St. Francis Institute of Technology (Engg. College)

Department of Information Technology

Second Year-Semester-III

Subject: Computer Programming Paradigm Lab

Year: 2023-24

Institute Vision, Mission, Quality Policy

Institute Vision

To be a chrysalis where bright youngsters are transformed into technological entrepreneurs and innovative leaders of tomorrow’s world, consistent with the Franciscan vision of integrity, peace and love

Institute Mission

To churn highly competent engineering graduates with a commitment to result oriented work, a perennial zest for learning, a quest for excellence, an open mind and the universal values of honesty, dignity and mutual care

Institute Quality Policy

To be the most preferred Institute of Engineering and Technology equipped with state-of-art facilities, to develop technically qualified techno -entrepreneurial class of value based global industry leaders, imparting comprehensive technical education, through highly competent and dedicated staff, ensuring continual improvement of QMS processes, meeting ISO 9001 -2015 statutory and regulatory requirements

Department Vision, Mission and Quality Policy

Department Vision

To create a conducive instrument for transforming the enrolled potential freshers into competent Information
Technology professional or entrepreneur with integrity and ethical value

Department Mission

1. To become unit of excellence in teaching, training, research, innovative applications and extension work in IT in co-operation with various departments

2. To make knowledge and expertise accessible with various dissemination strategies, including networking with research unit, colleges, government and industry along with the motivation for self-learning

3. To integrate teaching, research and practice along with higher education for generation and application of knowledge in line with emerging needs of industry, technical quality with market driven professional pursuits, programs, courses, collaboration.

4. To develop entrepreneur skills along with ethical and professional values among students

Department Program Educational Objectives (PEOs)

1. To provide an environment and to make knowledge & expertise accessible for students to work in multidisciplinary projects, to solve the real life problems with the help of modern tools and techniques and to lead towards a successful professional career.

2. To develop effective soft skills, inculcate team building capabilities such as leadership skills, managerial skills, and entrepreneurial skills and simultaneously nurture professional and ethical attitude in broad social context for sustainable development through lifelong learning.

3. To develop effective soft skills, inculcate team building capabilities such as leadership skills, managerial skills, and entrepreneurial skills and simultaneously nurture professional and ethical attitude in broad social context for sustainable development through lifelong learning.

Department Program Specific Outcomes (PSOs)

PSO1: Students will be able to acquire the basic knowledge of analysis and design, based on the comprehensive principles of Software Engineering, Project Management, Software Testing and Quality Assurance.

PSO2: Students will be able to apply research based approach using innovative tools and techniques in the field of Communication & Networks, Computer graphics & Image Processing and Information Security & Data Management.

PSO3: Students will be able to use the knowledge of Information Technology to develop end to end solutions in the field of SCAM (Social, Cloud, Analytics and Mobile).

PSO4: Students will be able to fuel entrepreneurship or to serve niche employment while portraying competencies like teamwork, efficient soft skills and a zeal for lifelong learning in order to contribute to society with moral and ethical values.

Program Objectives (POs)

1. Engineering knowledge (PO1)

Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.

2. Problem Analysis (PO2)

Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions (PO3)

Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems (PO4)

Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage (PO5)

Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modelling to complex engineering activities, with an understanding of the limitations.

6. The engineer and society (PO6)

Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability (PO7)

Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics (PO8)

Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work (PO9)

Function effectively as an individual, and as a member or leader in diverse teams, and in multi-disciplinary settings.

10. Communication (PO10)

Communicate effectively on complex engineering activities with the engineering community and with t h e society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations,

11. Project management and finance (PO11)

Demonstrate knowledge and understan ding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life- long learning (PO12)

Recognize the need for, and have the prepar ation and ability to engage in independent and life-long learning in the broadest context of technological change.
Graduates will be able to secure employment or be an entrepreneur with ability to apply professional knowledge with ethical responsibility.

and give and receive clear

LIST OF EXPERIMENTS

Sr. No.	Title of Experiment	CO Addressed	PSOs Addressed	PO Addressed
1	To demonstrate compilation and interpretation stages for C++ and JAVA	C303.1, C303.2	PSO2	PO1
2	To implement inheritance using C++ and JAVA	C303.1, C303.2		PO1
3	To implement Data Abstraction and encapsulation using C++/JAVA	C303.1, C303.2		PO1
4	Installation of Haskell Compiler and execute programs using arithmetic and logical operators	C303.3		PO1
5	To implement functions in Haskell Programming	C303.3		PO1
6	To implement list comprehensions using Haskell Programming	C303.3		PO1
7	To implement higher order and recursive functions using Haskell Programming	C303.3		PO1
8	To implement the game of nim using Haskell programming	C303.3		PO1
9	To install SWI Prolog and build knowledge base based on facts and rules	C303.4		PO1
10	To implement List comprehensions using Prolog	C303.4		PO1
11	To implement recursion and database manipulations using Prolog	C303.4		PO1
12	To implement logical expressions using Prolog	C303.4		PO1
13	To implement the concept of thread management and synchronization using concurrent programming	C303.5		PO2
14	To implement the concept of run time program management through exception handling using Java	C303.5		PO2
15	To implement dynamic website using alternative paradigm	C303.6	PSO2, PSO4	PO2

C303.1	To understand basic concepts of compilation and interpretation, compare and implement different programming paradigm concepts. (PSO2) (PO1)
C303.2	To understand and implement imperative programming paradigm through object oriented constructs. (PSO2) (PO1)
C303.3	To understand and implement declarative programming paradigm through functional programming (PSO2) (PO1)
C303.4	To understand, formulate and implement declarative programming paradigm through logic programming (PSO2) (PO1)
C303.5	To understand alternative paradigm through concurrent programming fundamentals and design, develop applications based on declarative paradigm(PSO2)(PO2)
C303.6	To understand alternative paradigm through scripting languages and formulate applications based on real life applications (PSO2,PSO4)(PO2)

Dr. Joanne Gomes and Ms Mrinmoyee Mukherjee
(Faculty in Charge)

Dr. Prachi Raut
(HOD-IT)

Mapping Course Outcomes (COs) to Program Outcomes (POs)

Lab Outcomes												
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
L303.1: To understand basic concepts of compilation and interpretation, compare and implement different programming paradigm concepts. (PSO2) (PO1)	**3**											
L303.2: To understand and implement imperative programming paradigm through object oriented constructs. (PSO2) (PO1)	**3**											
L303.3: To understand and implement declarative programming paradigm through functional programming (PSO2) (PO1)	**3**											
L303.4: To understand, formulate and implement declarative programming paradigm through logic programming (PSO2) (PO1)	**3**											
L303.5: To understand alternative paradigm through concurrent programming fundamentals and design, develop applications based on declarative paradigm(PSO2)(PO2)		**2**										
L303.6: To understand alternative paradigm through scripting languages and formulate applications based on real life applications (PSO2,PSO4)(PO2)		**2**										

Course Code	Course Name	Examination Scheme						
		Theory Marks				practs	oral	
		Internal assessment						
		Test1	Test 2	Avg. of 2 Tests				
ITL303	Computer Programming Paradigm Lab	--	--	--	--	25	25	50

Term Work:
Term Work shall consist of at least 15 practical's based on the above list.
Also Term work Journal must include at least 2 assignments. (Taken as descriptive tests)

Term Work Marks:
25 Marks (Total marks) = 15 Marks (Experiment) + 5 Marks (Assignments) + 5 Marks (Attendance)

Oral & Practical Exam:
An Oral & Practical exam will be held based on the above syllabus.

St. Francis Institute of Technology
Borivli (W), Mumbai 400103
Department of Information Technology

Experiment-1

1. Aim: **a) To implement inheritance using C++**
b) To implement inheritance using Java

2. Objectives: After performing the experiment, the students will be able to

- **Understand** the concept of inheritance
- **Execute** and **examine** single, multilevel and multiple inheritance

3. Lab Objective Mapped: To **understand** and **implement** imperative programming paradigm through object oriented constructs.

4. Prerequisite: Concept of classes and objects

5. Requirements: The following are the requirements – • **Internet connection**

- **Laptop/desktop with Windows/Linux/MAC operating system**
- **C++ and Java compiler**

6. Pre-Experiment Theory: Inheritance using C++

The capability of a class to derive properties and characteristics from another class is called inheritance. Inheritance is one of the most important features of Object Oriented Programming. While implementing inheritance two terms are important-
Sub Class/Derived Class: The class that inherits properties from other class is called subclass or derived class.
Super class/Base Class: The class whose properties are inherited by sub-class is called base class or super class. Syntax for creating sub-class in C++ is **class sub_class_name : visibility-mode base_class_name**

```
{
//body of subclass
};
```

Here sub_class_name is the name of the sub class, visibility-mode is the mode in which you want to inherit this subclass [public, private or protected] and base_class_name is the name of the base class from which you want to inherit the subclass. The colon indicates that the derived class name is derived from the base class name. The visibility mode is optional and if present, may be either **private** or **public.** The default visibility mode is **private.** The visibility mode specifies whether the features of the base class are privately derived or publicly derived. The visibility of the inherited members can be listed in the Table 1

Table 1: Access specifiers in C++

Base class visibility	Derived class visibility		
	Public derivation	Private derivation	Protected derivation
Private	Not inherited	Not inherited	Not inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

Inheritance using Java

The general format of implementing inheritance using Java is

```
class superclass
{
 // superclass data variables
 // superclass member functions
}
Class subclass extends superclass
{
 // subclass data variables
 // subclass member functions
}
```

The visibility mode of access specifier in Java are as listed in Table 2

Table 2: Access specifiers in Java

Visibility	Derived Class Visibility			
	Default	**Public**	**Protected**	**Private**
From same class	Yes	Yes	Yes	Yes
From same package subclass	Yes	Yes	Yes	No
Same package non sub-class	Yes	Yes	Yes	No
Different package sub-class	No	Yes	Yes	No
Different package non-subclass	No	Yes	No	No

Types of Inheritance

There can be three types of inheritances in Java- Single, Multilevel and Hierarchical as represented in Figure 1. In java programming, multiple and hybrid inheritance is supported only through interface.

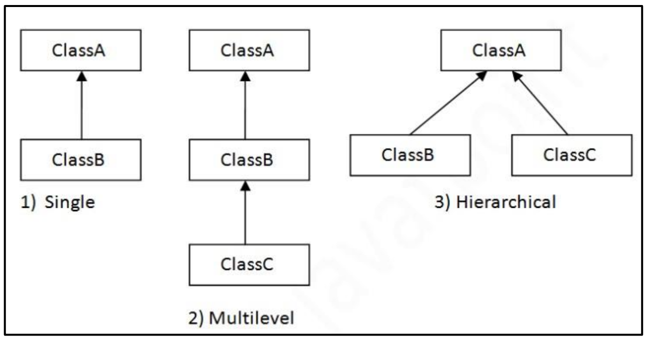


Figure 1: Types of inheritance in Java

In the above figure, in case of single inheritance Class B is the derived class and class A is the base class. In case of multilevel inheritance, Class C is derived from class B and class B is derived from class A. In hierarchical inheritance both class B and C are derived from class A. In figure 1 arrows are pointed towards the immediate base class(es). C++ supports single inheritance, multiple inheritance, multilevel inheritance, hierarchical inheritance and hybrid (virtual) inheritance as shown in Figure 2. In figure 2 the arrows are pointed towards the derived class

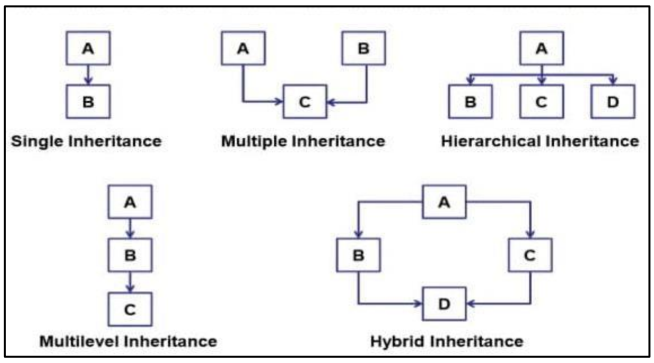


Figure 2: Types of inheritance in++

7. Lab Laboratory Exercise:

A. Procedure

- **Compilation using Turbo C++**
  - Launch the application
  - Select File, New to write a new file in C++
  - Save the file as .CPP
  - To compile the file, ALT+F9
  - To run the file, CTRL+F9
  - Check output

- **Compilation using JDK**
  - Launch text editor
  - Write your code and save as "abc.java"
  - Launch command prompt
  - Change directory to the path where your program is saved
  - Compile the code using "javac abc.java"

- Correct the errors if the compilation process gives error
- Run the program using “java abc”

B. Program Code

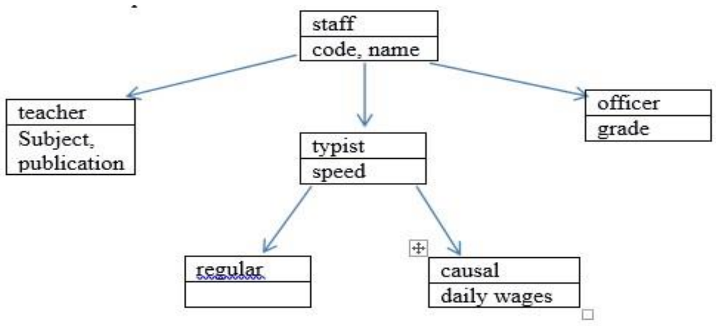
1. Write a program in C++ to implement single inheritance. Derive class typist class from class staff. Public members of class staff such as staff::getdata() and staff::display() are inherited to class typist. Access the member functions using objects of derived class.

