Jackie Diep

Professor Bryant Walley

CSC 408 G001

23 November 2020

Programming Languages Report

Find a problem, devise a solution, and iterate to improve. As a student and an individual, a significant portion of my identity follows these steps. Software Development is the ideal field for me because it enables me to utilize my identity as a career. I want to have a part in creating solutions and products that change the way people operate for the better.

Programming languages are developed as a means to instruct computers to perform specific tasks (Ferguson 1). They exist as a means for us to take advantage of technological advances over time. Currently, I am able to program in C++ and Java, and I intend to be able to program in C and Python in the near future. In this paper, I will be covering the four aforementioned programming languages as well as ALGOL because of its historical importance.

1. ALGOL

One of the most influential programming languages to ever exist, ALGOL was originally designed for publishing algorithms and completing computations ("The ALGOL Programming Language"). ALGOL was developed by a joint committee at the Swiss Federal Institute of Technology who were members of the Association of Computing Machinery. Academically, the language was popular for mathematical reasons and publishing algorithms, but commercially never became popularized outside of financial calculations (Ferguson 1). Though it is not in use today, ALGOL is the origin of many of the world's most popular programming languages, three of which are covered in this paper.

Code blocks and begin...end pairs became popularized with ALGOL along with the nested functions with lexical scopes. The language used three different forms of syntax: reference, publication, and implementation; allowing for different keyword names and conventions.

Due to the new elements implemented in the 1968 version of ALGOL, the language was considered overly complex and too difficult to use comfortably leading to the adoption of simpler and more compact languages (Ferguson 1). It is worth noting that it remained the algorithm publication standard for nearly three decades later (Wood).

An example of ALGOL 58 uses an array with the name of the array as a pointer, letting the REPLACE statement be used.

```
BEGIN
  FILE F(KIND=REMOTE);
  EBCDIC ARRAY E[0:6];
  REPLACE E BY "EXAMPLE";
  WRITE(F, *, E);
END.
```

One of the features added to later versions of the new defunct language was the I/O system named transput. Transput has three primary elements: books, channels, and files. Books are like data structures that are divided into sections called pages, that contain lines and characters like a more complicated but rigidly restricted array. A channel is a piece of hardware that is able to interact with the I/O of the system, like a scanner or a printer. A file is a term used to describe the actual method of communication between the operating system and the books opened by a channel. Files were implemented with many details to specify the purpose of each particular file, the position in which to begin reading information to and from the book, and routines to describe how to use the information.

An example of a formatted transput (“Format syntax in ALGOL 68G”) is:

```
printf (($2l"The sum is:"x, g(0)$, m + n));
print ((new line, new line, "The sum is:", space, whole (m + n, 0))
```

The code uses many layout routines such as “new line” and “space” and uses the “$” character to hold the desired format making it difficult to comprehend, and the code is rigid with little room for error. Though the code is unintuitive and many more similarly rigid and complex features were added, it is easy to see why future languages that simplified aspects of ALGOL 68 were able to reach such levels of popularity.

2. C

As a successor of the programming language B, C was developed to go along with the Unix operating system as a way to make utilities for the new platform (Ritchie 1). As a derivative of ALGOL, C has many of the previously popularized elements like lexical scopes, recursion, and structured programming. After a popularity boom in the 1980s, C is one of the most commonly used programming languages today ("Programming Language Popularity").

C was developed at Bell Labs by Dennis Ritchie, one of the developers of the Unix OS and the programming language B, in the 1970s. Ritchie and his partner, Ken Thompson, were recipients of the Turing Award, Hamming Medal, and the National Medal of Technology. While Thompson was credited with Unix, Ritchie is credited with porting the OS to many different platforms (“Pioneer Programmer Shaped the Evolution of Computers”). With the widespread popularity of C and the influence of Unix in modern OS, the two are some of the most influential computer scientists in history.

In C, executable code is written in subroutines called functions. Parameters are always passed by value; however, pointers may be used to pass-by-reference. C does not support features like object-oriented programming and garbage collection, though it did inspire many other languages that do including: C++, Java, Python, and more. An important ability of C is that it was released as a procedural programming language with low-level access to memory and machine instructions that was also portable enough to be used for cross-platform programming.

C is popular because of its flexibility in memory management and the control it gives the programmer. Programmers can choose how memory is used, when to change it, and where it should be placed. Along with these factors, C's low-level access and portability make it popular for system-level programming (Kumar).

