**Pointer Operations**

The article mentions a few aspects about the usage of pointers in C, in order to help the readers better understand how they work. 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 do 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.