2. Write a program in C++ to implement multiple inheritance. Create class Rectangle having two parent classes Area and Perimeter. Class 'Area' has a function getArea(int l, int b) which returns area. Class 'Perimeter' has a function getPerimeter (int l, int b) which returns the perimeter. Access the member functions using objects of derived class.
3. Write a program in Java to implement hierarchical inheritance. Create PermanentEmp class and TemporaryEmp class as the subclasses. Create class Employee as the super class that defines the salary of an employee. Class PermanentEmp has a hike of 35% of the amount defined in class Employee and Class TemporaryEmp has a hike of 20% of the amount defined in class Employee. Access the member functions using objects of derived class.

8. Post Experimental Exercise

A. Questions-

1. An educational institute wishes to maintain a database of its employees as illustrated in figure. The database is divided into a number of classes whose hierarchical relationships are shown in figure. Specify all the classes and define functions to create the database and retrieve individual information as and when required. **Prepare** the required classes to define the functions, **apply** the concepts of hierarchical inheritance to **implement** the above program in C++.



2. In the program codes 1, 2 and 3 **experiment** with change in the access specifiers to private and protected. **Analyze** the output with the change in access specifiers and summarize your observation.
3. **Implement** a C++ program to print the string in reverse order using the single inheritance in object-oriented programming. Also, **analyze** the output with the change in access specifiers and summarize your observation.

B. Results/Observations/Program Output

Present the program input/output results if any and comment on the same.

C. **Conclusion:**

1. Write what was performed in the experiment
2. Write which tools you used to perform the experiment
3. Write what you inferred from the output obtained

D. References:

[1] E Balaguguswamy, “Object Oriented Programming with C++”, second edition Tata McGraw Hill (Chapter 8)

[2] E Balaguguswamy, “Programming with Java-A primer”, third edition Tata McGraw Hill (Chapter 8)

St. Francis Institute of Technology
Borivli (W), Mumbai 400103
Department of Information Technology

Experiment-2

1. Aim: **a) To implement data abstraction in Java**
b) To implement data encapsulation using in Java

2. Objectives: After performing the experiment, the students will be able to

- **Understand** the concept of data abstraction and encapsulation
- **Execute** and **examine** data abstraction and encapsulation

3. Lab Objective Mapped: To **understand** and **implement** imperative programming paradigm through object-oriented constructs.

4. Prerequisite: Concept of classes and objects, inheritance

5. Requirements: The following are the requirements – • **Internet connection**

- **Laptop/desktop with Windows/Linux/MAC operating system**
- **Java compiler**

6. Pre-Experiment Theory:

DATA ABSTRACTION

Objects in an Object-Oriented Programming (OOP) language provide an abstraction that hides the internal implementation details. Data abstraction is a process to model/create our own user defined data types (using class and constructs) and then define variables (objects of those new data types). It is a simplified view of an object that includes only features one is interested in while hides away the unnecessary details. Data abstraction becomes an Abstract Data Type (ADT) or user defined type. Data abstraction in C++ is achieved using classes and objects. **Data abstraction in Java is achieved through interfaces and abstract classes.**

- The abstract keyword is a non-access modifier, used for classes and methods:
- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).An abstract class can have both abstract and regular methods
- An interface is a completely "**abstract class**" that is used to group related methods with empty bodies
- To access the interface methods, the interface must be "implemented" by another class with the implements keyword (instead of extends)
- The body of the interface method is provided by the "implement" class.

- On implementation of an interface, you must override all of its methods
- Interface methods are by default abstract and public
- Interface attributes are by default public, static and final

DATA ENCAPSUATION

It is a process of combining data members and functions in a single unit called class. The process is achieved using access modifiers. In encapsulation the variables and data in a class is hidden from any other class and can be accessed only through any member function of its own class in which it is declared. Encapsulation is achieved by declaring all the variables in the class as private and writing public methods in the class to set and get the values of the variables. Packages are also used to implement the concept of data encapsulation. Advantages of encapsulation includes-

- The field of a class can be made read only or write only
- The class can have total control over what is stored in its fields

7. Lab Laboratory Exercise:

A. Procedure

- **Compilation using JDK**
  - Launch text editor
  - Write your code and save as "abc.java"
  - Launch command prompt
  - Change directory to the path where your program is saved
  - Compile the code using "javac abc.java"
  - Correct the errors if the compilation process gives error
  - Run the program using "java abc"

B. Program Code

1. Write a program in Java to implement data encapsulation. Create class EncapTest that has private data members- name, age, idno of an employee. Get and set the values of the private data members using public methods- getName(), getAge(), getIdno(), setName(), setAge() and setIdno().

2. Write a program in Java to implement abstract classes. Create an abstract class animal having an abstract method animalSound(). Create subclass dog from superclass animal. Access the abstract class method by creating the object of subclass to implement data abstraction.

4. 8. Post Experimental Exercise

A. Questions-

1. **Implement** a program in Java to calculate the area of a rectangle, a square and a circle. Create an abstract superclass 'Shape' with three abstract methods namely 'RectangleArea' taking two parameters and 'SquareArea' and 'CircleArea' taking one parameter each. The

parameters of 'RectangleArea' are its length and breadth. The parameters of 'SquareArea' is its side and that of 'CircleArea' is its radius. Also create subclass ‘area’. **Apply** the concept of data abstraction to access the methods of the superclass by creating the object of subclass.

2. What is data hiding in Object Oriented Programming paradigm? **Describe** how data hiding is implemented in Java.

B. Results/Observations/Program Output

Present the program input/output results if any and comment on the same.

C. **Conclusion:**

1. Write what was performed in the experiment
2. Write which tools you used to perform the experiment
3. Write what you inferred from the output obtained

D. References:

[1] E Balaguguswamy, “Programming with Java-A primer”, third edition Tata McGraw Hill

[2] https://nptel.ac.in/courses/106/105/106105191/ [Week-3, Lecture 15, Week-4, Lecture 17 and 18]

St. Francis Institute of Technology
Borivli (West), Mumbai-400103
Department of Information Technology

Experiment – 3

1. **Aim:** To perform arithmetic and logical operators

2. **Objective:** After performing the experiment, the students will be able to understand and implement–
   - Download and install Haskell Compiler
   - Perform basic arithmetic operations
   - Perform basic logical operation

3. **Lab objective mapped:** To **understand** and **implement** declarative programming paradigm through functional programming (PSO2) (PO1)

4. **Prerequisite:** Nil

5. **Requirements:** The following are the requirements – Haskell Compiler
   - Arithmetic and logical operators in Haskell

6. **Pre-Experiment Theory:**

Functional Programming

Functional programming is a programming paradigm — a style of building the structure and elements of computer programs — that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
Advantages of functional programming include
- Functional programming is based on mathematical functions.
- Easier to determine inputs.
- Easier to determine outputs.
- Easier to demonstrate prove that you have a correct program.
- Easier to test programs that are too difficult to prove

Limitations of Functional Programming include-.
- Functional programming paradigm is not easy, so it is difficult to understand for the beginner
- Hard to maintain as many objects evolve during the coding
- Needs lots of mocking and extensive environmental setup
- Re-use is very complicated and needs constantly refactoring
- Objects may not represent the problem correctly

Arithmetic Operations in Haskell

The following are the arithmetic operators supported by Haskell
- Addition Operator-Addition operator (+) is used for addition in Haskell.
- Subtraction Operator-Subtraction operator (-) is used for subtraction in Haskell.

- Multiplication Operator-Multiplication operator (*) is used for multiplication in Haskell. Division Operator-Division operator (/) is used for division in Haskell.
- Sequence/Range Operator-It's a special operator (..) in Haskell. It is used while declaring the list with a sequence of values.
- Trigonometric operations- Haskell also supports Trigonometric operations. The trigonometric operations supported by Haskell are- sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, asinh, atanh, acosh.

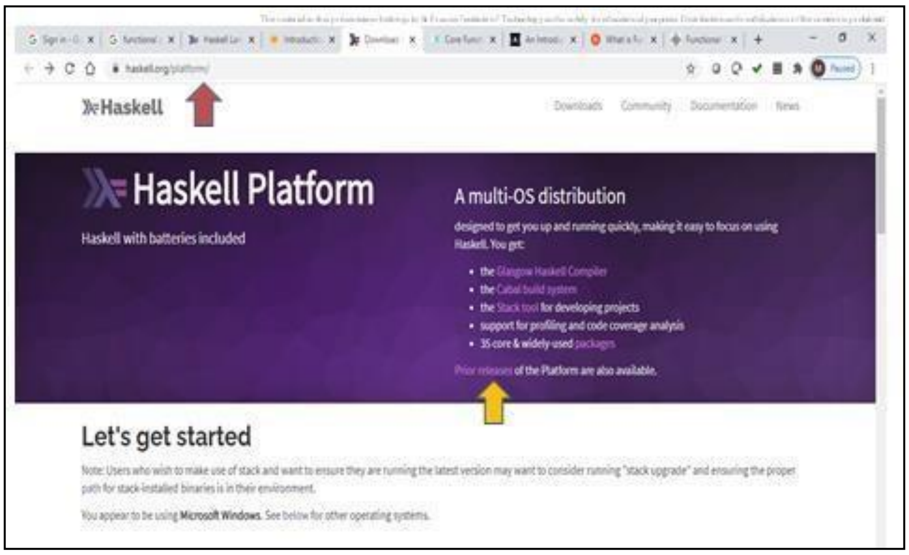
The other Mathematical operations supported by Haskell are pi, exp, sqrt, log and (**). Special rule while using negative numbers-The minus operator is Haskell's only unary arithmetic operator (or not?), i.e. it takes a single argument.

Logical and Comparison Operations in Haskell Haskell supports the following logical operations

- Logical AND (&&)
- Logical OR(||)
- Equality (==)
- Non Equality(/=)
- Lesser than(<)
- Lesser than equal to (<=)
- Greater than (>)
- Greater than equal to (>=)

7. Laboratory Exercise
A. Procedure for installation of Haskell compiler-

Step-1: Visit https://www.haskell.org/ and click on prior releases



haskell.org/platform/

Haskell

Downloads Community Documentation News

Haskell Platform

Haskell with batteries included

A multi-OS distribution

designed to get you up and running quickly, making it easy to focus on using Haskell. You get:

- the Glasgow Haskell Compiler
- the Cabal build system
- the Stack tool for developing projects
- support for profiling and code coverage analysis
- 35 core & widely-used packages

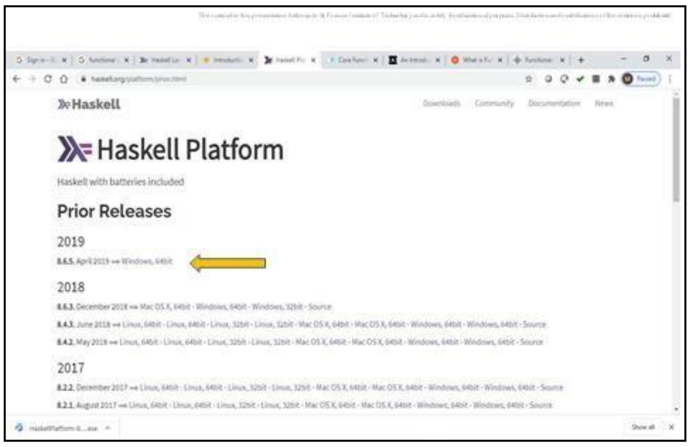
Prior releases of the Platform are also available.

Let's get started

Note: Users who wish to make use of stack and want to ensure they are running the latest version may want to consider running "stack upgrade" and ensuring the proper path for stack-installed binaries is in their environment.

You appear to be using **Microsoft Windows**. See below for other operating systems.

