Tampereen yliopisto

Shayan Shajarian

# THE C++ PROGRAMMING LANGUAGE IN MODERN COMPUTER SCIENCE

Faculty of Information Technology and Communication Sciences (ITC)
Master of Science Thesis
April 2020

# ABSTRACT

Shayan Shajarian: The C++ programming language in modern computer science
Master of science thesis
Tampere University
Information Technology
April 2020

This thesis has studied the C++ programming language's usefulness in modern computer science both in suitability for developers and education by overviewing its history and main features and comparing it to its main alternatives. The research was mainly conducted with literature reviews and methods used for studying the subject where both quantitative in form of performance analysis and qualitative in the form of analysis of non-numeric attributes. This thesis has found that the C++ programming language is a very capable programming language for overall development, but the language's popularity has shifted towards system-level programming while the language is losing popularity for higher-level applications. The C++ programming language is also quite complex, making it too difficult to learn for beginners. Despite the complexity, the C++ programming language remains a very good language in terms of education for students of computer science because the language gives a good overview of programming as a whole.

Keywords: C++, computer science, software production, education of programming

# TIIVISTELMÄ

Shayan Shajarian: The C++ programming language in modern computer science
Diplomityö
Tampereen yliopisto
Ohjelmistotuotanto
Huhtikuu 2020

Tämä diplomityö on tutkinut C++-ohjelmointikielen hyödyllisyyttä sekä modernissa ohjelmistotuotannossa, että työkaluna tietotekniikan ja ohjelmoinnin opetukseen. Tutkimus suoritettiin tutkimalla C++-ohjelmointikielen historiaa, kielen yleisiä ominaisuuksia ja vertailemalla sitä muihin ohjelmointikieliin. Tutkimuksen menetelminä käytettiin sekä kvantitatiivisia menetelmiä kuten suorituskyvyn mittauksia, että kvalitatiivisia menetelmiä, kuten laadullisten ja ei-numeeristen attribuuttien analysointia. Tutkimuksessa C++-ohjelmointikieli on todettu hyvin kykeneväksi ohjelmointikieleksi yleiseen ohjelmistotuotantoon, mutta kielen suosio on siirtynyt korkeamman abstraktiotason tehtävistä alempaan tasoon. Opetuksen kannalta, tutkimuksessa todettiin, että C++-ohjelmointikieli on liian monimutkainen ja vaikea kieli aloittelijoille. Ohjelmoinnin opiskelijoille C++-ohjelmointikielen opiskelu on kuitenkin hyvin hyödyllistä, sillä kielen kattavuus antaa opiskelijalle hyvän kokonaiskuvan ohjelmoinnista.

Avainsanat: C++, tietotekniikka, ohjelmistotuotanto, ohjelmoinnin opiskelu

# PREFACE

Writing this thesis has been an interesting learning experience, required using many of the skills I had acquired over the years in the university, from writing in English to studying the field itself. The topic for the thesis was chosen out of personal interest of computer science as a whole and the C++ programming language serving as the “protagonist” for this journey. Although writing this thesis has given me a valuable insight into the field of computer science, it feels like learning more about this subject just makes one feel how expansive the field is and how little actual knowledge I have acquired thus far. Although this thesis may mark the end of my academic career, in many ways it feels like I am only beginning to learn.

I would like to thank my instructors for this thesis, professors Matti Rintala and Kari Systä. With special thanks to professor Kari Systä for helping me throughout the process and providing feedback and a valuable insight into many of the topics.

Tampere, 29.04.2020

Shayan Shajarian

## CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| C++ | A multi-paradigm, compiled, system-level programming language |
| Int | Integer |
| Lambda | Anonymous function |
| WORA | Write once run anywhere |
| COCOMO | Constructive cost model |
| IDE | Integrated development environment |
| API | Application programming interface |
| I/O | Input / Output |
| HTML | Hypertext markup language |
| SaaS | Software as a service |
| DevOps | Software development Information technology operations |
| SQL | Structured query language |
| Hard Choice | A choice made because of necessity |
| Soft Choice | A choice made because of preference |
| TIOBE | The importance of being earnest |
| Java | A multi-paradigm, platform independent programming language |
| JVM | Java virtual machine |
| Garbage collection | A form of automatic memory management system |
| Bytecode | A form of instruction set designed for efficient execution by a software interpreter |
| C# | A multi-paradigm, programming language developed by Microsoft. |
| CLR | Common language runtime |
| Python | A multi-paradigm interpreted programming language |
| Rust | A multi-paradigm system-level programming language |
| Waterfall | A sequential software development model |
| Agile | An iterative model for software development |
| STL | Standard template library |
| SCRUM | An agile process framework |

# 1. INTRODUCTION

The C++ programming language was developed in 1979 as a general-purpose programming language by a Danish computer scientist by the name of Bjarne Stroustrup. Since it was first introduced, the C++ programming language has grown to be one of the most popular, influential and widely adopted programming languages in the world. However, the field of computer science has changed and evolved both in paradigms of programming and advances made in computer hardware since the era that the C++ programming language was first introduced. With the current advances and paradigm shifts the question is, has the C++ programming language managed to keep up with the times or will it slowly fade away into obsolescence?

The C++ programming language is still a very prevalent and widely used language in software development, education and as an overall general-purpose programming language. In many cases however, C++ is seen by many as overly complex, difficult to learn and prone to errors. In addition to many of the C++ programming languages issues, there are a variety of other programming languages that are perhaps better, or at the least more suited to what current developers are looking for in a programming language. As a response, there are many new and more current updates to the C++ programming language which have made strides to modernize the language and keep it as current as possible.

This thesis examines the C++ programming language in modern computer science for the purposes of determining how useful it is in modern software development, how effective it is for education and will the language remain relevant in the future or be slowly relegated in to specific niche roles instead of remaining as a general-purpose programming language. In addition, this thesis attempts to review modern software development and the place of the C++ programming language within it for the purposes of comparing it to its main alternatives and suggesting development ideas for the C++ programming language. The methods used for studying these questions are analysis of the C++ programming language's development history and paradigm shifts as well as studying software development and education. Comparisons and reviews are done by both quantitative methods whenever possible as well as qualitative methods to study and analyse the quality properties of any given attributes.

This thesis is structured by first overviewing the theoretical background of the subject. This is done in the second chapter of this thesis where overall features, programming paradigms and history of the C++ programming language are overviewed. The theoretical background also overviews a brief history of software development and discusses the education of programming. The third chapter is dedicated to explaining the methodologies used in this thesis for conducting the study and the analysis done. The fourth chapter compiles the results found in this thesis. This is done by first briefly overviewing specifically chosen programming languages and comparing them to C++ and studying on how the C++ programming language is suited for modern software development and how well it is suited as a tool  for the education of programming for both general engineers and computer science students. The fifth chapter of this thesis is dedicated for analysing the results found for the sixth chapter where conclusions are drawn from both the study and the analysis.

# 2. THEORETICAL BACKGROUND

The purpose of this chapter is to define the area of the research and familiarize the reader with the subject. The chapter gives a brief overview of the background and basic features, attributes and programming paradigms of the C++ programming language. Some of these attributes and features overviewed in the chapter are not inherent to the C++ programming language, but overall concepts in computer science. In addition, the chapter briefly overviews the evolution of software development and discusses effective ideas for education of programming for university students.

## 2.1 The C++ programming language

To describe the C++ programming language, it is crucial to understand what programming and perhaps more importantly, what programming languages are. In essence, programming is speaking to the computer by means of giving the computer a series of commands to execute. [10] Computers are only capable of understanding very rudimentary instructions. To bridge the gap between what both a computer processor could interpret and what a human could understand, the use of programming languages is necessary. This process can be achieved by turning the user written source code into machine code for the computer to execute by different means. [11]

To more effectively breach this language barrier and to more accurately give instructions for a computer on what to do, the development of different programming languages is imperative. Programming languages themselves can be categorized in a number of different ways by both their purpose, their operation and functional design decisions. One of such attributes is the abstraction level of the programming language.

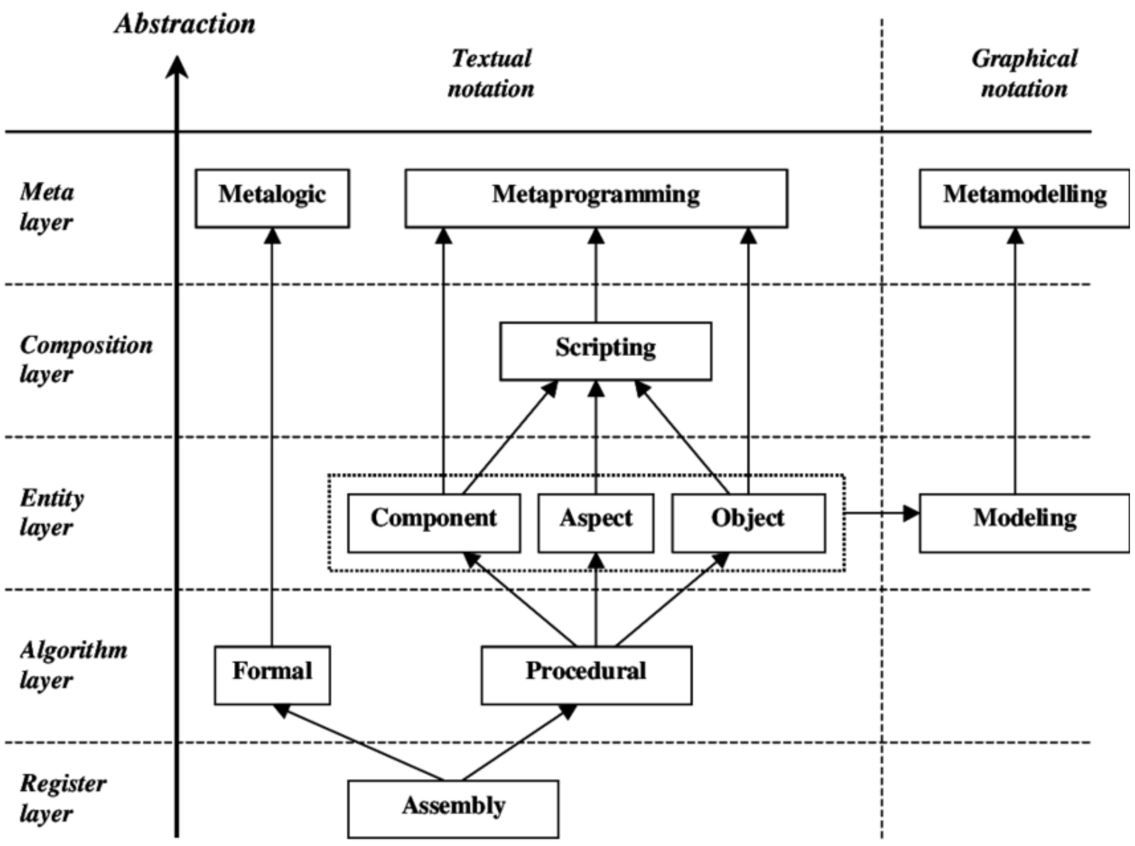


**Figure 1.** Layers of abstraction in software programming languages [12]

An abstraction level can be described as the level of detail of a software system by means of hiding “irrelevant” details from the user. In other words, the lower the abstraction layer is, the closer the user is to the computer’s hardware and may give more direct instructions to the computer. Figure 1 features different layers of abstraction as a visual aid. The level of abstraction in practical terms is how expressive a programming language is. The increase of the layer of abstraction allows the user to address more complex problems with fewer lines of code and in theory, fewer programming errors. While a programming language with a lower level of abstraction will grant the user more direct access to the computer’s hardware and allow the user more control. Programming languages can be classified in many different layers of abstraction. These layers can be described in hierarchical-levels from lower levels of abstraction which require more and more direct commands to the computer such as registers and memory addresses, to higher levels of abstraction and even meta-programming, where details of the execution are less visible to the user. These layers of abstraction can be distinguished from each other from lower- to higher-levels. The lowest level is the binary code for the processor. The next level higher from binary code is assembly language or the register layer, where operations are described on the processor. The step upward in abstraction can be classified as the algorithm layer where programming languages such as C, pascal or LISP operate. These languages are often procedural and can hide lower level details from the user such as registers but instead operate on a higher level and focus on the description of software algorithms to handle data. The next layer of abstraction is the entity layer in which many popular programming languages such as C++ or Java operate. In this abstraction layer, higher level operations such as classes and objects can be utilized, and the overall focus is on the description of domain entities. The next layer is the composition layer or scripting layer, where programming languages such as Python operate. In this level of abstraction, the focus is on writing existing software artefacts into larger systems. The main concern of programming in this level shifts from data or algorithms to software components, where the complexity of the programming language is reduced, and ever larger sections of the code is abstracted away. This in practical terms leads to decrease of lines of code to the user. The last and highest level of abstraction is the meta layer, where metamodeling or metaprogramming languages are used to describe components and creating code generators. These higher-level languages are based on higher-level programming such as metaprogramming or generative programming. [12]

With all this in mind, the C++ programming language can be described by the level of abstraction to be higher than that of assembly languages or the C programming language but not quite as high as some more modern programming languages. This in practical terms means that the C++ programming language can be utilized to manipulate data and memory directly, but the language also has the benefit of operating on higher levels of abstraction with abilities such as object orientation and classes. However, the C++ programming language does not operate quite on the same level of abstraction as many newer programming languages. This in practical terms means that the code written in C++ requires the user to describe more details than in that written in a language which has a higher level of abstraction and as a result a program written in C++ can be longer in length and more prone to user made errors while also giving the user more direct control.

### 2.1.1 C++ Compilation

Another defining feature for a programming language is how the user written code is turned into the form in which the computer can understand it. This process of turning the code of a higher-level programming language into the form which can be executed often is a defining feature and a clear distinction of certain programming languages. In the interest of brevity, the two main distinction discussed in this thesis are the differences between compiled languages and interpreted languages.

The C++ programming language is a “compiled” programming language. This in short, means that the user written source code is directly turned into operating system specific and computer processor specific machine code. [13] This process is done by taking each C++ source file and compiling it into an object file. The result of this compiling creates object files which are then linked together to create either an executable program or a library. [14]

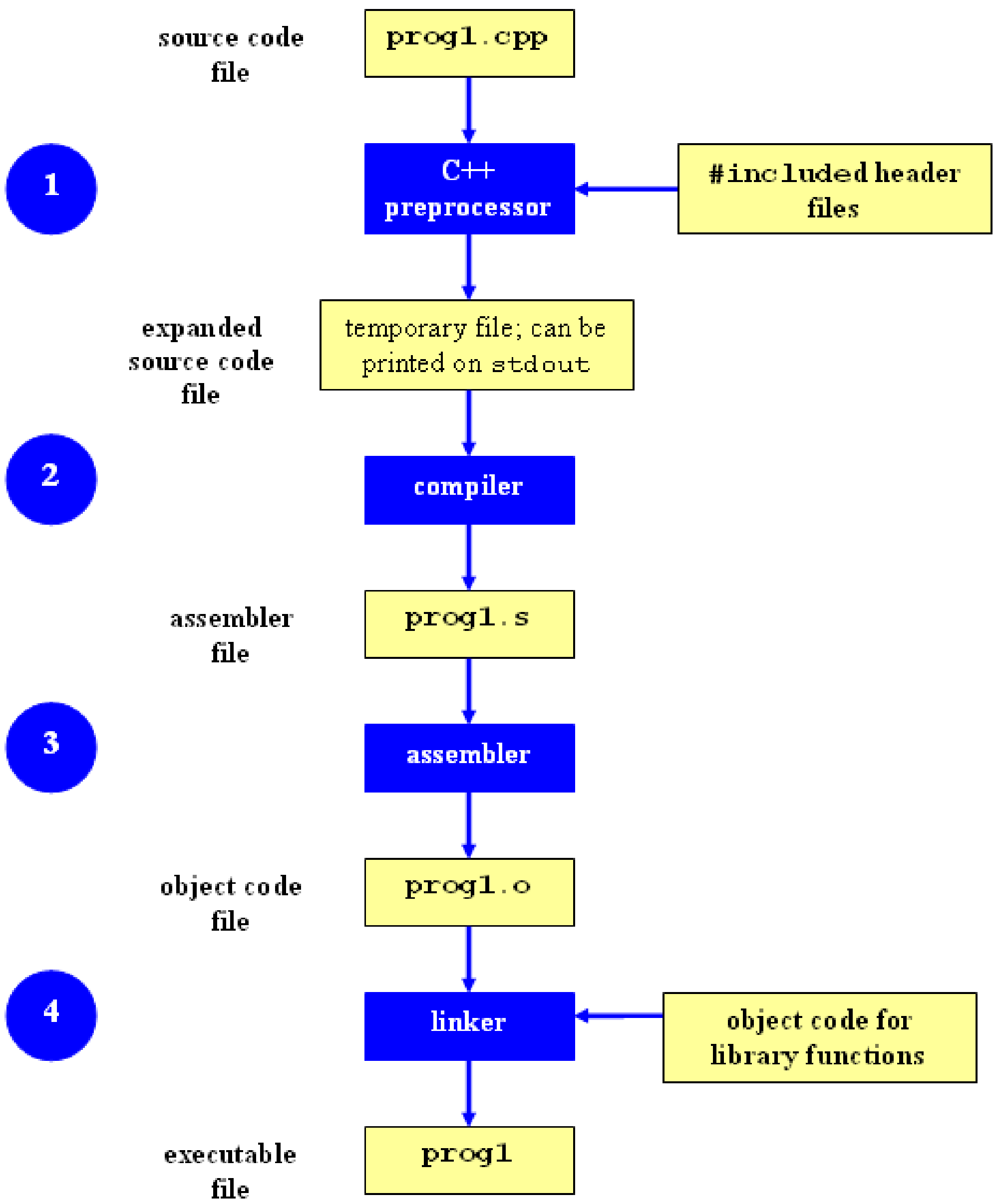


**Figure 2.** The C++ compilation process [15]

In more detail, the process of turning the user written code into what the computer is capable of executing is done in four distinct stages which are shown in figure 2. These stages are pre-processing, compiling, assembling and linking. In the first stage, or pre-processing stage the pre-processor directives are handled. This in practical terms in the C++ programming language means that any directive with the “#” sign such as #include or #define are handled. In the first stage or pre-processing stage, the user written source files are run through a special text processing program called a pre-processor. The pre-processor will then perform one or more of the following actions. Including the files containing the #include definition by searching for the specified identifier and including them into the source file. The conversion of values which are specified by the #define statements into constants and the conversion of the macro definitions into the corresponding code. Conditionally including or excluding parts of the code which with the #if, #elif or #endif directives. The output of this pre-processor is the C++ code in its final form which will then be passed to the next stage. [16] In the pre-processing the source code is temporarily expanded to prepare for the compilation stage and the file has a greater number of lines than the original source code. The pre-processor can also add markers to the code to aid the compiler for producing error messages for the user.

The next stage is compilation. In this stage, the compiler takes every output from the pre-processor and constructs an object file. This process is done by converting the pure C++ code, meaning the code without the pre-processor statements, into assembly code. During this process, the compiler can also optimize the source code and can point out failed overload resolution errors, syntactical errors or other compiler errors at this stage. The third stage is assembly. In this step of the process, the assembler goes over the assembly code and line by line converts it into bit code. The output of this process is a binary file typically in formats such as COFF.

The final stage of the process is linking. In this step, the linker produces either a dynamic library, a shared library or an executable file. Static libraries are created when object files are archived. These object files can then be used without the need to recompile them. Dynamic libraries are modules which contain data and functions which can then be used by another application. [17] In the linking process, the collection of object files created in the compiling stage are linked together by forming the resulting program memory map section out of individual contributions. In addition, the linker must resolve all the references. [16] In short, the linking process results in the creation of a single executable file from multiple object files. Once this process is complete, the executable file is created and the program can be run.

