AL-HIKMAH UNIVERSITY, ILORIN, NIGERIA
Adeta Road, Adewole Housing Estate, P.M.B. 1601, Ilorin
.....learning for wisdom and morality.....
CENTRE FOR ICT AND DISTANCE LEARNING (IDL)
e-CONTENT DEVELOPMENT (DL) UNIT

e-Note for CMP 332

A. Course Lecturer's Details
Names: OYELAKIN, A. M. & SALAU-IBRAHIM, T.T.
Mobile Number: +2348062738520, +2348034825810
Official Email Address: amoyelakin@alhikmah.edu.ng, ttsalau@alhikmah.edu.ng,
B. Faculty Department and Programme
Faculty: Natural and Applied Sciences
Department: Computer Science
Programme: B.Sc. Computer Science
C. Course Title: Survey of Programming Language
D. Course Credit: 03
E. Course Description

Overview of programming languages: History of programming languages, Brief survey of programming paradigms (procedural languages, Object-oriented languages, Functional languages, Declarative-non-algorithmic languages, scripting languages). The effects of scale on programming methodology; Languages Description: Syntactic structure (Expression notations, abstract Syntax Tree, Lexical Syntax, Grammars for Expression, variants of Grammars), Language semantics (Informal semantics, Overview of formal semantics, Denotation semantics, Axiomatic semantics, Operational semantics); Declarations and types: The concept of types, Declaration models (binding, visibility, scope, and lifetime), Overview of type-checking, Garbage collection; Abstraction mechanisms: procedures, function, and iteration as abstraction mechanism, parameterization mechanisms

F. Learning Objectives

At the completion of this e-note, it s expected that learners will achieve the following:

1. Learn some basic concepts on the evolution of programming languages
2. Be familiar with language definition structure
3. Identify different data types supported by selected programming languages
4. Review of implementation approaches of the data types in some languages (C, Java, Python and VBscript)
5. Know the basic design issues in compiler construction for some selected languages

6. Understand program translation processes
7. Be able to write and differentiate the various ways of writing programs in C, Java, Python and VBscript languages.

G. Notes on Topics to be taught

Important Notice

Based on the fact that there are several topics laid out in the NUC BMAS on this course, we have re-grouped the topics in this material into three sections as necessary. It is believed that this approach will facilitate the learning of the concepts in the course more easily. We enjoin you to follow the discussions on the course very carefully and then try to attempt some of the practice questions contained therein in the study question section. We wish you a fulfilling learning experience!

SECTION A: HISTORY AND BASIC CONCEPTS IN PROGRAMMING LANGUAGES

Computer Programming: A General Introduction

Computer programming is the craft of writing a set of instructions that can later be compiled and/or interpreted and then inherently transformed to an executable that computer system or other an electronic machine can execute. A programmer is expected to master the syntax and semantics of the programming language while at the same time understand the logic require for each problem. Programming languages are artificial languages while human languages are natural.

A programming language can be described by the combination of its semantics, syntax and pragmatic. Syntax is concerned with how expressions, command declaration and other constructs must be arranged to make a well-formed program. Semantics is concerned with the meaning of programs and how a well-formed program is expected to behave when executed. Pragmatics is concerned with the way in which the language is intended to be used in practice.

Programming is a very essential aspect of computing, because in it lies the power of innovative thinking that can aid in developing rich software solutions. To be able to program well, one needs to have passion for problem solving and continuous learning. A program is a set of instructions that tell the computer to do various things; sometimes the instruction it has to perform depends on what happened when it performed a previous instruction.

Reasons for Studying the Concepts of Programming Language

The reasons for studying the concepts of programming languages are as follows:

1. **To improve your ability to develop effective algorithms**

Having a basic knowledge of the principles and implementation techniques of programming languages allows the programmer to understand the cost of techniques in one language compared to another. For example to use concepts like OOP, logic programming, concurrent programming requires an understanding of languages that implement the concepts.

2. **To improve your use of existing programming languages**

When the features of a programming language is well understood, the ability to write efficient programs will increase.

3. **To increase your vocabulary of useful programming constructs**

By studying different programming language constructs, a programmer increases his programming vocabulary. Therefore the understanding of implementation techniques is important so that when one language does not provide a construct, the programmer can easily think of other programming languages that can be used.

4. **To allow better choice of programming languages**

Programming languages are many and there is a need to identify the features of each language while solving problem. A knowledge of different languages allows making just the right choice for a project at hand. It would be easier to know language best suited for mathematical application, AI application or business application as the case may be.

5. **To make it easier to learn a new language**

Having a sound knowledge of variety of programming language constructs and implementation techniques makes it easier for programmer to learn a new programming language easily.

6. **To make it easier to design a new language**

It would be easier to device better approaches for designing new languages if the existing languages are well studied to know its strengths and weaknesses.

7. **Advancement in computing**

Studying programming languages leads to an overall advancement in computing because state of the art application can be easily designed to solve current and challenging problems.

<u>Assignment</u>

Conduct a research on the on the history of programming language evolution from 1960s till date.

Roles of Programming Languages

Programming languages evolve and eventually pass out of use. Algo from 1960 is no longer used and was replaced by Pascal which in turn was replaced by C++ and Java. In addition, the older languages still is used have undergone periodic revisions to reflect changing influence from other areas of computing. As result, newer languages reflect a composite of experiences gain in the design and use of older languages. Having stated that, the roles of programming language are:-

1. **To improve computer capability**

Computer have evolve from the small, old and costly vacuum tubes machine of the 1950`s to the super and microcomputers of today. At the same time layer of OS software have been inserted between the programming language and they underlying computer hardware. This factors have influence both the structure and cost of using the features of high-level language.

2. **Improved application**

Computer use has moved rapidly from the original concentration on military, scientific business and industrial application where the cost could have been justified to computer games, artificial intelligence, robotics, machine learning and application in learning of every human activity. The requirement of these new application areas influence the designs of new language and the revision and extension of older ones.

3. **Improved programming method**

Language designs have evolved to reflect our changing understanding of good method for writing large and complex programs. It has also reflected the changing environment in which programming is done.

4. **Improved implementation method**

The development of better implementation method has affected the change of features in the new languages.

5. **Standardization**

The development of language that can be implemented easily on variety of computer system has made it easy for programs to be transported from one computer to another. This has provided a strong conservative influence on the evolution of language designs.

Attributes of a Good Program

Despite the importance of external influence, the programmer ultimately determines which language stand the test of time or which language is in high demand. Many reasons explain why programmer prefers one language to another some of the reasons are:

1. Clarity, simplicity and unity
2. Orthogonality
3. Naturalness is the anticipation
4. Support of abstraction
5. Ease of problem verification
6. Programming environment
7. Probability of program

1. **Clarity, simplicity and unity**

A good programming language should be an aid to the programmer by providing a clear, simple and unified set of concepts that can be used as primitives in developing algorithms. As for simplicity, a good programming language should have property in which constructs that mean different things look different, that is semantic difference should be clear in the language syntax

2. Orthogonality

This refers to the attribute of being able to combine various features of a language in all possible combinations, with every combination being meaningful. Orthogonality feature makes it easier to learn a programming language it also makes it easier to write program because there are fewer exceptions.

