Small and Slow, or Large and Fast?

The code for **`fgetc`** is in the C library, so
- your code calls it
- using **one instruction per call**
- but **subroutine calls take time**.

The code for **`getc`** is in a header file, so
- a **copy is inlined** every time your code "calls" the function,
- **making your code larger**
- **but probably faster**.

On modern desktop/laptop/server platforms, this choice matters less than on older machines.

Both `fgetc` and `getc` Return One Byte or EOF

```
int fgetc (FILE* stream);
int getc (FILE* stream);
```

Why return an integer instead of a byte?

What if something goes wrong?

What's the 8-bit value that isn't a byte?

There isn't one.

`EOF` (the `int` -1) means failure.

0xFF is a byte.

Use `fputc` or `putc` to Write a Character to a Stream

Writing a character offers same two choices:

```
int fputc (int c, FILE* stream);
int putc (int c, FILE* stream);
```

`fputc` is a library function.

`putc` is a preprocessor macro.

Character to write **passed as an `int`** (native integer) for speed.

Returns character written (zero-extended from low 8 bits of **`c`**) or **`EOF` on failure**.

Use `getchar`/`putchar` as Shortcuts with `stdin`/`stdout`

There are also shortcuts

- for **reading one character from `stdin`**:

```
int getchar (void);
```

- and for **writing one character to `stdout`**:

```
int putchar (int c);
```

