Introduction

Carlos Cruz

NASA GSFC Code 606 (ASTG)
Greenbelt, Maryland 20771

carlos.a.cruz@nasa.gov

October 24, 2018

Who we are?

Members of the Advanced Software Technology Group (ASTG) Code 606, NASA GSFC:

- Carlos A. Cruz (Computational Scientist)
- Jules Kouatchou (Computational Scientist)
- Bruce Van Aartsen (Senior Software Engineer)

Agenda

Day 1

- Introduction
- Variables and data types
- Conditionals and loops
- Array concepts
- Subroutines and functions
- Modules and interfaces
- File IO

Day 2

- Derived types and pointers
- Introduction to OOP
- IO Enhancements
- Inheritance
- Polymorphism
- Miscellaneous items
- Interoperability with C

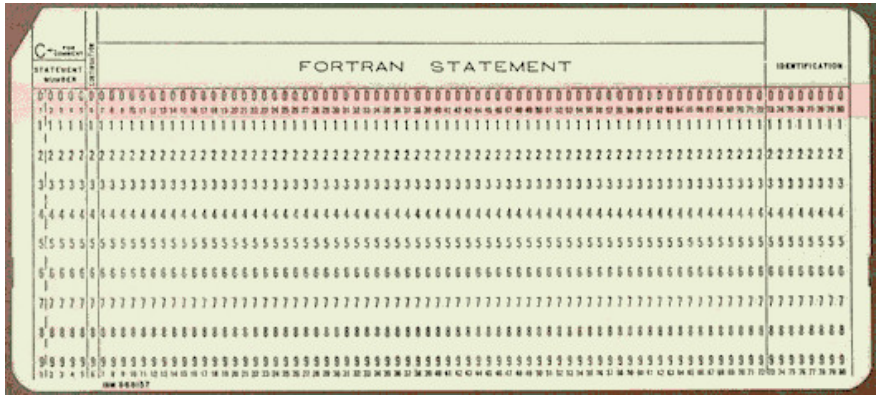
Introduction to Fortran 90
Introduction to Fortran 2003

Introduction

- History of Fortran
- Why Learn Fortran?
- Basic of Source Code
- Building fortran programs
  - Compilers
  - Makefiles

A Brief History of Fortran

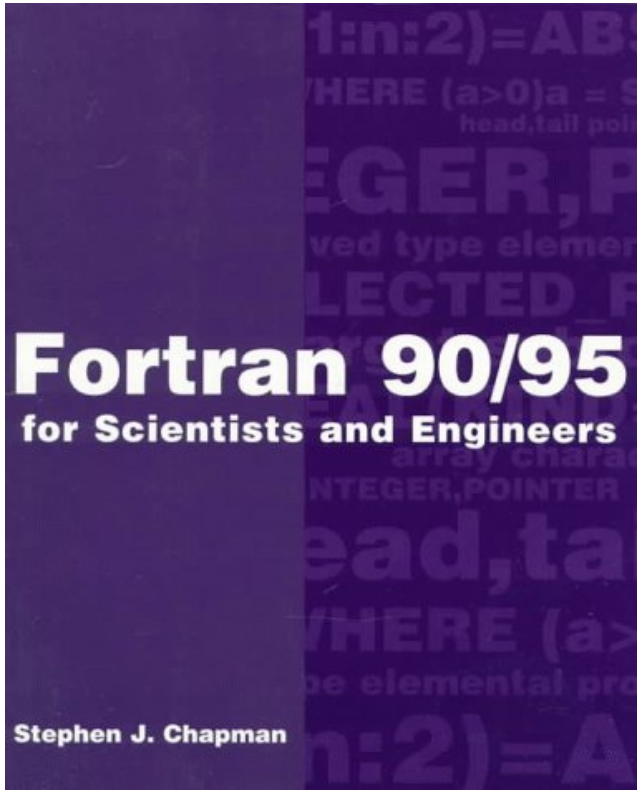
- Origins
  - Started ca 1954 by John Backus and his team at IBM
  - Name comes from **FOR**mula **TRAN**slation
  - First language standard in 1967 (Fortran 66)



- FORTRAN 77
  - New standard to overcome divergence in different implementations (1978)

