Name: Daniel Deen
Id: 0896283

Benefits

The distinction between a function and procedure. This one did not catch me by surprise as this distinction came up in the last assignment on Fortran. Having a function be more of a variable. While on the other hand, the procedure was only there to increase code reuse and make things more modular. Both have a clear and distinct use which make them really nice to have in a programming language. I am glad they have this feature in this language.

The concept of procedures brings me to another thing I like about Ada. Minimal pointers and memory management. The procedures were a blessing as I did not have to worry about how the heck I was going to get values out of my procedure. Instead, they were already there to begin with. I did not have to worry about allocating memory so that I could get more than one value out like C. It was a nice breath of fresh air compared to the schools C loving assignments.

The next thing I would like to talk about is declaring variables in a procedure. Variables are easily declared in an explicit type by calling the name of the variable, a colon, the intention of the variable and finally the type of the variable. This is in my opinion much better than an implicit variable as this reduces the scope of errors that could occur in the real world. However, the truly unique feature of Ada is specifying which procedure arguments get used as input, output and which ones do both. This is truly ingenious and goes hand in hand with its great memory management by again reducing the scope of errors that could occur. Making it a easier to program and a lot harder to make mistakes.

Having all local variables at the top of whatever code body you created was a nice touch I liked. This increased code readability and made it easier for you to tell what is a variable and what is not. You have to remember that functions look like variables(and kind of behave like one). Because of this it is definitely necessary to have this variable list at the top.

Limitations

Ada is a seemingly complicated Object Oriented Programming language. Maybe it was just because of the way I did it. Though I think the implementation of the stack program would have been much easier to read in Java, C++, Python, C# and basically any other mainstream language. To me not including a initialization on the same statement as the declaration variable is really bad. Here is an example of the creation of an object Car.(source: https://www.adaic.org/resources/add_content/docs/95style/html/sec_9/9-3-3.html)

```
Temp_Ptr := new Car;
Initialize (Temp_Ptr, Make, Model);
```

It not only affects readability but it allows you to create an instance which has not been initialized. This means you could accidently create an object and use one of its uninitialized values.

The lack of documentation/support for the language was noticeable. Unlike the mainstream languages you can not solve every single problem by googling it. This was kind of a buzzkill but life is life. I still think it is worthy enough to be mentioned though as having examples is a great way to learn a language.

I think having the list at the top of variable, while good for readability, is extremely bad for scope. Since you can not declare a variable within loops it increases the chance of maybe using a value you should not have.

Another thing I do not like is that you can not modify the input variables. This seems like it should be possible as we aren't returning these variables any ways. However you have to create entirely new local variables in order to use the ones you inputted. This decreases readability and could probably have been avoided. Since it is a compiled language you should be able to analyze if it is or is not going to be modified later in the execution.

Was Ada well suited to solving the problem?

No it would have been much easier in most other languages. This assignment was essentially implement a faux stack frame. If I used any language that already had the basic stack implemented(ex. Java, C++, etc) it would have saved me some time. I find it funny how its not already a thing in Ada.

As for the actual execution of the ackermann program I thought it was nice. I liked the idea of not having to provide a package name for each function. It made things extremely readable and easy to code. Specifying the in and outs for procedure variables had both ups and downs for this assignment. One upside being the fact that it reduced my errors a little bit. However this came at a cost of increased lines and variable number. I think that if I did this in most other languages the program would have been much, much shorter(and cleaner).

All in all this language would be much better for longer and more complicated programs. A short program like this is probably best suited for something like Java. In reality you are never going to test the upper reaches of this algorithm so speed is not really necessary. Java already has a stack implemented in its standard library so that's half the work already done for you. Honestly, there was not too much that stood out to me in terms of the language helping in the creation of this algorithm. Most languages do the same thing Ada does but better.(Ex. Fortran and it’s in and out subroutine variables)

What particular structures made Ada a good choice?

The lack package names for their respective procedures call was great for small programs like this one. Another thing I liked was the the functions, that made the code fairly readable. To be honest though, there was not enough coding in this assignment for me to make an accurate statement on Ada. Nothing jumped out spectacularly and told me that this was the correct tool for the job. Because of this lack of experience all I can say is that other languages do what Ada does, but better.