### 2.1.2 C++ Type checking

Another defining feature of a programming language is the type system that it uses. A type system is the component in a programming language that associates each expression of the language with a data type. These expressions are things such as variables, functions and operations just to name a few. A type system is a component of programming languages which associates each expression of the language with a data type. This component itself consists of a collection of types and type constructors which coupled with type rules determine the consistency of an expression with a given type. A common error in many programming languages is a type error. This error occurs when a violation of a type rule is detected, often by a program that specifically tests for type errors. This is known as type checking. [18]

In more practical terms, a type system refers to the rules that different variables in programming languages must adhere to. These type systems can also be more specifically defined in different attributes which more accurately describe them. These attributes can be described in the following ways. A type strength, expression, checking and safety. The type strength is often described as either weak or strong, the latter referring to a system which places restrictions on how different types of variables can be converted to each other without any converting statements. A weak typing system on the other hand, would attempt to find another way to make the type cast function. [19] In other words, a programming language has strong typing if a variable cannot be used in a way inappropriate for its type. As an example, attempting to add and integer value to a string in certain strongly typed languages would lead to a type error, while in a programming language with weak typing, this may both compile and execute. [90]

The next type system attribute is the type expression, which is an attribute which deals with how the interpreter or compiler of a programming language infers the types of variables. Type expressions can be either manifest of inferred. A manifest typing system requires the variable types to be explicitly defined whereas an inferred typing system will infer the type of the variables used based on context in which they are used in. The next attribute is the type checking which can be either static or dynamic. [19]

Statically typed programming languages have their types checked by either the compiler or interpreted before the program is ran or compiled. In a dynamically checked language, the type checking is done during the runtime of the program. The final attribute is the type safety which refers to the degree to which the programming language will prohibit operations on typed variables which may lead to either errors or undefined behaviour. A type safety can be either safe or unsafe and the differences between the two can be described by the degree in which the programming language will interfere with the type safety. An unsafe language relies on the user to handle the type safety while languages with safe types will do more to ensure that type conversions or problematic operations do not occur. [19] In other words, in a type safe language attempts to misinterpret data is caught at compile time or a well-specified error is generated at runtime. [91] A non-type-safe language can fail to catch this issue due to the search mechanism lacking the strict type information which would often be available for the compiler. This is how a type-safe and strong typed languages differ, because a strongly typed language type rules are automatically enforced by the compiler. [92]

The C++ programming language is strongly and statically typed, meaning that every object or variable has a type and that type will never change. In practical terms, when a variable is declared, the type of that variable must be explicitly specified to instruct the compiler. For instance, if a user wants to store the value of an integer, the type of this specific variable must be defined as “int” and the type of this specified variable cannot be changed later. Meaning that if a user stores “int x = 5” the value of x cannot then be changed to a value other than another integer. In addition, whenever a function is declared, the type of each argument and return value must be explicitly specified or void if there is no value to be returned. After a variable is declared, the type of that variable cannot be changed later. Type conversions are however possible in the C++ programming language, in the form of copying a variable’s value or a function’s return value into another variable of a different type.

The C++ programming language does not have a universal base type from which all other types are derived. Instead, the C++ programming language features a variety of types one of which are the fundamental or built-in types. These fundamental types include the very basic types featured in the programming language and include types varying byte sizes such as int, double, long bool and char. Many of the fundamental types also include an unsigned version, which modifies the range of the value of that variable. As an example of this, a typical integer or int can store values from -2 147 483 648 to 2 147 483 648, whereas an unsigned int can store values from 0 to 4 294 967 295. The size of both integer and unsigned integer are 32-bit. [20]

In addition to the fundamental types, the C++ programming language features other types such as the void type. The void type is unusual because a type cannot be declared as void, however a pointer to void can be declared as a variable. This feature can be used to allocate raw or un-typed memory. The use of this feature is however discouraged in modern C++ due to it not being type-safe. A more common and acceptable use of the void is in function declaration where a void return value is assigned to a function which does not return a value. [20]

Other types include in the C++ programming language include types such as the string types. The commonly used string is a standardized library type found with the inclusion of string (#include <string>) in the form of std::string. This 8-bit string is char-type character of strings.

The C++ programming language also has included user-defined types. These types are constructed whenever a class, struct, union or enumeration (enum) is defined. These user-defined types are treated by the language much in the same way as fundamental types. They have a known size in memory, and they adhere to certain rules for compile-time checking and the runtime of the program. There are two main differences between fundamental- and user-defined types. The first is function overloading. And the second is that the compiler has no built-in knowledge of a user-defined type and the compiler must learn the type when the definition is first encountered in the compilation process.

The C++ programming language also features the pointer type. The pointer type allows the user to declare a variable of a pointer by using the asterisk (*). Instead of a typical variable, the pointer type stores the memory address where the actual data value is stored. The pointer type is denoted by an asterisk and is also known as a raw pointer. This pointer is accessed through the special operators which are the asterisk "*" or "->". This operation is known as dereferencing. The use of raw pointer is however discouraged in modern C++ because the memory allocated is not automatically deallocated and can lead to memory leaks. For this reason, the use of smart pointers is encouraged. Smart pointers are another pointer type, featured in the C++ programming language. Smart pointers are used to assist in ensuring that programs are free of memory and resource leaks and are exception- safe. This is done by them deallocating the memory automatically when its destructor is invoked.

The types used in the C++ programming language can also be qualified by the const type qualifier. This feature allows user-defined types or functions to be qualified and even overloaded with the const keyword. The purpose of this feature is to ensure that the value of a const type cannot be modified after it is initialized. [20]

### 2.1.3 The Auto keyword

With the addition of newer standards, the C++ programming language has gained certain abilities which are more customary for dynamically typed programming languages. One of these features is the "auto" keyword which was introduced in the C++11 standard. Prior to the introduction of the auto keyword, the user had to explicitly specify the type of variable used or initialized. Since the auto keyword was introduced, the type of variable or even the type of function since C++14, can be assigned as auto. The auto keyword is then used to tell the compiler to deduce the actual type of variable being declared or initialized. [28] The auto keyword can in certain cases give the C++ programming language the luxuries of dynamically typed programming languages. It must however be noted, that even with the auto keyword, the C++ programming language is still statically typed, and with the use of the auto keyword the type of the variable is deduced at compile time in a process known as type inference. [29]

### 2.1.4 Memory management in C++

The basic memory architecture used in any C++ program is divided into four different segments. These segments which are showcased in figure 3. These segments are the heap, stack, data segment and code segment. The code segment houses the compiled program with executive instructions. This segment is read only, a fixed size and is kept under the heap or stack segments to avoid overriding them as heap or stack segments can grow or shrink in size. In more practical terms, the code segment contains the code that needs to be executed by the processor.

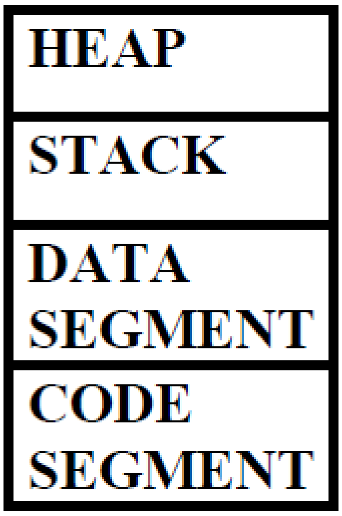


**Figure 3**. Memory architecture of a C++ program [47]

The data segment houses the global and static variables. The data segment can both read from and written into. The stack segment is typically pre-allocated memory space. The stack is also structured in the “last in first out” or LIFO type of data structure where each new variable is pushed into the stack and they are the first to be handled. The stack’s memory management functions by the scope of the program. Once a variable goes out of the scope of the program’s execution, the memory then becomes available for allocation again. In practical terms, in C++ code if a local (not created with the new-keyword or a pointer) variable is declared within a scope, for instance a function, once the execution of that function is completed, the space for that variable may be reallocated and used for something else. The stack can grow and shrink as functions push and pop local variables from it. The stack houses local data, return addresses, arguments passed to functions and current status of the memory. The heap segment is memory allocated during the execution of the program. This space in memory is allocated by using the new-keyword and the deallocation of memory must be done manually by the user. The deletion is handled by using the delete operator. [47]

### 2.1.5 Memory exploits in C++

The C++ programming language allows the user to access the memory but the language itself does very little in preventing the possibilities for problems or exploits. Therefore, the user is responsible for memory management and preventing vulnerabilities. Due to this direct access to memory the C++ programming language provides and the language's lack of certain features such as checking if memory indexes are exceeded, there are certain possibilities for exploits. One of the more common exploits is the buffer overflow. The buffer overflow exploits in the case of the C++ programming language can be broadly categorized in two distinct categories which are named after the region in memory where these exploits can occur. These two buffer exploits are the stack-based buffer exploit and the heap-based buffer overflow. It is also important to note, that these exploits may also occur in the data segment.

The stack-based buffer overflow occurs, when memory is allocated on the stack for an array and the data is written beyond what has been reserved for a specific memory space to an adjacent memory space. The adjacent memory space can contain certain crucial pieces of information or instruction for example the address to resume execution. This address can be overwritten by an attacker and the execution of the program can be derailed as a result. Note that this type of buffer overflow may also occur in other memory spaces.

The heap-based buffer overflow is quite similar to the stack-based vulnerability. Heap memory is allocated dynamically at run-time by the application. This exploit also relies on overwriting memory to gain access to the execution flow of the program. This is done by overwriting the heap-stored function pointers which are located after the buffer which is being overflown. Another way is to overwrite the heap-allocated object's virtual function pointer and pointing it to an attack-generated virtual function table, which upon execution of one of these virtual methods, will execute the code to which the attacker-controlled pointer refers.

Another memory exploit which can occur in the C++ programming language is the dangling pointers vulnerability. This vulnerability occurs when a pointer to a memory location is deallocated and the dereferencing of this pointer is unchecked. This space in memory can then be overflown to overwrite other pointers. [48]

## 2.2 History and development

The history of the C++ programming language begins in 1979 with a Danish computer scientist by the name of Bjarne Stroustrup [1]. The original idea of C++ was to provide the user with the efficiency and flexibility of the C programming language with the facilities of program organization of the Stimula programming language. The concept of the class structures found in Stimula where an essential part for what came to be the precursor of the C++ programming language known as “C with classes”. [2] In the early iterations of the C++ programming language, the object-oriented programming paradigms that came with adding classes was a major differentiating factor between the C and C++ programming languages.

The work on the original “C with classes” programming language began in 1979 and the initial set of features added where classes and derived classes and with them the control of public and private access to classes, constructors and destructors and finally function declarations with argument checking. These features added gave the language the object-oriented programming paradigms. In addition, the first library added to the language had support for non-pre-emptive concurrent tasks and random number generators. [2]

In 1983 the nomenclature C++ was adopted, renaming the “C with classes” programming language. In addition to the new name, the language received many new additions such as virtual functions, perform overloading, single-line comments and references with the const keyword. [3] It wasn’t until the year 1985 that C++ was implemented as an official commercial product but even still, the C++ was not officially standardized. During this year however, the language gained many new features such as in-lining, function and operator overloading, references and virtual functions. The virtual functions granted the language support for run-time polymorphism. [2] Through the late 1980s and 1990s the C++ programming language saw many new updates and developments. Most notably in 1989 the language was updated to include protected and static members and inheritance from several classes. In the year 1998, the C++ standards committee published the first international standard for the C++ programming language which is informally known as C++98. [4] This nomenclature has remained as a standard for newer standards of the C++ programming language where the name of the language is proceeded with the year of the adopted standard.

The subsequent additions or improvements to the C++ programming language where usually added with new standards to the language with the first ISO standardized version being the C++98. The next official standard named C++03 was adopted in 2003 and most notably with it, in addition to many bug fixes and improvements to the previous standard, came a feature for introduction of value initialization. [3]

The subsequent standard, named C++11 and adopted in the year 2011, was a major leap forward for the C++ programming language. This new standard fundamentally changed the way the language could be used. This was done in attempt to make the C++ programming language a more easily accessible and teachable language as well as a better language for system programming and library building. [5] The features added with the C++11 standard are numerous but some of the more important additions where nullptr, range-based for loops, override and final, strongly-typed enumeration, smart pointers and perhaps in practical terms, arguably the two most notable changes which where the auto keyword and lambdas. C++11 was the first standard to introduce the paradigms of functional programming to the C++ programming language in the form of lambda expressions. Lambdas or anonymous functions are commonly found in functional programming languages and with their addition, the C++ programming language gained the ability to use higher order functions. The auto keyword is also a very notable change. The auto keyword allows the programmer to not specify the type of a variable, but instead let the compiler deduce it. [6]

The changes after C++11 have been more subdued and more frequent. The next standard was released only three years later in the form of C++14. The new standard did not alter the paradigms of programming or implement any noticeable practical change, but instead the new standard can be seen as another evolutionary step for the language. The new standard even removed some features, such as gets() which was an unsafe input or output function from the C library and discouraged the use of rand() in lieu of the <random> library. The new features to the C++ programming language added with the C++14 standard are quite numerous. Some examples of the more notable changes are to the functional programming capabilities of the language. Added features are polymorphic lambdas which allows lambda expression to define function templates and generalized lambda captures which allows lambda captures to be initialized. In addition, the C++14 standard also had many other additions, such as a shared mutex, quoted strings, digit separators, variable templates and user-defined literals among many other additions. [7]

At the time of writing this thesis, the most current officially released standard for the C++ programming language is the C++17 standard released in 2017. This standard also removed or deprecated certain features of the language for instance the register keyword, trigraphs and certain types of dynamic exception specifications among many other features. The language also received four globally applicable new features to the core of the language, note that these features may happen to the user without their consent or even knowledge. These four new features are as follows. Exception specification as part of the type system.

Guaranteeing compiler optimization techniques such as copy elision which eliminate unnecessary copying of objects. The problems that this feature would solve is that without guaranteed copy elision, elision might not occur meaning that you cannot get rid of the move and copy constructors.[93] Guaranteed copy elision also allows the user to define functions with return types with guaranteed copy elision even for types that are not copyable or movable. Dynamic allocation of over aligned types and finally stricter order of expression evaluation which evaluates function arguments in an indeterminate order. In addition to the global features, the C++17 standard also includes new core language features with local applicability. These new features include fold expressions, which are for applying a binary operator iteratively to parameter packs, inline variables and among many other features, additions to the language's functional capabilities in the form of better features for lambdas. In addition to the languages core features, the C++17 standard also includes new library features. These features include new algorithms, features to parallelism and filesystems. [8]

The most current proposed standard for the C++ programming language will be the C++20 standard which as of writing this thesis, has not yet been adopted. Most of the features are however now either decided or being decided. In a talk with the C++ programming languages creator Bjarne Stroustrup described the C++20 standard as the best approximation of the C++ programming languages ideals with many useful new features such as modules, coroutines and better compile-time programming support among many other features. [9] For the sake of brevity and uncertainty at the time of writing this thesis, these new features will not be further discussed or showcased due to the fact that they may be subjected to change or modification.

## 2.3 Overview of programming paradigms

To better understand a specific programming language and what design decisions govern the choices for the language's different attributes, the understanding of the underlying programming paradigms is necessary. The definition of a programming paradigm is often considered a very vague and broad term but in essence, a programming paradigm is the specific style or general approach to writing code. [21] The term programming paradigm can be more accurately described as a model of programming based on a distinct set of concepts which shape the way a program is designed and organized. To gain a more comprehensive knowledge about programming paradigms for both the idea itself and to better understand programming languages, a few different programming paradigms and their attributes are listed.

### 2.3.1 Imperative paradigm

In an imperative programming paradigm functions are implicitly programmed in every step needed for solving a problem. This paradigm uses statements which alter the program's state for a certain goal to be achieved. In other words, every operation is programmed and the program itself specifies the solution for the problem. This also means that pre-programmed models are not called upon, meaning that every operation is coded and the code itself specifies how the problems is solved. In imperative programming, instead of relying on models to solve a problem, it requires the understanding of the necessary problem-solving functions to solve the problem. This means the focus of imperative programming is "how" a problem should be solved. This requires a step by step guide for the computer to follow, because the program performs functions instead of models. [22]

### 2.3.2 Procedural paradigm

A procedural programming paradigm is a subset of imperative programming and is an example of an early approach to programming which was utilized in the beginning of computing. In procedural programming, the program is comprised from a set of instruction for the computer to execute directly in the order in which they appear, unless otherwise instructed. Things which can change the control flow of the program are known as flow control structures, for instance if statements and for and while loops. The main difference between procedural- and imperative- programming paradigms is that in imperative programming commands are executed in the order they are given. Procedural programming, being a subset of imperative programming, does this as well but allows splitting these instructions into procedures or functions. Many of the early programming languages are based on the procedural programming paradigm. Programming languages such as Assembly, FORTRAN and C are an example of procedural languages. Code written with the approach of procedural programming paradigms are often easy to read and simple, when the program is short, however large programs written in this approach can be difficult to read, manage or debug. [23]

### 2.3.3 Object-oriented paradigm

An object-oriented programming paradigm is a model which is structured around objects rather than actions and the focus is on data rather than logic. In object-oriented programming, objects are regarded as separate entities with their own state which is altered only by specific built in procedures which are known as “methods”. Methods are in many ways just like functions, but instead of them using values of given parameters, they function on objects. Objects operate independently so they are encapsulated in modules which is why they contain both local environments and methods. Communicating with object is done by message passing. Structuring for objects is done by ordering them into classes, from where they can inherit equivalent variables and methods. One of the many benefits of the object-oriented programming paradigm is that code can be reused. This can be done by deriving a new class or a sub class from existing classes by a system known as inheritance. In this system, the derived class inherits all the features of the base class. [24]

### 2.3.4 Functional programming paradigm

The paradigm of functional programming is very different than the ones of procedural or imperative programming. In functional programming, functions are treated as first class objects. In the paradigm of functional programming, all subprograms are viewed as functions or to be more precise, mathematical functions. In this paradigm, arguments are taken, and a distinct solution is returned, just like in mathematical functions. The solution returned is informed only by the input given and is not affected by the time at which the function is called upon. The computational model of functional programming is one of function application and reduction. The paradigm of functional programming can give a programming language some distinct advantages. The first advantage is the higher level of abstraction. This allows the user to let the computer worry about many levels of details of programming and remove the opportunities of many classes of errors. In addition, the higher level of abstraction can also make the written code of the program significantly shorter. Other benefits of functional programming include attributes such as deficiency of dependence on assignment operations. This allows programs to be valued in many different orders which is a very useful attribute in parallel computing. In addition, the lack of assignment operations aids in the responsiveness for mathematical proof and analysis. Functional programming can also have certain disadvantages, mainly the complexity of the paradigm itself. This complexity and the deviation of norms can be a point of contention for many who are used to the more conventional paradigms of computing. In addition, functional programming can be less efficient in certain applications. [24]

### 2.3.5 The programming paradigms of the C++ language

The creator of the C++ programming language Bjarne Stroustrup describes the C++programming language as having four main features that are most directly supported. These "features" as he describes them but referred to as "programming paradigms" in this thesis, are procedural programming, data abstraction, object-oriented programming and generic programming. The idea for the paradigms of the C++ programming language, as described by the creator, is to facilitate an elegant combination of different programming styles and a support for a wide variety of programming techniques.

The support for procedural programming paradigms in C++ is focused on processing and designing suitable data structures. This paradigm in many ways, gives C++ the overall structure of the language, in terms of the way the instructions are given, the control flow is changed with loops and the functions that are used. This feature manifests itself in the C++ programming language in the form of built-in types, operators, statements functions, structs, unions and other such features.

The Data abstraction in C++ is focused on the design of interfaces. In general, this means hiding certain implementation details. The C++ programming language also supports both concrete and abstract classes for which there is an ability for class definition with private implementation details and both class constructors and destructors with the direct support of associated operations. In addition, there is direct support for abstract classes which have the ability for providing complete data hiding.

The paradigms of object-oriented programming in C++ are focused on designing, implementing and using class hierarchies. Class hierarchies can also provide run-time polymorphism and encapsulation of code.

Generic programming focuses on the design, implementation and the use of general algorithms. The word general in this context refers to the type of algorithm which can be designed to accept a variety of types as long as they meet the requirements dictated by the algorithm's arguments. In the C++ programming language, templates are the main examples of generic programming. Templates can provide compile-time parametric polymorphism. [25]

Since the C++11 standard, the C++ programming language has also adopted the paradigms of functional programming. The most concrete example of functional programming in the C++ programming language is the addition of lambda expressions. Lambda's are a convenient way of defining anonymous function objects also known as closures, precisely at the location where it is invoked or passed as an argument to a function. [26] This gives the C++ programming language a method to use higher order functions. Higher order functions in this context are referred to as a pervasive abstraction which encapsulate common programming patterns by calling other functions provided as input arguments [27]. This means that functions have the ability to take in other functions as parameters and also return functions as return values.

## 2.4 Software development

This section briefly overviews the history of software development and how it has changed and evolved over the years. In addition, this section goes over the productivity measurements and ideas used in software development.

