Once we've made these changes, we'll rebuild the `justify` program by recompiling `word.c` and `justify.c` and then relinking. There's no need to recompile `line.c`, which doesn't include `word.h` and therefore won't be affected by changes to it. With the GCC compiler, we could use the following command to rebuild the program:

```
gcc -o justify justify.c word.c line.o
```

Note the mention of `line.o` instead of `line.c`.

One of the advantages of using makefiles is that rebuilding is handled automatically. By examining the date of each file, the `make` utility can determine which files have changed since the program was last built. It then recompiles these files, together with all files that depend on them, either directly or indirectly. For example, if we make the indicated changes to `word.h`, `word.c`, and `justify.c` and then rebuild the `justify` program, `make` will perform the following actions:

1. Build `justify.o` by compiling `justify.c` (because `justify.c` and `word.h` were changed).
2. Build `word.o` by compiling `word.c` (because `word.c` and `word.h` were changed).
3. Build `justify` by linking `justify.o`, `word.o`, and `line.o` (because `justify.o` and `word.o` were changed).

Defining Macros Outside a Program

C compilers usually provide some method of specifying the value of a macro at the time a program is compiled. This ability makes it easy to change the value of a macro without editing any of the program's files. It's especially valuable when programs are built automatically using makefiles.

Most compilers (including GCC) support the `-D` option, which allows the value of a macro to be specified on the command line:

```
gcc -DDEBUG=1 foo.c
```

In this example, the `DEBUG` macro is defined to have the value `1` in the program `foo.c`, just as if the line

```
#define DEBUG 1
```

appeared at the beginning of `foo.c`. If the `-D` option names a macro without specifying its value, the value is taken to be 1.

Many compilers also support the `-U` option, which "undefines" a macro as if by using `#undef`. We can use `-U` to undefine a predefined macro or one that was defined earlier in the command line using `-D`.

predefined macros ➤ 14.3