**INTEGRAL**

In mathematics, an integral assigns numbers to functions in a way that can describe displacement, area, volume, and other concepts that arise by combining infinitesimal data. Integration is one of the two main operations of calculus, with its inverse, differentiation, being the other. Given a function f of a real variable x and an interval [a, b] of the real line, the definite integral is defined informally as the signed area of the region in the xy-plane that is bounded by the graph of f, the x-axis and the vertical lines x = a and x = b. The area above the x-axis adds to the total and that below the x-axis subtracts from the total.

Roughly speaking, the operation of integration is the reverse of differentiation. For this reason, the term integral may also refer to the related notion of the antiderivative, a function F whose derivative is the given function f. In this case, it is called an indefinite integral.

The principles of integration were formulated independently by Isaac Newton and Gottfried Wilhelm Leibniz in the late 17th century, who thought of the integral as an infinite sum of rectangles of infinitesimal width. Bernhard Riemann gave a rigorous mathematical definition of integrals. It is based on a limiting procedure that approximates the area of a curvilinear region by breaking the region into thin vertical slabs. Beginning in the nineteenth century, more sophisticated notions of integrals began to appear, where the type of the function as well as the domain over which the integration is performed has been generalised. A line integral is defined for functions of two or three variables, and the interval of integration [a, b] is replaced by a certain curve connecting two points on the plane or in the space. In a surface integral, the curve is replaced by a piece of a surface in the three-dimensional space.

Symbolic integration

Many problems in mathematics, physics, and engineering involve integration where an explicit formula for the integral is desired. Extensive tables of integrals have been compiled and published over the years for this purpose. With the spread of computers, many professionals, educators, and students have turned to computer algebra systems that are specifically designed to perform difficult or tedious tasks, including integration. Symbolic integration has been one of the motivations for the development of the first such systems, like Macsyma.

**GENERAL TERMS**

The web has become the dominant platform for new software applications. As a result, new web applications are being developed all the time, causing the security of such applications to become increasingly important. Web applications manage users’ personal, conﬁdential, and ﬁnancial data. Vulnerabilities in web applications can prove costly for organizations; costs may include direct ﬁnancial losses, increases in required technical support, and tarnished image and brand.

Security strategies of an organization often include developing processes and choosing tools that reduce the number of vulnerabilities present in live web applications. These software security measures are generally focused on some combination of building secure software, and ﬁnding and ﬁxing security vulnerabilities in software after it has been built.

Common considerations for choosing one programming language over another include:

- how familiar staff developers are with the language;

- if new developers are going to be hired, the current state of the market for developers with knowledge of the language;

- interoperability with and re-usability of existing in house and externally-developed components;

- perceptions of security, scalability, reliability, and maintainability of applications developed using that language.

Similar considerations exist for deciding which web application development framework to use and which process to use for ﬁnding vulnerabilities. This work begins an inquiry into how to improve one part of the last of these criteria: the basis for evaluating a tool’s inclination (or disinclination) to contribute to application security.

Past research and experience reveal that different tools can have different effects on application security. The software engineering and software development communities have seen that an effective way to preclude buffer overﬂow vulnerabilities when developing a new application is to simply use a language that offers automatic memory management. Everybody knows that even if other requirements dictate that the C language must be used for development, using the safer *strlcpy* instead of *strcpy* can preclude the introduction of many buffer overﬂow vulnerabilities.

**Programming language AND Web application development framework**

*Programming language.* It is important to measure the influence that programming language choice has on the security of the software developed using that language. If such an influence exists, software engineers (or their managers) could take it into account when planning which language to use for a given job. This information could help reduce risk and allocate resources more appropriately.

There are many reasons to believe that the features of a programming language could cause differences in the security of applications developed using that language. For example, research has shown that type systems can statically find (and therefore preclude, by halting compilation) certain types of vulnerabilities. In general, static typing can find bugs (any of which could be a vulnerability) that may not have been found until the time of exploitation in a dynamically-typed language.

Also, one language’s standard libraries might be more usable, and therefore less prone to error, than another’s. A modern exception handling mechanism might help developers identify and recover from dangerous scenarios.

But programming languages differ in many ways beyond the languages themselves. Each language has its own community, and these often differ in their philosophies and values. For example, the Perl community values TMTOWTDI (“There’s more than one way to do it”), but the Zen of Python states, “[t]here should be one – and preferably, only one – obvious way to do it.” Clear documentation could play a role as well.

