C Programming Language: Chapter Notes

Daniel Duggan

November 2, 2025

1 Chapter 2 Notes: Input and Output Functions

C provides powerful yet simple functions for performing input and output operations. Two of the most commonly used are `printf()` and `scanf()`, both defined in the `<stdio.h>` header file.

1.1 The `printf()` Function

The `printf()` function is used to send formatted output to the standard output stream (usually the screen). Its general form is:

```
printf("control string", expressions);
```

Listing 1: General form of printf()

The control string contains ordinary characters, which are printed as they appear, and conversion specifications, each beginning with a `%` character, which tell `printf()` how to format and display the corresponding expressions.

Specifier	Meaning
%d	prints an integer value
%f	prints a floating-point value
%c	prints a single character
%s	prints a string of characters
%02d	prints an integer padded with zeros (e.g. 05)
%.2f	prints a floating-point number to 2 decimal places

For example:

```
#include <stdio.h>

int main(void) {
    int hour = 9, minute = 5;
    float temperature = 18.437;

    printf("The time is %d:%02d\n", hour, minute);
    printf("Temperature: %.2f\\textdegree C\n", temperature);

    return 0;
}
```

Listing 2: Example of printf() usage

This will print:

```
The time is 9:05
Temperature: 18.44°C
```

Here, `%02d` tells `printf()` to print the integer using two digits, padding with a leading zero if necessary, while `%.2f` rounds the floating-point number to two decimal places.

1.2 The `scanf()` Function

The `scanf()` function reads formatted input from the standard input stream (usually the keyboard). Its general form is:

```
scanf("control string", address list);
```

Listing 3: General form of scanf()

The control string works similarly to that of `printf()`, using format specifiers to determine the type of data being read. Each variable to be read must be preceded by an `&` (address-of operator), because `scanf()` needs to know where to store the input value.

Specifier	Meaning
%d	reads an integer value
%f	reads a floating-point value
%c	reads a single character
%s	reads a string of characters (stops at whitespace)

For example:

```
#include <stdio.h>

int main(void) {
    int hour, minute;

    printf("Enter a 24-hour time (hh:mm): ");
    scanf("%d:%d", &hour, &minute);

    printf("You entered %02d:%02d\n", hour, minute);

    return 0;
}
```

Listing 4: Example of scanf() usage

If the user inputs `09:30`, the program will output:

```
You entered 09:30
```

The `scanf()` format string `"%d:%d"` expects the colon literally in the input, so the user must type it (e.g. `09:30`). The colon can appear directly in the control string when you want the input to match a specific pattern.

1.3 Important Notes and Common Pitfalls

- **Always match format specifiers to variable types.** Using the wrong one (for example, reading a float with `%d`) can lead to undefined behaviour.
- **Use `&` in `scanf()`.** Forgetting it causes the program to overwrite memory at an undefined location, often leading to crashes. `scanf("%d", n);` is incorrect — it should be `scanf("%d", &n);`.
- **Whitespace handling:** `scanf()` skips leading whitespace (spaces, newlines, tabs) for most format specifiers except `%c`, which reads the next character literally — even if it's a newline.
- **`%s` stops at whitespace.** It reads characters until the first space, tab, or newline. To read full lines of text including spaces, use `fgets()` instead.
- **Input validation:** If the user types a non-numeric value when `%d` or `%f` is expected, `scanf()` fails and leaves variables unchanged. Always check its return value (the number of items successfully read).
- **Clearing the input buffer:** After using `scanf()`, unwanted newline characters can remain in the input buffer and affect the next input. To safely handle mixed input (e.g. reading both numbers and characters), use:

```
while (getchar() != '\n'); // clear leftover characters

```

- **Printing special characters:** To print a percent sign, use `%%`. For example:

```
printf("Progress: 100%% complete\n");
```

1.4 Summary

- `printf()` is used for formatted output; `scanf()` for formatted input.
- Both functions use format specifiers beginning with `%` to handle different data types.
- Each variable in `scanf()` must be preceded by `&`.
- `scanf()` skips whitespace for most specifiers but not for `%c`.
- `printf()` allows control of output precision and padding with `0` and field width.
- Always ensure the types of variables match their format specifiers.