3. Naturalness for the application

A good programming language should provide appropriate data structures, operations, control structures, and a natural syntax for the problem to be solved.

4. Support for abstraction

An ideal programming language should allow data structures, data types and operations to be defined and maintained as self-contained abstractions.

5. Ease of program verification

The process of verifying that a program is correct should be easy because it is of great concern to the reliability of the program. Ease of verification is made easier by the simplicity of semantic and syntactic structure.

6. Programming environment

The availability of a reliable, efficient and well-documented implementation of a language is important. This ensures that the speed of creation, testing, maintaining and modifying large programs much simpler.

7. **Portability of programs**

Good programming language should have the attributes of transportability. That is, it should be easy to move program projects from the computer on which they are developed to other computer systems. In order words, the definitions of the program should be independent of the features of a particular machine.

Taxonomy of Programming Languages

Programming languages are classified into five (5) generations based on the level of abstraction.

1. Low Level Language
2. High Level Language
3. Middle Level Language
4. Very High Level Language
5. Higher Level Language

1. **Low Level Languages**

Low-level languages are written in the language that the computer understands i.e Zeros (0's) and ones (1's). Such languages have little or no abstraction between them and the machine language. Hence, compilers and interpreters are not required. The use of low-level language requires in-depth knowledge of computer architecture. The two types of languages in this category are Machine and Assembly language. Machine language uses binary digits 0"s and 1's while Assembly language uses alphanumeric symbols called Mnemonics. Machine language is the first generation computer languages while Assembly language is a 2nd generation language

Features of Low Level Languages

a. **Fast processing:** All the computer instructions are directly recognized. Hence, programs run very quickly.
b. **No need of Translator:** The machine code is the natural language directly understood by the computer. Therefore, translators are not required. Assembly language on the other hand requires assemblers for conversion from mnemonics to equivalent machine code.

c. **Error prone:** The instructions are written using 0's and 1's. This makes it a herculean task. Hence, there are more chances of error prone codes in these languages. Errors are reduced in assembly language when compared to machine language.
d. **Difficult to use:** The binary digits (0 and 1) are used to represent the complete data or instructions, so it is a tough task to memorize all the machine codes for human beings. Assembly language is easier to use when compared to machine language.
e. **Difficulty in debugging:**When there is a mistake within the logic of the program then it is difficultto find out the error (bug) and debug the machine language program.
f. **Difficult to understand:**It is very difficult to understand because it requires a substantial knowledge of machine code for different system architectures.
g. **Lack of portability:**They are computer dependent such that a program which is written for one computer cannot be run on different computer because every computer has unique architecture.
h. In-depth Knowledge of Computer Architecture: To program in the Assembly languages, the programmer need to understand the computer architecture as for asthe machine languages.

2. Middle-Level Languages

Middle-level languages bridge the gap between high-level andlow-level languages. Examples of these languages include C#, C++,Java, and FORTH. They are used for designing graphical user interfaces (GUIs) on personal computers. Middle-level languages are closer to human language when compared to low-level languages.

Features of Middle-Level Languages

a. They have the feature of accessing memory directly using pointers.
b. They use system registers for fast processing
c. They support high-level language features such as user friendly nature;

3. High-level Languages

High-level languages allow complex sequences of processor instructions. They are mainly written in format close to English language that can be easily understood by the human being. Being written in human language however, leads to the use of compilers and interpreter that can convert into the machine language equivalent. This category of languages are used for writing large pieces of software, like web browsers, word processors, computer games, audio video players, or a Financial Management system.

Features of High-Level languages

a. **Third generation Language:** High Level Languages are 3rd Generation Languages that are majorly refinement to the 2nd generation languages.
b. **Understandability:**programs written in High-level language is easily understood
c. **Debugging:** Errors can be easily located in theses languages because of the presence of standard error handling features.
d. **Portability:** programs written on one computer architecture can run on another computer. This also means that they are machine independent.
e. **Easy to Use:** Since they are written in English like statements, they are easy to use.
f. **Problem-oriented Languages:** High level languages are problem oriented such that they can be used for specific problems to be solved.
g. **Easy Maintenance:** We can easily maintain the programs written in these languages.
h. **Translator Required**: There is always a need to translate the high-level language program to machine language.

Higher Level programming languages

These are fifth Generation Languages (5GLs). Fifth Generation systems (5GSs) are characterized by large scale parallel processing (many instructions being executed simultaneously), different memory organizations, and novel hardware operations predominantly designed for symbol manipulation. Popular example of 5GL is prolog. These type of languages requires design of interface between human being and computer to permit affective use of natural language and images.

Translator used by Programming Languages

Programming languages are artificial languages and they use translators for the computer system to convert them to usable forms. There are three levels of programming languages: They are Machine Language (ML), Low Level Language (al known as Assembly Language) and High Level Language. The translators used by these languages differ and they have been classified into: Compiler, Interpreter or Assembler. Machine Language does not use translator

Low level language uses a translator called Assembler. An Assembler converts program written in Low Level Language (Assembly Lang) to machine code.High Level Language uses compiler, interpreter or both.Our emphasis in this course is on some selected High Level Languages. Some examples of HLL include C, C++, Delphi, PASCAL, FORTRAN, Scala, Python, PERL, Delphi, QBASIC and so on. The language that will be considered mostly are: C, Java, Python and VBscript.

General Comments on Some High Level Programming Languages

i. C is a compiled procedural, imperative programming language made popular as the basis of Unix. The language is very popular and is found useful in building mathematical and scientific applications.

ii. C++ is a compiled programming language that is based on C, with support for object-oriented programming. It is one of the most widely-used programming languages currently available. It is often considered to be the industry-standard language of game development, but is also very often used to write other types of computer software applications.

iii. C# is an object-oriented programming language developed by Microsoft as part of their .NET initiative, and later approved as a standard by ECMA and ISO. C# has a procedural, object oriented syntax based on C++ that includes aspects of several other programming languages (most notably Delphi, Visual Basic, and Java) with a particular emphasis on simplification.

iv. FORTRAN is a general-purpose, procedural, imperative programming language that is especially suited to numeric computation and scientific computing. Originally developed by International Business Machines Corporation (IBM) in the 1950s for scientific and engineering applications. FORTRAN has come in different versions over the years. We have FORTRAN II, FORTRAN 98 and so on.

v. VBScript(Visual Basic Script) is developed by Microsoft with the intention of developing dynamic web pages. It is client-side scripting language like JavaScript. VBScript is a light version of Microsoft Visual Basic. The syntax of VBScript is very similar to that of Visual Basic. The light language allows you to make your webpage to be more lively and interactive, then you can incorporate VBScript in your code.

vi. Java is an object oriented interpreted programming language. It has gained popularity in the past few years for its ability to be run on many platforms, including Microsoft Windows, Linux, Mac OS, and other systems. It was developed by Sun Microsystems.

vii. Pascal is a general-purpose structured language named after the famous mathematician and philosopher Blaise Pascal.

viii. BASIC (Beginner's All purpose Symbolic Instruction Code) was mostly used when microcomputers first hit the market, in the 1970s. Later, it largely replaced by other languages such as C, Pascal and Java. Whilst some commercial applications have been written in BASIC it has typically been seen as a language for learning/teaching programming rather than as a language for serious development.

ix. PHP is a programming language with focus on web design and a C-like syntax.