*Web application development framework.* Web application development frameworks provide a set of libraries and tools for performing tasks common in web application development. We want to evaluate the role that they play in the development of secure software. This can help developers make more informed decisions when choosing which technologies to use.

Recently, we have seen a trend of frameworks adding security features over time. Many modern frameworks take care of creating secure session identifiers (e.g., Zend, Ruby on Rails), and some have added support for automatically avoiding cross-site scripting (XSS) or cross-site request forgery (CSRF) vulnerabilities (e.g., Django, CodeIgniter). It is natural to wonder if frameworks that are proactive in developing security features yield software with measurably better security, but up to this point we have no data showing whether this is so.

**Early history of programming languages**

During a nine-month period in 1842–1843, Ada Lovelace translated the memoir of Italian mathematician, Luigi Menabrea about Charles Babbage's newest proposed machine, the Analytical Engine. With the article she appended a set of notes which specified in complete detail a method for calculating Bernoulli numbers with the Analytical Engine, recognized by some historians as the world's first computer program.

Herman Hollerith realized that he could encode information on punch cards when he observed that train conductors encode the appearance of the ticket holders on the train tickets using the position of punched holes on the tickets. Hollerith then encoded the 1890 American census data on punch cards.

The first computer codes were specialized for their applications. In the first decades of the 20th century, numerical calculations were based on decimal numbers. Eventually it was realized that logic could be represented with numbers, not only with words. For example, Alonzo Church was able to express the lambda calculus in a formulaic way. The Turing machine was an abstraction of the operation of a tape-marking machine, for example, in use at the telephone companies. Turing machines set the basis for storage of programs as data in the von Neumann architecture of computers by representing a machine through a finite number. However, unlike the lambda calculus, Turing's code does not serve well as a basis for higher-level languages – its principal use is in rigorous analyses of algorithmic complexity.

Like many "firsts" in history, the first modern programming language is hard to identify. From the start, the restrictions of the hardware defined the language. Punch cards allowed 80 columns, but some of the columns had to be used for a sorting number on each card. FORTRAN included some keywords which were the same as English words, such as "IF", "GOTO" (go to) and "CONTINUE". The use of a magnetic drum for memory meant that computer programs also had to be interleaved with the rotations of the drum. Thus the programs were more hardware-dependent.

To some people, what was the first modern programming language depends on how much power and human-readability is required before the status of "programming language" is granted. Jacquard looms and Charles Babbage's Difference Engine both had simple, extremely limited languages for describing the actions that these machines should perform.

**Micro-processors, -computers, -controllers**

While microprocessors, microcomputers and microcontrollers all share certain characteristics and the terms are often used interchangeably, there are certain distinctions that are used to classify them into separate categories.

*Microprocessor*

 The simplest of the three categories is the microprocessor. Also known as a CPU (Central Processing Unit), these devices are generally found at the heart of a much larger system such as a desktop computer and are primarily used as data processors. They generally consist of an arithmetic logic unit (ALU), an instruction decoder, a number of registers and digital input/output (DIO) lines. Some processors also include memory spaces such as a cache or stack which can be used for more rapid temporary storage and retrieval of data than having to access system memory. Additionally, the processor must connect to some form of data bus to access the memory and input/output peripherals external to the processor itself.

 *Microcomputer*

A microcomputer contains all the components of a computer in a small circuit, though not on a single chip. This term generally applies to laptops and desktop computers, however has fallen out of usage for these devices. The component devices of a microcomputer consist of a CPU (such as a microprocessor), memory and/or other storage devices, as well as IO devices. Several examples of I/O devices include a keyboard, display, network, etc.; but can be any device that the microcomputer uses to collect or distribute information.

*Microcontroller*

A microcontroller is, in some ways, a cross between a microprocessor and a microcomputer. Like microprocessors, the term microcontroller refers to a single device; however it contains the entire microcomputer on that single chip. Therefore a microcontroller will have a processor, on-board memory as well as a variety of IO devices. While using a microcontroller instead of a microcomputer simplifies the overall design, to accomplish this it sacrifices the flexibility. A microcomputer can be configured to have specific quantities of memory or devices attached. Microcontrollers are generally limited to the memory sizes and peripherals that the manufacturers dictate. There are a great many choices in microcontrollers and their capabilities, however this still can be a limitation in some circumstances.

