SCANF TUTORIAL

Prerequisites. A rudimentary knowledge of simple C programs and printf, cf. [KR 88], Chap. 1.

Short Description. Input and output facilities are not part of the C language itself. Nonetheless, programs interact with their environment! In C, **printf** is commonly used for printing output and **scanf** for reading input. The syntax of these two functions are similar, with one main difference : All arguments of scanf except the first are necessarily addresses indicating where the corresponding converted input should be stored. Both, scanf and printf, are available in the standard input/output library *stdio.h*.

If you don't know what addresses are, don't worry about it ! This will become clear later on, see [KR 88], chap. 4. For the moment, it is enough to remember to add the ampersand operator '&' before every variable v , such as '&v', with one exception : no '&' is needed before an array, since the value of an array name is an address !

Definitions: White space characters, words, and strings in C

- **White space characters** are blank, tab, newline, carriage return, vertical tab, and formfeed.
- A **word** is an array of characters containing neither white space characters, nor the special characters NULL ('\0', end of string) or EOF (end of file).
- A **string** is an array of any type of characters, including white space characters, and ended by the special NULL character ('\0', end of string).
- Similarly, a **Unix file** is a file of any type of characters, ended with the special character EOF (end of file).

Example 1: Character Reading. The following program excerpt reads the next character from the standard input and stores its value at the memory address &c of the variable c.

```
#include <stdio.h>
char c;
scanf("%c", &c);
```

Example 2: Input Fields. If we want to read input lines that contain dates of the form 1 August 2015, the scanf statement is :

```
int day, year; char monthname[20];
scanf("%d %s %d", &day, monthname, &year);
```

scanf stops at the end of its format string, in this example the string "%d %s %d". Here, this string contains three **input fields**.

For every input field scanf will read all characters until a white space or EOF character is encountered. There is an important and natural exception : if the input field is a single character (see example 1), then only the next character is read, whatever this character is.

The next call to scanf resumes right after the last *converted* character. On the end of file, EOF is returned. So the following program excerpt reads a list of dates from the standard input :

```
int day, year; char monthname[20];
while (scanf("%d %s %d", &day, monthname, &year) != EOF) {...}
```

Remark. For all input fields, but not for %c, **scanf** reads the next **word** (if the character input field is %c, it reads only the next character).

Remark 2 : The input/output field %s. Note that with the input field %s **scanf** reads a word (and adds the zero character '\0' to the end). But with the output field %s, **printf** prints a string !

Example 3: Literals. As in printf, literal characters can appear in the first argument : they must match the same characters in the input. So we could read dates of the form mm/dd/yy with the scanf statement :

```
int day, month, year;
scanf("%d/%d/%d", &month, &day, &year);
```

Example 4: Error Treatment. scanf stops when it reaches the end of its format string, or when some input fails to match the control specification. It returns the number of successfully matched and assigned input items as a value. This can be used to decide how many items were found and to make an appropriate error treatment. The next call to scanf resumes right after the last converted character.

The following program excerpt illustrates a simple case. It allows us to read an input line that contains a date of form 1 August 2015 until the input format is correct :

```
(1) int day, year;
(2) char monthname[20];
(3) int d;  // d for diagnostic

(4) while ((d = scanf("%d %s %d", &day, monthname, &year)) < 3){
(5)   printf("Number returned by scanf: %d\n", d);
(6)   while (getchar() != '\n') ; // skip the rest of the input line
(7) }
```

If the three input formats are correct, scanf will return the value 3. Otherwise, it will return the number of correctly read values, i.e. 0 if the first input field is not an integer. Note that you have to skip the rest of the line, since the next call to scanf resumes searching immediately after the last converted character: So in our case, if it does not skip the rest of the line, see line 6, scanf at line 4 would read the same improperly formatted input forever!

More sophisticated error handling is possible. For instance, as in example 2, if you want it to read a list of dates from the standard input which may contain improperly formatted data and stop when EOF is encountered. Note that these improperly formatted data could be a unintentional error, or just a comment in your file you would like to skip. The following program excerpt illustrates this case :

```
 (1) int day, year;
 (2) char monthname[20];
 (3) int d;  // d for diagnostic

 (4) while ((d = scanf("%d %s %d", &day, monthname, &year)) != EOF) {
 (5)   if (d >= 0 && d < 3) goto error;  //--- error treatment

 (6)   // do some useful work
 (7)   continue; // read the next date

 (8) error:  //--- error treatment
 (9)   while (getchar() != '\n') ; //skip the rest of the input line
(10) } // end of readings
```

In an interactive mode you would perhaps add the line:

```
(9b)   printf("Number returned by scanf: %d. Try again.\n", d);
```

Example 5: Summary. To conclude, scanf reads the next *character* for a %c input field, and a *word* for all other types of input fields. All arguments of scanf are addresses :

```
 (1) char c, s[10]; int i; float f;

 (2) scanf("%c", &c);  // reads next character and puts its value in c
 (3) scanf("%s",  s);  // reads next word and converts it to a string
 (4) scanf("%i", &i);  // reads next word and converts it to an integer
 (5) scanf("%f", &f);  // reads next word and converts it to a float

 (6) scanf("%c %s %i %f", &c, s, &i, &f);  // multiple reads
 (7) printf("c=%c s=%s i=%i r=%3.1f\n", c, s, i, f);

 (8) while (scanf("%c %s %i %f", %c, s, &i, &f) < 4) {
 (9)    // do the appropriate error treatment and retry
(10) }
(11) // at this step, the 4 input fields have been readed successfully
```

Note that no '&' is used with 's' at lines 3 and 6, since the value of an array name is an address.

Reference. [KR 88] B. W. Kernighan, D. M. Ritchie, *The C Programming Language, 2nd Ed.,* Prentice Hall, 1988, chap. 7.4

© Béat Hirsbrunner, University of Fribourg, 9 September 2007, rev. March 2008, July 2010, Sept. 2011