CMSC 124 Machine Problem 1
Topic Coverage: Background and Preliminaries

1. Choose 4 programming languages (hereafter referred to as PL) that you are comfortable working with.

   Chosen Programming Languages:
   - Python
   - C
   - C++
   - Java

2. Create a simple program that will ask for a temperature in Fahrenheit and will output its equivalent value in Celsius. Provide screenshots of your source code for every language.

PYTHON

C: > Users > mattk > Documents > Codes > Python > temperature.py > ...

```python
fahrenheit = float(input("Enter temperature in Fahrenheit:"))
celcius = (fahrenheit - 32) * 5/9

print('Temperature in Celcius is: %.2f Celcius' %celcius)
```

C

C: > Users > mattk > Documents > Codes > C > temperature.c > main()

```c
#include <stdio.h>

int main(){
    float fahrenheit, celcius;

    printf("Enter temperature in Fahrenheit:");
    scanf("%f", &fahrenheit);

    celcius = (fahrenheit - 32) * 5/9;

    printf("Temperature in Celcius is: %.2f Celcius", celcius);
    return 0;
}
```

C++

C: > Users > mattk > Documents > Codes > C++ > temperature.cpp > ...

```cpp
#include<iostream>
using namespace std;

int main(){
    float fahrenheit, celcius;
    cout<<"Enter temperature in Fahrenheit:";
    cin>>fahrenheit;

    celcius = (fahrenheit - 32) * 5/9;
    cout<<"Temperature in Celcius is: " <<celcius;
    return 0;
}
```

Java

C: > Users > mattk > Documents > Codes > Java > temperature.java > ...

```java
import java.util.Scanner;

public class temperature{
    Run | Debug
    public static void main(String[] string){
        Scanner input = new Scanner(System.in);
        System.out.print(s: "Enter temperature in Fahrenheit:");
        double fahrenheit = input.nextDouble();
        double celcius = (fahrenheit - 32) * 5/9;

        System.out.println("Temperature in Celcius is: " + celcius + " Celcius");
    }
}
```

3. Write a small program in your chosen PL that can store student information, such as the following: Name, Age, GPA (float), Grade Level (“Freshman”, “Sophomore”, “Junior”, “Senior”)

PYTHON

```python
class Student:

    def __init__(self, name, age, gpa, level):
        self.name = name;
        self.age = age;
        self.gpa = gpa;
        self.level = Level;

    def addStudent(self):
        Name = input("Enter student name: ")
        Age = int(input("Enter student age: "))
        GPA = float(input("Enter student GPA: "))
        print("Choose Grade Level:\n1.Freshman\n2.Sophomore\n3.Junior\n4.Senior")
        ch1 = int(input("Enter choice: "))
        if(ch1==1):
            Level = "Freshman"
        elif(ch==2):
            Level = "Sophomore"
        elif(ch==3):
            Level = "Junior"
        elif(ch==4):
            Level = "Senior"

        x = Student(Name, Age, GPA, Level)
        ls.append(x)

    def displayStudents(self, x):
        print("Name: ", x.name)
        print("Age: ", x.age)
        print("GPA: ", x.gpa)
        print("Grade Level: ", x.level)
        print("\n")

ls = []
while True:
    y = Student('', 0, 0, '')
    print("\n1.Enter new student\n2.Display student details\n3.Exit\n")
    ch = int(input("\nEnter choice:"))
    if(ch == 1):
        y.addStudent()
    elif(ch == 2):
        print("\nSTUDENT LIST")
        for i in range(ls.__len__()):
                y.displayStudents(ls[i])
    elif(ch == 3):
        break;
```