In my research and experience, C has a very important issue that I believe should be worked on. The issue is that very apparent errors are left unreported by the language, leading to unnoticed or difficult to locate bugs. For example, when working with pointers, arrays, or both, an unintuitive error may occur where a memory address is returned instead of a value. The memory address may then be used for mathematical operations while it should be clear to both the compiler and the user that it is not the intended purpose. This is just one example that shows that, for a programming language that allows as much freedom as it is unintuitive, C has a stunning lack of warnings. Also, because C is a programming language that naturally results in a lot of bugs that must be ironed out, development time and costs are higher when programs are developed in C (Barland).

As a band-aid solution to this problem, I believe warnings should be much more common in both C and C++. For example, when a memory address is used in a boolean or mathematical operation, a warning should be triggered and notify the user for safety's sake. I am not sure what a long-term solution would look like without a huge overhaul of the entire language.

Despite this issue, C is used very effectively in most modern OS and embedded systems in consumer products like cars, clocks, coffee makers, etc.

A simple “Hello World” print statement for C contains the void function main with a print function inside.

```
#include
void main ()
{
  printf (“Hello World”);
}
```

It is worth noting that strings in C are null character arrays and not strings like the datatype in C++. As C++ is “C with classes” it can also be said that C is “C++ with less fluff”. C gives developers a faster and more straightforward method to develop programs that may not require many of the additions that C++ provides. C may be used to mimic the effects of a class within a program using global and static variables, but this is a very inefficient and unstable road to follow that is worth sacrificing the simplicity of C to avoid in the case that classes are required.

Memory in C is something that often has to be taken into account by the developer. When using the heap and the malloc function, memory must be freed when the variable is no longer in use, or a memory leak will be created causing the program to use more memory than it should.

The prototype for “malloc()” and “free()” are:

```
void *malloc(size_t size)
void free(void *ptr)
```

where “size” is the number of bytes to set aside.

One of the biggest differences between C and other languages, like Python and Java, is pointers. A pointer is a variable that holds a memory address. Pointers are a double edged sword in that they have a variety of uses that greatly improve the flexibility and efficiency of C, but they also tend to become complicated and make it much more difficult to fully comprehend code. The fondness or dislike of pointers can sometimes be the deciding factor between using

C or using another language. Some of the benefits of pointers are pass by reference in function calling, dynamic memory allocation, for example, dynamically created arrays, and many data structures in C are made much more efficient by pointers.

For C and similarly C++, it is often said that their greatest strength is flexibility and the ability to allow a developer to do anything, and nothing makes that more apparent than pointers. While many other popular languages tend to stray away from pointers because of their complexity and can cause massive amounts of errors and security issues, C embraces pointers and all that they represent. The biggest difference that C and C++ have compared to the other languages that will be covered is that they are willing to accept complexity and difficulty in exchange for flexibility and general usability.

The following example for pass by reference function calling initializes a void function prototype with the “*” before the variable names to initialize it as a pointer. Inside the main function at some point, a pointer will be assigned a memory address which will be passed into the function call. By passing the variable this way, instead of a copy of the variable sent to the function, the memory address will be sent directly, and any changes made to the variable in the function will also carry over to main.

An example of pass by reference function calling:

```
void function(int *variable);

int main()
{
        int *variable = memory address;
        function(*variable);
}

void function(int *variable)
        {        //definition             }
```

The following example for a dynamically created array allows an array to be created during run time, which is useful when the size of the array is not known at compile time. To do this, we use malloc() to assign the address to the created pointer variable, and we prompt the user to input array size during runtime, though this can be done without user-specific input. An example of a dynamically created array:

```
int main()
{
        int *arrayPointer = NULL;
        int arraySize;
        printf("Enter the size of the array: ");
        scanf(“%d”, arraySize);
        arrayPointer = (int*) malloc (arraySize * sizeof(int));
}
```

One of the data structures in C that uses pointers is a linked list. Though the entirety of a linked list program may be a little lengthy for this paper, a description and the basic struct can fit. Linked lists are linear data structures that use structs commonly referred to as “Nodes”. The Nodes typically have two components, a variable to hold some form of data and a node pointer that holds the location of the next node in the list. The advantages of this data structure over other ones like arrays are that the size can be adjusted at any point in time and new Nodes can be inserted into the data structure without moving any of the other elements, making it much faster in large programs.

```
struct Node
{
dataType variable;
struct Node* pointer;
}
```

As most modern changes to programming languages occur in OOP languages, most changes to C are bug fixes, though in 2011 with C11, support for multithreading, Unicode, and other smaller features already available in other languages were implemented.

