CURTIN UNIVERSITY
FACULTY OF SCIENCE AND ENGINEERING: SCHOOL OF ELECTRICAL ENGINEERING, COMPUTING AND MATH SCIENCE

Programming Languages Assignment

Student name: *Tawana Kwaramba: 19476700*

Course: *Programming Languages - COMP2007* – Lecturer: *Ascsociate Lecturer: Arlen Brower*
Due date: *October 25th, 2021*

Contents

List of Figures

List of Tables

1. Introduction

you will need to write this one last, as you will need to know what you're talking about each of the sections which you're coding for

2. Fortran

2.1. Fortran: Discussion. Fortran was a first generation programming language which was mainly based on the punch card system which was used at the times **site me**. Fortran was a language intended for mathematical operations which can be used in the field of engineering and applied sciences. The name Fortran embodies this purpose as Fortran stands for *Formula translator*.

Due to Fortran being based on the punch card system this imposed specific rules on the programming language.Including the following: the first column of each row is going to be reserved for the comment character which is either "c" or "*"; column one to five of each row are going to be reserved for statements or labels; column six is going to be reserved for the continuation of a command from the previous line; each of your commands given a row much terminate on column seventy-two; and column seventy-three to eighty are going to be reserved for sequence numbers, to just name few of the rules which are going to emposed onto Fortran. As a consquence of some of these rules, this made Fortran more intellectually involving compared to previously written languages of the plethora of rules to be memorised this made it more intellectually involving as compared to previoulsy written languages of Java, C, and Python.

2.1.1. Fortran: Comparison with written languages. While writting the fizzbuzz programme in Fortran I noticed I was thinking more than I typically would be while programming, and fortran has a lot of similarities to the named programming languages.

While programming in Fortran I had be consistently worrying about the current column numbber which I was currently in, this is not a behaviour I typically do while writing C, Java, and Python programs. Typically writting these languages I don't really care which line number or column in, I would typically choose to keep each line below 80 characters to make it easier for others to read the my code. Although, in Fortran this is going to be a hard imposed rule.

Furthermore, Fortran variables can be only limited to one to six characters long therefore, while I was programming I had to always double check my variable names to ensure that they're going to be below six characters hence, limiting my expressivity in the purpose of variable. As compared to the name programming languages, I didn't have to think about the length of my variable names I typically would make them a length which is going to be sufficient in discribing their purpose hence, in the named programming languages I was able to express myself more.

Additionally, due to the lack of reserved words in the Fortran language, I had to always double check all my Fortran variables didn't have any other intended purposes as the Fortran complier will not tell you if a variable was reserved or not, and it will just allow you to override that variable. As compared to the named programmed languages I would typically just express the problem in the language, and rely on the compilier to warn me if I have tried to override a reserved word. Therefore, while programming in Fortran I had to rely more on myself, and in other languages I can rely more on the complier.

Lastly, in Fortran if you had forgetten to declare a variable the fortran complier would not warn you of your mistake but, instead will print our a random memory address. As compared to the name programming languages, the complier would warn you of your mistake immeditely and the line number which the mistake had occured. Therefore in Fortran you would have to spend more time debugging the code at hand.

Similarities of Fotran as compared to the named programming languages is that the first character of each variable has to start with a letter and can't start with a number, Fortran will require you to declare the types of your variables same as C, and Java. Furthermore, Fortran will require

the programme field to be the same name as the current file name which is a similar idea which is seen in Java wherby the file name has to be the same as the class name, and Fortran will require terminating statements for each command as seing in languages such as Java, and C.