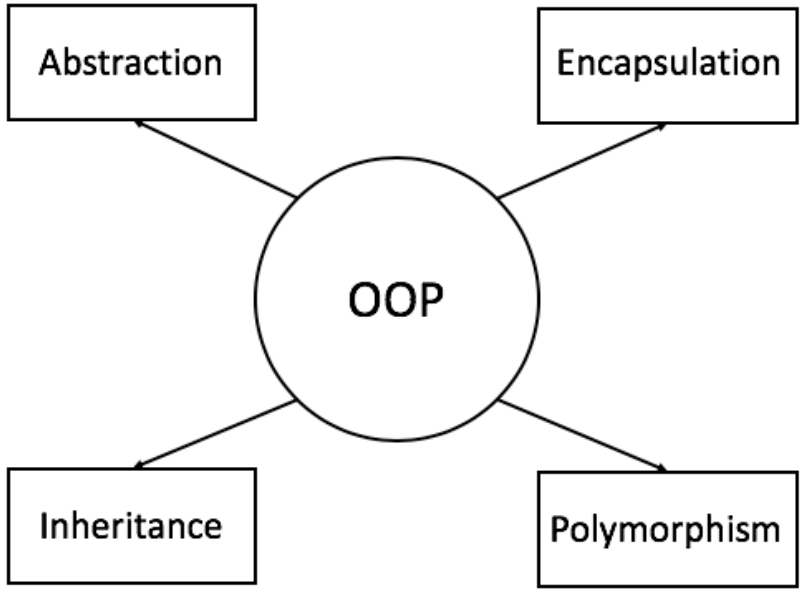
A Brief History of Fortran

- Fortran 90 (All caps were dropped)
  - Major revision. Added modules, derived data types, dynamic memory allocation
  - Retained backward compatibility
- Fortran 95
  - Minor revision. Added several HPC related features; forall, where, pure, elemental, pointers



A Brief History of Fortran

- Fortran 2003
  - Major revision with many new features including; OO capabilities, procedure pointers, IEEE arithmetic, C interoperability
- Fortran 2008 (latest *stable* release)
  - Minor revision. Added co-arrays and submodules

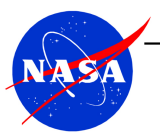


Object-Oriented Concepts using Fortran90:
http://www.cs.rpi.edu/~szymansk/OOF90/F90_Objects.html

Why Fortran?

- Fortran is the dominant language in HPC applications
  - climate/weather models
  - large scale molecular dynamics
  - electronic structure calculation codes
  - modeling of stars and galaxies
- Many dense linear algebra libraries developed in Fortran, e.g. BLAS, LAPACK, Scalapack.

Fortran continues and will continue to be a dominant language for large scale simulation of physical systems[1].



[1] Survey of Fortran users at the 2014 Supercomputing Convention, 100% of respondents said they thought they would still be using Fortran in five years

Why Fortran?

Unique benefits:

- Expressiveness / ease
  - Arrays lie at the heart of all physics/engineering calculations
  - Little to worry about pointers and memory allocation
    - Dynamic arrays in Fortran are not pointers, where in C/C++ they are pointers making them more difficult to deal with.
- Performance[2]
- Has added many modern features of programming into newer standards.
- Legacy code
- It is **not** a general purpose programming language like C, C++ and Python.
  - It is a language that has been designed exclusively for numerical computation and has applications only in computational science and engineering.

[2] Originally proposed as a practical alternative to **Assembly** language, i.e. it was designed to compete on performance with hand-coded assembler!

