***ISAIAH DYAJI***

***17/SCI01/027***

***CSC 302***

***ASSIGNMENT***

1. a). **Unstructured Programming Paradigm**

Unstructured programming is the historically earliest programming paradigm capable of creating Turing-complete algorithms. It is often contrasted with the structured programming paradigm, in particular with the use of unstructured control flow using goto statements or equivalent. The distinction was particularly stressed by the publication of the influential "GoTo Statement Considered Harmful" open letter in 1968 by Dutch computer scientist Edsger W. Dijkstra, who coined the term "structured programming".

Unstructured programming has been heavily criticized for producing hardly-readable ("spaghetti") code.

There are both high- and low-level programming languages that use non-structured programming. Some languages commonly cited as being non-structured include JOSS, FOCAL, TELCOMP, assembly languages, MS-DOS batch files, and early versions of BASIC, Fortran, COBOL, and MUMPS.

b). **Structured Programming Paradigm**

Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

It emerged in the late 1950s with the appearance of the ALGOL 58 and ALGOL 60 programming languages, with the latter including support for block structures. Contributing factors to its popularity and widespread acceptance, at first in academia and later among practitioners, include the discovery of what is now known as the structured program theorem in 1966. Structured programming is most frequently used with deviations that allow for clearer programs in some particular cases, such as when exception handling has to be performed. Some languages used are C, C++, C#, PHP, Ruby, PERL, ALGOL, Pascal, PL/I and Ada.

c). **Modular Programming Paradigm**

Modular programming is the process of subdividing a computer program into separate sub-programs.

A module is a separate software component. It can often be used in a variety of applications and functions with other components of the system. Similar functions are grouped in the same unit of programming code and separate functions are developed as separate units of code so that the code can be reused by other applications.

Object-oriented programming (OOP) is compatible with the modular programming concept to a large extent. Modular programming enables multiple programmers to divide up the work and debug pieces of the program independently. Modular programming has a main module and many auxiliary modules. The main module is compiled as an executable (EXE), which calls the auxiliary module functions. Auxiliary modules exist as separate executable files, which load when the main EXE runs. Each module has a unique name assigned in the PROGRAM statement. Function names across modules should be unique for easy access if functions used by the main module must be exported.

Languages that support the module concept are IBM Assembler, COBOL, RPG, FORTRAN, Morpho, Zonnon and Erlang.

d). **Object Oriented Programming Paradigm**

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well-suited for programs that are large, complex and actively updated or maintained.

The organization of an object-oriented program also makes the method beneficial to collaborative development, where projects are divided into groups. Some examples of languages used are Java

JavaScript, Python, C++, Visual Basic .NET, Ruby, Scala, PHP.

e). **Aspect Oriented Programming Paradigm**

Aspect-Oriented Programming (AOP) is a programming paradigm which complements Object-Oriented Programming (OOP) by separating concerns of a software application to improve modularization. The separation of concerns (SoC) aims for making a software easier to maintain by grouping features and behavior into manageable parts which all have a specific purpose and business to take care of.

OOP already allows for modularizing concerns into distinct methods, classes and packages. However, some concerns are difficult to place as they cross the boundaries of classes and even packages. One example for such a cross-cutting concern is security: Although the main purpose of a Forum package is to display and manage posts of a forum, it has to implement some kind of security to assert that only moderators can approve or delete posts. And many more packages need a similar functionality for protect the creation, deletion and update of records. AOP enables you to move the security (or any other) aspect into its own package and leave the other objects with clear responsibilities, probably not implementing any security themselves. Some examples of some languages used are C++, Smalltalk, C#, C, and Java.

f). **Activity Oriented Programming Paradigm**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities. Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores.

g). **Event Oriented Programming Paradigm**

 Event-driven programming is a computer programming paradigm in which control flow of the program is determined by the occurrence of events. These events are monitored by code known as an event listener. If it detects that an assigned event has occurred, it runs an event handler (a callback function or method that's triggered when the event occurs).

In theory, all programming languages support the event-driven style of programming, although some language features, such as closures, make it easier to implement. Other programming environments, such as Adobe Flash, are specifically tailored for triggering program code by events.

1. **The Evolution Of Programming Languages In Chronological Order**

1951 – Regional Assembly Language

1952 – Autocode

1954 – IPL (forerunner to LISP)

1955 – FLOW-MATIC (led to COBOL)

1957 – FORTRAN (first compiler)

1957 – COMTRAN (precursor to COBOL)

1958 – LISP

1958 – ALGOL 58

1959 – FACT (forerunner to COBOL)

1959 – COBOL

1959 – RPG

1962 – APL

1962 – Simula

1962 – SNOBOL

1963 – CPL (forerunner to C)

1964 – Speakeasy

1964 – BASIC

1964 – PL/I

1966 – JOSS

1966 - MUMPS

1967 – BCPL (forerunner to C)

1967 – BCPL (forerunner to B)

1968 – Logo

1969 – B (forerunner to C)

1970 – Pascal

1970 – Forth

1972 – C

1972 – Smalltalk

1972 – Prolog

1973 – ML

1975 – Scheme

1978 – SQL (a query language, later extended)

1980 – C++ (as C with classes, renamed in 1983)

1983 – Ada

1984 – Common Lisp

1984 – MATLAB

1984 – dBase III, dBase III Plus (Clipper and FoxPro as FoxBASE, later developing into Visual FoxPro)

1985 – Eiffel

1986 – Objective-C

1986 – LabVIEW (Visual Programming Language)

1986 – Erlang

1987 – Perl

1988 – Tcl

1988 – Wolfram Language (as part of Mathematica, only got a separate name in June 2013)

1989 – FL (Backus)

1990 – Haskell

1991 – Python

1991 – Visual Basic

1993 – Lua

1993 – R

1994 – CLOS (part of ANSI Common Lisp)

1995 – Ruby

1995 – Ada 95

1995 – Java

1995 – Delphi (Object Pascal)

1995 – JavaScript

1995 – PHP

1997 – Rebol

2000 – ActionScript

2001 – C#

2001 – D

2002 – Scratch

2003 – Groovy

2003 – Scala

2005 – F#

2006 – PowerShell

2007 – Clojure

2009 – Go

2010 – Rust

2011 – Dart

2011 – Kotlin

2011 – Elixir

2012 – Julia

2012 - TypeScript

2014 – Swift

1. **DIFFERENCE BETWEEN MODULAR PROGRAMMING AND OBJECT ORIENTED PROGRAMMING.**

 An object-oriented program usually contains different types of objects, each corresponding to a particular kind of complex data to manage, or perhaps to a real-world object or concept such as a bank account, a hockey player, or a bulldozer. While

 Modular programming (also called "top-down design" and "stepwise refinement") is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

 Differences that I can think of are that you can have more than one objects on a class, where as in modular programming you are supposed to have only 1 module (1 object) for one specific thing.