**BASIC - Beginner's All-purpose Symbolic Instruction Code**

Something that wasn’t as complex as FORTRAN or ALGOL, but something that could be

easy to do simple arithmetics with. A computer system that was easy to use for everybody

and a computing language similarly easy and accessible to everyone on the campus.

With those ideas John Kemeny’s and Thomas Kurtz’s goal became to develop a time

sharing program. Instead of trying to modify existing languages they came to the decision

that a new language was needed. Out came something that could be made simply and to be

simple - BASIC. Alongside BASIC the Dartmouth Time Sharing System was also developed.

This in conjunction with BASIC made it so that instead of person being able to use the one

huge computer, that was available at that time many people could instead use it

simultaneously. BASIC brought computing to the masses and caused it to evolve into

various dialects all over the world.

BASIC is small and compact, but because of the intended hardware and how it was made to

be used on make it slow. BASIC combines compiling and interpreting, which means unless

you compile it it will be compiled line by line at run time. Compiling can speed up the process

but it will still be slow. Rating D

Because the aim, when creating BASIC was to make it easily accessible and useable

making it a fairly simply readable language. Rating A

BASIC’s instructions are compared to today’s orthogonal, where an instruction modifies only

very registers. But because the amount of registers is not also very big orthogonality is

lowered. Rating C

The syntax because of the language is easily understandable but fairly limited- because of

memory restrictions at time, making it very concise. Rating A

Early computers very all different and all needed most of the time needed different ways to

use them. BASIC changed this by becoming embedded in different computers and allowing

people to program in BASIC on different systems. But because it is run line by line the

reliability decreases - errors are more prone to happen. Rating C

Abstraction in BASIC is limited to few data types as well as procedural concepts - this again

was made so because of the limitations of early computers. Allowing only 286 different

variable names to be used, consisting of either a single letter, or a single letter followed by a

single digit. Data types were all limited to float types. In addition arrays were also allowed.

Rating E.

Considering that all code had to be mostly physically transported to be shared the portability

is very low. However because BASIC was included in many computer it made it easier to be

shared. Rating D.

https://www.dartmouth.edu/basicfifty/basicmanual\_1964.pdf - First BASIC manual

https://www.dartmouth.edu/basicfifty/commands.html - BASIC Commands

https://www.youtube.com/watch?v=WYPNjSoDrqw - Birth of BASIC

https://www.youtube.com/watch?v=seM9SqTsRG4 The basics of BASIC, the programming

language of the 1980s.

C

Efficiency

Being a compiled language, it will take some time to first compile the code, however once

compiled the code itself runs efficiently, especially if is well optimized. Another issue with

compiled code is that, that testing code revisions requires additional time for re-compilation of

the modified files.

Well written and optimized C code is hard to beat in performance. To improve on C code even

further, it is possible to use inline assembly [21].

A downside of efficiency in C is that in the original standard there was no multithreading

support. In the beginning there were third party implementations of this available. As of C11

standard, the language has added native multithreading support to address this shortcoming

[12].

Simplicity

There are very few different constructs in C language, which makes learning them easy and

fast. It can be considered to consist only of the essentials. The original design by K&R intended

the language to be simple and beautiful to read and write, however the language itself does not

prohibit for a developer to write extremely obfuscated code. Due to C considering whitespace as

optional, only used for improving the readability of the code, the developers have gone as far as

to holding competitions for obfuscating C code [27].

Orthogonality

C cannot be considered an orthogonal language, as there are language quirks that that differ

between data types. E.g. when comparing data types, you can use ‘==’ to compare numbers,

but not strings [12]. Similar argument can be brought when comparing dynamic and static

arrays, as dynamic arrays need to be freed manually, but we need not worry about statically

declared arrays.

Definiteness

Even though C can be considered a basic language, it contains a few constructs that are

context-dependent. One of those would be the while and do while loops, which both contain the

keyword while, but the placement and additional keyword “do” will achieve a different behavior

of the conditional checking [12]. A second example of context-dependent semantics would be

the asterisk ‘\*’ symbol, which can mean refer to three different operations: multiplication,

dereferencing a pointer or a data type. A similar idea is followed by the inverse operation of

dereferencing, using the ampersand ‘&’ symbol. When used before the variable name, it asks

for the address of a variable, however it can also mean a bitwise logical operation AND. Finally

C is standardized by ISO (latest revision is ISO/IEC 9899:2018).

