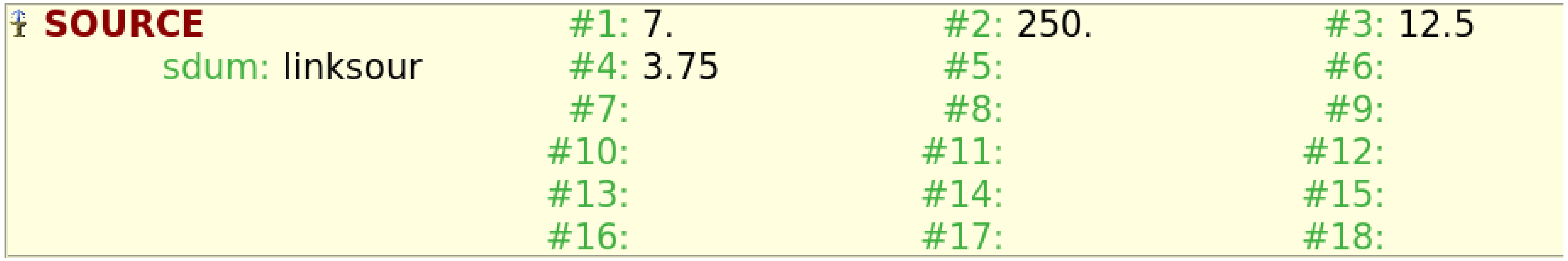
  Returns a Gaussian distributed random number
- `call FLNRR2(gauss1,gauss2)`

  Returns two uncorrelated Gaussian distributed random numbers
- `call SFECFE(sint,cost)`

  Returns sine and cosine of a random azimuthal angle

SOURCE card and passing parameters

- To invoke a source routine, it is necessary to add a **SOURCE** card
- A **SOURCE** card can be empty or can be used to pass parameters to the routine
- Max. 18 numerical values (**WHASOU(ii)**) and 1 string (max. 8 characters) (**SDUSOU**) can be



```
SOURCE                       #1: 7.            #2: 250.          #3: 12.5
        sdum: linksour       #4: 3.75          #5:               #6:
                             #7:               #8:               #9:
                            #10:              #11:              #12:
                            #13:              #14:              #15:
                            #16:              #17:              #18:
```