**CRYPTOGRAPHY**

During this time when the Internet provides essential communication between tens of millions of people and is being increasingly used as a tool for commerce, security becomes a tremendously important issue to deal with. There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords.

One essential aspect for secure communications is that of Cryptography. The concept of securing messages through cryptography has a long history. Indeed, Julius Caesar is credited with creating one of the earliest cryptographic systems to send military messages to his generals.

Cryptography is the science of using mathematics to encrypt and decrypt data. Cryptography enables you to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient. While cryptography is the science of securing data, cryptanalysis is the science of analyzing and breaking secure communication. Classical cryptanalysis involves an interesting combination of analytical reasoning, application of mathematical tools, pattern finding, patience, determination, and luck. Cryptanalysts are also called attackers. Cryptology embraces both cryptography and cryptanalysis.

A cryptographic algorithm, or cipher, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a key – a word, number, or phrase – to encrypt the plaintext. The same plaintext encrypts to different ciphertext with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cryptographic algorithm and the secrecy of the key. A cryptographic algorithm, plus all possible keys and all the protocols that make it work comprise a cryptosystem. "Cryptography" derives from the Greek word kruptos, meaning "hidden". The key to hiding data is to devise a hiding (encryption) mechanism that is very difficult to reverse (i.e., to find the original data) without using the decryption key. Usually, the harder it is to discover the key, the more secure the mechanism. In symmetric (also called "secret-key" and, unfortunately, "private key") encryption, the same key (or another key fairly easily computed from the first) is used for both encryption and decryption.

**The study of space**

Mathematics can, broadly speaking, be subdivided into the study of quantity, structure, space, and change (i.e. arithmetic, algebra, geometry, and analysis). In addition to these main concerns, there are also subdivisions dedicated to exploring links from the heart of mathematics to other fields: to logic, to set theory (foundations), to the empirical mathematics of the various sciences (applied mathematics), and more recently to the rigorous study of uncertainty.

The study of space originates with geometry – in particular, Euclidean geometry, which combines space and numbers, and encompasses the well-known Pythagorean theorem. Trigonometry is the branch of mathematics that deals with relationships between the sides and the angles of triangles and with the trigonometric functions. The modern study of space generalizes these ideas to include higher-dimensional geometry, non-Euclidean geometries (which play a central role in general relativity) and topology. Quantity and space both play a role in analytic geometry, differential geometry, and algebraic geometry. Convex and discrete geometry were developed to solve problems in number theory and functional analysis but now are pursued with an eye on applications in optimization and computer science. Within differential geometry are the concepts of fiber bundles and calculus on manifolds, in particular, vector and tensor calculus. Within algebraic geometry is the description of geometric objects as solution sets of polynomial equations, combining the concepts of quantity and space, and also the study of topological groups, which combine structure and space. Lie groups are used to study space, structure, and change. Topology in all its many ramifications may have been the greatest growth area in 20th-century mathematics; it includes point-set topology, set-theoretic topology, algebraic topology and differential topology. In particular, instances of modern-day topology are metrizability theory, axiomatic set theory, homotopy theory, and Morse theory. Topology also includes the now solved Poincaré conjecture, and the still unsolved areas of the Hodge conjecture. Other results in geometry and topology, including the four color theorem and Kepler conjecture, have been proved only with the help of computers.

**Fibonacci numbers**

In mathematics, the Fibonacci numbers are the numbers in the following integer sequence, called the Fibonacci sequence, and characterized by the fact that every number after the first two is the sum of the two preceding ones:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, etc.

Fibonacci numbers appear to have first arisen in perhaps 200 BC in work by Pingala on enumerating possible patterns of poetry formed from syllables of two lengths. The Fibonacci sequence is named after Italian mathematician Leonardo of Pisa, known as Fibonacci. His 1202 book Liber Abaci introduced the sequence to Western European mathematics, although the sequence had been described earlier in Indian mathematics. The sequence described in Liber Abaci began with F1 = 1. Fibonacci numbers were later independently discussed by Johannes Kepler in 1611 in connection with approximations to the pentagon. Their recurrence relation appears to have been understood from the early 1600s, but it has only been in the past very few decades that they have in general become widely discussed.

Fibonacci numbers appear unexpectedly often in mathematics, so much so that there is an entire journal dedicated to their study, the Fibonacci Quarterly. Applications of Fibonacci numbers include computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure, and graphs called Fibonacci cubes used for interconnecting parallel and distributed systems. They also appear in biological settings, such as branching in trees, phyllotaxis (the arrangement of leaves on a stem), the fruit sprouts of a pineapple, the flowering of an artichoke, an uncurling fern and the arrangement of a pine cone's bracts.

