Ackermann

By: Sameer Gulamali - 0808465

The ackermann function is a function that is used mostly to test computer efficiency. In this assignment, an iterative solution was implemented using stacks in different languages to measure efficiency and compare the overall languages.

C

Iterative

The professor provided code for the Ackermann function that I transitioned into C. At first, the code did not work on my laptop and this happened when I added the timer and efficiency implementation. The problem was with the library linking as the function ‘timersub’ could not be found. To fix this, I had to find the ‘timersub’ function used within the program and define it at the top of my program.

Review

All in all, programming in C was the easiest for me because of my background in this language. Although I had to do only one program, it was simple enough to follow through even though the code given was in Pascal. The algorithm was the same and adding a prompt and timing required minimal understanding. The structures used for the iterative solution were simple enough to follow using basic pointer knowledge. The things I like about C is its overall simplicity. The syntax is used globally and is used and has been adapted by many modern languages. The main disadvantage I have with C is its need to make the user manually manage memory. With other languages such as Java, memory management is dealt with and garbage collection is managed better in these such languages.

Ada

Iterative

As for the iterative solution to Ada, I made three separate files. One was “stack.ads”. This file contained the functions declarations. For the iterative approach, I only used three functions/procedures: push, pop, and stackEmpty. The second file was “stack.adb” where I implemented these functions and defined the stack. The stack was implemented as a record containing an array of integers ranging from 1 .. 100000 and an integer (defined at 0) that was the top of the stack.

As for the function implementations, I mostly used the code given by the professor to guide me. The implementations were simple enough to convert easily. The third file was “iterative.adb” and this file contained the main program as well as the ackermann function. The ackermann function itself was implemented similar to that in C.

Review

As for the overall implementation in Ada, I found it fairly simple to go about the iterative program. The structs were simple records that were similar enough with C as well as the pointer management being simple with C as well. I also found that the pointers were simple and easy to track and understand just like in C. Although Ada has minor limitations such as variable declaration being before the main, it still allows for simple and easy to understand programming. Although Ada is similar to C it still has quite a few words to write while programming. Words such as then, begin, end, is array, of, are used quite extensively in the language when otherwise wouldn’t need to with the use of more symbols.

Fortran

Iterative

When going about the iterative program in Fortran, I found that this was the hardest and messiest language to implement. Firstly, I started by making the basic stack structure. This was a module that contained the top and the integer with dimension 1000000 that made up the stack array. The functions were difficult to understand at first as every function had to declare “use stack” when using the module created. The functions were used using the “call” keyword in order to use the functions. The implementation of the functions themselves were straightforward when following the given iterative C code.

Review

All in all, I found Fortran to be the hardest language to program in. This was mainly because of the differences in pointers and function syntax fortran provides as opposed to Ada and C. The functions were different in that instead of using return values, result() was used in order to change variable values within the functions. This was very different from any other languages I've used and made tracking the variables somewhat difficult. Another point of difficulty was using functions within

subroutines. This involved declaring the variable name (as the function name). For the iterative program where the “r” value was changed, I had to declare an integer variable called “ackermann” that used the ackermann function to compute the result. Another difference was using the “intent” keyword when defining variables to add whether the function will be outputted or inputted to use in the function.
One of my most disliked aspects about Fortran is its variable declarations. For example, declaring an array is a mission. integer, dimension(1000000) :: items = 0 is the syntax for declaring an array named items and is quite ridiculous to think that this much needs to be inputted to create a simple array.

Python

Iterative

The iterative solution to python was straight forward unlike any other. Python includes great built in functions that allow for its wide use in the computer science world. I defined a Stack class that defined the basic stack functions: push, pop, and isEmpty using the already built in stack functions. In the ackermann function itself, the stack was creating using stack = Stack() and the rest of the function was implemented similarly to other languages.

