Unification of Programming Languages

Hafiz Razi Ahmad¹, Muhammad Idrees², Ashfaq Ahmad³, Muhammad Arif Butt², Saadia Shahzad⁴, Muhammad Shahzad⁵

¹PUCIT, University of Lahore, Lahore, Pakistan
²Department of Data Science, University of the Punjab, Lahore, Pakistan
³Department of Computer Science, Jazan University, Saudi Arabia
⁴Department of Computer Science, University of the Punjab, Lahore, Pakistan
⁵Department of Technology Education, IER, University of the Punjab, Lahore, Pakistan

Abstract – Practically every programming language is equipped with primitive data types, iteration and selection constructs, procedures, functions, class and structure definition, exceptions, and so forth to facilitate program development. These are similar in functionality but may differ a little in syntax. Similarly, libraries and packages for functionalities like collections, input/output, and so forth are available with different names to perform similar functions. A programmer has a hard time memorizing the syntax and features of each programming language. The research aims to choose a single unified high-level programming language, a Unified Programming Language (UPL), whose syntax is like of the most commonly existing programming languages. The benefits of learning and adopting a single programming language for programmers can be comforting, but there are some unintended consequences that need to be addressed. This research has gone through several brainstorming sessions and structured walkthroughs within an accessible community to identify and enlist the issues. The most common concern is the usage of a massive amount of existing code/libraries, which results in uncountable man-hours and worthwhile investment spent to build it. The subsequent concerns are the availability of programming tools like IDEs and various commonly required libraries and frameworks related to accessing range of databases, files, network...
streams, and building GUIs and reports. UPL, when released, will be equipped with all necessary libraries, and will provide a set of tools to hook existing libraries frameworks of existing commonly used programming languages like C/C++, JAVA, PHP, PYTHON, .NET languages and JAVASCRIPT, and more. This may be achieved through employing reflection API and documentation, whichever is appropriate. The UPL requires a compiler to generate binary executable code for various platforms.

**Keywords:** Programming Languages, Compiler, Translator, Interpreter, Programming techniques, language design and implementation

---

### 1 INTRODUCTION

At the time of the creation of the computer in the 1940s, programmers were required to write programs in binary form [1] and troubleshooting was difficult due to its complexity and the need-to-know underline hardware. To encourage the client to recall, PC researchers suggested Assembly Language (low-level computing construct), which is closest to machine language. The structure implies condensing because it is composed of Mnemonic. Various families of processors, including Intel, AMD, IBM, NVIDIA, etc. [2]–[5], have their low-level computing constructs and change in assembling framework. At that point, abnormal state dialects were presented, called third-era dialects, the high-level programming languages, which are convenient and machine-independent. Mediators and compilers are much more intensive, so they can generate local machine code or transfer code to a variety of low-level computer constructs. These dialects have different ideal models, such as Java, which is a completely typed safe and efficiently manages objects and memory [6]. Several ideal programming models were incorporated into the BETA [7], [8] pattern by Ole Lehrmann Madsen. As indicated by him, "it was important to bring together various existing elements." In this research, we are extending our proposed **Unified Programming Language** (UPL) [9], whose syntax is similar to the most commonly existing programming languages. Programmers do not have to learn and adopt more than one programming language, which simplifies their lives. It is necessary to identify and address some of the consequences of this
unification. In this research, we have devised a UPL that additionally listed these consequences and addressed them to make our proposed UPL acceptable by the programmers’ community and by the software industry and researchers doing research in the programming domain.

In addition to JAVA, .NET, C++, Ruby, JAVASCRIPT, PHP, C and others, there are several normal state dialects out there that differ based on various elements and standards. They are, however, presented by the same developers. A striking similarity exists between Java and C#. Changing from Java to C# requires less than 5% change [10]. Java has the advantage of being machine-independent, since it uses bytecode to make it machine-independent. A compiler will translate bytecode to the local dialect of the machine at runtime as per underline engineering. For this reason, it is said that it compiles once and runs anywhere. Additionally, Common Language Runtime (CLR) is included in the .Net Framework. With the CLR, the trying cause is to enable the execution of programs written in a variety of dialects and to promote interoperability between them [11]. As a result, source code written in C#, VB.NET, and .NET dialects is compiled to bytecode, a regular dialect. Bytecodes are converted to local code by the CLR for execution. .NET libraries are reusable in the sense that a segment created in one dialect can likewise be used in another dialect. Booch et al. (1999) designed the Unified Modeling Language (UML) to provide graphical documentation for product frameworks [12]. The effort was put in by Booch, Rumbaugh, and Jacobson. As a result, OMT, Booch, and CLS cooperate and attempt to tie their ideas together. UML was the name of their joint work. In 1997, creators working with designers gave the first adaptation of UML in response to OMG's request to join together the question-centered demonstrating process. UML instruments can produce code using various syntaxes or make changes to the engineering structure of what is currently being developed.

With the Unified Syntax, we aim to give a syntax that has grammar at its core, but with diverse elements integrated in one syntax, so software engineers and designers do not have to learn ideas, elements, and functionalities of several different languages or reverse engineer existing programs.
Also, the evolution of the programming languages as discussed earlier supports another level above the machine, assembly, and high-level paradigms to unify them.