Step-2: Download 8.6.5 (April 2019) release for 64 bit windows machine



Haskell

Downloads Community Documentation News

Haskell Platform

Haskell with batteries included

Prior Releases

2019

8.6.5, April 2019 ⟶ Windows, 64bit

2018

8.6.3, December 2018 ⟶ Mac OS X, 64bit · Windows, 64bit · Windows, 32bit · Source

8.4.3, June 2018 ⟶ Linux, 64bit · Linux, 64bit · Linux, 32bit · Linux, 32bit · Mac OS X, 64bit · Mac OS X, 64bit · Windows, 64bit · Windows, 64bit · Source

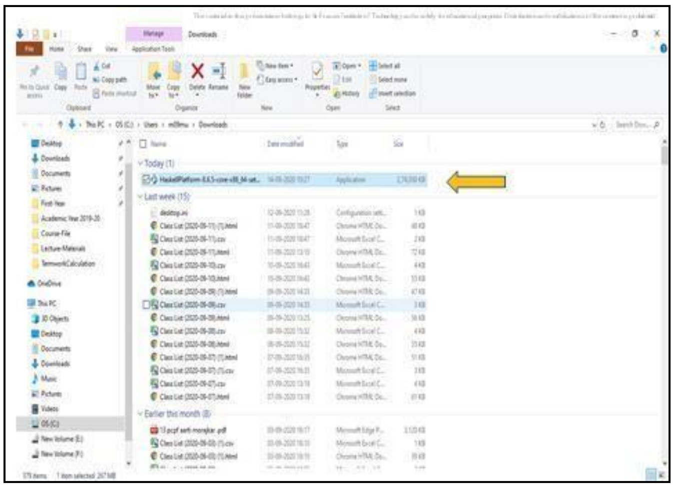
8.4.2, May 2018 ⟶ Linux, 64bit · Linux, 64bit · Linux, 32bit · Linux, 32bit · Mac OS X, 64bit · Mac OS X, 64bit · Windows, 64bit · Windows, 64bit · Source

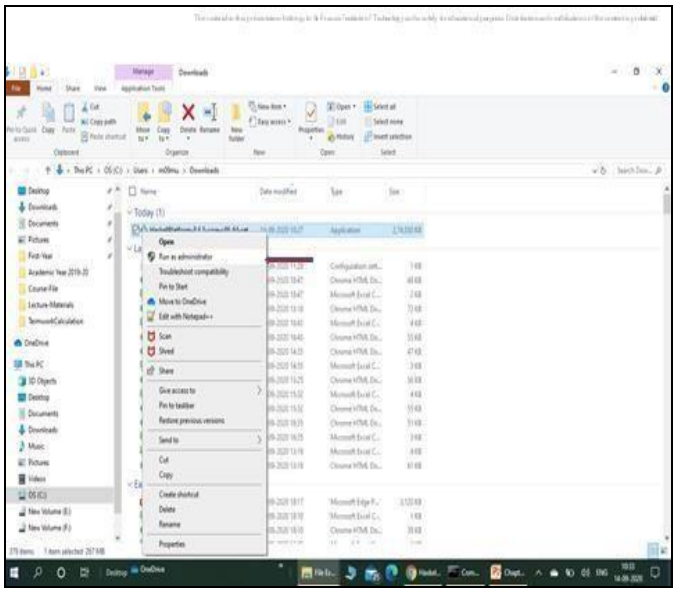
2017

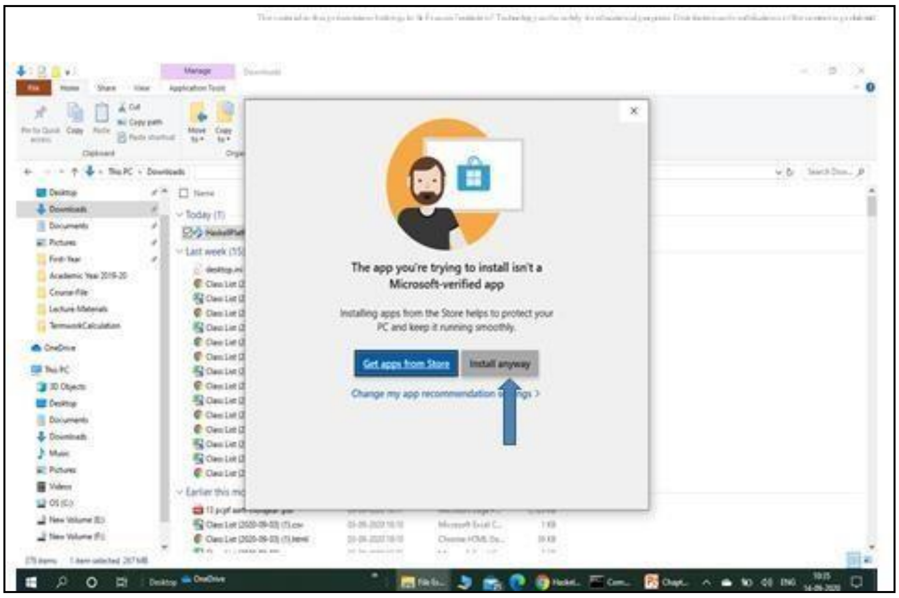
8.2.2, December 2017 ⟶ Linux, 64bit · Linux, 64bit · Linux, 32bit · Linux, 32bit · Mac OS X, 64bit · Mac OS X, 64bit · Windows, 64bit · Windows, 64bit · Source

8.2.1, August 2017 ⟶ Linux, 64bit · Linux, 64bit · Linux, 32bit · Linux, 32bit · Mac OS X, 64bit · Mac OS X, 64bit · Windows, 64bit · Windows, 64bit · Source

Step-3: In the Downloads folder, run the application as an administrator





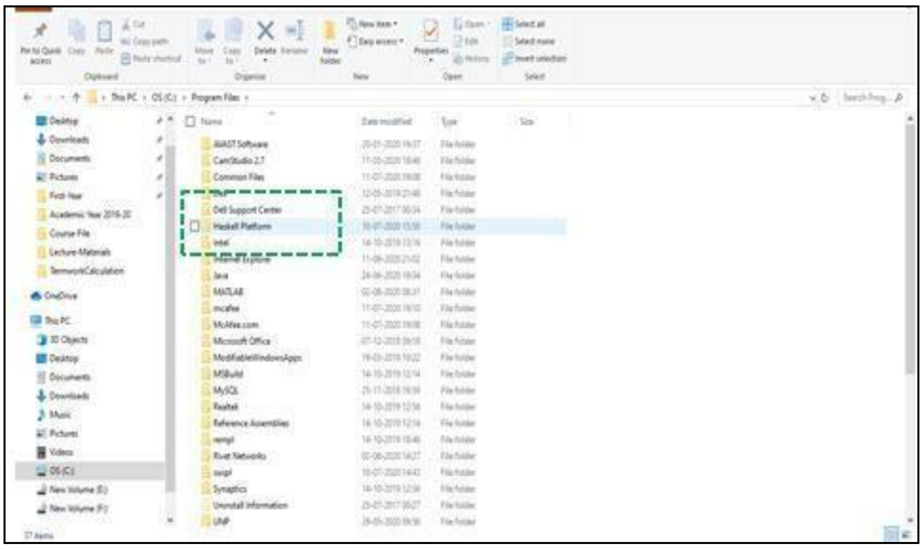


The app you're trying to install isn't a Microsoft-verified app

Installing apps from the Store helps to protect your PC and keep it running smoothly.

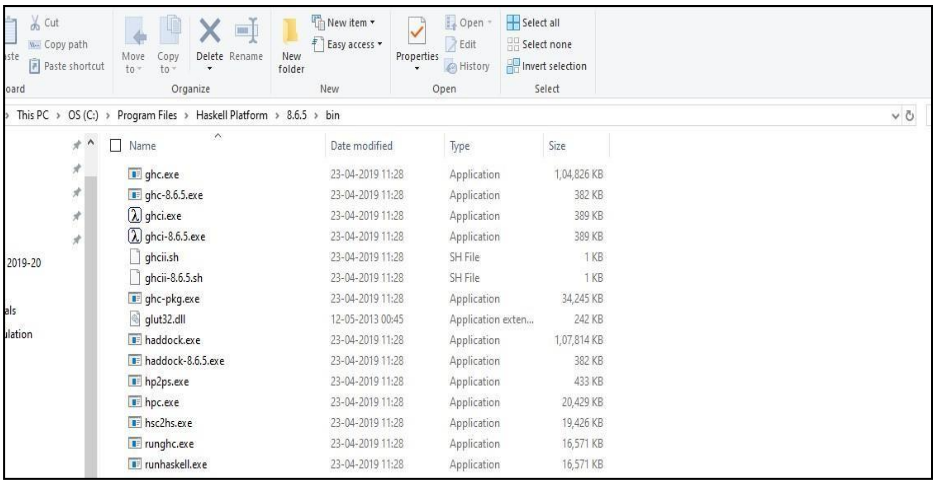
After successful installation it will be loaded in the path ‘C:\Program Files\Haskell Platform’ (by default)

Step-4: Copy the bin path



```
This PC > OS (C:) > Program Files >

Name                        Date modified      Type
AVAST Software              20-01-2020 19:37   File folder
CamStudio 2.7               11-05-2020 18:40   File folder
Common Files                11-07-2020 19:08   File folder
Dell                        12-05-2019 21:48   File folder
Dell Support Center         25-07-2017 00:34   File folder
Haskell Platform            10-07-2020 15:58   File folder
Intel                       14-10-2019 12:16   File folder
Internet Explorer           11-09-2020 21:02   File folder
Java                        24-06-2020 19:34   File folder
MATLAB                      02-08-2020 08:31   File folder
mcafee                      11-07-2020 18:10   File folder
McAfee.com                  11-07-2020 19:08   File folder
Microsoft Office            07-12-2018 09:18   File folder
ModifiableWindowsApps       19-03-2019 10:22   File folder
MSBuild                     14-10-2019 12:14   File folder
MySQL                       25-11-2018 19:59   File folder
Realtek                     14-10-2019 12:56   File folder
Reference Assemblies        14-10-2019 12:14   File folder
rempl                       14-10-2019 18:46   File folder
Rivet Networks              02-06-2020 14:27   File folder
swipl                       10-07-2020 14:43   File folder
Synaptics                   14-10-2019 12:56   File folder
Uninstall Information       25-07-2017 00:27   File folder
UNP                         28-05-2020 09:56   File folder
```

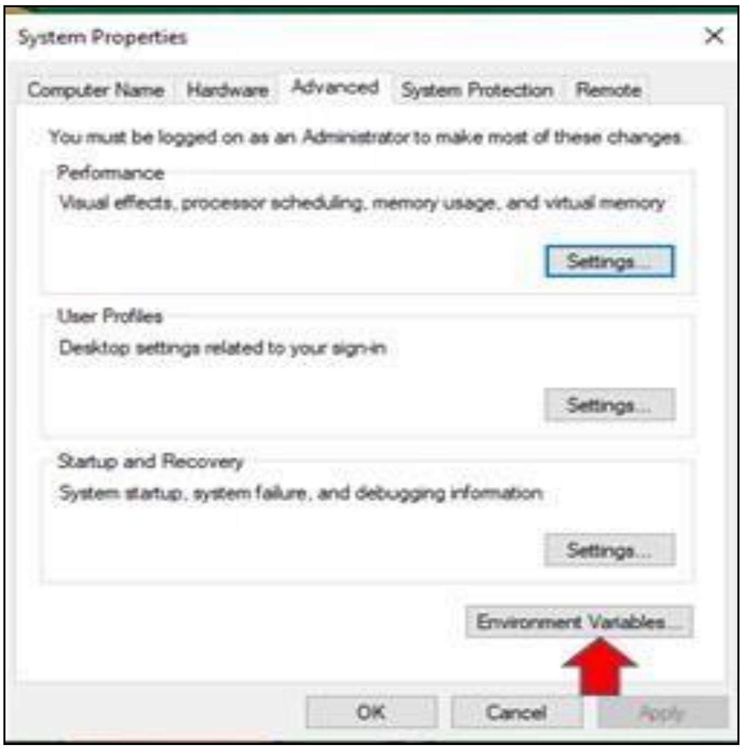


```
This PC > OS (C:) > Program Files > Haskell Platform > 8.6.5 > bin

Name               Date modified      Type                    Size
ghc.exe            23-04-2019 11:28   Application             1,04,826 KB
ghc-8.6.5.exe      23-04-2019 11:28   Application             382 KB
ghci.exe           23-04-2019 11:28   Application             389 KB
ghci-8.6.5.exe     23-04-2019 11:28   Application             389 KB
ghcii.sh           23-04-2019 11:28   SH File                 1 KB
ghcii-8.6.5.sh     23-04-2019 11:28   SH File                 1 KB
ghc-pkg.exe        23-04-2019 11:28   Application             34,245 KB
glut32.dll         12-05-2013 00:45   Application exten...    242 KB
haddock.exe        23-04-2019 11:28   Application             1,07,814 KB
haddock-8.6.5.exe  23-04-2019 11:28   Application             382 KB
hp2ps.exe          23-04-2019 11:28   Application             433 KB
hpc.exe            23-04-2019 11:28   Application             20,429 KB
hsc2hs.exe         23-04-2019 11:28   Application             19,426 KB
runghc.exe         23-04-2019 11:28   Application             16,571 KB
runhaskell.exe     23-04-2019 11:28   Application             16,571 KB
```

Step-5: Set the system variables

Go to advanced system settings and open the system properties tab. Select the environment variables.



