**FORTRAN**

**Author:**  John Backus

**History**

Fortran programming language designed by John Backus and his team in IBM. Name is an abbreviation for FORmula TRANslation. The language was designed to program IBM 704. Before Fortran IBM computers were programmed with assembly language. Instead, John Backus proposed such a language that is powerful and easy to code compared to assembly to his supervisors in 1950. Some of the Fortran developers were chess players. Here are the names of all developers; Richard Goldberg, Sheldon F. Best, Harlan Herrick, Peter Sheridan, Roy Nutt, Robert Nelson, Irving Ziller, Harold Stern, Lois Haibt, and David Sayre. After six years of development phase, in 1956 first user manual is published.

At the beginnings community of programmers did not accept this new high-level language because of compiler performance worries. But in Fortran there were less code lines than assembly language by factor of 20. This gave a freedom to compiler designer to design more efficient and faster compilers.

Fortran became widely used in engineering applications because the power of handling complex equations. Computer producers began to produce Fortran appendaged computers. There were over 40 compilers in 1963 due to vogue of Fortran. And Fortran became a cross-platform programming language.

**Why this language was invented?**

Before Fortran computers (IBM) were programmed in assembly language. And it was a very hard and time-consuming way to programming devices. Programmer needs to write thousands of code-lines. To have ease of coding and faster coding compared to assembly language John Backus came up with Fortran. With this new programming language, complicated mathematic formulas and operation on engineering applications are became easier to code and faster compilation by computers.

**When/why shall we use it?**

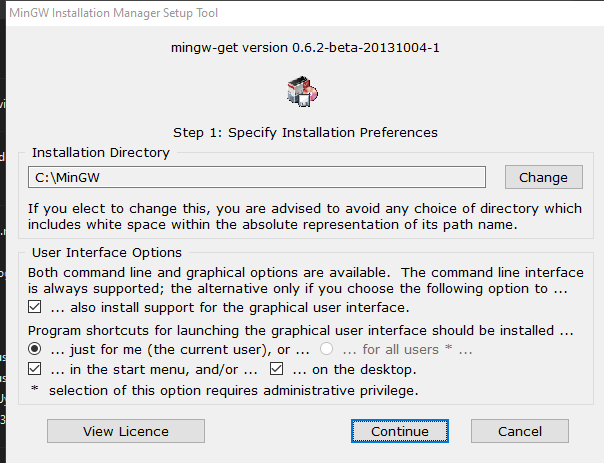
Fortran is still in use of scientific numerical programming. Because of its array handling and performance is better compared to matlab or python’s numpy library. And it is very hard to inadvertently code slow. Another benefit is there are tons of repositories about Fortran so it’s easy to get a solution if there is a problem or bug etc. And Fortran also supports object-oriented programming and parallel programing. In conclusion if we are dealing with scientific programming, Fortran can be used.

**How to setup Fortran;** *an environment to use it in different platforms (windows, mac, linux)*

**In Windows:**

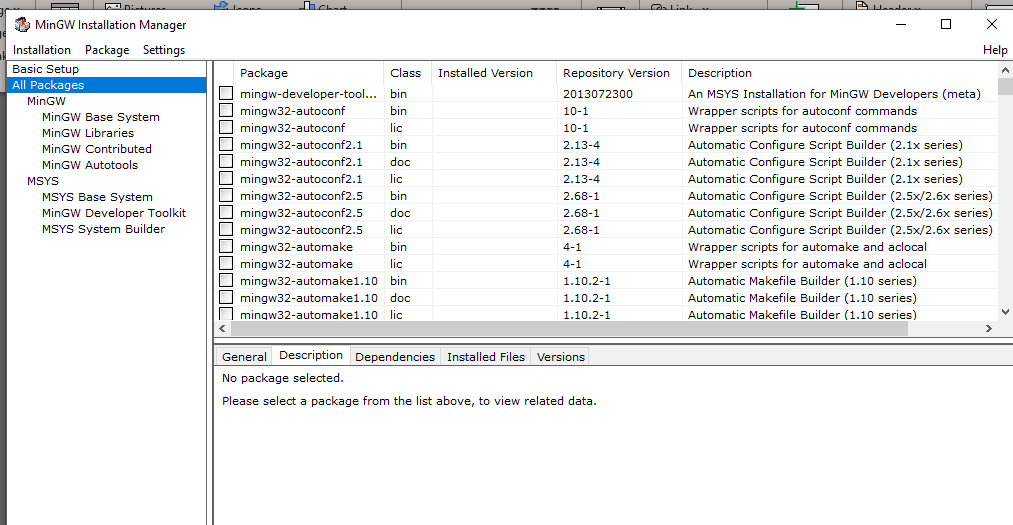
**Step 1:** Go to <https://sourceforge.net/projects/mingw/> to download MinGW.