Programming Language Paradigm

Different programming languages support different styles of programming approach (called programming paradigms). Part of the skill that a programmer should be well grounded in is selecting a programming language or combination of languages that is best suited for the task at hand. Different programming languages require different levels of detail to be handled by the programmer when implementing algorithms, often in a compromise between ease of use and performance. Programming paradigms are a way to classify programming languages based on their features. A particular language can be classified into multiple paradigms.Some programming paradigms are concerned mainly with implications for the execution model of the language, such as allowing side effects and whether the sequence of operations is defined by the execution model. Other paradigms are concerned mainly with the way code is organized. Either by grouping a code into units along with the state that is modified by the code. Lastly, others are concerned mainly with the style of syntax and grammar.

Types of programming paradigms

1) Imperative language
2) Functional language
3) Object-oriented language
4) Concurrent languages
5) Logic programming language

6) Scripting programming language.

Note: It is important to point out that some programming languages support multiple paradigms. For instance, Python and Java are in this category. These two languages are predominantly object oriented in nature but they have support for some other programming paradigms to some extent. Python is a High Level Programming Language that supports imperative, object oriented and functional programming paradigms.

1. Imperative Languages

Imperative programming is called so because it is based on commands that updatevariables held in storage. In the 1950s, the first programming language designers recognized that variables and assignment commands constitute a simple but useful abstraction from the memory fetch and update of computers' instruction sets. This close relationship with computer architecture has largely led to efficient implementation. Languages that fall into the imperative paradigm have two main features: they state the order in which operations occur, with constructs that explicitly control that order and they allow side effects. In which state can be modified at one point in time, within one unit of code, and then later read at a different point in time inside a different unit of code. The communication between the units of code in an imperative language is not explicit. The key concepts of imperative programming languages are

i. Variables
ii. Commands
iii. Procedures
iv. Data abstraction.

Examples of languages in this category include: FORTRAN (FORTRAN 77, FORTRAN 98), ALGOL (ALGOL 58, ALGOL 60), C, ADA and so on.

2. Functional Programming Languages

The model of computation in functional programming languages is the application of functions to arguments. The key concepts of functional programming are:

i. Expressions are key concept of functional programming because their purpose is to compute new values from old ones. This is the very essence of functional programming.

ii. Functions are key concept of functional programming because functions abstract over expressions.
iii. Parametric polymorphism is a key concept of functional programming because it enables a function to operate on values of a family of types. In practice, many useful functions are naturally polymorphic, and parametric polymorphism greatly magnifies the power and expressiveness of a functional language.
iv. Data abstraction is a key concept in the more modern functional languages such as ML and HASKELL. Data abstraction supports separation of important issues, which is essential for the design and implementation of large programs.
v. Lazy evaluation is based on the simple notion that an expression whose value is never used need never be evaluated.

3. Object Oriented Programming Languages

In object-oriented programming, OOP, a program code is organized into objects that contain state that is only modified by the code that is part of the object. Most object-oriented languages are also imperative languages to some extent. Examples of these languages include: Java, C++, Python, Smalltalk, C# and a host of others. The key concepts of OOP are

i. Objects
ii. Classes
iii. Subclasses
iv. Inheritance
v. Inclusion polymorphism.

Object Oriented Programming has been widely accepted in the computing world because objects give a very natural way to model both real-world and cyber-world entities. Classes and class hierarchies lead to highly suitable as well as reusable units for constructing large programs. Object-oriented programming fits well with object-oriented analysis and design and hence supports these development of large software systems.

4. Concurrent Programming Languages

A concurrent programming language is defined as one which uses the concept of simultaneously executing processes or threads of execution as a means of structuring a program. Concurrent programming allows the execution of commands to be overlapped, either by an arbitrary interleaving execution called multiprogramming or by simultaneous execution on multiple CPUs called multiprocessing. The key concepts of concurrent programming are

i. parallel execution of two or more processes: parallel execution is the essential difference between sequential and concurrent programming. Once we allow the ordering of events in the execution of a program to be weaker than total ordering, it leads to consequences such as (i) the possibility that update operations on variables might fail to produce valid results, and (ii) the loss of determinism. These issues amplify each other, and have the potential for complete chaos.
ii. inter-process synchronization: synchronization enables the programmer to ensure that processes interact with each other in an orderly manner, despite these difficulties. Lapses of synchronization are usually disastrous, causing sporadic and irreproducible failures.
iii. synchronized access to shared data by inter-process mutual exclusion: mutual exclusion of access to shared variables restores the semantics of variable access and update. This is achieved by synchronization operations allowing only one process to access a shared variable at any time. These operations are costly, and must be applied with total consistency. It is therefore desirable for high-level concurrent programming languages to offer a reliable, compiler-implemented means of mutual exclusion.
iv. synchronized transfer of data by inter-process communication: communication provides a more general form of interaction between processes, because it applies as well in distributed systems as in centralized systems. Some concurrent programming languages such as CSP (Communicating Sequential Processes) and OCCAM have been based entirely on communication and avoiding shared variables and their problems. Concurrent programming paradigms based on communication might dominate in the future if CPU technology continues to outstrip storage technology. This will ensure that accessing a variable becomes, for all practical purposes, an exercise in data communication. Until then shared data will continue to be preferred, because it capitalizes on and extends a conceptual model.
v. Concurrent control abstractions: concurrent control abstractions promote reliable concurrent programming by taking much of the burden of synchronization into the programming language. The conditional critical region construct is use to provide a high-

level abstraction of both mutual exclusion and communication while monitor construct offers similar advantages and adds data abstraction to the mix.

5. **Logic Programming**

A logic program implements a relation. Since relations are more general than mappings, logic programming is potentially higher-level than imperative or functional programming. The key concepts of logic programming are therefore:

- **i.** Assertions
- **ii.** Horn clauses
- **iii.** Relations

Prolog is an example of a programming language that follows the Logic paradigm. PROLOG's primitive values are numbers and atoms. Atoms can be compared with one another, but have no other properties. They are used to represent real-world objects that are primitive.

6. **Scripting Programming Languages**

A software system often consists of a number of subsystems controlled or connected by a script. Therefore, a script is used to glue subsystems together. One example of gluing is a system designed to create a new user account on a computer but also consist of a script that calls programs to perform the necessary system administration actions. Another example is a system that enables a user to fill a Web form, converts the form data into a database query, transmits the query to a database server, converts the query results into a dynamic Web page, and downloads the latter to the user's computer for display by the Web browser. In the examples given, each subsystem could be a complete program designed to stand alone, or it could be a program unit designed to be part of a larger system, or it could be itself a script. Each subsystem could be written in a different programming or scripting language. Scripting is similar to imperative programming because scripting languages support variables, commands, and procedures. The key concept includes

High-level string processing: All scripting languages provide very high-level support for string processing. The ubiquitous nature of textual data such as e-mail messages, database queries and results and HTML documents necessitated the need for High-level string processing.

High-level graphical user interface support: High-level support for building graphical user interfaces (GUIs) is vital to ensure loose coupling between the GUI and the application code. This is imperative because GUI can evolve rapidly as usability problems are exposed.