2 LITERATURE REVIEW

In this section, we are presenting a comprehensive review on historical and existing work on unification of programming languages. Development of Unified Modeling Language (UML) that unified all the major object-oriented modeling design and methods into one comprehensive design to model object-oriented design for programming and software engineering. UML is one of a great work which leads us to start this research. Our work also inspired from the JRE/JVM 4F\(^1\) [13] tools of Java programming language, which at a lower level, provides unification in the form of byte-code. Byte code is common for all platforms and it is executed using JRE/JVM. Success of Java, JRE/JVM, byte code influenced Microsoft to develop a similar architecture composed of a variety of programming languages like C#, VB.NET, etc. and tools like CIL/CLR/.NET\(^2\) Framework. BETA, an effort to unify programming languages [8] was developed to integrate various programming construct like while, if, etc. in a generalized pattern. According to Ole Lehrmann Madsen [8], there should be a future language that should integrate the best the available concepts / features so that programmer should not think in context of multi paradigms when using a given dialect. In BETA [7], classes, procedures, functions, types, etc. were unified into an abstraction mechanism to unify the paradigms, constructs, etc. Patterns form the basis of these paradigms. The term class is used to describe a pattern. For instance, a procedure pattern is referred to as a procedure. The cross-platform technology is increasingly used in mobile technology because it boosts developer productivity and saves time by enabling the development for multiple platforms such as Android, IOS, Windows Phone etc. HTML5 can be developed using multiple tools and then translated to various

\(^1\) JRE/JVM is Java runtime environment and Java virtual machine respectively

\(^2\) CIL/CLR/.NET are Common Intermediate language, Common language runtime and .NET Microsoft framework
platforms using translation tools. Among the tools available for app
development are Sencha, PhoneGap, Appcelerator Titanium, Qorna, QT,
Xamarin, Aplha Anywhere, and 5App. A few of these tools are open-source.
Cocos2d is one such tool for game development. Several mobile platforms have
been supported by Unity 3D and Corona.

2.1 Characteristics of High-Level Programming Languages

Here, we are discussing the basic construct of languages like primitive types,
literals, iterations, selections, loop controls, exception handling, and I/O
libraries.

2.1.1. Primitive Literals for basic data types

Any primitive data type (e.g., an Integer, a floating-point number, any
Booleans, or a character [14] is represented in a source as a primitive literal.
These are predefined by the languages and named by a keyword to assign
primitive literals.

Integer Literals

There are four ways to represent integer values in a high-level programming
language. Binary with base 2, Octal with base 8, and base 10 are represented as
Decimal, whereas hexadecimal is represented with base 16.

Binary Literal

Binary literal represented by prefix with 0b [15]. For example, 0b110011. Most
high-level programming languages do not support binary representation.

Octal Literal

To represent an integer in octal form, 0 is placed in front of a number. Only 0
to 7 digits are allowed to use for octal from. For example, 0124 which is
equivalent in decimal is _. But notation varies in different languages. Like in
JAVASCRIPT to show octal value, prefix 0o is used. 0o124 is the form of octal
in Java Script.
Decimal Literal
Everyone knows about decimal numbers since the first class. No prefix or suffix is used to represent it. But for Long numbers suffix L in lower or upper case is used.

Hexadecimal Literal
Hexadecimal short for hex number is represented using 16 distinct symbols. There is no single symbol in digits to present 10 to 15. Alphabets A to F are used in small or capital form. Hex symbols are 0 1 2 3 4 5 6 7 8 9 a b c d e f. In Java, it is represented in the form 0xDeadCafe. Although JAVA is case sensitive but in some cases like here acceptable to show in lower or upper or with the mixer of upper and lower.

Floating-Point Literals
Floating-point numbers have their representation either as exponential notation or as decimal fractions. By default, such numbers are defined as type double. If we want floating-point literals to assign to a floating-point variable then need to add suffix F in small or capital form with the literal. Some languages e.g. FORTRAN [16], and PYTHON [17] also have a complex number type consisting of two floating-point numbers: a real part and an imaginary part.

Boolean Literals
Boolean literals are the source code to represent the Boolean values. It is a logical type that is converted at the end into numeric form. It takes true/false or values 0/1 according to language support. Some languages support values instead of true or false.

Character Literals
A character is any symbol or an alphabet letter, a number, punctuation mark, or even a space, which can be typed on a computer. It is represented in single quotes. Different languages use different character encoding schemes to represent the character literals e.g., ASCII, UNICODE. Unicode is represented by \u prefix with any alphabet (‘\u0001’).
String Literals
Although String is a non-primitive type it can be represented as a literal form. It can be typed directly to a variable or in object form. Other than strings that can be typed directly as literal is an array.

2.1.2 Variables
In computer science, a variable associated with the storage location it is a symbolic name that is used for a given quantity instead of the actual value. The compiler or interpreter converts the symbolic name with the location address. Although data of locations are changed during the execution of the program address location remains constant whole the application. Variable can be all types of data, including Boolean, Integer, String, or any Object of class.

2.1.3 Sequence of Instructions
A computer program is written by a programmer to perform a specific task. Such a program comprises a sequence of instructions written using a programming language. Instruction in a high-level Programming Language is called a Statement. Some languages used a semicolon to demarcate between two separate statements [18]. Similarly, some languages use a new line for each statement.