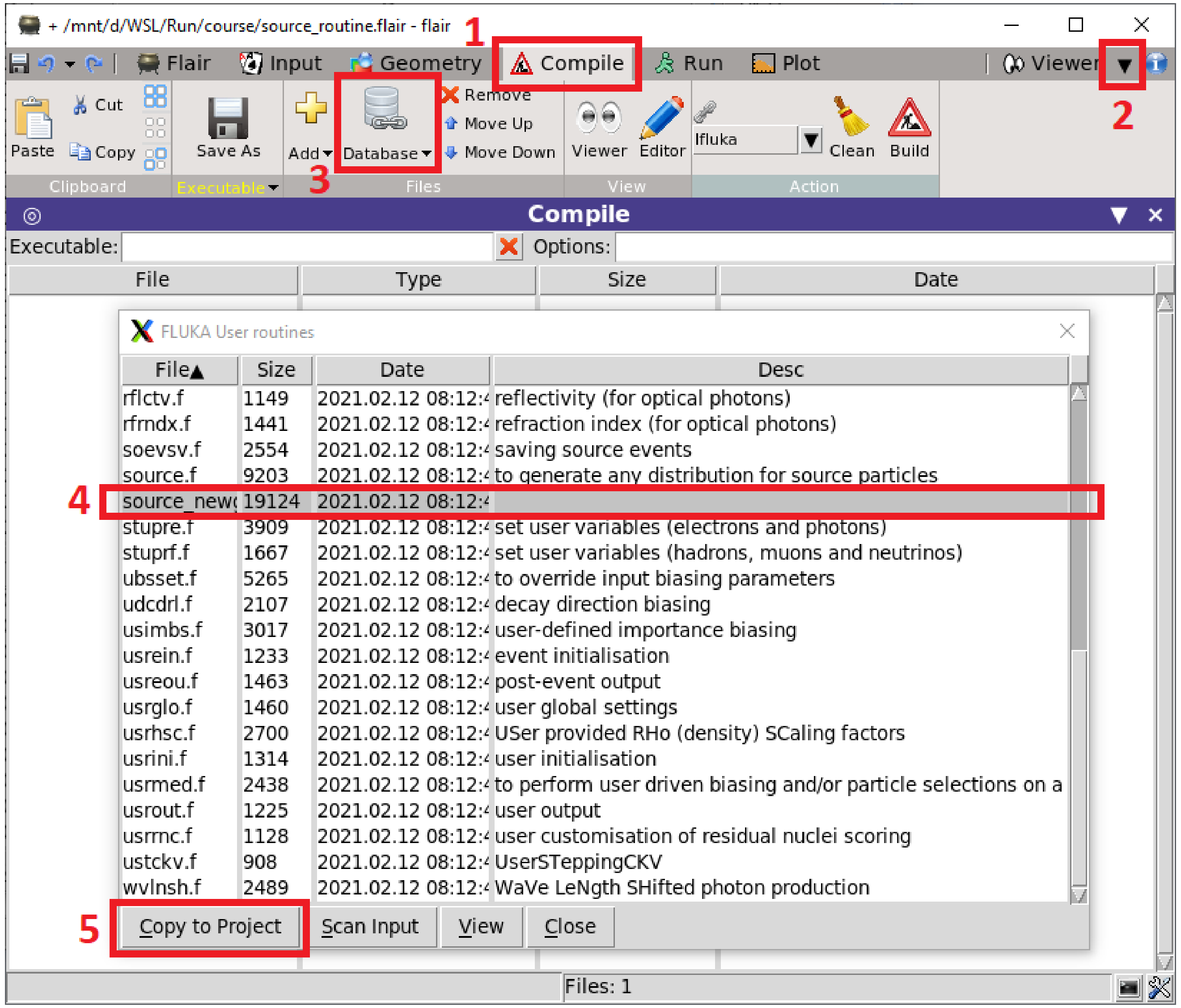
- Good practice advice:

  Even if the beam energy / momentum is defined in the source routine, specify it in the **BEAM** card as it is used for internal initialization.
  Set a momentum value higher than the maximum possible one.

Adding the user routine to the project folder

1. Open [Compile] tab
2. It is maybe hidden in the dropdown menu
3. Click the [Database] button (Use [Add] for an existing file)
4. Select the user routine you want to use
5. Click [Copy to Project]

