Assignment

4

GREGORY SCHOFIELD

CIS*3190 Assignment #4

Legacy Software Comparing Languages

Purpose of this program

This assignment is meant to compare the processing time needed to complete the Ackermann function, both iteratively and recursively. The languages compared in this study were Ada, C, FORTRAN, and Python, respectively. The goal was to come as close as possible to using identical algorithms in each language so the comparison results would be calculated fairly.

Getting Started

The iterative Ackermann functions were written using a stack structure and implemented in Ada, FORTRAN and C. Python offered a built-in list structure that allowed for replicating a stack so that structure was used instead of writing a simple stack.

The programs were tested in two different places, my personal computer and the school of computer science's Linux server @weasley.

All tests were given the input numbers 4 and 1.

Ex: Ackermann (4,1) = 65,553

Results

To time the functions I used the same psuedocode:

Grab the current time

Call Ackermann (4, 1)

Grab the current time

Subtract time1 from time2

1. C: C, as expected, was the overall winner in both categories. The average time for the iterative function was 61.52 seconds and the average time for the recursive function was a blazing fast 30.32 seconds.

2. Ada: Ada performed well both recursively and iteratively. In fact it came in second place in my tests with an iterative average time of 62.54 seconds. The recursive time also came in second place and was much faster than the iterative time with an average of 38.13 seconds to complete.

3. FORTRAN: I expected FORTRAN to either come in first or second place, close to the time that C would take to complete the function. To my surprise FORTRAN had an average iterative execution time of 95.90 seconds. The recursive function fared better but still not great with an execution time of 50.05 seconds.

4. Python: Python wasn't even in the same league. Barely even on the same planet. I thought at first there were possibly some issues with my code but everything is working properly and the right answer was eventually produced. The time for Python to perform the calculation iteratively was 28 minutes. Yes minutes.

The results of the tests were particularly interesting because of how the recursive algorithms outperformed the iterative algorithms. On average the recursive version of the algorithm performed the calculations nearly 50% faster than their iterative implementation. The Python results were also surprising because the code itself is the most succinct and it is based on the C language. Python apparently isn't great at performing Ackermann calculations.

Ada

Benefits

Designed by the military, Ada is often used in mission-critical applications. Military systems, aircraft computers, metro transit and other such high stakes applications are often built in the Ada language. Despite the fact that it is over 30 years old, it is a modern feeling programming language and based on the testing performed, also an efficient language. Ada is meant to be a safe, secure and reliable programming language for real time and embedded programs. The Ada standard supports Object-Oriented programming, exception handling, multitasking, and data abstraction so it is in many ways more modern than C.

Limitations

The main limitation of Ada for me is that it isn't a very popular language so documentation and community support for the language is slow. Also because the language includes runtime checks and very thorough syntax/error checking, It can take a lot longer than other languages to get a program up and running.

Usability

Out of all the legacy languages that we looked at in this course, Ada is the most usable in my opinion. In some ways I even like it more than C. The syntax is easy to understand and the error checking by the compiler is extremely good. I read somewhere that getting an Ada program to compile is very difficult but once it does compile, you can count on it working. I'd agree with that. The process for modularizing the programs feels familiar to me after using C and importing functions and libraries is trivial. As opposed to FORTRAN and Python, Ada requires a semicolon which I always preferred after programming so much in C and Java. Another thing that really enhances usability for me is that it seems like every keyword has an ending keyword. Procedure-end procedure, if-endif etc. This allows for nice looking code and enhanced readability which makes refactoring Ada a breeze.

Efficiency

Based on my tests Ada is a very efficient language. It is second only to C in speed required to complete the Ackermann function but in a lot of ways feels more modern than C and has a higher degree of readability.

C

Benefits

The benefits of C lie in its ability to access the low level aspects of the computer easily to allow the programmer greater flexibility. The C language tries to interact closely with the local environment and it allows you to do things like manage memory manually unlike some other high level languages. This means that more efficient programs are possible and this showed during the testing because C ended up being the fastest language to complete the Ackermann function. C also is a lot less verbose than Ada, FORTRAN or COBOL but the readability can suffer if the programmer isn't careful.

Limitations

The main limitation of C in my opinion is that it is so flexible that the quality of the program is extremely dependent on the programmer knowing exactly what they are doing. Because memory management is manual instead of garbage collected like in Java or Python, memory leaks are easily created. Also because C programs can be written very succinctly a programmer has to take care to make their programs readable or refactoring the code later can be difficult. The lack of exception handling in C can also be considered a sizable limitation for programmers with less experience.

Usability

The usability of C suffers a bit because of its flexibility. It is a language that can solve a problem in a number of different ways. Also the readability depends mostly on how the programmer has written the code.

Efficiency

C was the most efficient. Both the recursive and iterative functions performed the calculations the quickest of the four languages but the code would have taken the longest to write of the four languages.

FORTRAN

Benefits

FORTRAN is mainly for the development of programs with scientific and numerical computing requirements. This means that it is particularly well suited to calculations and number crunching which is supported by its powerful implementation of arrays. Also the FORTRAN standard does not allow aliasing so the compiler can just ignore possible aliasing, allowing for more efficient code.

Limitations

Some of the commonly described limitations of FORTRAN are the poor string handling and the pass-by-reference nature of the language. Since strings were not required by the tests, so that didn't really come into play for this assignment. Also FORTRAN didn't support recursive functions until recently so older versions of the standard lack the functionality to even perform the Ackermann function without an iterative implementation.

Usability

FORTRAN for me was the least usable of the four languages profiled. I don't find the code to be particularly readable despite the verbosity of it and the intent keyword instead of a return keyword felt cumbersome. I also don't like how a function and a subroutine are considered two separate entities but that's just personal preference.

Efficiency

Surprisingly, FORTRAN scored the lowest out of the three compiled languages in the Ackermann tests. In a newer FORTRAN standard (F2003) pass-by-value is now supported so if that standard is used, FORTRAN can produce similar results to C.

Python

Benefits

Python is a completely different beast than the other languages used in the project. First of all it is an interpreted language as opposed to a compiled language like Ada, FORTRAN, C and COBOL. This allows it to be a weakly typed language which is nice except the programmer has to keep track of all the variable types mentally. Also, unlike the other languages, the standard data types include some useful data structures like dictionaries and lists which I utilized for the Ackermann function. Using the list datatype, I was able to implement a stack trivially and in the least amount of lines by far.

Limitations

Some of the limitations include the fact that it is weakly typed. This means that data types could get mixed up if the programmer isn't careful. Also because it is an interpreted language, it's possible to have errors in your code that do things you don't want and you won't realize until after it has run, similar to a scripting language like Bash or Perl.

Usability

Usability wise, Python is in a different league than any of the programming languages that I've ever used. It was trivial to write an iterative Ackermann function because the stack data structure is easily implemented by the list data type. Also, because of the need to indent code for it to work properly, the code is readable and neat. The weakly typed nature of the language was also awesome

for usability, at least in a program this small, because it prevented me from declaring any data types and just let the interpreter figure it out. Unfortunately the code that took me the shortest time to write also took the longest time to execute.

Efficiency

Python scored dead last in the tests by a wide margin. The fact that it is an interpreted language meant that the Python interpreter needs to be invoked which surely slows the calculation down. The tradeoff for ease of use seems to be efficiency.