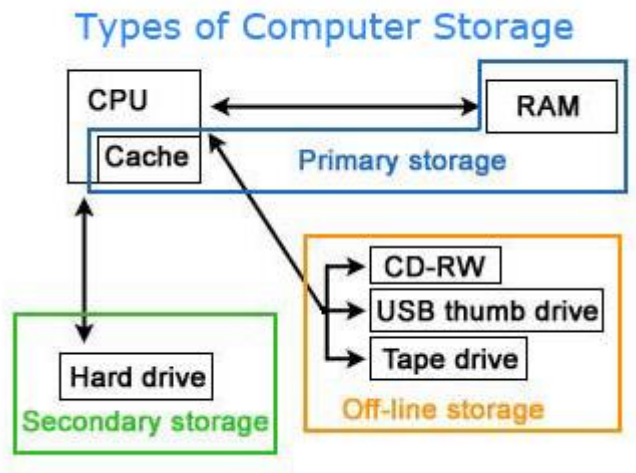
REVATURE

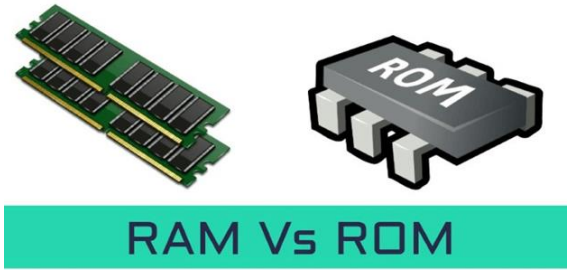
Introduction to Computer Fundamentals

Computer fundamentals: Types of Memory

REVATURE

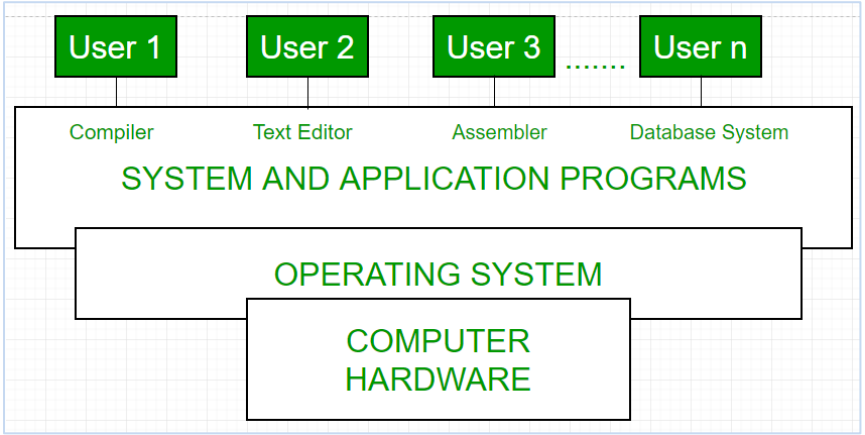
- *Memory is the most essential element of a computing system because without it computer can't perform simple tasks.*
- *Computer memory is of two basic types –*
  - ❑ *Primary memory(RAM and ROM) and*
  - ❑ *Secondary memory (hard drive, CD, etc).*
- *Random Access Memory (RAM) is primary-volatile memory and*
- *Read-Only Memory (ROM) is primary-non-volatile memory.*
- *ROM is also a primary memory just like RAM, but unlike RAM, ROM is able to store data permanently which makes it non-volatile. It is a programmable chip that stores all the most important instructions required to start the system, this process is also known as bootstrap.*
- *With ROM, the system will stay active and your data will not be overwritten, deleted or modified even if you shut it down. Thus its name, Read only memory since the data can only be read and accessed by the user.*



