Formatted I/O: scanf

CSC 100 Introduction to programming in C/C++, Summer 2024

Marcus Birkenkrahe

February 25, 2024

Contents

1 README

- There is much more to `scanf` and `printf` than we've seen

- I/O is where the pedal hits the metal - where man meets machine

- In this notebook: conversion specifications for `scanf`

- Practice workbooks, input files and PDF solution files in GitHub

2 scanf

- A `scanf` **format string** may contain ordinary characters and

conversion specifications like `d`, `e`, `f`, `g`

- **The *conversions** allowed with `scanf` are essentially the same as those used with `printf`

- The `scanf` format string tends to contain **only** conversion specs

3 First example

- Example input:

  ```
  1  -20   .3    -4.0e3
  ```

- Put the input into a file:

  ```
  echo "1  -20 .3 -4.0e+3" > io_scanf_input
  ```

- Example program to read this input:

  ```
  int i, j;
  float x, y;

  scanf("%d%d%f%f", &i, &j, &x, &y);

  printf("|%5d|%5d|%5.1f|%10.1f|\n", i, j, x, y);
  ```

- Can you tell from the code block header where the input file is?

  ```
  C :cmdline < ../data/io_scanf_input :results output :tangle io.c
  ```

- Practice creating input on the shell yourself now:
  1. In Emacs, open a shell with `M-x eshell`
  2. Put a string into a file on the shell, list it and print it: #+end_example

COMMAND	MEANING
`echo "hello there"`	print `hello there` to the screen
`echo "hello there" > hello`	save "hello there" to file `hello`
`ls -l hello`	long listing of file `hello`
`cat hello`	print content of file `hello`

4 Main traps

- The compiler will not check that specs and variable input match up

- The `&` pointer symbol must not miss in front of the input variable

- `scanf` works in mysterious ways (we'll see why in a mo')

5 How scanf works

- `scanf` is a pattern-matching function: it tries to math input groups with conversion specifications in the format string

- For each spec, it tries to locate an item in input

- It reads the item, and stops when it can't match

- If an item is not read successfully, `scanf` aborts

- Ignores white-space: space (`" "`), TAB (`\t`), new-line (`\n`)

- Input can be on one line or spread over several lines:

- `scanf` sees a character stream (`¤` = new-line, `s` = skip'd, `r` = read):

  `••1¤-20•••.3¤•••-4.0e3¤`
  `ssrsrrrsssrrssssrrrrrr`

  - When asked to read an **integer** (`%d` or `%i`), `scanf` searches for a digit, or a +/- sign, then reads until it encounters a non-digit
  - When asked to read a **float** (`%f`, `%g`, `%e`), `scanf` looks for +/- sign, digits, decimal point, or an exponent (`e+02`, `e-02`)
  - When used with `scanf`, `%e`, `%f`, `%g` are completely interchangeable
  - When it finds a character that cannot be part of the current item, the character is returned to be read again during the scanning of the next input item or the next call of `scanf`
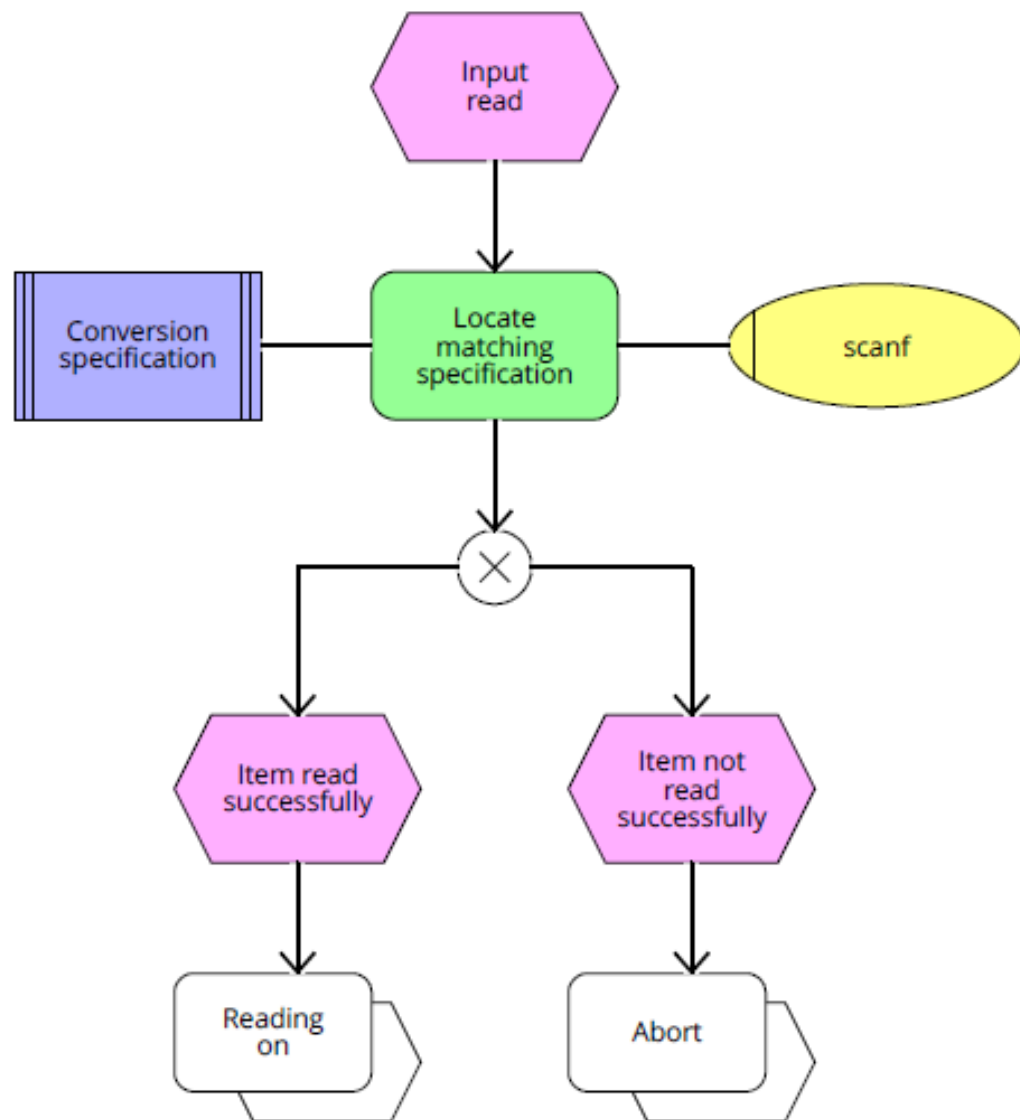
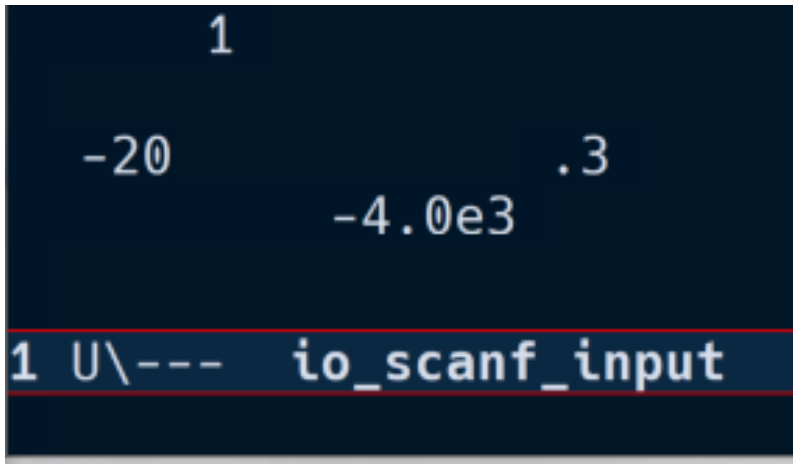Figure 1: How scanf works (Event-controlled Process Chain diagram)