### 2.4.1 A Brief history of software development

The world's very first piece of software was written by a computer scientist by the name of Tom Kilburn in the year 1948. This software was programmed to perform mathematical calculations by using machine code instructions. This first program took 52 minutes to compute the greatest divisor of 2 to the power of 18 (262 144). Computers where programmed with punch cards in which the holes denoted specific instructions in machine code. The word "software" was first used in the year 1958, one year after Fortran, one of the first higher level programming language was created. [30] In the following decades, the following programming languages and other milestones where reached in computer science. In 1962 Cobol, 1973 C and the Unix operating system, 1985 C++, 1990 Haskell, 1991 Python, 1995 Java and 2000 C#. Many of the listed programming languages where quite influential for computer science. [31]

For the interest of brevity, it is perhaps best to describe the evolution of software development in terms of different eras. In the beginning, computers and software development was mainly done by and for governments or large organizations purely as means of crunching numbers more quickly and reliably. This first era encapsulates everything from the early mechanical computing machines to more sophisticated and more modern computers capable of running software.

This era is then followed by the era of personal computing, during which the price of computers was significantly reduced to the point where they became commercially available products for not just governments and companies, but individuals as well. This Era started in the 1970s and has continued onward and is exemplified by the Unix operating system being developed in 1973 and the release of the first interactive PC, the Xerox PARC in 1975. [31] It is important to note, that during these times, computers where individual and independent systems and they did not interact with other computers and the software development of the times obviously reflected this. In addition, the performance of these machines was nothing compared to modern computers, meaning that the software driving them had to be as efficient and as fast as possible.

The 1990s onward marks the era of the internet or at least when the internet gained more mainstream popularity and adoption. During this time, computers where no longer just separate machines, but end-devices which where a part of an interconnected network. The rise of the internet also led into the demand for international information display and e-mails on the World Wide Web. Programs during this new era were required to handle illustrations, maps, photographs and other images in addition to simple animations. The growth of browser usage and the HTML language which it was running on changed the way information was displayed, retrieved and organized. These new avenues for using computers drastically changed and added to the way software was developed.

From the year 2000 onwards, the trends and ideas of computing have once again changed with lightweight methodologies and agile development becoming more popular and standardized in the software development industry. This new era was driven by the expanding demand for software in many smaller organizations which required inexpensive software solutions, and this led to the development of simpler and faster development methodologies. In addition, the use of rapid-prototyping, evolved into lightweight methodologies which attempted to simplify many areas of software engineering which include things such as requirement gathering and reliability testing. [33]

Modern day software development has further changed with new ideas to suit modern models of business. Many more modern pieces of software have changed from software bought and ran on a machine, to things such as cloud computing and software as a service rather than a product. This is a drastic shift both in terms of business models and software produced. Cloud computing means delivering different services such as data storage, tools or applications through the internet rather than keeping the data or programs on a device’s local storage. [76] Cloud computing is an example of this new software distribution model known as SaaS or software as a service. The software as a service model typically functions by hosting applications and making them available to customers over the internet. This model allows the service to be updated automatically with flexible payments and the ability to scale the demanded service quite easily. [77] In addition to these more recent models of computing and business models, the development ideas of software have also evolved. DevOps is one example of these newer ideas of development. The idea of DevOps is to apply agile approaches to operations work by collaboration of the staff of development and operations. This model has emerged largely as a response to service-oriented software and is meant to add to traditional agile methods. [78]

Programming languages such as C, Python and Java give an understanding of the era and requirements of the era that they were developed for. And through this, the understanding of why and how certain programming languages gained popularity can be better understood. This thesis focuses mainly on the more mainstream general-purpose programming languages and not more purpose specific languages such as Javascript.

The C programming language, which was created between 1967 and 1973, is still very popular and used quite extensively even to this day. The C++ programming language was in essence an extension of the C programming language. C++ was however in many ways a next-generation language with features such as object-oriented programming. These features where at the time considered phenomenal when compared to the more structural C programming language.

Some of the more recent programming languages popularized certain features. Java is one of such languages. Java is one of the most popular and successful programming languages of many decades. The Java programming language was created with a mission WORA or “Write Once Run Anywhere”. This in practical terms means that Java was designed to be a programming language with the ability to be ran on any platform. Or to be more precise, the same code can be run on any platform.

With faster hardware and with cases where the need to access the computer’s hardware became less frequent, more expressive languages also became viable. One of such languages is the now very popular Python programming language. Python was designed to be an easy to read and learn with fewer unnecessary symbols and very clear English names in its syntax. Python also has a very high emphasis on productivity, meaning the language is meant to produce more functionality with less development time. [94] The main goals of the Python programming language is therefore to be a higher-level general-purpose programming language with an emphasis on code readability in the syntax. Python has gained quite a lot of popularity because of its simplicity and is used extensively in web application development. [32]

With the development history of software and the programming languages emerging during the different eras, the requirements for a programming language in any given time can be better understood. From the early sequential logic required for basic computation to the modern agile software developed for mobile devices.

### 2.4.2 Productivity in software development

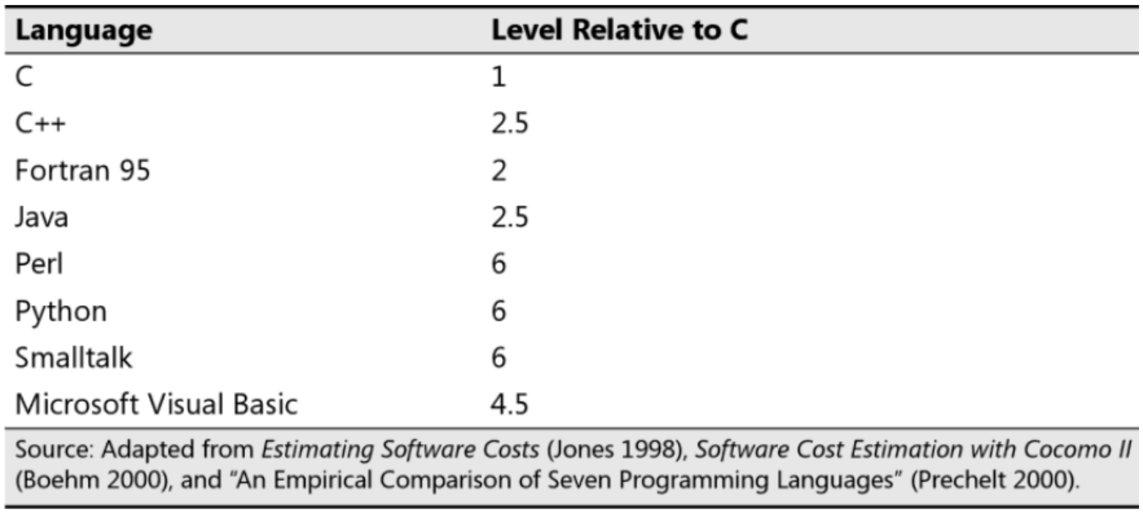
Software has become an integral part of the infrastructure of industries, commerce and governments. From creating programs for savings accounts, insurance or even manufacturing cars, software has become an integral part of many entities. Therefore, the efficient development of software is crucial for the modern world. [95]

Productivity is a very important part of any industry or venture, due to it being a driving factor for production cost. Productivity can be defined as the output divided by the effort required to produce said output. Measuring productivity for the purpose of improving it and therefore minimizing costs is very important. [95]

Measuring the productivity of software is however quite challenging due to the nature of software being abstract and seemingly trivial attributes such as size being challenging to measure. Many metrics have been developed and used for measuring the size of software. Things such as lines of code or function points. Lines of code are strictly the count of lines of code of a project and are from the developer's perspective, while function points are based on the functionality of the software as seen by the user. These metrics can however have certain drawbacks. Analysing the lines of code of a project is dependent on many factors such as what programming language is used and how much of the code is from reused materials and program generators. Function points can also have certain difficulties such as how functions are calculated and certain applications may give a misleading view of progress such as applications with high algorithmic complexity. These challenges have led to ever more sophisticated ideas of measuring software productivity and one of which is the COCOMO. [95]

The constructive cost model or COCOMO is a procedural cost estimate model which is based on the number of lines of code. This model is used to estimate the cost of software projects and it accounts for parameters such as size, efforts, cost, time and quality. [96] There are two separate COCOMO models which are COCOMO 1 and COCOMO 2. The COCOMO 2 model is a revisited version. The two different models are useful for two different styles of software development. COCOMO 1 is useful for traditional models of software development such as the waterfall model and it is measure in terms of size of software in lines of code. COCOMO 2 is useful in non-sequential models of software development with rapid development and reuse models of software with the size of software stated in terms of object points, function points and lines of code. [97]

**Table 1**. Productivity levels [98]



| Language | Level Relative to C |
|---|---|
| C | 1 |
| C++ | 2.5 |
| Fortran 95 | 2 |
| Java | 2.5 |
| Perl | 6 |
| Python | 6 |
| Smalltalk | 6 |
| Microsoft Visual Basic | 4.5 |

Source: Adapted from *Estimating Software Costs* (Jones 1998), *Software Cost Estimation with Cocomo II* (Boehm 2000), and "An Empirical Comparison of Seven Programming Languages" (Prechelt 2000).

In terms of productivity, programmers who use high-level programming languages achieve better productivity and quality than those who work with lower-level programming languages. This is due to the higher-level of abstraction which allows the programmer to achieve more functionality with the same effort. This level of productivity can be expressed numerically, and table 1 showcases the typical ratios of source statements of several languages when compared to the equivalent C code. The ratios depicted in the table state how much more the listed language accomplishes when compared then each line of code does in C. [98]

## 2.5 Education of programming

The purpose of this section is to view different programming languages and their underlying mechanics and attributes for the purpose of effective education. In other words, what types of programming languages are good for education and why. What underlying mechanics and types of choices made in the programming language contribute for effective learning and can more complicated programming languages give a better understanding of programming while being more difficult to adopt initially? To attempt to find an answer to these questions and more, this thesis examines different programming languages and reviews other such works to better understand the issue and attempts to draw conclusions from them.

There are many attributes that make for effective learning of anything. Programming at its core, is expression and problem solving. These attributes give programming and the learning of programming a certain set of skills necessary for effective learning regardless of the specific programming language used.

Learning programming requires the understanding of many different things such basic problem-solving skills, effective strategies and planning for the design of the program and implementation in addition to the programming tools and languages. Programming is a part of computer science and understanding the scientific method plays a major role in the learning of programming. General intelligence and mathematical skill seem to directly correlate with the learning of programming skills. [86]

The ability to break down obstacles and projects down to basic solvable problems is the very essence of programming. Other skills which directly contribute to the effective learning of programming are things such as the ability to understand and apply logic, abstract thinking and understanding patterns. Logical thinking obviously has a major role in the development of software. The ability to take basic tools and building blocks and use them to construct something is a common practice in coding. Understanding patterns is another major skill for programming. Computers are in many ways quite stupid and will mindlessly execute any and all commands precisely as they are given. This lack of any independent or creative thought creates many problems because of the disconnect between the man and the machine when it comes to giving instruction. Many of the greatest achievements in programming where done by the clever use of creative algorithms to make computers function in more independent, brilliant and efficient ways. Some examples of this are things such as sorting algorithms and even artificial intelligence. [34]

Choosing the correct language to teach programming to computer science students is difficult because there are many different programming languages, many with very different purposes and methods of operation. For the student to have a basic understanding of programming, choosing a niche programming language such as SQL or a programming language with a very specific set of paradigms and functions such as purely functional programming languages is perhaps unwise. Therefore, it is a good idea to initially teach a student a more general-purpose programming language.

There are however more than 300 programming languages and choosing a suitable one for students is challenging. In addition, programming languages are not typically designed for teaching programming to a student but instead to fill a role in software development. Learning a “throw-away” language just to learn programming is perhaps not a good idea but instead choose a suitable language with actual application. The goals for a good introductory programming language should be to teach the student basic programming skills, support logical thinking and further develop the student's problem solving abilities. [87]

To objectively and consistently compare different programming languages, some basic criteria are needed. In this thesis, the following criteria are selected as a benchmark of evaluation for any given programming language.

**Simplicity**: Simplicity of the language is very important, especially for a beginner and also quite desirable for anyone. In addition, simplicity could be interpreted as how close the programming language is to typical human languages.

**Writability**: The writability of the programming language is another important attribute for both novice and professional users. A good programming language should provide constructs and APIs (Application programming interface). This will greatly aid the user in creating both specific and general programs. In addition, a good programming language should provide support for a variety of data types. This feature would give the language significant choice in the method a problem is solved. [35] In other words, writability means how easily a language can be used to create programs for a chosen problem domain.

**Reliability and fault tolerance**: Needless to say, reliability of the language is important but also the ability to recover from possible errors or even preventing them equally so. Features such as support for assertions, error checking and exception handling are quite important for a programming language. [35]

**Market demand, community support and over all availability**: These are all attributers which are required. Availability in terms of both development tools, compilers or interpreters and IDEs (Integrated development environment) should be available and preferably free and easy to find. Support for a programming language is very important for finding help and solving problems. Market demand is crucial because the language needs to have a practical application in the senses of commercially available employment for the student investing time or other resources in education. [35]

**Coverage and extensions**: Coverage in this context, refers to the programming languages concepts and abilities to be applied in as many applications as possible. Coverage in terms of concepts includes topics such as object-orientation, abilities for multithreading, I/O (input and output), support for mobile applications, support for databases, system level programming etc. Extensions are also important for a programming language, so the user does not need to "re-invent the wheel" for every application, but instead use an available library, a driver for hardware interface etc. [35]

# 3. RESEARCH METHODOLOGY

The purpose of this chapter is to discuss the methodologies used in this thesis. This chapter also discusses the methodologies chosen and perhaps more importantly, why they are chosen for each individual section.

This thesis will focus mainly on examining existing data such as literary works on books and articles and reviewing existing figures in lieu of measurement or conducting independent researches. The conclusions or thoughts drawn in this thesis are derived by logical deductions and thinking supported by existing works. The generic methods employed in this thesis can be categorized in two general categories which are quantitative and qualitative methods.

## 3.1 Quantitative methods

Quantitative methods are a method of research that rely solely on the measurement of variables using numerical systems and analysing these measurements using any variety of models and reporting the relationships and associations of the studied variables [36]. In other words, the section of the research which focuses on measurable and quantifiable attributes.

The quantitative research methods in thesis are mainly focused on directly measurable aspects such as performance tests and popularity figures. The comparing and contrasting of these results can be done quite objectively relying for the most part on the numbers reported while bearing in mind certain aspects. These aspects can be things such as the efficiency of the different codes tested and the validity and sample size of the numbers collected. This thesis will focus on reported tests and studies for the purposes of drawing meaningful conclusions from them. This thesis will not independently run performance tests or collect independently sourced numerical data due to time constraints.

## 3.2 Qualitative methods

Qualitative research methods focus more on the quality aspects of the research area instead of just quantifiable things such as directly comparable numbers or statistics. This allows the research of selected issues in-depth and detail without being constrained by pre-determined categories of analysis. While quantitative research can be conducted with accurate and more scientific methods such as comparing figures or statistics, qualitative methods are much more difficult to judge and draw objective conclusions form. To attempt to make the results derived from qualitative research methods valid and objective, the following key aspects of the study must be focused on rigorously.

Context: Findings derived from quantitative research are very dependent on their context. Referencing the situation and the environment in which the findings where derived from is therefore very important.

Credibility and intellectual integrity: The research done by qualitative methods must remain as objective as possible. Anything that may affect the methods used, data collection, analysis, interpretation and conclusions drawn must be reported and explained.

Confirmability: In qualitative research it is vital to show how the study has come to the reported conclusions. This is done to showcase the inner workings of the research, so the trail of thought and the logical connections are understandable. [37]

The qualitative research methods employed in this thesis focus on certain aspects that cannot be numerically compared or contrasted, while still being able to be objectively delineated. Examples of these can be for instance the readability or simplicity of different programming languages or user friendliness.

## 3.3 Methods for choosing a programming language

In addition to very generic methodologies such as the quantitative and qualitative methods, some more specific methods can also be used for certain comparisons. These methods are more purpose specific and may include both quantitative and qualitative methodologies. For the purpose of choosing a programming language, a less generic comparing method can be advantageous.

Choosing a suitable programming language can be challenging. Some choices are quite obvious or in certain cases, there can only be a single programming language capable of the specific task in hand leaving no room for choice at all. In many cases, external factors can also affect the choice to use a certain programming language. To give these differing methods of choosing some more structure, this thesis factors the choices in two distinct categories. These two categories are hard choices and soft choices. [38]

Hard Choices: Certain projects can have very serious restrictions or limitations, which would necessitate the use of a certain programming language. Hard choices are choices which leave the user little room for choosing and a programming language is chosen because it is either the only or from a very select few applicable for the task.

Soft Choices: Soft choices give the user more freedom in terms of applicable programming languages to choose for the project but introduces other attributes for the decision-making process. These attributes are things such as productivity, environment, support, user friendliness and other such features. [38]

Measuring the real-world performance differences of programming languages is quite challenging. In addition to the difficulty of obtaining identical programs written on many different programming languages, there are many other factors which will affect the results such as how well the program is written or optimized. For the sake of brevity, this thesis focuses mainly on very basic benchmarks to give a rough estimate on how any given programming language is expected to perform when running basic short tasks.

The performance measurements of this thesis where mainly sourced from a website titled "The computer language Benchmark Game". This website catalogues an ever-changing user written set of ten different small programs which can be easily compared with one and other. The tables depicting the performance measurements catalogue the runtime in seconds denoted as "secs", the memory consumption of the program denoted as "mem", the size of the source code denoted as "gz", busy and computer processor load percentage depicted as "cpu load". More information on this specific testing methodology can be found at the following web address. https://benchmarksgame-team.pages.debian.net/benchmarksgame/how-programs-are-measured.html [44]

# 4. RESULTS

The purpose of this chapter is to catalogue and showcase the results found in this thesis. The results of the study will be further reflected upon in the following chapter.

## 4.1 Comparison with other programming languages

The purpose of this section is to compare the C++ programming language to its main alternatives in terms of general use and popularity depicted in figure 4. Figure 4 is an image from the TIOBE index which is a programming community index indicating the popularity of programming languages. The ratings of this index are based on the number of search engine results with the name of the programming language from many of popular sites such as Wikipedia, Google, YouTube etc. [39] This section briefly overviews a selection of programming languages and then compares their generic relevant features to C++. Certain features of C++ or other more specific programming languages may also be compared in certain specific situations for context. Purpose specific programming languages such as SQL or Matlab are not included in the comparison.

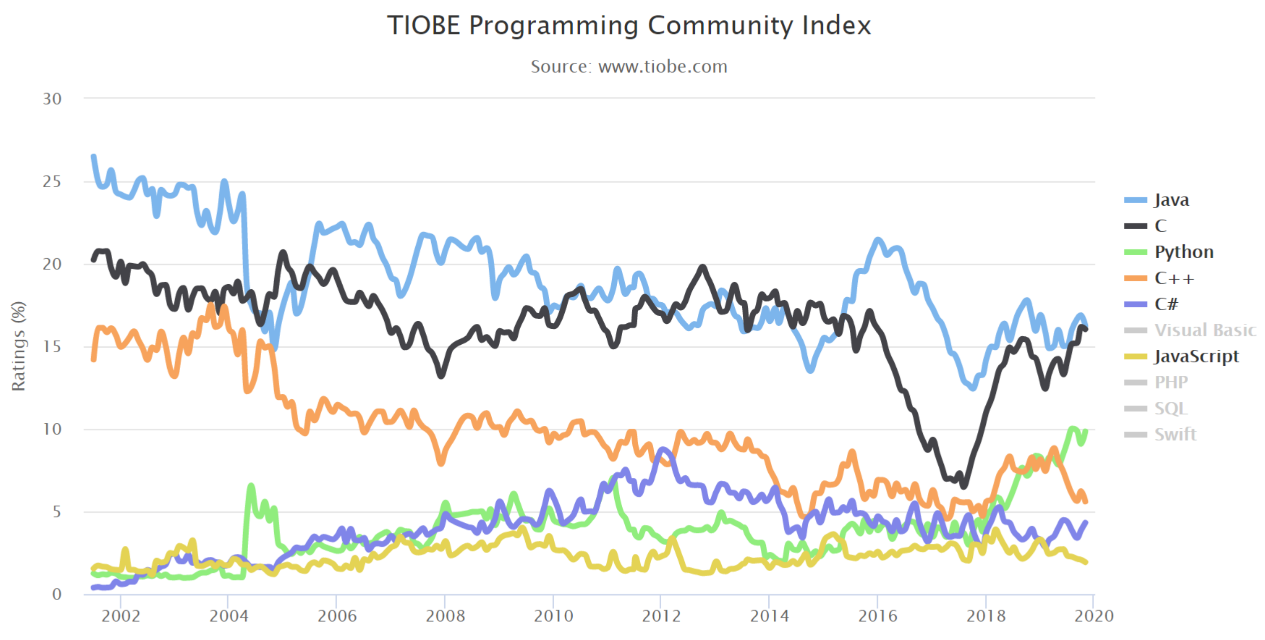


**Figure 4**. Tiobe Index [39]

There are many different reasons which contribute to the decision-making process when choosing a programming language for any given task. Many features and attributes both in and surrounding the programming language can affect this choice. For this comparison, a few programming languages and their differing attributes when comparing them to the C++ programming language are selected. The more detailed description of the testing methodologies where discussed in chapter 3 of this thesis.

