**KIMANZI LOICE KYENI  
COM/0023/18**

**CSC 324**

**PRINCIPLES OF PROGRAMMING**

**CAT 1 TAKE AWAY**

**How does the execution of a machine code program on a von Neumann computer architecture occur? (3marks)**

A Von Neumann architecture computer executes a single instructions at a time and each instruction operates on only a few pieces of data. Each instruction in a Von Neumann computer architecture undergoes four steps which are: fetch process, decode process, execute process and store process. During execution of machine code program, instructions which are stored in the memory as binary are fetched from one at a time and in series, the processor then decodes and executes an instruction before cycling around to fetch next instruction the cycle continues until there is no more instructions available.

**Explain TWO programming language deficiencies that were discovered as a result of the research in software development in the 1970’s? (2marks)**

Incomplete type checking -The researcher saw that there is a fault in the type checker we cannot convert one data type into the other data type also it gives error in converting the one data type into another.

Inadequacy of control statement-The control statement plays an important role in the programming language. When the condition of the multiple arises then in that condition we used the control statement in 1970 there is deficiencies of control statement

**Some programming languages such as Pascal have used the semicolon to separate statements while Java uses it to terminate statements. Which of these, in your opinion, is most natural and least likely to result in syntax errors? Support your answer. (3marks)**

Pascal.

Pascal usage of semicolon to separate statements is rather counter intuitive but possibly it’s because in I learned usage of semicolon to separate statements before I had knowledge about Pascal

In Pascal one needs to identify whether or not the next line is a statement before putting a semicolon hence requiring the programmer to go back and change code before adding a new code. Suppose I have a simple program in Pascal which is a if statement

If x=0 then

Answer is 1

In Pascal if you want to add an else clause you would have to go back and delete the semicolon because an else clause is not considered as a new statements. Sometimes Pascal compilers catches wrongly placed semicolons and declares them as errors.so usage of semicolon in Pascal relatively may result to errors than in java.

**Languages continually evolve. What sort of restrictions do you do you think are appropriate for changes in programming languages? Compare your answers with evolution of java? (3marks)**

The development of each programming language is based on a fact: there is a need to solve a problem that was not resolved by previous programming languages. Early programmers had to choose different programming languages, usually for various tasks, such as a specific language for a type of field. A certain language was sufficient to solve the problems of its field but was not able to solve the problems of other fields.

A good deal of restraint must be used in revising programming languages. The greatest danger is that the revision process will continually add new features, so that the language grows more and more complex. Compounding the problem is the reluctance, because of existing software, to remove obsolete features. Java application will run identically on any computer, regardless of hardware features or operating system, as long as it has a Java interpreter

In comparison with Java:

In June 1991,[James Gosling](https://www.javatpoint.com/james-gosling-father-of-java), Mike Sheridan, and Patrick Naughton initiated the Java language project. The small team of sun engineers called Green Team.

Initially it was designed for small, [embedded systems](https://www.javatpoint.com/embedded-system-tutorial) in electronic appliances like set-top boxes.

Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.

After that, it was called **Oak** and was developed as a part of the Green project.

In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies

Initially developed by James Gosling at [Sun Microsystems](https://www.javatpoint.com/sun-microsystems) (which is now a subsidiary of Oracle Corporation) and released in 1995.

In 1995, Time magazine called **Java one of the Ten Best Products of 1995.**

JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

**Describe two levels of use of operational semantics. Why can machine languages not be used to define statements in operational semantics? (4marks)**

Operational semantics is a category of programming language semantics in which certain desired properties of a program such as correctness, safety are verified by constructing proofs from logical statements about its execution and procedures rather than attaching mathematical meanings to its terms.

Operational semantics are classified into two levels:

1. Structural operational semantics

Also called small-step semantics which describes the individual steps of a computation taking place in a computer-based system.

2. Natural operational semantics

Also called big-step semantics and they describe how the overall results of the executions are obtained.

Machine languages cannot be used to define statements in operational semantics because the steps involved in execution of machine language and the resulting changes to the state of the machine are too small and too numerous .in addition the storage of a real complex computer is too large and complex.

**Why is it useful for a programmer to have some background in language design, even though he/she may never actually design a programming language? (3marks)**

Because understanding the basics of programming language:

1. Makes it easier to learn and adopt new languages
2. It also helps to design and develop better software
3. It increases capacity to express ideas
4. Improves background for choosing languages
5. Better understanding of the significance of implementation.

**Conventionally, all languages developments require mastery on grammar or syntax .In which form is the programming language syntax commonly described? (4marks)**

Grammar form.

