INTRODUCTOPN TO ADA PROGRAMMING LANGUAGE

[Tuesday,14 April]

Aya Mohamad	**1780367**	**EAR**
Wejdan Alzhrani	**1708094**	**CAR**
Hadeel Aloufi	**1805645**	**EAR**
Mona Hafez	**1780735**	**EAR**
Salwa Abbara	**1780293**	**EAR**

Table of Contents

1 Introduction

A programming language is a vocabulary and collection of grammatical rules for instruction to perform specific tasks on a computer The word programming language typically refers to high-level languages. Most programming languages consist of instructions for computers.Instraction is A command that computer hardware understands and obeys. Actually, there is thousands of different programming languages have been created, and more are being created every year. A lot of programming languages are written in an imperative form (a sequence of operations to perform) and the other languages use the declarative form (the desired result is specified, not how to achieve it). Actually there are reasons for studying programming languages.

1.1 Reasons for Studying Concepts of Programming Languages:

- ✓ Increased capacity to express ideas.
- ✓ Improved background for choosing appropriate languages
- ✓ Enhanced ability to learn new languages
- ✓ Good understanding of the significance of implementation.
- ✓ Better use of languages that are already known.
- ✓ Overall advancement of computing. (Sebesta, 2012)

In this report we will talk about Ada high level programing language in term of domains, BNF, writability, readability, reliability, type of variables, arrays, scope, and the implementation method. Ada is object oriented high-level programming language, extended from Pascal and other languages. Ada is a highly advanced programming language that is designed according to the fundamental software engineering principles of efficiency, reliability, portability and maintainability. ADA provides everything from information hiding to abstract data types to concurrent-oriented programming functionality also, Ada is a flexible programing language and has some benefits such as the syntax is very easy to learn, so users can create basic code in just a few minutes. Thus, it is easy for users to follow the program semantics and the connections between ideas, functions and language constructs also, Ada has excellent error handling.

2 Historical overview of Ada language

Department of Defense (DOD) invented the Ada language, so the state of their computing environment was influential in defining its type. By 1974, embedded systems were more than half of DoD computer applications. By using more than 450 different programming languages, the increasing complexity of systems has led to an increase in software costs. So, this excess of language caused application software to seldom be reused. Furthermore, no software development tools have been developed. For these reasons the Army, Navy, and Air Force each suggested the creation of a common high-level language for embedded systems separately in 1974. (Sebesta, 2012).

2.1 Design Process

In January 1975, Malcolm Currie, director of Defense Research and Engineering, formed the High-Order Language Working Group (HOLWG). Its initial charter was to do the following:

• Identify the requirements for a new DoD high-level language.

• Evaluate existing languages to determine whether there was a viable candidate.

• Recommend the implementation of a minimal set of programming languages.

In April 1975, the HOLWG produced the Strawman requirements document for the new language for military branches, federal agencies, selected industrial and university representatives, and interested parties in Europe. This document followed by several submitted proposals for the language until 1978. These proposals were narrowed down to four finalists, which based on Pascal, and finally, in May 1979, the Cii Honeywell/Bull language design proposal led by Jean Ichbiah in France was chosen from those four to be used.(Sebesta, 2012)

In the spring of 1979, Jack Cooper of the Navy Materiel Command called the new language, Ada, after Augusta Ada Byron (1815–1851) who was known as the world's first programmer. She worked with Charles Babbage on his mechanical computers, the Difference and Analytical Engines, writing programs for several numerical processes.(Sebesta, 2012)

The design and the rationale for Ada were published in its SIGPLAN Notices as (ACM, 1979) and were distributed to a readership of more than 10,000 people. A public test and evaluation conference were held in October 1979 in Boston. By November, more than 500 language reports had been received from 15 different countries suggesting just small modifications rather than

drastic changes and outright rejections. Therefore, the Stoneman document was released in February 1980 in response to requirements specification.(Sebesta, 2012)

In July 1980, it was accepted as MIL-STD 1815, which is the year of the birth of Augusta Ada Byron, as the standard Ada Language Reference Manual. In July 1982, this revised version was also released. In 1983, the American National Standards Institute standardized the final official version of Ada that described in Goos and Hartmanis (1983). The Ada language design was then frozen for a minimum of five years.(Sebesta, 2012)

3 The Implementation Method

A programming language implementation is a system for executing computer programs. There are two general approaches to programming language implementation: interpretation and compilation. In Ada language use compiler .