Dynamic typing: Many scripting languages are dynamically typed. Scripts must be able to pass data to and from subsystem written in different languages when used as glue. Scripts often process heterogeneous data, whether in forms, databases, spreadsheets, or Web pages.

SECTION B: LANGUAGE DESIGN AND PROGRAM TRANSLATION

Language Definition Structure

While programming as a beginning programmer, you are expected to have a real picture of how you want your program to look like or probably how it should behave/work after compilation.

It is important for every programmer to have an understanding of programming language definition structure.

Grammars

In Formal Language, a grammar is a set of production rules for strings in the language. The rules describe how to form strings from the language alphabets that are valued according to system. It is a grammar that consists of finite set of production rules.

Formal language is an important topic area in the study of programming languages. Every programming language has its set of grammar that should be well defined while building the language translator (compiler in particular). This material will only mention some of the key areas that students will be taught and introduced to. Some of the terms that students should master in this section are: Grammar, Compiler, Production rule, Program Syntax (or syntax), symbol table, parsing, terminal symbol. Non-terminal symbol, parse tree, syntax violation and so on.

A parse tree is also known as derivation tree. This is defined as an ordered, rooted tree that represent the syntactic structure of a string according to some context free grammar. Parse tree is a concept that has to be understood in the context of Compiler Construction. This tree shows the procedure that every compiler uses to parse program statements in a program being fed into it.

Note:

Efforts will be made during lectures to provide brief introduction on grammar and formal languages as applicable in building programming languages compilers. Students are only being provided with some introductory points in this section so as to aid their understanding, during class sessions.

Grammar

A grammar lets us transform a program, which is normally represented as a linear sequence of ASCII characters, into a syntax tree. Only programs that are syntactically valid can be transformed in this way. This tree will be the main data-structure that a compiler or interpreter uses to process the program. By traversing this tree the compiler can produce machine code, or can type check the program, for instance. And by traversing this very tree the interpreter can simulate the execution of the program.

The main notation used to represent grammars is the Backus-Naur Form (BNF). This notation, invented by John Backus and further improved by Peter Naur, was first used to describe the syntax of the ALGOL programming language. A BNF grammar is defined by a four-element tuple represented by (T, N, P, S). The meaning of these elements is as follows:

a. T is a set of tokens. Tokens form the vocabulary of the language and are the smallest units of syntax. These elements are the symbols that programmers see when they are typing their code, e.g., the while's, for's, +'s, ('s, etc.
b. N is a set of nonterminals. Nonterminals are not part of the language per se. Rather, they help to determine the structure of the derivation trees that can be derived from the grammar. Usually we enclose these symbols in angle brackets, to distinguish them from the terminals.
c. P is a set of productions rules. Each production is composed of a left-hand side, a separator and a right-hand side, e.g., <non-terminal> := <expr1> ... <exprN>, where ':=' is the separator. For convenience, productions with the same left-hand side can be abbreviated using the symbol '|'. The pipe, in this case, is used to separate different alternatives.

d. S is a start symbol. Any sequence of derivations that ultimately produces a grammatically valid program starts from this special non-terminal.

As an example, below we have a very simple grammar, that recognizes arithmetic expressions. In other words, any program in this simple language represents the product or the sum of names such as 'a', 'b' and 'c'.

```
<exp> ::= <exp> "+" <exp>
<exp> ::= <exp> "*" <exp>
<exp> ::= "(" <exp> ")"
<exp> ::= "a"
<exp> ::= "b"
<exp> ::= "c"
```

This grammar could be also represented in a more convenient way using a sequence of bar symbols, e.g.:

```
<exp> ::= <exp> "+" <exp> | <exp> "*" <exp> | "(" <exp> ")" | "a" | "b" | "c"
```

Notice that context-free grammars are not the only kind of grammar that computers can use to recognize languages. In fact, there exist a whole family of formal grammars, which have been first studied by Noam Chomsky, and today form what we usually call the Chomsky's hierarchy. Some members of this hierarchy, such as the regular grammars are very simple, and recognize a relatively small number of languages. Nevertheless, these grammars are still very useful.

Regular grammars are at the heart of a compiler's lexical analysis, for instance. Other types of grammars are very powerful. As an example, the unrestricted grammars are as computationally powerful as the Turing Machines. Nevertheless, in this book we will focus on context-free grammars, because they are the main tool that a compiler uses to convert a program into a format that it can easily process.

Operators in C / C++ and Java Programming Language

C Program Operators are the foundation of any programming language. Thus, the functionality of C/C++ /Java programming language is incomplete without the use of operators. We can define operators as symbols that helps us to perform specific mathematical and logical computations on operands. In other words we can say that an operator operates the operands. For example, consider the below statement:
c = a + b;

Here, '+' is the operator known as *addition operator* and 'a' and 'b' are operands. The addition operator tells the compiler to add both of the operands 'a' and 'b'. C/C++ has many built-in operator types and they can be classified as:

- **Arithmetic Operators**: These are the operators used to perform arithmetic/mathematical operations on operands. Examples: (+, -, *, /, %,++,–). Arithmetic operator are of two types:
  1. **Unary Operators**: Operators that operates or works with a single operand are unary operators.
     For example: (++ , –)
  2. **Binary Operators**: Operators that operates or works with two operands are binary operators.For example: (+ , – , * , /)

  To learn Arithmetic Operators in details visit this link.

- **Relational Operators**: Relational operators are used for comparison of the values of two operands. For example: checking if one operand is equal to the other operand or not, an operand is greater than the other operand or not etc. Some of the relational operators are (==, > , = , <=). To learn about each of these operators in details go to this link.
- **Logical Operators**: Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a boolean value either true or false. To learn about different logical operators in details please visit this link.
- **Bitwise Operators**: The Bitwise operators is used to perform bit-level operations on the operands. The operators are first converted to bit-level and then calculation is performed

on the operands. The mathematical operations such as addition, subtraction, multiplication etc. can be performed at bit-level for faster processing. To learn bitwise operators in details, visit this link.

- **Assignment Operators**: Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and right side operand of the assignment operator is a value. The value on the right side must be of the same data-type of variable on the left side otherwise the compiler will raise an error. Different types of assignment operators are shown below:

  “=”: This is the simplest assignment operator. This operator is used to assign the value on the right to the variable on the left.
  For example:

  ```
  a = 10;
  b = 20;
  ch = 'y';
  ```

  “+=”:This operator is combination of ‘+’ and ‘=’ operators. This operator first adds the current value of the variable on left to the value on right and then assigns the result to the variable on the left.
  Example:
  (a += b) can be written as (a = a + b)

  If initially value stored in a is 5. Then (a += 6) = 11.

  “-=”:This operator is combination of ‘-‘ and ‘=’ operators. This operator first subtracts the current value of the variable on left from the value on right and then assigns the result to the variable on the left.
  Example:
  (a -= b) can be written as (a = a - b)

  If initially value stored in a is 8. Then (a -= 6) = 2.

  “*=”:This operator is combination of ‘*’ and ‘=’ operators. This operator first multiplies the current value of the variable on left to the value on right and then

assigns the result to the variable on the left.
Example:
(a *= b) can be written as (a = a * b)

If initially value stored in a is 5. Then (a *= 6) = 30.