### 4.1.1 Comparison with Java

The first programming language selected for this comparison is the Java programming language. Java was selected for this comparison because of the language's popularity and because of the relevant differences between it and C++. Discussing the minute details and intricacies of each programming language are very difficult. For the interest of brevity, this thesis will mainly focus on the more generic differences between the Java and C++ programming languages.

The Java programming language was originally developed by James Gosling and released in 1995. Java derives many of its syntactic features from the C++ programming language while having fewer low-level facilities as either of them. [40] The Java programming language can be generically described as a general-purpose programming language which features the object-orientated-, imperative- and functional- programming paradigms and is statically typed. In terms of both paradigms of programming and syntax, Java and C++ are quite similar in many ways. The main practical differences between the two languages are Java's memory management and garbage collection and the compilation process of the language.

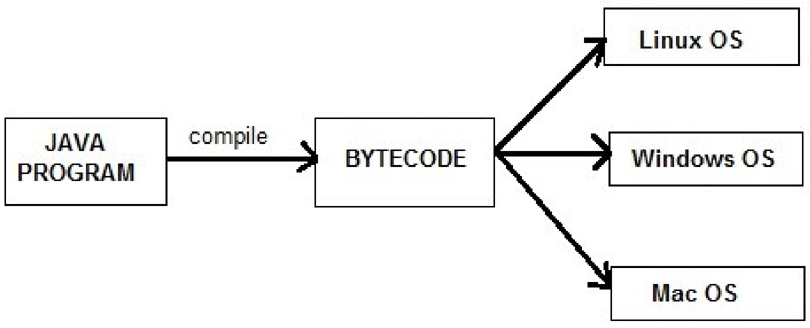


**Figure 5.** Compilation of Java [41]

The compilation and execution of Java and C++ differ greatly and is a very delineating factor between the two programming languages. To briefly explain this difference, the C++ programming language is compiled into platform and operating system specific machine code. This differs greatly from the Java programming language, which is compiled into bytecode. This bytecode is then run on Java's runtime environment for instance the

Java virtual machine which is commonly abbreviated to JVM. The diagram in figure 5 gives some visual aid for this process. In practical terms, this makes the Java programming language platform independent. Meaning that code written on Java can be ran on any environment where there is an existing runtime environment which supports Java. [41]

To achieve this feature of platform independence, the Java programming language does not work on a single step compilation. Instead, this process involves the execution of two steps, the first of which is an operating system independent compiler and the second is the Java virtual machine. This process is depicted in figure 6 for visual aid.

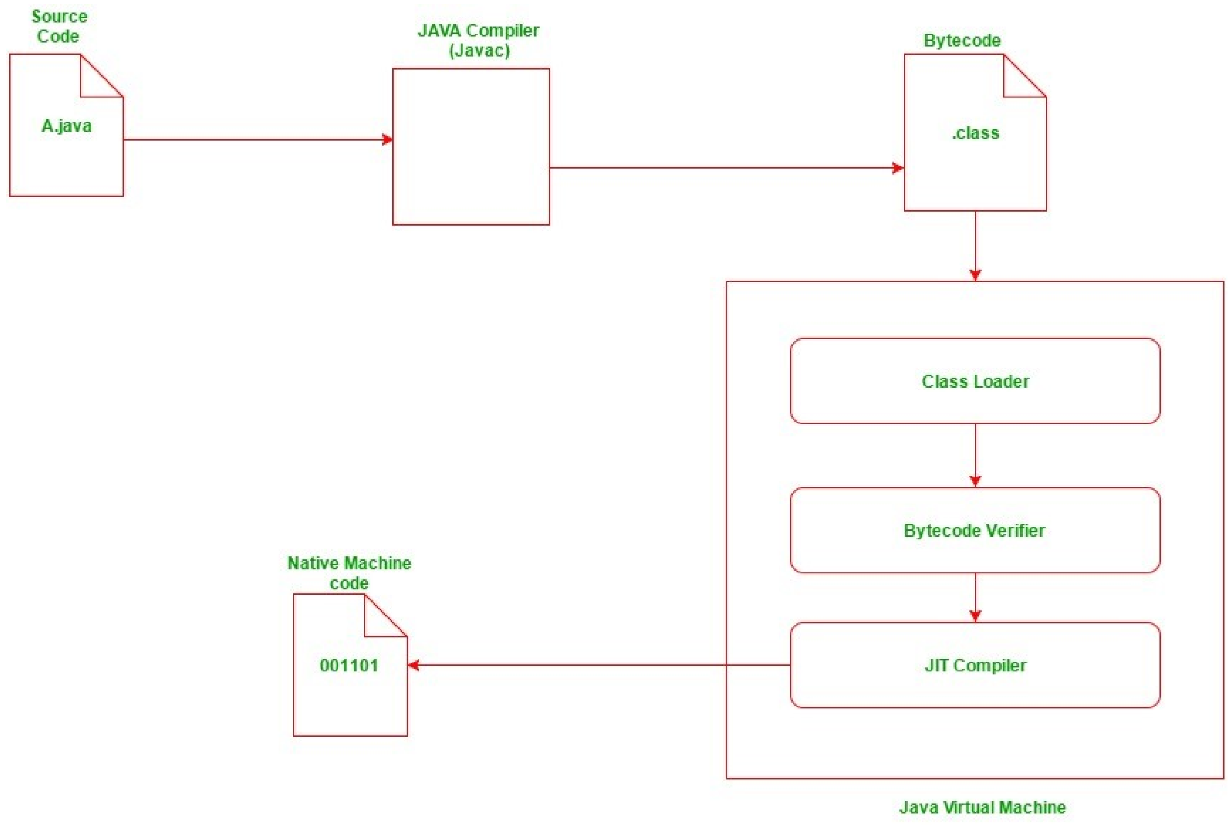


**Figure 6.** Compilation process of Java [42]

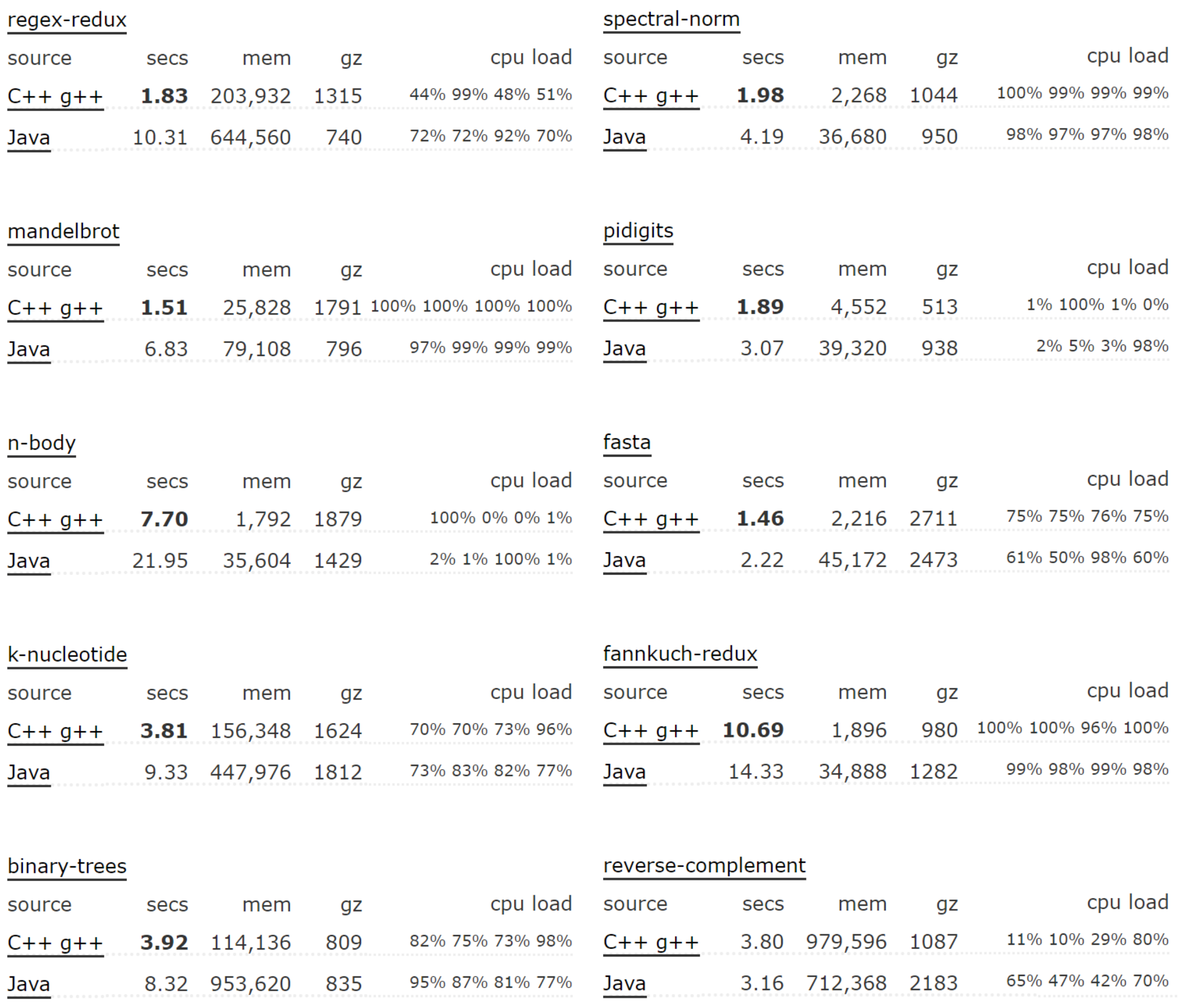
In the compilation stage, the Java source code contained in the .java file, is passed through the compiler, which encodes the source code into a machine independent encoding, also commonly known as bytecode. The source code also contains classes and the content of these classes is then stored in a separate .class file. The process of converting the source code into bytecode is done in the following seven steps. The first step is parsing, where a set of java source files are read, and the resulting tokens are mapped into nodes on an abstract syntax tree. The second step enters the symbols for the definitions into the symbol table. The third processes the annotations found in the specified compilation units if requested. The fourth step attributes the syntax trees which includes name resolution, type checking and constant folding. In the fifth step, dataflow analysis are performed on the syntax trees from the previous step. This process also checks for assignments and reachability. In the sixth step the abstract syntax tree is rewritten, and syntactic sugar is translated away. The seventh and final step, generates the .class files.

In the execution stage, the class files generated by the compiler are independent from the operating system or the machine. This allows them to be run on any system where the main class file is passed to the Java virtual machine. To execute the final machine code, three stages have to be gone through. The first stage is the class loader where the main class is loaded into the memory by passing its .class file to the Java virtual machine. In addition, all other classes referenced in the program are loaded through the class loader. The next stage is the bytecode verifier where the instructions are checked so they do not contain any damaging actions. The final stage is the Just-In-Time Compiler where the loaded bytecode is converted into machine code. [42]

The second major difference between C++ and Java is how the two languages handle memory allocation and deallocation. In the C++ programming language, the user is responsible for both creating and destroying objects and certain variables which are created by using the “new” keyword. If these allocated objects or variables are then not manually deallocated by the user, there can be certain problems such as memory leaks or other problems which may lead to the entire program crashing as a result. Though this can be mitigated in C++ by the use of newer features such as smart pointers. This process of deallocating memory is done automatically in the Java programming language by the use of the Garbage collector. The Garbage collector is responsible for deallocating memory allocated on the heap by automatically destroying unreachable objects. Unreachable objects are objects which do not contain any references to them. [43]

The Java programming language also offers the user many run-time features which can greatly aid in both error detection and prevention when compared to the C++ programming language. Run-time features and the ability to function platform independently however come at a cost to performance as seen in table 2. In addition to the execution of all the programs being slower than C++, table 2 also shows that the memory usage of Java is far greater in all but one of the benchmarks. The tables depicting the performance measurements catalogue the runtime in seconds denoted as “secs”, the memory consumption of the program denoted as “mem”, the size of the source code denoted as “gz”, busy and computer processor load percentage depicted as “cpu load”.

**Table 2.** C++ and Java performance test [45]



regex-redux

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C++ g++ | **1.83** | 203,932 | 1315 | 44% 99% 48% 51% |
| Java | 10.31 | 644,560 | 740 | 72% 72% 92% 70% |

spectral-norm

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C++ g++ | **1.98** | 2,268 | 1044 | 100% 99% 99% 99% |
| Java | 4.19 | 36,680 | 950 | 98% 97% 97% 98% |

mandelbrot

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C++ g++ | **1.51** | 25,828 | 1791 | 100% 100% 100% 100% |
| Java | 6.83 | 79,108 | 796 | 97% 99% 99% 99% |

pidigits

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C++ g++ | **1.89** | 4,552 | 513 | 1% 100% 1% 0% |
| Java | 3.07 | 39,320 | 938 | 2% 5% 3% 98% |

n-body

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C++ g++ | **7.70** | 1,792 | 1879 | 100% 0% 0% 1% |
| Java | 21.95 | 35,604 | 1429 | 2% 1% 100% 1% |

fasta

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C++ g++ | **1.46** | 2,216 | 2711 | 75% 75% 76% 75% |
| Java | 2.22 | 45,172 | 2473 | 61% 50% 98% 60% |

k-nucleotide

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C++ g++ | **3.81** | 156,348 | 1624 | 70% 70% 73% 96% |
| Java | 9.33 | 447,976 | 1812 | 73% 83% 82% 77% |

fannkuch-redux

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C++ g++ | **10.69** | 1,896 | 980 | 100% 100% 96% 100% |
| Java | 14.33 | 34,888 | 1282 | 99% 98% 99% 98% |

binary-trees

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C++ g++ | **3.92** | 114,136 | 809 | 82% 75% 73% 98% |
| Java | 8.32 | 953,620 | 835 | 95% 87% 81% 77% |

reverse-complement

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C++ g++ | 3.80 | 979,596 | 1087 | 11% 10% 29% 80% |
| Java | 3.16 | 712,368 | 2183 | 65% 47% 42% 70% |

Comparing the generic features and attributes of the Java and C++ programming languages can give a rough idea on what the two different languages are more suited to and do well in. Due to the popularity of the Java programming language, support and libraries found for the Java programming language are in many cases more numerous than what can be readily found for C++. However, due to the two languages having many different target-uses, there will be differences in support for certain applications. Java also has run-time features which allows for a variety of different attributes such as run-time error detections and better memory safety due to the garbage collection. The memory management in the Java programming language is handled by the system and the language does not use pointers. Java does have support for threads and interfaces. The object-oriented features in C++ extend further then Java's. The C++ programming language provides both single and multiple inheritance while Java only supports single inheritance for its classes. Multiple inheritance can be used in the Java programming language through interfaces which can be used to extend multiple interfaces. Program handling is also somewhat different between the two languages. In the C++ programming language, methods of objects and data can reside outside of classes and there are concepts of global file, namespaces and scopes available. This differs greatly from the Java programming language where methods and data reside within the class itself. Platform independence and portability are also a very strong feature that Java processes. The ability to write code which can be ran on any platform is a feature that the C++ programming language does not poses as the source code has to be recompiled for different platforms. [46] Because of the lower abstraction layer and lack for a need for a virtual machine, the C++ programming language manages to achieve the higher performance. In addition, the C++ programming language is more suited to be utilized in much more restricted environments such as embedded systems although the Java programming language can certainly be used in such applications and even offers tool for such use. The C++ programming language also gives the user more freedom in the approach of solving problems by means of allowing the programmer better access to the memory.

While the ability to directly access the computer’s hardware with few safeguards gives the user more freedom and perhaps a better avenue for solving certain problems, it can lead to other problems such as memory leaks and other exploits. The Java programming language can mitigate many of these issues by both restricting access to the memory and by run-time features. The memory security in the Java programming language is done by the Garbage collection where memory can be implicitly allocated but not freed. The type safety is done by strict type checking of instruction operands and there is no pointer arithmetic. There are also run-time checks for array accesses and casts. Despite all this, Java is by no means immune from security risks and some common attacks can be carried out by type confusion, class spoofing and bald implementation of system classes. [49]

### 4.1.2 Comparison with C#

The C# which is pronounced “C sharp” is selected for this comparison because of the popularity of the language and because it can be seen in some ways as an evolution or a modern version of C++ just as C++ was a modernized C at the time it was developed. The C# programming language also has many similarities with Java and is a more modern programming language. The C# programming language was developed by Microsoft and launched in 2001. [50] In terms of its underlying features, the C# programming language is a general-purpose programming language with strong typing and it is also type safe. In terms of programming paradigms, the C# programming language encompasses the paradigms of imperative-, declarative-, object-oriented- and functional- programming paradigms. [51] In terms of syntax, the C# programming language is very similar to C++ but with many simplifications. The syntax of the C# language was designed to do away with many of the complexities of C++ while still providing many of the features that it provides which the Java programming language does not provide. The C# programming language provides many powerful features such as nullable value types, enumeration, delegates, lambda expressions and even direct access to memory, which Java does not provide. In addition, the C# programming language supports generic methods and types which can provide both increased type safety and increased performance. Unlike C++ but much like Java, the C# programming language features an automatic memory management system in the form of the Garbage collector. The object-oriented features and abilities of the C# language are similar to the C++ language in terms that C# supports the concepts of encapsulation, inheritance and polymorphism however, C# is only capable of single inheritance.

The compilation and execution process of the C# language is different from C++ or Java but has many similar elements to both of them. This process is depicted in figure 7 and is quite similar to how Java functions. A program written on C# is run on the .NET framework. The .NET framework is an integral component of Windows and it includes a virtual execution system which is called the common language runtime or CLR and a unified set of class libraries. The source code written in C# is compiled into an intermediate language which conforms to the specifications of the common language infrastructure. The intermediate language's code and resources, such as bitmaps and strings, are stored on the disk in an executable file known as the assembly file. This assembly file contains a manifest which provides information about the assembly's types, versions, culture and security requirements.