A compiler is a computer program that transforms code written in a high-level programming language into the machine code. It is a program which translates the human-readable code to a language a computer processor understands (binary 1 and 0 bits). The computer processes the machine code to perform the corresponding tasks.

A compiler should comply with the syntax rule of that programming language in which it is written. However, the compiler is only a program and cannot fix errors found in that program. So, if you make a mistake, you need to make changes in the syntax of your program. Otherwise, it will not compile.

3.1 Programming Steps:

1. Create the program.
2. Compile will parse or analyses all of the language statements for its correctness. If incorrect, throws an error
3. If no error, the compiler will convert source code to machine code.
4. It links different code files into a runnable program(know as exe)
5. Run the Program.

3.2 Compiler Properties:

3.2.1 Advantage : The program code is already translated into machine code. Thus, it code execution time is less.

3.2.2 Disadvantage: You can't change the program without going back to the source code.

3.2.3Machine code: Store machine language as machine code on the disk.

3.2.4 Program generation: Generates output program (in the form of exe) which can be run independently from the original program

3.2.5 Execution : Program execution is separate from the compilation. It performed only after the entire output program is compiled.

3.2.6.Code Optimization: The compiler sees the entire code upfront. Hence, they perform lots of optimizations that make code run faster.

3.2.7 Errors: Display all errors after, compilation, all at the same time (guru99,m.d.)

3.3 How to Compile Ada Programs:

We will explain how to use your Ada compiler to create Ada programs.

The basic steps to commanding an Ada compiler are the same for all Ada compilers, but the details (the exact keys you press) vary from compiler to compiler. The basic steps are:

- Install the Ada compiler
- Create a project (directory)
- Create and edit file(s) containing Ada source code
- Compile the source code, creating object code
- Link the object code, creating a program
- Run the resulting program
- (Optionally) Debug the resulting program

For Ada compilation systems with graphical front-ends, every step but the first is usually obvious from the menus. However, if you're not using a graphical front end these steps may not be so obvious. Below are each of these steps, with some specific guidance for the GNAT and Rational VADS compilers when used without a graphical front-end (especially for MS-DOS/MS-Windows and Unix platforms).

Install the compiler. These are very compiler-specific, but installation instructions are included with every compiler. Most operating systems have a ``PATH" that must be correctly set up as part of the installation procedures.

Create a project (directory). Generally it's best to create a separate directory for your Ada files. In MS-DOS and Unix, the command "mkdir" may be used to create a new directory. Rational VADS requires that a directory for Ada files be specially prepared; use the "a.mklib" command to do this.

Create and edit file(s) containing the Ada source code. Use your favorite text editor to create Ada source code files (such as emacs, vi, edit, or Brief).

Rational VADS requires all Ada filenames to end with ".a"; I suggest naming files in the form hello_spec.a and hello_body.a.

GNAT is very picky about filenames. Filenames must have the same name as the compilation unit before the period. Also, filenames must end in ".ads" if they are specifications and ".adb" if they are bodies.

No matter what compiler you're using, use a separate file for each different compilation unit (GNAT requires this).

Compile.

Rational VADS' compilation command is "ada FILENAME"

GNAT's compilation command is "gcc -c FILENAME"

Link.

Rational VADS' link command is:

ada -M main_program_name -o object_file_name

GNAT's compilation command is:

gnatbl -o object_file_namemain_program_name.ali

Run. This is generally done by typing in the name of the object file created.

(Optionally) Debug. Rational VADS' debugger is called "a.db". GNAT's debugger is the normal gcc debugger "gdb".

Hopefully, at this point you can compile simple programs using by compiler.

(gedlc.ulpgc.es,m.d.)

4 Domains

Ada has a lot of successes documented internationally and that indicates that it has a lot of uses And we have a lot of information for the domains in ADA which are important uses in government projects (Ministry of Defense), medical aids, regulations, education, manufacturing, aviation, space projects and transport control system.

4.1 Used in Many Government Projects (The Ministry of Defense)

All military systems and applications are real-time. It is worth noting that real time is a term that means that it controls the environment by receiving and processing data and returning results quickly enough to influence the environment at that time.(Martin, 1965) Such as: robots to fight in war, Land and Naval Surveillance and Defense Systems and Missile systems.

4.2 Used in Medical Means

It is well known that Ada language is a language dependent on precision, so it is good choice for continuous real-time medical monitoring systems.(Feldman, 2014)

4.3 Banking Systems

It has many systems like:

1.Payroll systems.

2.Commercial banking systems.

3.Stock price transactions systems.

4.Language translation system.