"/=":This operator is combination of '/' and '=' operators. This operator first divides the current value of the variable on left by the value on right and then assigns the result to the variable on the left.
Example:
(a /= b) can be written as (a = a / b)

If initially value stored in a is 6. Then (a /= 2) = 3.

Examples on some Mathematical operators

Given that P=23 and Q=5

Find (i) R=P%Q
(ii) R1=P/Q

Solution

(i) R=P%Q

Note that the symbol % stands for modulus (remainder). Then, substitute the values of A and B into the equation

R=P%Q meaning, the remainder when P is divided by Q

R= 23%5

R=3 since 23 divided by 5 will produce 4 remainder 3.

(ii) R1=P/Q (Integer division)

R1=23/5

R1=4

C Programming Language

C Programming Language uses a compiler and is a very powerful structured programming language

```
#include <stdio.h>

int main() {
  /* my first program in C to be used as sample */
  printf("Hello, World! \n");
  return 0;
}
```

The description of the various parts of the above program is a s follows:

- The first line of the program #include <stdio.h> is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
- The next line int main() is the main function where the program execution begins.
- The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
- The next line printf(...) is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
- The next line **return 0;** terminates the main() function and returns the value 0.

C++

C++ is another example of an object oriented programming language.

Class Syntax:

```
class classname {
Access - Specifier:
Member Varibale Declaration;
Member Function Declaration;
}
```

The structure of a typical C++ program is as below:

```
#include <iostream.h>
using namespace std
int main() {
  /* my first program in C++ to be used as sample */
  Cout<<("Hello, World! \n");
  return 0;
}
```

The C++ programming language allows programmers to separate program-specific data types through the use of classes. Classes define types of data structures and the functions that operate on those data structures. Instances of these data types are known as objects and can contain member variables, constants, member functions, and overloaded operators defined by the programmer.

Java Programming Language

Java is an Object Oriented Language − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data. Java is case sensitive. Java is robust tt is architecture-neutral. These are some of the features of Java as a language.
Java uses hybrid software implementation. That is, Java language uses both compiler and interpreter.
Basic Concepts in Object Oriented Programming Language (Java Language as a case study)

- **Object** − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.
- **Class** − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.
- **Methods** − A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** − Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Structure of a Java program

The structure of a typical Java program is as follows:

```
//Comment that is used for documenting a Java program
//import statement as may be needed
public class ProgramExample
```

```
{
public static void main (String args[])
{
Variable declaration
Program codes written using Java syntax
}
}
```

Note: Each single program statement in Java, C or C++ must end with a semi colon. Each compound statement opens with an open brace and closes with a closing brace

{

Program codes

}

Another example of a simple Java Program that display a message

```
/* This is a Java program to display a message to the monitor screen */

   public static void main(String []args) {

   public class MyFirstJavaProgram {

      System.out.println("This is an introductory course on Object Oriented Java Programming");
   }
}
```

SECTION C: PROGRAMMING LANGUAGE EVALUATION

Programming languages differ in various ways. However, a good understanding of some of the features in a programming language can be of use. Some of the factors that may influence the choice of a programming language by a programmer include:

i. Programmer experience. This is basically the level of technical-know-how of the programmer
ii. Ease of Development and Maintenance (Simplicity)
iii. Suitability of the programming language for the problem
iv. Readability

v. Error Checking

vi. Reliability

vii. Portability- This feature allows the programmer to program to run on many different platforms, with minimal rewriting.

viii. Efficiency- The compiler should be fast. The code itself should be fast.

ix. Low Learning Curce- The language should be easy to learn. Training is expensive.

x. Reusability- Writing software components once is cheaper than writing them twice.

xi. Pedagogical value-The language should support and enforce the concepts you want to teach.

xii. Writeability- This criteria describes to the programmer to say what you mean, without excessive verbiage.

xiii. Orthogonality- The language should support the combination of its concepts/features in a meaningful way.

xiv. Consistency- The language should not include needless inconsistencies.

xv. Expressiveness- The programmer should be able to express their algorithm naturally.

xvi. Abstraction- The language should support a high level of data and control abstraction.

Data Types in Programming Languages

Programming languages have support for different data types. This is why they are able to handle input values to be supplied by the users of programs developed using such languages. Some programming languages categorize the kinds of data they handle into: Simple and Composite data.

The simple data types are generally termed primitives. The data types supported by programming languages differ in different forms.

For instance, C programming language has the following examples of data as being supported: Simple: integer, long integer, short integer, float, double, character, Boolean and so on. While the composite data types include: Enumeration, Structure, Array, and String.

The various data types are used to specify and handle the kind of data to be used in programming problem. These data types are declared in a programming problem through a process called variable declaration.

Variable Declaration in Languages

There are various kinds of programming languages. Some languages do not require that variables are declared in a program before being used while some require that variables are declared. We can therefore have implicit and explicit variable declaration. Variables can also be classified as global or local, depending on the level of access from within the program.

Implicit Variable Declaration

That is, in Implicit Variable Declaration programming languages in which the variables to be used in a program may not be declared are said to support implicit variable declaration. A good example of programming language that supports this variable declaration type is Python. That is, in Python, a programmer may declare or choose not to declare the variable that he intends using in a programming language.

Explicit Variable Declaration

Programming languages in which the variables to be used in a program should be declared are said to support Explicit variable declaration. Examples of such languages that support explicit variable declaration are: Java, C, C++, PASCAL, FORTRAN, and many others.

It is important for a programmer to always declare variable before using them in languages like C, C++, VBscript and Java. Depending on the data type to be used in a program, examples of variable declaration some selected languages are as below:

Variable Declaration in C Programming Language

```
int num1,num2,result;
float score1,score2,score3;
double x1,x2,x3,x4,total;
bool x,y;
```

Variable Declaration in C++ Programming Language

```
int num1,num2,result;
float score1,score2,score3;
double x1,x2,x3,x4,total;
bool x,y;
```

Variable Declaration in Java Programming Language

```
int num1,num2,result;
float score1,score2,score3;
double x1,x2,x3,x4,total;
bool x,y;
```

Global variable

A Global variable is the kind of variable that is accessible from other classes outside the program or class in which it is declared. Different programming languages have various of ways in which global variables are being declared, when the need arises.

Local Variable

A local variable is the kind of variable that is not accessible from other classes outside the program or class in which it is declared. These are variables that are used within the current program unit (or function) in a later section we will looking at global variables - variables that are available to all the program's functions.

The Basic data types associated with variables in languages such as C and Java. These include the following:

1. **int** - integer: a whole number.
2. **float** - floating point value: i.e a number with a fractional part.
3. **double** - a double-precision floating point value.
4. **char** - a single character.
5. **Byte**
6. **void** - valueless special purpose type which we will examine closely in later sections.

Note: In C++ and Java, the modifiers or specifiers are used to indicate whether a variable is global or local.

Data Structures in Java

It has earlier being pointed out that Java has a wide range of data structures. For instance, Java has It has earlier being pointed out that Java has a wide range of data structures.