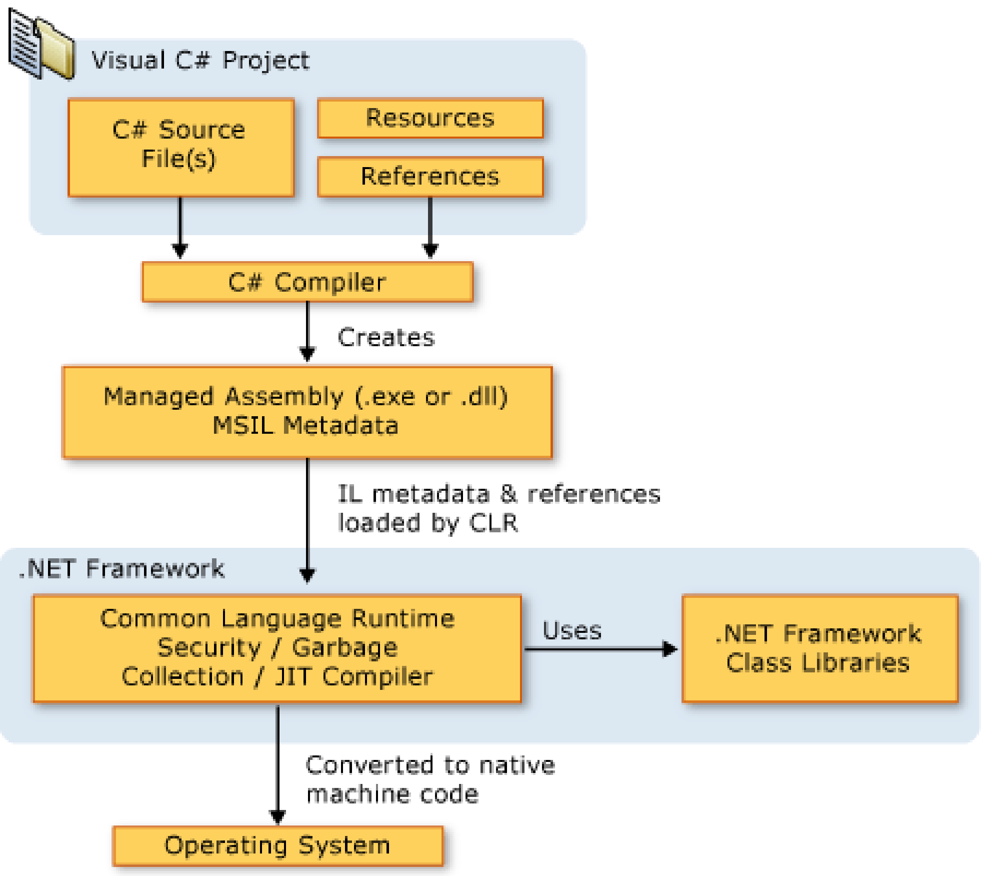
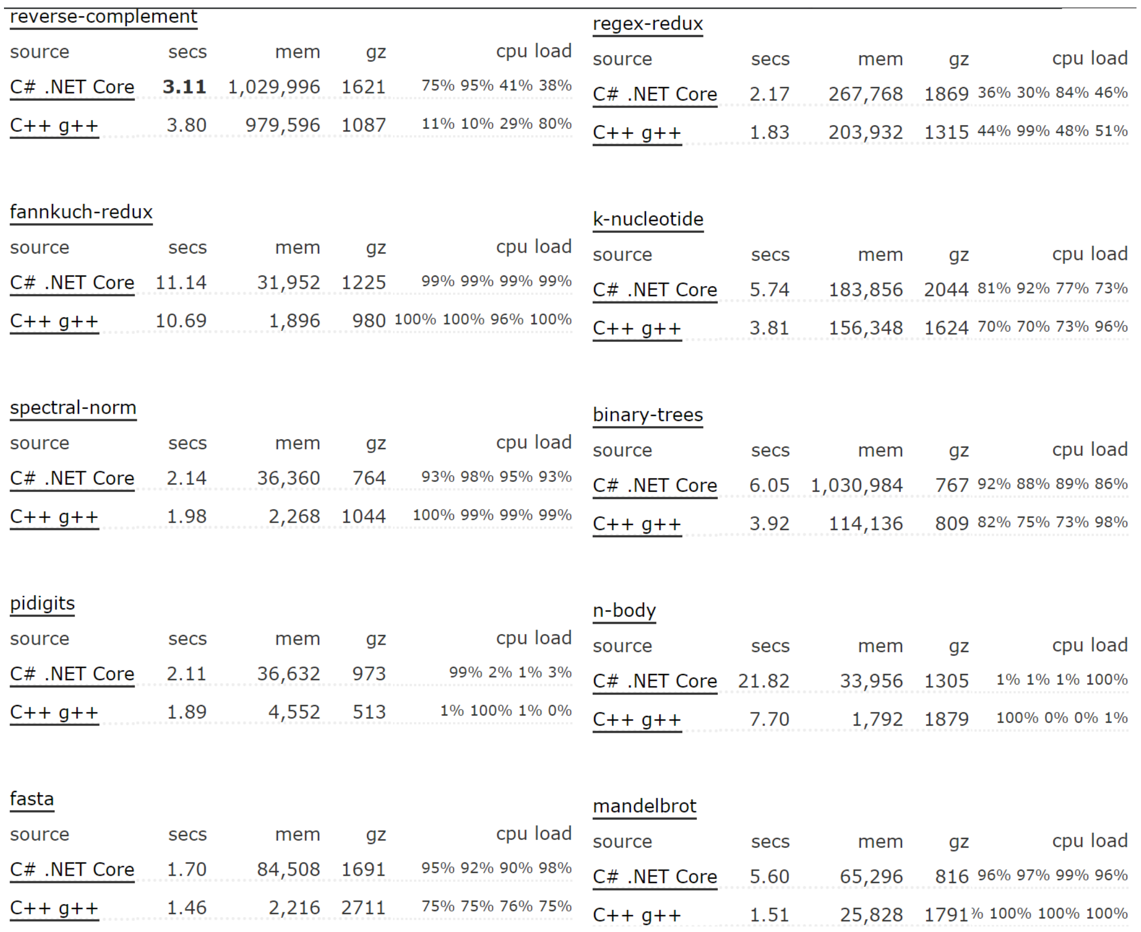

Visual C# Project
C# Source File(s)
Resources
References
C# Compiler
Creates
Managed Assembly (.exe or .dll) MSIL Metadata
IL metadata & references loaded by CLR
.NET Framework
Common Language Runtime Security / Garbage Collection / JIT Compiler
Uses
.NET Framework Class Libraries
Converted to native machine code
Operating System

**Figure 7.** C# Compilation [52]

The execution of a program written on the C# programming language starts by loading the assembled file into the common language runtime, which based upon the information in the manifest, may take different actions. After this process and if the security requirements are met, the common language runtime performs just in time compilation to convert the intermediate language code to native machine instructions. In addition, the common language runtime provides other services related to the automatic Garbage collection, exception handling and resource management. [52]

**Table 3.** C++ and C# performance test [53]



reverse-complement

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C# .NET Core | **3.11** | 1,029,996 | 1621 | 75% 95% 41% 38% |
| C++ g++ | 3.80 | 979,596 | 1087 | 11% 10% 29% 80% |

regex-redux

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C# .NET Core | 2.17 | 267,768 | 1869 | 36% 30% 84% 46% |
| C++ g++ | 1.83 | 203,932 | 1315 | 44% 99% 48% 51% |

fannkuch-redux

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C# .NET Core | 11.14 | 31,952 | 1225 | 99% 99% 99% 99% |
| C++ g++ | 10.69 | 1,896 | 980 | 100% 100% 96% 100% |

k-nucleotide

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C# .NET Core | 5.74 | 183,856 | 2044 | 81% 92% 77% 73% |
| C++ g++ | 3.81 | 156,348 | 1624 | 70% 70% 73% 96% |

spectral-norm

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C# .NET Core | 2.14 | 36,360 | 764 | 93% 98% 95% 93% |
| C++ g++ | 1.98 | 2,268 | 1044 | 100% 99% 99% 99% |

binary-trees

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C# .NET Core | 6.05 | 1,030,984 | 767 | 92% 88% 89% 86% |
| C++ g++ | 3.92 | 114,136 | 809 | 82% 75% 73% 98% |

pidigits

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C# .NET Core | 2.11 | 36,632 | 973 | 99% 2% 1% 3% |
| C++ g++ | 1.89 | 4,552 | 513 | 1% 100% 1% 0% |

n-body

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C# .NET Core | 21.82 | 33,956 | 1305 | 1% 1% 1% 100% |
| C++ g++ | 7.70 | 1,792 | 1879 | 100% 0% 0% 1% |

fasta

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C# .NET Core | 1.70 | 84,508 | 1691 | 95% 92% 90% 98% |
| C++ g++ | 1.46 | 2,216 | 2711 | 75% 75% 76% 75% |

mandelbrot

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C# .NET Core | 5.60 | 65,296 | 816 | 96% 97% 99% 96% |
| C++ g++ | 1.51 | 25,828 | 1791 | % 100% 100% 100% |

Describing the generic differences between the C#- and the C++ -programming languages gives an idea on how they differ from one and other on an overall level. The C# programming language is almost a hybrid of the C++ and Java programming languages in the sense that it encompasses many of the ideas and the syntax of C++ while having many of the more modern features of Java, for instance the Garbage collection system and run-time features. There are however many crucial differences between C# and C++. Starting with some of the practical differences, the C# programming language does not feature global functions, header files and pointers are not applicable. Because of many of these features, the C# programming language is considered to be a higher-level programming language and less difficult. [53] The portability of the C# programming language is also quite strange as it is mainly targeted (although not limited) for the Windows environment while C++ can be compiled for any platform. The C# programming language itself is platform independent however, the .NET platform and the associated libraries may not always be. There are many major differences in the performance between the C#- and the C++ -programming languages as depicted in table 3. In terms of runtime, the C# programming language manages to beat C++ in only one of the ten tests shown, while consuming considerably more memory in every test. The C++ programming language can also work in more restricted environments because it does not need a secondary environment to run on. In addition, the C++ programming language can give the user more freedom in terms of selecting different ways of solving problems and is overall less restrictive. The C# programming language does however have tighter security and run-time features which can both prevent and aid assisting in error cases. Buffer overflows are however possible even in the C# programming language by for instance, using the “unsafe” keyword for allocating pointers.

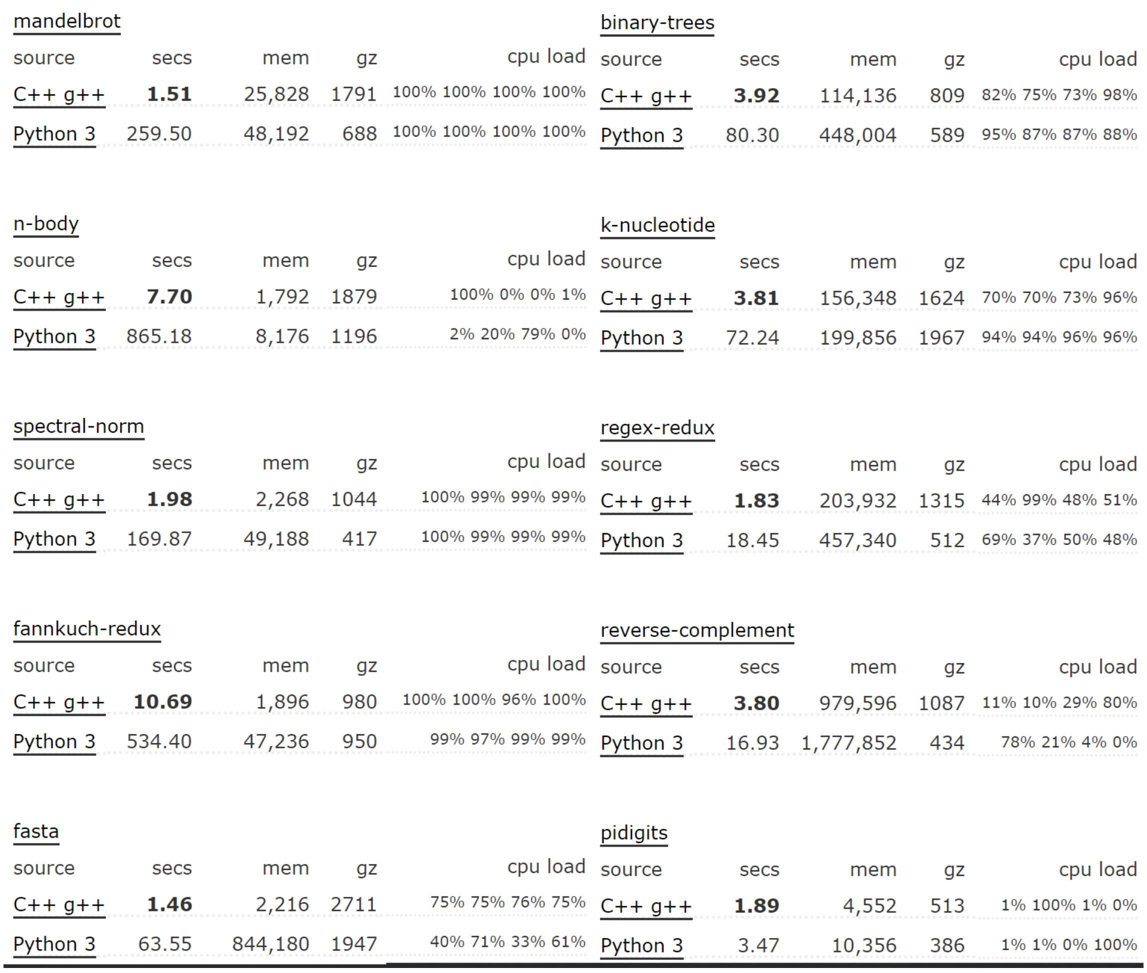
### 4.1.3 Comparison with Python

The Python programming language was selected for this comparison because it has recently overtaken the C++ programming language in popularity according to the Tiobe index. In addition, the Python programming language has many interesting differences to C++.

The Python programming language is a high-level general-purpose programming language, created by Guido Van Rossum and released in 1991. Python features many high-level features such as a dynamic type system, automatic memory management in the form of a Garbage collector and many other run-time features. In terms of programming paradigms, the Python programming language supports imperative-, procedural-, object-oriented- and functional- programming paradigms. [55] Dynamic typing in Python is the opposite of static typing found in languages such as C++ and this is a large practical difference between Python and C++. The definition of dynamic typing is that a programming language which is dynamically typed does not enforce or check type-safety at compile-time but instead defers them at run time. In practical terms, this means that the type of for instance a variable, does not need to be explicitly defined but instead the deduction of the variable type is done automatically. [56]

The programming languages discussed thus far, have all been compiled languages but Python is an interpreted programming language. An interpreted programming language is directly executed, rather than being first converted into a basic form such as machine code for execution as is the case with a compiled programming language [57]. In more practical terms, the source code written in Python is converted into an intermediate language which is then translated into the native language or machine language for the target platform. The code written in Python is first compiled into python bytecode, this bytecode compilation is done internally and is a step for the translation as the bytecode functions as a representation of a lower-level code which is platform independent. The compilation phase creates a .pyc file which is then executed on a virtual machine. The virtual machine is the runtime engine for the Python programming language, and it is the actual program which runs the Python scrips. [58] This process works by loading the byte code into the Python runtime and then interpreted by the Python Virtual Machine which then reads each instruction in the byte code and executes the operations as instructed. In addition, each time the interpreter program is run, the interpreter must convert the source code into machine code and pull in runtime libraries. [59] The Python programming language is a very portable language and can be run on any platform with the appropriate interpreter.

**Table 4.** C++ and Python performance test [60]



| | source | secs | mem | gz | cpu load |
|---|---|---|---|---|---|
| mandelbrot | C++ g++ | **1.51** | 25,828 | 1791 | 100% 100% 100% 100% |
| | Python 3 | 259.50 | 48,192 | 688 | 100% 100% 100% 100% |
| binary-trees | C++ g++ | **3.92** | 114,136 | 809 | 82% 75% 73% 98% |
| | Python 3 | 80.30 | 448,004 | 589 | 95% 87% 87% 88% |
| n-body | C++ g++ | **7.70** | 1,792 | 1879 | 100% 0% 0% 1% |
| | Python 3 | 865.18 | 8,176 | 1196 | 2% 20% 79% 0% |
| k-nucleotide | C++ g++ | **3.81** | 156,348 | 1624 | 70% 70% 73% 96% |
| | Python 3 | 72.24 | 199,856 | 1967 | 94% 94% 96% 96% |
| spectral-norm | C++ g++ | **1.98** | 2,268 | 1044 | 100% 99% 99% 99% |
| | Python 3 | 169.87 | 49,188 | 417 | 100% 99% 99% 99% |
| regex-redux | C++ g++ | **1.83** | 203,932 | 1315 | 44% 99% 48% 51% |
| | Python 3 | 18.45 | 457,340 | 512 | 69% 37% 50% 48% |
| fannkuch-redux | C++ g++ | **10.69** | 1,896 | 980 | 100% 100% 96% 100% |
| | Python 3 | 534.40 | 47,236 | 950 | 99% 97% 99% 99% |
| reverse-complement | C++ g++ | **3.80** | 979,596 | 1087 | 11% 10% 29% 80% |
| | Python 3 | 16.93 | 1,777,852 | 434 | 78% 21% 4% 0% |
| fasta | C++ g++ | **1.46** | 2,216 | 2711 | 75% 75% 76% 75% |
| | Python 3 | 63.55 | 844,180 | 1947 | 40% 71% 33% 61% |
| pidigits | C++ g++ | **1.89** | 4,552 | 513 | 1% 100% 1% 0% |
| | Python 3 | 3.47 | 10,356 | 386 | 1% 1% 0% 100% |

Even the generic differences between the Python- and C++- programming languages are quite numerous and well defined. The obvious practical difference between the two languages probably most evident to the user will be the syntactic difference between the two languages. The syntax of Python programming language closely resembles the English language in many of its operations. As an example, the print function in Python is just denoted as “Print” and instead of separators such as semicolons, Python code is delineated by the use of indentations. In addition, the overall code written in Python, when compared to C++, is much shorter and easier to understand. The Python programming language is therefore much more expressive and a higher-level programming language then C++. This also makes Python far less prone to lower-level programming errors or memory exploits when compared to the C++ programming language. Most of this is done by the run-time features featured in the Python programming language such as the Garbage collector.

The higher abstraction level and automation of many lower level features however come at a cost to performance which is depicted in table 4. As noted in table 4, the C++ programming language beats Python in every performance test and in some tests the C++ programming language is faster by more than 100 times. In addition to the slower run times, the Python programming language requires far more memory. This difference in performance and the requirement for the interpreter, rules the Python programming language out of many use cases where run-time performance is imperative or restricted environments. Overall, the C++- and Python- programming language differ quite a lot and this is apparent in the use-cases of the two languages. Comparing the two languages directly is therefore perhaps not very applicable to many cases except the education of programming.

### 4.1.4 Comparison with Rust

The Rust programming language was chosen for this comparison because it in many ways appears to be a proposed direct alternative to the C++ programming language. The Rust programming language is less in the mainstream of programming languages and therefore the comparison with Rust is only relegated to this section of the thesis and will be less examined for the subsequent sections.

The Rust programming language was created by Mozilla research and released in 2015. In terms of programming paradigms, the Rust programming language is functional, object-oriented and concurrent. [72] Rust is also directly compiled into machine code just like the C++ programming language and does not feature a garbage collection system. The syntax of Rust is meant to closely resemble the that of the C and C++ programming languages but there are many differences. The original aim of Rust was to be a systems language for high performance applications which are typically written in C or C++ but Rust was also meant to prevent certain types of problems such as invalid memory access which may occur in the C or C++ languages.

The Rust programming language was directly targeted as an alternative to C and C++ as a system programming language. Rust was meant to be safer than C and C++ with lesser likelihood of crashing, better memory safety and easier for concurrency. The developer of rust even went as far to state that the Rust programming language was meant as a safer and easier alternative to C++ in systems-level projects where performance and deployment characteristics are similar. [73]

Despite being developed for a very similar purpose as C++, there are certain key differences between the two programming languages. The first and one of the more deliberately made difference is the more secure memory usage of Rust. In Rust, there is a dedicated check for free variables usage and dangling pointers. In addition, there are tools which can find raw pointers used inside unsafe sections. This differs from C++ where the user must manually review the code and keep track of raw pointers. The C++ programming language does however feature smart pointers. Another feature in Rust for using the memory more securely is the check for null dereferencing errors. In the Rust programming language, option types can emulate null references and they need to be explicitly checked before use. In addition, raw pointers can be set to null only when they are used inside unsafe blocks. This differs from C++, because even with smart pointers null dereferencing is possible, and compilers will not catch this type of warning or possible error. The possibility of errors caused by buffer overflow in Rust is mitigated by enforcing automatic range checks on all slice types at runtime. In C++, range checks can only be the used inside of a wrapper class. [74]

Another key difference between Rust and C++ is how they handle concurrency and data races between threads. Rust checks for possible inconsistencies with the possibly unsafe modification of concurrent data and can trace them with the Rust reference model and a built-in borrow checker at compile time. In addition, unsafe misuse of mutexes can be made impossible by locking API unsafely. This differs from C++, where most of the responsibilities of concurrency fall on the shoulders of the user. [74] Obviously, there are many other major and minor differences between the two languages such as the way they handle compile-time polymorphism and pattern matching just to name a couple. But overall, the two programming languages are attempting to fill the same purpose in programming.

