The Stages of Compilation

Introduction

This describes the basic stages of compilation in a very non-traditional way …

```
From: Mateo.Burtch@Eng.Sun.COM (Room 101)
Subject: YADFH (Yet Another D*** Friday Hack)

Hi.  This isn't exactly a hack, and thus probably doesn't deserve to be called
a "Friday Hack," but SOOOOO many people have been coming up to me in the
shower, saying, Please, tell us the secrets of how the compilation process
works, that I finally decided to share with you the old family recipe for:

THE STAGES OF COMPILATION

SOURCE FILE:  This is the basic "code" that the engineer writes.  The compiler
will take code such as

            dweeble(flab, krimjaw)
            sneet flab *(**smicknat[])(blugnut);
            blook krimjaw;
            {
                  blark snapdaddle liederhosen ;

                  if (fleb <= OAK_TREE)
                        while (trousers(liederhosen))
                              thud;
                  else
                        brick(flab);
            }

and turn it into output that means, roughly, "point the hose away from yourself
while watering."

Obviously, the compiler is a lot smarter than we are.

PREPROCESSOR:  The preprocessor takes all sorts of special directives, like
#ifdefs, and converts them into conditional statements (known as "#ifdefs")
that the compiler uses to prepare the file (or "#ifdef") for processing.  (This
is a technical simplification that is in most, if not all, aspects wrong.)

The preprocessor also strips the source file of comments (not to be confused
with #ifdefs), leaving it a shaken husk of its former self. These comments are
then pieced together by a separate function and used for insulation in Building
12.

COMPILER:  The heart of the whole process.  The compiler takes the language
words, such as "if," "for," and "help!" and turns it into the bits, bytes, and
subroutines that give employment to a whole host of socially challenged people.

OPTIMIZATION:  This stage makes the generated code as efficient as possible.
The optimizer does this by carefully trimming the odd and dangly parts off of
```

numerals like "5" and "9" until all the numbers are either "1"s or "0"s.

ASSEMBLER: The assembler takes "machine code" (low-level instructions done by poorly-paid workers in windowless sweatshops called "assembler lines") and does something with it.

LINKER: Before we're done, we must link together all the various object files with libraries containing macros and canned routines. Don't ask why--it's like salmon swimming upstream to spawn. You just do it.

These libraries perform a special job in the grand pageant known as programming. The pre-written macros and functions they contain allow the programmer to save valuable time when writing a program, while the arcane linker options and obscure environment variables slow the programmer down just as much. This is known as "run-time equilibrium" and is rather similar to horizontal bungee-jumping.

EXECUTABLE: This is the final product, the finished masterpiece, the piece de resistance of the whole process. This file is what the engineer had in mind when he or she started out.

It's called "core."

--Mateo