Review

As for timing, I used the simple time.time() command by importing “time”. I used the built in library functions to have a start_time before the ackermann function call and a finish_time after the ackermann function call. I then subtracted the two and outputted the time in seconds.
Overall, I found Python very interesting and useful with all the built in functions already within it. It helped that the stack functions were already implemented within the language and this allowed for easy transition. Python excels when it comes to having extended libraries and built in functions. The only down side I saw to the language when programming was the efficiency in comparison with other languages. I found that the time taken to compute results were substantially larger than any other language.

Comparisons

Language	**If statement format**	**Notes**

C	if (st.top == 1000000){
	...
	}
	else if (...){
	…
	}
	else{
	…
	}
		If statements in C are easy to create and simple enough to implement. Curly braces can be added at the end of the if statement and at the end of the structure to accommodate for blocks of code. Conditionals are in symbolic format (==, &&, >=, …). “Else if” or “else” statements can be added after the control blocks (after the curly braces) to continue with the if structure.

Ada	if st.top = 1000000 then
	…
	elsif … then
	…
	else
	...
	end if;
		If statements in Ada are less appealing. Uses more words such as “then” at the end of the statement and “end if” to signify the end of the if statement. If “else if” and “else” statements want to be added, “elsif” or “else” is used before the “end if” to signify this. This means there is only one “end if” for the entire if statement. Conditionals are symbolic.

Fortran	if (st.top == 1000000) then
	…
	else if (...) then
	…
	else
	...
	end if
		Fortran if statements have elements of Ada and C in it. Like Ada, it uses words rather than braces such as “then” and “end if”. It also is similar in that the “else if” and “else” fall before the “end if” Similar to C, its conditionals are more traditional symbolically (== rather than =) and also the conditions fall within brackets ‘()’.

Python	if st.top == 1000000:
	…
	elif …:
	…
	else
	…
		Python is different from all the said languages because of the rules about indentation. It does not need an end since everything that is indentated after the statement is part of the condition. Additionally, it uses the simple colon to signify blocks of code. Conditionals are similar to previous examples and are symbolic. “Else if” and “else” use “elif” and “else” respectively.

Language	Struct format	Notes

C	typedef struct{
	  int ID;
	  char * name;
	}student;
	
	student Sameer;
	Sameer.ID = 0808465;
		Structs are basic and easily implementable in C. Name can be identified and the data within the struct has to have a defined data type associated with it. Typedef is often used to no longer require "struct" before the name. Data inside the struct is everything included within the curly braces. Struct is then defined by giving it a unique name and values of the struct can be used (Sameer.ID).

Ada	type q_stack is record
	  item : stackArray;
	  top : integer := 0;
	  end record;
	
	st : q_stack;
	
	st.top := 4;
		Structs in Ada are pretty tedious. They are known as "records". Inside a record, different data types can be stored and held. Anything between "is record" and "end record" are considered to be values within the struct. Name is defined at the end in this case (st). If i wanted to access top, I'd use st.top.