2.1.4 Exceptions
The term special case signifies a "remarkable condition" [6] that is the issue that happens during the execution of a program. At the point when the special case happened then the stream of programs exasperates and the program ends strangely. At the point when an extraordinary occasion happens, an exemption is said to be "tossed." The piece that is dependable to accomplish something is called a "special case handler" and it gets tossed into a special case. It can happen if an attempt to open a record that does not exist in the catalog or because of invalid information, association timeout amid correspondence with the database. Some of these can happen by client blunder, others by developer botch, or because physical assets fizzled. In light of this, it is classified into
three sections: "Checked special case" is likewise called accumulate time exemption that modified needs to deal with it in the program. There is a pseudo code of record perusing from the system. If there should be an occurrence of neglecting to get a document from the system then the catch part will be executed.

```
1. try {
2.   getTheFileFromOverNetwork
3.   readFromTheFileAndPopulateTable
4. }
5. catch (CantGetFileFromNetwork) {
6.   displayNetworkErrorMessage
7. }
```

Figure 1 structure of try-catch block

“Uncheck exception” it can occur at run time programming bug or logical error like class cast exception, divide by zero and third one “Error” which occur very rare by physical resource like stack overflow. It is normally ignored during the compile time.

2.1.5 Input / Output

Input/output is the communication means to read information from the keyboard, storage device, and network. Every language provides APIs to read/write the data from different streams. For example, In Java to write output on different streams like a console, a network socket, or a file, a common library (PrintWriter) is used.

2.1.6 Decision Statements

Decision-making structures are used to evaluate the conditional statements if the conditioned is determined to be true. Different decision-making statements are provided by languages like if statement, if…else statement, nested statements, and switch statements. In most languages their syntax is common. For example, if…
else statements are defined by JAVA, C++, and C #, OBJECTIVE-C, JAVASCRIPT, and PHP.

1. if ( 5 < 20 ){
2.   System.out.print ("This is if statement");
3. } else {
4.   System.out.print ("This is else statement");
5. }

Figure 2 – Decision statement

2.1.7 Loop Control
Sometimes we need to execute lines of codes repetitively in a different situation. Block of code repeats till at least one of the conditions is true or for a specified number of iterations. Most languages provided three flavors of loops which are while, do…while, and for. In Ruby for do…while loop begin keyword is used instead of do keyword. Similarly, in PYTHON, the while loop is defined as in figure 3.

1. count = 0
2. 
3. while count < 5:
4.    print count, " is less than 5"
5.    count = count + 1

Figure 3 – while loop in PYTHON

2.1.8 Method / Function
A function is a block of statements that is reusable code to act on a single task. It gives modularity to the code. It takes some arguments and returns the result after completing the execution of instructions. In Java, a function is defined in figure 4.
A method in object-oriented programming is a procedure associated with a class. It defines the behavior of the object. Conceptually Methods and Functions are distinct. The term ‘Method’ is used in the context of Object-Oriented. We can say the method is associated with the object that can act. How do they differ from each other? Methods are associated with the class, we can’t use them without creating the object of the class. On the other hand, Functions are independent of the Class’s object which means they can be called without creating the object.

### 2.1.9 Class / Interface

Class is a blueprint that represents the structure of the object which means how it will behave. It contains variables and methods. Engineers and factory workers use blueprints to build anything virtually. These blueprints may represent buildings, cars, or bridges. Patterns are used by the tailors for making our clothes. Most languages are object-based which shows the object relation like the real world. In figure 5, a sample program of the class in the JAVA language. The interface is like a class but it has abstract and public methods by default. The concrete class has to provide its implementation. Attributes defined in an interface are static, final, and public. Interfaces are used for polymorphism which means reference of the parent can refer to multiple child objects.
public class HelloWorld
{
  private String str="Hello World!";
  public static foo(){
      System.out.println(str);
  }
}

Figure 5 Class in JAVA language

2.1.10 Comments
To increase the readability of the program, comments/description is added before the statement, function, and class. These comments are ignored in the compilation because they just help a programmer to clarify the program. Comments can consist of single or multiline. Most of the languages use forward slashes to add the single line comment or use /* comments are here*/ structure.

public class HelloWorld
{
  /* multiline 
   comment in java 
  */
  private String str="Hello World!";

  public static foo(){
      // single line comment in java 
      System.out.println(str);
  }
}

Figure 6 - Java code with comments

2.1.11 Collections
Collections are the framework that provides the architecture to manipulate the group of objects. High-Level languages have different structures for searching, insertion, and deletion the data. Java provided multiple interfaces like Queue, Linked List, and Hash Map.
2.1.12 Packages

The package is a namespace to organize the group of classes. Conceptually, we can think of it as a folder that contains multiple files with a unique name. Different files with the same name can exist in various packages but can be identified with their package name. In Java, to define a namespace, the package keyword is used. C# provide namespace as a keyword.

2.2 Similarities and differences in constructs of Programming Languages

In this section, we are discussing about syntax similarities / difference to declare variables, iterations, conditions, functions and other constructs discussed in previous subsection. Semantically statements are the same but syntactically they differ with others languages. The topic needs a lot of writeup here, and we only present a few; the more details are discussed in the MS Thesis of the first author presented to the University of the Punjab, Lahore.

2.2.1 Literals

To represent binary literal, prefix `0b|0B` is used with binary number. Following languages support binary literals.