Grammars are commonly used to describe the syntax of programming languages and it is a syntax description formalism that became most widely used method for programming language syntax

**Using the grammar, show the parse tree and a leftmost derivation for each of the following statements: A=A\*(B+(C\*A))(4marks)**

**s**

**Parse tree:**

<assign>

<id> **=** <expr>

E <id> \* <expr>

A<id> + <expr>

B <expr>

<id> \* <expr>

<id>

C A

**Leftmost Derivation:**  
<assign> -> <id> = <expr>  
-> E= <expr>  
-> E = <id> \* <expr>  
-> E= A \* <expr>  
-> E = A \* (<expr>)  
-> E = A \* (<id> + <expr>)  
-> E = A \* (B + <expr>)  
-> E = A \* (B + (<expr>))  
-> E = A \* (B + (<id> \* <expr>))  
-> E = A \* (B + ( C \* <expr>))  
-> E= A \* (B + ( C \* <id>))  
-> E = A \* (B + (C \* A))

**Discuss any three criteria that is used when evaluating programming language and differentiate between total correctness and partial correctness. (4marks)**

**Writability-** this is a measure of how easily language can be used to code. Writability has following characteristics:

* Simplicity-a programming language should not involve complex syntax, many ways to perform a single task, overloading of methods and operators.
* Orthogonality-meaning a language should have small set of primitives that a construct can combine.
* Abstraction-a language should support data abstraction.

**Readability**-a programming language should be easy to code with and clear to understand. Readability also include following characteristics:

* Simplicity-should not involve complex syntax, many ways to perform a single task and overloading methods and operators.
* Orthogonality-a language should have relatively small set of primitives that constructs can combine.
* Syntax design-this affects readability through identifiers forms and special words.

**Cost**-is the total cost of a programming language including cost of trainer, cost of writing algorithm, cost of maintenance, cost of compiling the language and cost of hardware required for the language should be minimal.

**Write a simple assignment statement with one arithmetic operator in some language you know. For each component of the statement, list the various bindings that are required to determine the semantics when the statement is executed. For each binding, indicate the binding time used for the language? (5marks)**

Language C ++

int count;   
count = count\*3;

**Types are likely to be calculated**: the language specified at design time.

**Type count**: Linked at compile time.

**Set of possible values ​​of the count**: Tied at the time of compiler design.

**Value count**: Tied at execution time with this statement.

**The set of possible meanings for the operator symbol** **“”:**\* Tied on a definition of the language.

**\* The purpose of the operator symbol “” in this statement**: Linked at compile time.

**Internal representation of the literal “3**”: Tied at the time of compiler design.

**Discuss the key paradigms of Programming and show why they are necessary part of the solutions to various problem domains? (5marks)**

***Imperative programming paradigm***

* The imperative programming paradigm assumes that the computer can maintain through environments of variables any changes in a computation process. Computations are performed through a guided sequence of steps, in which these variables are referred to or changed. The order of the steps is crucial, because a given step will have different consequences depending on the current values of variables when the step is executed.
* Imperative Languages  
  Popular programming languages are imperative more often than they are any other paradigm studies in this course. There are two reasons for such popularity:
  1. the imperative paradigm most closely resembles the actual machine itself, so the programmer is much closer to the machine;
  2. Because of such closeness, the imperative paradigm was the only one efficient enough for widespread use until recently.

***Logical programming paradigm***

The Logical Paradigm takes a declarative approach to problem-solving. Various logical assertions about a situation are made, establishing all known facts. Then queries are made. The role of the computer becomes maintaining data and logical deduction.  
A logical program is divided into three sections:

* 1. a series of definitions/declarations that define the problem domain
  2. statements of relevant facts
  3. statement of goals in the form of a query

***Functional programming paradigm***

* The Functional Programming paradigm views all subprograms as functions in the mathematical sense-informally, they take in arguments and return a single solution. The solution returned is based entirely on the input, and the time at which a function is called has no relevance. The computational model is therefore one of function application and reduction.
* Functional languages are created based on the functional paradigm. Such languages permit functional solutions to problems by permitting a programmer to treat functions as first-class objects (they can be treated as data, assumed to have the value of what they return; therefore, they can be passed to other functions as arguments or returned from functions).

***Object-oriented programming paradigm***

* Object Oriented Programming (OOP) is a paradigm in which real-world objects are each viewed as seperate entities having their own state which is modified only by built in procedures, called methods.  
    
  Because objects operate independently, they are encapsulated into modules which contain both local environments and methods. Communication with an object is done by message passing.  
    
  Objects are organized into classes, from which they inherit methods and equivalent variables. The object-oriented paradigm provides key benefits of reusable code and code extensibility.
* *Features & Benefits*  
  A new class (called a *derived class* or *subclass*) may be *derived* from another class (called a *base class* or *superclass*) by a mechanism called inheritance. The derived class *inherits* all the features of the base class: its structure and behavior(response to messages). In addition, the derived class may contain *additional* state (instance variables), and may exhibit additional behavior (new methods to resond to new messages). Significantly, the derived class can also *override* behavior corresponding to some of the methods of the base class: there would be a different *method* to respond to the same message. Also, the inheritance mechanism is allowed even *without access to the source code of the base class*.
* The ability to use inheritance is the single most distinguishing feature of the OOP paradigm. Inheritance gives OOP its chief benefit over other programming paradigms - relatively easy *code reuse* and *extension* without the need to change existing source code.  
    
  The mechanism of modeling a program as a collection of objects of various classes, and furthermore describing many classes as extensions or modifications of other classes, provides a high degree of modularity.  
    
  Ideally, the state of an object is manipulated and accessed only by that object's methods. (Most O-O languages allow direct manipulation of the state, but such access is stylistically discouraged). In this way, a class' *interface* (how objects of that class are accessed) is separate from the class' *implementation* (the actual code of the class' methods). Thus *encapsulation* and *information hiding* are inherent benefits of OOP.