FORTAN BY TRAN

A PREPRINT

Graham S. Walker
gwalker1@email.arizona.edu

Akshith Thumma
akshiththumma@email.arizona.edu

April 29, 2020

ABSTRACT

In this paper we discuss the various aspects of FORTRAN and what its contribution in the field of Computer Science is. This paper also focuses on the unique features of the language and lays out examples of each topic being explained along. The paper also includes the History behind the development of the Language and the various versions FORTRAN has been through. Does not fail to mention about the compilers and different ways a program can be written including the concept of parallelization. Overall the paper is a very detailed description of a combination of some of the basic and Unique features of FORTRAN.

1 Introduction

This issue consists of basic theory of FORTRAN by TRAN. Although it focuses mostly on FORTRAN 95, it also has a detailed evolution of the Language and the history of it. The paper also talks about the basic Data Types and Control Structures including of all the variations in different FORTRAN versions. The paper also provides some unique code snippets which helps understand the concept in a practical way and help analyze the advantages and disadvantages of the given concept. The paper does not miss to describe the different subprograms and subroutines the language consists of and the code snippets again help understand the nature of the language and its unique existence. First off starting it of with a bit of the history of the language will give us a brief evolutionary start.

2 History

The history of Fortran is very different from the history of many high-level languages, because FORTRAN was the first high-level language, using the first compiler ever developed. Fortran was developed by a team of programmers at IBM led by John Backus, and was first published in 1957. At first, one might be surprised why anyone would develop a language like Fortran when it reduces the execution efficiency by 20 compared to assembly language, but the programmers at IBM saw a bigger picture which would change the way of writing programs for complex mathematical expressions for a long time in future and is also simple for people to learn. Yes, the big advantage was that unlike assembly, everyone found Fortran very easy to learn and code 500 faster than before. This revolutionary language was one of the big stepping stones because it was not just the first high-level language, but also because it introduced one of the greatest concepts to the field of Computer Science, The Compiler Theory.

Due to its different dialects, Fortran faced a lot of criticism and had to be revised more number of times than you would imagine. Although they released a standardized version in 1966 as FORTRAN ‘66, they had to review it again in 1978 due to new dialects being surfaced, but in 1990 a new fortran came into existence with a lot more features and blended into the new age programming perfectly fine. Today, FORTRAN ‘90 still stands high in the list due to various reasons and one of the most important ones being the knowledge transfer IBM has managed to have. Almost every old age programmer knows Fortan and at a stage even new age programmers become familiar with it due to the usage of the language by many big companies like Intel, Android and HP.It is not just the language and the software but also its impact on the hardware such as the Raspberry Pi, which contributes towards the success of Fortran in the field of Computer Science.

3 Control Structures

In many programming languages we know that the flow of the program is controlled by the control structures. In FORTRAN we have a variety of control structures right from the Boolean checks to switch statements to Do Loops. Although FORTRAN is not completely different from many high-level languages, control structures in FORTRAN operate a bit different from Java and C.

In FORTRAN we have three main if conditions we might be able to use: Arithmetic if, Logical if and Block if. Arithmetic If: This if condition was introduced with the very first version of FORTRAN and is a bit different from modern high level programming languages. This type of conditionals use goto statements to execute what the programmer wishes to execute. Although due to the dangers goto statements possess, Fortran 2018 no longer supports this kind of if conditionals. In this format the "if" checks the condition specified by the programmer and the "goto" executes a jump to the statement associated with the goto and the else case is handled by an extra goto which is executed in case all the if checks return false.

Logical If: Before FORTRAN 77 was introduced FORTRAN IV was one version of the language where logical if conditionals were introduced to make if conditionals more efficient and avoid the use of goto statements. Although developers of the language tried to avoid the goto statements with Logical If statements, it was not completely achieved because after a Logical If condition only one statement declaration was allowed and it was not the best solution with large programs. With Logical If the use of goto statements was not necessary and programmers could control the flow of the program smoothly unlike the Arithmetic conditions. In this format the "if" checks the condition and executes the one line which has been declared by the programmer and continues with the rest of the program without executing the conditions if the condition evaluates to be false.