As per table 1, only C++, JAVA, Ruby, JAVASCRIPT, PHP, and Python support binary literals, and these used 0b as a prefix. We observed only one choice in existing well-known languages.

Similarly, Hexadecimal literal is represented with a prefix 0x or 0X, and Octal literal is defined by a prefix 0. All Languages which are chosen for unification, support the decimal literal. These languages do not use a prefix for decimal numbers. Character literal, when supported, is enclosed in a quote, e.g., ‘a’. Boolean literal is represented by a true/false word. PHP, VB.NET, and RUBY are case insensitive to represent the Boolean literal. We can use true/false either in the lower or an upper case. String literal is enclosed in a double quote, e.g., “Hello”. 

http://www.webology.org
Table 1- binary literals

<table>
<thead>
<tr>
<th>Language</th>
<th>Binary Digits</th>
<th>Prefix Characters</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB.NET</td>
<td>Not available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>0 1</td>
<td>0b</td>
<td>0B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2- 0B1000</td>
</tr>
<tr>
<td>C#</td>
<td>Not available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objective-C</td>
<td>Not available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAVA</td>
<td>0 1</td>
<td>0b</td>
<td>0B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2- 0B1000</td>
</tr>
<tr>
<td>JAVASCRIPT</td>
<td>0 1</td>
<td>0b</td>
<td>0B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2- 0B1000</td>
</tr>
<tr>
<td>PHP</td>
<td>0 1</td>
<td>0b</td>
<td>0B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2- 0B1000</td>
</tr>
<tr>
<td>RUBY</td>
<td>0 1</td>
<td>0b</td>
<td>0B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2- 0B1000</td>
</tr>
<tr>
<td>PYTHON</td>
<td>0 1</td>
<td>0b</td>
<td>0B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2- 0B1000</td>
</tr>
</tbody>
</table>

2.2.2 Primitive type

Non-Scripting languages are strongly type coupled. Assigning a value to a variable needs Type of variable. Byte type is used to store the small integer values in the memory. Value can be a negative or positive number. If we use byte then we can store values up to 255 but with a signed byte, values can be stored -128 to 127. To store the values greater than 255 the short type is used normally. It can store up to 65,535 for an unsigned short and for signed short, it ranges between -32768 to 32,767. In C#, C++ unsigned prefix is used with the short keyword. If values cannot be stored in byte or in short then integer primitive data type is used for large values. Long is used when the value is too large that cannot be stored in an integer primitive data type. In C# u is used as a prefix for unsigned values and in C++ unsigned is used before a long keyword for unsigned values. It consists of 4 bytes but depends upon the underline
hardware. To store the fractional value, a float primitive data type is used. It also consists of 4 bytes to store the value. The following table shows the syntax of the various languages. If the float value is too large that cannot be assigned to the float type, then the double type is used. In VB.Net type of double can be explicitly unsigned. To store the character literal, a Character data type is used. Its size also consists of 2 bytes. All object-oriented languages support the char data type and have common syntax among them. In C++ unsigned/signed prefix can also be used with the char keyword. On the other hand, scripting languages do not support char data type. The Boolean primitive data type is used to store the Boolean literals. String type is used to store the string literals. It is a special data type in object-based languages because string literal can be assigned either with a new keyword or with double-quotes.

Like, Table 1, the thesis of the first author contains 16 tables to compare all the types mentioned in the above paragraph.

### 2.2.3 Variable Declaration and Initialization

To declare or initialize a variable, it is declared as datatype, followed by the name of variable followed by initialization. Following table shows the syntax of well-known languages.

<table>
<thead>
<tr>
<th>Language</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>datatype identifiers[=initialization];</td>
<td>1- int x = 3;</td>
</tr>
<tr>
<td>C#</td>
<td>datatype identifiers[=initialization];</td>
<td>1- int x = 3;</td>
</tr>
</tbody>
</table>
As per table 2, C++, C#, Objective-C and JAVA have common structure for declaration and initialization. JAVASCRIPT has optional datatype var to declare variable as a global. PHP, RUBY and PYTHON almost common with other languages syntax but VB.NET has their own syntax.

### 2.2.4 Sequence of Instructions/Statements

To separate the statements in programming languages, different terminator is used. Semicolon in JavaScript can be omitted if statement is followed by line break or there is only one statement in a {block}. The following table shows the statement separator terminator. VB.NET, Ruby and Python used new line and other considered languages are using semicolon as statement terminators.

### 2.2.5 Exception Handling

Try-catch block is used to prevent the application from crashing. Every language is supported the exception handling. Keywords in VB.NET are case insensitive. It can be written either in upper case or lower case. Following table shows the syntax of various languages.
Table 3- exception catching