**Table 5.** C++ and Rust performance test [75]

reverse-complement

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| Rust | **1.69** | 994,564 | 1330 | 45% 26% 47% 59% |
| C++ g++ | 4.71 | 500,036 | 840 | 14% 86% 0% 1% |

fasta

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| Rust | 1.51 | 2,044 | 1906 | 74% 73% 82% 75% |
| C++ g++ | 1.46 | 2,216 | 2711 | 75% 75% 76% 75% |

n-body

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| Rust | **6.07** | 940 | 1753 | 100% 1% 0% 0% |
| C++ g++ | 7.70 | 1,792 | 1879 | 100% 0% 0% 1% |

fannkuch-redux

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| Rust | 11.08 | 944 | 1016 | 100% 94% 100% 99% |
| C++ g++ | 10.69 | 1,896 | 980 | 100% 100% 96% 100% |

binary-trees

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| Rust | **3.31** | 199,852 | 721 | 88% 88% 87% 100% |
| C++ g++ | 3.92 | 114,136 | 809 | 82% 75% 73% 98% |

mandelbrot

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| Rust | 1.70 | 30,544 | 1332 | 98% 98% 99% 98% |
| C++ g++ | 1.51 | 25,828 | 1791 | 100% 100% 100% 100% |

pidigits

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| Rust | **1.75** | 3,036 | 1366 | 100% 0% 1% 1% |
| C++ g++ | 1.89 | 4,552 | 513 | 1% 100% 1% 0% |

regex-redux

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| Rust | 2.12 | 153,104 | 986 | 17% 14% 33% 84% |
| C++ g++ | 1.84 | 203,912 | 1315 | 49% 44% 99% 49% |

spectral-norm

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| Rust | **1.98** | 2,244 | 1126 | 99% 100% 99% 99% |
| C++ g++ | 1.98 | 2,268 | 1044 | 100% 99% 99% 99% |

k-nucleotide

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| Rust | 5.38 | 135,192 | 1749 | 64% 73% 89% 91% |
| C++ g++ | 3.90 | 156,216 | 1624 | 72% 72% 71% 96% |

In terms of performance, Rust and C++ perform very similarly. Table 5 showcases some performance tests and in terms of run-times, Rust manages to beat C++ in four of the ten tests. Runtimes are quite similar throughout the table however. In terms of memory usage or the size of the projects featured in the tests of table 5, there does not appear to be a clear winner in these regards and the two languages seem to be very similar.

Overall, the Rust programming language is meant to fill the same role of system-level programming as C++ does but with many proposed improvements. Although the Rust programming language has gained some adoption, it still has not really made its way to the mainstream at least yet.

### 4.1.5 Conclusions from the comparisons

Comparing the C++ programming language to its main alternatives gives a basic understanding of the different programming languages each with their own attributes. The term “general-purpose programming language” can be used to describe all of the featured languages, but despite all of the languages falling under this term, they are used in very different applications.

Despite their similarities, there are a number of crucial differences between all of the languages even in general terms. The Python programming language is perhaps more different than the rest due to it being an interpreted and dynamically typed language. This makes Python more of a scripting language and it is notably higher in the level of abstraction. This in practical terms can make Python easier and more productive for certain applications such as scripting. However, the performance of Python both in terms of runtimes and resource usage is far higher than the other languages meaning that it rules the use of this language out completely in applications where runtimes are imperative or the platform has certain restrictions. The comparison with Python and C++ is therefore not very applicable in many cases due to the languages being designed and used for very different applications. The comparison is however useful when comparing the two language’s merits for things such as education.

Java and C# are in many ways quite similar and comparing the general differences between the two languages is very nuanced. With both languages having many similar ideas and filling similar roles. Comparing C# and Java with C++ however, is not quite as difficult. In general terms, the compilation, memory management and runtime features are some of the more practical differences when comparing C++ with Java and C#. Both Java and C# have a higher level of abstraction. While this can make C# and Java easier to work with in certain applications it also means that their ability to access the computer’s hardware or memory is more restricted. In addition, in terms of raw performance in the benchmarks neither can match the runtimes or low memory usage of C++.

Despite their similarities, the C#, Java and Python programming languages are all suited to more higher-level development and this shows in their overall features such as higher-level of expression which can give them better productivity, runtime features and platform independence. Whereas the C++ programming language is more suited to applications where performance is imperative and lower-level applications where the application necessitates features such as direct access to the computer’s hardware or the environment has certain restrictions. The C++ programming language therefore has more coverage, because it still can be utilized in higher-level applications whereas the Python Java or C# may not be able to function in the same capacity in the lower-level applications. While the C++ programming language can be utilized in the same applications as its newer alternatives, the data in figure 4 suggest that most developers are choosing other alternatives.

## 4.2 Choosing a programming language for education

The purpose of this section is to compare different programming languages for effective education of computer science. In the last section, the general differences between different programming languages were discussed. This section focuses on the attributes of different programming languages and they are compared with one and other to attempt to find out how will they contribute for the education of computer science.

In education of programming, it is important to bear in mind for whom this education is targeted both in terms of age and level of academia. This thesis will focus mainly on education for general engineers and computer science students. The divide between general engineers and computer science students is done because computer science students will undoubtably learn many programming languages but in the case of general engineers it is less likely that they will continue beyond the first language [87]. In addition, general engineers will most likely have less time and interest for programming.

Attributes for the education of programming where discussed in section 2.5 of this thesis. The attributes discussed where simplicity, writability, reliability and fault tolerance, market demand, coverage and extensions. To apply these criteria to the different programming languages showcased, the programming languages must be evaluated with these criteria in mind.

In addition, the mentioned criteria, the requirements for a first programming language can be further specified with more distinct attributes. Choosing an effective first programming language for both general engineers and computer science students is challenging and has many difficulties for the student. There are many difficulties for learning the first programming language and the syntax and structure of the programming language are some of the more distinct factors. The rigid rules in the computer language’s syntax with all the precise rules which must be followed can be overwhelming for students and choosing a simpler or at least syntactically more familiar language would aid the learning process. The structure of programming languages is another unfamiliar attribute for many and is quite unlike other fields of study. [88] Therefore it is beneficial for a programming language to have simplicity in attributes such as functions and parameter passing.

Many of these required attributes for a fist programming language can be described in the following metrics. Directness or linearity, meaning that the user may begin to code without the need for terms and structures which maybe unclear to the user at the beginning. Transparency, which in this context refers to the readability of the code even for someone not familiar with programming. Resistance to errors and ease of recovering and fixing them. The level of abstraction must be suitable for the students. The development environment in terms of both availability and the complexity of setting it up and using it. [62]

### 4.2.1 Examining the criteria selected for C++ in teaching

The C++ programming language has been and is a popular programming language when it comes to teaching. Evaluating the C++ programming language on the criteria mentioned gives some context on how well the programming language is suited for this purpose. This section will first examine the C++ programming language with the attributes discussed in section 2.5 for a general understanding on how well suited the C++ programming language is for education.

The first criteria is **simplicity** and the C++ programming language is not very simple and can be quite overwhelming especially for a beginner. The syntax of the C++ is not very understandable for someone unfamiliar with programming and does not very closely resemble any spoken language. This is the case despite many of the commands used are abbreviations from the English language. The C++ programming language is also quite comprehensive, and although this can be a positive attribute, it can also be quite difficult for a beginner. C++ has seen many updates over the years and the language has grown to a very large size in terms of different practices and operations. With the updates and modernizations however, the older and perhaps outdated operations have not been removed which also is a contributing factor in the language's complexity and reading older C++ code can be challenging even for an experienced user. The static typing of the language is both a good and a bad thing for beginners, because it makes the user be aware of the types being used at a glance, however constructing generic functions can be more challenging with static typing.

The parameter passing is also a point of contention for many beginners in C++. Parameters can be passed in a number of different methods such as value or reference etc. and depending on the method used, will drastically change the actual operation done. This feature is useful in many use cases but can be difficult to grasp for a beginner.

The second attribute is **writability** and the C++ programming language does very well in terms of different methods which can be used to solve problems. The data types offered in C++ are quite various and with the standard template library or STL, many complicated data structures are made accessible and easy to use.

**Reliability and fault tolerance** are points which continue to get better in C++ by many of the new updates focusing on making compile-time checks and overall attempting to make the language more robust. The C++ programming language also provides exception handling and is able to both handle and recover from exceptions. Unfortunately, the C++ programming language does have many possibilities for problems due to the lower abstraction level and the freedom in grants to the user such as accessing the memory. This allows for many exploits such as memory exploits which the user must take into account. To make matters worse, the C++ programming language does not feature runtime features such as a Garbage collector, meaning that a user must deallocate memory manually. These attributes contribute to the performance of the programming language, but for a beginner, they can be problematic. The programming language's performance is not a very useful feature for teaching. In addition, badly written code will not be efficient high performing or in some cases even functional. Although updates to the C++ programming language have made strides to improve these aspects with features such as smart pointers.

**For market demand, community support** and overall availability, the C++ programming language does quite well in all of these. The market demand for C++ is usually quite high due to the widespread use of the language. In addition, many older programs have been written on it so there will be a need to maintain or update legacy code. There are many books and reference material about the C++ programming language and because of the language's long history and wide range of uses, community support and overall guides and references are often easy to find. In addition, tools and integrated development environments for the C++ programming language are free and easily available and many environments have large companies such as Microsoft behind them in the case of programs such as Microsoft Visual Studio. All of these attributes are good for education because the student can find the tools necessary easily and for free and find material or support to aid the learning process.

Coverage and extensions are the last criteria discussed in section 2.5 and for both of these criteria, the C++ programming language does very well. Coverage for the C++ programming language is excellent, because it can be utilized in a very broad range of applications, although more modern languages are perhaps more suited for some of them, for instance web applications. The C++ programming language is a multi-paradigm language and it incorporates many programming styles. This grants the user many freedoms to approach each problem in different ways. With this criteria, the C++ programming language does especially well compared to even more modern equivalents. Extensions for the C++ programming language is decent and there are many libraries and other material available for it.

### 4.2.2 Examining alternative programming languages

This thesis will mainly focus on the C++ programming language. For the interest of brevity, the subsequent programming languages will be examined more briefly.

In terms of **simplicity**, the Java-, C#- and Python -programming languages are all simpler than C++. Java and C# are simpler due to their fewer operations and restrictions to certain abilities when compared to C++. This can be exemplified on how readable the code written on different programming languages is. For instance, Java only has around 50 keywords. The C++ programming language has around 84 keywords which are less like spoken languages and more difficult to understand for someone not familiar with the language. Languages such as Java also have a very consistent conventions when dealing with certain aspects such as exceptions, importing libraries and variable scopes. While C++ often has few rules, but several exceptions to these rules. As an example, the C++ programming language has many ambiguous symbols which may cause confusion to many such as the asterisk symbol *, which may denote multiplication, pointer declaration or an indirection. The Python programming language is however far simpler even than Java or C# and it contains only about 35 keywords. [89] While the syntax of Java and C# both quite closely resemble that of C++, Python manages to be quite different. Table 6 showcases the difference in a rudimentary insertion sort program written in C++, Java and Python. The comparison is meant to showcase the syntactic differences between the different languages both in terms of commands and other symbols used. While C++ and Java use symbols such as semicolons and brackets, Python uses indentations. While there can be differing opinions on how effective or unequivocal the language's syntax is, the Python programming language with its syntax resembles traditional languages more closely and in this particular example, the code written in Python is far shorter.

**Table 6.** Insertion sort code comparison [85]

C++

```
#include <bits/stdc++.h>
using namespace std;

void insertionSort(int arr[], int n)  {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;  }  }

void printArray(int arr[], int n)  {
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl; }

int main()  {
    int arr[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);
    printArray(arr, n);

    return 0;  }
```

Java

```
class InsertionSort {
    void sort(int arr[]) {
        int n = arr.length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
            int j = i - 1;

            while (j >= 0 && arr[j] > key)
            {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key; } }

    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");

        System.out.println(); }

    public static void main(String args[]) {
        int arr[] = { 12, 11, 13, 5, 6 };

        InsertionSort ob = new InsertionSort();
        ob.sort(arr);

        printArray(arr); } }
```

Python

```
def insertionSort(arr):

    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >= 0 and key < arr[j] :
                arr[j + 1] = arr[j]
                j -= 1
        arr[j + 1] = key

arr = [12, 11, 13, 5, 6]
insertionSort(arr)
for i in range(len(arr)):
        print ("% d" % arr[i])
```

A beginner may not be familiar with typical instructions of programming languages, but instead understands typical spoken languages. This would make giving instructions in for instance English far more familiar and easy to grasp for a novice. This can give a beginner an easier avenue into programming because both the higher level of abstraction and the more familiar syntax will let the student start writing functioning code, instead of struggling with unfamiliar commands and symbols. The syntax of Python resembles the English language quite closely to the point where even a non-programmer could quite easily understand what is broadly being done in the code just by understanding English.

In terms of **writability**, meaning how easy it is to create programs for a chosen problem domain with the programming language, the comparison becomes challenging. The coverage of the C++ programming language is and the freedom it grants to the user, gives the user many avenues for solving problems and a greater amount of use cases when compared to the other programming languages featured in this comparison. However, in terms of how easy it is to work with, the C++ programming language can be more complex and challenging with fewer safeguards such as runtime features aiding the process. The same problems which were discussed when examining the languages simplicity are all valid in terms of writability as well. In general, the C++ programming language is more challenging to use but also offers the user more avenues for problem solving.

**Reliability and fault tolerance** are attributes where the newer programming languages start to really outclass the C++ programming language. Because of the higher level of abstraction and run-time features such as the Garbage collection, the Java-, C#- and Python- programming languages are much more capable of avoiding errors and dealing with them if they should occur. In addition, all three are also capable of exception handling.

**For market demand, community support** and overall availability, all of the programming languages featured are quite popular. Java is the most popular programming language featured and development environments are free and readily available for it. Community support and market demand are also high for the Java programming language because of the language's popularity, wide-spread use and wide variety of applications it can be used for. The C#- and Python- programming languages are very popular in mobile- and web- applications. All of the newer programming languages are well suited for web-applications and with web development being one of the original purposes of the Java programming language and Python including many web frameworks such as Django.[89]  And with increasing expansion and demand in these fields, it is safe to assume, these programming languages will only grow in popularity.  There are free development environments easily available and free for both of them and market demand and community support is excellent for both of them. In addition, there are many libraries and ready-made structures available for all of the programming languages.

### 4.2.3 Comparing programming languages for education

Now that the main broad points and attributes of both the C++ programming language and some of its alternatives have been overviewed, some additional analysis can be done to look at the overall picture. Evaluated on the criteria discussed in section 2.5, the

C++ programming language does do quite well on many of the mentioned criteria for educational purposes. However, some of the mentioned criteria showcase many of the drawbacks of C++ and the age of the language does start to show when comparing it to its more modern alternatives. In section 4.2 some additional criteria were discussed in choosing a first programming language.

The C++ programming language can now also be further analysed as a first programming language with the additional criteria. These criteria are directness or linearity, transparency and resistance to errors and ease of recovering and fixing them.

In terms of **directness and linearity**, the C++ programming language when comparing it to languages such as Python does not do all that well. This is due a number of different attributes. The C++ programming requires the user to write some code to set up basic functionality, which for a novice may prove to be confusing. In addition, the overall complexity with understanding things such as different types of parameter passing, static typing and an overall more challenging syntax, can be overwhelming for a beginner.

**Transparency**, meaning how well a code can be read by someone not familiar with programming, is another point where languages such as Python are easier for a novice. This is due to the more simplistic syntax both in terms of symbols and the wording of the commands used. In addition, the length of the code written in C++ is often longer.

**Resistance to errors and recovering from them**, is where the C++ programming language has many issues, especially for beginners. Although the C++ programming languages features mechanics such as exception handling and is capable of recovering from errors, this is an area where all of the other programming languages do better than C++. This is in part due to the higher level of abstraction and runtime features, which aids in automatically dealing or preventing certain possible errors such as memory leaks.

Analysing the C++ programming language as a first programming language for computer science students and perhaps the only programming language for general engineers using these criteria can give a somewhat different picture. In terms of readability or understanding the code for a non-programmer, the C++ programming language does not do well compared to other programming languages featured in the comparison. For a beginner, the Python programming language is perhaps the simplest and easiest to understand, but both Java and C# are still in certain ways simpler than C++. In addition, to get any code running on C++, there are many lines of code necessary to complete even the most rudimentary program. This can further hurt the C++ programming language when compared to programming languages with a higher level of abstraction, for instance Python. Resistance to errors and recovering from them is also more troublesome for C++. In addition to possible low-level memory exploits, the user must keep track other factors, such as sizes of integers and other mathematical errors such as not dividing with 0. In addition, the lack of run-time features necessitates the user to be much more aware and responsible for finding and handling errors. This analysis will be further explored in chapter 5 of this thesis.

## 4.3 C++ in software development

The purpose of this section is to see how the C++ programming language fits in modern software development. What applications is C++ suited for and how well does it fit in more modern software development ideas such as agile development.

Bearing in mind the history of software development discussed in section 2.4 of this thesis, it is easy to understand why C++ became such an influential, widely adopted and popular programming language. In an era where computers where not connected to the world wide web and performance of the hardware necessitated extreme levels of efficiency, the C++ programming language stands as an excellent compromise. With the performance of lower level programming languages such as C and a language with enough expressivity and richness of features to rival even modern programming languages. These attributes made C++ a very appealing choice for developers. But with paradigm shifts in computing from both performance of hardware, rapid development and prototyping, is C++ still and will it be in the foreseeable future a relevant programming language or will it be replaced with more modern alternatives?

### 4.3.1 What is C++ used for today

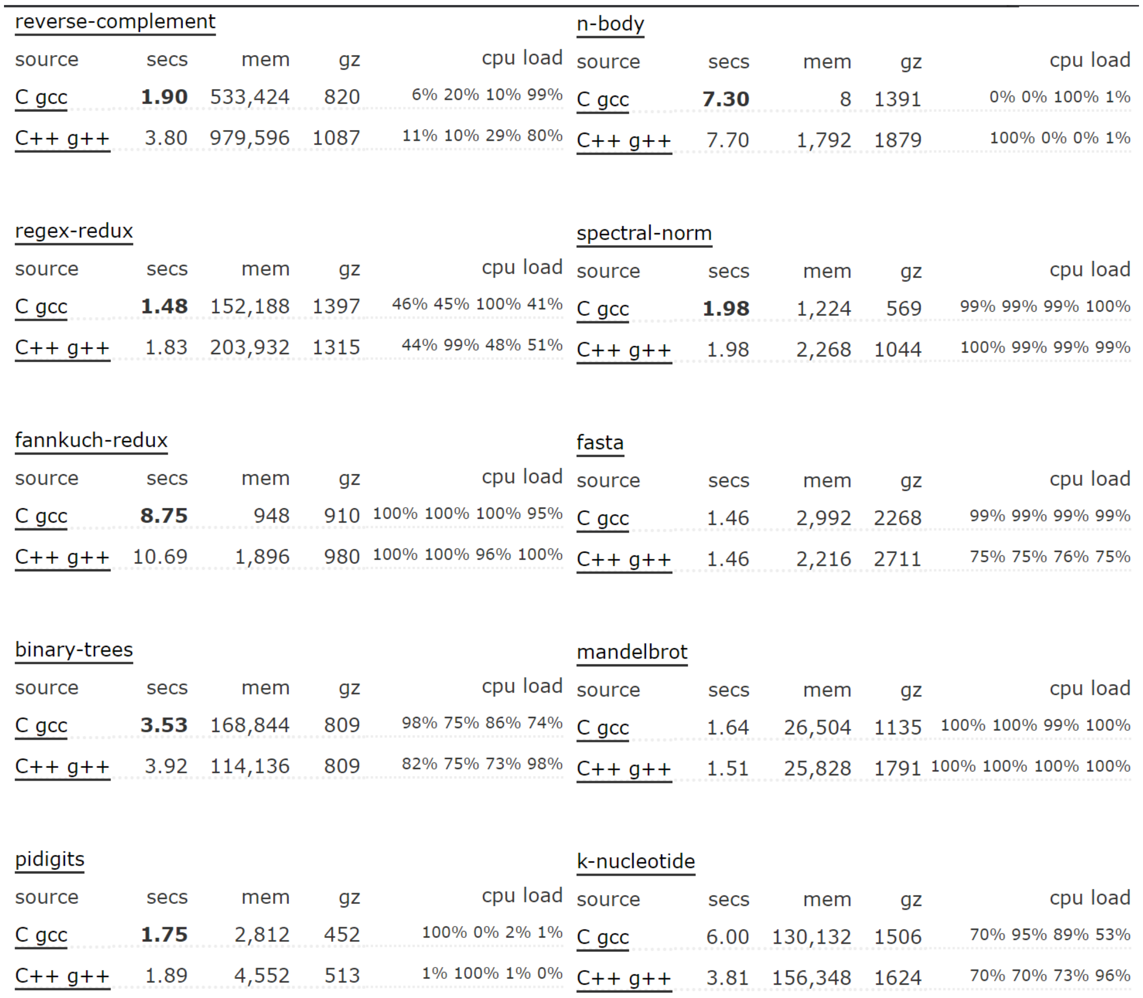
In addition to education, the C++ programming language is heavily utilized today in many applications. Because of the C++ programming language's high performance, relatively low level of abstraction and the ability to be compiled for virtually any platform, the C++ programming language is ideal for many use cases. Because the C++ programming language is a general-purpose programming language, it can be utilized in virtually anything. The following are some examples where it has and continues to be used in software development. [63]

Applications such as operating systems which require the user to access the computer's hardware and allow for manipulation of memory. All of the major operating systems such as Linux, Microsoft Windows or MacOS rely heavily on C++. In addition, many of Microsoft's major programs have been written nearly entirely on C++ such as Internet explorer, Visual studio or Office. [64]

Game engines are another major application for C++. Game engines such as the Source engine from Valve, Unity (the game engine itself is written in C++, but Unity Scripting API is C#), Frostbite engine from Dice, The Unreal Engine and many others are all written in C++. Some of the reasons the C++ programming language is very popular in game engines and game making in general is that the language allows direct control of hardware and the ability to optimize performance.