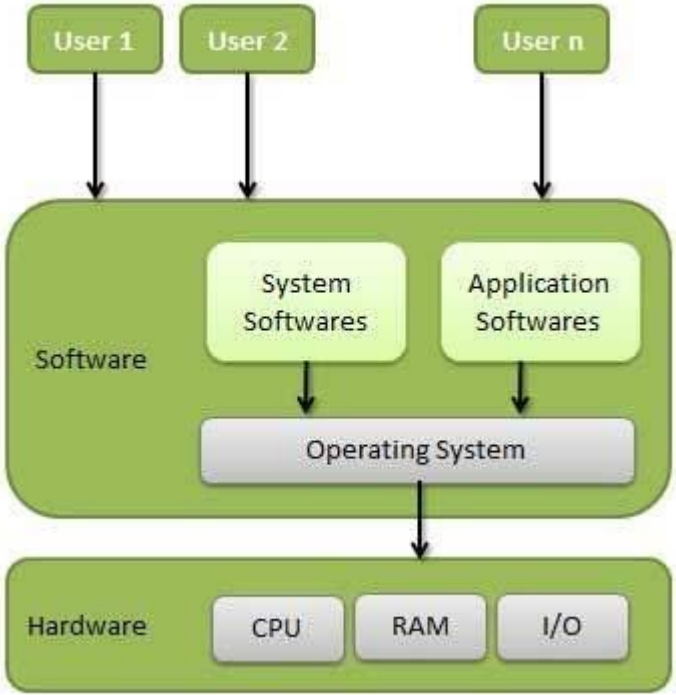


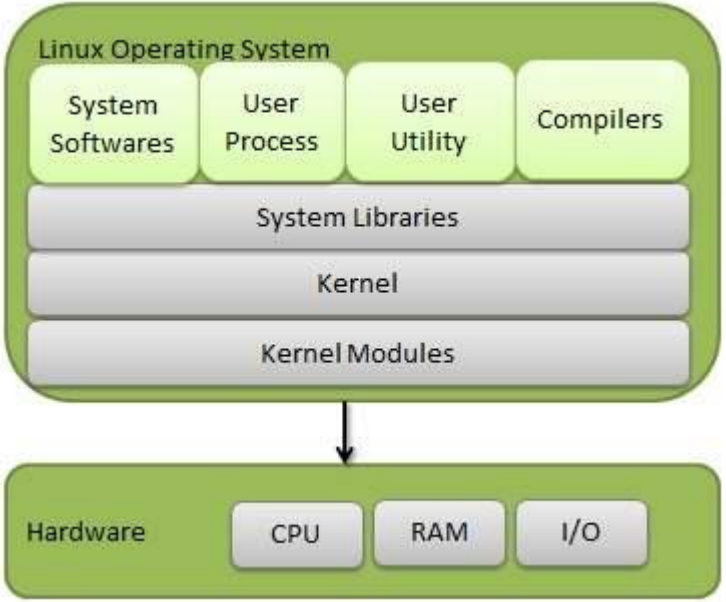
OS: Fundamentals

- *An operating system acts as an intermediary between the user of a computer and computer hardware.*
- *The purpose of an operating system is to provide an environment in which a user can execute programs conveniently and efficiently.*
- *An operating system is a software that manages computer hardware.*
- *The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.*
- *An operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.*
- *A more common definition is that the operating system is the one program running at all times on the computer (usually called the kernel), with all else being application programs.*
- *An operating system is concerned with the allocation of resources and services, such as memory, processors, devices, and information.*
- *The operating system correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, a memory management module, I/O programs, and a file system.*