<table>
<thead>
<tr>
<th>Language</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB.Net</td>
<td>Try [statements] [Exit Try] [Catch[arg As exceptionClass]] [When expression] [statements] [Exit Try] [Catch…] [Finally [statements]] End Try</td>
<td>1- Try 2- result = num1 \ num2 3- Catch e As 4- DivideByZeroException 5- result =0 6- Finally 7- Console.WriteLine(result) 8- End Try</td>
</tr>
<tr>
<td>C++</td>
<td>try{ [statements] }catch(exceptionClass arg){ [statements] }</td>
<td>1- try{ 2- result=num1/num2; 3- }catch(Exception e){ 4- result=0; 5- }</td>
</tr>
<tr>
<td>C#</td>
<td>try{ [statements] }catch(exceptionClass arg){ [statements] } catch… [finally { [statements] }]}</td>
<td>1- try{ 2- result=num1/num2; 3- }catch(Exception e){ 4- result=0; 5- }</td>
</tr>
<tr>
<td>Language</td>
<td>Code Segment</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
</tbody>
</table>
| Objective-C | @try {  
  [statements]  
}[@catch(exceptionClass arg){  
  [statements]  
}[@catch…]  
[@finally{  
  [statements]  
}]] |
| JAVA | try{  
  [statements]  
}[@catch(exceptionClass arg){  
  [statements]  
}[@catch…]  
[@finally{  
  [statements]  
}]] |
| JavaScript | try {  
  statements  
}  
catch (arg){  
  statements  
}  
[ finally {  
  statements  
}]

```java
1- try {  
2- result=num1/num2;  
3- }[@catch (NSException *exception)] {  
4- result=0;  
5- }
```

```javascript
1- try {  
2- result=num1/num2;  
3- }catch(Exception e){  
4- result=0;  
5- }
```
<table>
<thead>
<tr>
<th>Language</th>
<th>Code Snippet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP</td>
<td>try {</td>
<td>try {</td>
</tr>
<tr>
<td></td>
<td>statements</td>
<td>$result=$num1/$num2;</td>
</tr>
<tr>
<td></td>
<td>} catch (exceptionClass argu) {</td>
<td>} catch (Exception $e) {</td>
</tr>
<tr>
<td></td>
<td>statements</td>
<td>$result=0;</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>RUBY</td>
<td>begin</td>
<td>begin</td>
</tr>
<tr>
<td></td>
<td>statements</td>
<td>result = num1/num2</td>
</tr>
<tr>
<td></td>
<td>rescue</td>
<td>rescue Exception =&gt; e</td>
</tr>
<tr>
<td></td>
<td>[exceptionClass=&gt;argu] statements</td>
<td>result=0</td>
</tr>
<tr>
<td></td>
<td>[rescue ...]</td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>[else</td>
<td></td>
</tr>
<tr>
<td></td>
<td>statements]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ensure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>statements]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>end</td>
<td></td>
</tr>
<tr>
<td>PYTHON</td>
<td>try:</td>
<td>try:</td>
</tr>
<tr>
<td></td>
<td>statements</td>
<td>result=num1/num2;</td>
</tr>
<tr>
<td></td>
<td>except [exceptionClass]: statements</td>
<td>except:</td>
</tr>
<tr>
<td></td>
<td>[ except… ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>else:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>statements</td>
<td></td>
</tr>
</tbody>
</table>

As per table 3, we can observe, that VB.NET, C++, C#, JAVA, JAVASCRIPT, and PHP have the same syntax to add the try-catch block. Syntax of the Objective-C also is the same but it adds @ sign before the keyword. RUBY uses the same structure but its keyword is different from other languages. PYTHON language also has its own syntax. There are multiple options to select the syntax for a unified programming language.
2.2.6 Decisions and Loops
Decisions and Loops usually start with a keyword like if, switch, case, while, repeat, do, perform, etc having a condition and followed by a block to be executed when the condition is true (maybe false in some cases). The block may be enclosed in braces, on ends at some reserved word like fi or wend, or even just a group of statements indented a level further. The keywords like break and continue may be used to further control the decisions and loops. The do, for and foreach loops are also in use.

2.2.7 Function, Classes, Comments, Arrays, and Packages
Like the above-mentioned constructs, we have analyzed the selected languages for the Function, Classes, Comments, Arrays, Packages, and presented tables in the thesis.

3 METHODOLOGY
Based on facts about existing programming languages presented in the last section, we are now presenting the specifications of our unified programming language.

3.1 Unified programming language (UPL)
From various online programming language rankings, it is observed that Java, C#, C, JavaScript, and Python are among the top ranked languages. PYPL [18] and TIOBE [19] communities rank the languages annually, REDMONK [20] do it biannually, and in 2017 it ranked JAVASCRIPT on top of first ten languages. IEEE Spectrum [21] is ranking annually based on statistical analysis, and last year, it ranked programming language C as topmost and JAVA as the second top in rank. We have gathered information about the ranking from online sources and tabulated it in Table 4. It shows top 10 programming languages based on their ranking by different communities by their statistics and index search-based method.
Based on usage, popularity and ranking, we have observed that Java and C/C++, and C#, along with JavaScript and python, are placed at top among other programming languages. Syntax of all Java, C/C++, C#, and JavaScript is mostly similar with exception of python. Python syntax for input/output is more intuitive and easier.

### 3.2 UPL Specification

For UPL, based upon above description, we have selected syntax of most of constructs from Java (C/C++, C#, and JavaScript) but for some constructs like input/output, we have selected syntax based on python language. Somewhere we have used syntax from low ranked languages or even customized it. In the following sections, syntax for various constructs of UPL are describes with examples.

#### 3.2.1 Variable Declaration and Initialization

The syntax for variable declaration and initialization is similar to that in C++/Java; that is, the data type of variable is followed by variable name and optionally followed by = operator and an appropriate expression. The statement
terminator semicolon has to appear at the end of the statement. Table 5 shows the syntax with an example for variable declaration and initialization.