**Step 2**: Double click to downloaded file. You can see



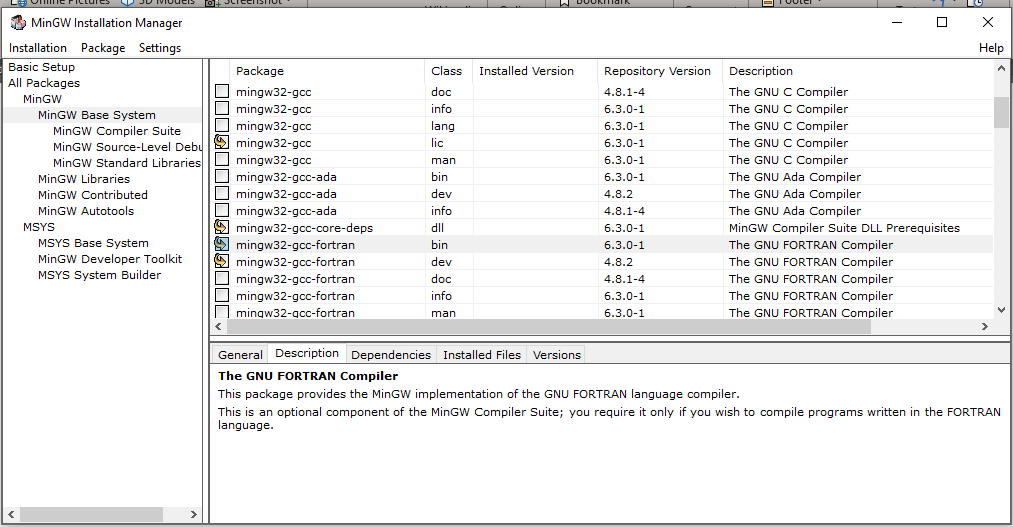
It is better to setup on recommended directory to not encounter further errors. Then click the continue button, wait for a while.

**Step 3:** When it is done a window is going to pop-up as below.

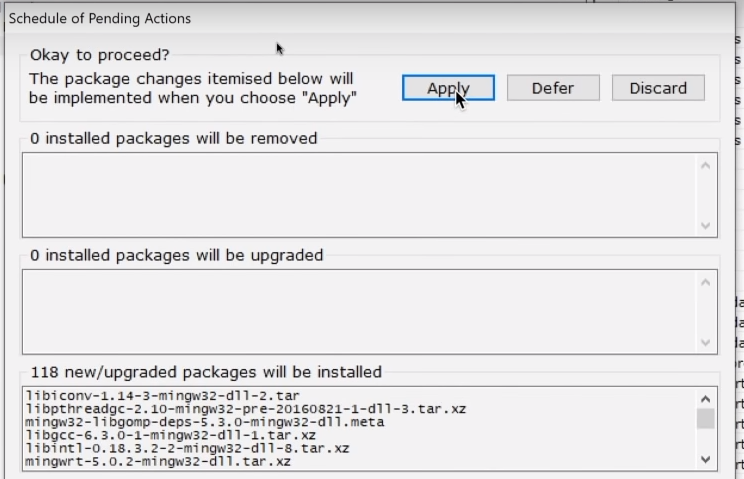


Select mingw-developer-toolkit by left-click then it is automatically selected all the needed ones. Select mingw-base.

