

Problem 6.1

1. What are the arguments for and against representing Boolean values as single bits in memory?

PRO: Representing Boolean values as single bits allows for much more efficient use of memory, which can be a significant factor in resource-starved environments such as embedded systems. It is also the best match for the information represented and eliminated the potential for ambiguity. For instance, if flag1 and flag2 are both Boolean values represented using bytes, then how should (flag1==flag2) if flag1 is 42 and flag2 is -3?

CON: Nearly all processors are incapable of directly accessing individual bits and, instead, but access larger blocks of bits, often 32-bit or 64-bit on modern machines, and perform 'bit-banging' operations to read/write individual bits without disturbing the other bits in the value. This slows processing and can have a significant impact in speed-sensitive applications.

Problem 6.7

7. What significant justification is there for the -> operator in C and C++?

The language allows the use of

ptr->member

to dereference a pointer to a structure purely for readability and writability. Without this operator, to dereference a pointer to a structure you would need to use the syntax

(*ptr).member

If the parentheses are left off and this is written as just

*ptr.member

then most humans will read it as intended, but due to the precedence of operators in C/C++, the compiler would see it as

*(ptr.member)

Problem 6.9

9. The unions in C and C++ are separate from the records of those languages, rather than combined as they are in Ada. What are the advantages and disadvantages to these two choices?

The free unions used in C allow for more flexible and efficient code, however it makes it impossible to perform type-checking on unions. In Ada, being combined with a record enables the structure to 'tag' the union to track which of the variants it is currently associated with. However, this requires more memory and processing overhead, either or both of which might be in short supply.

Problem 6.10

10. Multidimensional arrays can be stored in row major order, as in C++, or in column major order, as in Fortran. Develop the access functions for both of these arrangements for three-dimensional arrays.

$x[\text{LEVEL}][\text{ROWS}][\text{COLS}]$;

$x[\text{level}][\text{row}][\text{col}] \Leftrightarrow x[\text{offset}]$

Row major: $\text{offset} = \text{level} * (\text{ROWS} * \text{COLS}) + \text{row} * (\text{COLS}) + \text{col}$;

Column major: $\text{offset} = \text{col} * (\text{LEVELS} * \text{ROWS}) + \text{row} * (\text{LEVELS}) + \text{level}$;

Problem 6.14

14. Write a short discussion of what was lost and what was gained in Java's designers' decision to not include the pointers of C++.

Giving the programmer direct access to memory structures via pointers allows for very powerful, efficient code to be written. It also creates the potential for the problems associated with dangling pointers and memory leaks.

Problem 6.15

15. What are the arguments for and against Java's implicit heap storage recovery, when compared with the explicit heap storage recovery required in C++? Consider real-time systems.

Java's implicit heap storage recovery largely eliminates the problems caused by dangling pointers and memory leaks. Instead, the compiler generates code to monitor each block and marks it for collection (storage recovery) upon detecting that it can no longer be referenced. However, there is computational overhead associated with this plus the programmer has no control over when the actual collection will occur, which can be problematic if it happens during time-critical periods of execution.

Problem 6.22

22. Explain how coercion rules can weaken the beneficial effect of strong typing?

The primary intent of strong typing is to identify all instances in which a type mismatch could cause a logic error during program execution. Any coercion (implicit type conversion) has the potential to let logic errors go uncaught if there are any values in the original type that are not exactly representable in the target type. However, if the target type at least has reasonable approximations to all of the values in the original type, then the likelihood of such an error causing a problem are low and the risk that such "widening" coercions present are generally considered an acceptable price for the increased readability and writability that they permit. However, "narrowing" coercions in which the target representation may not even be able to represent a rough approximation of the original value, pose a significant threat to program reliability unless the programmer takes steps to manage the risk, either through careful algorithm design or through explicit runtime range checking. Unfortunately, the very presence of narrowing coercions, such as in languages like C, often lull the programmer into ignoring these issues entirely while the compile-time errors that would otherwise occur might force the programmer to devote the necessary thought and effort into dealing with them.