5. Database management system.(Feldman, 2014)

4.4 Commercial Aviation and Military Aviation.

It has many systems like:

1. Air traffic control systems.

2. Onboard detection and guidance systems.

3. Flight training simulation.

4. Flight control / Flight display systems.(Feldman, 2014)

4.5 Education

Ada is a language of instruction for both introductory and advanced computer science courses at multiple universities.

4.6 Manufacturing

It has many systems like:

1.Automated manufacturing systems

2. Automated material handling systems

3. Automatic welding systems

4. Inventory management systems.

5. Smart oven.

4.7 Used in Space Projects

Many projects for space like:

1. Astronomical telescope

2. Lunar CubeSat

The Vermont Technical College has successfully launched the lunar cube satellite into Earth's orbit and will stay for three years to test the systems that will be used. (Ayre.J, Burdekin.O, & Glockner.J, 2013)

3. Argos Satellite Project

They are used in the development of built-in software for a unique, global, satellite-based location and data collection system designed to study and protect the environment.(Thales Selects AdaCore Toolset for Argos Satellite Project, 2011)

4.8 Transport Control System

Railway and subway traffic control systems and merchant ship control systems.

5 BNF

5.1 Identifiers

identifier : := identifier_letter { ["_"](identifier_letter | digit) } .(cui.unige.ch,m.d.)

- Identifiers: begin with letters then have any number of letters, digits, and underscores .
- We cannot use Reserved words as an identifiers.
- Ada is NOT case sensitive:

Example:Hi, hi and HI all refer to the same variable.

- Conventions:
  - Variable names: camel notation (someName)
  - Procedures and functions: camel notation (someProc)
  - Types: capitalized (Integer)
  - Packages: capitalized (Ada.Text_IO)

- Warning: Operating systems can be case sensitive, so watch file names
  - For gnatmake, file name must be same as procedure name, but must be all lower case: hi.adb, not Hi.adb. (radford.edu,m.d.)

5.2 Declaration

Basic_declaration : := type_declaration | subtype_declaration
| object_declaration | number_declaration
| subprogram_declaration |abtract_subprogram_declaration
|package_declaration | renaming_declaration
| exception_declaration | generic_declaration
| generic_instantation . (cui.unige.ch,m.d.)

Example:

A,B : Integer :=0;
C : Integer :=100;
D : Integer ; . (learn.adacore,m.d.)

5.3 Three Kinds of Loops

loop_statement::= [statement_identifier":"]

[(**"while"**condition) | (**"for"**
defining_identifier**"in"** [**"reverse"**] discrete_subtype_definition)]
"loop"
sequence_of_statements
"end""loop" [statement_identifier] **";" .** (cui.unige.ch,m.d.)

5.3.1For loop:

For i in 1..10 loop
…
end loop

5.3.2.While loop

i:=1;
while i<= 10 loop
…
i:= i+1

end loop;

5.3.3 General loop (infinite loop that can have an exit)

i := 1;
loop
exit when i=10;
…
i:= i+1;
end loop;

Loops allow you to have a set of statements repeated over and over again.
Endless Loop

The endless loop is a loop which never ends and the statements inside are repeated forever. Never is meant as a relative term here — if the computer is switched off then even endless loops will end very abruptly.

5.3.4 Endless_Loop : loop

Do_Something;

end loop Endless_Loop;

The loop name (in this case, "Endless_Loop") is an optional feature of Ada. Naming loops is nice for readability but not strictly needed. Loop names are useful though if the program should jump out of an inner loop, see below.

Loop with Condition at the Beginning

This loop has a condition at the beginning. The statements are repeated as long as the condition is

Control

met. If the condition is not met at the very beginning then the statements inside the loop are never executed.

While_Loop :while X <= 5 loop

X := Calculate_Something;

end loop While_Loop;

loop with condition at the end

This loop has a condition at the end and the statements are repeated until the condition is met. Since the check is at the end the statements are at least executed once.

5.3.5 Until_Loop :loop

X := Calculate_Something;

exit Until_Loop when X >5; end loop Until_Loop;

loop with condition in the middle

Sometimes you need to first make a calculation and exit the loop when a certain criterion is met. However when the criterion is not met there is something else to be done. Hence you need a loop where the exit condition is in the middle.