**Step 4:** Go to left panel and open MinGW Base System. Select mingw-gcc-fortran

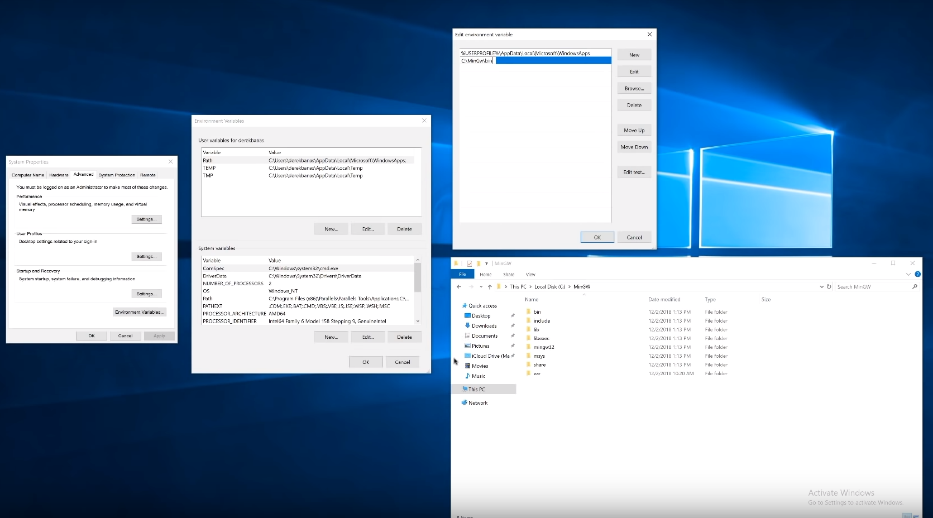


**Step 5:** On the top of the window click on install and press apply changes. Then click on “Appl” on the pop- up

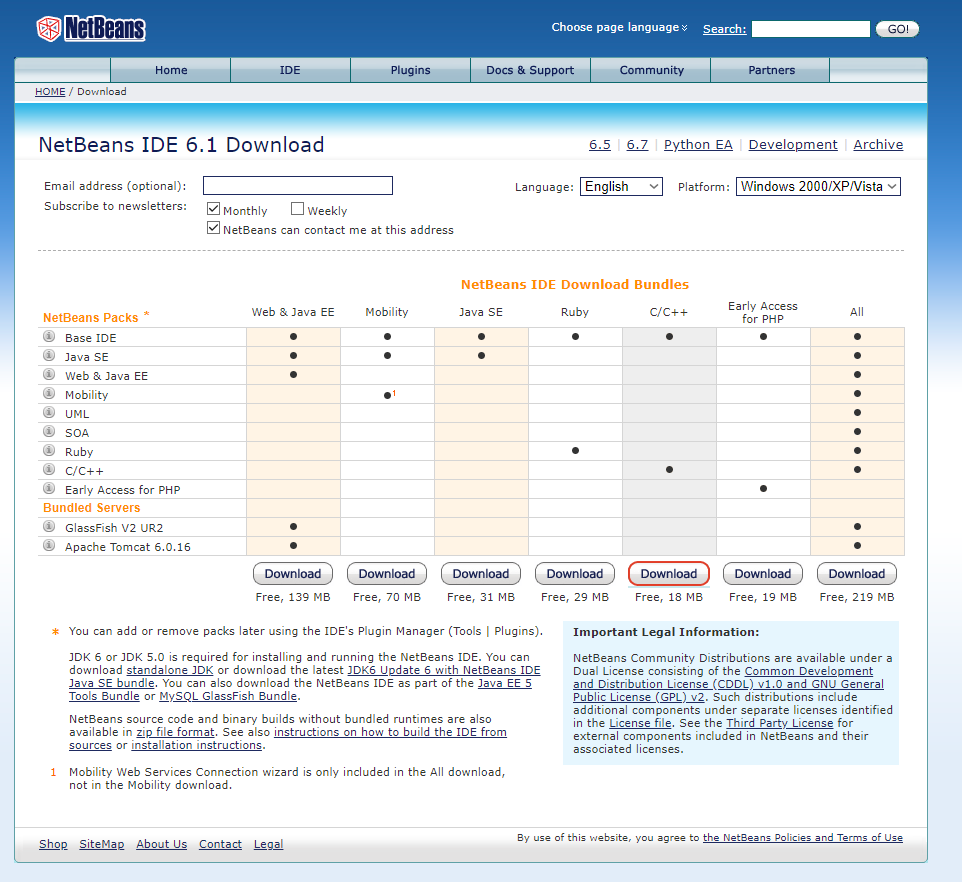


**Step 6:** Go to environment variables. On the user variables, click “Path” and click “edit” and click “New”.

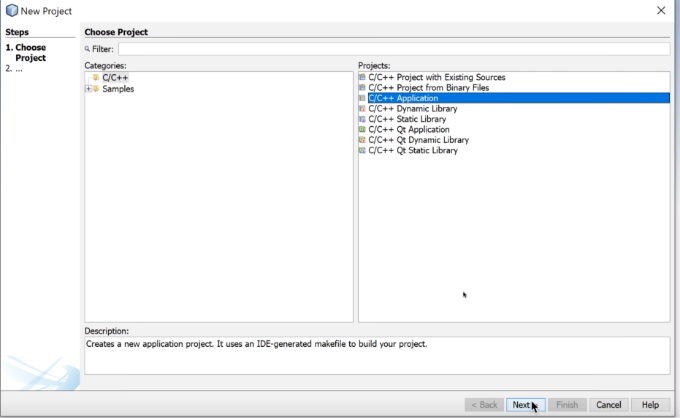
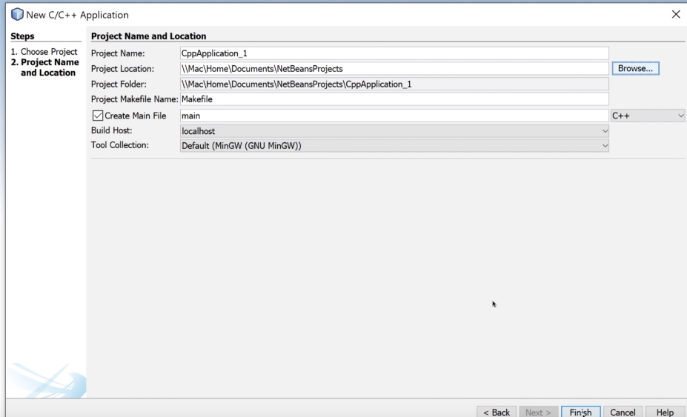
Type “C:\MinGW\bin” click on “OK”