General Structure of Fortran Programs

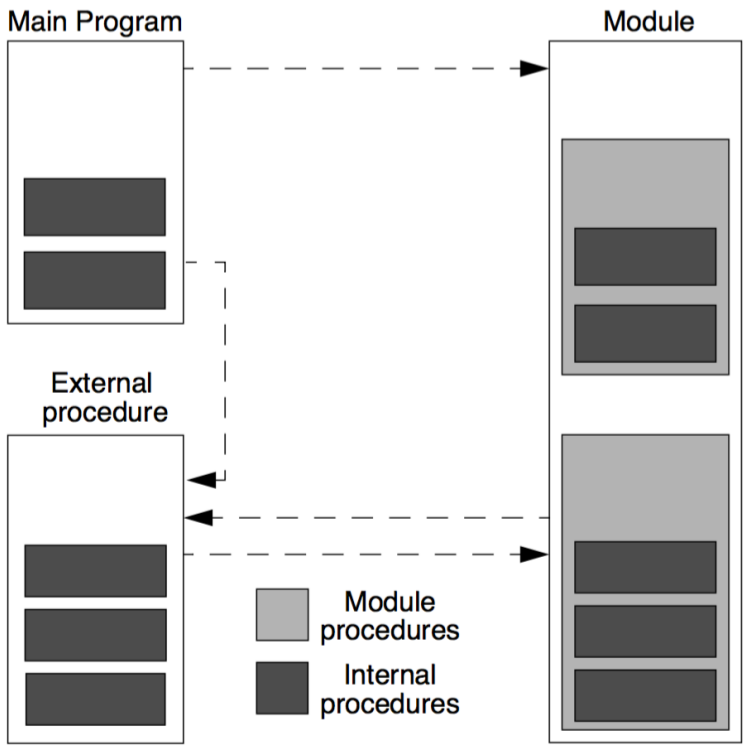
Source Code

- Fortran source code is in ASCII text that can be written in any plain-text editor
  - Use a good editor, e.g. vim or emacs - both are capable of syntax highlighting.
- Fortran source code is **case insensitive**, i.e. X is the same x
- Use uppercase letters for Fortran keywords ... or not
  - Use one convention and be consistent.

Basic Program Structure

The general structure of a Fortran **program** is as follows:

```
1   ! keyword PROGRAM
2   PROGRAM program_name
3   !  All variable MUST be
        declared
4   IMPLICIT NONE
5   ! Variable declarations
6    ...   type declarations
7   ! Main program
8    ...    Fortran statements here
9   ! subprograms
10   ...    optional subprogram
        units
11  END PROGRAM
```



The unit containing the PROGRAM attribute is often called the **main program** or **main**.

Hello World!

hello.F90:

```
1  program hello
2  ! A simple hello world program
3     print *,"Hello World!"
4  end program
```

The standard extension for Fortran source files is .f90, i.e., the source files are named <name>.f90. But:

- FORTRAN77:
  - .f ← *fixed-form* source code
- Fortran90 and later:
  - * .F ← *fixed or free form* source code that must be preprocessed
  - * .F90 ← *free form* source code that must be preprocessed

Compiling, Linking and Running Fortran Programs

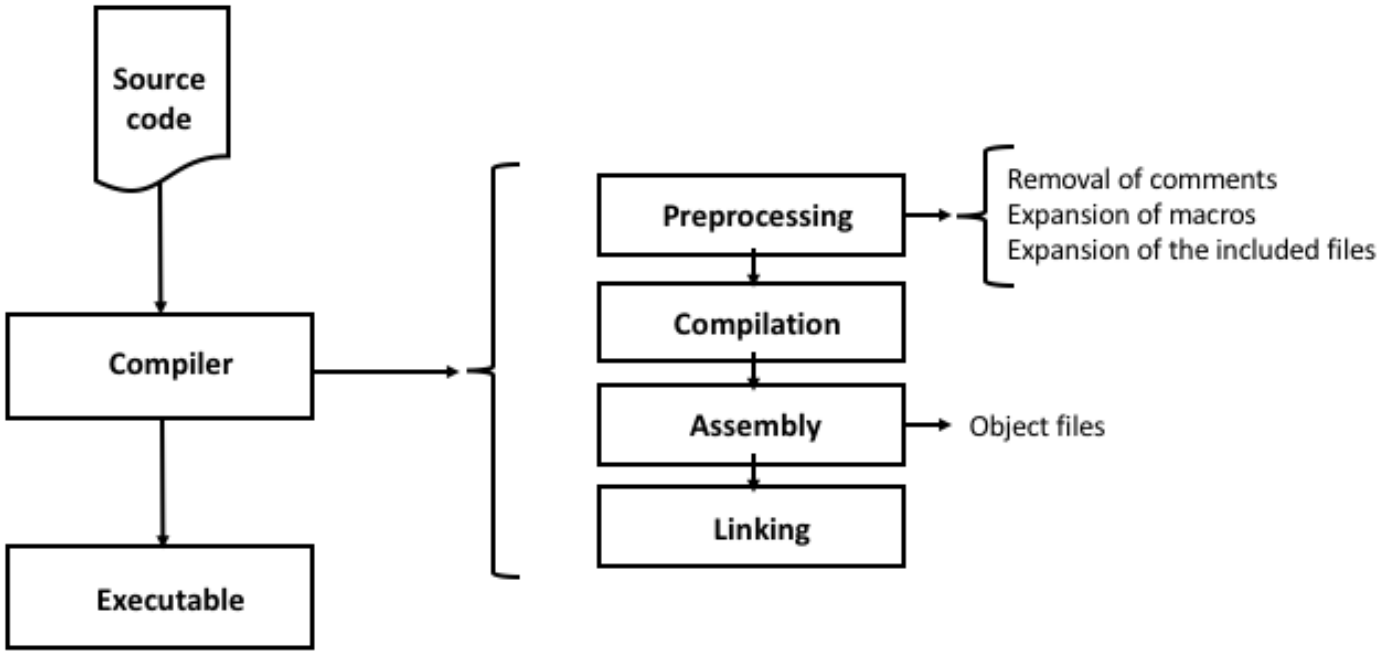
Compilers

To execute a Fortran program, you need to **compile** it to obtain an executable.

