TALLINN UNIVERSITY OF TECHNOLOGY
Department of Computer Systems
Study Centre for Computer Systems

Comparison of programming languages

Authors: **René Pihlak** **178247IASM**
Tanel Peet **178248IASM**
Risto Heinsar **178446IASM**
Tarmo Prillop **178206IASM**

Tallinn 2018

Introduction

For this project, we have selected the following languages: C, C++, Rust and Go. The basis on the selection was to have languages that belong to the same family, being similar in nature to one and the other, however have various improvements and alterations made to them. The work intends to compare C language to the languages that have been developed with the intention to improve on it. In this case, C++ can be considered the object oriented version of C. Rust and Go in term are improved versions of C with safety and modern features built into them such as memory-safe operations, multithreading support etc.

Each of the languages compared will be graded in eight categories from ‘A’ to ‘E’, ‘A’ being the highest grade we can give and ‘E’ the lowest. The grades are based on our research to these languages and the findings presented in the work. The comparisons are made only between the languages targeted by this work, relative to each other. For the comparisons to have any kind of meaning, it does not consider any of the other languages available for developers, as this would possibly marginalize the differences between our research subjects and not objectively highlight differences. The full table can be seen in Appendix: Grades for select programming languages.

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 languages, one of them being the gets() function which is replaced with gets_s() [12].

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.

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].

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-that-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

Appendix: Grades for select programming languages

	C	C++	Rust	Go
Efficiency	A	B	B	E
Simplicity	A	C	E	B
Orthogonality	D	D	B	C
Definiteness	B	B	D	A
Reliability	E	D	C	A
Program verification	E	D	B	A
Abstraction facilities	E	A	C	C
Portability	A	A	B	B