2 Chapter 5 Notes: Selection Statements

2.1 Relational Operators

Symbol	Meaning
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

The relational operators produce 0 when false and 1 when true e.g. `1 < 2` produces 1 (true). `1> 2` produces 0 (false).
The expression `i < j < k` doesn´t have the meaning one may expect, as < is left associative. It is equivalent to `(i<j)<k` i.e. it tests whether i is less than j, and then the 1 or 0 is compared with the value of k.
To test if j lies within i and k, the correct expression is `i<j && j<k`.

2.2 Equality Operators

Symbol	Meaning
==	equal to
!=	not equal to

Like the relational operators, the equality operators are left associative and produce 0 or 1. However, they have *lower precedence* than the relational operators, meaning the expression `i<j == j<k` is equivalent to `(i<j) == (j<k)`.

2.3 Logical Operators

Symbol	Meaning
!	logical negation
&&	logical *and*
||	logical *or*

The logical operators often produce 0 or 1 as there result. They behave as follows:

- `!expr` has the value 1 if *expr* has the value 0.
- `expr1 && expr2` has the value 1 if both expressions are non-zero.
- `expr1 || expr2` has the value 1 if either `expr1` or `expr2` (or both) has a non-zero value.

2.4 if statement

An if statement takes the form

```
if (expression) statement
```

Listing 5: if statement

If you want numeroues statements, one can use the curly braces:

```
if (line_num == max_lines){

  line_num = 0;
  page_num++;
}
```

Curly braces can be add to if and if else statements even when not strictly required to aid in readability:

```
#include <stdio.h>

int main(void) {
    int score = 85;

    // Even though braces aren't strictly required for single statements,
    // adding them improves clarity and helps prevent future errors.
    if (score >= 90) {
        printf("Grade: A\n");
    } else if (score >= 80) {
        printf("Grade: B\n");
    } else {
        printf("Grade: C or below\n");
    }

    return 0;
}
```

Listing 6: Using curly braces for readability in if-else statements

2.4.1 Dangling else Problem

An else statement belongs to the nearest if statement that hasn´t already been paired with an else. Using curly braces aids in recognising which else statement belongs to which if statement.

2.5 Conditional Expressions

C provides an operator that allows an expression to produce one of two values depending on the value of a condition. This operator cosists of two symbols (? and :) which have to be used in the following way.

```
expr1 ? expr2 : expr3
```

Listing 7: Using curly braces for readability in if-else statements

The expressions can be of any type. This operator is unique in C in that it requires 3 operands instead of the usual one or two and is therefore called the ternary operator. It is read as *"if expr1 thhen expr2 else expr3"*.
expr1 is evaluated, if it is not 0, then expr2 is evaluated and it´s value is the value of the expression. If expr1 is 0, then the value of expr3 is the value of the expression as a whole. For example:

```
i = 1;
j = 2;
k = i>j ? i : j; // if i is greater than j, k is value of i (1), else it is value of
  j (2).
                 // so here, k is 2
k = (i>=0 ? i : 0) + j; // k is now 3 (i is greater than 0, so expr2 is evaluated, and
    has +j added to it)
```

2.6 Boolean Types

None in C89. Need to define a macro ifelsebraces

```
#define BOOL int
.
.
.
BOOL flag
```

In C99 there are boolean types declared basicstyle

```
_Bool flag
```

where *_Bool* is an unsigned integer type that can only be assigned 0 or 1. Giving value greater than 1 usually causes it to be assigned 1. The header file $< stdbool.h >$ can also be used which makes it easier to work with boolean values. It provides a macro, bool, that stands for *_Bool*.

2.7 Switch Statement

Similar to a case statement in VHDL/Ada. Its syntax is:

```
switch (expression) {
  case constant-expression : statements
  ...
  case constant-exprsion : statements
  default : statements
}
```

For example:

```

  int grade = 2;

switch (grade) {
    case 1:
        printf("Grade: A\n");
        break;
    case 2:
        printf("Grade: B\n");
        break;
    case 3:
        printf("Grade: C\n");
        break;
    default:
        printf("Grade: F (fail)\n");
        break;
}
```

Here, the value of the variable grade is tested against different values. As int grade = 2, when case 2 is executed, the statement inside this case will execute. Each case ends with a break; to prevent "fall-through". Without the break;, the next case is evaluated even if the condition doesn´t match.