Exam 2, CS-415: Principles of Programming Languages

Spring 2020, Iyengar

**Your Name: ____Adam Stammer____**

What design methodology was supported and used with block structured languages? Describe this design. (10)

**Block structured design allowed for scoping, and assisted with abstraction and parallel programming. Blocks are used to define a given scope, such that variables can be declared and used throughout a program. It also allowed for procedures to be created with their own variables which is key to reusable procedures, which directly leads to modularity.**

Who is the developer of Pascal? (5)

**Nickaus Wirth developed it and then named it after an old mathematician.**

Implementing block structured code requires run-time support from the operating system. Describe the use of an operating system feature needed to implement Activation Records. (10)

**Activation records, a type of stack frame, relies on the ability to produce and maintain an activation stack.**

List some of the information stored in an Activation Record. How is static scoping – in block structured program and dynamic return to call occur? (10)

**Parameters, Current Machine Status (Registers), Return Address, Return Value (if there is one), among others things. The contents of these activation records are not standardized and can vary from language to language, but in general they have the same stuff. Once an activation record is run through (like finishing a procedure) the program looks at that activation record for an address to return to, usually the location the procedure was called from.**

Pascal was not case sensitive. Is this an advantage or disadvantage? Explain. (10)

**Generally considered a disadvantage because it increases the complexity for the programmer. Not only does the programmer now need to remember the case of any labels, but it's a possible error that can be very hard to detect. With this in mind, one could argue that this violates the Labeling Principle and the Impossible Error Principle, among others. It also limits the available labels, though I doubt this is ever an actual problem.**

Discuss the advantages and disadvantages of Strong Typing. (10)

**On the upside, strong typing can help avoid runtime errors involving data types. These types of errors can be very difficult to debug, and may also take a long time to even show themselves. This makes Strong Typing a valuable tool in many cases. This would also be an example of the Impossible Error Principle.**

**On the other hand, extra rules tend to make a language harder to learn, and can limit the abilities of the language users, or at least make a task much harder to do. Strong Typing also tends to put a bigger**

**load on compilers. Extra data types and rules associated with them means the compiler has to work harder to compile a program. Today, that cost is generally not relevant but back in the day it sure was.**

What lack of support in Pascal forced large sized program development and discouraged team development? Describe what steps needed to be taken before this support was made available. (10)

**To begin with, Pascal varied a lot from one implementation to another. This often meant that separate groups/companies/schools couldn't directly share code with each other because it might compile with one group and not with the other. Standardizing compilers was the solution to this but really didn't become reliable until the next generation of languages.**

**Early Pascal also required code to all be in one file. It's difficult to get multiple people to edit the same file(Even today this can be a limiting factor though solutions are growing). It also relied on variables being declared at the beginning of a program, so any concurrently developed code would have to be manually copied in to the main program piece by piece, first the variable definitions then the code. This was time consuming and hard to debug. Block structuring that allows for staticly scoped variables, made this copying over much easier, but it wasn't until Modularity/Libraries that concurrent programming really became smooth.**

**Personally, I think concurrent programming remains one of the biggest challenges of coding today.**

If you were to develop a solution to the Pascal Assignment in Fortran – what difficulties would you have encountered. (5)

**The syntax alone would certainly make it more difficult to code and debug. From the meaningless white-space to the lack of a line delimiter (;), unless you have a lot more experience with FORTRAN than Pascal, it would prove a challenge. Line length limitations would've probably been a struggle too, with how I did my assignment.**

**IO in FORTRAN was also much less programmer friendly, though this is less the case today than it used to be. This goes hand in hand with array assignment, in the context of our assignment, which was also considerably more difficult in FORTRAN than Pascal.**

Pascal had local variables or non-local variables – and access through static scoping. It did not permit data-hiding. Explain. (10)

**If a variable was declared and then used within a block, that block could still access that declared variable. If on the other hand, a variable of the same name was declared within that block, any call to that variable would access the local (block) variable.**

List and describe some of the remedies in Modula-2 that made it attractive. (10)

**Modula-2 was developed by the same person that developed Pascal. It was generally considered the first language to directly support modularity, which directly supports concurrent programming. This alone makes it very attractive to any group projects. The syntax was also based off of Pascal, making it not too difficult for an Pascal developer to learn. It also had features that lent toward multi-process coding that other languages didn't have.**

Why was ADA developed? Describe some of the salient features of ADA. (10)

**ADA was developed for the Department of Defense as an alternative to their many many different languages. Each system/project used different development environments and ADA was meant to standardize it all. As such it needed to be able to do everything all of the other languages could do at the time. Static Scoping, Procedure Abstraction, Exception Handling, Parallel Processing, etc. where just some of those features. Some of the things that really made it stand out compared to other languages, were excessively strong typing, preference for keywords over key symbols, and the very exhaustive available procedures for use. That last point unfortunately made compilation take a very long time.**

Define: (25)

Static Scoping: **A given definition, variable, procedure, etc., can only be referenced within the block in which it is defined. If you create variable n within procedure A, then procedure B cannot call that variable n. This makes it much harder to accidentally reference the wrong variable without it being caught at compile time, but can limit abstraction and lead to repeated definitions.**

Dynamic Scoping: **A given definition, variable, procedure, etc., can be referenced based on execution order. If you create variable n within procedure A, then procedure B uses variable n, future references to n will include any changes made by procedure B. This is generally considered harder to logic your way through than static scoping. It is relatively uncommon within modern programming languages.**

Pass by Value: **When a procedure is called, an activation record is pushed onto the stack and with it any parameters of that procedure. Pass by Value dictates that any parameters added to this activation record are done so as whatever is represented by that parameter, not where it was stored. Int x = 3; proc(x); → if passed by value, this would add the number 3 to the activation record**

Pass by Reference: **When a procedure is called, an activation record is pushed onto the stack and with it any parameters of that procedure. Pass by Reference dictates that any parameters added to this activation record are done so by adding the location of the parameter, rather than the value it represents. Int x = 3; proc(x); → if passed by reference, this would add the address of the variable x to the activation record.**