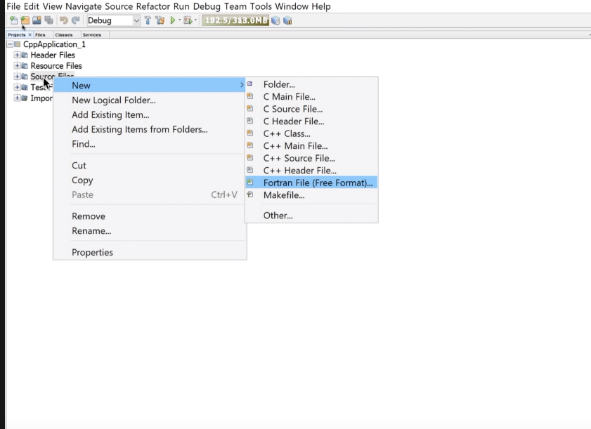
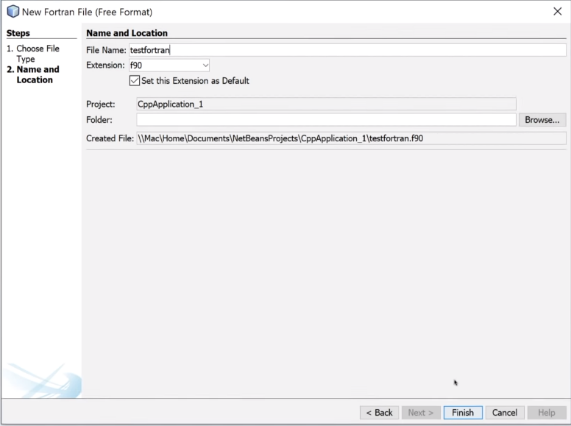
**Step 7:** Download NetBeans IDE for C/C++ and install it.



**Step 8:** Starta new project on NetBeans IDE. Select C/C++ Application. Click next, on the next page leave everything as suggested and click Finish.

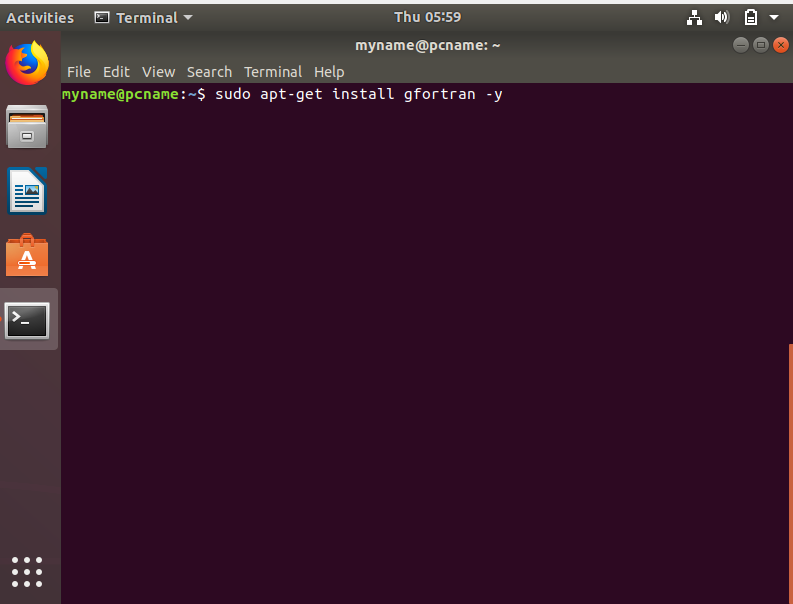
 

**Step 9:** Right click on Source files, select Fortran File under New. Give a name as whatever you want, click on finish. Then you are ready to code in Fortran.