The data structures provided by the Java utility package are very powerful and perform a wide range of functions. These data structures consist of the following interface and classes −

- Enumeration
- BitSet

- Vector
- Stack
- Dictionary
- Hashtable
- Properties

Python Programming Language (An Introduction)

Python is another High Level Programming Language that is used for building a wide range of software solutions (applications). This programming language is fast becoming popular among data scientists and Machine Learning researchers. The language has support for procedural and object oriented programming.

Python comes with supports for various data types and has been widely used among programmers in different domains. In python, the programmer may choose to declare the variables he wants to use or not. That is, python supports implicit variable declaration unlike Java that supports explicit variable declaration.

VB Script (A Quick Introduction)

Visual Basic Script popularly called VBScript is a scripting language that is used for tying web interface with its back end. VBscript is used in conjunction with an application interface called Activex Server Pages (ASP). As a scripting language, the language is very light and has a set of features that make its use in Web Development a choice for some web developers.VB Script is a scripting language developed by Microsoft. It is a light version of Microsoft Visual Basic and the VBScript syntax is very similar to that of Visual Basic.

VBScript Example:

Open your text editor (Here, Notepad is used. You can use whichever text editor you want) and add the following lines of code.

```
<html>
<head>
<title>My First VBScript Code!!!</title>
</head>
```

```
<body>

<script type="text/vbscript">
document.write("Yes!!! I have started learning VBScript.")
</script>

</body>
</html>
```

Type Conversion in Languages

Different programming languages supports different kinds of data types. At times in programming, there may be a need to convert a data from one type to the other. The term used for describing this process is called Type Conversion. Java and C++ (and some other languages) are good examples of programming language that support type conversion.

Type Conversion in C

The **type conversion process in C** is basically converting one type of data type to other to perform some operation. The conversion is done only between those datatypes wherein the conversion is possible ex – char to int and vice versa.

1) Implicit Type Conversion

This type of conversion is usually performed by the compiler when necessary without any commands by the user. Thus it is also called **"Automatic Type Conversion"**.

The compiler usually performs this type of conversion when a particular expression contains more than one data type. In such cases either type promotion or demotion takes place.

Examples

(1)

```
int a = 20;
   double b = 20.5;
   a + b
char ch='a';
```

```
int a =13;

a + c
```

Explicit Type Conversion

Explicit type conversion rules out the use of compiler for converting one data type to another instead the user explicitly defines within the program the datatype of the operands in the expression.

The example below illustrates how explicit conversion is done by the user.

Example:

```
double da = 4.5;
double db = 4.6;
double dc = 4.9;

//explicitly defined by user
int result = (int)da + (int)db + (int)dc;

printf("result = %d", result);
```

Expected Output (When these lines of codes are run in C environment)

```
result = 12
```

Thus, in the above example we find that the output result is 12 because in the result expression the user has explicitly defined the operands (variables) as integer data type. Hence, there is no implicit conversion of data type by the compiler.

Type Conversion in Java

When you assign value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not then they need to be casted or converted explicitly. For example, assigning an int value to a long variable.

Automatic Type Conversion in Java

Widening conversion takes place when two data types are automatically converted. This happens when:

i. the two data types are compatible.
ii. When we assign value of a smaller data type to a bigger data type.

For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.

```
class Test
{
   public static void main(String[] args)
   {
      int i = 100;

      // automatic type conversion
      long l = i;

      // automatic type conversion
      float f = l;
      System.out.println("Int value "+i);
      System.out.println("Long value "+l);
      System.out.println("Float value "+f);
   }
}
```

Narrowing or Explicit Conversion in Java

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

- This is useful for incompatible data types where automatic conversion cannot be done.
- Here, target-type specifies the desired type to convert the specified value to.
- char and number are not compatible with each other. Let's see when we try to convert one into other.
- *filter_none*
- *edit*
- *play_arrow*
- *brightness_4*

```
//Java program to illustrate incompatible data
// type for explicit type conversion
public class Test
{
  public static void main(String[] argv)
  {
    char ch = 'c';
    int num = 88;
    ch = num;
  }
}
```

The above lines of code will result in an error as shown below:

```
7: error: incompatible types: possible lossy conversion from int to char
    ch = num;
         ^
1 error
```

How to do Explicit Conversion?

Example:

filter_none

edit

play_arrow

brightness_4

```
//Java program to illustrate explicit type conversion
class Test
{
    public static void main(String[] args)
    {
        double d = 100.04;

        //explicit type casting
        long l = (long)d;

        //explicit type casting
        int i = (int)l;
        System.out.println("Double value "+d);

        //fractional part lost
        System.out.println("Long value "+l);

        //fractional part lost
        System.out.println("Int value "+i);
    }

}
```

```
/Java program to illustrate Conversion of int and double to byte
class Test
{
    public static void main(String args[])
    {
        byte b;
        int i = 257;
        double d = 323.142;
        System.out.println("Conversion of int to byte.");

        //i%256
        b = (byte) i;
        System.out.println("i = " + i + " b = " + b);
        System.out.println("\nConversion of double to byte.");

        //d%256
        b = (byte) d;
        System.out.println("d = " + d + " b= " + b);
    }
}
```

Runtime Consideration in Programming Languages

Runtime programming is about being able to specify program logic during application execution, without going through the code-compile-execute cycle. This article describes key elements of the infrastructure required to support runtime programming in Java, and presents a fairly detailed analysis of the design.

Programmers are expected to consider the run time for his choice of implementation in the chosen programming language.

Binding in Programming Languages

We may have to start this part by asking the question “What is binding in programming languages?”

Recall that a variable is the storage location for the storing of data in a program. Binding simply means the association of attributes with program entities. Binding can be static or dynamic. C, Java are examples of programming languages that support binding. Dynamic Binding allows greater flexibility but at the expense of readability, efficiency and reliability.

Demonstration of Binding in C and Java

As an example, let us take a look at the program below, implemented in C. In line 1, we have defined three names: int, i and x. One of them represents a type while the others represent the declaration of two variables. The specification of the C language defines the meaning of the keyword int. The properties related to this specification are bound when the language is defined. There are other properties that are left out of the language definition. An example of this is the range of values for the int type. In this way, the implementation of a compiler can choose a particular range for the int type that is the most natural for a specific machine. The type of variables i and x in the first line is bound at compilation time. In line 4, the program calls the function do_something whose definition can be in another source file. This reference is solved at link time.

The linker tries to find the function definition for generating the executable file. At loading time, just before a program starts running, the memory location for main, do_something, i and x are bound. Some bindings occur when the program is running, i.e., at runtime. An example is the possible values attributed to i and x during the execution of the program.

```
inti,x=0;
voidmain(){
for(i=1;i<=50;i++)
x+=do_something(x);
}
```

The same implementation can also be done in Java, which is as follows:

```
publicclassExample{
inti,x=0;

publicstaticvoidmain(String[]args){
for(i=1;i<=50;i++){
x+=do_something(x);
}
}
```

}

Initialization as a concept in programming languages

Initialization is the binding of a variable to a value at the time the variable is being bounded to storage. For instance in Java, C and C++, a variable can be declared and initialized at the same time as follows

float x1,x2;

int num1,num2,total,avareagevalue;

bool P,R,Q;

Flow Control in Programming Languages

Different programming languages have support for flow control. Flow control constructs such as if-then-else, while-do, and Do while take as arguments a boolean expression and one or more action statements. The actions are either flow control constructs or assignment operations, which can be constructed recursively too. The Java, or C++ class written to provide an if<condition<then<action 1<else<action 2< construct takes three arguments in its constructor, a boolean result object and two action objects, each constructed using one or more primitive constructs. The evaluate() method of the if-then-else class performs the if-then-else logic upon invocation with appropriate arguments. Most flow control constructs can be provided in a similar manner.