3. Python

Python was developed to be as “unpretentious” as possible. Python embraces the motto “there should be one - and preferably only one - obvious way to do it” design philosophy (Peters). The language’s goal is for a simple and low-clutter syntax with grammar that still gives programmers a choice in style.

In the late 1980s Guido van Rossum created Python in the Netherlands at Centrum Wiskunde & Informatica (CWI). Rossum held a number of issues with the ABC language, but he was a fan of many of the features it provided, mainly the lack of extensibility to remedy other issues. While working with the Amoeba group at the CWI, Rossum decided to combine the Amoeba system calls and the syntax of ABC to fulfill his needs, leading to Python (Peters). In 2018, Rossum stepped away from his position for the Python language “Benevolent Dictator for Life”, and officially retired in 2020. As of 2020, Python is considered the top programming language in the world by the PYPL.

Python’s design philosophy makes it an excellent teaching tool and approachable language. The readability and simplicity of the language paired with the OOP support makes it stand out from other languages for both small and large projects (Kuhlman 1). It also does not require curly brackets or semicolons to delimit blocks or end statements, instead using whitespace indentation to promote visual structure.

Two important elements of Python that have existed since its original state are fewer exceptions and special cases than other popular languages like C and Pascal, and the use of keywords instead of punctuation to provide immediate recognition of a code’s purpose.

Due to the aforementioned friendly elements and the concept of a “correct” way to do things, Python became and has remained near the top of the most used and taught programming languages in Software Development today. These days, Python is used to run essential services like Google's search engine, Youtube's video integration, and various GPS navigation apps.

Python 2.0 was released on October 16, 2000 with new features such as a garbage collector and Unicode support; however, in 2020, Python 2.0 has been discontinued and only Python 3.5, a majorly revised version of the language, and onwards will be supported (Kuchling, Zadka).

Python 3.5 is an actively developed and supported programming language that attempts to further the strengths of Python 2.0 and offer better support for more modern processes like AI and machine learning ("Python 2 vs Python 3: Key Differences"). To better the simplicity of the language, integer division will return a float instead of an integer, global variable value will not be changed inside of loops, strings are stored in Unicode by default instead of requiring definitions, etc. In efforts to increase cross language and teaching uses, some simple changes like changing the syntax of certain statements were implemented. For example, in Python 2.0, the print statement does not use parentheses, but in Python 3.5, the print statement was made to more closely resemble other languages.

```
Python 2.0:   print "Hello World!"
Python 3.5:   print ("Hello World!")
```

In particular, increases in cross language usability are interesting because of Python's ease of use for simple tasks, and the advantage that other platforms have in more complicated and efficiency based tasks.

As a language similar to C++, Python supports both multiple inheritance and multilevel inheritance.

Multiple inheritance where the attributes of the base classes "One" and "Two" are inherited by derived class "Three":

```
class One:...
class Two:...
class Three(One, Two):...
```

Multilevel inheritance where the attributes of a base class “One” is inherited through the attributes of a derived class “Two”:

```
class One:...
class Two(One):...
class Three(Two):...
```

As per usual, Python’s syntax for inheritance is much easier and understandable than C++, but it is also limited by many of the issues that Python has, such as speed and efficiency. In line with this, Python’s memory management is typically easier than C. Python’s garbage collector uses reference-counting to release the memory of a variable when there are no more references to it in the program (Jain). Python’s memory manager will typically manage all memory unless the developer specifically allocates memory otherwise.

One of the most versatile features of Python that other languages do not have is the Python list data type. Whereas in C or other languages, something like stack may take several functions to implement, a list can be used as a stack pretty easily with the append and pop functions provided by Python.

Example of a list used as a stack:

```
listStack = [1, 2, 3]
# the stack is currently 1, 2, 3
stack.append(4)
# the stack is now 1, 2, 3, 4
stack.pop()
# the stack is now 1, 2, 3
```

Another versatile but extremely simple feature of Python is the range() function. The range() function is a built in iterator function that takes up to three parameters: Start, Stop, and Step. The Start and Stop parameters obviously set up the beginning and end of the iterator, and the Step iterator tells the function what amount to iterate by. The range() function is interesting because it simplifies many of the tasks that could be done in other languages. For example, the function could be used as the sole parameter of a for loop in Python, or it could be used to list the multiples of a number between certain values.