C++

```cpp
#include <iostream>
#include <string.h>
using namespace std;

int i = 0;

struct Student{
    char Name[50];
    int Level;
    int Age;
    float GPA;
} s[50];

void addStudent(){
    cout << "Enter student name: ";
    cin >> s[i].Name;

    cout << "Enter student age: ";
    cin >> s[i].Age;

    cout << "Enter student GPA: ";
    cin >> s[i].GPA;

    cout << "Choose Grade Level:\n1.Freshman\n2.Sophomore\n3.Junior\n4.Senior\n";
    cin >> s[i].Level;

    i = i + 1;
}

int displayStudent(){
    cout << "\nDISPLAYING INFORMATION:\n";
    for(int j = 0; j<i; ++j){
        cout << "\nStudent Name: " << s[j].Name << endl;
        cout << "Age: "<< s[j].Age << endl;
        cout << "GPA: "<< s[j].GPA << endl;

        switch(s[j].Level){
            case 1:
                cout << "Level: Freshman" << endl;
                break;
            case 2:
                cout << "Level: Sophomore" << endl;
                break;
            case 3:
                cout << "Level: Junior" << endl;
                break;
            case 4:
                cout << "Level: Senior" << endl;
                break;
            default:
                break;
        }
    }
    return 0;
}

int main(){
    int ch1;
    int status = 1;

    while(status == 1){
        cout << "\n1.Enter new student\n2.Display student details\n3.Exit\n";
        cin >> ch1;

        switch(ch1){
            case 1:
                addStudent();
                break;
            case 2:
                displayStudent();
                break;
            case 3:
                status = 0;
                break;
            default:
                break;
        }
    }
    return 0;
}
```

C

```c
#include <stdio.h>
#include <stdlib.h>

int i = 0;

struct Student{
    char Name[50];
    int Level;
    int Age;
    float GPA;
} s[50];

void addStudent(){
    printf("Enter student name: ");
    scanf("%s", &s[i].Name);

    printf("Enter student age: ");
    scanf("%d", &s[i].Age);

    printf("Enter student GPA: ");
    scanf("%f", &s[i].GPA);

    printf("Choose Grade Level:\n1.Freshman\n2.Sophomore\n3.Junior\n4.Senior\n");
    scanf("%d", &s[i].Level);

    i = i + 1;
}

int displayStudent(){
    printf("\nDISPLAYING INFORMATION:\n");
    for(int j = 0; j<i; ++j){
        printf("\nStudent Name: %s\n", s[j].Name);
        printf("Age: %d\n", s[j].Age);
        printf("GPA: %f\n", s[j].GPA);

        switch(s[j].Level){
            case 1:
                printf("Level: Freshman\n");
                break;
            case 2:
                printf("Level: Sophomore\n");
                break;
            case 3:
                printf("Level: Junior\n");
                break;
            case 4:
                printf("Level: Senior\n");
                break;
            default:
                break;
        }
    }
    return 0;
}

int main(){
    int ch1;
    int status = 1;

    while(status == 1){
        printf("\n1.Enter new student\n2.Display student details\n3.Exit\n");
        scanf("%d", &ch1);

        switch(ch1){
            case 1:
                addStudent();
                break;
            case 2:
                displayStudent();
                break;
            case 3:
                status = 0;
                break;
            default:
                break;
        }
    }
    return 0;
}
```

JAVA

```java
import java.util.Scanner;

public class StudentInfo {


    public static class Student{
        public String Name;
        public int Age;
        public float GPA;
        public int level;
    }

    public static void addStudent(){
        Scanner scan1 = new Scanner(System.in);
        System.out.println("Enter student name: ");
        stud[i].Name = scan1.nextLine();

        Scanner scan2 = new Scanner(System.in);
        System.out.println("Enter student age: ");
        stud[i].Age = scan2.nextLine();

        Scanner scan3 = new Scanner(System.in);
        System.out.println("Enter student GPA: ");
        stud[i].GPA = scan3.nextLine();

        Scanner scan4 = new Scanner(System.in);
        System.out.println("Choose Grade Level:\n1.Freshman\n2.Sophomore\n3.Junior\n
4.Senior\n");
        stud[i].level = scan4.nextLine();

        i = i + 1;

    }

    public static void displayStudent(){
        System.out.println("\nDISPLAYING INFORMATION:\n");

        for(int j = 0; j<i; ++j){
            System.out.println("\nStudent Name: " + stud[j].Name + "\n");
            System.out.println("Age: " + stud[j].Age + "\n");
            System.out.println("GPA: " + stud[j].GPA + "\n");

            switch(stud[j].level){
                case 1:
                    System.out.println("Level: Freshman\n");
                    break;
                case 2:
                    System.out.println("Level: Sophomore\n");
                    break;
                case 3:
                    System.out.println("Level: Junior\n");
                    break;
                case 4:
                    System.out.println("Level: Senior\n");
                    break;
                default:
                    break;
            }
        }
    }


    public static void main(string[] args) {
        int temp;
        int status = 1;
        int i = 0;
        Student[] stud = new Student[50];

        while(status == 1){
            Scanner scan = new Scanner(System.in);
            System.out.println("\n1.Enter new student\n2.Display student details\n3.Exit\n
");
            temp = scan.nextLine();

            switch(temp){
                case 1:
                    addStudent();
                    break;
                case 2:
                    displayStudent();
                    break;
                case 3:
                    status = 0;
                    break;
                default:
                    break;
            }
        }
    }
}
```