Figure 2: Input file for scanf

6 Walk through example

This example has the same spec as our earlier example: `"%d%d%f%f"`,`&i,&j&x&y`.
This is what the computer "sees":

`1-20.3-4.0e3¤`

1. Expects `%d`. Stores `1` in `i`, returns -

2. Expects `%d`. Stores `-20` in `j`, returns .

3. Expects `%f`. Stores `0.3` in `x`, returns -

4. Expects `%f`. Stores `-4.0 x 10^3` in `y`, returns ¤ and finishes.

7 Ordinary characters in format strings

- `scanf` reads white-space until it reaches a symbol

- When it reaches a symbol, it tries to match to next input

- It now either continues processing or aborts

- Example: input contains `"1. 3.56 100 5 .1"` - how to scan?

```
float x=2.,y=8.,z; int i=10, j=20;

scanf("%f%f%d%d%f", &x, &y, &i, &j, &z);
printf("%.f %.2f %d %d %.1f",  x,  y,  i,  j, z);
```

- To create the input file on the shell[1]:

```
echo "1. 3.56 100 5 .1" > input
cat ./input*
```

8 Example with ordinary characters

- Open the practice file at *"Scanning ordinary characters"*.

- If the format string is **"%d/%d"** and the input is •5/•96, scanf succeeds.

- If the input is •5•/•96 , scanf fails, because the / in the format string doesn't match the space in the input.

- Upon encountering the / in •5•/•96, scanf will abort, since it expects a digit or a +/- sign. The resulting value in the second variable is not 96 but some other random number or memory address.

- To allow spaces after the first number, use **"%d/•%d"** instead.

9 Common mistakes:

1. putting & in front of variables in a printf call

   ```
   printf("%d %d\n", &i, &j);  /*** WRONG ***/
   ```

2. assuming that scanf should resemble printf formats

   ```
   scanf("%d, %d", &i, &j);
   ```

   - After storing i, scanf will try to match a comma with the next input character. If it's a space, it will abort.

---

[1]This should really work inside Emacs, too - in a bash or sh code block provided that you have one of these programs installed (e.g. via Cygwin). But Windows puts a weird symbol at the end of the filename so that it cannot be read. The cat command works with input* but the :cmdline < input command in the Org-mode code block header does not, alas.

- Only this input will work: `100, 100` but not `100 100`

3. putting a `\n` character at the end of `scanf` string

```
scanf("%d\n", &i);
```

- To `scanf`, the new-line is *white-space*. It will advance to the next white-space character and not finding one will hang forever