Example of the range function as a parameter of a for loop:

```
for x in range(1, 100):
   print(x)
# By leaving the Step parameter empty, the function iterates by 1
# If only one parameter, the range would use it as the Stop parameter
#       and Start would be assumed to be 0
# 1-100 would be printed 1 integer at a time
```

Example of the range function being used to print a list of multiples of a number:

```
print(list(range(3, 100, 3)))
# Start at 3, End at 100, iterate by 3
#       Prints the multiples of 3 that are between 3 and 100
```

While both of the above examples could easily be completed using other languages, it is always several lines shorter using range(), and it is much simpler to understand how it works which fits perfectly with Python. Another thing to note is that, though the print statements can be pretty obvious, being able to use an iterator to quickly create a list that can be used as a parameter for a larger and more complex function is an extremely powerful tool to have.

While there are many more examples of functions like range() that simplify tasks in Python, I believe that listing more would be a moot point because the point has been made. Python is built to be an easy to use and no nonsense language, the existence of functions like range() that simplify various tasks and the versatility of lists make it clear how true that really is. This is not to say that Python is better at everything, though. For example, Python definitely does not have as much flexibility that C and C++ are given due to pointers, though it does have work arounds that may or may not be worth using depending on the situation. This leads into the next point.

Although Python is an easily readable and useful language, there are still some imperfections. For one, it is objectively slower than other popular languages in situations that require good performance. Another issue that many programmers run into is a lack of static typing. Python will not stop users from redefining variables that have already been declared, so if a team of developers are working on a project, variable names may accidentally be reused (Pillai).

As a tradeoff for its ease of use, Python suffers from a few primary issues. It is generally slower than other languages because it uses an interpreter rather than a compiler. Whereas a compiler would read the source code into machine code, interpreters read code in line by line. This usually makes things like debugging easier, but speed during execution is much slower. That being said, being dynamically typed makes certain issues only appear during runtime and it is possible that debugging can take extra time. Python is also less memory efficient than other languages. In exchange for memory efficiency, Python allows ease of use capabilities like flexible data types and lists (“Disadvantages of Python”).

Python’s biggest issue that must be remedied is multithreading capabilities. As technology continues to progress, the uses of multithreading become more commonly required. Though Python’s multithreading capabilities exist, they are not able to properly utilize more than one core and workarounds are not always efficient enough to be worth using (Yegulalp).

On the bright side, Python is a very actively developed language and, while there are no obvious solutions to the aforementioned problems, small changes are always being attempted to help remedy them.

Personally, I think that Python is a language best used in conjunction with other languages. I have a lot of hope for Python’s future development with cross language compatibility to complete less efficiency based tasks in the middle of any development cycle. In exchange for many features and efficiency, Python is the most simple, intuitive, and, in terms of development time, efficient programming language, and the constant development of Python is an exciting prospect.

4. C++

C++, a derivative of ALGOL and C, was developed initially as “C with classes”. It was designed primarily with efficiency and flexibility in system programming and embedded software in large systems in mind (Stroustrup).

In 1979, C++ was developed by Bjarne Stroustrup, a Danish computer scientist. Simula, another language that Stroustrup had used and found helpful was too slow to use, so he chose to combine Simula-like features with other languages including ALGOL but mainly the faster and portable C. After implementing many class features, default arguments, and an input/output library, the first edition of C++ was released in 1985 (Stroustrup, *Lecture:The essence of C++*). Today, Stroustrup is a Technical Fellow and Managing Director at Morgan Stanley, an investment bank and financial services company.

C++ primarily has two components that separate it from competitor languages. Due to its inheritance from C, C++ has a direct mapping of hardware features that results in zero-overhead abstractions. This means that C++ is able to *efficiently* have both hardware access and abstraction (Stroustrup, *Thoughts on C++17 - An Interview*).

On release, C++ introduced OOP to C. The implementation of classes also provides four key features: abstraction, encapsulation, inheritance, and polymorphism. Unlike other OOP supporting classes, C++ classes support deterministic destructors leading to RAII. RAII, a term for automatic resource management, is widely used in C++ because destructors are called when the object goes out of scope.

