Programming Languages and Paradigms: C++

Qasim Warraich
Department of Informatics
University of Zurich
Zurich, Switzerland
qasim.warraich@uzh.ch

***Abstract*—This is a semester paper for the Programming Languages and Paradigms seminar at the University of Zurich under the direction of Professor Carol Alexandru. This paper aims to provide a concise overview about the history of the C++ programming language, some of the features that make it unique and offer a general overview and reflection on the experience programming in C++ during the semester.**

***Index Terms*—cpp, c++, programming languages**

I. Introduction

The C++ programming language is an extremely successful and widely used multi paradigm general purpose programming language. It has many reputations, some admire it for its versatility, some adore its portability and low level features and others criticise it for its lack of direction and complexity but it has managed to stand the test of time for almost four decades and shows no signs of slowing down. It is still one of the most widely used programming languages and finds itself at the core of many popular pieces of software e.g. Mozilla Firefox, MySQL, certain Adobe software, Microsoft Office, KDE, etc. A non-exhaustive list of C++ applications can be found on the creator of the language Bjarne Stroustrup's website here [1] .

II. History

The history of the C++ programming language, or rather the pre-history as it was then known as *C with Classes*, begins in 1979 when Bjarne Stroustrup was an employee at AT&T Bell Labs. As the name suggests the motivation behind the creation of the language was to try to synthesize the complementary characteristics of the C programming language, also a Bell Labs creation, and Obejct Orientated languages. Bjarne found Simula67's Object Orientation style to be helpful for large scale software development during his experience with the language in his PHD Thesis, but was unimpressed by its performance characteristics. The goal of C with Classes was build upon C, which is renowned for its high performance and to expand it with Simula like Object Orientation [2].

The initial implementation of C with Classes added features on top of an existing C compiler basically creating a C with Classes preprocesseor called Cpre. Cpre was responsible for processing the code in to C which was then compiled into a binary. This version of the implementation included features such as; classes, derived classes, public/private access, constructors and destructors, call and return functions, friend classes, strong type checking and conversion, default and inline functions and overloading of the assignment operator.

In 1982 Stroustrup started work on the successor to C with classes which was where the name C++ was born. The name is a play on the increment operator "++" in C, the actual creation of the name is credited to Rick Miscitti [3]. The name reflected Bjarne's desire to not only create a successor to C with classes but also to C. This ambition can still be found on the page on C++ on Bjarne Stroustrup personal website where he claims C++ "is a better C" [4]. Along with the new name came some new features. Some of the new features of C++ at this early stage included; virtual functions, function name and operator overloading, references, constants, type-safe free-store memory allocation (new/delete), improved type checking, and BCPL style single-line comments with two forward slashes (//) [3].

By 1984 the Cfront compiler for C++ was implemented by Bjarne, originally written in C with classes but then written in C84, the very first "standard" for C++. Also in 1984, Bjarne implemented the stream input/output **(IO)** library which employed the idea of IO being an operator rather than a function. This idea has been accredited to Doug Mcllroy of Unix pipes fame [3].

1985 brought with it the release of the first edition of *The C++ Programming Language* [5] which became the definitive reference for the language as this predates any actual standards for C++. The first commercial version of C++ became available in October of 1985 [6].

In 1989 C++ 2.0 was released alongside the updated version of the official reference book *The C++ Programming Language* coming in 1991. Some of the new features in C++ 2.0 included multiple inheritance, abstract classes, static member functions, const member functions, and protected members. In 1990, an annotated reference manual was published and became the basis for the future standardisation of C++. Later features introduced to C++ 2.0 included some of the most prolific of the language such as; templates, exceptions, namespaces, new casts and a Boolean type. 1993, marked the beginning of Alex Stepanov's work on the Standard Template Library **(STL)**. The STL is now essentially as much a part of what we now call the C++ standard as the language itself.

1998, marked the first standardised release of C++, C++98. This brought with it the standard naming convention we are familiar with today ("C++" + 2 digit year"). The standard of C++ would now be defined by an ISO/IEC committee which

would work together on shaping the languages future features and implementation. After this point subsequent standards brought on relatively minor changes up until the release of C++11 in 2011. C++11 was a major update bringing with it heaps of new features to the language and greatly expanding the STL. Some of the most significant new additions included; Variadic templates, Hash tables, Range based for loops, auto as a type inference operator rather than for storage duration, a nullptr type, lambdas and smart pointers. C++11 began the era we can term as "modern C++". A basic example of some of these features of "modern C++" can be found at Code 3. Subsequent standards included mostly minor additions bringing us to where we are today as of the righting of this paper, C++20.

A. Relavent Literature

Relevant literature can be found at [2]–[5], [7]–[9].