Block If: One could say a Block If conditional is the perfect if condition in FORTRAN which is now close enough to a lot of popular high level programming languages like Java. This kind of if conditional is very similar to Haskell's if-then-else format of conditions. The main advantage of this type of condition is that one can avoid goto statements and at the same time execute multiple statements unlike the above two conditionals. In this format the "if" checks the conditions an the then does the execution and finally the "else" handles the false case of the condition.

```
if (x .gt. 0) then
        print *, "x is greater than 0"
else
        print *, "x is not greater than 0"
end if
```

Select Switches: According to the previous paragraph about the Block If condition being the more prioritized if condition, a select switch format of conditioning basically acts as a substitute to the Block If statement. In this kind of conditioning the select case statements opens up the list of cases we are about to check and each case statement acts as and if condition checking and comparing the condition in the current case to the base case which is also known as the select case. The command to be executed after a case has been matched will follow the case statement and again gives the programmer the advantage of avoiding goto statements. The "end select" statement indicates that the select switch block has been completed.

```
integer :: n = 3

message: select case (n)
        case (0)
                print *, "not too great"
        case (2)
                print *, "wow, what a treat"
        case (3)
                print *, "omg, like , totes awesome"
        case default
                print *, "no, absolutely not"
end select message
```

Control structures in FORTRAN also include loops like the Do Loop and Do-while Loop to execute a command or commands repetitively until a condition is met. Below are the most useful loops in FORTRAN: Do Loop: A Do Loop executes a list of commands until an "exit" occurs. This is like an uncontrolled loop and if there is no way the pointer reaches the exit statement; we might experience and infinite loop which would not be the most desired solution by a programmer. Below is an example of how Do Loop does recurring execution of commands.

```
do i = 0,20,2  !the 2 is the step size. It is optional
        print *, i
end do
```

Do-while Loop: A Do-while Loop is very similar to a Do Loop, but in this format of the loop, the pointer does not look or wait for an exit statement, but executes the list of commands until a certain condition is met. The main advantage with a Do-while Loop is that it can be a bit more controlled by the programmer and helps avoid fatal infinite loops. Below is an example of how the Do-while loop does recurring execution of commands.

```
integer :: i = 0

do while (i .lt. 10)
        print *, i
        i = i + 1
end do
```

Both the above loops use incrementation to reach the end statement of the loop i.e. “end do”. When the pointer reaches the end do statement, it is a quick message to the compiler that all the iterations of the loop are completed in order for the program to continue with the flow of the program.

4 Data Types

Data types in FORTRAN are like any other high level language, but with different names. As we all know Data types are one of the most important parts of a language, they help the compiler identify a variable to a particular type and helps in compiling the program. When a variable is declared, we declare it along with the data type associated with it and not only does this help compiling the program, but also sets limitations to the things we can do to the variable to avoid errors. For Example in FORTRAN if we declare a variable with the Logical Data type to true, we can only check if something is true or false and acts as a Boolean, but if we have a variable declared as Character Type and it’s value is True, the declaration limits us from using the variable as a Boolean check, but lets us do all the operations associated with the Character Type. FORTRAN now offers us a FIVE main Data Types along with an option to derive our own Data Types as well. Below is a detailed explanation about every default Data Type FORTRAN offers us.

Integer Type: The Integer Type only holds Integer Values. This means that the Data Type allows us to only operate on digits and perform arithmetic operations on it. Below is an Example of how an Integer Type operates in FORTRAN.

```
!declare w/o initializing
integer :: i
!initialize
i = 5

!declare and initialize
integer :: n = 3

!specify bytes for shorts and longs with "kind" specifier.
!the default is a 4 byte integer
integer(kind = 2) :: short
integer(kind = 8) :: long
```

Real Type: Real Type acts as a double from most high level languages like Java. This Data Type also allows us to operate on numbers and perform arithmetic operations. Below is an Example of how a Real Type operates in FORTRAN.

```
!declare and initialize real number
real :: r = 3.14
```

Complex Type: This is used for storing complex numbers. A complex number has two parts, the real part and the imaginary part. Two consecutive numeric storage units store these two parts. Below is an Example of how a Complex Type operates in FORTRAN.

```
complex :: com
!this represents 5.0-7.0i
com(5, -7)
```