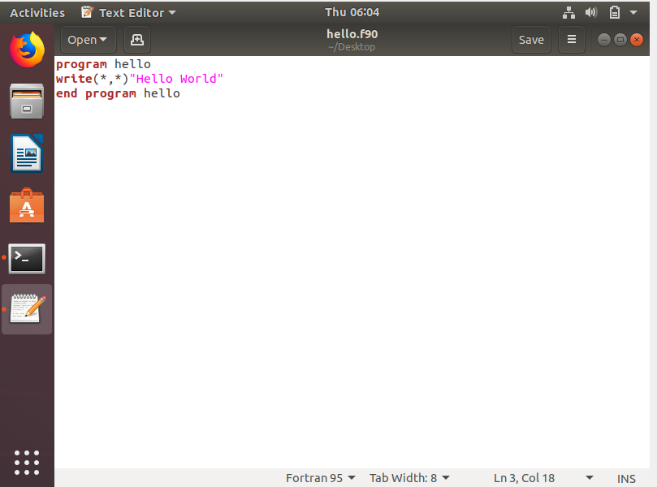
 

**In Linux:**

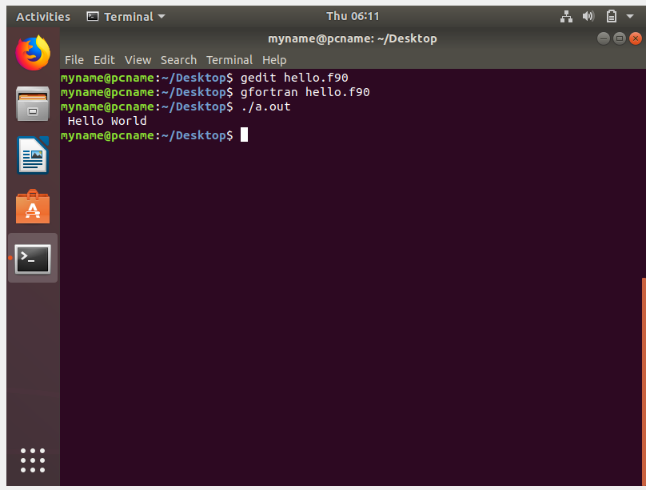
**Step 1:** Open terminal and write “sudo apt-get install gfortran -y” then enter your password.



**Step 2:** Open a text editor named “hello.f90 ” (before dot you can give any name) and write as below.

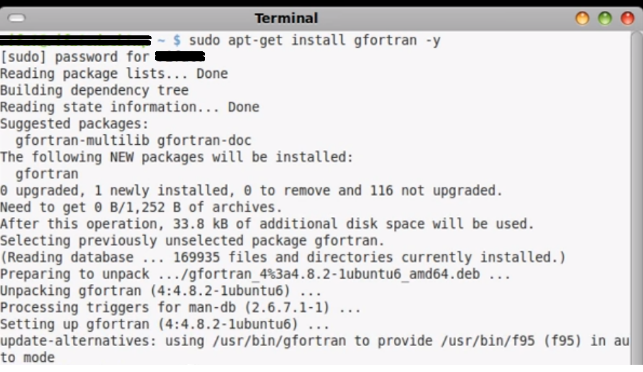


**Step 3:** To compile your program type “gfortran hello.f90 ” in terminal. And type “./a.out” to run the program.

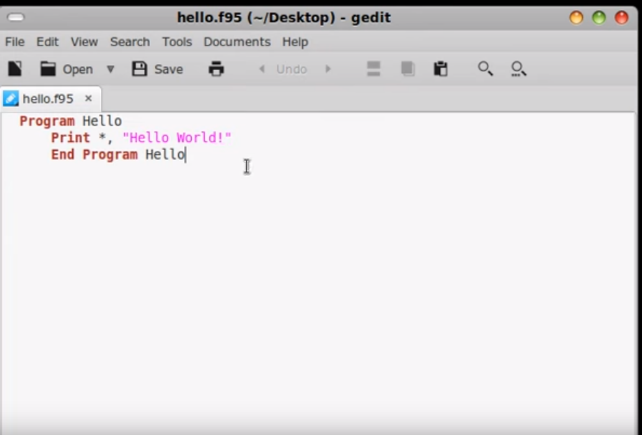


**On Mac:**

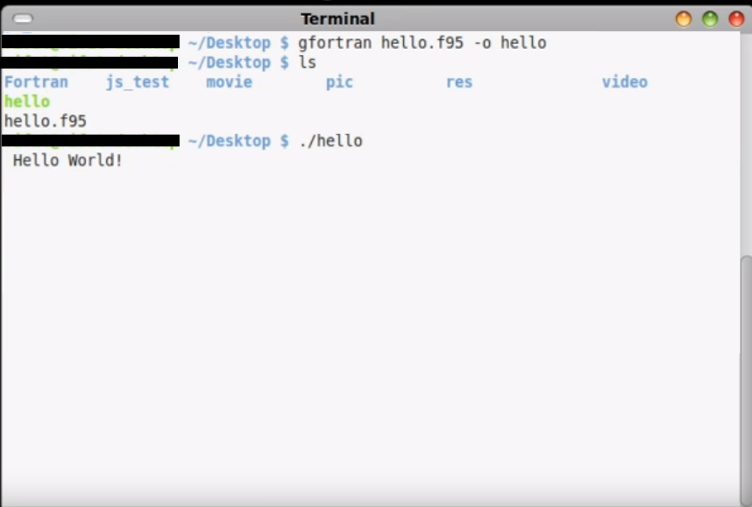
**Step 1:** Open terminal and write “sudo apt-get install gfortran -y” then enter your password.