*Fibonacci numbers in nature*

Fibonacci sequences appear in biological settings, in two consecutive Fibonacci numbers, such as branching in trees, arrangement of leaves on a stem, the fruitlets of a pineapple, the flowering of artichoke, an uncurling fern and the arrangement of a pine cone, and the family tree of honeybees. However, numerous poorly substantiated claims of Fibonacci numbers or golden sections in nature are found in popular sources, e.g., relating to the breeding of rabbits in Fibonacci's own unrealistic example, the seeds on a sunflower, the spirals of shells, and the curve of waves.

**ALGORITHM**

An algorithm is a reliable, definable procedure for solving a problem. The idea of the algorithm goes back to the beginnings of mathematics and elementary school students are usually taught a variety of algorithms. For example, the procedure for long division by successive division, subtraction, and attaching the next digit is an algorithm. Since a bona fide algorithm is guaranteed to work given the specified type of data and the rote following of a series of steps, the algorithmic approach is naturally suited to mechanical computation.

*Algorithms in Computer Science*

Just as a cook learns both general techniques such as how to sauté or how to reduce a sauce and a repertoire of specific recipes, a student of computer science learns both general problem-solving principles and the details of common algorithms. These include a variety of algorithms for organizing data, for numeric problems (such as generating random numbers or finding primes), and for the manipulation of data structures. A working programmer faced with a new task first tries to think of familiar algorithms that might be applicable to the current problem, perhaps with some adaptation. For example, since a variety of well-tested and well-understood sorting algorithms have been developed, a programmer is likely to apply an existing algorithm to a sorting problem rather than attempt to come up with something entirely new. Indeed, for most widely used programming languages there are packages of modules or procedures that implement commonly needed data structures and algorithms. If a problem requires the development of a new algorithm, the designer will first attempt to determine whether the problem can, at least in theory, be solved. Some kinds of problems have been shown to have no guaranteed answer. If a new algorithm seems feasible, principles found to be effective in the past will be employed, such as breaking complex problems down into component parts or building up from the simplest case to generate a solution. For example, the merge-sort algorithm divides the data to be sorted into successively smaller portions until they are sorted, and then merges the sorted portions back together.

Another important aspect of algorithm design is choosing an appropriate way to organize the data.

**The Macintosh**

By the early 1980s Steve Jobs had turned his attention to designing a radically new personal computer. Using technology that Jobs had observed at the Xerox Palo Alto Research Center (PARC), the new machine would have a fully graphical interface with icons and menus and the ability to select items with a mouse. The first such machine, the Apple Lisa, came out in 1983. The machine cost almost $10,000, however, and proved a commercial failure.

In 1984, however, Apple launched a much less expensive version. Viewers of the 1984 Super Bowl saw a remarkable Apple commercial in which a female figure runs through a group of corporate drones (representing IBM) and smashes a screen. The “Mac” sold reasonably well, particularly as it was given more processing power and memory and was accompanied by new software that could take advantage of its capabilities. In particular, the Mac came to dominate the desktop publishing market, thanks to Adobe’s PageMaker program.

In the 1990s Apple diversified the Macintosh line with a portable version (the PowerBook) that largely set the standard for the modern laptop computer. By then Apple had acquired a reputation for stylish design and superior ease of use. However, the development of the rather similar Windows operating system by Microsoft (see Microsoft Windows) as well as constantly dropping prices for IBMcompatible hardware put increasing pressure on Apple and kept its market share limited. (Apple’s legal challenge to Microsoft alleging misappropriation of intellectual property proved to be a protracted and costly failure.)

Apple’s many Macintosh variants of the later 1990s proved confusing to consumers, and sales appeared to bog down. The company was accused of trying to rely on an increasingly nonexistent advantage, keeping prices high, and failing to innovate.

However, in 1997 Steve Jobs, who had been forced out of the company in an earlier dispute, returned to the company and brought with him some new ideas. In hardware there was the iMac, a sleek all-in-one system with an unmistakable appearance that restored Apple to profitability in 1998. On the software side, Apple introduced new video-editing software for home users and a thoroughly redesigned UNIX-based operating system (see OS X). In general, the new incarnation of the Macintosh was promoted as the ideal companion for a media-hungry generation.