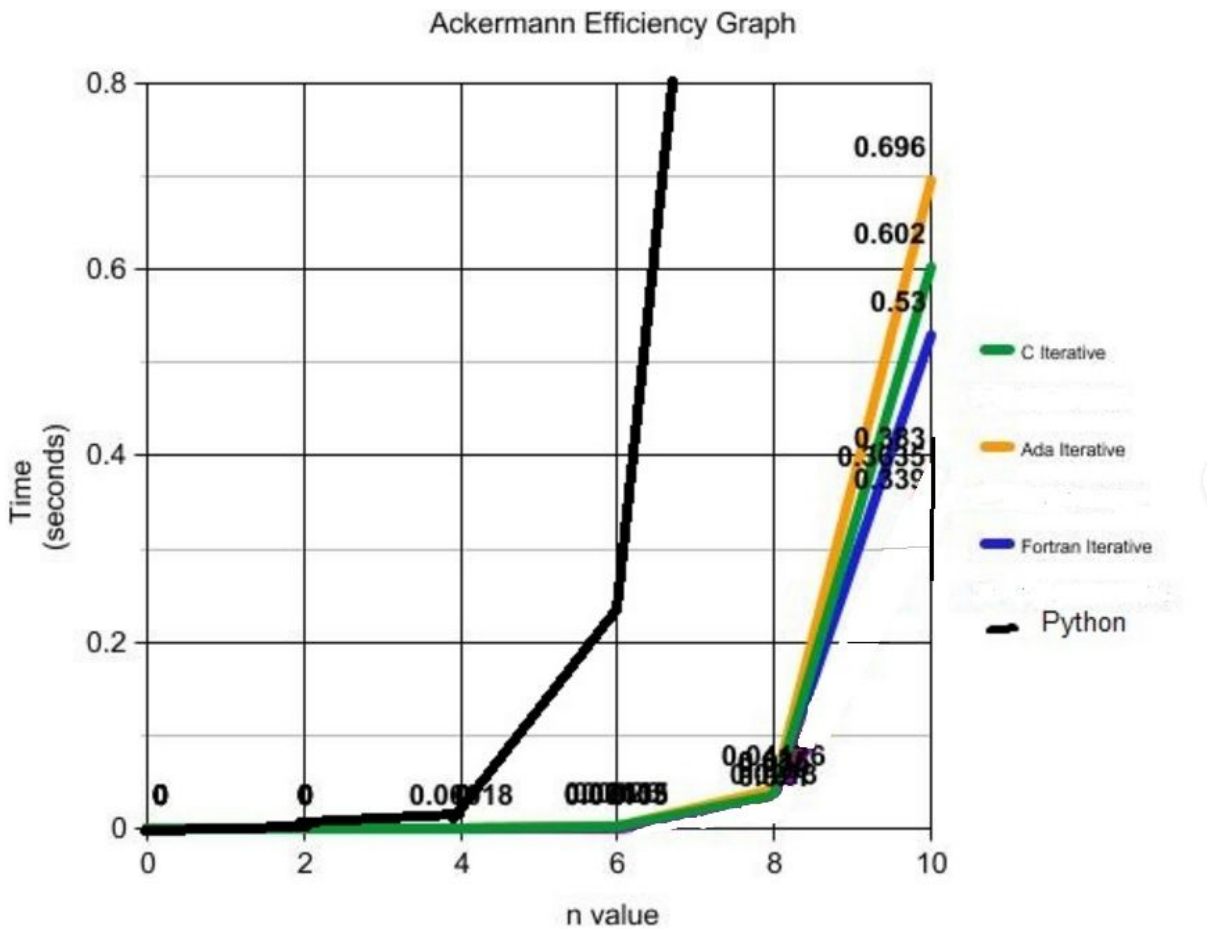
Fortran	module stack
	  integer :: top = 0
	  integer,
	dimension(1000000) ::
	items = 0
	end module
	
	use stack
	
	top = 5
		In the case of this assignment, a module was used to define a "struct". In this case, the module acts as a struct as it defines two integers, items and top. When using the variables within the module named "stack" in this case, the procedure must call it using "use stack". Then, the values are ported over and can be used normally without having stack.top (similar to other languages).

Python	class Stack:
	  def __init__(self):
	  self.items = []
	
	st = Stack()
	st.items != []
		Structs in Python are rather different in that they utilize classes instead. Procedures and functions can be defined inside a class (similar to Java) being object oriented. However, since they can also store variables they are also considered "structs". The list "items" is stored in the "Stack" struct. It is later defined as st and st.items can be used to access the items variable within the struct

		(similar to other programming languages). Self, as the name suggests, refers to itself so the name is not repeated.

Efficiency

All languages with the exception of Python had fairly similar results in regards to efficiency. Fortran ended up taking the lead in terms of efficiency, while C and Ada didn't fall too short behind. When it came to Python however, the timings were extremely high and had more time consumed than other languages combined. This was interesting to note and could be because of the use of the built in stack functions in Python.
When testing for efficiency, I always had m = 3 and had the value of n range from 0 - 10 (Below is a graph of the timings of each program).



References

This assignment has substantially similar elements with a previous assignment in a previous course taken (CIS*3190) and therefore large portions of this document (and code) was used from the previous assignment. Although it is from a different course and still an assignment, all documents are mine and have not been shared.