Table 5 - UPL Variable Declaration and Initialization

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>type identifier = expression;</code></td>
<td>int a = 2+3; double d; Student s = new Student(1, “Arshad”);</td>
</tr>
</tbody>
</table>

3.2.2 Sequence of Instructions
To demarcate the two separate instructions, a semicolon is used for termination. Its syntax is like a JAVA/C#/C++/Objective-C.

Table 6 - Statement terminator in UPL

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>statement;</code></td>
<td>int a = 2; int y = 8;</td>
</tr>
</tbody>
</table>

3.2.3 Exception
To add a try-catch block in UPL syntax is the same as JAVA/ C#. Try-Catch block is defined in table 7, where finally block is optional.

Table 7 - try/catch block

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>`try {</td>
<td>try {</td>
</tr>
<tr>
<td>statement (s)</td>
<td>int a=1; int b=0; int c=a/b;</td>
</tr>
<tr>
<td>} catch ( argument ) {</td>
<td>}catch (Exception e) {</td>
</tr>
<tr>
<td>statement(s)</td>
<td>output 0, “denominator cannot be zero”;</td>
</tr>
<tr>
<td>} finally {</td>
<td>}</td>
</tr>
<tr>
<td>statement(s) }</td>
<td>}</td>
</tr>
</tbody>
</table>

3.2.4 Decision
The syntax to define conditional statement is more likely that JAVA, C#, C++, JAVASCRIPT, etc. which is if keyword followed by condition, then followed
by statement(s) and optionally followed by else /else if statement(s). Similarly, to use select statement instead of multiple else statements: its syntax is also similar to JAVA, C#, JAVASCRIPT, PYTHON, and PHP. Examples of decision statements are shown in table 8.

Table 8 - Conditional Statement in UPL

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>if (condition) statement(s)</code></td>
<td>if (a &gt; 0) output 0, “Good”;</td>
</tr>
<tr>
<td><code>[else][if(condition)]statement(s)</code></td>
<td>switch (a) {</td>
</tr>
<tr>
<td><code>switch (expression) {</code></td>
<td>case 0: output 0, “Good”;</td>
</tr>
<tr>
<td>case constant :</td>
<td>break;</td>
</tr>
<tr>
<td>statement(s)</td>
<td>case 1: output 0, “Bad”;</td>
</tr>
<tr>
<td>[default : statement(s)]</td>
<td>break;</td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
</tr>
</tbody>
</table>

3.2.5 Iteration

The Syntax of for/while/do-while loop is the same as of the JAVA, C#, C++, Objective-C, JAVASCRIPT, and PHP. Another form of a loop which is called an enhanced loop to iterate over the collection is similar to JAVA. Table 9 shows the loop structure in UPL.

Table 9 - LOOP in UPL

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>for{expression;expression;</code></td>
<td>for { int i=0; i &lt; 10; i++ }</td>
</tr>
<tr>
<td>expression}</td>
<td>output 0, i;</td>
</tr>
<tr>
<td>statement(s)</td>
<td>int array [ ] = {1, 2, 3, 4, 5}</td>
</tr>
<tr>
<td><code>for{variable:expression}</code></td>
<td>for { int a : array }</td>
</tr>
<tr>
<td>statement(s)</td>
<td>output 0, a;</td>
</tr>
<tr>
<td><code>while(expression)</code></td>
<td>while (a &lt; 10 )</td>
</tr>
<tr>
<td>statement(s)</td>
<td>output 0, a;</td>
</tr>
</tbody>
</table>
3.2.6 Function
The construct of method/function is taken from JAVA. In the function, a modifier is used for the access level of the function. It can be private, public, or protected. Table 10 shows the signature of the function.

Table 10 - Function definition in UPL

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifier returnType</td>
<td>void display ( int a, int b ){ output 0, a, b; }</td>
</tr>
<tr>
<td>identifier(arguments)</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>statement(s)</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

3.2.7 Output
The structure of the output function is similar to Python to print the values on a console, on a file, or a network stream. Where output is reserve word and N is device identifier. Expressions 1 to K are comma-separated expressions whose values are going to device N without any separator between them.

Table 11 - output structure

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>output N, expression1,expression1,</td>
<td>output 0, “Hello from UPL”; will be display the result: Hello from UPL</td>
</tr>
<tr>
<td>expressionK;</td>
<td>output 0, 2+6, 5, 7*2; will print the result of expression: 8514</td>
</tr>
<tr>
<td></td>
<td>output 0, ‘ ‘, 2+6, ‘ ‘, 5, ‘ ‘, 7*2; output of expression will be 8514</td>
</tr>
</tbody>
</table>

3.2.8 Input
The syntax of the input function is also similar to the Python language. In UPL input is the reserved word and N is the device identifier. Variables 1 to K are comma-separated whose values are coming from device N.
Table 12 - input syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>input N, variable1, variable2,..., variableK;</code></td>
<td><code>input 0,a,b,c;</code></td>
</tr>
</tbody>
</table>

3.2.9 Comments

To add the comment in Unified programming language, java like structure is used. There can be added a single line or multiline comment can be added.

Table 13 - Comments in UPL

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>// single line comment</code></td>
<td><code>// value of x must be even</code></td>
</tr>
<tr>
<td><code>/* multiline Comment */</code></td>
<td><code>int x = 2;</code></td>
</tr>
</tbody>
</table>

3.2.10 Class

The structure of the class is taken from JAVA / C#. The syntax of the class is access-modifier followed by class name and then by ‘{’ left curly brace after which body of the class and then subsequently by right curly brace ‘}’.

