number, we'll have it push the number onto the stack. If it reads an operator, we'll have it pop two numbers from the stack, perform the operation, and then push the result back onto the stack. When the program reaches the end of the user's input, the value of the expression will be on the stack. For example, the program will evaluate the expression `30 5 - 7 *` in the following way:

1. Push 30 onto the stack.
2. Push 5 onto the stack.
3. Pop the top two numbers from the stack, subtract 5 from 30, giving 25, and then push the result back onto the stack.
4. Push 7 onto the stack.
5. Pop the top two numbers from the stack, multiply them, and then push the result back onto the stack.

After these steps, the stack will contain the value of the expression (175).

Turning this strategy into a program isn't hard. The program's `main` function will contain a loop that performs the following actions:

```
Read a "token" (a number or an operator).
If the token is a number, push it onto the stack.
If the token is an operator, pop its operands from the stack, perform the operation, and then push the result back onto the stack.
```

When dividing a program like this one into files, it makes sense to put related functions and variables into the same file. The function that reads tokens could go into one source file (`token.c`, say), together with any functions that have to do with tokens. Stack-related functions such as `push`, `pop`, `make_empty`, `is_empty`, and `is_full` could go into a different file, `stack.c`. The variables that represent the stack would also go into `stack.c`. The `main` function would go into yet another file, `calc.c`.

Splitting a program into multiple source files has significant advantages:

- Grouping related functions and variables into a single file helps clarify the structure of the program.
- Each source file can be compiled separately—a great time-saver if the program is large and must be changed frequently (which is common during program development).
- Functions are more easily reused in other programs when grouped in separate source files. In our example, splitting off `stack.c` and `token.c` from the `main` function makes it simpler to reuse the stack functions and token functions in the future.

15.2 Header Files

When we divide a program into several source files, problems arise: How can a function in one file call a function that's defined in another file? How can a func-