The copied user routine will be in the Flair projects directory

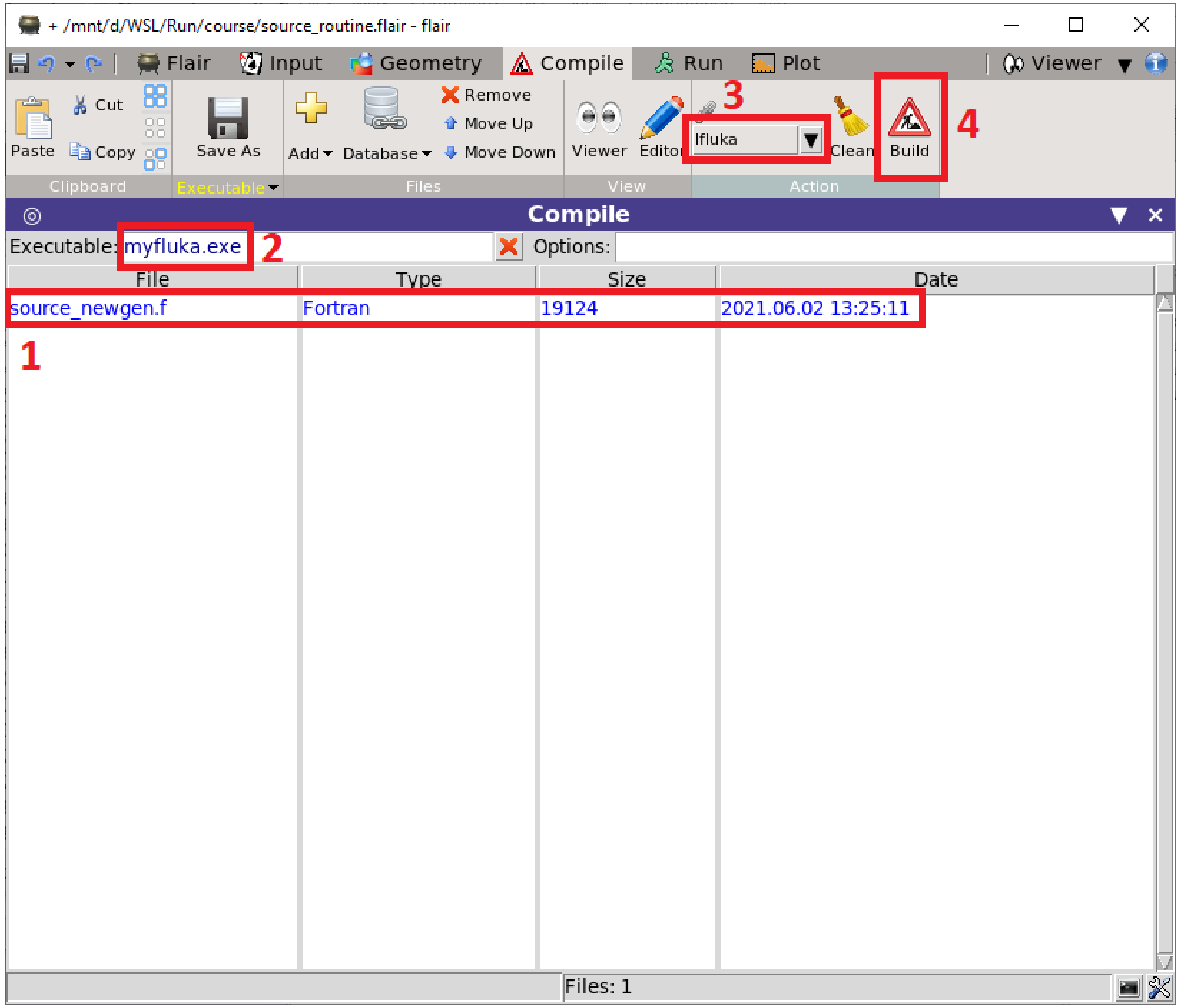


File▲	Size	Date	Desc
rflctv.f	1149	2021.02.12 08:12:	reflectivity (for optical photons)
rfrndx.f	1441	2021.02.12 08:12:	refraction index (for optical photons)
soevsv.f	2554	2021.02.12 08:12:	saving source events
source.f	9203	2021.02.12 08:12:	to generate any distribution for source particles
source_new	19124	2021.02.12 08:12:	
stupre.f	3909	2021.02.12 08:12:	set user variables (electrons and photons)
stuprf.f	1667	2021.02.12 08:12:	set user variables (hadrons, muons and neutrinos)
ubsset.f	5265	2021.02.12 08:12:	to override input biasing parameters
udcdrl.f	2107	2021.02.12 08:12:	decay direction biasing
usimbs.f	3017	2021.02.12 08:12:	user-defined importance biasing
usrein.f	1233	2021.02.12 08:12:	event initialisation
usreou.f	1463	2021.02.12 08:12:	post-event output
usrglo.f	1460	2021.02.12 08:12:	user global settings
usrhsc.f	2700	2021.02.12 08:12:	USer provided RHo (density) SCaling factors
usrini.f	1314	2021.02.12 08:12:	user initialisation
usrmed.f	2438	2021.02.12 08:12:	to perform user driven biasing and/or particle selections on a
usrout.f	1225	2021.02.12 08:12:	user output
usrrnc.f	1128	2021.02.12 08:12:	user customisation of residual nuclei scoring
ustckv.f	908	2021.02.12 08:12:	UserSTeppingCKV
wvlnsh.f	2489	2021.02.12 08:12:	WaVe LeNgth SHifted photon production

Compiling a custom FLUKA executable

1. Verify that the user routine is in the list
2. Name your custom executable
3. Select the appropriate linker:
   a. Use *lfluka* by default
   b. Use *ldpmqmd* if DPMJET or RQMD models are needed
4. Compile the executable