System Properties

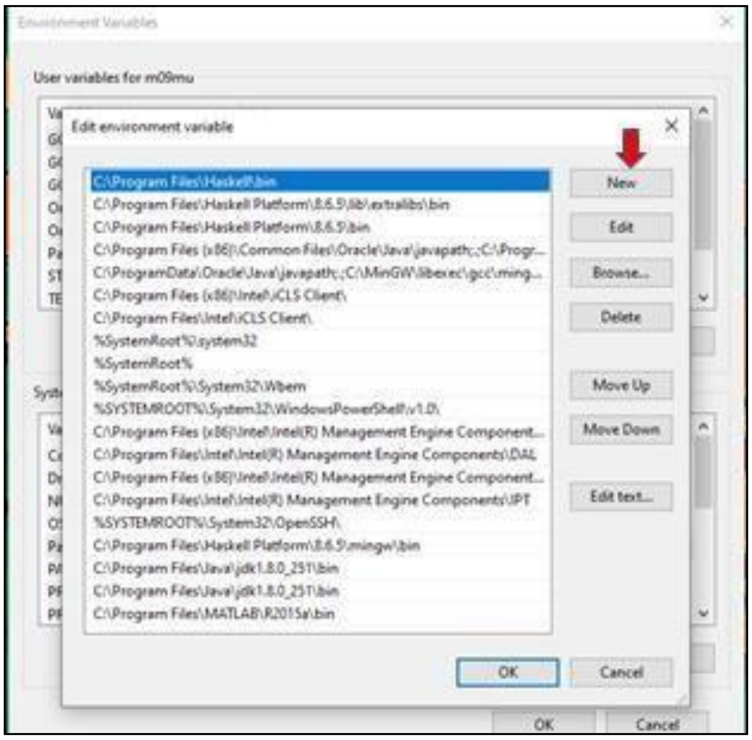
You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your sign-in

Startup and Recovery
System startup, system failure, and debugging information

Set the system variables path as ‘C:\Program Files\Haskell Platform\bin’

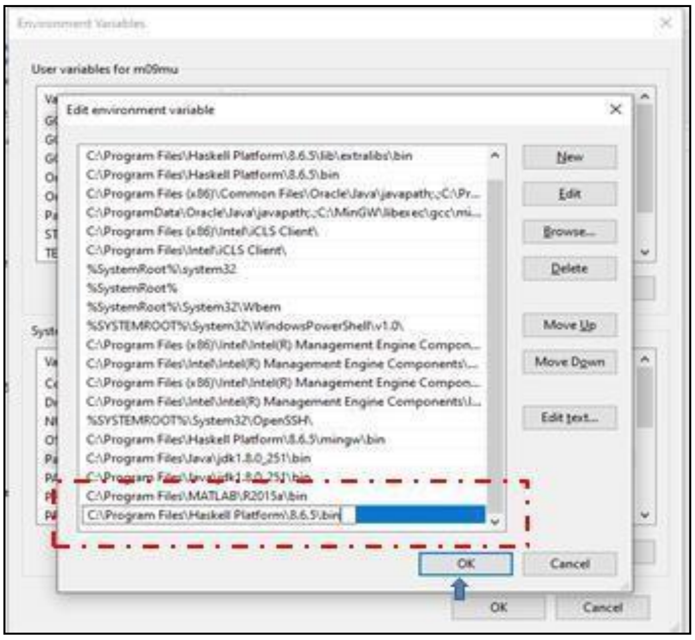


Environment Variables

User variables for m09mu

Edit environment variable

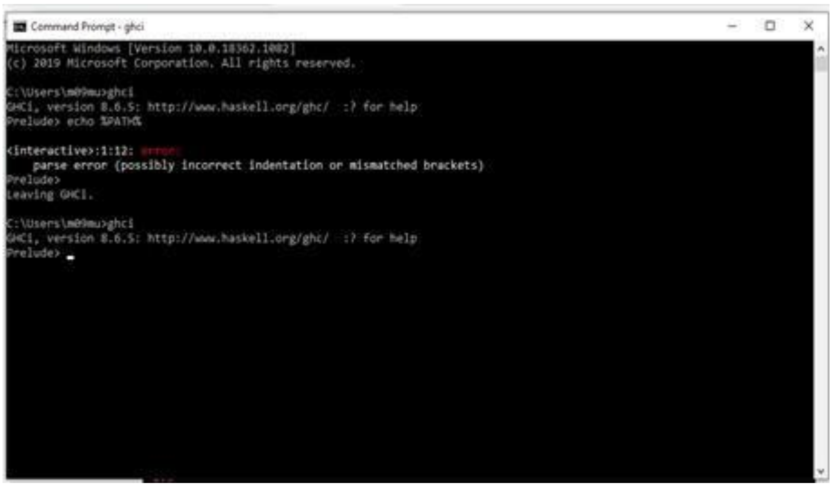
```
C:\Program Files\Haskell\bin
C:\Program Files\Haskell Platform\8.6.5\lib\extralibs\bin
C:\Program Files\Haskell Platform\8.6.5\bin
C:\Program Files (x86)\Common Files\Oracle\Java\javapath;;C:\Progr...
C:\ProgramData\Oracle\Java\javapath;;C:\MinGW\libexec\gcc\ming...
C:\Program Files (x86)\Intel\iCLS Client\
C:\Program Files\Intel\iCLS Client\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
C:\Program Files (x86)\Intel\Intel(R) Management Engine Component...
C:\Program Files\Intel\Intel(R) Management Engine Components\DAL
C:\Program Files (x86)\Intel\Intel(R) Management Engine Component...
C:\Program Files\Intel\Intel(R) Management Engine Components\IPT
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Haskell Platform\8.6.5\mingw\bin
C:\Program Files\Java\jdk1.8.0_251\bin
C:\Program Files\Java\jdk1.8.0_251\bin
C:\Program Files\MATLAB\R2015a\bin
```



```
C:\Program Files\Haskell Platform\8.6.5\lib\extralibs\bin
C:\Program Files\Haskell Platform\8.6.5\bin
C:\Program Files (x86)\Common Files\Oracle\Java\javapath;.;C:\Pr...
C:\ProgramData\Oracle\Java\javapath;.;C:\MinGW\libexec\gcc\mi...
C:\Program Files (x86)\Intel\iCLS Client\
C:\Program Files\Intel\iCLS Client\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
C:\Program Files (x86)\Intel\Intel(R) Management Engine Compon...
C:\Program Files\Intel\Intel(R) Management Engine Components\...
C:\Program Files (x86)\Intel\Intel(R) Management Engine Compon...
C:\Program Files\Intel\Intel(R) Management Engine Components\I...
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Haskell Platform\8.6.5\mingw\bin
C:\Program Files\Java\jdk1.8.0_251\bin
C:\Program Files\Java\jdk1.8.0_251\bin
C:\Program Files\MATLAB\R2015a\bin
C:\Program Files\Haskell Platform\8.6.5\bin
```

Step-6: Check the compiler from command prompt

Open command prompt, at the prompt, type-‘ghci’. If the Haskell compiler is successfully installed, it will change the prompt to ‘Prelude’. You are now ready to work with Haskell



```
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\m09mu>ghci
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> echo %PATH%

<interactive>:1:12: error:
    parse error (possibly incorrect indentation or mismatched brackets)
Prelude>
Leaving GHCi.

C:\Users\m09mu>ghci
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> _
```

B. Program Code

1. **Execute the following Arithmetic Operations at Prelude prompt. Carefully note down the outputs.**

   2+3
   2-3
   2+(-3)
   2*3
   2/3
   3/2
   (40*100)-3000+50/5
   (40*100-3000+50)/5
   sqrt(36) sqrt(36)+sqrt(25)
   2^3

2**3 2**3+2^3 sin(90)
asin(90) asin (pi/4) asin (pi/6)
acos(pi/6) asinh (pi/4) acosh
(pi/2) acosh (pi/4) acosh
(pi/2)+acosh(pi/4) acosh
(pi/2)+atanh(pi/4)+2**3

2. **Execute the following Logical and Comparasion Operations at Prelude prompt. Carefully note down the outputs.**
   True && True
   True && False
   False && True
   False && False
   True|| True
   True || False
   False || True False
   || False not(True)
   not(False)
   100>3
   100<3
   100==100 100/=100 not(100/=100)
   not(100/=100)&& True
   not(100/=100)&& (100>3)
   not(100/=100)&& (100>3)|| (2==3)
   not(not(100/=100)&& (100>3)|| (2==3))

3. Write a program in Haskell to display “Hello SFIT”.
4. Write a program in Haskell to add and subtract two integer numbers

8. Post Experimental Exercise-

A. Questions:

1. List basic data types in Haskell and Describe the features in Haskell
2. Write a program in Haskell to add, subtract, multiply two fractional numbers. Take the numbers from the users.
3. Write a program in Haskell to evaluate the mathematical operations Sqrt(36)+sqrt(25)

B. Results/Observations/Program output:
Present the program input/output results if any and comment on the same.

C. Conclusion:
1. Write what was performed in the experiment
2. Write which tools you used to perform the experiment
3. Write what you inferred from the output obtained

D. References:
[1] https://www.haskell.org/
[2] http://learnyouahaskell.com/
[3] Michael L Scott, “ Programming Language Pragmatics”, Third edition, Elsevier publication

Experiment – 4

1. **Aim: a)** To implement functions in Haskell Programming
   b) To implement recursive functions in Haskell Programming

2. **Objective:** After performing the experiment, the students will be able to understand and implement–
   - Functions in Haskell programming
   - Develop small user defined and recursive functions using Haskell programming

3. **Lab objective mapped:** To **understand** and **implement** declarative programming paradigm through functional programming (PSO2) (PO1)

4. **Prerequisite:** Knowledge of arithmetic operations, Lambda calculus and functions implementation

5. **Requirements:** The following are the requirements –
   - Haskell Compiler

6. **Pre-Experiment Theory:**

Lambda Calculus

Lambda calculus (also written as λ**-calculus**) is a formal system in mathematical logic for expressing computation based on function abstraction and application using variable binding and substitution. Lambda calculus involves writing **Lambda Abstraction** for the function to be implemented. Lambda abstraction consists of the following sub parts-Function signifier, parameter variable and return expression as shown in figure 1. Figure 1 illustrates

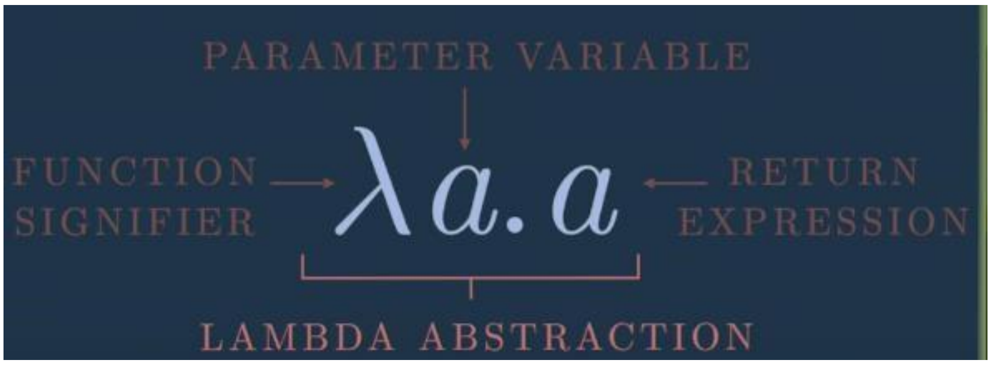


Figure 1: Lambda abstraction for identity function

Example- The Lambda abstraction for succ function [f(x)=x+1] can be written as [x. x+1]. In the expression the symbol lambda (λ) is the abstraction operator, x is the parameter variable and x+1 is the return expression.

Functions

Functions in Haskell are based on Lambda Calculus. Functions play a major role in Haskell. Function declaration consists of the function name and its argument list along with its output function definition. Functions in Haskell consists of the following sub- parts-

- Function declaration
- Function definition

1

- Function calling

Instead of using equations to define functions, they are defined "anonymously" via a *lambda abstraction*. For example, a function equivalent to inc could be written as \x -> x+1. Similarly, the function add is equivalent to \x -> \y -> x+y.

Example:

```
         add :: Integer -> Integer -> Integer                    --function declaration
     add x y = x + y                                             --function definition

main = do putStrLn "The addition of the two numbers is:"
        print(add 2 5)                  --calling a function
```

Some of the important characteristics of functions are –

- Haskell functions are applied to only one argument
- Function application is evaluated left to right, i.e. is left associative.
- This means that a function f of two arguments f x y is correctly interpreted as f being applied to x to give a function which is applied to y, i.e. f x y == (f x) y

7. Laboratory Exercise
A. Steps to be implemented

1. Open notepad/text editor
2. Write your function code and save the file as ‘filename.hs’ file.
3. While saving make sure to keep *save as type* as ‘All files’
4. Open command prompt and type ‘ghci’.This will take you to ‘Prelude’ prompt.
5. Change the path (if required) to the place where you have saved your file using the following command **prelude>: cd** ‘write your complete path here’
6. To compile, load the program using the following command **prelude>: l** your-file name [do not write the file name with .hs extension]
7. To run, write the name of the function at prelude prompt **Main>: name of function**
8. Check the output

B. Program Code

1. Implement the following inbuilt functions using Haskell compiler at prelude prompt

   1. succ 6
   2. succ (succ 5)
   3. min 5 6   4. max 5 6

5. max 101 101
6. succ 9 + max 5 4 + 1
7. (max 5 4)+(succ 9)+1
8. (succ 9) + (max 5 4) + 1
9. We wanted to get the successor of the product of numbers 9 and 10. we couldn't write **succ 9 * 10** because that would get the successor of 9, which would then be multiplied by 10
10. Succ 9*10
11. succ (9*10)
12. div 92 10
13. div 3 4
14. div 4 3
15. 4/3
16. mod 7 5
17. mod 3 1
18. mod 7 2
19. x=45
20. print x
21. return True
22. return False
23. x<-return 35
24. print x
25. putStrLn “Hello”
26. y<-getLine
27. print y

2. Write a higher order function to –

a. Write a function applyTwice to add and multiply two numbers that can take functions as parameters and also return functions.
b. Write a function multThree ((multThree 3) 5) 9 that can take functions as parameters and also return functions.
c. Write a function applyString to append two strings using higher order functions
d. Write a program in Haskell to evaluate factorial of a number using recursion
e. Write a program in Haskell to reverse a string using recursion

3. Write a function in Haskell to implement lambda abstraction for

a. Addition, subtraction, multiplication and division of two numbers for number (integer) inputs
b. Exponent function x^y. The base and power are of type number (integer)
c. Square root of a number [x^0.5]
d. Add two numbers and then evaluate its square root

For all the above function, write the expression for lambda abstraction

9. Post Experimental Exercise-

A. Questions:

1. List important characteristics of functions in Haskell

2. Write the lambda abstraction expression for-
 Unity operation
- succ 6 operation
- plus n (succ 16) operation
- Multiplication by 2 operation

3. Write a function in Haskell to evaluate –
 Area of a triangle
- Area of a circle
- Implement XOR operation

4. Implement the following at the prelude prompt

```
Prelude>zipWith (+) [4, 2, 5, 6] [2, 6, 2, 3]
Prelude> zipWith max [6, 3, 2, 1] [7, 3, 1, 5]
Prelude> zipWith (++) ["foo ", "bar ", "baz "] ["fighters", "hoppers", "aldrin"]
Prelude> zipWith (*) (replicate 5 2) [1..]
Prelude> map (map (^2)) [[1,2],[3,4,5,6],[7,8]]
Prelude> map (++ "!") ["BIFF", "BANG", "POW"]
Prelude> map (replicate 3) [3..6]
```

5. Write a function flip that simply takes a function and returns a function that is like our original function.
6. Write a function dividebyTen that divides the argument by 10

B. Results/Observations/Program output:

Present the program input/output results if any and comment on the same.