5.3.6 Exit_Loop :loop

X := Calculate_Something;

exit Exit_Loop when X > 5;

```
Do_Something (X);
 end loop  Exit_Loop;
```

In Ada the exit condition can be combined with any other loop statement as well. You can also have more then one exit statement. You can also exit a named outer loop if you have several loops inside each other. (Wikibooks contributors,2004–2007)

5.4 IF-STATEMENT

if_statement selects for execution at most one of the enclosed sequences_of_statements, depending on the (truth) value of one or more corresponding conditions.

- No parens around condition
- then required
- end if; is required, even if body has only a single statement
- keyword elsif
    - allows multiple conditions in single if
    - avoids nesting
    - Can have as many elsif statements as desired
    - else statement optional

Remember: if and for statements have end if; and end for. (adaic.org,m.d.)

Syntax

if_statement ::=

```
   if condition then
     sequence_of_statements
  {elsif condition then
     sequence_of_statements}
  [else
     sequence_of_statements]
   end if;
```

condition ::= boolean_expression . (cui.unige.ch,m.d.)

Examples

```
if Month = December and Day = 31 then
   Month := January;
   Day   := 1;
   Year  := Year + 1;
end if;
if Line_Too_Short then
Layout_Error;
elsif Line_Full then
   New_Line;
   Put(Item);
else   Put(Item);
end if;
```

5.5 Expressions

Expressions may appear in many contexts, within both declarations and statements. Expressions are similar to expressions in most programming languages: they may refer to variables, constants and literals, and they may use any of the value-returning operations. An expression produces a value. Every expression has a type that is known at compile time.

Expressions may appear in many contexts, within both declarations and statements. Expressions are similar to expressions in most programming languages: they may refer to variables, constants and literals, and they may use any of the value-returning operations. An expression produces a value. Every expression has a type that is known at compile time.

expression ::=

relation { **"and"**relation } | relation { **"and""then"**relation }
| relation { **"or"**relation } | relation { **"or""else"**relation }
| relation { **"xor"**relation }

relation ::=

(simple_expression [

("=" | "/=" | "<" | "<=" | ">" | ">=")

simple_expression])

| (simple_expression[**"not"**] **"in"** (range | subtype_mark))

simple_expression ::= [("+" | "-")] term

{ ("+" | "-" | **"&"**) term }

term ::= factor { (**"*"** | "/" | **"mod"** | **"rem"**) factor }

factor ::= (primary [**"**"**primary]) | (**"abs"**primary) | (**"not"**primary). (cui.unige.ch,m.d.)

6 The evaluation of Ada programing language

The design and evaluation of programming languages is a challenging area because there is no such thing as a best language. Instead, existing languages are strong by some criteria and weak by the other. we will talk about some of these criteria like readability, writability, reliability.

6.1 Readability

Readability is a significant determinant of how simple a program is to maintain. You need to be in a position to read and understand what the program does to change it. Ada's impact on readability was partly due to the use of data types, control statements and structures. Additionally, using special words increases this language's readability considerably. Also, Ada prefers full English words rather than abbreviations because that full words be simpler to read. Ada text is dense in keywords, which some mistakenly consider noisy and redundant. In fact, those keywords act as natural punctuation marks, and make program reading more natural. For example, begin-end bracketing is more natural sounding than {…}. In Ada language all variable must be declared on the contrary of lisp and python languages and that also increase the readability. (Louden&Lambert,2011)

6.2 Writability:

The ease with which a language can be used to create programs.

Ada language is so broad and complex it reduce the language's writability. However, there are also many factors which profit Ada's writability. They include abstraction, expressivity and generic procedures. A compiler can produce a version of the generic procedure using a particular data type, thus reducing the amount of code to write.

Ada provides some mechanisms which allow you to write software which is flexible enough that you can reuse it fairly easily in a variety of different situations. Learning to make the most of these mechanisms is always useful but it becomes a necessity as soon as programs start to go above a few thousand lines of code in size. (Johnson,n.d.)

6.3 Reliability:

The performance of a program to its specifications under all conditions.

Ada has a simple syntax, structured control statements, flexible data composition facilities, strong type checking(detects errors more easily in both initial and separate unit compilations), traditional features for code modularization (“subprograms”), and a mechanism for detecting and responding to exceptional run-time conditions (“exception handling”).Ada's exception handling mechanism supports fault-tolerant applications by providing a complete and portable way of detecting and gracefully responding to error conditions.

Ada's semantic foundation is based on enforcing program consistency, safety, and security at compile time if possible, with checks that prevent data-type mismatches, and at run time when necessary, detecting buffer overflow and other problems.(Brosogol,2006)

7 The Variable

Variables are used to store information in a computer program which is to be referenced and manipulated. Also, they provide a way to label data with a descriptive name, so that the reader and ourselves can understand our programs more clearly. It is helpful to think of variables as containers containing information. The sole purpose is to memorize and label data. You can then use this data in your programmed.