III. The C++ Programming Language

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello World\n";
    return 0;
}
```

Code 1: Classic Hello World Example.

The C++ programming language comprises of two main implementation goals. It aims to provide direct access to hardware level features, primarily through its C subset and then build on those mappings by providing abstractions. During a lecture at the University of Edinburg entitled *The essence of C++*, Stroustrup describes C++ as "a light-weight abstraction programming language [designed] for building and using efficient and elegant abstractions" [10]. The language is truly multi paradigm offering first class support for both procedural and object orientated paradigms. Whilst some may argue that C++ is not purely object orientated, these sort of discussions are more of a *No true Scotsman* fallacy than constructive debates. In reality C++ covers almost every pillar of object orientated design, it just doesn't force the developer down that road unless they decide that's the path to take. In recent standards support for certain functional programming paradigms have also been introduced but are far from being considered as first class features.

A. The C in C++

The syntax of C++ is mostly inherited from C as illustrated by the Hello World code in Code 1. This trivial example highlights the clear similarities between the two languages but also some ways in which they diverge. This is clearly seen by C++'s usage of an operator (« and ») to handle IO rather than the C equivalent library functions. This extremely simple example also introduces two more features that are extremely characteristic of C++. Firstly the usage of namespaces as shown by the reference to "std" followed by the scope operator (::) and secondly the usage of of the STL where the function "cout" resides. The similarities and compatibility with C is a reoccurring theme in the design of C++ and a note worthy example of this is the fact that C++ has full support for both C styled null terminated strings in addition to its very own string class. In fact command line argument parsing is exactly as it is in C and uses C style strings. Similar to C, C++ also employs the use of a main function as the starting point of the program. Compatability with C is not 100% and C is by no means a subset of C++, but its influence on the language and the design around keeping some semblance of compatibility (even of libraries) are a clear indicator of the roots of the language. A more comprehensive review can be found here [11].

B. Memory Management

C++ is well known for its deliberate design choice to not support garbage collection natively. The language does however support four styles of memory management.

- Static Storage: Objects created before the entry of main() and destroyed in reverse order after main() exits.
- Thread Storage: Objects are created similarly as in static storage but the creation time is thread dependent. Creation is just prior to thread creation and destruction occurs just after the thread has been joined.
- Automatic Storage: This is scope based memory management and creation occurs at the point the execution passes the declaration and destruction follows as son as the local scope of declaration is closed. Member variables are created along side the parent and are destroyed when the parent is. This sort of automatic management is the source of a widely used paradigm in C++ termed *Resource Acquisition Is Initialization* or **(RAII)**.
- Dynamic Storage: In dynamic storage objects have a dynamic life span and creation/deletion is explicitly handled by the respective operators new/delete. As of C++11 and the introduction of smart pointers the usage of new has been advised against in favour of smart pointers such as `make _unique<T>` and `make _shared<T>`.

C. Templates

C++ uses templates to enable generic programming. Template are compile time parametrised functions or classes written without specific instantiation arguments. Instantiation is handled at compile time and uses a substitution approach. Parameterization can be done by types, compile time constants and other templates. An example of a generic quicksort lomuto partitioning using templates done during the seminar can be found here Code 2 and here (Github).

D. The Standard Template Library

The STL is as much a part of the C++ standard as the language itself. The STL is a collection of types, data structures (containers), algorithms and iterators that offer programmers

```cpp
template <class T>
int partition(std::vector <T> &arr, int low, int high)
{
    auto pivot = arr[low];
    int i = low;

    for (int j = i + 1; j <= high; j++) {
        if (arr[j] <= pivot){
            i++;
            std::swap(arr[i], arr[j]);
        }
    }
    std::swap(arr[i], arr[low]);
    return i;
}
```

Code 2: A generic implementation of lomuto partitioning using templates.

a wide variety of design freedom and can make implementing programs much faster in comparison to languages like C. STL library features are typically very well tested and optimised to some degree that it is often safer to pick an STL algorithm than to implement one by hand. Some examples of functionality provided by the STL include; sorting algorithms, vectors, associative arrays, lists and iterators. The STL is implemented in a way that it capitalises upon a lot of the facilities for generic programming provided in C++, most significantly through the use of templates. A breadth of information about the STL can be found here [8].

```cpp
//auto is also useful for reducing the verbosity of
//the code. For instance, instead of writing

for (std::vector<int>::const_iterator itr =
    myvec.cbegin();
    itr != myvec.cend();
    ++itr)

//the programmer can use the shorter

for (auto itr = myvec.cbegin();
    itr != myvec.cend();
    ++itr)

//which can be further compacted since "myvec"
//implements begin/end iterators:

for (auto& x : myvec)
```

Code 3: Modern C++ example: auto operator and range based for loops [12].

IV. Discussion

C++ is an extremely versatile language with a myriad of applications. It does have a bit of a reputation as a complicated and confusing language with a huge amount of features which only a small subset can be realistically considered at one time. The creator of C++ himself has been quoted as saying *"There are only two kinds of languages: the ones people complain about and the ones nobody uses"* [13]. It seems that opinions about the language are quite personal and almost polarising. Take for example the impressions of two of Bjarne's coworkers at AT&T Bell Labs and extremely accomplished and highly respected computer scientists Ken Thompson (Unix) and Brian Kerrigan (AWK).

Ken Thompson: "It certainly has its good points. But by and large I think its a bad language. It does a lot of things half well and its just a garbage heap of ideas that are mutually exclusive. Everybody I know, whether its personal or corporate, selects a subset and these subsets are different. So its not a good language to transport an algorithm to say, I wrote it; here, take it. Its way too big, way too complex. And its obviously built by a committee. Stroustrup campaigned for years and years and years, way beyond any sort of technical contributions he made to the language, to get it adopted and used. And he sort of ran all the standards committees with a whip and a chair. And he said no to no one. He put every feature in that language that ever existed. It wasnt cleanly designedit was just the union of everything that came along. And I think it suffered drastically from that." [14].

Brian Kernighan: "C++ has been enormously influential. ... Lots of people say C++ is too big and too complicated etc. etc. but in fact it is a very powerful language and pretty much everything that is in there is there for a really sound reason: it is not somebody doing random invention, it is actually people trying to solve real world problems. Now a lot of the programs that we take for granted today, that we just use, are C++ programs." [15].

There is a lot to be said about the complexity of C++, even Bjarne Stroustrup admits that "within C++, there is a much smaller and cleaner language struggling to get out" [13]. But there is also a lot to be said about the community and interest behind C++. The language has carved its niche as the language for when a combination of performance and implementation flexibility is crucial. Being based on C and using a similar compiler and tooling suite has probably also done it some favours along the way in terms of adoption and its popularity in systems and embedded programming. The language seems to show no evidence of slowing down and the community and standards committee are constantly working and iterating on the language to address the issues of complexity (see: Code 3) and to make it the language that can do all things for all people . Whether this is a worthwhile goal to set is subject to debate.

V. Conclusion

This work provides a brief overview of the C++ Programming Language. A basic outline of the history of the language has been provided as well as some anecdotes relevant to its creation and origins. Supporting features such as the Standard Template Library and the standardisation process in

general are also discussed. Furthermore, this work aims to introduce the reader to some of the paradigms supported by the language. Some of the more prolific features of the language were also introduced and illustrate through the use of simple examples. Of course this work only offers a small glimpse into the massive world behind C++. For further reading on the language please refer to the references section of this paper.

REFERENCES

[1] S. Bjarne. C++ applications. (2020, Oct 27). [Online]. Available: https://stroustrup.com/applications.html

[2] B. Stroustrup, "Evolving a language in and for the real world: C++ 1991-2006," in *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, 2007, pp. 4–1.

[3] ——, "A history of c++ 1979–1991," in *History of programming languages—II*, 1996, pp. 699–769.

[4] S. Bjarne. C++. (2021, May 10). [Online]. Available: https://stroustrup.com/C++.html

[5] B. Stroustrup, *The C++ Programming Language*. Addison-Wesley, 1985.

[6] S. Bjarne. C++ Invention: FAQ Entry. (2020, Nov 23). [Online]. Available: https://www.stroustrup.com/bs_faq.html#invention

[7] B. Stroustrup, "An overview of the c++ programming language," *Handbook of object technology*, 1999.

[8] N. M. Josuttis, *The C++ standard library: a tutorial and reference*. Addison-Wesley, 2012.

[9] A. Alexandrescu, *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley, 2001.

[10] B. Stroustrup. (2014) The essence of c++. [Online]. Available: https://www.youtube.com/watch?v=86xWVb4XIyE

[11] B. Calder, D. Grunwald, and B. Zorn, "Quantifying behavioral differences between c and c++ programs," *Journal of Programming languages*, vol. 2, no. 4, pp. 313–351, 1994.

[12] (2021, Apr) Page Version ID: 1015973137. [Online]. Available: https://en.wikipedia.org/w/index.php?title=C%2B%2B11&oldid=1015973137

[13] S. Bjarne. Bjarne Stroustrup Quotes. (2020, Dec 19). [Online]. Available: https://www.stroustrup.com/quotes.html

[14] P. Seibel, *Coders at work: reflections on the craft of programming*. Apress, 2009.

[15] (2018). [Online]. Available: https://www.youtube.com/watch?v=zmYhR8cUX90

[16] P. Seibel, *Coders at work: Reflections on the craft of programming*. Apress, 2009.

[17] B. Stroustrup, "Multiple inheritance for c++," *Computing Systems*, vol. 2, no. 4, pp. 367–395, 1989.

[18] J. Coplien, "C++ idioms." in *EuroPLoP*, 1998, pp. 11–34.

[19] S. Koranne, "Boost c++ libraries," in *Handbook of open source tools*. Springer, 2011, pp. 127–143.

[20] B. Karlsson, *Beyond the C++ standard library: an introduction to boost*. Pearson Education, 2005.

[21] D. Abrahams and A. Gurtovoy, *C++ template metaprogramming: concepts, tools, and techniques from Boost and beyond*. Pearson Education, 2004.

[22] S. Bjarne. An incomplete List of C++ Compilers. (2019, Apr 28). [Online]. Available: https://stroustrup.com/compilers.html

[23] B. Stroustrup, "An overview of the c++ programming language," *Overload 161*, 2021, https://accu.org/journals/overload/29/161/stroustrup/.