C. Conclusion:

1. Write what was performed in the experiment
2. Write which tools you used to perform the experiment
3. Write what you inferred from the output obtained

D. References:

[1] https://www.haskell.org/
[2] http://learnyouahaskell.com/
[3] Michael L Scott, " Programming Language Pragmatics", Third edition, Elsevier publication

St. Francis Institute of Technology
Borivli (West), Mumbai-400103
Department of Information Technology

Experiment – 5

1. Aim: To implement list comprehensions using Haskell Programming

2. Objective: After performing the experiment, the students will be able to –

- Create and display lists□
- Perform list comprehensions operations□
- Develop small user defined functions using Haskell programming□

3. Lab objective mapped: To **understand** and **implement** declarative programming paradigm through functional programming (PSO2) (PO1)

4. Prerequisite: Knowledge of arithmetic operations, logical and comparative operations

5. Requirements: The following are the requirements –

- Haskell Compiler□

6. Pre-Experiment Theory:

List in Haskell is very important

- It's the most used data structure□
- In Haskell, lists are a **homogenous** data structure. It stores several elements of the same type.□
- That means that we can have a list of integers or a list of characters but we can't have a list that has a few integers and then a few characters.□

List in Haskell are denoted by square brackets and the values in the list are separated by commas.
Examples

Prelude> let x1=[1,2,3,4,5,6]
Is a list consisting of Num type of data. The list is stored in variable x1.

Prelude>let x2=[1.1,2.1,3.1,4.1,5.1,6.1]
Is a list consisting of fractional type of data. The list is stored in variable x2.

Prelude>let x3=[‘a’,’b’,’c’]
Is a list consisting of character type of data. The list is stored in variable x3.

Prelude>let y=[1,2,3,’c’, ‘a’]
The above example will give error as list y consist of heterogeneous data types.

The List comprehension commands are as under-

- **head**- Takes a list and returns its head. The head is the first element of the list.
- **tail** -Takes a list and returns its tail. The tail in list consists of the remaining elements of list
- **last**- Takes a list and returns its last element.
- **init**- Takes a list and returns everything except its last element.
- **length**- Takes a list and returns its length.
- **null**- Checks if the list is a null list. If the list is a null list then it returns true else false.
- **reverse**-Reverses a given list
- **take**- Takes a number and a list. It extracts that many elements from a list.
- **drop**- Takes a number and a list. It drops that many numbers from the beginning of a list.
- **maximum**- Takes a list and returns the maximum value of the list.
- **minimum**- Takes a list and returns the minimum value of the list.
- **sum**- Takes a list and evaluates the sum of all numbers of a list.
- **product**- Takes a list and evaluates the product of all numbers of a list.
- **elem**-Takes a thing and a list of things and tells us if that thing is an element of list

7. Laboratory Exercise

A. Steps to be implemented

To work on Prelude prompt, use the command ***ghci.*** To work on Haskel files, follow the instructions given below

1. Open notepad/text editor
2. Write your function code and save the file as 'filename.hs' file.
3. While saving make sure to keep *save as type* as 'All files'
4. Open command prompt and type 'ghci'.This will take you to 'Prelude' prompt.
5. Change the path (if required) to the place where you have saved your file using the following command **prelude>: cd** 'write your complete path here'
6. To compile, load the program using the following command **prelude>: l** your-file name [do not write the file name with .hs extension]
7. To run, write the name of the function at prelude prompt **Main>: name of function**
8. Check the output

B. Program Code

1. Execute the following inbuilt functions using Haskell compiler at prelude prompt
   - Prelude> x=[1,2,3]
   - Prelude> print x
   - Prelude> let x1=['a','b']
   - Prelude> print x1
   - Prelude> [1,2,3,4]++[5,6,7]
   - Prelude> [1.1,2.1,3.1,4.1]++[5.1,6.1,7.1]
   - Prelude>['a','b']++['s','v']
   - Prelude> "hello"++" "++"world"
   - Prelude> 'A': "Small Dog"
   - Prelude> 'A':" "++"Small Dog"
   - Prelude> '1':" "++"Small Dog"

- Prelude> 6:[1,2,3,4]
- Prelude> 6.1:[1,2,3,4]
- Prelude> 6.1:[1,2,3,4]++ 4:[1,2,3,4]
- Prelude> []
- Prelude> "Steve Buscemi" !!4
- Prelude> let x="Steve Buscemi" !!4
- Prelude> print x
- Prelude> let b=[[1,2,3],[4,5,6],[7,8,9]]
- Prelude> print b
- Prelude> b ++[[1,1,1]]
- Prelude> let y1=b++[[1,1,1]]
- Prelude> print y
- Prelude>length [4,5,6,8]
- Prelude> length []
- Prelude> length [[]]
- Prelude> length [[[]]]
- Prelude> length [[],[]]
- Prelude> reverse [7,5,4,9,10]
- Prelude> take 3 [7,5,4,9,10]
- Prelude> drop 3 [7,5,4,9,10]
- Prelude>head [7,5,4,9,10]
- Prelude> tail [7,5,4,9,10]
- Prelude> init[7,5,4,9,10]
- Prelude> last [7,5,4,9,10]
- Prelude> sum[7,5,4,9,10]
- Prelude> product[7,5,4,9,10]
- Prelude> maximum[4,6,7,8,9]
- Prelude> minimum[2,4,5,6,7]
- Prelude> 4 `elem` [6,7,8,4]
- Prelude> zipWith(+)[1,2,3] [4,5,6]
- Prelude> zipWith(-)[1,2,3] [4,5,6]
- Prelude> zipWith(*)[1,2,3] [4,5,6]
- Prelude> zipWith(/)[1,2,3] [4,5,6]

2. Perform the following list comprehensions using Haskell compiler at the prelude prompt and explain each comprehension in your own words
   - Prelude>[x*2 | x <- [1..10]]
   - Prelude> [x*2 | x <- [1..100], x ‘mod’ 2 == 0]
   - Prelude> [x | x <- [10..20], x /= 13, x /= 15, x /= 17]
   - Prelude> [x | x <- [1..100], x ‘mod’ 7 == 0, x >=50]
   - Prelude> [x^2 | x <- [1..10]]
   - Prelude> [2^x|x<-[1..20]]
   - Prelude> let nouns=["hobo","frog","pope"]
   - Prelude> let adj=["lazy", "scram","grouchy"]
   - Prelude> [adj++" "++noun|adj<-adj,nouns<-nouns]
   - Prelude> fst(8,9)
   - Prelude> snd(8,0)
   - Prelude> snd(8,"wow")

- Prelude> zip [1,2,3,4] [3,3,3,3]
- Prelude> zip [1..5] [3,3,3,3]
- Prelude> zip [1..5] ["a","b","c","d","e"]

3. Write a function in Haskell to

- Define a function doublePost that doubles the positive elements in the list of integers
- Define a function **spaces n** which returns a string of n spaces
- Generate a list of **triples (x,y,z)** such that $x^2+y^2=z^2$
- Define a function **factor n** which returns a list of integers that divide n. Omit the trival factors of 1 and n

9. Post Experimental Exercise-

A. Questions:

1. List important characteristics of Lists in Haskell
2. Write a Haskell snippet to create a list of first 10 numbers in numbers
3. Write a Haskell snippet to filter the elements from the list whose square root is greater than seven
4. Write a Haskell snippet to implement Fibonacci series. Define an expression fibs :: [Integer] that generates this infinite sequence.

B. Results/Observations/Program output:

Present the program input/output results if any and comment on the same.

C. Conclusion:

1. Write what was performed in the experiment
2. Write which tools you used to perform the experiment
3. Write what you inferred from the output obtained

D. References:

[1] https://www.haskell.org/

[2] http://learnyouahaskell.com/

[3] Michael L Scott, " Programming Language Pragmatics", Third edition, Elsevier publication

St. Francis Institute of Technology
Borivli (W), Mumbai 400103
Department of Information Technology

Experiment-6

1. Aim: a) To build knowledge base based on facts and rules
b) To implement logical expressions and codes using Prolog

2. Objectives: After performing the experiment, the students will be able to

- Download and install SWIProlog
- **Construct** knowledge base based on facts and rules
- Write logical expressions

3. Lab Objective Mapped: To understand, formulate and implement declarative programming paradigm through logic programming

4. Prerequisite: Nil

5. Requirements: The following are the requirements – • **Internet connection**

- **Laptop/desktop with Windows/Linux/MAC operating system**
- **SWI Prolog**

6. Pre-Experiment Theory:
Logic programming is a distinctive style of programming. It is based on collection of logical propositions and questions. There are only two components to any logic program: facts and rules.

- The program is a knowledge base of facts and a series of rules to be applied to knowledge base
- Programs are based on the techniques developed by logicians to form valid conclusions from available evidence (knowledge base)

Most widely used logic programming language is Prolog. The name stands for **Programming in Logic.**

FACTS

- Facts are statements about what is true about a problem, instead of instructions how to accomplish the solution.
- The Prolog system uses the facts to work out how to accomplish the solution by searching through the space of possible solutions.
  - It is defined by an identifier followed by an n-tuple of constants.
  - A relation identifier is referred to as a predicate
- When a tuple of values is in a relation we say the tuple satisfies the predicate.

Syntax of facts

- Names of relationship and objects must begin with a lower- case letter.

- Relationship is written *first* (typically the *predicate* of the sentence).
- *Objects* are written separated by commas and are enclosed by a pair of round brackets.
- The full stop character '.' must come at the end of a fact.

Example of Facts

valuable(gold). /*Gold is Valuable*/
owns(john, gold). /*John owns gold*/ father(john, mary). /*John is the father of Mary*/
gives (john, book,mary). /*John gives the book to Mary*/
likes(john, Y), likes(Y, john). /* John likes everybody and everybody likes John */

RULES

A **rule** is a predicate expression that uses logical implication (:-) to describe a relationship among facts. Thus a Prolog rule takes the form **left_hand_side :- right_hand_side .**

This sentence is interpreted as: *left_hand_side* ***if*** *right_hand_side*. The **left_hand_side** is restricted to a **single, positive, literal**, which means it must consist of a positive atomic expression. It cannot be negated and it cannot contain logical connectives.

- Rules specify under what conditions a tuple of values satisfies a predicate.
- The basic building block of a rule is called an *atom*
- Rules consists of Head of clause, body of clause and neck of clause

Examples of valid rules:

friends(X,Y) :- likes(X,Y),likes(Y,X). /* X and Y are friends if they like each other */ hates(X,Y) :- not(likes(X,Y)). /* X hates Y if X does not like Y. */

Examples of invalid rules:

left_of(X,Y) :- right_of(Y,X) /* Missing a period */ likes(X,Y),likes(Y,X) :- friends(X,Y). /* LHS is not a single literal */ not(likes(X,Y)) :- hates(X,Y). /* LHS cannot be negated */

QUERIES

The Prolog interpreter responds to **queries** about the facts and rules represented in its database. The database is assumed to represent what is true about a particular problem domain. In making a query you are asking Prolog whether it can prove that your query is true. Whenever you run the Prolog interpreter, it will **prompt** you with **?-**.
Suppose your knowledge base consists of the following facts

food(burger). food(sandwich).
food(pizza).
lunch(sandwich). dinner(pizza).

Then if you put a query,
?- food(pizza). /* Is pizza a food?
The answer to this query will be **True**

The standard logical operators in Prolog are as listed below in the Table 1.