4. Now, evaluate each PL according to the different language evaluation criteria discussed in class.
   Readability – Consider the following characteristics that contribute to the readability of a language: simplicity, orthogonality, control statements, data types and structures, and syntax designs.
   Writability – Consider the following characteristics that contribute to the writability of a language: simplicity, orthogonality and expressivity.
   Reliability – Consider the following characteristics that contributes to the reliability of a language: type checking, exception handling and aliasing.

Personal rating on the PL's language evaluation criteria:

PL	Readability	Writability	Reliability
Python	Excellent	Excellent	Great
Java	Great	Good	Excellent
C	Great	Great	Good
C++	Good	Great	Poor

In-depth reasoning on the programming languages evaluation:

PYTHON:

Readability – Python has a high readability in the sense that readability is at the heart of the design of the Programming Language. For example, Pythons' formatting and structure is usually tidy and orderly due to the lack of curly brackets and instead opting for whitespace indentation, while also often using English keywords which most users would be familiar with such as input() to indicate user input, print() to print the specified string or variable, and generally has easily understood statements and control flows like 'if', 'while', and commonly used assignment statements in "=".

Writability – Python can be considered to have an easy writing language which is suitable for beginner programmers. It also contains a lot of functions which makes coding easier and much more efficient like 'append()' to add inputs to a list, 'lower()' to convert all uppercase letters to lowercase, and even 'sorted()' to sort elements from an iterable object. Writing programs is also much easier considering that python has a straightforward syntax compared to PL's like C++ and Java. Python functions also have the ability to return more than one parameter which lessens the worry when it comes to managing errors.

Reliability – Python has great reliability with it often receiving updates. Python is also dynamically typed which makes it easier to write generic code and run the code after changing it when encountering type errors. In terms of exception handling, python can handle exceptions at runtime, and is also able to catch and handle exceptions through the use of a try-block and a try-except block. Python also allows the use of standardized libraries which helps in ensuring that the program runs correctly.

C:

Readability – C has great readability for a low-level language. For example, the syntax in C may not be the best for new programmers, however it compensates with its numerous library functions which can help in enhancing the readability of the PL. C's syntax is also pretty readable and its data structures are easy to understand. Unlike python, C uses curly braces which helps in structuring the program but also diminishes its readability if the source code becomes too long.

Writability – Similar to Python, C is also considered as an easy programming language. It has good syntax and also has numerous libraries which can be used to make writing code easier. C also has good expressivity where instead of doing "i = i + 1", we can simply do "i++".

Reliability – The language C is generally not known for its reliability. Mostly due to the fact that this PL has been around for a long time hence it would be inevitable that it would still have some form of design flaws. However, C also has good type checking since it is a statically typed language, meaning that the type of the variable is known at compile time. If the variable "number" is declared as "int", then it is guaranteed to be an "int" at runtime. Generally, this would help in discovering bugs and errors at an early stage. C has strict aliasing, however it does not support exception and error handling, hence diminishing its reliability as a programming language.

C++:

Readability – C++ is an extension of, hence it also has great readability but is also considered a harder programming language. However, due to its peculiar semantics, it may not be the best for new programmers. It has a complex syntax that supports versatility, however it makes it harder for programmers to read the code. For example, in printing out a statement we use 'cout' and the operator "<<" which is generally uncommon in other programming languages. The same can be said when accepting an input from the user through 'cin' and the operator ">>". This makes it so that programmers who are unfamiliar with C++ would have a hard time understanding the program. Even users who are familiar with C would need to adapt to understanding this new syntax. Generally we can say that C++ is readable not because of its simplicity and orthogonality but its familiarity to the programming language C.

Writability – In terms of writing code in C++, it can be rather easy to write a bad code and rather hard to write a good one. The programming language is pretty flexible when it comes to writing code due to its complexity making it a very powerful language. For example, the pointers in C++ are flexible and powerful however it can also result in errors and potential issues.