OS: Fundamentals



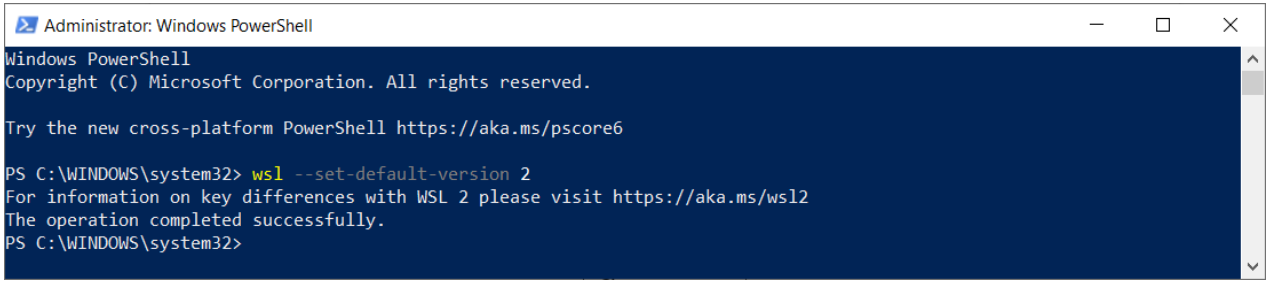


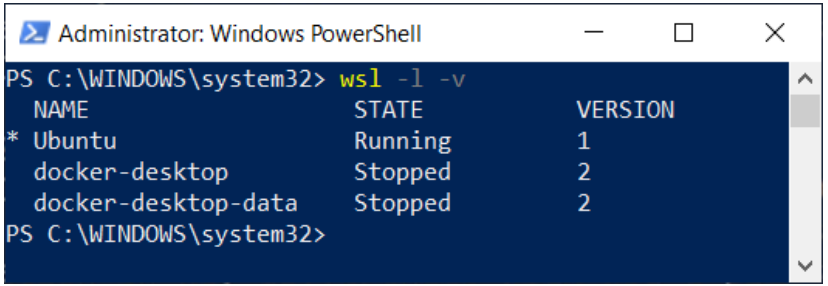
Linux

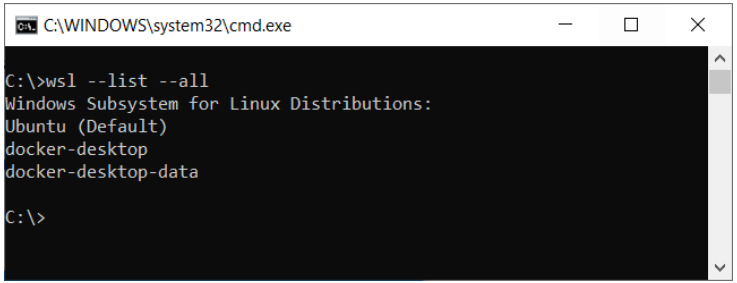
Check all WSL in Windows 10 | Microsoft introduced in 2016

REVATURE

How to install WSL(Windows Subsystem for Linux) Ubuntu on Windows 10:
https://docs.microsoft.com/en-us/windows/wsl/tutorials/gui-apps
https://docs.microsoft.com/en-us/windows/wsl/about
https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-10#1-overview
https://pureinfotech.com/run-linux-gui-apps-wsl-windows-10/







Command-line Text Editors

You can write files in the shell by making use of built-in text editors such as nano and vim or vi.
Vi/Vim
Vi originally came from the word "visual" as it was an early attempt at a more visual text editor. In order to use Vi, you can run vi file-path, and it will open in your terminal. Vi is set up such that there are two main "modes" that you can switch between - command mode and insert mode. When you first open Vi, you start in command mode, where you will be unable to type anything into the file. You can delete, copy, cut, paste, and undo, however, using the following commands:

- *x to delete the character under the cursor.*
- *dd to delete the current line.*
- *v to mark the starting point of text you want to copy or cut.*
- *y to copy anything between the cursor and the place where you typed v.*
- *x to cut anything between the cursor and the place where you typed v.*
- *p to paste.*
- *u to undo.*

After pressing i in command mode, you will switch to insert mode, where you can freely type in your file. After you've typed whatever you want to type, you can use the esc key to go back to command mode, and use any of the following commands:

- *:w to save.*
- *:q to quit without saving.*
- *:wq to save and quit.*
- *:q! to force quit (if you've made changes that you don't want to save).*

Vim is short for "Vi IMproved", and it is simply an improved version of Vi. You can use it by running vim file-path. Some of the improvements that were made include the ability to undo more than one action, open multiple windows, and more.
Vi/Vim are liked because they make complex text edits easy and fast. Once you get comfortable with the commands and switching between modes, it's a quick, light tool, great for performing for complex edits efficiently, although there is more of a learning curve.