**Step 2:** Open a text editor named “hello.f95 ” (before dot you can give any name) and write as below.



**Step 3:** To compile your program type “gfortran hello.f95 -o hello ” in terminal. And type “./hello” to run the program.



**Example codes**

**Example 1:**

**program** hello

**print** **\*,** **"Hello World!"**

**end** **program** hello

**Example 2: Greatest Common Divisor in Fortran 77**

**PROGRAM** EUCLID

**PRINT** **\*,** 'A?'

**READ** **\*,** NA

**IF** **(**NA.LE.0**)** **THEN**

**PRINT** **\*,** 'A must be a positive integer.'

**STOP**

**END** **IF**

**PRINT** **\*,** 'B?'

**READ** **\*,** NB

**IF** **(**NB.LE.0**)** **THEN**

**PRINT** **\*,** 'B must be a positive integer.'

**STOP**

**END** **IF**

**PRINT** **\*,** 'The GCD of'**,** NA**,** ' and'**,** NB**,** ' is'**,** NGCD**(**NA**,** NB**),** '.'

**STOP**

**END**

**FUNCTION** NGCD**(**NA**,** NB**)**

IA **=** NA

IB **=** NB

**1** **IF** **(**IB.NE.0**)** **THEN**

ITEMP **=** IA

IA **=** IB

IB **=** **MOD(**ITEMP**,** IB**)**

**GOTO** 1

**END** **IF**

NGCD **=** IA

**RETURN**

**END**

**Example 3: Object Oriented Programming**

**module** test\_module

**implicit** **none**

**private**

**integer,** **public** **::** a**=**1

**integer,** **public,** **protected** **::** b**=**1

**integer,** **private** **::** c**=**1

**end** **module** test\_module

!> import all public data of test\_module

**program** main

**use** test\_module

**print** **\*,** a**,** b

**end** **program** main

!> import all data, and rename

**program** main

**use** test\_module**,** better\_name **=>** a

! new name use available

**print** **\*,** better\_name

! old name is not available anymore

!print \*, a <- ERROR

**end** **program** main

!> import only a subset of the public data

**program** main

**use** test\_module**,** **only** **:** a

! only a is loaded

**print** **\*,** a

! b is not loaded

!print \*, b <- ERROR

**end** **program** main

**Example 4: Parallel Programming Example in Fortran90**

**use** omp\_lib

**implicit** **none**

**integer** **(** **kind** **=** 4 **),** **parameter** **::** nv **=** 6

**integer** **(** **kind** **=** 4 **)** i

**integer** **(** **kind** **=** 4 **)** **::** i4\_huge **=** 2147483647

**integer** **(** **kind** **=** 4 **)** j

**integer** **(** **kind** **=** 4 **)** mind**(**nv**)**

**integer** **(** **kind** **=** 4 **)** ohd**(**nv**,**nv**)**

**call** timestamp **(** **)**

**write** **(** **\*,** '(a)' **)** ' '

**write** **(** **\*,** '(a)' **)** 'DIJKSTRA\_OPENMP:'

**write** **(** **\*,** '(a)' **)** ' FORTRAN90 version'

**write** **(** **\*,** '(a)' **)** ' Use Dijkstra''s algorithm to determine the minimum'

**write** **(** **\*,** '(a)' **)** ' distance from node 1 to each node in a graph,'

**write** **(** **\*,** '(a)' **)** ' given the distances between each pair of nodes.'

**write** **(** **\*,** '(a)' **)** ' '

**write** **(** **\*,** '(a)' **)** ' Although a very small example is considered, we'

**write** **(** **\*,** '(a)' **)** ' demonstrate the use of OpenMP directives for'

**write** **(** **\*,** '(a)' **)** ' parallel execution.'

!

! Initialize the problem data.

!

**call** init **(** nv**,** ohd **)**

!

! Print the distance matrix.

!

**write** **(** **\*,** '(a)' **)** ' '

**write** **(** **\*,** '(a)' **)** ' Distance matrix:'

**write** **(** **\*,** '(a)' **)** ' '

**do** i **=** 1**,** nv

**do** j **=** 1**,** nv

**if** **(** ohd**(**i**,**j**)** **==** i4\_huge **)** **then**

**write** **(** **\*,** '(2x,a)'**,** **advance** **=** 'NO' **)** 'Inf'

**else**

**write** **(** **\*,** '(2x,i3)'**,** **advance** **=** 'NO' **)** ohd**(**i**,**j**)**

**end** **if**

**end** **do**

**write** **(** **\*,** '(a)'**,** **advance** **=** 'yes' **)**

**end** **do**

!