Reliability

The language does not provide almost any safety at all. Any kind of errors will allow for attacks

against the software and program crashes, errors, corruption etc. It is hard to write safe to use

code in C. Often times, wrapper functions are written to help the developers write safer code.

Many other languages were also developed for the same reason.

Program verification

There is no native mechanic in C to write unit tests or do formal verification, but there are plenty

of ways to implement formal verification using third party tools. C compiler also provides data

type correctness correctness, however it can be considered partial since the developer can use

type casting to suppress the compiler warnings.

Abstraction facilities

The C can be considered a very primitive language in terms of abstraction, as most what can be

done is either handled by void pointers to data, function pointers, unions and structures. It is

also possible to have function pointers as part of structures [12].

Portability

The portability in the C language is two-fold. Even though the C language is very close to

machine language, compilers exist for almost any architecture imaginable. However the code

needs to be compiled separately for different architectures and operating systems. This allows

for decent portability, but only as long as standard libraries are in use. Once system libraries

need to be used, it needs to be rewritten for each system. This is very prevalent when graphical,

sound or audio libraries are in use. Another portability issue with the C language are struct

bit-fields, which need to be rewritten depending on whether the system is using little or big

endian.

One of the key portability benefits in C language is that it’s a subset of C++ language. This

means that any C++ compiler will be able to compile C code. So even if they didn’t design the

compiler for C code, it comes as a byproduct of the C++ specification. This of course may

change with newer versions of C and C++ being developed, as older unsafe features are being

continuously deprecated from both.

C++

Efficiency

Being a compiled language, the one-time compilation process time must be accounted for.

However more importantly, when comparing code execution, C++ is typically performing nearly

as well as C. The overhead is minimal and for most cases, negligible [15].

Simplicity

Even though C++ is based on C, the developers of this language felt that C was lacking in

features. By adding loads of new features, including the object oriented approach, the language

grew to a point where it is really hard to comprehend for an unexperienced developer.

Orthogonality

C++ is non-orthogonal language as it has non-orthogonality types (e.g. classes, strings, arrays

enumerators), expressions (e.g. prefix and postfix decrement, postfix and postfix increment) and

flow controls (e.g. conditional branches, virtual functions). [16]

Definiteness

C++ is similar to C, but there are some differences such as “ const int \* const ptr; ”. Regarding

data types the definiteness isn’t really the best, only *signed char* and *unsigned char* are

guaranteed to hold specific value range, for other primitive data types the value range is

dependent on the architecture of target system (i.e integers, *int* , can have different value ranges

in different targets).[28] Regarding documentation everything about C++ is very well defined, the

knowledge about every semantic aspect is well know and the standard library is documented

with good detail. C++ is also standardized by ISO, latest stable revision of the standard being

ISO/IEC 14882:2017 .

Reliability

All of the problems that C language has carry over to C++, as C is a subset of C++. However

C++ does introduce concepts like lists, which provide ways for a programmer to handle data

with less issues. Handling input is also simpler in C++, as cin and cout will take care of picking

the right formats.

Program verification

Since C++ is more strongly typed than C, C++ makes it easier to catch type casting errors.

Following good design practices, with regards to modern C++ (C++11 and onwards), can

greatly reduce common type issues (usage of *auto* keyword for variable type definition), but in

general the verification of the program is difficult with

Abstraction facilities

C++ has much more abstraction facilities compared to C. C++ incorporates a notion of

templates, thus making it possible to write code in more abstract level so that the type of

specific implementation is (close to) irrelevant. Template metaprogramming takes the

abstraction capabilities of C++ above other languages compared here.

Portability

In general, the portability can be considered similar, if not equal to C. The most common

compiler, GCC, typically supports both out of the box. GCC in itself is available on almost any

platform imaginable. Writing portable code needs some considerations as the primitive default

datatypes can vary between platforms, even the standard specifies them by saying that they

occupy “at least” a definite number of bytes. [29] Portable C++ code thus has to consider all the

possible architectures or use the *std::numeric\_limits* package definitions of the minimum size of

data types and apply sufficient corrections during compilation.