Another very prevalent application of the C++ programming language is its use in embedded systems. Typically, lower level programming languages such as C or even assembly languages have been used for embedded systems programming, but with the price of 32-bit microcontrollers being under a dollar, the C++ programming language is now a viable option for this application. A common school of thought is that the lower the abstraction layer of a language, the higher performance and lower overhead it has. [65] This is true to a certain extent, but if examined the contents of table 7, we can see how this is not always the case. From these particular set of tests catalogued in table 7, we can determine that in terms of both runtime performance and the size of the programs denoted as “gz” in the table, there is very little difference. Memory usage is quite similar as well apart from the n-body test. The performance is remarkably similar and in two test cases being exactly the same. In addition, it is noteworthy that in some cases the C++ programming language managed to run even faster than C.

**Table 7.** C++ and C performance test [66]



reverse-complement

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C gcc | **1.90** | 533,424 | 820 | 6% 20% 10% 99% |
| C++ g++ | 3.80 | 979,596 | 1087 | 11% 10% 29% 80% |

n-body

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C gcc | **7.30** | 8 | 1391 | 0% 0% 100% 1% |
| C++ g++ | 7.70 | 1,792 | 1879 | 100% 0% 0% 1% |

regex-redux

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C gcc | **1.48** | 152,188 | 1397 | 46% 45% 100% 41% |
| C++ g++ | 1.83 | 203,932 | 1315 | 44% 99% 48% 51% |

spectral-norm

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C gcc | **1.98** | 1,224 | 569 | 99% 99% 99% 100% |
| C++ g++ | 1.98 | 2,268 | 1044 | 100% 99% 99% 99% |

fannkuch-redux

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C gcc | **8.75** | 948 | 910 | 100% 100% 100% 95% |
| C++ g++ | 10.69 | 1,896 | 980 | 100% 100% 96% 100% |

fasta

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C gcc | 1.46 | 2,992 | 2268 | 99% 99% 99% 99% |
| C++ g++ | 1.46 | 2,216 | 2711 | 75% 75% 76% 75% |

binary-trees

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C gcc | **3.53** | 168,844 | 809 | 98% 75% 86% 74% |
| C++ g++ | 3.92 | 114,136 | 809 | 82% 75% 73% 98% |

mandelbrot

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C gcc | 1.64 | 26,504 | 1135 | 100% 100% 99% 100% |
| C++ g++ | 1.51 | 25,828 | 1791 | 100% 100% 100% 100% |

pidigits

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C gcc | **1.75** | 2,812 | 452 | 100% 0% 2% 1% |
| C++ g++ | 1.89 | 4,552 | 513 | 1% 100% 1% 0% |

k-nucleotide

| source | secs | mem | gz | cpu load |
|---|---|---|---|---|
| C gcc | 6.00 | 130,132 | 1506 | 70% 95% 89% 53% |
| C++ g++ | 3.81 | 156,348 | 1624 | 70% 70% 73% 96% |

While the performance difference between the C and C++ programming languages are negligible, in many ways this makes the C++ programming language a much more powerful tool when compared to C because of the higher level of abstraction and added features with C++. In addition, the C++ programming language is almost exactly a superset of C, meaning many pieces of code written in C can be directly compiled into C++ [65].

There are many concerns when writing programs for embedded systems such as the following. Reliability, because failure can be very expensive. Limited resources, because of the limitations of embedded systems, there is often a very limited supply of memory, network connections or file handles to name a few. Requirements for real-time response. In addition, embedded systems have often very little downtime, concurrency and may need to run reliably for a very long time. [67] The C++ programming language manages to work very well in this application because it has provisions to deal with all of the mentioned attributes.

In addition to the mentioned specific uses, the C++ programming language is utilized quite heavily in applications where performance is imperative and there is a need to access the computer’s hardware directly. Some general examples of these applications are high frequency finance, 3D animation software and modelling and rendering, web browsers such as Google Chrome and Mozilla Firefox have been written in C++, database access programs such as MySQL. In addition, because of the C++ programming language’s age and long-lasting popularity, another very frequent use of C++ comes from updating and maintaining existing software written in C++. [68]

### 4.3.2 C++ for the waterfall model of development

A very common and traditional software development process is commonly known as the “waterfall” model for software development. This model is depicted in figure 8 for visual aid. The waterfall model is a linear and sequential model for software development, meaning that each stage must be completed before the next one may begin. In addition, each stage of the project is reviewed to ensure that they meet the specified requirements. The stages of this process are requirement gathering and analysis, system design, implementation, testing and development of system and maintenance. [79]

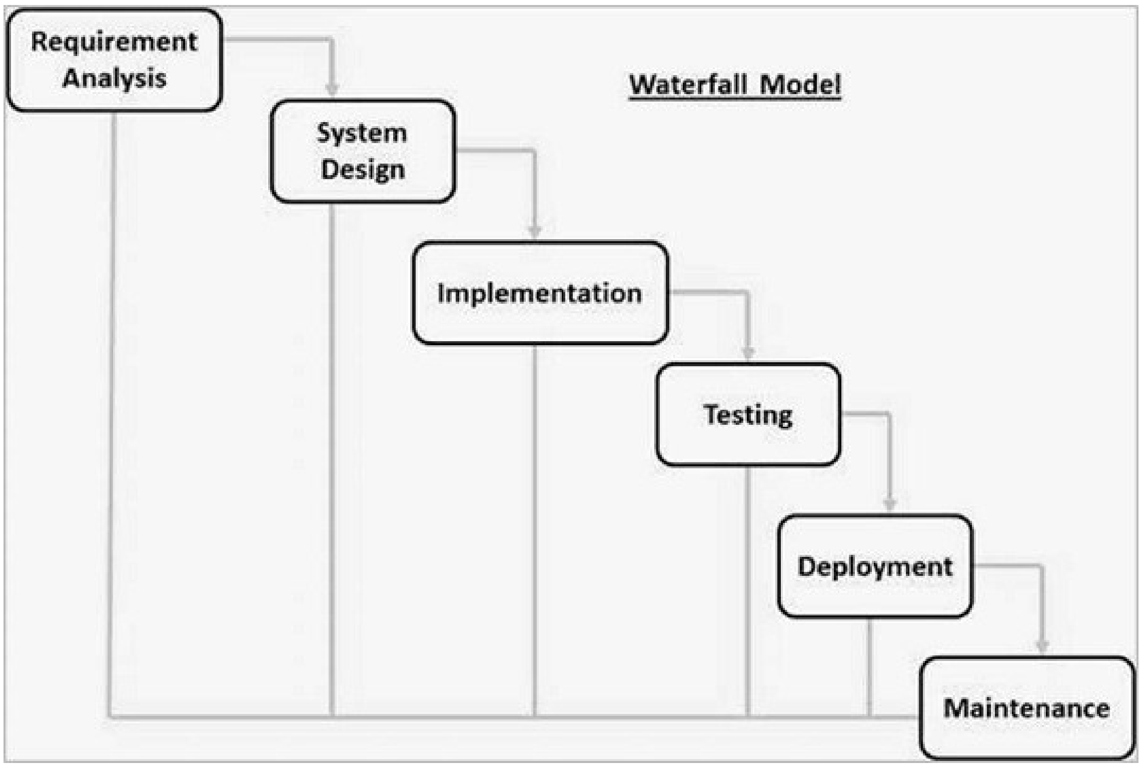


**Figure 8.** Waterfall Model [79]

This model is meant to be simple, easy to understand and use with clearly defined stages and well-defined milestones. In addition, this process facilitates easy task arrangement and good documentation. This process however can have certain drawbacks. Due to the rigid and well-defined delineation between the structures, this process cannot accommodate changing requirements. Adjusting of the projects scope during the process can also be troublesome. The sequential nature of the model only produces working software at the very end of the process and overall, this model is not ideal for complex or long projects due to difficulties in measuring progress within each stage. [79]

It is important to note, that a development process can be used with any programming language. This section is meant to examine in theory, how well the C++ programming language facilitates a given system of development and how well it is suited to it.

Examining how well the C++ programming language would work with the waterfall model for software development requires analysis of each section of the model. This is done to examine how well this model works with C++. Due to the segmented nature of this model, each section can be examined separately and then the entire model can be analysed as a whole. For the requirement gathering and analysis stage, the overall requirements of the software are made and requirement specification documents are written. The programming language does not affect the process in this stage.

For the system design stage, the system design is prepared. This is done to specify the hardware- and system- requirements and the overall system architecture is defined. This works well with the C++ programming language. This is due to the language's large scope and coverage, meaning there are many alternative ways of achieving the required functionality and the best approaches can be selected to fit each project. In addition, if the project scope and platform is known, the appropriate frameworks, libraries and compilers can be selected. [84]

The implementation stage segments the program into small sections which are created and tested. With the C++ programming language's coverage, overall features and available support, this process is easily facilitated. With unit testing having many readily available frameworks. In addition, with the proper documentation and testing, the complexity and possible vulnerabilities of the C++ programming language can be dealt with.

In the integration and testing stage, the smaller pieces of the software are integrated to construct the project. During this stage, the entire software is then tested as a whole instead of just unit tests. This stage can be problematic with the C++ programming language if the size of the project is large. There are many reasons which contribute to this and some are due to the language's complexity. With multiple people working on the project, the integration of the separate source codes can have certain incompatibilities and linking errors. However, this can be advantageous, because certain errors can then be found and dealt with at the linking stage. This problem can be more troublesome with interpreted languages, where compatibility issues are discovered at runtime instead of the linking stage. These possible challenges can be mitigated with appropriate compilers but the issues with code dependencies due to the compiler's optimizations can be troublesome. [80] The C++ programming language is also an object-oriented language. Object orientation is another point where problems may arise. This is in part to do with the potential number of control flow paths with object-oriented systems which may grow exceedingly large and difficult to manage or test. [81]

For system testing, projects done on the C++ programming language may also prove to be quite difficult and there are certain attributes of C++ which makes system level testing challenging. The build speeds of projects done on C++ can be quite slow, especially on larger projects. This is due to the necessity of building and linking the entire project comprising of static project libraries and third-party libraries into a single large structure to create an executable. In addition, the dependencies may once again have issues. There are other features of the C++ programming language which may also hinder system testing such as the weak type system which allows any pointer to be cast to any other pointer. In addition, C++ is a compiled language and compiled languages can have certain tendencies for errors. With compiler optimized code also having the possibility of creating troublesome errors. [82]

The deployment stage is when the product is deployed in the customer's environment or released into the market. Deploying a project done on C++ is different depending on the platform and environment used. For instance, deployment with the Visual C++ requires the application and the required library files on the target system. This system enables three different types of deployment, which are central deployment where the files are located into the windows directory, local deployment where the files are located in the application and static linking where the library code is bound to the application. [83] Deployment with the C++ programming language as long as the platform is known ahead of time, should be relatively simple with different choices to help suit each situation.

The final phase is maintaining the software. The beginning of the maintenance phase is to fix issues which may arise once the project has been deployed in the client's environment. This stage for a project done on the C++ programming language differs with the deployment method used. With central deployment and with the library files being located in the Windows directory, Windows update service can update them automatically. With local deployment the libraries must be redeployed if they are updated. With static linking, the application must be recompiled and redeployed. [83]

Developing software with the C++ programming language while utilizing the waterfall software development model should in theory work quite well. This structured model has many attributes which play into the C++ programming language's strong sides while taking into account certain possible draw backs. This is likely at least in part due to the fact that during the time the waterfall model for software development was prominently used, the C or C++ programming languages where at the height of their popularity.

### 4.3.3 C++ in agile development

Section 2.4 of this thesis showcased a brief history of software development and the rise of agile methods in the early 2000s. Agile methods are very popular in modern software development and for a programming language to have as much reach as possible, it is important to be suited for a modern development environment. The purpose of this sub section is to examine how well the C++ programming language fairs in agile software development.

The manifesto for agile development favours individuals and interactions over processes and tools, working software over comprehensive documentation, collaborating with the customer over contract negotiation and responding to change over following a plan [69]. In addition, the agile manifesto contains twelve principles to support this process. For the sake of brevity, these twelve principles are not listed in this thesis but the main idea of them is to support the process of software development through interaction with the customer, iterations and rapid adaptiveness to change.  There are many different variants of agile methods such as SCRUM, Test-Driven Development, Extreme Programming and many others, but the main principles behind them are to adhere to the methods and ideas of the agile way of software development.

It is also important to note, that agile development in many ways is not dependent on a programming language, but a set of ideas for software development. If a project necessitates the use of a specific programming language, the agile process can be utilized, whatever this language may be. The purpose of this section is to merely examine how well do the general features of C++ enable the facilitation of agile methods and how well it does in a general sense for this purpose.

In a general sense, the C++ programming language can certainly be utilized in an agile development process. Because of the language's large coverage and the ability to used in just about any purpose, the suitability for any given purpose has to be further analysed.

Section 3.3 of this thesis overviewed certain ideas on how to choose programming languages and the idea of hard- and soft- choices was used. This section will focus mainly on the soft choices, meaning that a language is chosen because of preference and not due to constraints or mandates. Section 3.2 of this thesis also went over criteria for studying such aspects as they fall under the qualitative research methods and these criteria are context, credibility and confirmability. With these parameters in mind, the general comparison of C++ with other programming languages for agile development can be done.

Section 2.4.2 of this thesis overviewed the productivity of software development. There are many different factors which contribute for the actual productivity of software such as the skill of the programmers, the tools used, team cohesion and many other factors which models such as the COCOMO and COCOMO 2 take into account. This thesis however will focus solely on the programming language used and treat all other factors as constants.

The C++ programming language can function very well as an alternative to many lower-level programming languages. For agile development, using the C++ programming language as an alternative for lower-level languages such as C, Cobol or even in some cases assembly languages, can be far better because of the flexibility and the expressiveness when compared to those languages. Table 1 showcases that C++ could be up to 2.5 times more productive when compared to C. This would give the C++ programming language an advantage for both creating working software quicker and having far better adaptiveness to change. In addition, aspects such as the object orientation of C++ supports reuse of code which may also improve productivity. Platform dependencies can be more challenging for C++, however in overall use the higher level of expression can alone make C++ a favourable choice for many. In addition, higher level programming languages often offer better support for testing. Writing testable code especially in more modern C++ is facilitated with many modern tools and easy to find practices such as Microsoft's .NET frameworks for unit testing [70].

Comparing the C++ programming language to higher-level languages however, makes the decision far more difficult. Some newer programming languages, such as Java offer the user the luxury of platform independence, making them more flexible. This can be achieved with C++, but the process takes more effort and a started project has to be partially re-done if the platform requirements were to change, hindering the C++ programming language's agility in this regard. The lower layer of abstraction will also be a hinderance with C++, because the user must write more code to achieve the same functionality when compared to certain newer programming languages such as Python. This is showcased in table 1 where C++ is 2.5 times more productive then C while Python is 6, making Python more than twice as productive than C++. The build speeds of projects done on C++ can be quite slow, especially on larger projects and this would make iterative development troublesome with C++.[82] In addition, because of the lower level of abstraction of the C++ programming language, there is a possibility with memory leaks or other lower level errors which the user must take into account. [71] This makes testing an imperative process for C++ whereas many of the lower-level testing can be subsided with newer memory-safe programming languages.

### 4.3.4 Improving the C++ programming language

The beginning of this thesis overviewed the development history of the C++ programming language. Even the brief overview of the improvements showcased that the C++ programming language has grown to be a very feature-rich programming language with many updates and upgrades to make the language more robust and easier to work with. In some ways, even in practical use, modern C++ is nearly an entirely different programming language to its original version. In addition, the C++ programming language is well supported and new standards are meant to be released every third year.

The C++ programming language has many benefits but unfortunately the language also has many problems as well. Some of these problems are the ability for the user to allocate memory on the heap, older or outdated features which may have vulnerabilities and the overall complexity of the language.

The obvious solution for this would be to fix the problematic attributes. Trying to improve the C++ programming language is however perhaps more difficult than improving most other programming languages. This is in part to do with the language’s coverage. As an example, the C++ programming language lacks run-time features and many safeguards when it comes to accessing memory. The easy solution for this would be to limit the user’s access to the memory and add run-time features. This however would create a lot of problems, because although higher-level applications could greatly benefit from many of the said features, the language’s usage for lower-level applications would then become difficult or even impossible. Many of these additions would also add to the language’s overhead which would decrease the performance and compromise the minimal memory requirements of the language. Improving the C++ programming language is very delicate, because any changes could render the programming language worse in some other crucial application.

# 5. ANALYSIS

The purpose of this chapter is to analyse the study done and results found in this thesis. The results are analysed in separate sections following the study done in the previous chapters.

## 5.1 Analysis of C++ in modern computer science

In this section, the different applications and attributes of the C++ programming language are analysed to further realize how well it is suited for each individual purpose. This is done for the purposes of drawing conclusions to the study and perhaps suggest improvements or alternatives to the C++ programming language.

### 5.1.1 Analysis of C++ in education

Section 4.2 of this thesis studied programming languages for the purpose of education for both computer science students and general students of engineering. The study conducted showcased many positive and negative attributes of C++ as a programming language for education. Upon further analysis of the C++ programming language for education it can be stated that C++ is not perhaps the best first programming language for general engineering students because of the language's complexity and lower level of abstraction. The C++ programming language requires a moderate amount of code to set up general functionality and this may prove counter-productive to those who perhaps may not have any knowledge of what computer programming is. In addition, the complexity and the overall features of C++ can be overwhelming for even a seasoned programmer let alone a beginner. Newer and more higher-level programming languages such as Python are much easier and more forgiving for a beginner.

For students of computer science however, there are many benefits of still learning C++. Because of the complexity and the overall features, the C++ programming language can easily bridge the gap between lower- and higher-level programming languages. The complexity and the overall features can work in the C++ programming language's favour in this regard. Because of the complexity and the lower level of abstraction in C++, a student who can learn this language will be more familiar with the internal workings of programming languages with regards to things such as memory addresses, different forms of parameter passing and type systems just to name a few. In addition, if a student learns a programming language with more complexity and a lower level of abstraction, it is then far easier to transition to a higher-level programming language rather than doing this process the other way around. The coverage of C++ can also give a student a good overall understanding of programming in general.

In conclusion, it is better for both general engineers and computer science students to first work with an easier language such as Python. In addition to being an easier language, Python is very prevalently utilized as a scripting language for web applications and even many scientific purposes such as mathematics or artificial intelligence. This would perhaps give general engineers many practical uses for the language other than just as a tool for learning programming. For students of computer science, the Python programming language would be a great starting point, however learning a more comprehensive system-level programming language such as C++ would certainly give the student a much more thorough understanding of programming.

### 5.1.2 Analysis of C++ in software development

The C++ programming language has in many ways cemented itself in software development. This is in-part to do with existing code and traditions. Due to many existing programs being made in C or C++, the maintenance and update of these codes will be a very common practice for the foreseeable future. In addition, the sheer amount of existing support and knowledge of the C++ programming language will ensure its usefulness. The C++ programming language is also ever changing and evolving. With many new updates which have made the language more expressive, secure and easy to work with.

There are however certain drawbacks to the C++ programming language for software development. Section 3.3 of this thesis discussed a set of ideas for choosing programming languages and the decision-making process was broken down into soft- and hard-choices. With this in mind the hard choices, meaning a choice which is made out of necessity, favour the C++ programming language quite heavily in many modern software development projects. This is often the case with many applications where runtimes and memory usage are crucial or there is a necessity for a lower-level programming language. Applications such as embedded systems, operating systems or game engines are an example of such applications where performance is imperative and there are requirements for accessing hardware directly.