The custom executable should be set default on the [Run] tab automatically



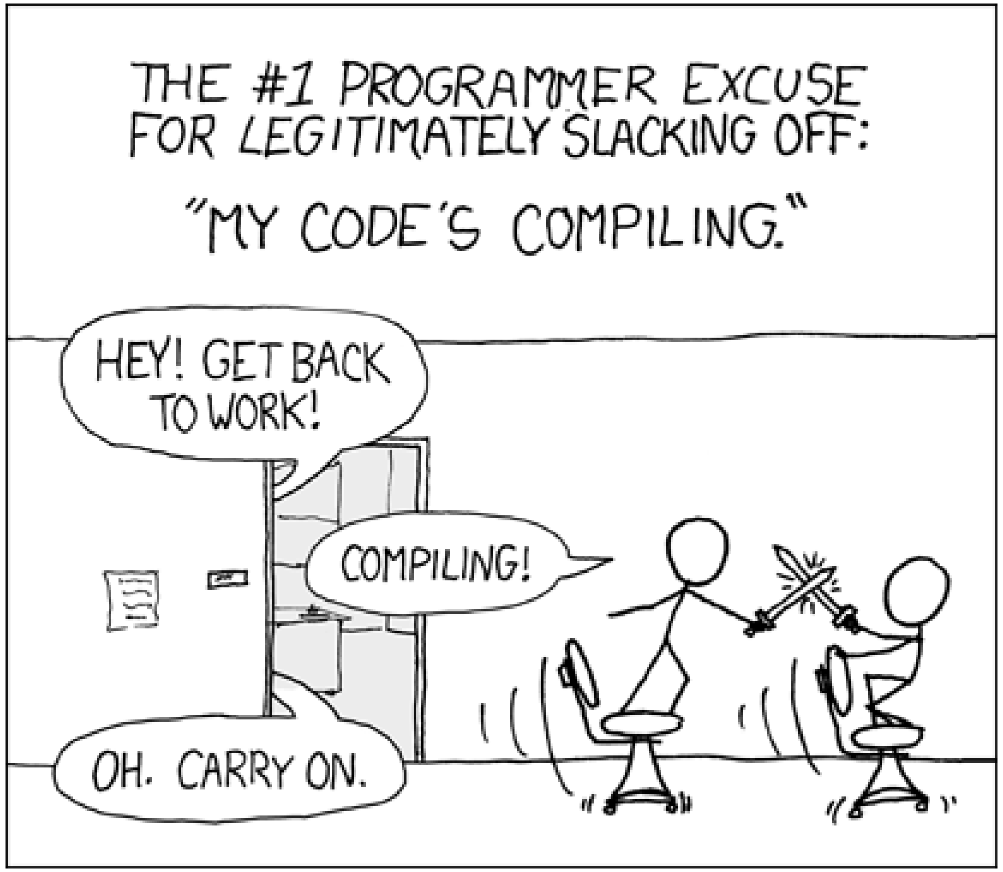
Executable: myfluka.exe Options:

File	Type	Size	Date
source_newgen.f	Fortran	19124	2021.06.02 13:25:11

Files: 1

Time to do some hands-on practice!

- We will now see together a few small examples of the “new” source routine



xkcd.com/303

FLUKA

FORTRAN primer

History of Fortran

Fortran born in the early 1950s, and the first compiler was released in 1957

Standards:

- Fortran 66 – The first standard
- Fortran 77 – Extension on Fortran 66

- Fortran 90 – Dynamic memory allocation
- Fortran 95 – High performance Fortran specification

} Introduction of the *Free* format

- Fortran 2003 – Object oriented programming
- Fortran 2008 / 2018 – Extensions of Fortran 2003

} “Modern” Fortran

File format

- Fortran 77 uses the *Fixed* file format (extensions: **.f** or **.for**):
  - Maximum 72 characters in one line
  - First 6 are reserved for special function:
    - If the first character is ‘c’, ‘C’ or ‘*’, then the line is a comment
    - The 1st – 5th characters can be used for statement labels
    - If the 6th position is not empty, then the line is treated as a continuation of the previous one (Often the ‘&’ character is used)

```fortran
*...5....0....5....0....5
      program hello
c This is a comment
         print  *, 'Hello,
     & World!'
      end program hello
```

File format

- Fortran 90 introduced the *Free* format (extensions: **.f90**, [**.f95**, etc.]):
  - Code can start at the 1st column
  - Inline comments with ‘!’
  - Continuation lines

```
program hello
   print *, 'Hello,&
      & World!' ! This is a comment
end program hello
```

- *Note*: It is not possible to mix both formats in a single source file.
  The compiler expects the “correct” format based on the file extension.

