Historic Programming Languages

Sandro Lobbene

Toygun Ejder

Tufan Alacapunar

03.05.2023

**Abstract**

In today’s world, there is an overwhelming variety of programming languages, which leads to learning a modernized language. However, understanding how programming languages evolved in the past decades, will lay a great foundation to comprehend certain concepts, which are still relevant to this day in the field of computer science. This allows us to gain insights into how to improve coding in the future. Fortran, Lisp and ALGOL 60 all contributed indispensable features, even though they had limitations.

**1. Fortran**

Fortran is a procedural programming language, which influenced many other programming languages. It was developed by IBM in the 1950s for scientific and engineering applications [1]. Fortran is a general-purpose, imperative programming language and plays a big role in numeric computation as well as scientific computing. It is known for high-performance computing and is used for programs that benchmark and rank the world's fastest supercomputers [2] [3].

Fortran consists of numerous versions, which have added support for structured programming (FORTRAN 77), array programming, modular programming and generic programming (Fortran 90), high performance (Fortran 95), object-oriented programming since 2003 and concurrent programming in 2008 [1]. Influenced languages by Fortran include BASIC, PL/I, and C, as well as Algol and Java. Fortran is still used today for programming scientific and engineering applications. It is the primary language for some of the most intensive supercomputing tasks, such as weather forecasting, nuclear security and medicine [4].

Fortran is short for "Formula Translation", because it was designed to allow easy translation of math formulas into code. It is a compiled language, which means that the source code is compiled into a machine code before execution [6]. Also, it is case-insensitive, which means that the keywords can be written in any case, even though "FORTRAN" itself was written in uppercase letters, until Fortran 90, since old keyboards did not feature a shift key.

**1.1 Features**

**1.1.1 Basic structure**

A Fortran program consists of a main program, which is the starting point of the program and declared with the keyword "*program*", followed by the name of the program. In the end, the program is terminated with the keyword "*end program*", again followed by the name of the program.

*program helloworld ! Comments start with an “!” and are equivalent to “//” in Java*

*print \*, "Hello World!"*

*end program helloworld*

**1.1.2 Data Types**

Fortran is a statically typed language, which means that the type of a variable is known at compile time [5]. Fortran’s data types can be divided into two categories. While intrinsic types are provided by the compiler. Derived types are user-defined. Fortran’s intrinsic Types are *Integer, Real, Complex, Logical* and *Character*. The number of bytes used by these types can be specified with the “*kind*” keyword. Complex is used to store complex numbers, which consist of a real and an imaginary part and can be declared with the complex keyword. The real and imaginary part can be accessed with the “*real”* and “*aimag*” functions.

|  |  |  |
| --- | --- | --- |
| FORTRAN | Java | characteristics |
| integer | int | - |
| real | double | - |
| logical | boolean | .true. instead of true |
| character | String or char | Length of String can be specified |

**1.1.4 Variables**

Similarly to Java, variables in Fortran are declared with the object type followed by "::" and the name of the variable. You can initialize a variable by assigning a value to it like this:

*integer :: year*

*year = 2023*

or this

*integer :: year = 2023*

Constants are declared with the parameter keyword.

*parameter (pi = 3.1415926535)* or *real, parameter :: pi = 3.1415926535* [7]

**1.1.5 Operators**

Fortran supports Arithmetic, Relational and Logical Operators.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Arithmetical Operators | Addition | Subtraction | Multiplication | Division | Exponentiation | Concatenation [8] |
| + | - | \* | / | \*\* | // |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Relational Operators | Equal to | Not equal to | Less than | Less than or equal to | Greater than | Greater than or equal to |
| .EQ. | .NE. | .LT. | .LE. | .GT. | .GE. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Logical Operators | Logical not | Logical and | Logical or | Logical equivalence | Logical  non-equivalence |
| .NOT. | .AND. | .OR. | .EQV. | .NEQV. |

|  |  |  |  |
| --- | --- | --- | --- |
| Examples | Arithmetical | Relational | Logical |
| 4 + 5 returns 9 | 9 .GE. 69 returns .false. | .false. .EQV. .true. returns .false. |

**1.1.6 Control Structures**

Control structures in Fortran are similar to other programming languages, such as C or Java. They make it possible to control the flow of a program. Supported control structures are **if-then**, **if-then-else**, **do**, **do-while**, **select-case** and **cycle** [9]. Apart from the wording, the if-then block is identical to Java’s if block. The same applies to the if-then-else structure. The do control structure is comparable to Java’s for loop, while the do-while loop matches the while loop in Java.

Furthermore, the select case is an ancestor of the switch statement and the break and continue keyword were influenced by the exit and cycle keyword from Fortran.

**1.1.7 Functions and Subroutines**

Functions and subroutines are used to divide a program into smaller parts, which make it easier to read and maintain, hence the name "structured programming" as one of Fortran’s paradigms.

Functions and subroutines are similar, but functions can return a value, while subroutines cannot. This is because subroutines are used to perform a task, while functions are used to calculate a value. The same concept is used in Java with functions and methods.

