```
    scanf("%f", &fahrenheit);
    celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;
    printf("Celsius equivalent: %.1f\n", celsius);
    return 0;
}
```

The statement

```
celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;
```

converts the Fahrenheit temperature to Celsius. Since `FREEZING_PT` stands for `32.0f` and `SCALE_FACTOR` stands for `(5.0f / 9.0f)`, the compiler sees this statement as

```
celsius = (fahrenheit - 32.0f) * (5.0f / 9.0f);
```

Defining `SCALE_FACTOR` to be `(5.0f / 9.0f)` instead of `(5 / 9)` is important, because C truncates the result when two integers are divided. The value of `(5 / 9)` would be 0, which definitely isn't what we want.

The call of `printf` writes the Celsius temperature:

```
printf("Celsius equivalent: %.1f\n", celsius);
```

Notice the use of `%.1f` to display `celsius` with just one digit after the decimal point.

2.7 Identifiers

As we're writing a program, we'll have to choose names for variables, functions, macros, and other entities. These names are called *identifiers*. In C, an identifier may contain letters, digits, and underscores, but must begin with a letter or underscore. (In C99, identifiers may contain certain "universal character names" as well.)

C99

universal character names ➤25.4

Here are some examples of legal identifiers:

```
times10   get_next_char   _done
```

The following are *not* legal identifiers:

```
10times   get-next-char
```

The symbol `10times` begins with a digit, not a letter or underscore. `get-next-char` contains minus signs, not underscores.

C is *case-sensitive:* it distinguishes between upper-case and lower-case letters in identifiers. For example, the following identifiers are all different:

```
job   joB   jOb   jOB   Job   JoB   JOb   JOB
```