NOTE: this section is incomplete -> notes for O'Reilly C++ optimization will be added

1. make the if clause more likely than the else clause
    - this allows the processor to take advantage of instructions already in the instruction fetch buffer
    - place early returns (especially for error conditions) in the else statement
2. compilers optimize for and while loops to move the conditional to the end and then execute an unconditional segement
    - switch for and while loops to do while loops (NOTE: this produces ugly code)

```
while (list != NULL) {
    ...
}

// becomes
if (list != NULL) {
    do {
        ...
    } while (list != NULL)
}
```

1. compilers generate either a binary search or a jump table for switch statements depending on the number of cases
    - avoid switches for high performance code - changes in case values can result in non-obvious performance impacts
    - convert small switches to to a series of if statements, or, if possible, a branch-free expression
    - convert medium sized switches to jump tables
    - convert large, sparse switches to smaller ones by pre-processing the cases using a hash function and switch to a jump table
    - NOTE: the hash function can result in an index into an array of function pointers
2. GCC has default static branch prediction that modifies conditionals to improve branch probability - DO NOT MANUALLY MODIFY IT
3. Avoid complex boolean operations - this affects GCC's ability to generate better static branch prediction
4. Simple traces (not shorter traces) will generate better code in GCC
    - boolean operations
    - use the LOAD->COMPARE->COMBINE pattern

5. Bitwise operators generate fewer branches and microcoded instructions than logical operators
    - combine boolean results with bitwise operators

NOTE: review and document Optimized C++ book