7.1 Declaring a Variable

Before using a variable in Ada, you must declare it. The declaration starts with the name of the variable:

- The name of a variable must be in one word
- It must start with a letter
- It can include a combination of letters, digits, and underscores
- It must not contain special characters
- Ada uses some words(keyword) that you must not use to name your variables

We can declare each variable on it is own line or we can declare more than one variable On one line but those variables use the same data type.

If we want to declare one variable in line, we can use this formula:

Variable Name: DataType;

If we want to declare more than one variable have the same type in line, we can use this formula:

VariableName1, VariableName2: DataType1;

7.2 Initializing a Variable

Initializing a variable consists of assigning a value to it before using it. You have two options to initialize a variable when declaring it, after the name of the variable, type := followed by an appropriate value. This would be done as follows:

VariableName :DataType := Value;

The option consists of assigning a value after declaring it. This can be done as follows:

VariableName :DataType

begin

VariableName:= Value;

End

7.3 Primitive Data Types

7.3.1 Characters

A character is a letter, a symbol, or a digit. To declare a variable that can hold a variable, use the **character** keyword. To initialize it, include the value is single-quotes. For example:

gender: character: = 'F';

7.3.2 String

A string is a combination of characters. To represent strings, Ada uses the **String** data

type. When declaring the variable, to initialize it, include its value in double-quotes.

For example:

sentence: String: = "Welcome to the wonderful world of Ada programming!";

7.3.3 Integers

An integer is a numeric value for a natural number. In Ada, an integral value is represented with the **integer** data type.

For example:

number: integer

7.3.4 Floating Point Numbers

To declare a floating-point variable, use the float keyword. To initialize it, assign a number that includes one decimal separator. Here is an example:

value: float;

7.3.5 Constants

A constant is a value that doesn't change. To create a constant, you use the constant keyword as follows:

VariableName : constant DataType := Value;

7.4 Storage of variable and lifetime

Programming languages are usually classified as having either static storage allocation or dynamic storage allocation. Static storage allocation associate's variables with memory locations at compile or link time; dynamic storage allocation associates variables with memory locations at execution time. Ada is a language that supports dynamic storage allocation. The lifetime of a variable is the time during Which it is bound to a particular memory cell. So, the lifetime of a variable begins when it is bound to Specific cell and end when it is unbound from that cell. (Dale& McCormick, 2006)

8 Type of arrays

8.1 simple array

Ada actually uses the keyword array to denote the declaration of an array. then it specifies the index range for the array in parentheses, in this case 1000 elements having indexes 1 to 1000 and the type of the array is integer

A: array (1..1000) ofinteger;

8.2 multidimensional Arrays

B :array (1..4, 1..5) of float;

B is a two-dimensional array of float. Elements are accessed using B(i,j).

8.3 Array type

typeMy_Array is array(Index) of My_Int;

where

My_Int_Array: is the name of the array

My_Int:typeMy_Int is range 0 .. 1000;

Index: type Index is range 1 .. 5;

My_Array is an array and My_Int is the Type of elements also,we declared an integer type named Index ranging from 1 to 5, so each array instance will have 5 elements, with the initial element at index1 and the last element at index 5. If we need to access element in the array we uses the same syntax as for function calls: that is, the array object followed by the index in parentheses.

8.4 Array slices

Another feature of Ada arrays is array slices. It is possible to take and use aslice of an array(a contiguous sequence of elements).

str: String:= "Hello ...";

We have here string of character

str(7 ..9) := "Bob";

We need to replace index from 7 to 9 from str string with “BoB” string and the str becomes “Hello Bob”.

9 scope

9.1 Type of scope (static or dynamic):

The scope of a variable x is the region of the program in which uses of x refers to its declaration. One of the basic reasons of scoping is to keep variables in various parts of program distinct from one another.

Scoping is generally divided into two classes:

1.Static Scoping

2.Dynamic Scoping

Static scoping is sometimes called lexical scoping. In this scoping a variable always refers to its top level environment. (Geeks for Geeks,n.d.)Ada language is a static scope language, show the example below.

```
procedure S is
 X: Integer:= 1;//line1
 procedure N is//line2
    X: Integer := 2;//line3
  begin//line4
......>>>>>> if there x here where should be the reference of this x

    Put(X);     -- writes 2//line7
    Put(S.X);   -- writes 1//line8
  end N;//line9
......>>>>>> if there x here where should be the reference of this x
end S;//line11
```

(Scope,n.d.)