**1.1.8 Arrays**

|  |  |
| --- | --- |
| **Entries** | **Real** |
| Entries(1) | 3.141 |
| Entries(2) | 2.718 |

Arrays are used to store multiple values of the same type. The main difference between arrays in Fortran and other programming languages is that Fortran arrays start at 1, while most other programming languages start at 0 [10]. An array in Fortran can have up to 7 dimensions, while in Java, 256 dimensional arrays are possible, to increase accessibility.

*real, dimension(2) :: Entries*

*Entries(1) = 3.141*

*Entries(2) = 2.718*

In the following, an array with multiple dimensions is declared

*integer, dimension(3,3) :: tictactoe*

**1.1.9 Strings**

Strings can be declared with the character type, initialized with a string literal and concatenated as mentioned in the Operators section:

*character(len = 4) :: name = "John"*

*character :: gender = “m”*

*character(len = 8) :: doublename = name//name*

*len(name)*

One can also access a single character of a string, similarly to Python:

*name(1:1)*

This would return the first character of the string.

The length of a string can be limited by adding a number after the type declaration like this:

*character(10) :: name*

This would limit the length of the string to 10 characters.

**1.1.10 Input Output**

Input and output in Fortran is done with the *read()* and *print()* functions.

The *read()* function is used to read input from the user and the *print()* function is used to write output to the console.

*read \*,name*

*print \*,name*

The asterisk (\*) is used to indicate that the input or output is done to the console [12].

**1.1.11 File Handling**

File handling in Fortran is done with the open(), close(), read() and write() functions.

The names of the functions are self-explanatory.

Example:

*open(unit=1, file="file.txt", status="old")*

*close(unit=1)*

*read(1,\*) name*

*write(1,\*) "Hello ", name*

The unit parameter is used to specify the file to be used. The file parameter is used to specify the name of the file. The status parameter is used to specify the status of the file (old, new, scratch, replace). The first asterisk (\*) is used to indicate the source or destination, and the second asterisk is used to indicate the type of the input or output [13].

**1.1.12 Object-Oriented Programming** Fortran is not an object-oriented programming language, but it is possible to use object-oriented programming in Fortran by using modules. Those are used to group variables, functions and subroutines together.

Example:

*module myModule*

*integer :: number*

*contains*

*subroutine mySubroutine()*

*write(\*,\*) "Hello World"*

*end subroutine mySubroutine*

*end module myModule*

This module can then be used in another program like this:

*program myProgram*

*use myModule*

*call mySubroutine()*

*end program myProgram*

This would print "Hello World" to the console.

**1.2 Conclusion**

Fortran is an old programming language, since it was the first high-level programming language that is still used today. It is a compiled, statically typed, structured, portable and parallel programming language, which makes it a good choice for scientific computing due to its stability. However, it is neither an object-oriented nor a functional programming language, which makes it less flexible than other programming languages like python or Java. The readability of Fortran is also not as good as other programming languages, which makes it harder to learn. For beginners, it is recommended to start with a more modern programming language like Python or Java.

A

B

C

nil

the only difference being the utilization of the “defconstant” construct instead of “defvar”.

|  |  |
| --- | --- |
| Operator | Example |
| + | (+ 6 3) = 9 |
| - | (- 6 3) = 3 |
| \* | (\* 6 3) = 18 |
| / | (/ 6 3) = 2 |
| mod, rem | (mod 6 3) = 0 |
| incf | (incf 6 4) = 10 |
| decf | (decf 6 2) = 4 |

|  |  |
| --- | --- |
| Operator | Example |
| = | (= 2 3) = false |
| /= | (/= 2 3) = true |
| < | (< 2 3) = true |
| > | (> 2 3) = false |
| <= | (<= 2 3) = true |
| >= | (>= 2 3) = false |
| max | (max 2 3) = 3 |
| min | (min 2 3) = 2 |

**3.1.4 Operators** We will briefly check what the arithmetic and comparison operations look like.

**References**

[1] https://en.wikipedia.org/wiki/Fortran

[2] https://en.wikibooks.org/wiki/Shelf:Fortran\_programming\_language

[3] https://gcn.com/cloud-infrastructure/2023/04/can-fortran-survive-another-15-years/385726/

[4] https://programmingforresearchers.leeds.ac.uk/fortran/

[5] https://www.baeldung.com/cs/statically-vs-dynamically-typed-languages

[6] https://ourcodingclub.github.io/tutorials/fortran-intro/#:~:text=Fortran%20is%20a%20compiled%20language,run%20it%20on%20a%20computer

[7] https://www.tutorialspoint.com/fortran/fortran\_constants.htm

[8] https://stackoverflow.com/questions/34622995/how-to-combine-two-strings-in-fortran

[9] https://pages.mtu.edu/~shene/COURSES/cs201/NOTES/chap03/else-if.html

[10] https://docs.oracle.com/cd/E19957-01/805-4940/z400091044d0/index.html

[11] https://www.cism.ucl.ac.be/Services/Formations/fortran/2016/fortranCISM.pdf

[12] https://www.mrao.cam.ac.uk/~pa/f90Notes/HTMLNotesnode40.html

[13] https://web.stanford.edu/class/me200c/tutorial\_77/05\_variables.html