2.2. Fortran: Reflection. Fortran doesn't really have any scope as consquence, you were planning to build a full operational programme all the variables can be accesssed from anywhere therefore, unattended side affects may occur in the excution of the programme as the variables can be modified by anything as they're not protected by scope. Therefore, this is going to make it harder to debugg the current programme, as the programmer would have to have full knowledge of the programme as a hole instead of knowledged of the current scopes. As a consqunce, this is going to make Fortran diffucult to strucuture code in the appropriate heirachies hence, violating the *structured programming principle*. Additionally, since they is not really any scope in Fortran this is going to make it hard to hide the implmentations of any subroutines and fucntions therefore violating the *information hiding principle*.

As discussed in the previous section the Fortran complier is not helpful as it doesn't warn the programmer of common programming mistakes. Therefore, one small mistak made by the programme can result of the built programme not running as intetended. The complier in modern languages is more of a helping guide in writing correct and relabile code therefore, since Fortran complier doesn't warn the programmer about these mistakes this makes the programmes exprience less reliable violiting the *reliability principle*.

In relation to Fortran violating the *reliability principle*, the complier will see all commands and variables as upper cases hence, making the language case insensitive. Further adding on onto the unrealiblity of the Fortran programming language.

As discussed previously Fortran variable names will have to be between one to six characters long. The principle of *zero one ifinity* outlines thtat the only valid length of anything is going to be either one, zero or ifinity. As can be seeing with the variable length names Fortran clearly violates the *zero one ifinity* principle.

talk about how it violates the readibility principle

2.3. Fortran: Testing and compiling. you will actually need to go and do this at uni

2.3.1. Results from running programme. **you will actually need to go and do this at uni**

3. Algo68

3.1. Algo 68: Fizzbuzz activation record. Don't forget a picture of it here dawg

3.2. Algo 68: Reflectoin. Writting the fizz buzz programme with Algo 68 was more plesant than Fortran. Algo 68 is starting to represent the langauges which we're more accostumed too. Alog 68 strongly reminds me of the bash scripting language as the constructs are going to end with a word instead of terminating delimeter.

Algo 68 is going to be a language which is going to be scopped hence, the variables which are in a *BEGIN* and *END* block are only going to be found to be accessed within that block, and variables which are going to be defined in any given construct are going to be only accessed within those constructs. Therefore in Algo 68 you're not going to get the side effects whichy you may get in Fortran as the variables are going to be protected by scope therfore, Algo 68 adhering to the *information hiding principle*.

Additionally, due to the introduction of scope, and the introduction of clear termination of constructs, it's a lot easier to look at an Algo 68 programme and get a good idea on what the intededed purpose of the programme is going to be hence, adhering to the *readability principle*. Since, the algo-68 is going to be highly structured this also adheres to the *structured programming*

principle. As a consquence to these observation, wrting a programme in Algo-68 was a lot easier than writing a programme in Fortran.

4. ADA

4.1. ADA: comparison with other buble sorts. A difference between the two lagnauges is going to be the way which they treat fucntions. In C the construction of function which will return nothing or something is going to have the same prototype, ADA will actually will clearly segement these two classes of functions. Functions which are going to return nothing are going to be referred to as *procedures*, and functions which are going to return something are going to be refferred to as *sub-routines*.

A difference between C and ADA is going to be the choice of terminating characters for commands. In C the end of a construct such as a "if and else" statement will need to be terminated withe a semi-colon, and in C the end of a "if and else" statement is going to be terminated with a right parenthesis.

A similarity between C and ADA implementation is going to be found in the positioning of the functions in the file. In C this phenomenon is going to be refferred to as forward declartion whereby, the functions (procedures) have to be declared before the main body of code otherwise both the ADA and C compilier will complain that it doesn't kow what the following funciton is going to be. This idea will also extend to variable decelartoin positioning as well, in C-89 variables will have to be decaled first before any commands are written, and in ADA variables and types will have to be declared before the *begin* key word.

Another similarity point between the ADA and C implementation is going to be the use of pointer manipulation in order to swap any element which is going to be greater than the element which is going to be infront of it as both algorithms will derefrence that memory location to get what is going to be stored their.

Furthermore, ADA and C

4.2. ADA: reflection.