Table 14 - Class definition

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>class Modifier class identifier {</code></td>
<td><code>class Student {</code></td>
</tr>
<tr>
<td><code>   //class members and</code></td>
<td><code>    // body of the class</code></td>
</tr>
<tr>
<td><code>    functions</code></td>
<td><code>    }</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
</tr>
</tbody>
</table>

3.2.11 Inheritance

Inheritance is used for the parent-child relationship. To enhance the functionality of the parent class to the child class, the Parent Class is extended with the child class. In UPL inherit word has been reserved to extend the parent class.
### 3.2.12 Array

The syntax for array declaration and initialization is similar to that in C#/Java, which is the data type of variable followed by variable name then follow by [] and optionally followed by = operator and an appropriate expression. The statement terminator semicolon has to appear at end of the statement. Table 16 shows the syntax along with an example for array declaration and initialization.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>type identifier[] = expression</td>
<td>boolean array [];</td>
</tr>
<tr>
<td></td>
<td>int array [] = { 1, 2, 3 };</td>
</tr>
<tr>
<td></td>
<td>int array [] = new int [5];</td>
</tr>
</tbody>
</table>

### 3.2.13 Package / Namespace

To group the classes in UPL, the structure of the package is taken from JAVA. The syntax of the package is the package keyword followed by name of the package then followed by a semicolon.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>package identifier;</td>
<td>package upl.lib;</td>
</tr>
</tbody>
</table>
4 IMPLEMENTATION OF UPL

In this section, we present the prototype of UPL implementation as proof of the concept.

4.1 UPL Compiler

A translation strategy known as a transpiler/transcompiler [22] used for source-to-source language translation is employed to produce identical source code in another language from the wellspring of one dialect. The java2c-transpiler is utilized to make Java into C (a programming language). J2Objc [23] is used for transcompiling Java code into Objective-C. Transcompilation of PHP to C++ is achieved through the HipHop [24] transcompiler. Different instruments, such as LLVM, ROSE, and ANTLR, can generate lexers and parsers for specific languages. An abstract syntax tree is created by taking the source code and linguistic use records of the source code, which determines the sentence structure control of the dialect. You can then change each hub of the tree by rewriting the code.

4.1.1 LLVM

LLVM [25] is a collection of projects for compiler construction and reusable toolchain technologies []. It is an open-source license project. Primarily it was built to generate the target machine code along with the optimizer feature. Clang and vmkit are the subprojects built based on LLVM core libraries. Clang implementation is an “LLVM native C/C++/Objective-C compiler” to provide a fast compilation. Clang compiler generates the AST tree. So, source-to-source transformation can be performed by using Clang-provided interfaces to fire the event while entering/existing on the AST Node. Another project which is vmkit is implemented for JAVA, .Net virtual machines.

4.1.2 ROSE

ROSE [26] compiler is by and large intended to bolster examination and streamlining for source code. Because of fast changes in the processor to a multicore framework, it is excessively expensive to modify the current application as per the new frameworks. As a source-to-source compiler
foundation, it gives a change of substantial scale application for future PC structures and programming investigation and confirmation. ROSE gives diverse levels of interfaces to bolster a piece of code for change by crossing dynamic linguistic structure trees. At the low level, APIs give distinctive part capacity to various kinds of items like tree hub, image tables, record area data, preprocessing data, and traits.

In the abnormal state AST interface, subtrees can be worked by given capacities and can be converted into existing AST. There are likewise aide capacities exist for tree walker, erase or duplicate the Node. It bolsters various dialects like JAVA, C, C++, FORTRAN, and so forth for change from one source to another source.

![ROSE compiler architecture](image)

**Figure 7- ROSE compiler architecture**

### 4.1.3 ANTLR

ANTLR [27] is another tool to transform code from one source to another source. It generates the lexer and parser according to the grammar of the language. It is used not only for the construction of the tree but also to support tree walking, in addition to error recovery and error reporting. It provides the listener interface that contains enter and exist Rule function, which is called via manipulating the tree.

http://www.webology.org
If there is grammar rule $S \rightarrow AB$, $A \rightarrow a$, $B \rightarrow b$ then listener interface will provide the call back functions which would be like this, enterS(Context), existS(Context), enterA(Context), exitA(Context), enterB(Context), exitB(Context), and visitTerminal(TerimanlNode). In figure 9, there is grammar for the arithmetic expression for addition, subtraction, multiplication, and division where prog is the start rule of the grammar. We can give multiple expressions with lines separated. If we execute the following command for the expression 100+2*34, it will produce the graphical parseTree.
To implement UPL, we have selected ANTLR, as LLVM is more low-level and ROSE focus on optimization. ANTLR provides high-level source code to source code translation, which is the fundamental requirement, we need. To generate the Lexer and Parser for Unified language, we build the grammar file whose rules are close to JAVA and C# grammar and most of the rules are in the Chomsky normal form. As in the picture we can see, that UPLLexer, UPLParser, and UPLListener are generated by the ANTLR tool. It gives the ParseTreeWalker API that takes the implementation of a listener as well as AST return by UPLParser as an argument. During traversing transformation, callback functions are called while entering and leaving the tree node.