! Carry out the algorithm.

!

**call** dijkstra\_distance **(** nv**,** ohd**,** mind **)**

!

! Print the results.

!

**write** **(** **\*,** '(a)' **)** ' '

**write** **(** **\*,** '(a)' **)** ' Minimum distances from node 1:'

**write** **(** **\*,** '(a)' **)** ' '

**do** i **=** 1**,** nv

**write** **(** **\*,** '(2x,i2,2x,i2)' **)** i**,** mind**(**i**)**

**end** **do**

!

! Terminate.

!

**write** **(** **\*,** '(a)' **)** ' '

**write** **(** **\*,** '(a)' **)** 'DIJKSTRA\_OPENMP:'

**write** **(** **\*,** '(a)' **)** ' Normal end of execution.'

**write** **(** **\*,** '(a)' **)** ' '

**call** timestamp **(** **)**

**stop**

**end**

**Example 5: Scientific Programming**

! ProjectileAir . f90 : Projectile Program using Dislin

! −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

**Subroutine** buildPlot **(** Npts **,** x0 **,** y0 **,** v0 **,** theta **,** k**)**

**Use** d i s l i n

**Implicit** **None**

**Integer** **,** **intent** **(** **in** **)** **::** Npts

**Real** ∗8 **,** i n t e n t **(** **in** **)** **:** **:** x0 **,** y0 **,** v0 **,** theta **,** k

**Real** ∗8 **:** **:** dt **,** g **,** vx **,** vy **,** xmax **,** ymax **,** t

**Real** ∗8**,** **dimension** **(** Npts **)** **::** aXValues **,** aYValues **,** nXValues **,**

nYValues

**Integer** **::** i

t **=** 0.

g **=** 9.81

dt **=** 2∗v0∗ **sin** **(** theta **)** **/(** Npts ∗g **)**

vx **=** v0∗ **cos** **(** theta **)**

vy **=** v0∗ **sin** **(** theta **)**

nXValues **(**1**)** **=** x0

nYValues **(**1**)** **=** y0

**do** i **=**1**,**Npts **,**1

t **=** **(**i −1**)**∗ d t

! Analytic

aXValues **(** i **)** **=** x0**+**v0∗ **cos** **(** theta **)** ∗ t

aYValues **(** i **)** **=** y0**+**v0∗ **sin** **(** theta **)** ∗ t−g∗ t ∗ t **/**2.

i f **(** i **>=**2**)** **then**

vx **=** vx−k∗vx∗ d t

vy **=** vy−g∗ dt−k∗vy∗ d t

! Numberic

nXValues **(** i **)** **=** nXValues **(** i −1**)** **+** vx∗ d t

nYValues **(** i **)** **=** nYValues **(** i −1**)** **+** vy∗ d t

**end** i f

**end** **do**

! Dislin plotting routines

**call** metafl **(**’XWIN’**)**

**call** disini

**call** **name** **(**’x**-**axis’ **,**’X’**)**

**call** **name** **(**’y**-**axis’ **,**’Y’**)**

**call** labdig**(**−1 **,**’X’**)**

**call** ticks **(**10 **,**’XY’**)**

**call** titlin **(**’ Analytic **(**green**)** vs Numerical’ **,**1**)**

xmax **=** 2. ∗ v0 ∗∗2∗ **cos** **(** theta **)** ∗ **sin** **(** theta **)** **/**g

ymax **=** **(** v0∗ **sin** **(** theta **))** ∗ ∗ 2**/(**2. ∗ g **)**

c a l l graf **(** x0 **,** x0**+**xmax **,** x0 **,** xmax **/** 1 0 **.** **,** y0 **,** y0**+**ymax **,** y0 **,** ymax **/**10.**)**

**call** title **()**

**call** color **(**’RED’**)**

**call** curve **(** aXValues **,** aYValues **,** Npts **)**

**call** color **(**’GREEN’**)**

**call** curve **(** nXValues **,** nYValues **,** Npts **)**

**call** color **(**’FORE’**)**

**call** dash

**call** disfin

**End** **Subroutine** b u i l d P l o t

!