NANO Text Editor in Linux

REVATURE

Open a file using Nano by running nano file-path. A key feature of Nano is that it shows you a list of useful commands right at the bottom of the terminal which makes it easy to use, especially for beginners. Most commands use the CTRL key, marked by ^ in the list. For example, pressing CTRL+G (Get Help) will show all commands.
Unlike Vi/Vim, Nano does not feature different modes - insertion and commands can both be done without needing to switch anything. This, along with the list of commands on the bottom, make the editor easy to use, especially for beginners and/or those who are more accustomed to modern GUI text editors.

NANO editor

- *nano <File Name>*

To save a file

- *press Ctrl+o*

To close a file

- *press Ctrl+x*
- *press ENTER to save*

VIM Editor in Linux

REVATURE

VIM editor

- *vim <File Name>*
- *INSERT key*

Exit vim without saving changes i.e. discards any changes you made.

- *Press the ESC key*
- *Type :q!*
- *Press the ENTER key*

Save a file and exit.

- *Press the ESC key*
- *Type :w <filename> to overrite :w! <filename>*
- *Press the ENTER key*
- *Type :q to close*
- *Press the ENTER key*

Save a file and exit.

- *Press the ESC key*
- *Type :x*
- *Press the ENTER key*

Commands

REVATURE

- *In this section, we'll be covering some basic Unix commands.*
- *Before we begin, we should cover the basic aliases for locations in your file system.*
- *A file system is generally comprised of a directory, or folder, that itself can contain files and directories in a tree structure.*
- *The top-level directory is known as the **root directory** and it is the folder that contains all of the other folders on the drive partition you are currently accessing.*
- *In Unix, your **root directory** is represented by the / character.*
- *For example, if you wished to change directory to the **root directory** from anywhere on the partition, you would issue the following command: cd /*
- *That's all well and good, but what about the directory you're currently in? The . character represents the current directory. Furthermore, .. represents the parent directory, so if you wanted to copy everything from the current directory up to the directory immediately above it, you would issue the following command: **cp -r . ..***
- *In Unix, we also have a directory called the **home directory**. This directory is usually the one that our terminal starts in and it is where our personal files are generally stored. This directory is represented by the ~ character.*
- *Now, let's talk about some Unix commands:*

Commands

Arguments and Flags

- *In Unix, commands can be modified in two ways.*
- *Commands can have arguments. Arguments given to a command take the form of strings written after the command. ex.* ***command arg1 arg2 arg3***
- *Commands can have flags. Flags are special arguments given to a command. There are two kinds of flags in Unix, short-hand or character flags, a single character (or group of characters), prefixed by a single dash* ***-c****, and full flags, the full name of the flag, prefixed by a double dash* ***--flag****.*
- *Arguments to your command usually represent variables or targets for your command, and flags usually represent options you wish to enable for your command.*
- *Ex. The following two commands do the same thing:*

```
cp -r hi bye
cp --recursive hi bye
```

- *The above commands copy the contents of the* ***hi*** *directory to the* ***bye*** *directory recursively, using the* ***-r*** *and* ***--recursive*** *flags. This means that the* ***cp*** *command has been told to copy the contents of a directory instead of it's default mode which copies a specific file.*
- ***hi*** *and* ***bye*** *are arguments to the command.*

Commands

The Most Important Command

- ***man*** *- The manual command will print to the terminal the manual for using a particular command.*
- *If you are unsure what flags or arguments a command takes, you simply type man command.*
- *For example, if you wished to see the manual for the copy command, you would issue the command: man cp*

Commands

REVATURE

Directory Commands

- `cd` - The change directory command allows us to navigate to a different directory on the drive.
  - go to root directory: `cd /`
  - go to home directory: `cd` or `cd ~`
  - navigate one directory up: `cd ..`
  - navigate into the `hi` directory, which is inside the `bye` directory: `cd ./bye/hi`
  - change to the previous directory: `cd -`
- `ls` - The list directory command allows us to see the contents of a particular directory. When given no arguments, it lists the contents of the current directory. The `-a` flag allows you to see hidden items in the directory.
  - list the contents of the current directory: `ls`
  - list the contents of the `hi` directory: `ls hi` or `ls ./hi`
  - list the contents of the directory including the "hidden" contents: `ls -a`
- `mkdir` - The make directory command allows us to create a new directory. mkdir takes an argument representing the name of the directory you wish to create.
  - create a directory named `hi`: `mkdir hi`
- `pwd` - The print working directory command prints the full name of the directory you are currently working in. For example, if you were working in the `home` directory inside of the `root directory` the output of `pwd` might be `/home`.