Let consider we have variable X at line 5 in N so the reference to this variable is the local variable X declared in N because the declaration of X in N make the global variable X in procedure S to be hidden and if we need to access this variable we can write ancestor scope's name (dot)the variable name S.X. What if there is no declaration for X in N so the reference to X in line 5 will be to its static parent S.Let consider we have variable X at line 10 in S so the reference to this variable is the X declared in S. In Ada the scope of global variable is visible anyplace in the range of its scope, but if another variable with the same name is defined in any subprogram it will hide this global variable. It should be clear that the scope of the local variable extends to the end of the executable portion of the subprogram in which it is declared.

Conclusion

We discuss in the report about the high-level programming language in the Ada system invented by the Ministry of Defense (DOD), then in terms of areas used in many government projects (the Ministry of Defense), used in medical means, banking services, commercial systems and military aviation education, manufacture and use in space projects and transportation control system. It includes some of these important criteria for any language that is evaluated because there is no better programming language or worse programming language and such criteria as readability and writability and reliability which Ada is famous for its readability because it is based on English words not symbols also Ada language is so broad and complex it reduce the language's writability. The reliability of Ada is very high because it has a simple syntax, structured control statements, flexible data composition facilities, strong type checking so that it can detect errors more easily. Ada language has primitive data types such as characters, string, integers, floating point numbers and constants. Storage of variable and lifetime in Ada language are static storage allocation or dynamic storage allocation. We discuss many types of array are simple array, multidimensional array, array type and array slices. Ada is a language that supports a static scoping.

References

1-Adaic(n.d.). Ada Refernce Manual. Retrieved from https://www.adaic.org/resources/add_content/standards/05rm/html/RM-5-3.html

2-Ayre.J & Burdekin.O & Glockner.J (2013, November 19). "AdaCore and Altran Toolsets Help Launch CubeSat into Orbit".AdaCore.from https://www.adacore.com/press/cubesat

3- Brosgol, B. M. (2006). Ada 2005: a language for high-integrity applications. CrossTalk–The Journal of Defense Systems, 19(8), 8-11.

4-Cui.unige.ch (n.d.).BNF of the Ada Programming Language Retrieved from http://cui.unige.ch/isi/bnf/Ada95/BNFindex.html

5- Dale, N. & McCormick, J. (2006). ADA Plus Data Structures: An Object-oriented Approach. Jones & Bartlett Learning; 2ed.P.900. ISBN-10: 0763737941

6-Feldman, M. "Who's using Ada?". SIGAda Education Working Group.

7-Gedlc(n.d.). How to Compile Ada Programs. Retrieved from http://www.gedlc.ulpgc.es/docencia/mp_i/Tutor/Ada/how2comp.htm

8- Geeks for Geeks. (n.d.). Static and Dynamic Scoping. Retrieved from https://www.geeksforgeeks.org/static-and-dynamic-scoping/

9- Guru99 (n.d.). Compiler vs Interpreter:Complete Difference Between compiler and Interpreter .Retrieved from https://www.guru99.com/difference-compiler-vs-interpreter.html

10- Johnson, et al., (n.d.). Retrieved from http://l.web.umkc.edu/lz74d/CS441/Homework3.htm

11- Learn.adacore (n.d.). Statements,Declaration,andControl Structures. Retrieved fromhttps://learn.adacore.com/courses/Ada_For_The_CPP_Java_Developer/chapters/04_Statements_Declarations_and_Control_Structures.html

12- Louden, k. & Lambert, K. (2011). Programming Languages: Principles and Practices - 3rd Edition. Hardcover Book. Course Technology. P. 662.

12-Martin, J (1965). Programming Real-time Computer Systems. Englewood Cliffs, NJ: Prentice-Hall Inc. p. 4. ISBN 978-0-13-730507-0.

14-Radford(n.d.).AdaLanguageFundamentals.Retrievedfrom https://www.radford.edu/~itec320/2018fall-nokie/fundamentals/fundamentals.html

15- Scope. (n.d.). Retrieved from https://cs.lmu.edu/~ray/notes/scope/

16-Sebesta, RW. (2012). Concepts of programming languages. Addison Wesley, 10th Edition

17-Thales Selects AdaCore Toolset for Argos Satellite Project.(2011,May 2).AdaCore.from https://www.adacore.com/press/thales-argos-satellite