C++ is popular because it iterates off of the already popular and portable C, and introduces higher level programming styles in the same efficient manner.

In combination with C, C++ is used in many operating systems and embedded programs. It is also a primary component of many databases, browsers, GUI, and other facets of life (“What Is C++ Used For?”).

Newer editions of C++ have been released that further build on OOP capabilities such as multiple inheritance, static and const member functions, and protected members. The most recent versions of C++ have even further expanded libraries and features to provide as many capabilities to programmers as possible (Stroustrup, *The C++ Programming Language Second Edition*).

The “Hello World” print statement for C++ is similar to the C statement, but it includes an int function main and the standard library.

```
#include <iostream>
int main()
{
 std::cout << "Hello world!";
}
```

As “C with classes”, C++ has a big focus on classes, so obviously C++ supports both multilevel and multiple inheritance.

Multilevel inheritance where class “Three” inherits the attributes of base class “One” through derived class “Two”:

```
class One{};
class Two: public One{};
class Three: public Two{};
```

Multiple inheritance where class “Three” inherits both the attributes of classes “One” and “Two”:

```
class One{};
class Two{};
class Three: public One, public Two {};
```

Many of the modern C++ enhancements also include improvements to ease of use, such as the “auto” keyword. As long as the variable is initialized with a value, the “auto” keyword can be used in place of a data type to request that the compiler assign one automatically (Chowdhury).

```
auto data = 5;          // Automatically declared as an integer
auto data = 5.0;        // Automatically declared as a float
```

In comparison to C, C++ arguably only has advantages in terms of memory management. C++ provides the same features as C, in addition to constructors, destructors, pointers, and other C++ specific features. The most commonly used difference between C and C++ in memory allocation is the “new” operator.

ClassName *classObject = new ClassName;

The “new” operator allocates the memory required for the object and then calls the operator, simplifying the process for the developer. It is typically not as efficient as malloc() and the “delete” operator still has to be called, similar to free().

delete classObject;

Similarly, many of the uses of pointers also use the “new” operator instead of malloc(). For example, the creation of a dynamic array in C vs C++ is listed below:

```
C
        int main()
        {
                int *arrayPointer = NULL;
                int arraySize;
                printf("Enter the size of the array: ");
                scanf(“%d”, arraySize);
                arrayPointer = (int*) malloc (arraySize * sizeof(int));    //malloc()
        }

C++
        int main()
        {
                int *arrayPointer = NULL;
                int arraySize;
                std::cout << “Enter the size of the array: “;
                std::cin << arraySize
                arrayPointer = new int[arraySize];          // “new” operator
        }
```

Above, the C and C++ examples use different output and input statements, but the main point is the difference between malloc() and “new”. It is much simpler than malloc(), but it is less efficient in terms of memory usage as less information is used to call it.

An interesting aspect of pointers and classes is that a pointer can be used to access members of a class using the Member Access operator (->).

```
class classExample
{
        public:
                classExample(int variable = 1)
                {
                        exampleVariable = variable;
                }
                int getVariable()
                {
                        std::cout << “Variable: “ << exampleVariable << endl;
                }

        private:
                int exampleVariable;
};

int main()
{
        classExample exampleObject;
        classExample *examplePointer;
        examplePointer = &exampleObject;                //& operator to = memory address

        cout << examplePointer->getVariable() << endl;      //Output is Variable: 1
                                                             //Member Access Operator ->
}
```

In the code above, the pointer is able to access the members of the specific class object because it is a pointer of the same class that is pointed at that particular object’s memory location. Likewise, if an object were to be declared with classExample exampleObject(5) and the pointer were to point towards that object, the output would be “Variable: 5”.The & operator is used to set examplePointer equal to the memory location of the object, and the -> (Member Access) operator is used to access the member and call the getVariable() function of that object.

Similarly to C, pointers are one of the most polarizing aspects of the language because of the huge amount of flexibility they provide at the cost of complexity and difficulty in programming. C++ gains even more on both sides of the argument because of its inclusion of classes and the things that pointers can do when combined with them, like displayed above.

C++ is often referenced to have issues with code bloat, slower compile times than some competitors, and not being entirely compatible with exceptions. Truthfully, the majority of problems found in C++ are derived from C, and as such, my opinion of C++ is very similar to my opinion of C. The lack of warnings present for many of the common and unintuitive errors that programmers run into is unacceptable, but C and C++ are still extremely powerful programming languages. There is a reason that they are so commonly used, even with some of the irritating quirks that they have.

