Unit 2

Computing Practice and Programming

Part 1: Use of Programming Languages to Solve Problems

Programming Languages and Paradigms

Low-Level Programming Language: A programming language that is closer to processor instructions than human language and provides little or no abstraction. This generally refers to machine code or assembly language, which are hard to read and write but can be directly processed by a computer.

Example:

Each of the following are actual examples of code fragments that print “Hello world” to the screen.

Machine Code:	Assembly Code:
ba 0c 01	mov dx, 010ch
b4 09	mov ah, 09
cd 21	int 21h
b8 00 4c	mov ax, 4c00h
cd 21	int 21h
48 65 6c 6c 6f 2c 20 57 6f 72 6c 64 21 0d 0a 24	db 'Hello, World!', '$'

Machine code consists of nothing but hexadecimal numbers that represent instructions that can be understood directly by the processor. Assembly code makes these instructions more human-readable, but is still very cryptic and incredibly hard to understand.

High-Level Programming Language: A programming language that is closer to human language than machine language making it easier to read and write but cannot be run without being

translated into machine language. High-level languages are necessary to make programming more accessible and efficient.

Example:

Each of the following are actual examples of code fragments that print “Hello world” to the screen.

Java:

```
System.out.print(“Hello world!”);
```

C++:

```
std::cout << “Hello World!”;
```

Python:

```
print(“Hello World!”)
```

Written Languages are programming languages that consist of written lines of code.

Graphical Languages are programming languages that consist of graphical and symbolic segments of code that often resemble a flowchart.

Example:

Written language (Java)

```
public static void main(String [] args) {
        System.out.println(“Hello World!”);
}
```

Graphical language (scratch)

Both of these snippets of code print the message, “Hello World!”

Programmers generally write programs in a high-level language because it is much easier to understand than a low-level language. Since computers only understand machine code, it is necessary to translate the human-readable code into machine code that can be executed by the computer. Two main ways of doing this have emerged and become dominant in computers: Compilers and Interpreters. We will look at the differences between the two.

Compiler

A compiler is its own standalone program. Compilers scan a source code file and compile all the instructions given in the source code directly into machine code. This machine code can then be directly executed by the operating system. The compiler also checks for errors in the source code, and can help with debugging a program.

Advantages of compiled languages:

- More efficient because source is translated directly into machine code.
- Once compiled, the program package contains all of the necessary components to run.

Disadvantages of compiled languages:

- Different processors use different machine languages. This requires us to write a new compiler for every possible processor architecture.
- It is a greater process to make changes to the code as it must be recompiled after each change.

Example Languages that use compilers:

C and C++

Interpreter

In an interpreted language, the source code is translated into bytecode, a syntax tree, a tokenized

representation of the source program, or some other intermediate form that cannot be directly run by the operating system. Another program, called the *interpreter*, then examines it and performs whatever actions are called for. The interpreter, in effect, is a middle-man that goes between the operating system and the intermediate code.

Advantages of interpreted languages:

- The code is easy to read and edit at any point during the process.
- Programs are more flexible, as the code can be run on different machines, using their own interpreter

Disadvantages of interpreted languages:

- The program will run slower because it must be processed through the interpreter as an additional step.

Example Languages that use interpreters:

Python (We will work with Python in Unit 5, Activity 2) and Ruby

Programming Paradigms

Procedural languages are languages that specify a large amount of pre-made functions (also called procedures) that allow a programmer to write a program. Programmers do not need to worry about objects in procedural languages. These languages are often more simple than object-oriented programming languages.

Example:

The programming language, *C*, is a very popular procedural language.

```
1    #include <stdio.h>
2    int main() {
3        printf("Hello world!");
4        return 0;
5    }
```

On the first line, we are telling the computer to include all the functions in the "stdio" library. We need to do this because we need to use the printf() function.

On the second line, we have the main function header. This function is run whenever the program is ran. The "int" means that the program will return an integer to the operating system to let it know if something went wrong. The brace lets the computer know that the function has began.

On the third line, we call the printf function with the arguments "Hello world!". This prints "Hello world!" to the screen.

On the fourth line, we return the integer "0" to the operating system. This is done by convention to let the OS know everything went OK.

On the last line, we have a closing brace to let the computer know the function is over.

Object-oriented languages are languages that focus heavily around classes and objects. Classes are like blueprints for objects that can have their own data and functions. Whenever the programmer wants to make use of a class, he must make an instance of that class, called an object. These types of languages are often more abstract and harder to grasp.

Example:

The programming language, *Java*, is a very popular Object-oriented language.

```
1    public class Hello {
2
3        public static void main(String [] args) {
4            System.out.println("Hello world!");
5        }
6
7    }
```

On line one, we define the Hello class.

On line three, we define the main method. Don't worry about the "public static void" and the "String [] args". Just know that the main method is, once again, ran whenever the program is run.

On line 4, we call the println function, which is a function of the out class, which is a member of the System class. The result is that "Hello world" is printed to the screen.