In programming languages, a complex program logic can be constructed as a hierarchy of objects corresponding to the primitive constructs. Such a conglomerate of objects, each individually performing a simple task, and cooperating together to achieve a complex one is the key design principle in this technology and ideally suited for distributed processing.

Other programming languages like C, PASCAL, DELPHI, Python, FORTRAN, Scala among others have their syntaxes for handling flow control. The flow control structures are generally used when there is a need to reach some conclusion based on some set of conditions.

Strongly Typed or Weakly Typed Languages

Type checking is another feature of programming languages that students have to be well familiar with. Under this section, programming languages can either be strongly typed or weakly typed. A good example of strongly type language is LISP. A good example of weakly typed language is . However, the program designer can choose to implement as strict type checking as desired, by checking for type-compatibility during construction of each of the primitive constructs.

Dynamism of Programming Languages

Dynamic programming language is a class of high-level programming languages which, at runtime, execute many common programming behaviors that static programming languages perform during compilation. These behaviors could include extension of the program, by adding new code, by extending objects and definitions, or by modifying the type system. Although similar behaviours can be emulated in nearly any language, with varying degrees of difficulty, complexity and performance costs, dynamic languages provide direct tools to make use of them.

Many of these features were first implemented as native features in the Lisp programming language. Most dynamic languages are also dynamically typed, but not all are.

Runtime of a Program

Run time is also called execution time. It is the time during which a program is running (executing), in contrast to other program lifecycle phases such as compile time, link time and load time. When a program is to be executed, a loader first performs the necessary memory setup and links the program with any dynamically linked libraries it needs, and then the execution begins starting from the program's entry point. Some program debugging can only be performed (or is more efficient or accurate when performed) at runtime. Logic errors and array bounds checking are examples of such errors in programming language.

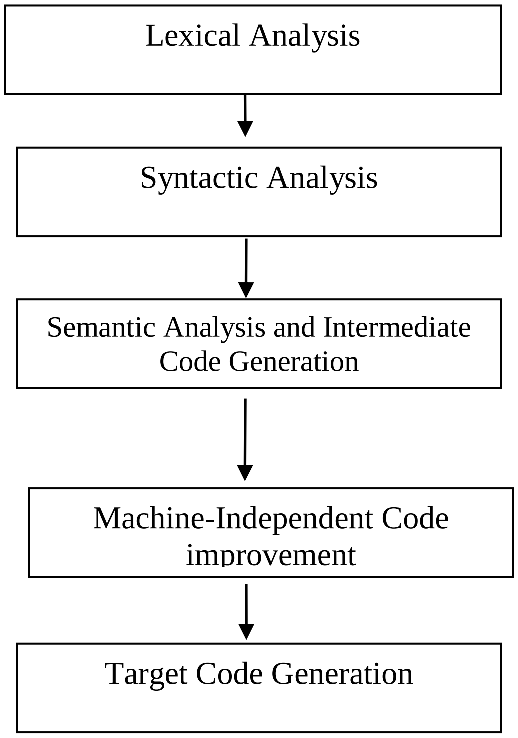
Exception Handling in Programming Languages

Exception handling is a language feature designed to handle runtime errors, providing a structured way to catch completely unexpected situations as well as predictable errors or unusual results without the amount of inline error checking required of languages without it. More recent advancements in runtime engines enable automated exception handling which provides 'root-cause' debug information for every exception of interest and is implemented independent of the source code, by attaching a special software product to the runtime engine. When an error occurs in a Java program it results in an exception being thrown. It can then be handled using various exception handling techniques.

Lexical Analysis and Parsing

In compilation, a High Level Language Program (source language) is translated into machine code called the object code. Compilation process has several phases. This includes lexical analysis which converts characters in the source program into lexical units (identifiers, operators and keywords). The other stages is syntactic analysis: This transforms lexical units into parse tress which represent the syntactic structure of a program. Semantic Analysis and then code generation.

Diagrammatically, the representation of compilation process can be as below:


Lexical Analysis
Syntactic Analysis
Semantic Analysis and Intermediate Code Generation
Machine-Independent Code improvement
Target Code Generation

Brief notes on Semantic Analysis

Syntactic Analysis is one of the stages of compilation. This transforms lexical units into parse trees which represent the syntactic structure of program. Semantic Analysis checks for errors that are hard to detect during syntactic analysis and then generate intermediate code.

It is the task of ensuring that the declarations and statements of a program are semantically correct. That is the meaning of the statements is clear and consistent with the way in which control structures and data types are supposed to be used.

Semantic Analysis is the third phase of Compiler. Semantic Analysis makes sure that declarations and statements of program are semantically correct. It is a collection of procedures which is called by parser as and when required by grammar. Both syntax tree of previous phase and symbol table are used to check the consistency of the given code.

Type checking is an important part of semantic analysis where compiler makes sure that each operator has matching operands. Semantic Analyzer uses syntax tree and symbol table to check whether the given program is semantically consistent with language definition. It gathers type information and stores it in either syntax tree or symbol table. This type information is subsequently used by compiler during intermediate-code generation.

Some of the functions of Semantic Analysis include:

i. **Type Checking –**Ensures that data types are used in a way consistent with their definition.
ii. **Label Checking –**A program should contain labels references.
iii. **Flow Control Check –**Keeps a check that control structures are used in a proper manner.(example: no break statement outside a loop)

Parameter Passing Mechanisms in Programming Languages

Different programming languages employ various parameter passing mechanisms. For instance, the different mechanisms for passing parameters to functions:

i. value
ii. reference
iii. result
iv. value-result
v. name

However, in modern programming languages, parameters passing are generally carried out through different techniques. However, the two popular techniques that are supported that we will be focusing on in this course are using: parameter passing by value and parameter passing by reference. In this material, we will briefly explain how parameter passing is carried out in C and Java to some extent.

Parameter Passing in C

A Parameter is the symbolic name for "data" that goes into a function. There are two ways to pass parameters in C programming language. These two ways are named: Pass by Value and Pass by Reference.

Pass by Value

Pass by Value, means that a copy of the data is made and stored by way of the name of the parameter. Any changes to the parameter have no effect on data in the calling function.

Pass by Reference

A reference parameter "refers" to the original data in the calling function. Thus any changes made to the parameter are also made to the original variable.

