**The C Language**

The chosen excerpt introduces the reader to the many advantages and the rare disadvantages of the C language compared to the already existing ones. By the time the magazine where this article was published the C language was celebrating its tenth anniversary. This language was already being held at a high regard by the programming community and this excerpt came only to elevate it even higher.

The C language was created so that programs could be more portable, fast and compact. It filled a gap that existed between high-level programming languages at the time (such as BASIC, COBOL and Pascal) and low-level assembly languages that were tightly intertwined with specific processors.

At the time, the C language was mostly used for writing operating systems, utilities, languages and applications. In fact, C was even used to create the graphics sequences in *Star Trek II* and in computer-aided animation in *Star Wars:* *Return of the Jedi*.

The article dives into many of the revolutionary aspects of this language, which we’ll be summarily explaining in the topics below.

**Pointer Operations**

Since the assembly language mostly involved programming at a low-level and tampering directly with the computer’s memory compartments, the pointers in C helped make the transition from one language to another more smoothly.

The purpose of a pointer is to allow the user to access directly to the memory of an object previously defined. One example given in the article is as follows: If **x** is an integer and **p** is a pointer to **x**, **x** will be given any integer value whereas **p** will be set to the address of x, typically a hexadecimal value. This alone isn’t very impressive because Pascal, an older language, already had this feature. However, C included the ability to manipulate pointers to the users liking. For example, if you increment a pointer that is currently defined to a byte, it will point to the next byte, which makes pointers a very useful tool in array indexing. Other languages of the time like FORTRAN and BASIC did not allow pointers, which made C look better in comparison.

**Casts**

Casts are a way the C Language helps the operating system deal with a few unusual events. They allow the programmer to persuade the compiler that an object of a certain type should be treated as one of another type. This let programmers get a chance of treating pointers as integers for operations like checking whether the address they were pointing to was odd or even. Yet another improvement at the time, because this was much more complicated to implement in other languages.

**Higher-Level Models**

The C Language provided a good way of supporting data structures. The struct mechanism is used to store a bunch of data associated with a certain object. The language is merciless, which means that you need to choose the correct model you intend to work with because it’s very hard to restart. Writing code in a high-level language makes it more comprehensible and more likely to be revised and changed in the future. Thankfully, many C environments have tools that show the user which parts of the code are critical, and which are not.

**The Portability of C**

In 1983, C compilers were already being built for over 40 different machines, including the most powerful computer at the time, the Cray-1. The increasing popularity of the Unix system demanded an increased concern on portability.

Fortunately, the I/O library and data models provided by C are supported by most operating systems. The issues occur when there's no standardization on the industry relating to, for instance, the ordering of bytes in a word.

The PDP-11, a series of 16-bit minicomputers sold by DEC from 1970 into the 1990s stores bytes from low to high whilst models like the Motorola 68000 microprocessor store from high to low. The size of integers may vary and worse, some machines support 8-bit bytes, others 7 and others 9. Naturally, the portability of a program is compromised if it depends on any of these features.

However, if software isn't directly compatible, it is possible to set properties specific to a particular machine. For instance, if the size of sectors on disk varies from computer to computer, the program can properly handle that variable without the need of rewriting it from scratch. This is the case of UNIX.

**Design decisions**

The language features some wild design decisions which are definitely open for debate, such as floating-point handling or the seemingly random evaluation order in an expression.

**The C programmer**

The C model assumes the programmer makes no mistakes, resulting on sometimes not prohibiting odd constructs. Conversely, Pascal and Ada make it harder to formulate complex instructions and simultaneously detect errors more often.

The article cites C as a language for professionals and that statement still holds as of today. Its freedom allows for truly spectacular mistakes yet its portability and low-level nature are still appealing in the present day.