C++ (II wrong)

C++ can be viewed as updated version of C, adding new features to an already very solid foundation. This includes the low-level memory management, what gives the C and C++ edge in performance and efficiency, compared to some other languages, such as Java or Python, that rely on “garbage collector” subsystem. This feature does exist in C++, but external tools are needed, which are not very efficient, can cause losses in performance, so most developers will stick to manual memory management. This is the very reason why most game engines are built on C++, to allow the developer squeeze out every drop of performance possible. Another way to compare which language is the most effective, is to compare them based on time, memory and energy. Article published by the Universidade do Minho in Portugal, compared most popular programming languages how much energy, time and memory was needed to complete certain tasks. The results were very fascinating, with C being the clear winner, and the C++ right behind him. The main reason for this, was combination of manual memory management and the requirement of strong typed language. This efficiency does not come without a cost. Some developers have noted, learning and mastering C++ can be very difficult and bothersome. Some languages offer the ability to do things with few simple lines of code, but thanks to strong structure here we cannot do things as fast as we want. Every little detail must be described. What type of variable are we dealing with, how long a certain array must be or even how many bytes of data we want to allocate to a certain function. Portability of C++ is strong, meaning that it can run on most system, but it is nowhere close to Java capabilities. Setting up the development environment, for some Windows systems it can be tedious and sometimes nerve-racking. In Linux most of the heavy lifting is already been done natively. Another major is issue is the standard. Are we dealing with C++98, C++17 or C++03. When running the same code, but using a different standard, many things can happen. For example, lambda expressions or automatic type deduction and decltype. In older version of C++, such as C++03, during the declaration we had to specify the type of an object. In C++11 you don’t have to do it anymore using “auto” method, which will identify what type of variable are we dealing with. When looking C++ reliability, program verification, orthogonality, data and procedural and syntax and semantics, we can see a how they are all connected to each other. Because of the strict structure and the need of defining every variable with a certain type can be intimidating for those who are just starting to program, but in the long run, it gives a very detailed view how everything is done and can save a lot of time. In python we don’t need describe whether our variable is an integer or float. Because of the type safety, the complier will not allow to compile programs, that have not properly typed, which can cause core dumps, ending in a crashed program. Strongly typed language allows us to minimize the orthogonality, meaning that one operation won’t affect another. For example. An array can be returned if it is inside a structure. In conclusion, C++ is a more modern version of C, bringing new features, such as object orientated programming, but at the same time retaining the good old solid fountation that the C was built upon. Manual memory management, allowing us to write code for any purpose as needed, very portable. Those features do come at a cost, such a the time needed to master C++, setting up development environment, have a deep understanding of syntax and how thing work on low level, very close to hardware. Despite the negatives, it is still quite popular, maybe not as popular as Java, but still very competive, especially in embedded system, servers and game development.

**JAVA**

**Efficiency**

In software development, the programming language Java was historically considered slower than

the fastest 3rd generation typed languages such as C and C++.The main reason being a different

language design, where after compiling, Java programs run on a Java virtual machine (JVM) rather

than directly on the computer's processor as native code, as do C and C++ programs [1].

**Simplicity**

Part of the JAVAs simplicity comes from the JVM. In the sense that the user doesn't have to too any

sort of memory allocation or memory clean up [2]. The JVM has a garbage-collection algorithm

that cleans the memory. Also thanks to the JVM the programer doesn't have to worry about running

the machine that the code is executed .

**Orthogonality**

In computer programming, orthogonality means that operations change just one thing without

affecting others. The term is most-frequently used regarding assembly instruction sets, as

orthogonal instruction set. In Java it is possible to follow orthogonality but in some cases the

orthogonality has to be broken [3].

**Syntax and semantics**

Java syntax derived mostly from C and C++. Java doesn't allow global variables and functions. All

code in java has to be in classes and all the values are objects. Java is consider as a strongly typed

language, all the variables have to have a declared type [4]. In Java you are not able to overload

operators or use unsigned integers.

**Reliability and Portability**

Java is considered one of the most safe and reliable language. In its core it's has been able eliminate

a lot of security bugs by restricting certain actions [5]. JAVA is executed in JVM which makes it

compatible and runnable on every system that can fit . Biggest key aspects of JAVA is it is able to

run different version of JAVA at the same time by capsulating the code.

