**HISTORY AND EVOLUTION OF COMPUTER PROGRAMMING LANGUAGES**

****

**EDDIE FULLER**

**DECEMBER 15TH, 2009**

**MATH 492**

**DR ELISABETH BROWN**

**Abstract**

This paper was written for the MATH 492 class, “History of Mathematics”, for the Fall 2009 semester. In this paper, we’ll take a look at the foundations and history of computers, and more importantly, computer programming and computer programming languages. This paper takes a look at programming languages, starting with what is contested to be world’s first computer program to a projection of what the next decade holds in store for computers and computer programming. Throughout the course of this paper, we take a look not only at the history of computer programming, but some brief profiles on some of the more influential people that molded computers and computer programming into what it is today, as well as brief profiles on several influential programming languages themselves. Since many of the terms brought up in this paper are technical in nature, a glossary has been provided at the end of the paper. Non-header terms in bold text can be further defined by looking them up in the provided glossary.

**WORKS CITED**

1. Sammet, Jean E. (1969). *Programming Languages: History and Fundamentals*. Englewood Cliffs, NJ: Prentice-Hall.
2. History of Programming Languages.(n.d.). In *Wikipedia*. Retrieved November 27th, 2009, from <http://en.wikipedia.org/wiki/History_of_programming_languages>

This source was a backbone of my research, which allowed me to find other sources from which to expand upon the elements in this report. Since the article was so well layed out, I used this as an outline and stepping stone for my report, fleshing in details as needed.

1. Klerer, Melvin. (1991). *Design of Very High-Level Computer Languages*. USA: McGraw-Hill.

From this book, I grabbed specifics and examples of mathematics involved in the design of high level computer programming languages, as this paper focuses on the mathematics involved in high level languages alone, although the same concepts undoubtedly apply to assembly language as well.

1. Bergin, Thomas J. & Gibson, Richard G. (1996). *History of Programming Languages*. New York, NY: ACM Press.

This was used to flesh in the details given from the original Wikipedia outline, and additional information as well.

1. Koffman, Elliot B. & Friedman, Frank L. (1993). *Fortran With Engineering Applications: 5th Edition*. Reading, MA: Addison-Wesley Publishing Company.

This book, an old Fortran textbook of mine, actually has a few historical snippets used in this report, as well as some general details about the Fortran 77 language.

1. Saari, Peggy. "Science And Invention - Who Wrote The First Computer Program?." History Fact Finder. Ed. Julie L. Carnagie. UXL-GALE, 2001.eNotes.com. 2006. 6 Dec, 2009 <http://www.enotes.com/history-fact-finder/science-invention/who-wrote-first-computer-program>
2. BBC News. “PM apology after Turing petition”. (2009). 11 Sep, 2009. Online news article. <<http://news.bbc.co.uk/2/hi/technology/8249792.stm> >

**Appendix A: Glossary**

|  |  |
| --- | --- |
| **Analytical Engine** | The Analytical Engine was a design created by Charles Babbage in 1837 that he worked on until his death in 1871. In its logical design, the machine was essentially modern, anticipating the first completed general-purpose computer by about 100 years. |
| **Assembly Languages** | Assembly languages are a family of low-level languages for programming computers, microprocessors, microcontrollers, and other integrated circuits. Assembly languages implement a symbolic representation of the binary machine codes, and are usually tailored to a specific CPU architecture. |
| **Asymptotic Notation** | Also known as Big O notation, asymptotic notation describes the limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions. |
| **Backus-Naur Form** | BNF is a metasyntax used to express context-free grammars, or a formal way to describe formal languages. It is widely used as a notation for grammar in computer programming languages, instruction sets, and communication protocols. |
| **Bernoulli numbers** | Bernoulli numbers are a sequence of rational numbers with deep connections to number theory. |
| **Bytecode** | Bytecode is a form of instructions executed by Java virtual machines. Bytecode is essentially the Java equivalent of assembly code in other languages. |
| **Encapsulation** | Encapsulation is an attribute of object-oriented programming in which all of the object’s data is contained and hidden within the object itself, and access to it is restricted to members of that class. |
| **Exception Handling** | Exception handling is a language construct or hardware mechanism designed to handle the occurrence of exceptions, special conditions that change the normal flow of program execution. |
| **Free-form Language** | A free-form language is a programming language in which the positioning of characters on the page in program text is not significant. The program text does not need to be placed in specific columns as on old punched card systems or some assembly languages. |
|  |  |
| **Inheritance** | Inheritance is a way to form new classes (or objects) using classes that have already been defined. Classes created in this fashion are known as derived classes; the class inherited from is then referred to as the base class, or ancestor class. |
| **Interleaving** | Interleaving is a way to arrange data in a non-contiguous way to increase performance. |
| **Imperative Programming** | Imperative Programming is a **programming paradigm** that describes computation in terms of statements that change a program state. |
| **Object-Oriented Programming** | Object-Oriented Programming (OOP) is a programming paradigm that uses “objects”, or data structures consisting of datafields and methods together with their interactions, to design software with information hiding, data abstraction, **encapsulation**, modularity, **polymorphism**, and **inheritance**. |
| **Open Source Software** | Open source software, or OSS, is computer software for which the source code and certain other rights normally reserved for copyright holders are provided under a software license that meets the Open Source Definition, or is in the public domain. |
| **Operator Overloading** | Operating overloading is a specific case of **polymorphism** in which some or all of the operators like +, =, or == have different implementations depending on the types of their arguments. |
| **Polymorphism** | Polymorphism, in the context of object-oriented programming, is the ability of one type, A, to appear as and be used like another type, B. |
| **Programming Paradigm** | Programming paradigms are fundamental styles of computer programming. Paradigms differ in the concepts and abstractions used to represent the elements of a program, and the steps that compose a computation. |
| **Static Typing** | A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time. Data types are associated with variables, not values. |
| **Template** | A template is a feature of some programming languages that allow functions and classes to operate with generic types, allowing a function or class to work on many different data types without being rewritten for each one. |
|  |  |
|  |  |
| **Turing Complete** | Turing Completeness refers to a computer that can computer every Turing-computable function, which is to say, any conceivable function that can be done by a machine. While a truly Turing Complete machine would require infinite storage, a machine may be said to be Turing Complete if it WOULD be truly Turing Complete given unlimited resources. |
| **Virtual Function** | A virtual function is a function or method whose behavior can be overridden within an inheriting class by a function with the same signature, or function name/data inputs. This is an important part of **polymorphism** in object-oriented programming. |
| **Virtual Machine** | A virtual machine, or VM, is a software implementation of a machine (computer) that executes programs like a physical machine. |
| **von Neumann Architecture** | von Neumann architecture is a design model in electronics for a stored-program digital computer that uses a processing unit and a single separate storage structure to hold both instructions and data. |