START EARLY:

Programming takes time, usually with less active brain activity than a lot of other things, because much of the activity (syntax, file saving, compiling, testing) is routine and repetitive.

BE METICULOUSLY CAREFUL:

Semicolons, commas, etc., almost always matter. (This is what we mean by “syntax.”)

CONTROL YOUR ANGER AND FRUSTRATION:

In driving, it’s called “road rage.” In programming, it’s the “fist through the screen” frustration.

SPITE CAN BE HELPFUL:

This is not that hard. Legions of programmers have worked for years to produce code that sends you an incorrect phone bill or counts electronic ballots incorrectly. You are at least as smart as those people who can’t get it right.

CONTROL THE COMPLEXITY:

Separate the “logic” from the “computation.” Get the flow of control right, then add the part that actually does the work.
Jargon terms and error messages

- warnings versus errors
- undeclared
- parse error
- invalid lvalue in assignment
- missing terminating ” character
  possible start of unterminated string literal
- line numbers
- warning: control reaches end of non-void function
- warning: implicit declaration of function ‘sqrt’
Some more on formatting

The **scanf** and **printf** are similar but not identical in syntax.

\[
\text{scanf("format string", list of variables);} \\
\]

The **list of variables** needs to be a list of declared variable names separated by commas, **WITH THE VARIABLE NAMES EACH PRECEDED BY AN AMPERSAND**. White space in this list is irrelevant.

The **format string** PROBABLY ought to be a sequence of \%d (for input of **int** variables) and \%ld (for input of **long** variables) and \%le or \%lf (for input of **double** variables) each separated by a single space.

The list of \%d, \%ld, \%le, \%lf mentioned above should match up in order with the data types of the list of variable names that follows. For example,

\[
\begin{align*}
\text{int } & \text{n;} \\
\text{long } & \text{m,p;} \\
\text{double } & \text{a,b,c;} \\
\text{scanf(\"%lf %ld %d %ld %f %lf\", &a,&m,&n,&p,&b,&c);} \\
\end{align*}
\]

Mysterious things to be explained later will happen if you read with the wrong format (reading a **double** with a \%ld, for example) or if you forget the ampersand.
Some more on formatting (2)

printf("format string",list of variables);

The list of variables needs to be a list of declared variable names separated by commas, WITHOUT THE AMPERSAND. White space in this list is irrelevant.

The format string will include the \%d, \%ld, \%le, and \%lf, as well as text to be printed. Text other than these format strings is printed exactly as written, so white space is relevant.

In order to get a “newline” (a.k.a. “carriage return”) printed you need to include that explicitly at the end of the format string.

The list of \%d, \%ld, \%le, \%lf mentioned above should match up in order with the data types of the list of variable names that follows. For example,

long m,n;
double a,b,c;

printf(" a=%lf m=%ld n=%ld b=%lf c=%lf\n",a,m,n,b,c);

Mysterious things to be explained later will happen if you print with the wrong format or if you include the ampersand.
Some more on formatting (3)

To line up decimal points and produce nice table formats:

```c
printf("%12.7lf %10ld\n", a, m);
```

- print the `double` variable `a` in a field 12 spaces wide, with 7 digits to the right of the decimal point.

- print the `long` variable `m` in a field 10 spaces wide.

Comments:

The numbers are printed right justified (I think) with white space to the left.

If you print numbers too small for the format (a number 0.001 in a format `%10.2lf`, for example, then you’ll just get zero.

If the numbers won’t fit in the number of spaces asked for, C prints the numbers anyway but the decimal points won’t line up. This is only an aesthetic problem; at least you get the numbers printed out.

The format `%10ld` is large enough to accommodate all `short`, `int`, and `long` variables.

There’s no point in printing a `float` with more than seven digits to the right of the decimal point and no point in printing a `double` with more than 15 digits to the right of the decimal point, because the numbers in the machine aren’t any more accurate than that.