**Refrences:**

1. http://www.scribblethink.org/Computer/javaCbenchmark.html

2. https://www.artima.com/insidejvm/ed2/index.html

3. https://www.javaworld.com/article/2078767/open-source-tools/java-tip-orthogonality-by-example.html

4. https://docs.oracle.com/javase/tutorial/java/javaOO/variables.html

5. https://ieeexplore.ieee.org/document/722326

Python

**Efficiency (A)**

Python is really efficient tool (programming language). You can use it, when creating a

fast prototype in order to test You business logic in very early development phase as

well as production ready core applications. There are some major widely used platforms

that use Python. For example, Dropbox, Reddit and even some parts of Instagram and

Google. (1, 3, 4)

**Simplicity (A)**

Compared to the C, for example, Python is relatively simple. Therefore, it can be used

by people, who are not very familiar with programming. Finally, mathematicians can test

their algorithms without much struggle. Some universities (including TalcTech) are

taking advantage of this aspect and introduce first steps in programming in Python in

order to make programming more clearer. (1)

**Orthogonality (C)**

It is really easy to mess things up in Python - there are a lot of roughly said “global”

context. For example, many libraries take advantage of the built on “logging”

functionality. That said, it is a real struggle to synchronize the logic level. Also as a

dynamically typed language, mixing the data type operations is hardly inevitable. (1, 4)

**Syntax and semantics (A)**

Python has really good syntax. It is so plain and simple and therefore is easily teachable

even to the children in primary schools. The main reason behind it that it was designed

so in the very beginning of creating a language. (4)

**Reliability (A)**

Python runs in its own environment, which means that all the resources are given out as

they should. Developers do not have to struggle with low level program like buffer

overflow derived from memory allocation problems. (1, 3)

**Program verification (D)**

It is really hard to say if Your code does not contain any errors. One can have faulty

code running for several months without realising it. This happens, when faulty part

does not get interpreted (is hidden in some rarely used if statement). There are some

tools to overcome this, but sometimes these tools are just not good enough. (1)

**Data and procedural (B)**

One can define almost anything in Python. It has very useful data structures such as

dictionaries and stacks. But it comes with a price. As the programmers do not have to

specify data types, it is really hard to say, which kind of arguments functions/methods

may have. (1)

**Portability (B)**

As Python is interpreted in its own so called “special environment”, it is really easy to

transport Your code from one machine to another. The only strong requirement is that

this other machine (computer) has to have Python environment installed. Grade “B” is

given because of the difference between Python 2.x and 3.x, which is not backwards

compatible. (1)

**Total score: B**

*References:*

1) https://www.python.org/

2) https://dzone.com/articles/four-reasons-why-python-is-a-good-programming-lang

3) https://hackernoon.com/how-is-python-different-from-other-programming-languag

es-63311390f8dd

4) https://www.python-course.eu/python3\_history\_and\_philosophy.php

**Smalltalk**

Full procedural object oriented language based on Simula 67. First created in 1972 by

Alan Kay at Xerox’s Palo Alto Research Center (PARC). Smalltalk was one of the most

popular languages in the low time-to-market communities and with those who value low

software maintenance costs. The language has no *de facto* language standard to this

day as the ANSI Smalltalk standard offers no real differences to the first commercial

Smalltalk-80 release and many dialects of the language exist(Squeak, Pharo, and

Dolphin Smalltalk etc.). Smalltalk is considered by many to be the starter of the Object

Oriented Programming revolution.[1],[2],[3], [4]

Smalltalk uses a virtual machine interpreter which runs the code on the computer. Code

is compiled into bytecode, which is in turn interpreted by the virtual machine or

dynamically translated to machine-native code. No hardware specific optimisations and

the virtual machine is written mainly in Smalltalk itself, offering no efficiency gains but

raising its portability considerably. [3]

Efficiency GRADE: C

The language itself is very compact and the syntax can be fitted onto a postcard . It is

designed to be easily understood and written. Thus it is suitable for fast prototyping and

and has low time-to-market functionality. [1],[2]

Simplicity GRADE: A

As the language is a high level, object oriented language, its orthogonality is low.

