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. Fortran is a statically typed language, which means that the type of a variable is known at compile time.[5]

1.1 Features

1.1.1 Basic structure

A Fortran program consists of a main program and one or more subprograms. The main program 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*".

A comment in Fortran starts with an exclamation mark (!) and ends at the end of the line.

Example:

```
program helloworld ! This is a comment
print *, "Hello World!"
end program helloworld
```

1.1.2 Data Types

Fortran's data types can be divided into two categories: intrinsic and derived types. 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.

An integer is used to store whole numbers, comparable to the int type in C. It can be declared with the integer keyword. Complex is used to store complex numbers, which consist of a real and an imaginary part. It can be declared with the complex keyword. The real and imaginary part can be accessed with the "*real"* and "*aimag*" functions. Real is used to store floating point numbers, comparable to the float type in C. It can be declared with the real keyword. Logical is used to store boolean values, which can be either true or false. It can be declared with the logical keyword. and initialized with either .true. or .false. Character is used to store a single character or an entire String. It can be declared with the character keyword, while also specifying the length of the string.

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

Variables can also be initialized when they are declared like this: "*integer :: year = 2023*"

Constants are declared with the parameter keyword. Like this:
parameter (pi = 3.1415926535)
Or like this: *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.

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] The if-then control structure is used to execute a block of code if a condition is true. The if-then-else control structure is used to execute a block of code if a condition is true and a different block of code if the condition is false. The do control structure is comparable to the for loop, 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 makes it easier to read and maintain, hence the name "structured programming" as one of Fortran's paradigms. Functions and subroutines are similar, but they have some differences, such as that 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.

1.1.8 Arrays

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. Arrays make it possible to store multiple values in one variable, which makes it easier to access them.
real, dimension(5) :: Entries
Entries(1) = 3.14
Entries(2) = 2.718

Entries(1)	Entries(2)	Entries(3)	Entries(4)

1.1.9 Strings

Strings are used to store text. They are declared with the character type and can be initialized with a string literal:
character :: name = "John"
Strings can be concatenated with the // operator. *character doublename = name//name*
The length of a string can be determined with the len() function like this:
len(name)
One can also access a single character of a string with the following syntax:
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 and Output

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

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

```
read(*,*) name
write(*,*) "Hello ", name
```

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

The first asterisk is used to indicate the format of the input or output.

The second asterisk is used to indicate the type of the input or output.

An example where the type is specified would be:

read(*,*) integer :: number
This would read an integer from the console, such as the amount of times a loop should be executed. [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 format of the input or output, 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.

1.1.13 Parallel Programming

Parallel programming in Fortran is done with the OpenMP API.
OpenMP is a library that makes it possible to write parallel programs in Fortran.
Example:

```
program myProgram
use omp_lib
integer :: i
!$OMP PARALLEL DO
do i = 1, 10
write(*,*) i
end do
!$OMP END PARALLEL DO
end program myProgram
```

This program would print the numbers 1 to 10 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. 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.

2 ALGOL

ALGOL (short for Algorithmic Language) is a family of imperative computer programming languages developed and designed by an international committee of the Association of Computing Machinery (ACM) in late 1950 and later frequently updated by the International Federation for Information Processing (IFIP) until 1980. Despite a lot of similarities in syntax, semantics, and overall structure, every version is unique and has its history. There are three official main branches of the ALGOL family: ALGOL 58, ALGOL 60, and ALGOL 68 with the numbers standing each for the year of release. Of these, ALGOL 60 was the most widely known. Because of it being the first structured programming language, ALGOL set the standards in a sense with a significant impact on the further development of programming languages. The programming language Simula, for example, is a further development of Algol 60, and the programming language C is a further development of ALGOL 68. [14] [15]

2.1 ALGOL 60

ALGOL 60 is the most widely known language of the ALGOL family and inspired many languages that followed it. It is the successor to ALGOL 58 which introduced code blocks, representing a key advance in the rise of structural programming. The goal was to develop an

international algorithmic language for expressing mathematical algorithms clearly and concisely and designed to avoid some of the perceived problems with FORTRAN and added features missing like recursion and the structured language. It was implemented on a stack and all memory allocation was to take place on the stack. [19] With that in mind, ALGOL 60 was the first language that implemented ***nested function*** (2.1.2) definitions with ***lexical scope*** (2.1.3). Additionally, the language made use of the Backus-Naur form, which was developed by John Backus and revised and expanded by Peter Naur, to give the language a universal notation for context-free grammar, therefore called a 'metasyntax'. [16] [18] [23]