Table 1: Prolog logical operators

Prolog Symbol	Read as	Logical Operations
,	AND	Conjunction
;	OR	Disjunction
not	NOT	Negation

Each of the logical operation is evaluated based on the truth table. The truth tables of the operations are given below-

NOT operation: If the input is true then the output will be false and vice versa

Table 2: Prolog NOT operation

Input	Output=not(input)
True	False
False	True

AND Operation: The output will be true if both the inputs are true

Table 3: Prolog AND operation

Input 1	Input 2	Output= Input 1, Input 2
False	False	False
False	True	False
True	False	False
True	True	True

OR Operation: The output will be true if any one of the inputs will be true

Table 4: Prolog OR operation

Input 1	Input 2	Output= Input 1; Input 2
False	False	False
False	True	True
True	False	True
True	True	True

NAND Operation- It's NOT of AND operation. The output will be false, if both the inputs will be true. The truth table is given below

Table 5: Prolog NAND operation

Input 1	Input 2	Output= not(Input 1, Input 2)
False	False	True
False	True	True
True	False	True
True	True	False

NOR Operation- It's NOT of OR operation. The output will be false, if any one of the inputs will be true. The truth table is given below

Table 6: Prolog NOR operation

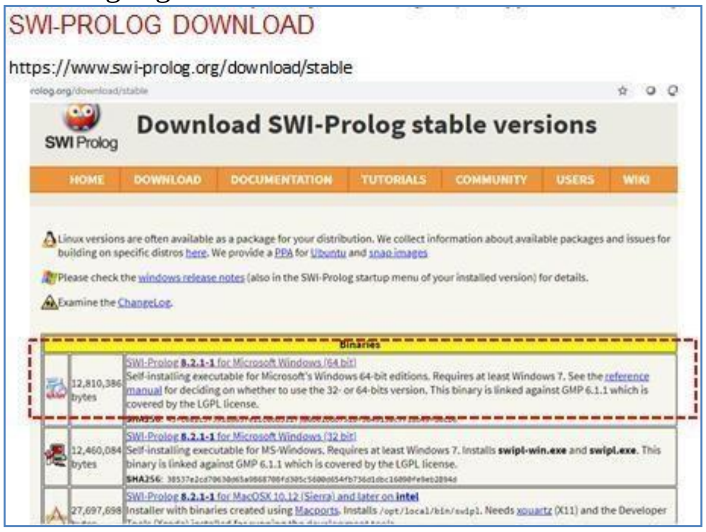
Input 1	Input 2	Output= Input 1; Input 2
False	False	True
False	True	False
True	False	False

True	True	False

7. Lab Laboratory Exercise:

A. Procedure

1. Visit https://www.swi-prolog.org/download/stable
2. Click on the highlighted box


SWI-PROLOG DOWNLOAD

https://www.swi-prolog.org/download/stable

Download SWI-Prolog stable versions

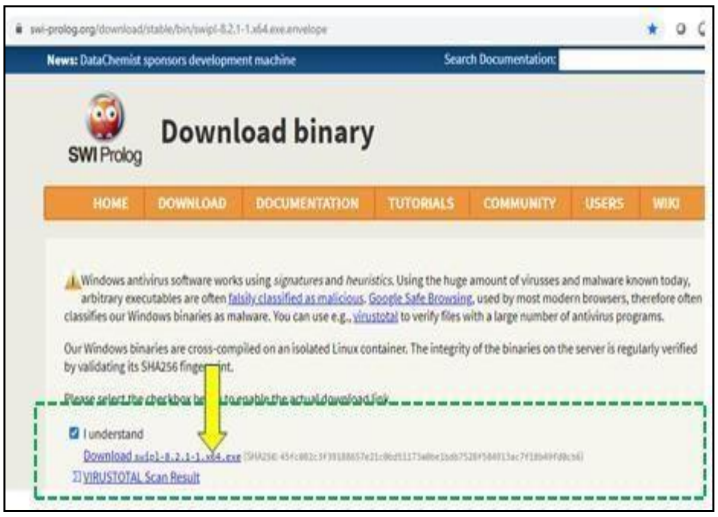
Linux versions are often available as a package for your distribution. We collect information about available packages and issues for building on specific distros here. We provide a PPA for Ubuntu and snap images

Please check the windows release notes (also in the SWI-Prolog startup menu of your installed version) for details.

Examine the ChangeLog.

Binaries		
	12,810,386 bytes	SWI-Prolog **8.2.1-1** for Microsoft Windows (64 bit) Self-installing executable for Microsoft's Windows 64-bit editions. Requires at least Windows 7. See the reference manual for deciding on whether to use the 32- or 64-bits version. This binary is linked against GMP 6.1.1 which is covered by the LGPL license.
	12,460,084 bytes	SWI-Prolog **8.2.1-1** for Microsoft Windows (32 bit) Self-installing executable for MS-Windows. Requires at least Windows 7. Installs **swipl-win.exe** and **swipl.exe**. This binary is linked against GMP 6.1.1 which is covered by the LGPL license. **SHA256**: 38537e2cd7063d65a9868708fd385c5600d654fb736d1dbc16090fe9eb2894d
	27,697,698	SWI-Prolog **8.2.1-1** for MacOSX 10.12 (Sierra) and later on **intel** Installer with binaries created using Macports. Installs `/opt/local/bin/swipl`. Needs xquartz (X11) and the Developer

3.Download the highlighted version


swi-prolog.org/download/stable/bin/swipl-8.2.1-1.x64.exe.envelope

News: DataChemist sponsors development machine

Download binary

Windows antivirus software works using *signatures* and *heuristics*. Using the huge amount of virusses and malware known today, arbitrary executables are often falsily classified as malicious. Google Safe Browsing, used by most modern browsers, therefore often classifies our Windows binaries as malware. You can use e.g., virustotal to verify files with a large number of antivirus programs.

Our Windows binaries are cross-compiled on an isolated Linux container. The integrity of the binaries on the server is regularly verified by validating its SHA256 fingerprint.

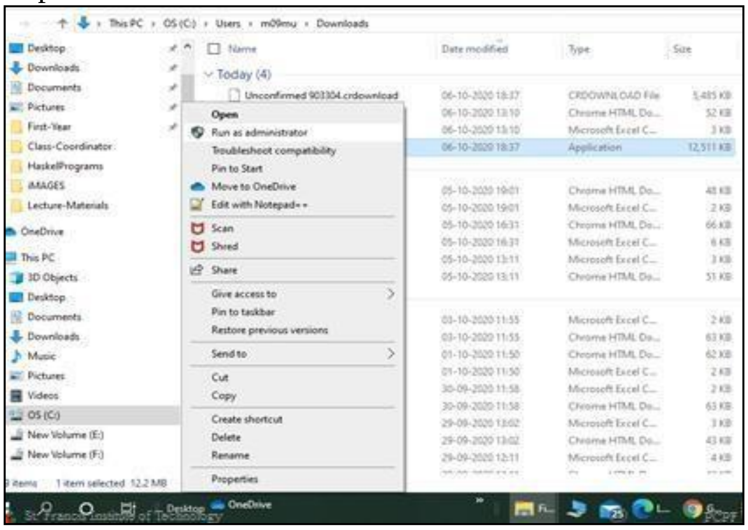
Please select the checkbox below to enable the actual download link.

[✓] I understand

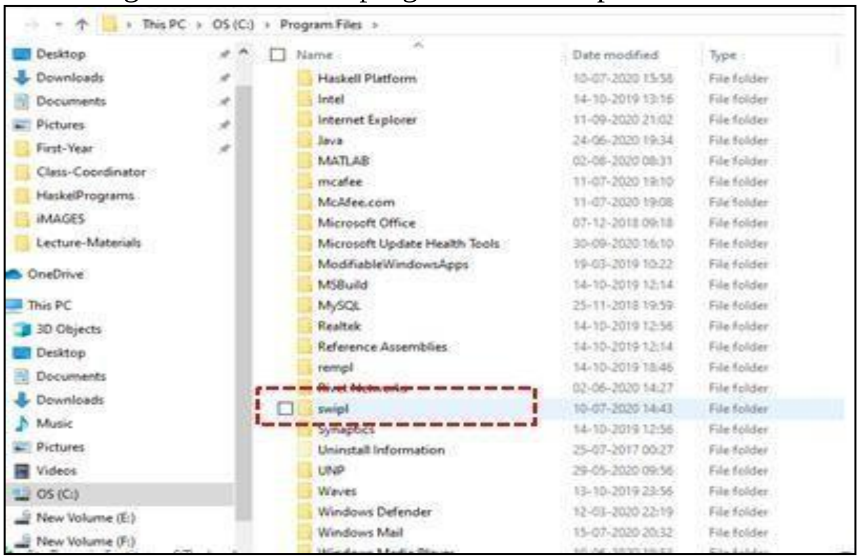
Download swipl-8.2.1-1.x64.exe

VIRUSTOTAL Scan Result

4. After successful download, install using ‘Run as administrator’. This will start the installation process.



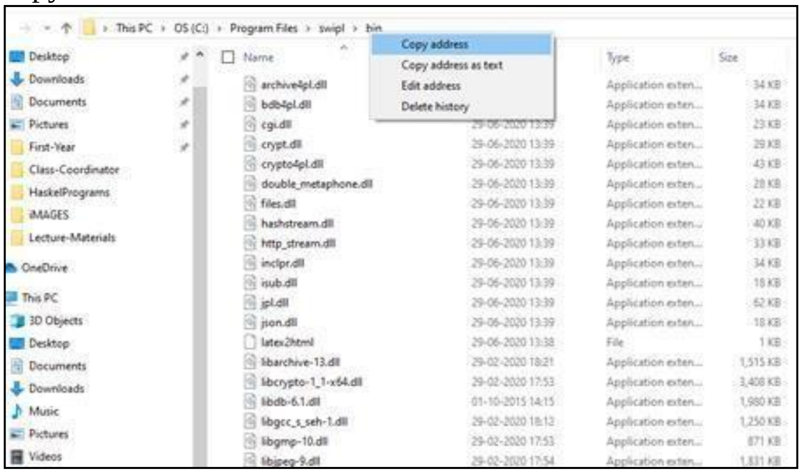
5. By default it will get installed in C:\program Files\Swipl folder



This PC › OS (C:) › Program Files ›

Name	Date modified	Type
Haskell Platform	10-07-2020 15:58	File folder
Intel	14-10-2019 13:16	File folder
Internet Explorer	11-09-2020 21:02	File folder
Java	24-06-2020 19:34	File folder
MATLAB	02-08-2020 08:31	File folder
mcafee	11-07-2020 19:10	File folder
McAfee.com	11-07-2020 19:08	File folder
Microsoft Office	07-12-2018 09:18	File folder
Microsoft Update Health Tools	30-09-2020 16:10	File folder
ModifiableWindowsApps	19-03-2019 10:22	File folder
MSBuild	14-10-2019 12:14	File folder
MySQL	25-11-2018 19:59	File folder
Realtek	14-10-2019 12:56	File folder
Reference Assemblies	14-10-2019 12:14	File folder
rempl	14-10-2019 18:46	File folder
[illegible]	02-06-2020 14:27	File folder
swipl	10-07-2020 14:43	File folder
Synaptics	14-10-2019 12:56	File folder
Uninstall Information	25-07-2017 00:27	File folder
UNP	29-05-2020 09:56	File folder
Waves	13-10-2019 23:56	File folder
Windows Defender	12-03-2020 22:19	File folder
Windows Mail	15-07-2020 20:32	File folder

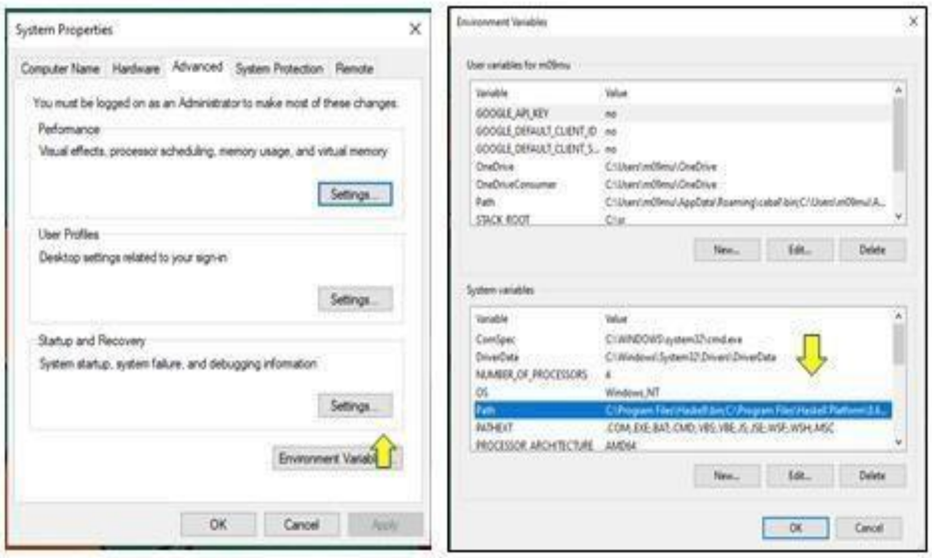
6. To set the path copy the entire address till bin



This PC › OS (C:) › Program Files › swipl › bin

Name	Date modified	Type	Size
archive4pl.dll	[illegible]	Application exten...	34 KB
bdb4pl.dll	[illegible]	Application exten...	34 KB
cgi.dll	29-06-2020 13:39	Application exten...	23 KB
crypt.dll	29-06-2020 13:39	Application exten...	29 KB
crypto4pl.dll	29-06-2020 13:39	Application exten...	43 KB
double_metaphone.dll	29-06-2020 13:39	Application exten...	28 KB
files.dll	29-06-2020 13:39	Application exten...	22 KB
hashstream.dll	29-06-2020 13:39	Application exten...	40 KB
http_stream.dll	29-06-2020 13:39	Application exten...	33 KB
inclpr.dll	29-06-2020 13:39	Application exten...	34 KB
isub.dll	29-06-2020 13:39	Application exten...	18 KB
jpl.dll	29-06-2020 13:39	Application exten...	62 KB
json.dll	29-06-2020 13:39	Application exten...	18 KB
latex2html	29-06-2020 13:38	File	1 KB
libarchive-13.dll	29-02-2020 18:21	Application exten...	1,515 KB
libcrypto-1_1-x64.dll	29-02-2020 17:53	Application exten...	3,408 KB
libdb-6.1.dll	01-10-2015 14:15	Application exten...	1,980 KB
libgcc_s_seh-1.dll	29-02-2020 18:12	Application exten...	1,250 KB
libgmp-10.dll	29-02-2020 17:53	Application exten...	871 KB
libjpeg-9.dll	29-02-2020 17:54	Application exten...	1,831 KB