Orthogonality GRADE: C

The syntax and semantics of the language is similar to other high level languages. It

also allows more flexibility in control flow than for example C programming language

with user defined control flow statements. [3]

Definiteness GRADE: B

Smalltalk is an interpreted language, thus it has high reliability compared to other truly

compiled languages like C++. [3]

Reliability GRADE: B

The language supports a wide variety of abstraction facilities which can be found in

modern programming languages: classes, widely known flow control statements, code

blocks, return statements. Classes support inheritance, variables and methods (static

member functions in Java/C++). No constructors, operators or keywords. Operators are

Analysis of Programming Languages (IAG0450) Karl Laanemets 178191IASM

defined in the receiving objects class. Supports user defined flow-of-control constructs.

[2],[3] Abstraction facilities GRADE: B.

Smalltalk code runs on any system identically regardless on which system the code was

written. It has better hardware portability than Java. [3] Portability GRADE: A

Refrences:

[1] http://sdmeta.gforge.inria.fr/Programmez/OnTheWeb/Art2-Eng-AminimalistSyntax.pdf

[2] http://logos.cs.uic.edu/476/resources/SmallTalk/cs476\_Smalltalk/Smalltalk.htm

[3] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

[4] https://stackoverflow.com/questions/6368337/whats-the-difference-of-ansi-smalltalk-a

nd-smalltalk-80

**Rust**

Efficiency

Rust, being a compiled language, means that translating the written code to machine code

takes time, though correctly compiler machine code will most probably run faster than code that

is interpreted and ran on some virtual machine.

Simplicity

Rust is not a simple language by any means. Even the basic concepts are different compared to

C/C++ derived languages so learning curve is very steep, for example simple variable

ownership that is designed to be thread safe. The number of language features on the other

hand isn’t as big as in C++ meaning that obtaining the knowledge on the language features is

somewhat more manageable.[3]

Orthogonality

Rust is quite orthogonal as many keywords can be combined in different ways.[3] But it can not

be compared with the orthogonality offered by Go language. [13, 14] Orthogonality in Rust can

be demonstrated by the feature that every piece of language can nest inside another, for

example we can have a whole module (equivalent to namespace in C++) inside some *if*

statement.[20]

Definiteness

Rust is definite in a sense that it is designed to be a systems programming language (like C).In

the sense of consistency Rust is not the greatest language out there. For example there are

multiple different operationions that are performed by the same operator : *..expr* can either

mean right-exclusive range literal or struct literal updating; *Self* and *self* keywords differ in

meaning, where the first refers to type aliasing mechanism and the latter method subject,

current instance (similar to *this* keyword in C++). All this results in a code where the definition of

how some operation is done isn’t obvious immediately but the context and exact listing must be

taken into account. In addition there doesn’t seem to be available strict semantics of the

language (or the memory model used) which is interesting as the language is called and thought

to be safe.

Reliability

The type system and memory management of Rust is far superior compared to C and C++.

Essentially the language itself prevents you from segmentation faults (access violations). Also

the same implementation is designed to handle race condition issues in multithreading. In

conclusion Rust can be seen as a language when after writing the code one can be fairly certain

that the program will not have data and memory management issues that C and C++ have.

Although it must be mentioned that the *unsafe* keyword, present in the Rust language, allows to

bypass the checks in a similar way that type casting is done by C and C++ but the specific

keyword allows easier identification of problematic and error-prone code areas.[17]

Program verification

As mentioned before the type and memory handling systems in Rust are very rigorous. Meaning

that verification of the software can be limited to checking the functional aspects of the code.

This is much easier compared to C and C++ as Rust has built-in capabilities of automated

testing (unit-testing).[18] .

Abstraction facilities

Rust supports abstraction facilities like traits, iterators etc. Rust creators promote that the

language has “zero-cost” abstractions. Diving deeper and comparing traits with abstract classes

of C++ one can see that the support is quite limited. Regarding data type related abstraction

Rust supports generic data types that is comparable to the template options and *auto* keyword

provided by C++.[19]

Portability

Rust can be compiled and run on variety of platforms. Rust has grouped supported platforms

into tiers: Tier 1 (guaranteed to work), Tier 2 (guaranteed to build) and Tier 3 (code base has

support, but not build and tested automatically). [11] As Rust can be still considered as a young

language, that is gaining popularity, the variety of platforms where it can be used will definitely

rise.