Despite all the languages breakthroughs and importance for the future of imperative programming languages at that time, its usage back then was pretty limited. A major problem turned out to be that there were no explicit references to I/O processes so its implementation varied greatly between compilers and operating systems, leading to the language only being used by some companies. Therefore it was mostly used by computer scientists. [16] [25]

Unlike FORTRAN, ALGOL 60 is no longer used nowadays. The major reason was the lack of input and output statements in the language, which affected the portability of ALGOL programs because the implementation of I/O required being machine depended. [15]

2.1.1 Syntax

The syntax of ALGOL 60, while complex and not always intuitive, introduced new programming concepts that have had a significant impact on the development of modern programming languages. The language was designed to express complex mathematical algorithms clearly. ALGOL 60 introduced the concept of block structures, which allowed code to be organized into nested blocks, and the support for *lexical scope* and *nested functions*, which allowed for greater code organization and made it possible to write more complex algorithms clearly and concisely. [16] [18]
Each variable used in the program is defined by the user.

arithmetic operators	relational operators
+	<
-	≤
x	=
/ (real division)	≥
÷ (integer devision)	>
↑ (power)	≠

Table of standard operators [24]

Declerations
integer
real
boolean
array
procedure (to implement subprograms)
own (extends validity of variables when re-entering blocks and subprograms)

Table of decleration types [24

2.1.2 Arrays

In ALGOL 60, arrays of integers and reals can be defined by the pattern:

```
integer array A[start:end];
real array A[start:end];
```

Like in Java, it is necessary to declare the type of the array. In ALGOL 60 it is only possible to define integer and real arrays. Boolean arrays do not exist in ALGOL 60, unlike in Java where any datatype could be turned into an array. Declaring an array without a datatype in ALGOL 60 leads to the default array type real.

Array elements are accessed in the modern way by simply calling it by the index: „A[index]“.[27]

2.1.3 For-Loop

For-loops in ALGOL 60 are defined by this pattern:

```
for i:=5 step 5 until 25 do
  outinteger(i)
```

The loop body starts with the keyword 'DO', whereas in Java, the loop body is enclosed in curly braces '{' and '}'.

In this code, the identifier i starts at 5, increments by 5 each iteration, and ends at 25.

2.1.4 Code Blocks and Nested Functions

Code blocks in ALGOL 60 are defined using the keywords BEGIN and END. Any code between these two keywords is considered to be part of the same block. In Java, we know these as the {code block} symbols. Blocks can be nested, meaning that one block can be contained within another block. This allows for greater code organization and makes it possible to break up large programs into more manageable pieces.

Nested functions are functions that are defined within other functions. This allows for even greater code organization and makes it possible to write more complex algorithms clearly and concisely. In ALGOL 60, functions can be defined within any block, including other functions. This means that functions can access variables and other functions defined in the same block, as well as variables and functions defined in any enclosing block. [20]

2.1.5 Lexical Scoping

One of the most powerful features of ALGOL 60 was its support for lexical scoping, which allowed variables and functions to be defined in a specific context and to be used only within that context. In ALGOL 60, the scope of a variable or function is determined by its position in the code. When a variable or function is defined in a block, it can only be accessed within that block and any blocks nested within it. This means that a variable or function defined in a higher-level block cannot be accessed within a lower-level block. This allows for greater control over the use of variables and functions and can help to prevent naming conflicts and other issues that can arise

in large programs. Lexical scoping has been incorporated into several modern programming languages, including Python, Java, and JavaScript, and continues to be an important concept in programming today. [21] [22]

```
begin
  integer A, X; comment outer X;
  A := 5;
  X := 8;
  begin
    integer X, Y; comment inner X;
    X := 5; comment inner X assigned here;
    Y := 10;
  end
  outinteger(1, X); comment prints "8", not "5" since this is outer scope;
Y := 12; comment error! Y not defined in outer scope;
end
```

Scope and Block Code example [27]

2.2 Advantages and Disadvantages

Advantages

- ALGOL 60 was designed with mathematical notation in mind, which made it suitable for use in scientific applications.
- It was also one of the first programming languages to use the concept of recursion.
- ALGOL 60 is undeniably one of the most influential programming languages and it introduced many concepts that are now standard in modern programming languages, such as the use of code block structures and the idea of local parameters.

Disadvantages

- ALGOL 60 did not have any standard built-in input/output facilities, which meant that it was heavily dependend on individual configurations like the compiler and the computer
- The limited set of data types made it difficult to write programs with a complex data structure.
- The language lacks modern programming constructs, such as object-oriented programming and exception handling, which makes it less useful for developing large-scale software systems.