7. To set the path , go to system variables (under advanced system settings) and select the 'path' variable



You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your sign-in

Startup and Recovery
System startup, system failure, and debugging information

User variables for m09mu

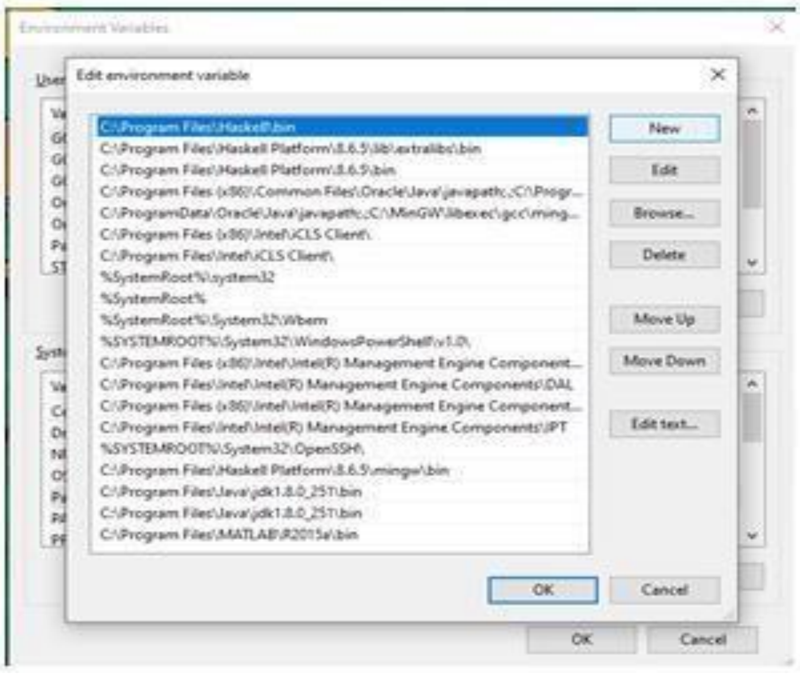
Variable	Value
GOOGLE_API_KEY	no
GOOGLE_DEFAULT_CLIENT_ID	no
GOOGLE_DEFAULT_CLIENT_S...	no
OneDrive	C:\Users\m09mu\OneDrive
OneDriveConsumer	C:\Users\m09mu\OneDrive
Path	C:\Users\m09mu\AppData\Roaming\cabal\bin;C:\Users\m09mu\A...
STACK_ROOT	C:\sr

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Program Files\Haskell\bin;C:\Program Files\Haskell Platform\8.6...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

Double click on path to get the new window…

8. Click on NEW and paste the entire copied path till the bin to set the path. Your path is now set successfully



```
C:\Program Files\Haskell\bin
C:\Program Files\Haskell Platform\8.6.5\lib\extralibs\bin
C:\Program Files\Haskell Platform\8.6.5\bin
C:\Program Files (x86)\Common Files\Oracle\Java\javapath;.;C:\Progr...
C:\ProgramData\Oracle\Java\javapath;.;C:\MinGW\libexec\gcc\ming...
C:\Program Files (x86)\Intel\iCLS Client\
C:\Program Files\Intel\iCLS Client\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
C:\Program Files (x86)\Intel\Intel(R) Management Engine Component...
C:\Program Files\Intel\Intel(R) Management Engine Components\DAL
C:\Program Files (x86)\Intel\Intel(R) Management Engine Component...
C:\Program Files\Intel\Intel(R) Management Engine Components\IPT
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Haskell Platform\8.6.5\mingw\bin
C:\Program Files\Java\jdk1.8.0_251\bin
C:\Program Files\Java\jdk1.8.0_251\bin
C:\Program Files\MATLAB\R2015a\bin
```

9. On the desktop, double click on the SWI-Prolog Icon to open the prompt

B. Program Code

1. WAP in prolog to create knowledge base using the following facts

woman(mia). woman(jody).
woman(yolanda).
playsAirGuitar(jody).

For the above knowledge base, fire the following queries and list the outputs ?-

woman(mia).

?- woman(jody).

?- woman(yolanda).

?- playsAirGuitar(jody).

?- playsAirGuitar(mia).

2. WAP in prolog to create knowledge base using the following facts and rules
happy(yolanda). listens2Music(mia).
listens2Music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2Music(mia).
playsAirGuitar(yolanda):-listens2Music(yolanda)
For the above knowledge base, fire the following queries- ?-happy(yolanda).

?-happy(mia).

?-listens2Music(mia).

?-listens2Music(yolanda).

?-happy(X).

?-playsAirGuitar(Y)

3. Write a prolog to create knowledge base using the following facts and rules

studies(charlie, csc135). studies(olivia, csc135).

studies(jack, csc131). studies(arthur, csc134).

teaches(kirke, csc135). teaches(collins,

csc131). teaches(collins, csc171).

teaches(juniper, csc134). professor(X, Y) :-

teaches(X, C), studies(Y, C).

For the above knowledge base, answer the following queries- ?-

studies(charlie, What).

?- professor(kirke, Students).

Analyse and explain the output

4 WAP in Prolog to implement the truth tables of the logical operations-NOT, AND, OR, NAND and NOR operations.

5 Implement the following logical operations • 6*6=:=36;10=8+3.

- a sqrt(36)+4=:=5*11-45.
- b 10=\=8+3.
- c 111=\=8+3.
- d 6+4==3+7.
- e 6<3;7 is 5+2. not(111=\=8+3).

- 111=\=8+3,11=\=3. 111=\=8+3;11=\=3.
- sqrt(36)+4=:=5*11-45;false. sqrt(36)+4=:=5*11-45,false.
- not(6<3;7 is 5+2).
- not(not(6<3;7 is 5+2)).
- not(not(6<3;7 is 5+2)),true.
- not(not(6<3;7 is 5+2)),false.
- not(not(6<3;7 is 5+2));false. not(not(6<3;7 is 5+2));true.

8. Post Experimental Exercise-

A. Questions:

1. Define the terms- (i) atom (ii) variable. Give examples
2. For the above two knowledge, make a table to enlist the English meaning of the facts and rules
3. Explain the significance of the operators **‘,’** and **‘:-‘** operators w.r.t. to Prolog

4 WAP to create the following knowledge base loves(vincent,mia). loves(marsellus,mia). loves(pumpkin,honey_bunny). loves(honey_bunny,pumpkin). jealous(X,Y):- loves(X,Z), loves(Y,Z). Also generate five different queries to get results

5. Write a program in Prolog to include the following facts and rules in the knowledge base dog(fido). dog(kitty). cat(sweety). cat(micky). animal(X):-dog(X). noanimal(Y):cat(Y).

On this knowledge base, implement the following queries and record your answers not(dog(fido)). not(dog(kitty)). not(cat(sweety)). not(cat(micky)). not(dog(fiddo)). not(cat(sweetty)). not(cat(sweety)),not(cat(sweetty)). not(cat(sweety));not(cat(miccy)). not(cat(sweety));not(cat(micky)).

6 Describe all the logical operators in prolog.
7 On the knowledge base of program 1, implement the NAND and NOR operation

B. Results/Observations/Program output:

Present the program input/output results if any and comment on the same.

C. Conclusion:

1. Write what was performed in the experiment

2. Write which tools you used to perform the experiment

3. Write what you inferred from the output obtained

D. References:

[1] Michael L Scott, “Programming Language Pragmatics”, Third edition, Elsevier publication
[2] Max Bramer, “ Logic Programming with Prolog”, Springer, 2005

St. Francis Institute of Technology
Borivli (W), Mumbai 400103
Department of Information Technology

Experiment-7

1. Aim: To implement list comprehensions using Prolog

2. Objectives: After performing the experiment, the students will be able to

- Understand Lists and construct List
- Perform List operations

3. Lab Objective Mapped: To understand, formulate and implement declarative programming paradigm through logic programming

4. Prerequisite: Knowledge of facts, rules, constants and variables

5. Requirements: The following are the requirements – • **Internet connection**

- **Laptop/desktop with Windows/Linux/MAC operating system**
- **SWI Prolog**

6. Pre-Experiment Theory:

Lists in Prolog refers to an ordered sequence of elements. It is also a collection of terms, which is useful for grouping items together, or for dealing with large volumes of related data.

Example
[red, white, black, yellow]
Lists are enclosed by square brackets, and items are separated by commas. The length of a list is the number of items it contains. The length of this list is 4.

List is made up of two parts: the first element, known as the **Head,** and everything else, called the **Tail.** Prolog uses a built-in operator, the **pipe (|)** in order to split the list as Head and Tail. If unification is applied with the above example then,

[Head|Tail] = [red, white, black, yellow]. Will result in following output- Head
= red
Tail = [white, black, yellow]

The other important points related to lists are-

- A single element in a list can be represented as [a]
- An empty list can be represented as []
- The elements of lists are separated by commas. Compound lists are also possible
- [first, second, third] = [A|B] where A = first and B= [second, third]. If the unification succeeds, then A is bound to the first item in the list, and B to the remaining list.
- [ ] is a special list, it is called the empty list because it contains nothing. Its length is 0

The following list comprehensions/ Operations can be performed in Prolog at prolog prompt

Sr. No	List Operation	Definition	Example
1	Membership Checking	To verify whether a given element is member of specified list or not	member(x, [x,y,z]).
2	Length Calculation	To find the length of a list.	length([a,b],M).
3	Append Items	To append one list into another (as an item).	append([a,b],[1,2,3],L).
4	Reverse	To reverse the elements of list	reverse([1,2,3],A).

7. Lab Laboratory Exercise:
A. Procedure

Steps to be implemented

1. Open SWI-Prolog
2. Go to ….File-> new
3. New prolog editor will open up.
4. Write your program (collection of facts, rules, clauses)
5. Save the file at the desired location as 'abc.pl' file
6. Follow the steps -> Save buffer, Make and Compile buffer
7. After successful compilation, at the prompt, first change to working directory using cd command Eg. cd('D:/PCPF/AY-2021-22/Course Lab/Prolog Codes').
8. Load the required file Eg. [abc]
9. To check the outputs, fire appropriate queries at the prompt

B. Program Code

1. Execute the following List commands at Prolog Prompt. Make proper observations of the outputs. Make a note of the commands that give error. Analyse the reason for the error

   member (x, [x,y,z]). member(p, [x,y,z]). member(my(x,y,z),[q,r,s,my(x,y,z),w]) . member(v,[]). length([a,b],M). length([1,2,3],M). length([1,2,3],a). length([1,2,3],X1). length([1,2,3],X-1). length([[a,c],[e,f],[h,i]],N). length([],P). length([a,b,c],3). reverse([1,2,3],A). reverse(B, [1,2,3]). reverse([[dog,cat],[1,2],[bird,mouse]],L). reverse([1,2,3,4],[4,3,6,8]). reverse([1,2,3,4],[4,3,2,1]).

append([],[1,2,3],L). append([a,b],[1,2,3],L).
append([a,b,23],[1,2,3],L).

2. Write a program in Prolog to concatenate two lists.
3. Write a program in Prolog to delete an item from a given list.
4. Write a program in Prolog to insert an item in a given list.

8. Post Experimental Exercise-

A. Questions:

1. Define the terms-> (i) Unification (ii) Resolution. Give suitable examples.
2. Write a program in Prolog to check if the given list is a palindrome list.
3. Write a program in Prolog to find all possible subsets of a given list.

B. Results/Observations/Program output:

Present the program input/output results if any and comment on the same.

C. Conclusion:

1. Write what was performed in the experiment

2. Write which tools you used to perform the experiment

3. Write what you inferred from the output obtained

D. References:

[1] Michael L Scott, “Programming Language Pragmatics”, Third edition, Elsevier publication

[2] Max Bramer, “ Logic Programming with Prolog”, Springer, 2005

[3] https://www.youtube.com/watch?v=iJhtgWAGUAQ [Lecture 14 Prolog Programming]

St. Francis Institute of Technology
Borivli (W), Mumbai 400103
Department of Information Technology

Experiment-8

1. **Aim: a)** To implement recursion functions in prolog
   b) To perform database manipulations using Prolog

2. **Objectives:** After performing the experiment, the students will be able to
   - Understand and implement recursion
   - Perform simple database manipulation operations

3. **Lab Objective Mapped:** To understand, formulate and implement declarative programming paradigm through logic programming

4. **Prerequisite:** Knowledge of facts, rules, constants and variables

5. **Requirements:** The following are the requirements – • **Internet connection**
   - **Laptop/desktop with Windows/Linux/MAC operating system**
   - **SWI Prolog**

6. **Pre-Experiment Theory:**

Recursive Programming

Most programming languages provide loops that allow a set of instructions to be executed repeatedly either a fixed number of times or until a given condition is met. Prolog has no looping facilities. The same effects can be obtained (that enable a sequence of events to be evaluated repeatedly) through backtracking, recursion, built in predicates or a combination of both. A recursive definition in Prolog always has at-least two parts- a first fact that acts like a stopping condition and a rule that calls itself simplified. At each level the first fact is checked. If the fact is true then the recursion ends. If not the recursion continues. A recursive rule must never call itself with same arguments. If that happens the program will never end. When writing such recursive programs in Prolog, a standard recursive pattern always has at least two parts:

Base (non-recursive) clause: Typically the base-case rule(s) will represent the smallest possible example(s) of the problem that you are trying to solve - a list with no members, or just one member. It non-recursively describes the base of the recursive process.