**Go**

Efficiency

The go language is a compiled language, so the process here is the same as with C. However

as the language provides memory handling and safety for the developer, it boasts about twice

the memory usage and slower overall speed, algorithm dependent [6]. The language is

designed natively to be efficient in multithreading and networking with the purpose of developing

server side software [7].

Simplicity

Compared to C, Go boasts better support for IDEs by removing the symbol table, which made C

hard to handle. It’s simpler to develop various tools and debuggers [7]. They’ve also simplified

variable declarations by moving the type after the variable name(s) and allowing to declare

variables without implicitly specifying the variable type. Go introduces a garbage collector to

lessen the load on bookkeeping by the developer. They have also removed code elements

which were designed for lexers, but unfriendly for human operators.

Orthogonality

Orthogonality plays important role in designing Go language and it’s libraries. It is inspired by

Unix Philosophy, keeping in mind the principles that code units should kept simple,

concentrating on doing one task well and having standard way to communicate. [9]

Definiteness

The Go language design philosophy has been reducing complexity from the beginning,

removing many of the repetitive ways of doing the same things in C and other language [7]. A

good example of this would be to remove all of the loop types except for loop, which now

handles all possible cases [22]. Only thing that remains context dependent is the asterisk

symbol, which is still used for both pointer operations and multiplication.

Reliability

One of the main purposes of Go is to improve on the unsafe practices in C. By introducing

memory and type safety there are a lot less caveats where the programmer might

unintentionally introduce bugs [7].

Program verification

Go makes program verification very easy. It can be built and tested quickly, is statically typed, is

simple, concise, but expressive, has clear paths of error and recovery, has well maintained and

extensive standard library [10]. Go language also has built in testing facilities, which allows for

very simple unit testing. It also provides means for runtime analysis, creating options for simple

performance profiling [23].

8

Abstraction facilities

Go is designed to be a simple language for new programmers and therefore does not have

loads of abstractions[7]. The abstraction capability for Go is similar in nature to C language,

while some of the constructs have been reworked. Function pointers in C have been updated for

better readability [25], but are still usable. Structures are also slightly more powerful, however

Go does not consider itself as an object-oriented language, thus they are not as powerful as

classes in other object oriented languages. They are lacking in hierarchy, however now it is

possible to have methods inside of classes. In C it was possible to have function pointers inside

of classes, Go’s implementation of methods is more powerful. The language developers

themselves consider Go not to have classes, but structures instead [26].

Portability

Although Go compiles code into executable for a specific platform, it is made very easy and

hassle-free to choose the target platform(s) [8]. This is limited in nature however to more

powerful systems, as the language was designed by google to help replace C++ in creating

server software for their ever-growing operations [7]. This means that, even though it can be

considered an all-purpose language, compiling for embedded systems and various other

lower-performing hardware can be problematic due to the added safety features and garbage

collector, which are taking up resources. This isn’t however impossible, as there are plenty of

projects out there, such as Go, Robot, Go, which are designed to help use the language on

embedded systems [24]. That all said, being a newer language, Go cannot compete with the

evergreen C and C++ languages and their compatibility with older systems.

List of references

[1] Molotnikov, Z., Schorp, K., Arvantinos, V., Schätz, B., Future Programming Paradigms in

the Automotive Industry. [Online].

Available: https://www.vda.de/dam/vda/publications/2016/FAT/FAT-Schriftenreihe\_287.pdf

.Accessed on: Nov. 1, 2018

[2] Trejo, D., After All These Years, the World is Still Powered by C Programming. [Online].

Available:

https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming

.Accessed on: Nov. 1, 2018

[3] Mollevik, I., Olsson, S. Vikdahl, M., Weijand, S., Westin, J. Seminar: The Rust

Programming Language. [Online].

Available: https://www8.cs.umu.se/kurser/5DV086/VT18/resources/seminar/rust.pdf .Accessed

on: Nov. 1, 2018

[4] A vision for portability in Rust. [Online].

Available: http://aturon.github.io/2018/02/06/portability-vision/ .Accessed on: Nov. 1, 2018

[5] Baranowski, M., He, S., Rakamar, Z. Verifying Rust Programs with SMACK.