The first goal of UPL is the development of a language construct that can be viewed as a generalization of all the known programming languages. As in the

4.2 UPL Implementation

To implement UPL, we have selected ANTLR, as LLVM is more low-level and ROSE focus on optimization. ANTLR provides high-level source code to source code translation, which is the fundamental requirement, we need.

To generate the Lexer and Parser for Unified language, we build the grammar file whose rules are close to JAVA and C# grammar and most of the rules are in the Chomsky normal form. As in the picture we can see, that UPLLexer, UPLParser, and UPLListener are generated by the ANTLR tool. It gives the ParseTreeWalker API that takes the implementation of a listener as well as AST return by UPLParser as an argument. During traversing transformation, callback functions are called while entering and leaving the tree node.

The first goal of UPL is the development of a language construct that can be viewed as a generalization of all the known programming languages. As in the
snippet, we have created a Hello World example. The first line represents the namespace/package in which a group of classes is organized. All libraries are imported/included after the package declaration in java but in C#, all the libraries are included before the namespace declaration. Although semantically statement is true in C# but syntactically it would be wrong. To transform from UPL to C# re-positioned package statement and replace the package keyword with namespace keyword. Most of the languages support the class structure, like Java, C#, PHP, VB.Net, etc. which represent the blueprint of the object. The rule of the class statement is that modifier followed by class keyword followed by class Name. Here the public is the modifier which means the scope of the class is public and can be shared object of this class with other elements. The statement output’s first argument is the integer values that is representing which socket output will be written. If the value is 0 then it will print on the console, same as 1 for file and 2 for the network. Then multiple arguments can be passed with commas separated after specifying the socket mode. To translate the above code, into High-level programming languages, API will create the Abstract syntax tree after parsing the lexeme that can be seen in a graphical form in figures 11 and 12.

Figure 11- ANTLR generated classes

Figure 12- UPL HelloWorld program

Here is the start rule that is compilationUnit can be packageDeclaration followed by typeDeclaration followed by EOF. In the image, we can see the first rule created three child nodes. Similarly, other nodes show their child node as per their rule. All terminals are on the leaf nodes. Every non-leaf node has a call
back function enter and exist which will be called at the time of entering the node. After traversing all their child nodes exist rule will be called. So, at the time of entering or existing, we can easily replace the keyword/statement with the actual desired language. Implementation can be found in the src-code of JavaTranslation and CSharpTranslation files which are extending the UPLBaseListener class. Translation.java is the main class that takes three arguments first is the language name in which code will be translated then ‘-ps’ followed by the path of the input file/folder. We can see the result of the translation file which has generated the files with the extension .java and .cs respectively.

Figure 13- Abstract syntax tree of Helloworld program

4.3 UPL in Action

To compile and run the .upl file, we need to install the UPL on windows provided by the installer. During the installation of UPL, environment variables will be set in the classpath of the system variable. After installation, write a program and save the file with the .upl extension. For compilation “uc” command is used. Format of the command “uc [-cs/java] file1 file2…” where uc is the command name followed by optionally output language attribute followed by filenames. By default, it will convert the file with the .java extension. In figure 13 we can see the HelloWorld program in upl language. To generate the code in JAVA and CSharp, run the command “uc HelloWold.upl”
and “uc –cs HelloWorld.upl” respectively. As in figure 14, we can see the, there are two files have been generated with the .java and .cs extension.

![Command Prompt]

Figure 14- Output of UPL

5 CONCLUSION

As we tend to know, abundant of the planet is currently in a very digital era, and thereupon programming and jobs have drowned. Writing computer programs is a need in our reality now. Expertise individuals need to be aware! Previously, programming or coding was looked down by many people because many thought it was "nerdy". Nonetheless, with the rise of the digital market, programming is currently a worthwhile quality that several individuals need to acquire, and lots of enterprises are trying to find programmers to hire. There's a massive market for programmers, however a very restricted pool of programmers. Like this skill, many others are widely available and it'd be nice to find out at least one among these skills.

The reason for the UPL venture is to bolster the engineers/developers to take in the rationale as opposed to investing the energy in linguistic structure learning of various dialects. The application zone of UPL to help the understudies in the field of programming as a first programming dialect in their foundation.
There are numerous provocations to present another Programming language. UPL is anticipated to be a latest dialect in the High-level ritual. In present circumstances with expertise, it is likely to frame a less troublesome, versatile, and skilled dialect considering a High-level dialect. Regardless valuable programming, UPL supports dissent arranged programming. Work is going on to upgrade the bolster for OOP. The overall perspective must be dissent arranged.

The upcoming difficulties will be identifying grammar mistakes and parallel ordering through a module/IDE for UPL. The upcoming research project aims to investigate the ultimate success of UPL implementation according to these future difficulties. On the next pages, these difficulties or in other words challenges are presented in a listed form and these are real inspirations for the researcher and developers to contribute to the area related to Unified Programming Languages.

6 ACKNOWLEDGEMENTS

This paper is based on the research and development work conducted at the University of the Punjab by the first author of the paper as partial fulfillment of his M.Phil. Computer Science degree. As the consequence, a major portion of the text, tables, and diagrams are taken directly from his thesis, which is not published anywhere else except in his submitted thesis.

REFERENCES


http://www.webology.org