Logical Type: There are only Two values associated with the Logical Type, true and false. This is a very important Data type when it comes to conditional checks. Below is an Example of how a Logical Type operates in FORTRAN.

```
logical :: bool
!set bool to false
bool = .false.
!set bool to true
bool = .true.
```

Character Type: The Character type stores Characters and Strings. The length of the string can be specified with the "len" specifier. The default length of a character is 1. Below is an Example of how a Character Type operates in FORTRAN.

```
!single char
character :: c = "a"

!string of length up to 50
character(len = 50) :: str = "Hello,...Newman"

!len keyword is optional
character(50) :: str2
```

Derived types: Fortran allows for custom types much like a struct in c.

```
!this is a custom "Student" type
type Student
    character(50) :: name
    integer :: studentID
end type student

!declare a Student variable
type(Student) :: stu
type(Student) :: stu2

!access/initialize the Student
stu%name = "Graham Walker"
stu%studentID = 123

!we can also initialize like this
stu2 = Student("Akshith Thumma", 321)
```

Pointers/Targets: Pointers in Fortran operate much like other languages in that they allow us to point to locations in memory. If we want to point at a variable we must both declare a variable as a pointer and we must declare another variable as a target. If we print or do arithmetic with a pointer, it is automatically dereferenced.

```
integer, pointer :: ptr
integer, target :: tar = 4

!associate the pointer w/ target (point to it)
ptr=>tar

!this will print the value 4. that is, it will print
!the value of the target that ptr is associated with
print *, ptr

!disassociate the pointer from the target
nullify(ptr)
```

5 Subprograms

Sub-Programs: Sub-programs in general are basically a set of instructions packaged as a unit to perform a specific task. Sub-programs could be Functions, Methods or any control structures. Basically we can use a subprogram in a programming language for simple tasks like computing a value in the program or a more specific task like writing a function which reads in data from another file. FORTRAN offers two types of sub-programs: Functions and Subroutines. Although both the subprograms assist doing the same task more or less, it is very crucial to understand the minute difference between them to use them in the appropriate place while writing large programs.

Function Subprogram: Function subprograms have an explicit type and are intended to return one value. The name of the function subprogram needs to have a numerical or logical value. Can be invoked by just calling the name of the function subprogram and may be used in the same way as an arithmetic statement. A function subprogram needs to have at least one argument. A function subprogram need to have a return statement.

```
program p4_function
    implicit none
    integer :: a = 2
    integer :: b = 3
    integer :: result
    result = sum(a, b)
    print *, "The sum returned from the function was:", result

contains

integer function sum(a, b)
    implicit none
    integer :: a, b
    sum = a + b
    return
end function sum
end program p4_function
```

Subroutine Subprogram: Subroutine subprograms have an no explicit type and are intended to return multiple values. No value is generally associated with the name of the subroutine program. The programmer might be able to invoke the subroutine subprogram only by a special call statement. A subroutine subprogram can have no arguments at all. A subroutine subprogram does not require a return statement.

```
program p4_subroutine
    implicit none
    integer :: i = 5

    call printNTimes(i)

contains

subroutine printNTimes(n)
    integer :: n
    integer :: start

    do start = 1, n
            print *, "You're a jabronie"
    end do
end subroutine printNTimes
end program p4_subroutine
```

6 Summary

This detailed description of the Language although talks about the technicalities of the Language the, the practicality of the language is that FORTRAN is a language one uses when it comes to mind boggling numeric operations. The core of

it being built for scientific computing it makes FORTRAN one of the best choices to operate on. This computational language carries its legacy from way past in time and has proved to be one of the considerable high level languages even after many powerful modern languages like Java, C++ etc... One of the other best advantages of FORTRAN practically is that due to the longest existence of the language all the computational literature is found in FORTRAN and makes it easier for Research and more. Finally, we can say that if FORTRAN was able to fight through all the ,major modern high level languages to stand up there , it does have a lot in unique. We can also conclude to say that FORTRAN was itself a corner stone of modern programming.

`https://www.ctan.org/pkg/booktabs`

References

[1] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.

[2] George Kour and Raid Saabne. Fast classification of handwritten on-line arabic characters. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 312–318. IEEE, 2014.

[3] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.