Proceedings of the 16th International Symposium on Automated Technology for Verification and

Analysis (ATVA), 2018

[6] Go versus C gcc fastest programs. [Online]

Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/faster/go-gcc.html

.Accessed on: Nov. 1, 2018

[7] The Go Programming Language. Frequently Asked Questions (FAQ). [Online]

Available: https://golang.org/doc/faq .Accessed on: Nov. 1, 2018

[8] The Go Programming Language. Package build. [Online]

Available: https://golang.org/pkg/go/build .Accessed on: Nov. 1, 2018

[9] Schlesinger, A. Orthogonality in Go. [Online]

Available: https://arschles.svbtle.com/orthogonality-in-go .Accessed on: Nov. 1, 2018

[10] Kol, T. How to write bulletproof code in Go: a workflow for servers that can’t fail. [Online]

Available:

https://medium.freecodecamp.org/how-to-write-bulletproof-code-in-go-a-workflow-for-servers-th

at-cant-fail-10a14a765f22 .

Accessed on: Nov. 1, 2018

[11] Rust Forge. Rust Platform Support. [Online]

Available: https://forge.rust-lang.org/platform-support.htm .Accessed on: Nov. 1, 2018

[12] ISO/IEC 9899:2011 Information technology - Programming languages - C, 2011

[13] Go and Rust. [Online]

Available: https://www.davidb.org/post/go-and-rust/ .Accessed on: Nov. 1, 2018

[14] Mollevik, I., Olsson, S. Vikdahl, M., Weijand, S., Westin, J. Seminar: The Rust

Programming Language. [Online].

Available: https://www8.cs.umu.se/kurser/5DV086/VT18/resources/seminar/rust.pdf .Accessed

on: Nov. 1, 2018

[15] C++ g++ versus C gcc fastest programs. [Online]

Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/faster/cpp.html

.Accessed on: Nov. 1, 2018

[16] Yang, Feng-Jen, The orthogonality in C++, J. Comput. Sci. Coll., Consortium for

Computing Sciences in Colleges, 2008

[17] The Rust Programming Language. Unsafe Rust. [Online]

Available: https://doc.rust-lang.org/book/second-edition/ch19-01-unsafe-rust.html#unsafe-rust

.Accessed on: Nov. 1, 2018

[18] The Rust Programming Language. Writing Automated Tests. [Online]

Available: https://doc.rust-lang.org/book/second-edition/ch11-00-testing.html .Accessed on:

Nov. 1, 2018

[19] The Rust Programming Language. Generic Data Types. [Online]

Available: https://doc.rust-lang.org/book/second-edition/ch10-01-syntax.html . Accessed on:

Nov. 1, 2018

[20] Blandy, J., Orendorff, J., Programming Rust: Fast, Safe Systems Development, O’Reilly

Media, 2017

[21] Coppen, W., S., Optimizing C/C++ with Inline Assembly Programming. [Online]

Available: http://www.drdobbs.com/optimizing-cc-with-inline-assembly-progr/184401967

.Accessed on: Nov. 1, 2018

[22] The Go Programming Language. The Go Programming Language Specification. [Online]

Available: https://golang.org/ref/spec .Accessed on: Nov. 1, 2018

[23] The Go Programming Language. Package testing. [Online]

Available: https://golang.org/pkg/testing/ .Accessed on: Nov. 1, 2018

[24] Gobot - Golang framework for robotics, drones and the Internet of Things (IoT). [Online]

Available: https://gobot.io/ .Accessed on: Nov. 1, 2018

[25] The Go Blog. Go's Declaration Syntax. [Online]

Available: https://blog.golang.org/gos-declaration-syntax Accessed on: Nov. 1, 2018

[26] Golang tutorial series. Part 26: Structs Instead of Classes - OOP in Go. [Online]

Available: https://golangbot.com/structs-instead-of-classes/ .Accessed on: Nov. 1, 2018

[27] The International Obfuscated C Code Contest. [Online]

Available: https://www.ioccc.org/ . Accessed on: Nov. 1, 2018

[28] Stroustrup, Bjarne, The C++ Programming Language, Fourth Edition, Addison-Wesley

Professional, 2013

[29] Cppreference. Fundamental types. [Online]

Available: https://en.cppreference.com/w/cpp/language/types . Accessed on: Nov. 1, 2018