Commands

REVATURE

General Purpose Commands

- `su` - The substitute user command allows you to switch users. With no argument, this defaults to the root user, which has higher priveleges. This can be useful if you need to perform multiple commands with elevated priveleges but is generally considered to be bad practice in preference to `sudo`, for administrative logging purposes.
- `sudo` - the sudo command allows you to run a particular command as the root user.
- `clear` - the clear command usually prints a number of blank lines such that all previous commands are no longer on the screen. There is a shortcut for this command, `ctrl-l`
- `echo` - the echo command will print a string or the result of a command to the console.
- `>` - The `>` operator will redirect the output of a command to a file. The file will be created or overwritten if it already exists. ex. `ls . > log.txt`
- `>>` - The `>>` operator acts the same way as the `>` operator but appends output to the file instead of overwriting if it exists.
- `grep` - the grep command prints any lines in a file or files that match a given pattern. By default, grep interprets the pattern as a basic regular expression.
    - Print all lines in `hello.txt` that contain the word `goodbye`: `grep goodbye hello.txt`

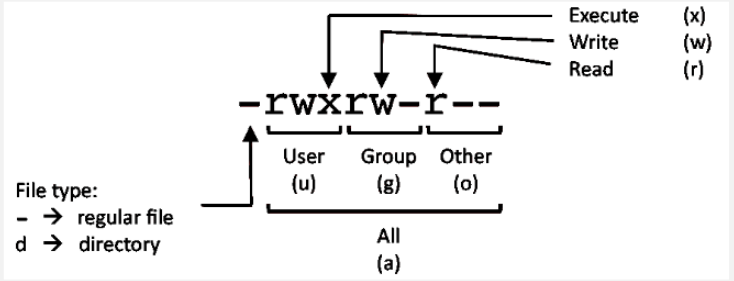
Commands

REVATURE

File Commands

- `cat` - the concatenate command prints the contents of a file to the console. `cat hello.txt`
- `head` - the head command prints the first ten lines of a file to the console. `head hello.txt`
- `tail` - the tail command prints the last ten lines of a file to the console. `tail hello.txt`
- `touch` - the touch command allows you to modify the timestamp of a file. This command is usually used to create empty files, as an empty file is created if touch is given a file name that does not exist. `touch hello.txt`
- `cp` - the copy command creates a copy of the specified file at the location specified. If the recursive glag is used, it will operate on directories.
    - copy a `hello.txt` to `goodbye.txt`: `cp hello.txt goodby.txt`
    - copy the `hello` directory to the `goodbye` directory: `cp -r hello goodbye`
- `mv` - the move command will rename or move a file or entire directory with the recursive flag.
    - rename a `hello.txt` to `goodbye.txt`: `mv hello.txt goodbye.txt`
    - move `hello.txt` to the `goodbye` directory: `mv hello.txt goodbye/.`
    - rename the `hello` directory to `goodbye`: `mv -r hello goodbye`
- `rm` - the remove command will delete a file. If you use the recursive flag, it can delete a directory. The force flag will cause the command to delete files without prompting the user if there are warnings. The command `rm -rf .` is extremely dangerous.
    - remove `hello.txt`: `rm hello.txt`
    - remove the `hello` directory: `rm -r hello`
- `wc` - the word count command will print the number of words in a file. This command has several flags available
    - `-c, --bytes` - prints the byte count
    - `-m, --chars` - prints the character count
    - `-l, --lines` - prints the lines
    - `-w, --words` - prints the word count (default)
- `ln` - the link command creates a link between files. This allows you to make a shortcut to a file in one location without copying it over.