A **compiler** is a program that translates source code written in some high-level language - like Fortran - into low-level machine code. The machine code is generally specific to the computer architecture where it was compiled.

Compilation at a glance

How does a compiler translate source code into executable code?



Some Fortran compiler vendors

- GNU
- IBM
- Intel
- NAG
- PGI

`https://en.wikipedia.org/wiki/List_of_compilers#Fortran_compilers`

The GNU Fortran compiler

The GNU Fortran compiler supports the Fortran 77, 90 and 95 standards completely, parts of the Fortran 2003 and Fortran 2008 standards, and several vendor extensions.
The GNU Fortran compiler has several components:

- A version of the *gcc* command that also understands and accepts Fortran source code.
- The *gfortran* command. The difference with *gcc* is that *gfortran* will automatically link the correct libraries to your program.
- A collection of run-time libraries.
- The Fortran compiler itself, *f951*. *f951* **translates** the source code to assembler code.

GNU Fortran is a part of GCC, the *GNU Compiler Collection*.

Using the GNU compiler

Use *gfortran* command:

- gfortran hello.F90 ← no *output* name for the executable specified
  - ./a.out ← default output file name
- gfortran -c hello.F90 ← compile only - generates *object* file
  - hello.o
- gfortran -c hello.F90 -o hello.o ← same as previous case
  - hello.o
- gfortran -o hello.exe hello.F90 ← compile and link
  - ./hello.exe

Using the GNU compiler

Use *gfortran* command:

- gfortran hello.F90 ← no *output* name for the executable specified
  - ./a.out ← default output file name
- gfortran -c hello.F90 ← compile only - generates *object* file
  - hello.o
- gfortran -c hello.F90 -o hello.o ← same as previous case
  - hello.o
- gfortran -o hello.exe hello.F90 ← compile and link
  - ./hello.exe

Using the GNU compiler

Use *gfortran* command:

- gfortran hello.F90 ← no *output* name for the executable specified
  - ./a.out ← default output file name
- gfortran -c hello.F90 ← compile only - generates *object* file
  - hello.o
- gfortran -c hello.F90 -o hello.o ← same as previous case
  - hello.o
- gfortran -o hello.exe hello.F90 ← compile and link
  - ./hello.exe

Libraries

Generally, we will have many fortran files that, upon compilation, will produce many object files. It is often useful to combine the object files into a *library*. To create a *static* library we use the program"ar"(stands for archiver) as follows:

```
$ gfortran -c *.F90
$ ar -r libmystatlib.a *.o
```

one can also create a *dynamic* library...

```
$ gfortran -c -fPIC *.F90
$ gfortran -shared -o libmydynlib.so *.o
```

make

What is **make**?

- The **make** utility is a tool for managing and maintaining computer programs. It is generally used to build executable programs, libraries, and other files.
- Created at Bell Labs ca. 1977
- The input to make is a file called the **makefile**.
- The **makefile** describes a set of targets, which are the objects that can be **made**.
- **make** is programming language agnostic.
- **make** is one of the most essential tools for programmers.

Reference: `http://www.gnu.org/software/make/`

The makefile

The **makefile** is a collection of *rules*.

- The rules instruct **make** how to build a target
- A rule also specifies dependencies of the target.

The dependency rules must be executed first depending on whether that is already processed by looking at the time stamps.

Rule syntax:

target: *dependencies* ← AKA a *prerequisite*
commands
↑
Tab

The makefile

- Standard names: makefile, Makefile, GNUmakefile.
- To run:

```
$ make
$ make some_target
```

- If using a non-standard name, say my_app.make

```
$ make -f my_app.make
```

The makefile

```
hello.exe: hello.o
        gfortran hello.o -o hello.exe

hello.o: hello.F90
        gfortran -c hello.F90
clean:
        rm *.o *.exe
```

```
$ make  (or make hello.exe)
gfortran -o hello.exe hello.F90 -I.
$ ls
GNUmakefile  hello.exe  hello.F90
$ ./hello.exe
 hello world!
```

Conclusion

References:

- `https://gcc.gnu.org/fortran/`
- `http://fortranwiki.org/fortran/show/HomePage`
- `https://en.wikipedia.org/wiki/Fortran`
- `https://www.gnu.org/software/make/manual/make.html`