5. Java

Java is an OOP language that was created with the “write once, run anywhere” (WORA) principle in mind. Typically, this means that code programmed in Java, once compiled, can be run on any other machine that Java supports without being compiled again.

The productions of Java began in 1991 with James Gosling, Mike Sheridan, and Patrick Naughton, though Gosling is credited with creating the language.. The language went through a variety of names before falling upon Java, and it was originally designed for television. Gosling thought of the Java WORA principle while writing a porting program by emulating hardware. After Sun Microsystems, the company Gosling worked for, was acquired by Oracle, Gosling left and maintains a critical stance towards Oracle (Guevin).

The primary aspect of Java that stands out from other languages is the Java Virtual Machine (JVM). The JVM allows any platform that can support Java to run compiled Java code without being recompiled. Similar to the porting program written by Gosling, the JVM emulates hardware such that WORA is possible. JVM instructions for optimization and garbage collection are not specified to not constrain implementation (Venners 5).

Java was released with syntax similar to C++ and C, along with WORA portability, automatic memory management, and decently configurable security made it popular with web browsers and client-server applications. Today, Java has maintained this popularity and sees widespread use in coding of well known names like the Maestro Mars Rover, Minecraft, Facebook, and Twitter.

Over time, Java has become a constantly updated open source programming language with over a dozen changes and versions in the last two decades. Some of the changes include support for text blocks and hidden classes, which are classes that cannot be used by other classes and are primarily created during runtime.

The "Hello World" print statement for Java uses similar syntax to the C and C++ statements, but it includes a class definition matching the name of the file, and a main *method*.

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

Because Java is a completely object oriented language, Java differs heavily from C++ and Python in that it only allows single class inheritance. Java interfaces have to be used to create multiple inheritance, and pointers are much more limited in Java than they are other popular languages. Many programmers prefer Java for its system-based memory management and criticize C++ for otherwise.

To extend inheritance using multiple interfaces where class "one" inherits, or "implements", the "two", "three", and "four" interfaces:

```
public class one implements two, three, four{}
```

Another difference between Java and C++, is that the try/catch block is a necessity because Java does not support destructors. The try/catch block allows for runtime testing and a definition of steps to be taken during exceptions.

```
try{...}
catch(exception){...}
```

Also because Java is completely object oriented, global scopes do not exist and all functions and variables must be within a class, though package scopes do exist (“C++ Vs Java”).

One of the biggest differences between Java and C++, is that Java does not support pointers in the same way. Pointers in java are used for references to class objects, but they are not supported for developer use. For example, by referencing a class object, say “ref1”, and setting that reference equal to another reference, say “ref2”, it can be seen that changes to one reference’s class object variables also changes the other’s class object variables. This is because they use pointers in the background. Even though they use pointers, they only use the fact that pointers to an object. They do not typically use pointers for anything else.

For example, displaying that Java uses pointers in object references (Bagri):

```
class ExampleClass
{
        int exampleVariable;
}

class ExampleClassDisplay
{
        public static void main(String args[])
        {
                ExampleClass ref1 = new ExampleClass();
                ExampleClass ref2 = ref1;
                ref1.variable = 325;
                ref2.variable = 499;
                System.out.println(“Ref1: “ + ref1.variable);
                System.out.println(“Ref2: “ + ref2.variable);
                        //      The output is
                        //      Ref1: 499
                        //      Ref2: 499
        }
{
```

In the above code, the example described in the previous paragraph is shown. A new class object ref1 is created and set equal to ref2 to prove that changes to either reference causes a change in both, which proves that references use pointers similarly to C and C++.

The restrictions on Java's pointers is because it does not truly have a need for them as an OOP language, and it is better to avoid the complexity and security problems they bring if there are no benefits to be gained (Bagri). Many of the essential uses of pointers in C and C++ are handled by imported frameworks in Java, such as the linked list data structure.