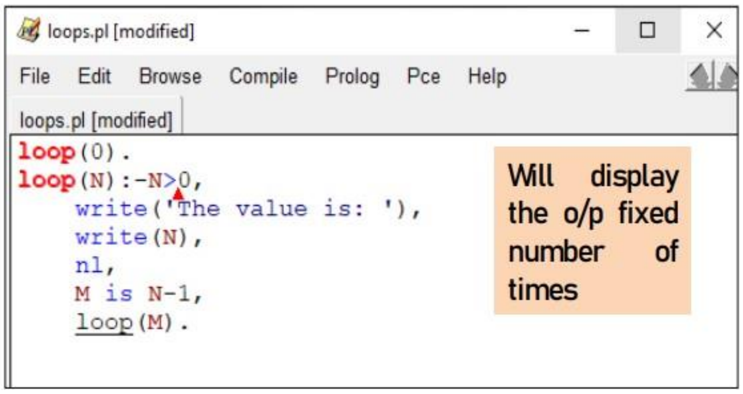
Recursive (continuing) clause: Contains any required logic including a call to itself, continuing recursion.

In prolog, a clause can call itself recursively.

Example: To display n natural numbers recursively in descending order.

The first line of the code defines the non-recursive (base) clause. For the base case the outputs are always pre-defined. In this case, the clause defines to stop for the value of N=0. The base case is preceded by a dot operator.

The loop definition starts from the second line, and continues to call it itself recursively, till the base clause is satisfied (true).


loops.pl [modified]
File Edit Browse Compile Prolog Pce Help
loops.pl [modified]

```
loop(0).
loop(N):-N>0,
    write('The value is: '),
    write(N),
    nl,
    M is N-1,
    loop(M).
```

Will display the o/p fixed number of times

Fig 1: Recursive program to display n natural numbers

Database Manipulation

A Prolog program can be viewed as a database where the specification of relations between data is partly explicit (facts) and partly implicit (rules). Sometimes we want to be able to manipulate this database during execution (e.g. update facts, deduce new rules, etc). Prolog has four database manipulation commands: *assert*, *retract*, *asserta*, and *assertz*. The function of the basic database manipulation commands are listed in table 1.

Table 1: Database manipulation commands

Sr. No	Prolog Command	Function	Example
1.	listing	Provides the associated facts of database	?- listing.
2.	assert	To add more facts to the listing	?-assert(son(tom,sue)).
3.	retract	To remove facts from the database when we no longer need them.	?- retract(happy(marcellus)).
4.	assertz.	To place asserted material at the *end* of the database.	assert(p(b)), assertz(p(c)), asserta(p(a)).
5.	asserta.	To place asserted material at the *beginning* of the database.	

7. Lab Laboratory Exercise:

A. Procedure

Steps to be implemented (For recursive)

1. Open SWI-Prolog
2. Go to ….File-> new
3. New prolog editor will open up.
4. Write your program (collection of facts, rules, clauses)

5. Save the file at the desired location as ‘abc.pl’ file
6. Follow the steps -> Save buffer, Make and Compile buffer
7. After successful compilation, at the prompt, first change to working directory using cd command Eg. cd('D:/PCPF/AY-2021-22/Course Lab/Prolog Codes').
8. Load the required file Eg. [abc]
9. To check the outputs, fire appropriate queries at the prompt
10. For database commands execute at the Prolog prompt.

B. Program Code

1. WAP in Prolog to to calculate factorial of a number using recursion
2. WAP in Prolog to find the sum of first n natural numbers using recursion
3. Create a knowledge base1 to include few facts and rules. On this knowledge base, using database manipulation command- (a) append facts (b) append rules (c) delete facts (d) delete rule (e) append facts at the beginning and end of knowledge base

8. Post Experimental Exercise-

A. Questions:

1. Write a program in Prolog to print fibonacci series of number N using recursion
2. Create a knowledge base consisting of the following facts happy(mia). happy(vincent). happy(marsellus). happy(butch).
   happy(vincent).
   To this knowledge base append the following facts happy(jia) at the beginning and happy(john) at the end. Also delete the fact happy(marsellus).
3. For given program how would prolog respond to the query if you keep entering ';' after each solution?
   Program: p(a,b). p(b,c).
   p(X, Y):-p(Y, X).
   Query :
   ?- p(X,Y).

B. Results/Observations/Program output:

Present the program input/output results if any and comment on the same.

C. Conclusion:

1. Write what was performed in the experiment
2. Write which tools you used to perform the experiment
3. Write what you inferred from the output obtained

D. References:

[1] Michael L Scott, “Programming Language Pragmatics”, Third edition, Elsevier publication
[2] Max Bramer, “ Logic Programming with Prolog”, Springer, 2005
[3] https://www.youtube.com/watch?v=iJhtgWAGUAQ [Lecture 14 Prolog Programming]

St. Francis Institute of Technology
Borivli (W), Mumbai 400103
Department of Information Technology

Experiment – 9

1. **Aim:** To implement the concept of thread management and synchronization using concurrent programming

2. **Objective:** After performing the experiment, the students will be able to implement

   - Understand the concept of process and thread
   - Create a thread
   - Perform simple thread operations

3. **Lab objective mapped:** To **understand** alternative paradigm through concurrent programming fundamentals and **design**, **develop** applications based on concurrent programming (PSO2) (PO2)

4. **Prerequisite:** Basics of Java programming- classes, objects, functions, data abstraction

5. **Requirements:** The following are the requirements – Java (JDK8) Compiler

6. **Pre-Experiment Theory:**

Concurrency generally refers to events or circumstances that are happening or existing at the same time. In programming terms, concurrent programming is a technique in which
- Two or more processes start
- Run in an interleaved fashion through switching and
- Complete in an overlapping time period by managing access to shared resources

Process
- Process means any program is in execution.
- Process control block contains information about processes for example Process priority, process id, process state, CPU, register, etc.
- A process can create other processes which are known as **Child Processes**.
- Process takes more time to terminate and it is isolated means it does not share memory with any other process.
- Process is called heavy-weight process **Thread**
- One or more **threads** run in the context of the **process**.
- A **thread** is the basic unit to which the operating system allocates processor time.
- A **thread** can execute any part of the **process** code, including parts currently being executed by another **thread**

 Thread is called light weight process

7. Laboratory Exercise
A. Steps to be implemented

 1. **Compilation using JDK 8 using Turbo C**
- Write the code in notepad and save as .java file.

- Run command prompt and set the path (Eg. Set path= ‘C:\Users\m09mu\Desktop\Javacodes’)
- Compile the code using the command ‘javac name_of_file.java’ Correct compile time errors (if any) and rerun the code

1

- After successful compilation, run the code using the command ‘java name_of_file’

 Using Online IDE for C/C++/Java

- Log on to www.onlinegdb.com,/ www.jdoodle.com
- Select the programming language for coding
- Create new project
- Save the project using CTRL+S
- Run the program using F9
- For debugging use F8, along with step-into function of onlinegdb

B. Program Code

1. Create a thread t1 to display message “Hello World”.
2. Create two threads t1 and t2 to display message “Welcome to SFIT” and “Welcome to IT”.
3. Create a thread t1 by implementing runnable interface to display “Hello world” message 4. WAP to display n-thread ids

8. Post Experimental Exercise-

A. Questions:

1. Define Process and highlight its important points
2. Define thread and highlight its important points
3. Explain the life cycle of a thread with proper diagram.

B. Results/Observations/Program output:

Present the program input/output results if any and comment on the same.

C. Conclusion:

1. Write what was performed in the experiment
2. Write which tools you used to perform the experiment
3. Write what you inferred from the output obtained

D. References:

[1] Michael L Scott, “Programming Language Pragmatics”, Third edition, Elsevier publication [2] Doug Lea, “Concurrent Programming in Java: Design Principles and Pattern

St. Francis Institute of Technology
Borivli (W), Mumbai 400103
Department of Information Technology

Experiment – 10

1. Aim: To implement the concept of run time program management through exception handling using Java

2. Objective: After performing the experiment, the students will be able to implement

- Create a thread and Perform simple thread operations
- Synchronize the threads
- Exception handling

3. Lab objective mapped: To **understand** alternative paradigm through concurrent programming fundamentals and **design**, **develop** applications based on concurrent programming (PSO2) (PO2)

4. Prerequisite: Basics of Java programming- classes, objects, functions, data abstraction

5. Requirements: The following are the requirements – Java (JDK8) Compiler

6. Pre-Experiment Theory:

Concurrency generally refers to events or circumstances that are happening or existing at the same time. In programming terms, concurrent programming is a technique in which

- Two or more processes start
  - Run in an interleaved fashion through switching and complete in an overlapping time period by managing access to shared resources process
- Process means any program is in execution.
- Process control block contains information about processes for example Process priority, process id, process state, CPU, register, etc.
- A process can create other processes which are known as **Child Processes**.
- Process takes more time to terminate and it is isolated means it does not share memory with any other process.
- Process is called heavy-weight process **Thread**
- One or more **threads** run in the context of the **process**.
- A **thread** is the basic unit to which the operating system allocates processor time.
- A **thread** can execute any part of the **process** code, including parts currently being executed by another **thread**
- **Thread is called light weight process Synchronization**
- Multithreaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.
- So it needs to be made sure by some synchronization method that only one thread can access the resource at a given point of time.
- Java provides a way of creating threads and synchronizing their task by using synchronized blocks.
- Synchronized blocks in Java are marked with the synchronized keyword.
- A synchronized block in Java is synchronized on some object.
  - All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time.

- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Exception Handling

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained. The core advantage of exception handling is **to maintain the normal flow of the application**.

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in try block.

7. Laboratory Exercise

A. Steps to be implemented

 1. **Compilation using JDK 8 using Turbo C**

- Write the code in notepad and save as .java file.
- Run command prompt and set the path (Eg. Set path= ‘C:\Users\m09mu\Desktop\Javacodes’)
- Compile the code using the command ‘javac name_of_file.java’
- Correct compile time errors (if any) and rerun the code
- After successful compilation, run the code using the command ‘java name_of_file’

 2. **Using Online IDE for C/C++/Java**

- Log on to www.onlinegdb.com,/ www.jdoodle.com
- Select the programming language for coding
- Create new project
- Save the project using CTRL+S
- Run the program using F9
- For debugging use F8, along with step-into function of onlinegdb

B. Program Code

1. WAP in Java to implement thread synchronization
2. WAP in Java to implement exception handling using try and catch blocks

8. Post Experimental Exercise-

A. Questions:

1. What are various methods to create thread.
2. Explain the concept of deadlock in java

B. Results/Observations/Program output:

Present the program input/output results if any and comment on the same.

C. Conclusion:

1. Write what was performed in the experiment
2. Write which tools you used to perform the experiment
3. Write what you inferred from the output obtained

D. References:

[1] Michael L Scott, “Programming Language Pragmatics”, Third edition, Elsevier publication [2] Doug Lea, “Concurrent Programming in Java: Design Principles and Pattern

St. Francis Institute of Technology
Borivli (W), Mumbai 400103
Department of Information Technology

Experiment – 11

1. **Aim:** Comparative Study of Programming Languages for Web Development.

2. **Objective:** The aim of this laboratory experiment is to compare and evaluate the performance and efficiency of different programming languages commonly used in web development. The experiment will focus on the development of a simple web application using three different programming languages: Python (with Django framework), JavaScript (with Node.js and Express.js), and HTML-CSS. The study will analyze various aspects, including development speed, code readability, performance, and ease of implementation.

3. **Experiment Setup**:

Participants: A group of students with prior programming knowledge, including familiarity with Python, JavaScript, and HTML.

4. **Equipment**:

- Computers with the necessary development environments and web browsers.
- Text editors or Integrated Development Environments (IDEs) for coding.
- Internet connectivity for accessing documentation and resources.

5. Tasks:

a. Pre-Experiment Setup:

Participants will be divided into three groups, ensuring each group has members with expertise in one of the languages.

b. Experiment Execution:

- Each group is assigned the task of building a simple web application with similar functionality, such as a to-do list or a blog site.
- The Python group will use Django framework, the JavaScript group will use Node.js and Express.js, and the Ruby group will use Ruby on Rails.
- Participants will have a fixed amount of time to complete the development of their respective web applications.

c. Metrics and Evaluation: Case study will be evaluated based on following factors

- Measure the development time to complete the assigned task.
- Evaluate the readability and maintainability of the code in each language.
- Conduct performance tests to compare the speed & responsiveness of web applications.

d. Data Analysis:

- Analyze the data collected in development process and performance tests.
- Identify strengths and weaknesses of each programming language in the context of web development.

6 Discussion and Conclusion:

Share and discuss the findings.

References:

1. A case study on refactoring in Haskell programs