2.3 Conclusion

In conclusion, ALGOL 60 was a significant programming language that introduced several concepts that are now standard in modern programming languages. Its support for block structures, lexical scoping, and nested functions allowed for greater code organization and made

it possible to write complex algorithms more clearly and concisely. However, the lack of standard input/output facilities and the limited set of data types made it less useful for developing large-scale software systems. Despite its limitations, ALGOL 60 paved the way for the development of future imperative programming languages and its impact is still felt today. and is now considered a legacy language with limited practical use.

3 Lisp

Lisp (an acronym for **lis**t **p**rocessing) is a multi-paradigmatic, functional, and procedural programming language which has been designed by John McCarthy in 1958 at the Massachusetts Institute of Technology (MIT). The first purpose of the specification of Lisp was a mathematical notation system for computers which is reflected in the language's notation, influenced by the notation of lambda calculus by mathematician Alonzo Church. It is the second high-level programming language ever created after FORTRAN. Later Seve Russell, who is a student of McCarthy, developed the first implementation of Lisp using punched cards on an IBM 704 computer where the code was run by using an interpreter which makes Lisp the first high-level programming language run by an interpreter. [28]

Being one of the oldest programming languages, Lisp has introduced and pioneered numerous concepts in the field of computer science in general. Ideas such as garbage collection, dynamic typing, higher-order functions, tree-data-structures, recursion, macros, and one of the main selling points of Lisp: S-expressions (symbolic expressions). These are only a few examples of why Lisp has left many traces in the field of programming languages and many dialects have been derived from Lisp. The dialects can be domain specific like AutoLisp but variants like Common Lisp, Clojure, and Racket are leaning towards general-purpose. Not only dialects were influenced by Lisp but also various programming languages. For instance, Python, Perl, Logo, and Haskell, which we will be comparing to Lisp in a later section (1.3 Comparison with Haskell). [28]

The early application field of Lisp was research on artificial intelligence. It was used in 1968-1970 in the implementation of the AI system SHRDLU. Other use cases are in the field of aerospace. The airplane manufacturer Boeing uses a Common Lisp software package called Piano in the models Boeing 747 and 777. Another airplane manufacturer that uses Piano is Airbus. The German engine manufacturer MTU Aero Engines employs the dialect AllegroCL and the North American supermarket chain Walmart utilizes Clojure in its data management system which is used in more than 5000 stores. The Lisp family of programming languages was and is still in use even after 60 years after its debut.
In 1988 Lisp was ranked 3rd and in 1998 listed as the 6th most popular programming language on the TIOBE Index. In 2008 the rating dropped to 16th place and in the April 2023 rating Lisp has been placed 34th while FORTRAN is the 20th most popular programming language.
[29] [30] [31] [32]

3.1 Syntax

In Lisp there is no distinction made between data and code, both are written in expressions. After an expression is evaluated, it produces a value that can be used in another expression. The expressions in Lisp are so-called “S-expressions” (symbolic expressions). Lisp has been called “Lots of Irritating Superfluous Parentheses” to ridicule the extensive utilization of parentheses in the code. Another point to keep in mind is that everything is a list and that Lisp uses the prefix notation (also known as the Polish notation). There are three types of elements that are constants and return their own value after evaluation: numbers, the symbol ”t“ (logical true), and “nil” (logical false) which stands for the empty list.

A Lisp program that is being interpreted goes through a “read-evaluate-print-loop” (REPL). That means that the source code is being read, then evaluated and finally, the values returned are being printed. Important to note is that everything is being evaluated if it is not specifically marked to be ignored. [33]

The basic building blocks of Lisp code are atoms, strings, and lists. These three elements form the famous s-expressions. An atom is a number or a string of characters, numbers, and special characters. Here are some valid and invalid examples:

Valid atoms	Invalid atoms
123987	(hi
name	123hi
number#3	hi world

Everything that is enclosed by quotation marks (“ ”) is a string or example *“Hello world!123”* but *“hello* and *hello* are not strings. We will have a closer look at the lists in the next section. [34]

3.1.1 Lists

Most of the code in Lisp consists of lists. A list in Lisp looks like this:

```
(A B C D)
```

This is the list consisting of the letters A, B, C, and D which are separated by using whitespace. The list elements are written inside parentheses. We already know that the interpreter evaluates everything in the code. We put a quote symbol in front of the parenthesis to prevent the evaluation of a list.