Variable and procedure names

- Fortran 77:
  - Limited to 6 alphanumerical characters
  - Have to start with a letter
  - Case insensitive

- Starting with Fortran 90:
  - Can be up to 31 character long
  - Can contain letters, numbers and underscore (‘_’)
  - Have to start with a letter
  - Case insensitive

- *Note:* Try to use descriptive names, to make code readable

Variable declaration

- Fortran by default uses *implicit declaration*, which means the type of the variable (`integer`, `real`, etc.) is determined by a preset rule.
- The default rule is:
  - If the variable starts with the letter ‘I’, ‘J’, ‘K’, ‘L’, ‘M’, or ‘N’ it is an `integer`
  - Otherwise, it is a `real` (single precision float)

- It is possible (and necessary) to overwrite this with *explicit declaration*, where you manually specify another variable type, like:

```
double precision :: my_number
logical :: my_flag
```

Issues with implicit declaration

- Typos remain hidden
  - If you have a typo in a variable name, the compiler won’t raise an error
  - It is a different, but valid variable usually without a value
  - Using it in calculations will lead to unexpected results
- Unexpected type conversion
  - For example: Information is lost if you want to assign a `double precision` number to `integer` variable
- Solution
  - Force explicit declaration with the statement:

```
implicit none
```

Comparison of Fortran 77 and 90+

	Fortran 77		Fortran 90+
Format	Fixed (.f, .for)	$\longrightarrow$	Free (.f90, .f95, …)
Maximum line length	72		132
Variable name max. length	6		31
Variable declaration (usually)	implicit ★	★ $\longrightarrow$	forced explicit

- FLUKA user routines are somewhere in-between
  - ★ Implicit declaration using `double precision` numbers instead of `real`s

- Modernization effort for a future release
  - ★ A new version of the source routine is already available (fixed format, forced explicit declaration)

Variables

- Declaration:

```
integer :: amount, counter
real :: pi, sqrt_two
double precision :: energy
complex :: frequency
character :: initial
logical :: okay
```

- Assignment:

```
amount = 10
pi = 0.3141592e1
energy = 1.0d-3
frequency = (1.0, -0.5)
initial = 'F'   ! Or "F"
okay = .true.   ! Or .false.
```

Arrays and strings

- Arrays:

```
! 1D integer array
integer, dimension(10) :: array1

! An equivalent array declaration
integer :: array2(10)

! 2D real array
real, dimension(10, 10) :: array3

! Custom lower and upper
! index bounds
real :: array4(0:9)
real :: array5(-5:5)
```

- Strings:

```
character(len=10) :: string1

! Or
character(10) :: string2

string2 = 'FLUKA'
```

Note: Strings are padded with “space” to the specified length, i.e. `'FLUKA     '`.

To omit the padding use the `trim()` function

Logical operators

- Relational operators:

Equal:		
	`a .eq. b`	`a == b`
Not equal:		
	`a .ne. b`	`a /= b`
Greater than:		
	`a .gt. b`	`a > b`
Less than:		
	`a .lt. b`	`a < b`
Greater than or equal:		
	`a .ge. b`	`a >= b`
Less than or equal:		
	`a .le. b`	`a <= b`

- Logical operators:

`.true.` if both operands are `.true.`:

```
a .and. b
```

`.true.` if one of operands is `.true.`:

```
a .or. b
```

`.true.` if the operand is .false.:

```
.not. a
```

`.true.` if the operands are the same:

```
a .eqv. b
```

`.true.` if the operands are the opposite:

```
a .neqv. b
```

Conditional (if) and loop (do) constructs

- Conditional (if) construct:

```
if (angle < 90.0) then
   print *, ‘Angle is acute‘
else if (angle > 180.0) then
   print *, ‘Angle is reflex‘
else
   print *, ‘Angle is obtuse‘
end if
```