Commands




```
        PERMISSION      EXAMPLE

         U   G   W
        rwx rwx rwx     chmod 777 filename
        rwx rwx r-x     chmod 775 filename
        rwx r-x r-x     chmod 755 filename
        rw- rw- r--     chmod 664 filename
        rw- r-- r--     chmod 644 filename

# NOTE: Use 777 sparingly!

        LEGEND
        U = User
        G = Group
        W = World

        r = Read
        w = write
        x = execute
        - = no access
```

Activity

REVATURE

```
gd@Rev-PG02RELN:~$ ls
ganagdhar.parde  gd  gd.save  gd.txt  gd.txt.save  gd@.txt  md.txt  search.txt  temp2
gd@Rev-PG02RELN:~$ rm -v *
removed 'ganagdhar.parde'
removed 'gd'
removed 'gd.save'
removed 'gd.txt'
removed 'gd.txt.save'
removed 'gd@.txt'
removed 'md.txt'
removed 'search.txt'
rm: cannot remove 'temp2': Is a directory
gd@Rev-PG02RELN:~$ ls
temp2
gd@Rev-PG02RELN:~$ rm -r temp2/
gd@Rev-PG02RELN:~$ ls
gd@Rev-PG02RELN:~$
```

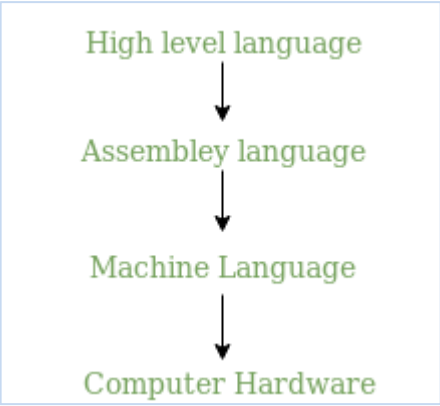
Execute grep command:

Example: https://www.geeksforgeeks.org/grep-command-in-unixlinux/

Introduction to Programming Languages

REVATURE

- *A programming language is a set of symbols, grammars and rules with the help of which one is able to translate algorithms to programs that will be executed by the computer.*
- *The programmer communicates with a machine using programming languages.*
- *Most of the programs have a highly structured set of rules.*
- *The primary classifications of programming languages are:*
    1. *Machine Languages.*
    2. *Assembly Languages.*
    3. *High level Languages.*



Introduction to Programming Languages

REVATURE

Machine language

- *It is a collection of binary digits or bits that the computer reads and interprets.*
- *Machine language is the only language a computer is capable of understanding.*
- *Machine level language is a language that supports the machine side of the programming or does not provide human side of the programming.*
- *It consists of (binary) zeros and ones.*
- *Each instruction in a program is represented by a numeric code, and numerical addresses are used throughout the program to refer to memory locations in the computer's memory.*

Assembly language

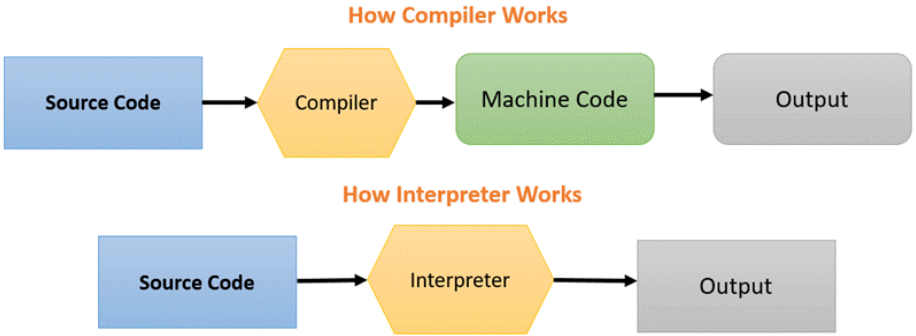
- *Assembly language is easier to use than machine language.*
- *An assembler is useful for detecting programming errors.*
- *Programmers do not have the absolute address of data items.*
- *Assembly language encourage modular programming.*

High level language

- *High level language is a language that supports the human and the application sides of the programming.*
- *A language is a machine independent way to specify the sequence of operations necessary to accomplish a task.*
- *A line in a high level language can execute powerful operations, and correspond to tens, or hundreds, of instructions at the machine level. Consequently more programming is now done in high level languages.*
- *Examples of high level languages are BASIC, FORTRAN etc*