For example, the linked list data structure in Java ("LinkedList in Java"):

```
import java.util.*;
public class exampleLinked
{
        public static void main(String args[])
        {
                LinkedList<String> list = newLinkedList<String>();
                list.add("1");                          // Printing the list would yield [1]
                list.addFirst("0");                     // Printing the list would yield [0, 1]
                list.addLast("2");                      // Printing the list would yield [0, 1, 2]
                list.remove("1");                       // Printing the list would yield [0, 2]
                list.remove(1);                         // Printing the list would yield [0]
                list.set(0, 9);                         // Printing the list would yield [9]
        }
}
```

In the above code, the creation of a linked list using the LinkedList framework element in Java. Various functions are shown to add, remove, and change the elements of the linked list, and it is also possible to iterate through the linked list using something like a for loop with an index variable. By using similar tactics for other pointer functions, Java helps make up for the lack of pointers in an arguably much simpler and easily understood way.

In many ways, the simplification of what pointers do into imported functions is a great choice that is the preference of many people, though there is something to be said about the flexibility that comes with developer control of pointers that comes with C++ and C.

While in the past, Java received many complaints relating to speed, the JIT (Just in Time) compiler has more or less resolved that issue.

Java lacks support for two prevalent features: unsigned integers and operator overloading. Without unsigned integer support, Java cannot properly be used in popular numerically heavy fields, and as a side note, it is much more of a task to convert data between languages. While conversions can be made to mitigate this issue, they are extremely memory inefficient to use and undesirable on a large scale ("Java libraries should…”). While less inhibiting than unsigned integers, a lack of operator overloading makes it more difficult to clean up and personalize code.

One of the biggest complaints about Java is the actual usefulness of the WORA principle. While the application itself is transferable, many features that are related to the particular JVM will cause problems, sometimes regardless of whether or not they are invoked. Working around this issue may require more work than would be worth using the portability features for (Wong). It is important to note that this is not always the case, and WORA more often than not cuts down development time, decreases costs, and increases flexibility.

In my opinion, Java’s greatest strength is WORA and its platform independence. Though Java is a far from perfect language and I do not particularly enjoy the completely OOP nature of it, the flexibility that Java provides, especially in networked environments that use many different systems, along with automatic memory management and garbage collection make it an enticing language to use where the speed and memory efficiency of C or C++ are not required.

6. Conclusion

Software Development is a diverse, varied field that uses many programming languages that offer different styles and benefits. In my report, I decided to describe and expand upon ALGOL and some of its popular derivative languages. Each programming language has its own strength and place to be used; such as C's low-level access, flexibility in memory management, and portability; C++'s efficient integration of OOP and RAII into C; Python's simplicity and visual clarity; and Java's WORA principle and JVM. This is not to say that preference does not have a place in Software Development. Developers are a diverse group of people with strong, differing opinions. For example, many developers prefer Python's simplicity whenever possible, and they will not utilize another language until absolutely necessary. On the other hand, many developers will accept the complexity of C and C++ and use them for every task imaginable because they are so versatile. In even more contrast to these developers, many developers use Java because of the lack of pointers and the easier recycling of functions. Throughout the course of my research, I realized that creating a programming language follows the same process that made me decide to pursue Software Development. Find a problem, devise a solution, and iterate to improve.

Bibliography

[1] Bagri, Megha. “C/C++ Pointers vs Java References”. GeeksforGeeks. August 8, 2017. https://www.geeksforgeeks.org/is-there-any-concept-of-pointers-in-java/. Accessed 21 Nov. 2020.

[2] Barland, Ian. “Why C and C++ are Awful Programming Languages”. Radford. April 29, 2015. https://www.radford.edu/ibarland/Manifestoes/whyC++isBad.shtml. Accessed 11 Nov. 2020.

[3] *C++ Vs Java: Top 30 Differences Between C++ And Java With Examples*. SoftwareTestingHelp. 13 Sept. 2020, www.softwaretestinghelp.com/cpp-vs-java/. Accessed 6 Nov. 2020.

[4] Chowdhury, M. “Some Awesome Modern C++ Features That Every Developer Should Know.” FreeCodeCamp.org, 9 Mar. 2020, www.freecodecamp.org/news/some-awesome-modern-c-features-that-every-developer-should-know-5e3bf6f79a3c/. Accessed 5 Nov. 2020.

[5] “Disadvantages of Python”. GeeksforGeeks. January 14, 2019. https://www.geeksforgeeks.org/disadvantages-of-python/#:~:text=Speed%3A%20Python%20is%20interpreted%20language,to%20C%2FC%2B%2B%20or%20Java.&text=Memory%20Consumption%3A%20For%20any%20memory,flexibility%20of%20the%20data%20types. Accessed 11 Nov. 2020.