However, in the case of soft choices, meaning choices made from preference, there are many reasons to choose other programming languages instead of C++. Because of the lower level of abstraction of the C++ programming language, many newer languages can achieve the same functionality with far fewer lines of code and with far less risk for certain types of errors such as memory leaks. This can put C++ in more of a niche role for modern software development. Because if a language is only used out of necessity rather than preference it can be just relegated into very certain and specific roles and lose popularity from mainstream development. With embedded systems and many lower level hardware becoming more powerful, the C++ programming language is becoming ever more useful for these types of cases. In terms of popularity, higher level programming languages are becoming more favourable in projects where performance is not quite so imperative but for performance centric applications, the C++ programming language is gaining popularity in lieu of lower level languages such as C or Assembly languages.

Section 2.4 of this thesis overviewed a brief history of software development and how both the hardware and development practices have changed. The C++ programming language emerging in 1979 is very much a product of its time and fit the needs of developers at that period of time. Looking at figure 4 from section 4.1 of this thesis which showcased the popularity of certain programming languages, we can see that the C++ programming language is losing overall popularity and newer and more expressive programming languages are gaining it. Losing overall popularity however is perhaps not so important because the entire industry of software development has grown. In addition, the C++ programming language is still a very popular language overall instead of some programming languages from the same era such as Fortran or Cobol which disappeared entirely from the mainstream. The focus of the language is however changing from more overall use to system level programming, where its popularity is growing. On system level programming, the C++ programming language is a very compelling option but in recent years, certain newer programming languages such as Rust are also becoming valid alternatives.

# 6. CONCLUSION

This thesis has overviewed the main attributes and development history of the C++ programming language and examined its usefulness in both modern software development and education for the purposes of finding out how suitable it is in modern software development, education and for the future. With newer standards and overall development, the C++ programming language has grown to be a very feature rich and capable programming language incorporating even the paradigm of functional programming. Overall, the C++ programming language is more modern and robust than ever while still maintaining fast runtimes and minimal overhead. The C++ programming language is however quite complex, with new additions incorporating older and perhaps outdated features as well, meaning that the language has expanded greatly. In addition, the level of abstraction of the C++ programming language is quite low when compared to many more modern languages. This makes the C++ programming language less favourable in certain applications.

For educational purposes, this thesis concludes with stating that the C++ programming language is not a very good first programming language for general engineers or perhaps even students of software engineering. With the C++ programming language's low level of abstraction and difficult syntax hindering the student's ability to learn. Newer more expressive and more forgiving programming languages such as Python are better suited for this purpose. For software engineers however, learning the C++ programming language will give a valuable insight into programming languages. The overall features of the C++ programming language and both the freedom it grants and the responsibilities it requires from the user are all very useful tools for learning. It is therefore preferable to start the learning process with a more simple scripting language and then move to a more comprehensive system-level programming language.

The C++ programming language has historically been a very influential programming language dating back more than four decades. In many ways however, the C++ programming language is also a product of its time and lacks many of the features and attributes found in more modern programming languages. Attributes and features such as the Java programming language's platform independence or run-time features such as the garbage collector found in many more modern languages such as Java or C#. In addition, the level of abstraction, complexity and possibility for certain exploits have made the C++ programming language less favourable for many purposes in higher-level software development. Language's such as C#, Java or Python are more expressive, more easily portable and often do not need as much testing for lower-level exploits. As a result, many modern developers favour newer programming languages for higher-level applications over C++ where there is room for choice. Advances in computing and changes in development however are not just happening on the higher-level applications, but lower cost processors in embedded systems are also becoming more powerful. This has made the C++ programming language a candidate for replacing lower-level languages such as C or even assembly languages. In system-level applications and applications where performance is imperative, the C++ programming language stands well above both lower level programming languages in terms if features and expressivity as well as above the mentioned newer programming languages in terms of performance.

It is perhaps unwise to study a programming language’s popularity and focus more on how useful a language is or can be. In many ways the need for a “general-purpose” programming language can be over-estimated and even looking at overall popularity figures is perhaps not useful. The field of software development has expanded and with this expansion the requirement to find a suitable tool for different use cases has also grown. The choice of which programming language is used is more often than not a compromise which is often driven by external factors such as existing knowledge and frameworks instead of just the merits of the programming language itself.

In terms of software development, the popularity of the C++ programming language has shifted more towards the system-level applications where it is the most popular programming language and the tool of choice to a very large segment of the development industry. There are however attempts to make new and more user-friendly programming languages for this very area such as Rust which may at some point become a valid alternative to C++. Although the overall popularity of the C++ programming language may have decreased, the language is unlikely to fade into obsolescence like Cobol or Fortran due to the language’s overall qualities, frequent updates and improvements, existing knowledge, traditions and maintaining older software.

Writing this thesis to me personally has given an overview and a valuable insight into not only the C++ programming language, but the field of computer science. With many of the sections of the thesis requiring overviewing many subjects such as history of software development, attributes for education of programming and in-depth analysis of a programming language’s internal workings among other things. The overall focus of the work however was to examine how well the C++ programming language fits in modern computer science as a general-purpose programming language and how suited it is for developers and education. This however, turned out to be quite a muddled subject with difficulties finding usable data for internal decision-making processes of choosing programming languages within industries and what attributes contribute for effective learning of programming. In addition, the realisation that requiring a single programming language to be viable in most applications is not perhaps the best idea. This changed the focus of the thesis. With the expansion of the field of computer science, the tools necessary to deal with each application or situation necessitates more compromises and using different programming languages. The overall focus of the thesis has become more of an overview of existing knowledge and analysis of known attributes instead of focusing on a single subject.

For future work, many of the overviewed fields can be further expanded and studied. Some more interesting ideas such as further improving the C++ programming language to suit more modern development both in terms of embedded systems and higher-level development. The field of education of programming could also be far more thoroughly studied with actual testing and proper result analysis. In addition, the effectiveness of newer programming languages with less support and fewer users such as Rust in modern development as an alternative to C++ could be an interesting subject.

# REFERENCES

[1] CHM. Bjarne Stroustrup. 2015 Cited 09.10.2019 Available: https://www.computerhistory.org/fellowawards/hall/bjarne-stroustrup/

[2] Stroustrup B. Tour of C++. Pearson Education; 2013

[3] S Pramanick. History of C++. Cited 09.10.2019. Available https://www.geeksforgeeks.org/history-of-c/

[4] Fendadis John. History of C++ language. 15.02.2018 Cited 10.10.2019. Available https://www.tutorialspoint.com/History-of-Cplusplus-language

[5] Sroustrup B. C++11- the new ISO C++ Standard, Modified 19.08.2016. Availabla http://www.stroustrup.com/C++11FAQ.html Cited 13.10.2019

[6] Bancila M. Tan C++11 Features Every C++ Developer Should Use 02.04.2013. Available https://www.codeproject.com/Articles/570638/Ten-Cplusplus11-Features-Every-Cplusplus-Developer#lambdas Cited 13.10.2019

[7] Köppe T. Changes between C++11 and C++14. 08.10.2018 Available: http://open-std.org/JTC1/SC22/WG21/docs/papers/2018/p1319r0.html Cited 14.10.2019

[8] Köppe T. Changes between C++14 and C++17 DIS. 02.04.2017 Available: https://isocpp.org/files/papers/p0636r0.html Cited 17.10.2019

[9] Stroustrup B. C++20: C++ at 40 17.09.2019

[10] McCandless K. What is Computer Programming? 13.06.2018 Available: https://news.codecademy.com/what-is-computer-programming/ Cited 21.10.2019

[11] Bolton, David What Is computer Programming? 03.07.2019 Available: https://www.thoughtco.com/what-is-programming-958331 Cited 21.10.2019

[12] Damasevicius, Robertas. 2006. On the quantitative estimation of abstraction level increase in metaprograms. Comput. Sci. Inf. Syst.. 3. 53-64. 10.2298/CSIS0601053D.

[13] freeCodeCamp Available: https://guide.freecodecamp.org/computer-science/compiled-versus-interpreted-languages/ Cited 27.10.2019

[14] D. Munoz Trejo. How C++ Works: Understanding Compilation. Available https://www.toptal.com/c-plus-plus/c-plus-plus-understanding-compilation Cited 27.10.2019

[15] The C++ compilation process http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

[16] Stevanovic M. and I. Books24x7, Advanced C and C++ Compiling. 2014.

[17] Karinsky L. How C++ Works: Understanding Compilation 31.08.2017 Available: https://www.freelancer.com/community/articles/how-c-works-understanding-compilation# Cited 28.10.2019

[18] Butterfield A, Ngondi GE, Kerr A. A Dictionary of Computer Science [Internet]. Oxford University Press; 2016. Available from: http://www.oxfordreference.com/view/10.1093/acref/9780199688975.001.0001/acref-9780199688975

[19] Albatross. An Overview of Programs and Programming Languages. Available: http://www.cplusplus.com/info/description/ Cited 06.11.2019

[20] C++ Type System (Modern C++) Available: https://docs.microsoft.com/en-us/cpp/cpp/cpp-type-system-modern-cpp?view=vs-2019 Cited 08.11.2019

[21] Smyth P. An Introduction to Programming Paradigms. Available: https://digitalfellows.commons.gc.cuny.edu/2018/03/12/an-introduction-to-programming-paradigms/ Cited 13.11.2019

[22] Rouse M. imperative programming. Available: https://whatis.techtarget.com/definition/imperative-programming Cited: 14.11.2019

[23] Bartoníček J. Programming Language Paradigms & The Main Principles of Object-Oriented Programming. CRIS - Bulletin of the Centre for Research and Interdisciplinary Study. 2014;2014(1):93-9

[24] Velare P, V. Programming paradigm. 21.08.2017. Available: https://whatis.techtarget.com/definition/imperative-programming Cited 14.11.2019

[25] Stroustrup B. The C++ Programming language Fourth edition. Pg. 11.

[26] Lambda Expressions in C++. 05.07.2019. Available: https://docs.microsoft.com/en-us/cpp/cpp/lambda-expressions-in-cpp?view=vs-2019 Cited: 16.11.2019

[27] Richmond D, Althoff A, Kastner R. Synthesizable Higher-Order Functions for C++. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2018;37(11):2835-44.

[28] Anggoro W. Learning C++ Functional Programming. Packt Publishing; 2017.

[29] Gregoire, M., Solter, N. & Kleper, S., Professional C++, 3nd. Wiley, Hobo-ken, N.J, 2011. Cited 18.11.2019

[30] Yost M. A Brief History of Software Development 25.01.2018 Available: https://medium.com/@micahyost/a-brief-history-of-software-development-f67a6e6ddae0

[31] A Brief History of Software Engineering. Available: https://www.vikingcodeschool.com/software-engineering-basics/a-brief-history-of-software-engineering Cited: 18.11.2019

[32] Javin P. The 10 Most Influential Programming languages of the Last 50 Years and Their Creators. Available: https://medium.com/better-programming/the-10-most-influential-programming-languages-of-the-last-50-years-and-their-creators-6559bb9ce224 Cited 18.11.2019

[33] Jared King's "The History of Software" Available: https://learn.saylor.org/mod/page/view.php?id=12353 Cited: 19.11.2019

[34] 10 Skills necessary for coding. Available: https://www.computerscience-zone.org/10-skills-necessary-coding/ Cited: 24.11.2019

[35] Sheikh, Ghazala & Islam, Noman. 2016. A qualitative study of major programming languages: teaching programming languages to computer science students. International Journal of Information and Communication Technology.

[36] Lucas-Alferi D. 2015 Marketing the 21st Century Library: 3 Marketing plan research and assessment

[37] What Is Qualitative Research? Available: https://nursing.utah.edu/research/qualitative-research/what-is-qualitative-research.php Cited 28.11.2019

[38] Spinellis, D, Choosing a programming language, IEEE Software, vol. 23, no. 4, 2006, pp. 62–63.

[39] TIOBE Index for November 2019. Available: https://www.tiobe.com/tiobe-index/ Cited: 01.12.2019

[40] Gupta L. What is Java programming language? Available: https://howtodoinjava.com/java/basics/what-is-java-programming-language/ Cited: 02.12.2019

[41] Features of Java. Available: https://www.studytonight.com/java/features-of-java.php Cited: 03.12.2019

[42] Compilation and Execution of a Java Program. Available: https://www.geeksforgeeks.org/compilation-execution-java-program/ Cited 03.12.2019

[43] Arnolds L, Gosling J, Holems D. 2005. The Java Programming Language, Fourth Edition.

[44] Gouy, Isaac. The Computer Language Benchmarks Game.Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame Cited: 04.12.2019

[45] Gouy, Isaac. The Computer Language Benchmarks Game. Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/gpp-java.html Cited: 04.12.2019

[46] C++ vs Java. Available: https://www.educba.com/c-plus-plus-vs-java/ Cited: 05.12.2019

[47] Dynamic Memory Allocation in C++. Available: https://www.studytonight.com/cpp/memory-management-in-cpp.php Cited: 07.12.2019

[48] Younan Y, Joosen W, Piessens F. 2004. Code Injection in C and C++ : A Survey of Vulnerabilities and Countermeasures.

[49] The Last Stage of Delirium Research Group. 2002. Java and Java Virtual Machine security vulnerabilities and their exploit techniques

[50] Chand M. 08.07.2019. What Is C# and What is C# Used For. Available: https://www.c-sharpcorner.com/article/what-is-c-sharp/ Cited: 08.12.2019

[51] Kostadinova K. 10.06.2019. Programming paradigms of C#. Available: https://blog.softuni.org/2019/06/programming-paradigms-of-c-sharp/ Cited 08.12.2019

[52] Introduction to the C# Language and the .NET Framework. 20.07.2015. Available: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework Cited 08.12.2019

[53] Gouy, Isaac. The Computer Language Benchmarks Game. Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/csharpcore-gpp.html Cited: 09.12.2019

[54] Laura. C# vs. C++: Which One Should You Learn? 29.10.2019. Available: https://www.bitdegree.org/tutorials/c-sharp-vs-c-plus-plus/ Cited 09.12.2019

[55] Python Notes for Professionals book. Available: https://books.goalkicker.com/PythonBook/

[56] Tratt L. 13.03.2009. Dynamically Typed Languages

[57] compiled programming language. Oxford Reference. Retrieved 4 Apr. 2020, from https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095629795

[58] Net-informs.com. How Python is Interpreted? Available: http://net-informations.com/python/iq/how.htm Cited: 12.12.2019

[59] Net-informs.com. Compiling and Linking in Python. Available: http://net-informations.com/python/iq/linking.htm Cited: 12.12.2019

[60] Gouy, Isaac. The Computer Language Benchmarks Game.Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/gpp-python3.html Cited: 04.12.2019

[61] Programming Langugage Comparison. Available: https://programming.dojo.net.nz/resources/programming-language-comparison/index Cited: 14.12.2019

[62] Klimekova E. 2015. Is Python an Appropriate Programming Language for Teaching Programming in Secondary Schools?

[63] Why C++. Available: https://cppinstitute.org/why-cpp-is-a-good-choice Cited 20.12.2019

[64] Stroustrup B. C++ Applications. 09.04.2019 Available: http://www.stroustrup.com/applications.html Cited: 20.12.2019

[65] Herity D. Modern C++ in embedded system – Part 1: Myth and Reality. Available: https://www.embedded.com/modern-c-in-embedded-systems-part-1-myth-and-reality/ Cited 22.12.2019

[66] Gouy, Isaac. The Computer Language Benchmarks Game. Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/c.html Cited: 22.12.2019

[67] Hong K. C++ Tutorial – embedded systems programming – 2018. Available: https://www.bogotobogo.com/cplusplus/embeddedSystemsProgramming.php Cited: 24.12.2019

[68] Top Uses OF C++. Available: https://www.educba.com/uses-of-c-plus-plus/ Cited 24.12.2019

[69] Beck K. Et al. Manifesto for Agile Software Development. 2001. Available: http://agilemanifesto.org/ Cited: 27.12.2019

[70] Socha-Leialoha J. 2011. Agile C++ - Agile C++ Development and Testing Visual Studio and TFS. Available: https://docs.microsoft.com/en-us/archive/msdn-magazine/2011/june/msdn-magazine-agile-c-agile-c-development-and-testing-with-visual-studio-and-tfs Cited 09.01.2020

[71] Asproni G. et al. 2004. An Experience Report on Implementing a Custom Agile Methodology on a C++/Python Project  Available: https://www.asprotunity.com/resources/articles/ExperienceReportCustomAgileMethodology.pdf Cited: 09.01.2020

[72] Computer Hope. Rust. Available: https://www.computerhope.com/jargon/r/rust.htm Cited: 11.01.2020

[73] Avram A. InfoQ.com Interview on Rust, a Systems Programming Language Developed by Mozilla. 03.08.2012 Available: https://www.infoq.com/news/2012/08/Interview-Rust/ Cited: 11.01.2020

[74] Introduction on RUST vs C++ Available: https://www.educba.com/rust-vs-c/ Cited: 12.01.2020

[75] Gouy, Isaac. The Computer Language Benchmarks Game. Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust-gpp.html Cited: 12.01.2020

[76] Drankenfield J. 18.03.2019 Cloud Computing. Available: https://www.investopedia.com/terms/c/cloud-computing.asp Cited 24.01.2020

[77] Rouse M. Software as a Service (SaaS). Available: https://searchcloudcomputing.techtarget.com/definition/Software-as-a-Service Cited 24.01.2020

[78] The agile adming. What Is DevOps? Available: https://theagileadmin.com/what-is-devops/ Cited: 24.01.2019

[79] SDLC – Waterfall Model. Available: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm Cited: 30.01.2020

[80] DevOps and Continuous Integration challenges in C/C++ projects 14.03.2017. Available: https://blog.conan.io/2017/03/14/Devops-and-Continouous-Integration-Challenges-in-C-C++-Projects.html Cited 31.01.2020

[81] Lakos J. Large-Scale C++ Software Design. Section 0.2

[82] Sayfan G. 29.01.2013. Testing Complec C++ Systems. Available: https://www.drdobbs.com/cpp/testing-complex-c-systems/240147275?pgno=1 Cited 03.02.2020

[83] Deployment in Visual C++ 11.05.2018. Available: https://docs.microsoft.com/en-us/cpp/windows/deployment-in-visual-cpp?view=vs-2019 Cited 04.02.2020

[84] Verkest, Diederik & Kunkel, Joachim & Schirrmeister, Frank. (2000). System level design using C++. 74-81.

[85] GeeksforGeeks. Insertion Sort. Available: https://www.geeksforgeeks.org/insertion-sort/ Cited: 08.03.2020

[86] Ala-Mutka, K. Problems in learning and teaching programming -a literature study for developing visualizations in the Codewitz-Minerva project. Tampere University of Technology, Finland.

[87] Duffany, J L. Choiuce of Language for an Introduction to Programming Course. Universidad del Turabo, Gurabo, PR, USA. 2014.

[88] Ali, A., & Smith, D. Teaching an introductory programming language in a general education course. Journal of Information Technology Education: Innovations in Practice, 13, 57-67. 2014. Retrieved from http://www.jite.org/documents/Vol13/JITEv13IIPp057-067Ali0496.pdf

[89] Foster, E. A COMPARITIVE ANALYSIS OF THE C++, JAVA, AND PYTHON LANGUAGES. 2014

[90] Hobbs Chris. 16.3.7 Enforcement of Strong Typing. In: Embedded Software Development for Safety-Critical Systems [Internet]. Taylor & Francis; 2016. p. 6–7.

[91] Gries D. Safety and strong versus weak typing 2018

[92] Swan J, Krawiec K, Ghani N, Squillero G. Polytypic Genetic Programming. In Springer; 2017. p. ications of Evolutionary Computation. LNCS . Springer , Amsterdam , pp. 66–81. Available from: http://eprints.whiterose.ac.uk/117964/1/polytypic_genetic_programming.pdf

[93] Devlieghere J. C++ Guaranteed Copy Elision. 21.11.2016 Available: https://jonasdevlieghere.com/guaranteed-copy-elision/ Cited 19.03.2020

[94] Donaldson T. Python. 2nd ed. Berkeley, Calif: Peachpit Press; 2009.

[95] Maxwell KD. Software Development Productivity. In: Advances In Computers. Elsevier Science & Technology. 2003

[96] Asija S. Software Engineering | COCOMO model. Available: https://www.geeksforgeeks.org/software-engineering-cocomo-model/ Cited: 07.04.2020

[97] Singh S. Difference between COCOMO 1 and COCOMO 2. Available: https://www.geeksforgeeks.org/difference-between-cocomo-1-and-cocomo-2/ Cited: 07.04.2020

[98] McConnell S. Code complete. 2nd ed. Redmond, WA: Microsoft Press; 2004.