**Program** projectileAir

**Implicit** **None**

! Set−up a l l needed parameters

**Integer** **:** **:** Npts **=** 200

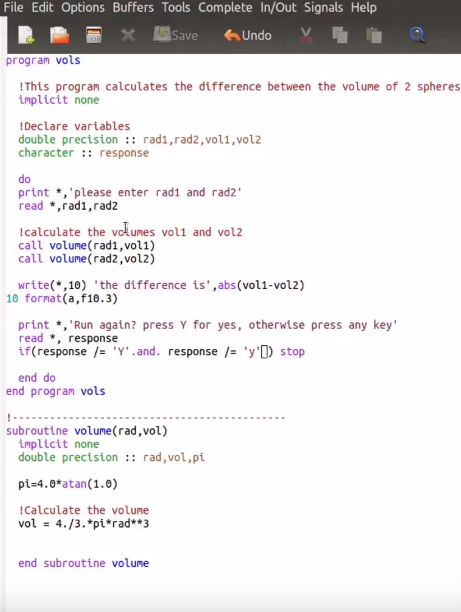
**Real** ∗8 **:** **:** x0 **=**0. **,** y0 **=**0. **,** v0 **=**20. **,** theta **=**3.14159265358979**/**4. **,** k **=**0.25

**call** buildPlot **(** Npts **,** x0 **,** y0 **,** v0 **,** theta **,** k**)**

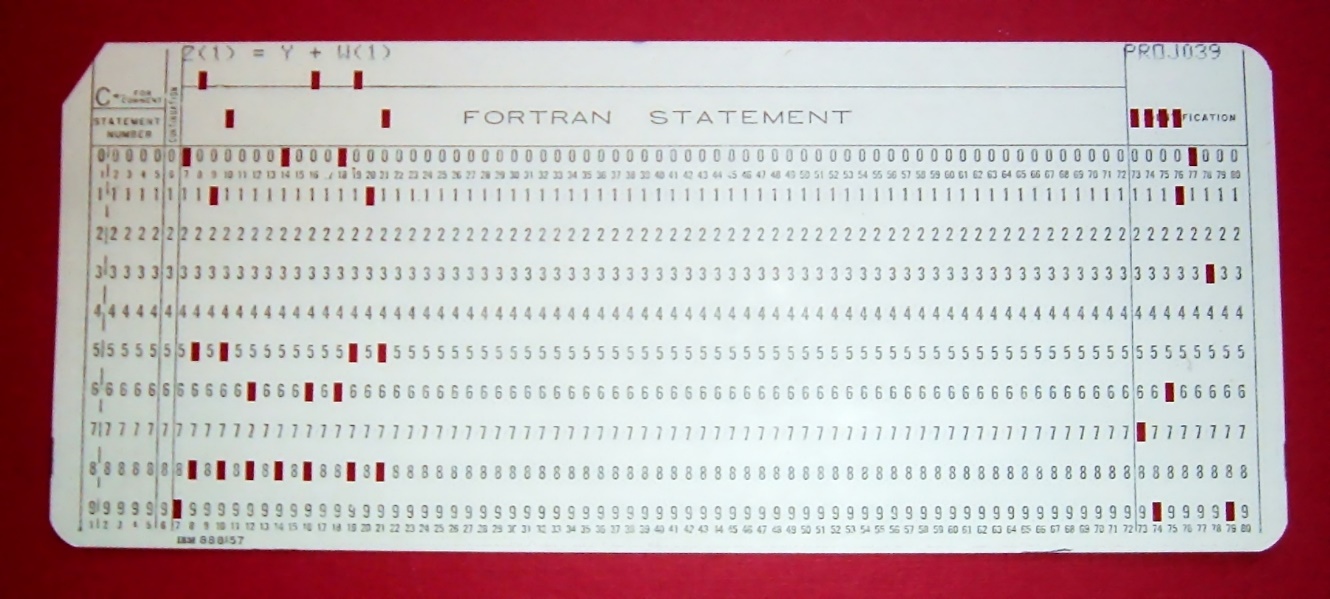
**End** **Program** p r o j e c t i l e A i r

**Specific Things About Fortran**

**1.Subroutines:** Subroutinesare very useful to compute repetitive things. Subroutines are defined outside of the program. Unlike functions they return more than one thing. Or they do not return a value at all, just do an operation like swapping. In the example below, there is a subroutine that calculates volume difference between two spheres until user press any other key than ‘Y’ or ‘y’.



**2. Punch cards:** Fortran was uses with punch-cards also, in its early times. Here is a punch card picture below.



**3. Types: I**n Fortran, floating point numbers stored by “Real Type”. Means that they are represented with “real” keyword. Also, complex numbers can be stored with “complex” keyword. In older version of Fortran, there was a feature called “implicit typing”. you do not have to declare the variables before use. If a variable is not declared, then the first letter of its name will determine its type. Variable names starting with i, j, k, l, m, or n, are considered to be for integer variable and others are real variables. However, you must declare all the variables as it is good programming practice. For that you start your program with the “implicit none” statement.