- Loop (do) construct:

```
integer :: i

do i = 1, 10
   print *, i
end do
```

- Conditional loop (do while):

```
i = 1
do while (i < 11)
   print *, i
   i = i + 1
end do
```

- Loop with skip:

```
do i = 1, 10, 2
   ! Print only odd numbers
   print *, i
end do
```

Procedures

- Functions:

  Invoked within an expression or assignment

  Returns a value

```
integer function cube(i)
   integer :: i

   cube = i**3
end function cube
```

```
program main
   integer :: cube
   integer :: i, j

   i = 3
   j = cube(i)
end program main
```

- Subroutines:

  Invoked by a `call` statement

  No return value

```
subroutine print_mx(n, m, A)
   integer :: n, m
   integer :: i
   real :: A(n, m)

   do i = 1, n
      print *, A(i, 1:m)
   end do
end subroutine print_mx
```

```
real :: mat(3, 4)
...
call print_mx(3, 4, mat)
```

Passing arguments to procedures

- Many programming languages by default only pass the values of the arguments to the procedures.
  Meaning, changing the value in the procedure doesn’t have any effect on the value of the original argument.

- However in Fortran, the arguments by themselves are passed to the procedures. This means, the changes made to the values of the arguments will remain after the procedure completes.
- Useful when more than one value must be returned.

- *Safe practice:* Only use functions which don’t change the arguments. Otherwise use subroutines.

Save statement

- Variables declared with the **save** statement retain their value between calls to procedures

```
integer, save :: amount
real, dimension(10), save :: array
```

- This allows to create sections of code which only executed at the first call

```
logical, save :: lfirst = .true.
integer, save :: reg_number
integer :: ierr

if (lfirst) then
   call geon2r('TARGET  ', reg_number, ierr)
   lfirst = .false.
end if
```

Opening files

- To open a file in Fortran:

```
open(unit=<unit>, file='<filename>', status='<status>', form='<form>')
```

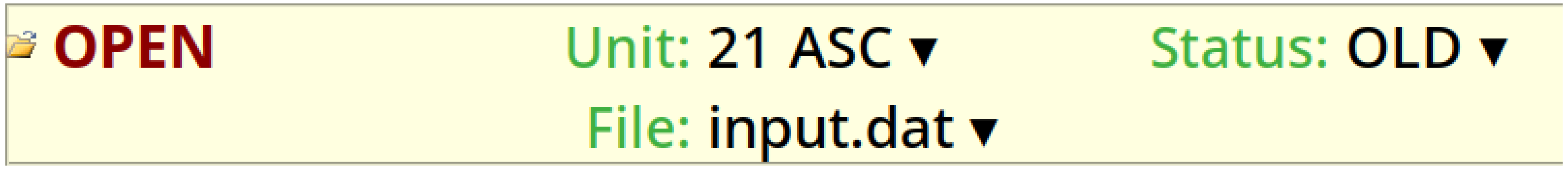
Unit number: used to reference the file in the read/write comments

  - Some units numbers are predefined
  - *FLUKA specific:* Unit numbers ≤ 20 and the ones in scorings can't be used

- FLUKA subroutine: Looks for the file in multiple directories

```
call oauxfi('<filename>', <unit>, '<form_and_status>', <ierr>)
```

- FLUKA `OPEN` card:



OPEN Unit: 21 ASC ▾ Status: OLD ▾
File: input.dat ▾

Input from files

- Reading from a file:

```
read(<source>, 'format') a, b, …
```

Source: Unit number or a string

Format: Use the default `*`. Fortran will try to figure it out based on the type of the variables

```
real, dimension(20) :: a, b
integer :: i

open(unit=21, file='input.dat', status='old', form='formatted')

do i = 1, 20
   read(21, *) a(i), b(i)
end do
```

Output to files

- Writing to a file:

```
write(<target>, ‘format‘) a, b, …
```

Target: Unit number or string
Format: The default is `*` for automatic formatting

```
integer :: i

open(unit=22, file=‘output.txt‘, status=‘new‘, form=‘formatted‘)

do i = 1, 10
   write(22, *) i, cube(i)
end do
```

Predefined units for writing to the FLUKA output files:

.out file:
```
write(lunout, *) a, …
```

.err file:
```
write(lunerr, *) a, …
```

.log file:
```
write(*, *) a, …
```

I/O formatting

- The format string lists the format specifiers for the printed variables and it is enclosed in round brackets:

  `'(A10, 5X, I4, /, F8.3, E15.7)'`

- Integer:

  `'(Iw)'`

  w characters long

- Real:

  `'(Fw.d)'`

  w characters long,
  fractional part **d** characters

  `'(Ew.d)'`

  Exponential form, **w** characters long,
  fractional part **d** characters

- String:

  `'(Aw)'`

  w characters long

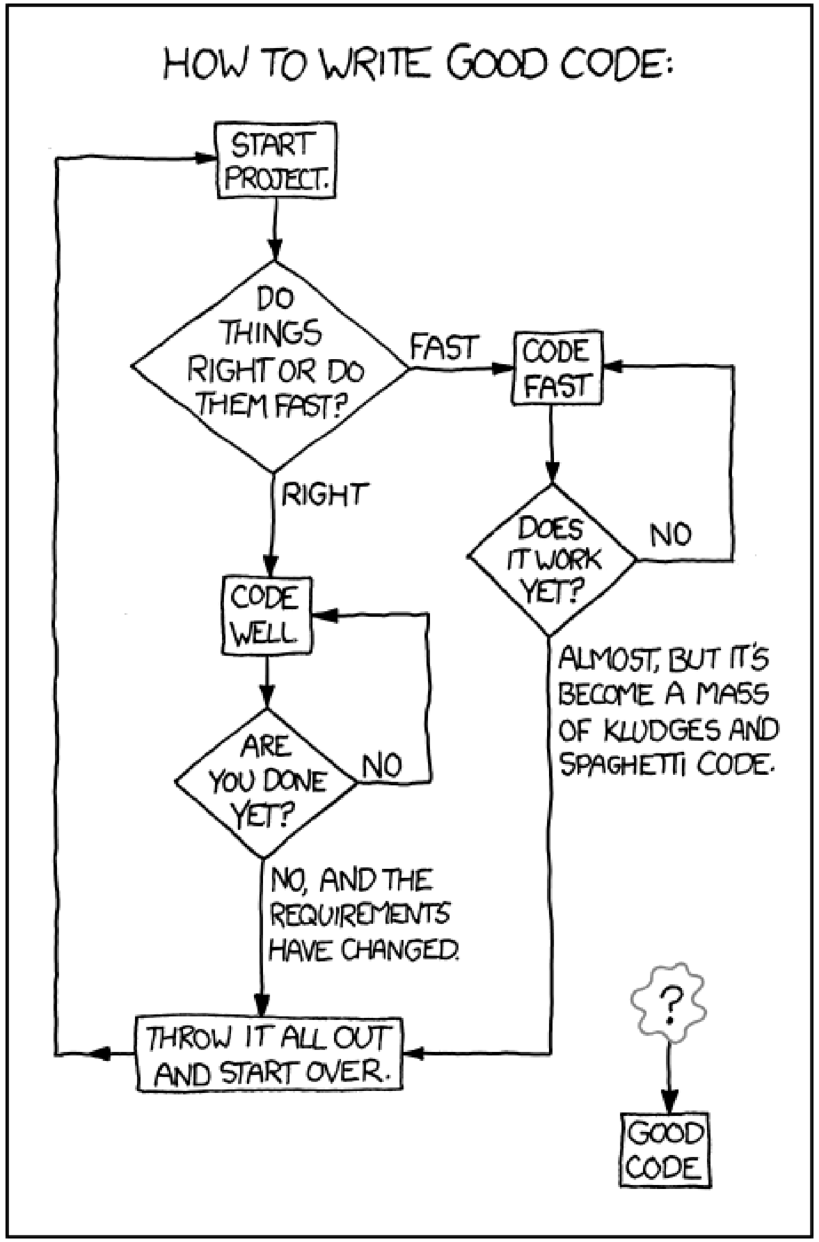
- Blank space:

  `'(nX)'`

  n characters long

- New line:

  `'(/)'`


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
DOES IT WORK YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
RIGHT
CODE WELL.
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE

xkcd.com/844

FLUKA