Formatted I/O: scanf

CSC 100 Introduction to programming in C/C++, Spring 2025

Marcus Birkenkrahe

February 11, 2025

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
```

- Emacs: Create input file

```
echo "1  -20 .3 -4.0e+3" > input  # store string in file‘input‘
cat input  # view the file ‘input‘
```

- Example program to read this input:

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

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

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

|    1|  -20|  0.3|  -4.0e+03|
```

4 Emacs VIPs only

- To run the code block above in a *new* file, you need to add a couple of header arguments:

  ```
  :main yes :includes <stdio.h>
  ```

  1. The first one wraps the code block into a `main` function
  2. The second one includes the input/output header file `stdio.h`

- 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`

- If you entered the code block and tangled it, you now have a file `io.c` in the same directory as your Org-mode file: compile and run it.

  ```
  ls -l io.c  # check the file is there
  gcc io.c -o io  # compile it and name the executable file io
  ls -l io  # check that the executable was created
  ./io < input  # run file with input file
  ```

- The last command `io < input` will not work in `eshell` because *redirection* (with <) is not supported. There may be a workaround:

  ```
  cat input | ./io  # directs output to stdout and pipes it into the file io
  ```

- Note: the file `io` has to be run `./io` on Unix-type shells to let the computer know that the file is in the current (`.`) directory. On the Windows CMD line, `io` is sufficient.

5 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 moment)

6 How scanf works

- `scanf` is a pattern-matching function: it tries to match 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:
- **Try this in OneCompiler.com now!**
- `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 (**try that in OneCompiler.com with the last format specifier**).
  - 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`.

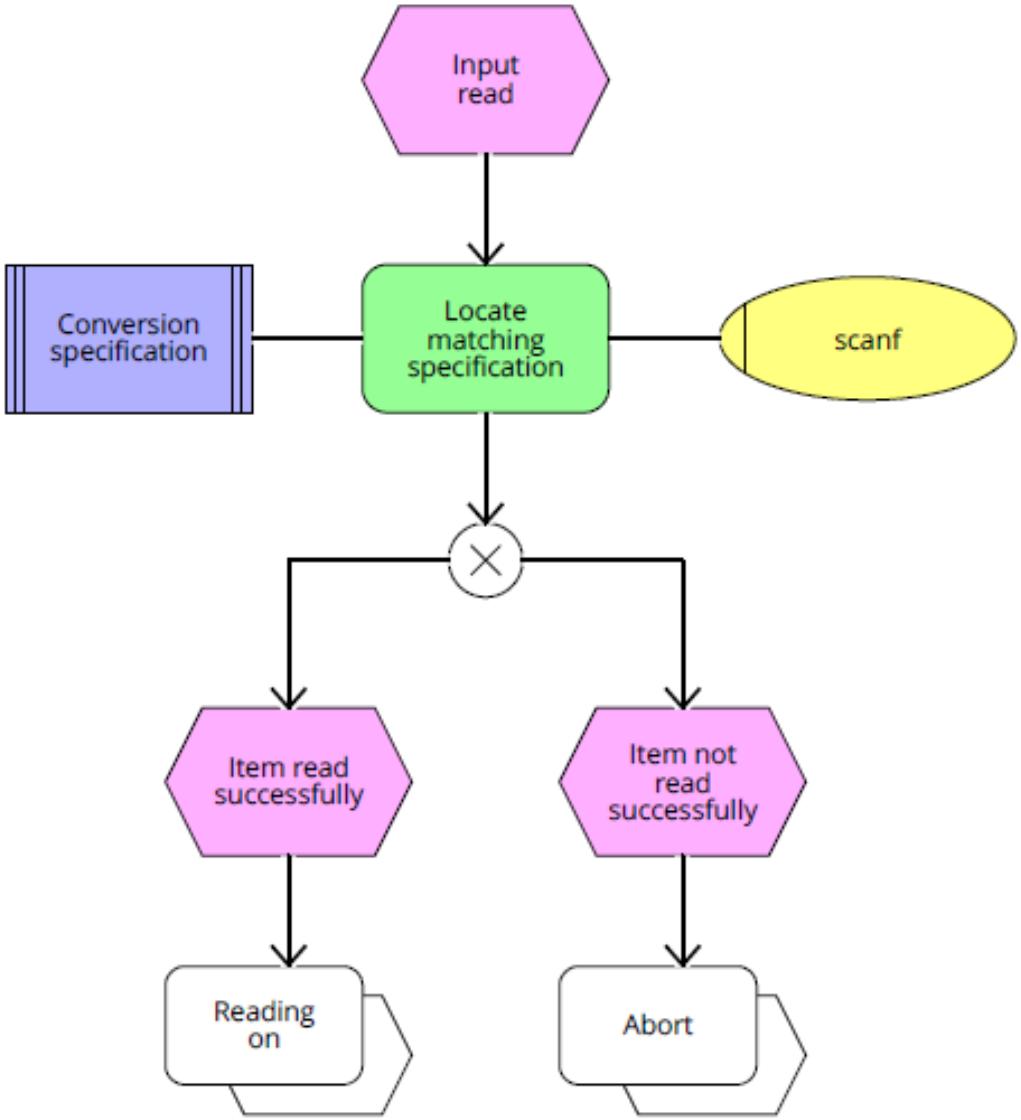

Figure 1: How scanf works (Event-controlled Process Chain diagram)