Reliability – Due to C++'s complexity, it is not as reliable as other programming languages. Another point that supports this unreliability is the fact that it allows loose type checking and operator overloading. It generally allows possible mistakes to

happen when compiling and writing the code. However, unlike C, it supports exception handling.

JAVA:

Readability – Similar to Python, Java is considered as a beginner friendly programming language. It is simple and much more readable compared to C++. It's syntax is intuitive and user friendly while also leaving little room for deviation making it readable for beginners. Java also does not support pointers but instead have references.

Writability – Java has good writability however it isn't as powerful as other programming languages like C++. It is writable simply because of its simplicity and familiarity, but it isn't as flexible and expressivity as other PL's.

Reliability – Java is a strictly and statically typed language hence making it a lot more reliable than C++. Variable types must be explicitly declared and java blocks must be defined with curly brackets. This strict structure makes it reliable since if an error occurs, the program won't run at all. Java is also more reliable than python since the syntax is tested prior to running instead of testing it at runtime. Java also has strong type checking and supports exception handling unlike the PL C.

5. Determine the paradigm(s) of your chosen PL. Remember that there is no particular paradigm suited to a specific PL, so list down all applicable paradigm.

PYTHON	C	C++	JAVA
Imperative	Imperative	Imperative	Imperative
Procedural	Procedural	Procedural	Procedural
Functional	Structured	Functional	Functional
Object-Oriented		Object-Oriented	Object-Oriented
Structured		Generic	Generic
Modular		Modular	Concurrent
			Reflective

6. Describe the method of implementation for every PL you have chosen. Explain in detail what happens to your source code upon compilation, down to execution.

Answer:

Python – In terms of implementation, python can be said to be a mix of interpretation and compilation when it comes to its method of implementation but it is primarily known as an interpreted language. Firstly, when we execute the source code, it first gets compiled to give us what is called the byte code which is then interpreted in a machine language by a special virtual machine. The byte code is a set of instructions for the virtual machine and is not the binary machine code which can be run by the machine. An example of a virtual machine for python would be the PVM (Python Virtual Machine). The PVM accepts the byte code and is then interpreted by the machine into machine code which is then executed by the virtual machine during runtime.

C – In terms of implementation, the programming language C is a compiled language. Firstly, the preprocessor reads the source code in search for any preprocessor directives such as include-files. The compiler then takes the preprocessed source code and converts or assembles it into a machine language module or also called an object file. A specialized program called the linker then combines the object file with other compiled objects like library files to create a single executable file. This is different to python in the sense that the conversion is performed before the executable file is created, instead of during run-time similar to python.

C++ - Similar to C, the programming language C++ is also a compiled language. Similar to C, the preprocessor first reads the source code for any preprocessor directives. The compiler then takes the output of the preprocessor and translates it line by line into the appropriate machine language and then creates an object file. However, this object file isn't a working executable for the C++ source code. Hence, the linker then combines the object file with other necessary files such as library extensions to create the executable file for the program.

Java – Similar to python, Java can also be said to be an interpreted or a compiled language depending on the execution environment. Unlike C and C++, Java programs are not compiled into executable files but into bytecode. This bytecode is then saved to the disk with the extension '.class'. The java's virtual machine, specifically the JVM or Java Virtual Machine then accepts the bytecode and converts it into machine code which then is executed at runtime by the virtual machine.

References:

Mustapha. (2019). *Can Python be compiled? is it compiled or interpreted?.* Retrieved from https://www.astateofdata.com/python-programming/can-python-be-compiled/.

Bagheri, R. (2020). *Understanding Python Bytecode.* Retrieved from https://towardsdatascience.com/understanding-python-bytecode-e7edaae8734d

The Programming Process. (n.d.). Retrieved from http://www2.hawaii.edu/~takebaya/ics111/process_of_programming/process_of_programming.html

Language Evaluation Criteria. (2018). Retrieved from https://progr-harrykar.blogspot.com/2018/11/language-evaluation-criteria.html

Gupta, S. (2020). *Easiest and Hardest Programming Languages to Learn*. Retrieved from https://www.springboard.com/blog/software-engineering/top-programming-languages

Reminders

Plagiarism will not be tolerated. Kindly read the guidelines on Source Code Plagiarism as defined here in details: http://web.science.mq.edu.au/~mtaylor/ponline/index.php?id=source-codeplagiarism