The next example shows a list inside another list:

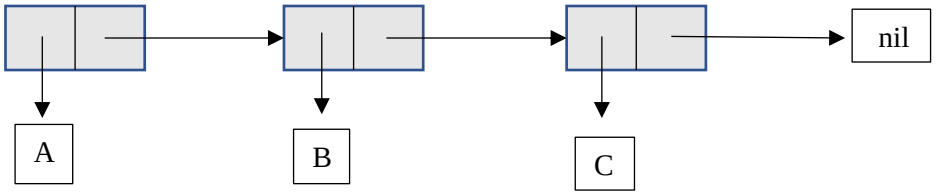
```
(A B (C D) )
```

The only element in Lisp that is an atom and a list at the same time is the empty list ( ) which is also called nil.

Lists in Lisp are actually made out of cons cells. The "original" way of creating a List, which is too long, is shown in the following:

```
(cons A (cons B ( cons C ( cons D nil ))))
```

This equals to the first list we have looked at (A B C D). Cons cells can be visualized as a linked list. Every cons cell contains two pointers: one points at the actual data and the second pointer refers to the next cons cell. The last element in this linked list is the empty list nil.



There are two fundamental functions that operate on lists that are called "car" and "cdr".

```
(car '(A B C))     Output: A

(cdr '(A B C))     Output: (B C)
```

The car function ("contents of the address part of register number") returns the first element of the list while the cdr function ("contents of the decrement part of the register number") returns the rest of the list (next address of the first element).

3.1.2 Functions

A function is written inside parentheses and the first string represents the function name. The arguments of the function come right after the name. Optional arguments can be added after the "%optional" parameter:

```
(function arg1 arg2 … argn %optional args)
```

The definition of a function looks like this:

```
(defun function (args)
        (…)
        (output) )
```

To create functions, the keyword "defun" is followed by the function name that is wanted and the

arguments that the function needs. After the main structure is established the code that is meant to be evaluated can be implemented after the arguments are defined. The implementation can consist of more expressions than one but the last expression's return value will be the whole function's return value. [33] [34]

3.1.3 Variables

Global variables can be declared with the "defvar" keyword. If wanted, the value can be directly initialized:

```
(defvar var value)
```

The keyword "setq" is employed to set or change the value of a variable. It can also be used to set local variables:

```
(setq var value)
```

Because there is no type declaration in Lisp, we can directly assign the value to the variable. Local variables can be created with the two keywords "let" and "prog":

```
(let (variable value) (...))
```

This can produce more than one local variable at the same time. Since we only have limited pages, the usage of "prog" can be read in the source [33]. Constants are defined exactly like variables, the only difference being the utilization of the "defconstant" construct instead of "defvar".

3.1.4 Operators

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

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

3.2 Conclusion Lisp

After all these years Lisp is still in use, thanks to its powerful macro system and the ability to treat code as data which allows the extension of the language into any shape that is required in various sectors of the world.

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

[14] https://de.wikipedia.org/wiki/ALGOL

[15] http://www.computernostalgia.net/articles/algol60.htm#:~:text=Tracing

[16] https://en.wikipedia.org/wiki/ALGOL_60

[17] https://www.fruehes.berlin/wp-content/uploads/2013/12/ALGOL-60-V4.22.pdf

[18] https://www.encyclopedia.com/computing/news-wires-white-papers-and-books/algol-60-report#:~:text=The

[19] https://dl.acm.org/doi/pdf/10.1145/159544.159553

[20] https://en.wikipedia.org/wiki/Block_(programming)

[21] https://en.wikipedia.org/wiki/Scope_(computer_science)#Lexical_scope

[22] https://www.techtarget.com/whatis/definition/lexical-scoping-static-scoping

[23] https://www.itwissen.info/Backus-Naur-Form-BNF-Backus-Naur-form-BNF.html

[24] https://dl.acm.org/doi/pdf/10.1145/367236.367262

[25] https://dl.acm.org/doi/pdf/10.1145/364099.364222

[26] https://www.techtarget.com/whatis/definition/lexical-scoping-static-scoping

[27] https://code.fandom.com/wiki/Algol_60#Scalars

[28] https://en.wikipedia.org/wiki/Lisp_(programming_language)

[29] https://typeable.io/blog/2021-10-04-lisp-usage.html

[30] https://common-lisp.net/lisp-companies

[31] https://www.tiobe.com

[32] https://www.cognitect.com/blog/2015/6/30/walmart-runs-clojure-at-scale

[33] https://www.tutorialspoint.com/lisp/lisp_program_structure.htm

[34] https://www.gnu.org/software/emacs/manual/html_node/eintr/index.html#SEC_Contents

[35] https://www.tutorialspoint.com/haskell-program-to-read-numbers-from-standard-input