Interpreter Vs Compiler

- *We generally write a computer program using a high-level language. A high-level language is one that is understandable by us, humans. This is called source code.*
- *However, a computer does not understand high-level language. It only understands the program written in 0's and 1's in binary, called the machine code.*
- *To convert source code into machine code, we use either a compiler or an interpreter.*
- *We have system software known as compilers and interpreters that aids in making the conveyance smoother.*



Working of the Compiler

REVATURE

- *A compiler is computer software that readily translates programming language into machine code or assembly language or low-level language.*
- *It translates every program to binary(1's and 0's) that a computer feasibly understands and does the task that corresponds to the code.*
- *One condition that a compiler has to follow is the syntax of the programming language that is used.*
- *Thus, if the syntax of the program does not match the analysis of the compiler, an error arises that has to be corrected manually in the program written.*

Working of the Compiler

REVATURE

The main work of the compiler is to translate the program into machine code and let the programmer know if there are any errors, ranges, limits, etc., especially the syntactical errors in the program. It analyses the entire program and converts it into machine code. The working of a compiler can be categorized into the following phases:

- *Lexical analysis: Splitting of source code into an abstract fragment known as lexeme. A token is generated for each of the lexemes, referring to whether it is a keyword, a string, or some other variable.*
- *Syntax Analysis: The tokens assigned are structured to form an Abstract Syntax Tree(AST) and checked for errors in the syntax.*
- *Semantic Analysis: The AST is checked for semantic errors like the wrong variable assigned, using an undeclared variable, using keywords as variable, etc.*
- *Intermediate Code Generation: The process of compilation generates two or more intermediate code forms.*
- *Optimization: The compilation process looks for multiple ways through which the task can be enhanced.*
- *Code generation: The compiler converts the intermediate optimized code into a machine code after which the source program is converted to an object program.*

Role of a Compiler

- *It reads the source code and provides an executable code.*
- *Translates programs written in a high-level language to a language that the CPU can understand.*
- *The process is relatively complicated and takes time for analysis.*
- *The executable code will be in machine-specific binary code.*
- *Total run time is more and occupies a large part of the memory.*

What is an Interpreter?

REVATURE

An interpreter is a computer program that converts program statements into machine code. Program statements include source code, pre-compiled code, and scripts.

Working of an Interpreter

An interpreter works more or less similar to a compiler. The only difference between their working is that the interpreter does not generate any intermediate code forms, reads the program line to line checking for errors, and runs the program simultaneously.

Role of an Interpreter

- *It converts program statements, line by line, into machine code.*
- *Allows modification of program while executing.*
- *Relatively lesser time is consumed for analysis as it runs line by line.*
- *Execution of the program is relatively slow as analysis takes place every time the program is run.*

Interpreter Vs Compiler

REVATURE

Basis	Compiler	Interpreter
Analysis	The entire program is analyzed in a compiler.	Line by line of the program is analyzed in an interpreter.
Machine Code	Stores machine code in the disk storage.	Machine code is not stored anywhere.
Execution	The execution of the program happens only after the entire program is compiled.	The execution of the program takes place after every line is evaluated and hence the error is raised line by line if any.
Run Time	Compiled program runs faster	Interpreted program runs slower.
Generation	The compilation gives an output program that runs independently from the source file.	The interpretation does not give any output program and is thus evaluated on every execution.
Optimization	The compiler reads the entire program and searches multiple times for a time-saving execution.	No rigorous optimization takes place as code is evaluated line by line
Error and error execution	All the errors are shown at the end of the compilation and the program cannot be run until the error is resolved	Displays the errors from line to line. The program runs till the error is found and proceeds further on resolving.
Input	The compiler takes in the entire program for analysis.	The interpreter takes in lines of code for analysis.
Output	The compiler gives intermediate code forms or object code	The interpreter does not generate any intermediate code forms.
Programming languages	C, C++, C#, Java are compiler-based programming languages	PHP, PERL, Ruby are interpreter-based programming languages

