When reading from input, it is usually easiest to read directly the items mentioned in the problem rather than to read full lines and process those. We might read:

- a single character, read as `int` with `getchar()` or stored in a single byte (`char`) if read with format `%c` in `scanf` to a `char *` address
- a number, with format `%d`, `%u`, `%f`, etc. in `scanf`
- a *word* (arbitrary string without whitespace, NOT just letters!), using the format `%MAXs`, where `MAX` is an integer constant, and the array needs to hold one character more, for the `\0` terminator
- a *line* (until `\n`), with `fgets(buf, sizeof(buf), stdin)`, for an array like `char buf[80];` the length read is at most one less than the array size, which includes `\0`. (ONLY use `sizeof` for an array, NOT for a pointer variable, since that would mean the size of the pointer!)
- a string consisting only of certain characters, e.g., with format `%20[0-9]` or `%63[A-za-z]`, or a a string which does NOT (`^`) contain certain characters, e.g. `%20[^,]` or `%31[^<>/]`

In all cases, one must check the return value to know whether the read has succeeded:

- for `getchar()`, it should not be `EOF` (`-1`)
- for `fgets`, it should not be `NULL`
- for `scanf`, it should be equal to the number of items attempted to be read

As an exception, we may ignore the check if our purpose is to read and ignore something, whether it appears or not: `scanf(" ")` consumes any whitespace, and `scanf("%*[^\n]")` consumes everything up to (but not including) the next newline.

Exercise: Write a program that extracts URLs from standard input. A URL is written between double quote signs `"`, starts with `http://` and contains at least one period; if a further slash character `/` appears, there must be at least one period before the slash. Print only the domain part of the URL, starting with `http://` and before the first `/` that follows, or until the end, if there is none.

Solution: As stated, finding URLs does not consider the structuring of text in lines, nor are URLs individual words (strings). We need to look for text *between* quotes `"`. This matches the way URLs are present in HTML, e.g., `<a href="http://www.upt.ro/some/page">link text</a>`. As we do not need to print the entire URL, we only need to store the (shorter) initial part with the host name.

First, we find a starting double-quote. This involves two steps: skipping all characters until a quote or end of input, and consuming the double-quote. We can do this either with an explicit loop:

```
int c; while ((c = getchar()) != '"' && c != EOF);
```

or by using the `*` modifier in `scanf` to read a string without storing it:

```
scanf("%*[^\"]"); getchar();
```

In variant 1, the first test is not enough; the quote may never appear, so one must also test for `EOF`. In variant 2, the quote must be escaped `\"` inside the format string; `scanf` only reads up to the quote, which must be read separately. We cannot read the quote in the same call: ~~`scanf("%*[^\"]\"");`~~ since if there are no characters before the quote, the first match fails and the quote is no longer read.

We use variant 2 to loop as long as a quote is found. Since the result of `scanf` does not matter, we group the two expressions into one with a comma operator; the second expression is the one tested:

```
while (scanf("%*[^\"]"), getchar() == '"') ...
```

In the cycle, after having read the quote, we check for the string `http://` by reading 7 characters and comparing. We do not read a word (with format `%7s`) since that might consume the next quote `"` (in case of a short string). This is why we read a string that does not contain `"`. Declaring arrays for the URL parts: `char prefix[8], dom[128];` the following conditions must hold for a valid URL:

- `scanf("%7[^\"]", prefix) == 1` can read an initial part, stopping at 7 chars or `"`
- `strcmp(prefix, "http://") == 0` the initial part read is `http://`
- `scanf("%127[^\"/]", dom) == 1` can read a followup part, up to `"` or `/`

If any of these conditions do not hold, we need to skip the remainder of the string within quotes, and continue the loop, searching for the next URL. If all conditions hold, we need to check whether the URL satisfies the other constraints (here, dot before /), and read the rest of the URL. To separate concerns and keep the code simple, this justifies a separate function. We choose to write the bad (error) case first, thus we negate the three conditions and connect them by OR. The code becomes:

```
char prefix[8], dom[128];
while (scanf("%*[^\"]"), getchar() == '"') // find a quote
  if (scanf("%7[^\"]", prefix) != 1 || strcmp(prefix, "http://")
      || scanf("%127[^\"/]", dom) != 1) {   // not a URL
    scanf("%*[^\"]");                 // skip until quote
    getchar();                        // get the quote if any
  } else if (checkurl(dom))           // good URL
    printf("http://%s\n", dom);
```

We are left to write the function that checks the domain part of the URL and reads the rest of it:

```
int checkurl(char *dom)
{
  int c = getchar(), retcode = 1;
  if (c == '/' && !strchr(dom, '.')) {
    fputs("no period before /\n", stderr);
    retcode = 0;
  }
  if (c != '"') {     // find end of URL
    scanf("%*[^\"]"); // skip until quote
    if ((c = getchar()) != '"') {
      fputs("premature end without \"", stderr);
      retcode = 0;
    }
  }
  return retcode;
}
```

Since reading the string `dom` stopped at `"` or `/` (or `EOF`), the function first reads the next character, and, if `/`, checks that `dom` contains at least one dot. Next, if the ending quote has not yet been read, the remainder of the URL is skipped and the existence of an ending quote checked. The return value is initialized to `1` and reset if any error condition is detected. Putting the previous code (in `main`) together with this function completes the program.