

C basics

C Reference

main, printf & functions

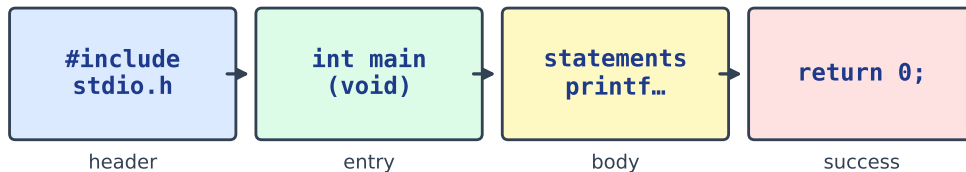
Every C program starts in `main`. `#include <stdio.h>` pulls in a header so you can use `printf` to print. A function is a named block you can call; `\n` starts a new line. `main` returns 0 to mean "success".

```
#include <stdio.h>

void greet(void) {
    printf("Hello, world!\n");
}

int main(void) {
    greet();
    printf("I am learning C.\n");
    return 0;
}
```

Every C program runs from main



Exam tip: `#include` for libraries; `main` returns 0 for success; end statements with `;`

Every C program runs from main after including headers

Variables, types & arithmetic

C needs a type for every variable: `int` (integer), `double` (floating-point), `char` (one character). `printf` uses a format specifier – `%d`, `%f`, `%c` – for each value. `/` between two ints is integer division; `%` is the remainder.

```
#include <stdio.h>

int main(void) {
    int n = 17;
    double pi = 3.14;
    char grade = 'A';
    printf("%d %.2f %c\n", n, pi, grade); // 17 3.14 A
    printf("%d %d\n", 7 / 2, 7 % 2);      // 3 1
    return 0;
}
```

The format string drives both printing and reading:

Specifier	Type	Example output
%d	int	17
%f	double (%.2f = 2 decimal places)	3.14
%c	char	A
%s	a string	Mei
%zu	a size (from <code>sizeof</code> / <code>strlen</code>)	3
%x	unsigned int, printed in base 16	ff

`scanf("%d", &n)` reads an `int` through its address (note the `&`). Input and output formats are not identical: `printf("%f", x)` prints a double, but `scanf("%lf", &x)` reads a double; input `%f` needs a float `*`. Check the returned count of successfully assigned items. Its sibling `sscanf` parses values out of a string:

```
#include <stdio.h>

int main(void) {
    char line[] = "Mei 88";
    char name[20];
    int score;
    if (sscanf(line, "%19s %d", name, &score) != 2) return 1;
    printf("%s scored %d\n", name, score); // Mei scored 88
    return 0;
}
```

Comments & style

A comment is a note for humans; the compiler ignores it. Use `//` for one line and `/* ... */` for a block. Good indentation and clear names make code easy to read.

```
#include <stdio.h>

int main(void) {
    // a single-line comment
    /* a block
       comment */
    int total = 3 + 4;    // clear names help
    printf("%d\n", total); // 7
    return 0;
}
```

Common mistakes

- Declarations and expression statements end with `;;`; a compound block `{ ... }` needs no extra semicolon. `main` should return `int` (`return 0;`).
- `printf` needs a format string: `%d` for `int`, `%f` for `double`, and `\n` for a new line.
- A wrong format like `%d` for a `double` causes undefined behaviour; it is not guaranteed merely to print garbage. Match the argument type.