Introduction of Programming Paradigms

REVATURE

- *Paradigm can also be termed as method to solve some problem or do some task.*
- *programming paradigms are a way to classify programming languages based on their features.*
- *Programming paradigm is an approach to solve problem using some programming language or also we can say it is a method to solve a problem using tools and techniques that are available to us following some approach.*
- *There are lots for programming language that are known but all of them need to follow some strategy when they are implemented, and this methodology/strategy is paradigms. Apart from varieties of programming language there are lots of paradigms to fulfill each and every demand.*

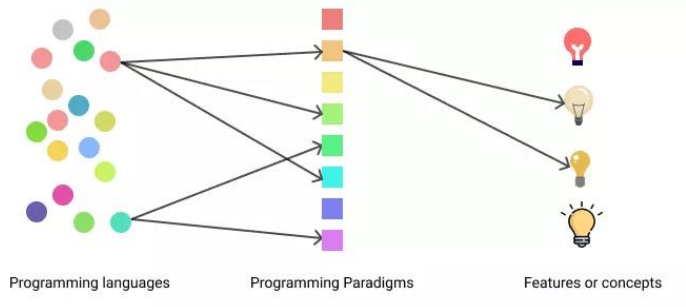
Introduction of Programming Paradigms

REVATURE

- *Paradigm can also be termed as method to solve some problem or do some task.*
- *programming paradigms are a way to classify programming languages based on their features.*
- *Programming paradigm is an approach to solve problem using some programming language or also we can say it is a method to solve a problem using tools and techniques that are available to us following some approach.*
- *There are lots for programming language that are known but all of them need to follow some strategy when they are implemented, and this methodology/strategy is paradigms. Apart from varieties of programming language there are lots of paradigms to fulfill each and every demand.*
- *Two types of programming paradigms:*
    1. *Imperative - specifies the procedure, telling the computer how to do*
    2. *Declarative. - tells the computer what to do.*

```
//Imperative programming
total = function(numbers){
  let total = 0
  numbers.forEach(function(number){
    total = total + number
  })
  return total
}


//Declarative programming
average = function(numbers){
  return total(numbers) / numbers.length
}
```



Introduction of Programming Paradigms

REVATURE

- *The **imperative programming paradigm** is the oldest paradigm and follows the most basic idea of programming. The programs in these paradigms are executed step by step by changing the current state of the program.*
- *Some of the programming languages that support the imperative paradigm are:*
  - *C*
  - *C++*
  - *Java*
  - *PHP*
  - *Ruby*
  - *Scala*
  - *Pascal*

HOW IT SHOULD BE DONE?

- *A **declarative programming paradigm** are those paradigms in which the programmer describes the property of the result without focusing on how to achieve it (imperative programming focuses on how to achieve the task by changing the state of the program).*
- *The approach of this programming is to create some object or elements within the program.*
- *The main focus in this kind of programming is what is to be done rather than how it should be done.*
- *It does not talk about the work process of the result and only describes what the program must accomplish.*
- *Declarative programming is used in programming languages used in a database query, regular expression, functional programming, logical programming, etc.*
- *Some of the programming languages that support the declarative paradigm are:*
  - *Prolog*
  - *javascript*
  - *Scala*
  - *Lisp*
  - *SQL*
  - *XQuery*
  - *Clojure*

WHAT IS TO BE DONE?

Sub Categories

REVATURE

Imperative *in which the programmer instructs the machine how to change its state,*

- ***procedural*** *which groups instructions into procedures,*
- ***object-oriented*** *which groups instructions with the part of the state they operate on,*

Declarative *in which the programmer merely declares properties of the desired result, but not how to compute it*

- ***functional*** *in which the desired result is declared as the value of a series of function applications,*
- ***logic*** *in which the desired result is declared as the answer to a question about a system of facts and rules,*
- ***mathematical*** *in which the desired result is declared as the solution of an optimization problem*
- ***reactive*** *in which the desired result is declared with data streams and the propagation of change*

Additional References | Optional

Appendix

REVATURE

https://www.tutorialspoint.com/operating_system/os_overview.htm
https://www.linuxtrainingacademy.com/linux-commands-cheat-sheet/