[6] Ferguson, Andrew. *A History of Computer Programming Languages*. Brown University, 2000.

[7] Guevin, Jennifer. "Java co-creator James Gosling leaves Oracle". CNET. April 10, 2010. Accessed 18 Sept. 2020.

[8] "Is Python a good language for beginning programmers?". *General Python FAQ*. Python Software Foundation. Accessed 18 Sept. 2020.

[9] Jain, Anisha. "How Does Memory Allocation Work in Python (and Other Languages)?" Medium, Data Driven Investor, 7 May 2019, medium.com/datadriveninvestor/how-does-memory-allocation-work-in-python-and-other-languages-d2d8a9398543. Accessed 6 Nov. 2020.

[10] "Java libraries should provide support for unsigned integer arithmetic". BugDatabase, Sun Developer Network. Oracle. Accessed 11 Nov. 2020.

[11] Kuchling, A. M.; Zadka, Moshe. "What's New in Python 2.0". Python Software Foundation, October 16, 2000. Accessed 18 Sept. 2020.

[12] Kuhlman, Dave. "A Python Book: Beginning Python, Advanced Python, and Python Exercises". DaveKuhlman.org, 2013.

[13] Kumar, Krishan. "Why We Use C Programming Language?". CS-Fundamentals. Accessed 18 Sept. 2020.

[14] "Linked List in Java". GeeksforGeeks. June 7, 2020. https://www.geeksforgeeks.org/linked-list-in-java/. Accessed 21 Nov. 2020.

[15] Peters, Tim. "PEP 20 – The Zen of Python". *Python Enhancement Proposals*, Python Software Foundation, August 19, 2004. Accessed 18 Sept. 2020.

[16] Pillai, Premshree. "Introduction to Static and Dynamic Typing". Sitepoint, June 18, 2004. Accessed 18 Sept. 2020.

[17] "Pioneer Programmer Shaped the Evolution of Computers". Wall Street Journal, October 14, 2011. Accessed 18 Sept. 2020.

[18] "Programming Language Popularity". 2009. Archived from the original on January 16, 2009. Langpop. Accessed 18 Sept. 2020.

[19] “Python 2 vs Python 3: Key Differences.” Guru99, www.guru99.com/python-2-vs-python-3.html. Accessed 5 Nov. 2020.

[20] Ritchie, Dennis M. *The Development of the C Language*. ACM SIGPLAN Notices, 1993. doi:10.1145/155360.155580.

[21] Stroustrup, B. "Lecture:The essence of C++”. University of Edinburgh, May 6, 2014. Accessed 18 Sept. 2020.

[22] Stroustrup, B. "Stroustrup: Thoughts on C++17 - An Interview", April 30, 2015.

[23] Stroustrup, B. *The C++ Programming Language (Second ed.)*. 1991. Accessed 18 Sept. 2020.

[24] "The ALGOL Programming Language". Archived 6 October 2016 at the Wayback Machine, University of Michigan-Dearborn. Accessed 18 Sept. 2020.

[25] Veneers, Bill. *Inside the Java Virtual Machine*. Computing McGraw-Hill. 1998. Accessed 18 Sept. 2020.

[26] Verr, Marcel van der. “Format Syntax in ALGOL 68G”. Archived 2008-01-09 at the Wayback Machine, Algol 68 Genie. Accessed 3 Nov. 2020.

[27] “What Is C++ Used For?”. SoftwareTestingHelp, September 13, 2020. Accessed 18 Sept. 2020.

[28] "Why was Python created in the first place?". *General Python FAQ*. Python Software Foundation. Accessed 18 Sept. 2020.

[29] Wong, William. “Write Once, Debug Everywhere”. ElectronicDesign. May 27, 2002. https://web.archive.org/web/20090321180726/http://electronicdesign.com/Articles/Index.cfm?ArticleID=2255&pg=3. Accessed 11 Nov. 2020.

[30] Wood, Adam Michael. "ALGOL: The Best Language You’ve Never Heard Of". Who Is Hosting This?, https://www.whoishostingthis.com/resources/algol/. Accessed 18 Sept. 2020.

[31] Yegulalp, Serdar. “3 Major Python Shortcomings — and Their Solutions”. InfoWorld. August 7, 2019. https://www.infoworld.com/article/3429565/3-major-python-shortcomings-and-their-solutions.html. Accessed 11 Nov. 2020.