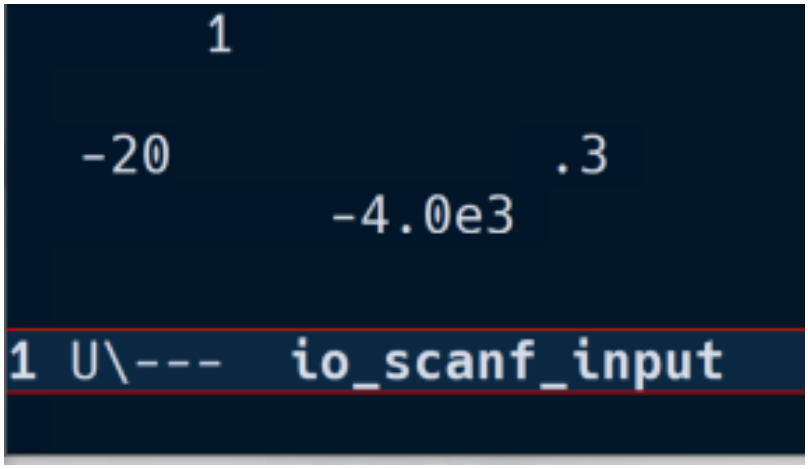

Figure 2: Input file for scanf

7 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.

8 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; // initial values
int   i=10, j=20;

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

1.0 3.56 100 5 0.1
```

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

```
echo "1. 3.56 100 5 .1" > input2
cat ./input2
```

9 Example with ordinary characters

- If the format string is `"%d/%d"` and the input is `•5/•96`, `scanf` succeeds: once the `/` is scanned, any number of white spaces are ignored.
- If the input is `•5•/•96` , `scanf` fails, because the `/` in the format string doesn't match the space in the input: an `/` is expected immediately[2].
- To allow spaces after the first number, use `"%d•/%d"` instead.

10 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);
```

[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 or MSYS2). 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.

[2] After reading the first integer, `scanf` expects to find a `/` character immediately. It encounters a whitespace character instead, which is not skipped because the whitespace is not leading (from scanf's perspective at this point; it's looking for a specific non-whitespace character, `"/"`, and aborts.

- After storing `i`, `scanf` will try to match a comma with the next input character. If it's a space, it will abort.
- For this example, only the input `100, 100` works, 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

11 PRACTICE Reading input with `scanf`

- You can open the exercises here on GitHub: tinyurl.com/scanf-practice
- In OneCompiler, create a `NEW` file for each of the exercises below.
- These exercises aren't going to be as much fun in OneCompiler as in Emacs. If you work in Emacs, you can fetch the practice file from here: tinyurl.com/scanf-practice-org
- Upload your program URL to Canvas (`"In-class practice 7:  scanf"`)

12 Scan integer and floating-point input

1. Define two *integer* variables `k`, `l`, and two *floating-point* variables `u` and `v`
2. Complete the `scanf` *format string* and enter the variables list to scan these variables
3. Use the following input: 100 -1000 .456 -9.34e2
4. Desired output:

   ```
   |  100|-1000|0.456| -934|
   ```

13 Scanning ordinary characters

1. Run the code block below with two inputs to compare:
   - `•5/•96` - this input should succeed
   - `•5 /•96` - this input should fail
2. Code:

```
int i,j;

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

printf("|%5d|%5d|\n", i, j);
```

14 Match input patterns exactly

This is useful for the programming exercise "phone numbers":

1. Use the following input: `444==++//555`
2. Complete the code below to pick up only the numbers in the input file. Remember that the format string for `scanf` must match the input format **exactly**.

```
int foo, bar;

scanf(...)
printf("The numbers were %d and %d\n", foo,  bar);
```

15 Add fractions

1. The program below prompts the user to add two fractions and then display their sum.

   The sample output for the input 5/6 and 3/4 is:

```
5/6 + 3/4 = 38/24
```

2. Use the following input:

```
5/6•
3/4
```

3. Complete the format strings below so that the program runs as intended! The output should be: `5/6 + 3/4 = 38/24`

```
// declare variables
int num1, denom1, num2, denom2, result_num, result_denom;

// scan input
scanf("...", &num1, &denom1);
scanf("...", &num2, &denom2);

// compute numerator and denominator
result_num = num1 * denom2 + num2 *denom1;
result_denom = denom1 * denom2;

// print result
printf("%d/%d + %d/%d = %d/%d\n",
       num1, denom1, num2, denom2,
       result_num, result_denom);
```

4. Modify the program so that there is only **on** `scanf` statement. Make sure that the modified program yields the same result as before.