However, in C language, the default is to pass a parameter by value

An example of Parameter passing by value in C language is as shown below

```
// C function using pass by value.
void
doit( int x )
{
    x = 5;
}

//
// Test function for passing by value (i.e., making a copy)
//
int
main()
{
  int z = 27;
  doit( z );
  printf("z is now %d\n", z);

  return 0;
}
```

Example of Passing by Reference in C

```
// "Pure" C code using Reference Parameter! (aka pointers)
      //
      void
      doit( int * x )
      {
         *x = 5;
      }
```

```
//
// Test code for passing by a variable by reference
// Note the use of the & (ampersand) in the function call.
//
int
main()
{
  int z = 27;
  doit( & z );
  printf("z is now %d\n", z);

  return 0;
}
```

Parameter Passing n Java

To aid the understanding of the learner, let me explain the parameter passing mechanisms in Java as follows:

In most cases, parameter passing in Java is always Pass-by-Value. However, the context changes depending upon whether we're dealing with Primitives or Objects. This statement means that the two popular parameter passing in Java (pass-by-reference and pass-by-value) occur under the following conditions:

i. For Primitive types, parameters are pass-by-value
ii. For Object types, the object reference is pass-by-value

Passing Object References

In Java, all objects are dynamically stored in Heap space under the hood. These objects are referred from references called reference variables.

A Java object, in contrast to Primitives, is stored in two stages. The reference variables are stored in stack memory and the object that they're referring to, are stored in a Heap memory.

Whenever an object is passed as an argument, an exact copy of the reference variable is created which points to the same location of the object in heap memory as the original reference variable.

As a result of this, whenever we make any change in the same object in the method, that change is reflected in the original object. However, if we allocate a new object to the passed reference variable, then it won't be reflected in the original object.

Pass-by-Reference in Java

Here, we are explaining pass-by-reference further. When a parameter is pass-by-reference, the caller and the callee operate on the same object.

It means that when a variable is pass-by-reference, the unique identifier of the object is sent to the method. Any changes to the parameter's instance members will result in that change being made to the original value.

The fundamental concepts in any programming language are "values" and "references". In Java, Primitive variables store the actual values, whereas Non-Primitives store the reference variables which point to the addresses of the objects they're referring to. Both values and references are stored in the stack memory. That is, arguments in Java are always passed-by-value.

In summary, Java uses both pass by value and pass by sharing –Variables of primitive built-in types are passed by value –Class instances are passed by sharing.

Sample Questions and Short Answers

(1) Write short notes on Global and Local Variable in C programming Language

b. (i) Global variable refers to as the variable that is declared in a program and is accessible from other function/part from within the program. Different programming languages support declaring variables as global and therefore allows such languages to be rich and powerful.

(ii)Local variable is the kind of variable that can only be accessed or referenced from within the program in which it is declared.

(2) What are Java Arrays?

Answer: Java Arrays

Arrays are objects that store multiple variables of the same type. However, an array itself is an object on the heap. We will look into how to declare, construct, and initialize in the upcoming chapters.

3. Write brief notes on Java Enumerated Data Types

Answer: Java Enums (Enumerated Feature in Java)

Enums were introduced in Java 5.0. Enums restrict a variable to have one of only a few predefined values. The values in this enumerated list are called enums. With the use of enums it is possible to reduce the number of bugs in your code.

For example, if we consider an application for a fresh juice shop, it would be possible to restrict the glass size to small, medium, and large. This would make sure that it would not allow anyone to order any size other than small, medium, or large.

Example

```
class FreshJuice {
  enum FreshJuiceSize{ SMALL, MEDIUM, LARGE }
  FreshJuiceSize size;
}

public class FreshJuiceTest {

  public static void main(String args[]) {
    FreshJuice juice = new FreshJuice();
    juice.size = FreshJuice.FreshJuiceSize.MEDIUM ;
    System.out.println("Size: " + juice.size);
  }
}
```

H. Study Questions (For practice only)

i. Why is there a need to study programming languages?

ii. Identify some of the factors that influence the choice of programming language for a software development project

iii. List and explain the three different types of translators

iv. Given that x1=34.5683 and x2=34.2311, write a C program segment to declare and initialize x1 and x2 respectively

v. Explain, with examples, how automatic type conversion is carried out in Java Language

vi. Itemise any five features of a good programming language

vii. Compare and contrast the basic features in C and C++

viii. Explain briefly how Java supports binding

ix. Explain briefly how execution is handled in Java programming language

x. Explain the case sensitivity feature of Java, C and C+++ programming languages

xi. Explain each of the following terms: (i) Strongly typed language (ii) Scope of a variable

xii. Explain each of the following terms: (i) Global variable (ii) Local Variable

xiii. Explain what we mean by running time of a program

xiv. Arrays are categorized based on the binding to subscript ranges, the binding to storage and from where the storage is allocated. Discuss

xv. Given that X=12 and B=7

Find (i) C=A%B

D=A/B

K=A*4B-D

xvi. In tabular form, itemize any three differences between human and computer languages

xvii. Write short notes on each of the following : (a) Lexical analysis (b) Semantic analysis

xviii. Explain each of the following programming paradigms:

(a)Procedural Language

(b) Object Oriented Language

(c) Functional Language

xix. Explain flow control in C and Java Programming languages

xx. What do you understand by Intermediate Code Generation?

xxi. Identify any six language evaluation criteria that can be used by a programmer

xxii. Write short notes on post fix notation on VBScript as a Scripting language for the web

xxiii. Briefly explain how parameter passing mechanisms are carried out under C and Java.

I. References

1. Bloch Stepehn (2011). Survey of Programming Languages, retrieved from home.adelphi.edu

2. Deitel Paul and Deitel Harvey (2010). C How to Program 6th Edition, Pearson Education, Inc. Upper Saddle River, New Jersey 07458
3. Krishnamurthi Shriram (2003). Programming Languages: Application and Interpretation, *retrieved from https://cs.brown.edu/~sk/Publications/Books/ProgLangs/2007-04-26/plai-2007-04-26.pdf*
4. Deitel & Deitel (2015). Java: How to Program (Early Objects) 9th Edition, Paul J. Deitel, Deitel & Associates, Inc.
5. TutorialPoint (n.d.). Java Tutorial retrieved from https://www.tutorialspoint.com/java/index.htm
6. Guru99 (n.d). Introduction to VBscript, retrieved from https://www.guru99.com/introduction-to-vbscript.html
7. Geeks for Geeks (n.d.) retrieved from https://www.geeksforgeeks.org/semantic-analysis-in-compiler-design/
8. Kahanwal (2013) Abstraction level taxonomy of programming language frameworks. Retrieved from https://arxiv.org/abs/1311.3293
9. David A. Watt (1990) Programming Language Design Concepts, University of Glasgow with contributions by William Findlay.
10. Utah University (n.d.). Functions in the C programming Languages retrieved form https://www.cs.utah.edu/~germain/PPS/Topics/C_Language/c_functions.html

J. Recommended books/Materials /Web Documents

1. Bloch Stepehn (2011). Survey of Programming Languages, retrieved from home.adelphi.edu
2. Deitel & Deitel (2015). Java: How to Program (Early Objects) 9th Edition, Paul J. Deitel, Deitel & Associates, Inc.

3. TutorialPoint (n.d.). Java Tutorial retrieved from https://www.tutorialspoint.com/java/index.htm
4. Utah University (n.d.). Functions in the C programming Languages retrieved form https://www.cs.utah.edu/~germain/PPS/Topics/C_Language/c_functions.html