IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE CS

IAE Academy free study material · Updated 29 September 2026

Programming in C Formula Sheet

INSIDE

- 17 formulas in 3 topics
- When to use each one
- Taken from the free topic notes on iaeacademy.in

Online, one page per topic: **iaeacademy.in/study/c-programming/formula-sheet**

FREE COUNSELLING

Preparing for GATE CS? Talk to IAE Academy.

Or call **+91 84204 36524**

Every day, 10:30 am to 7:30 pm

Chat on WhatsApp

Free. Not for sale. Pass it on.

C fundamentals: types, operators, expressions and control flow

N-BIT INTEGER RANGES

$$\text{unsigned } n\text{-bit range} : [0,\ 2^n - 1] \qquad \text{signed (two's complement) } n\text{-bit range} : [-2^{n-1},\ 2^{n-1} - 1]$$

Holds for any fixed-width integer type once its width n in bits and its signedness are known. Use it to check whether a constant fits the type it is stored in.

STORING AN OUT-OF-RANGE VALUE V IN A SIGNED N-BIT TYPE

$$r = v \bmod 2^n \ (0 \le r \le 2^n - 1), \qquad \text{stored value} = \begin{cases} r & \text{if } r \le 2^{n-1} - 1 \\ r - 2^n & \text{otherwise} \end{cases}$$

The standard leaves this conversion implementation-defined. On the two's complement machines these notes assume, the result is the value above. For example, 200 stored in a signed 8-bit `char` gives $200 - 256 = -56$. An unsigned n-bit type always stores r, and that case is defined by the standard.

TWO'S COMPLEMENT NEGATION AND UNSIGNED WRAPAROUND

$$-x \equiv (\sim x) + 1 \pmod{2^n} \qquad a + b \equiv (a + b) \bmod 2^n \text{ for unsigned } n\text{-bit arithmetic}$$

Here $\sim x$ is the bitwise complement. Arithmetic done in `unsigned int` (or a wider unsigned type) never overflows in the undefined-behaviour sense: it is defined to wrap modulo 2^n. Signed overflow, such as `INT_MAX + 1`, is undefined behaviour. Converting an out-of-range value to a signed type is implementation-defined (see the storing rule in the previous section).

INTEGER DIVISION IDENTITY IN C

$$a = (a/b) \cdot b + (a \,\%\, b), \qquad a/b \text{ truncated toward } 0 \quad (b \neq 0)$$

Holds for C integer operands when a/b is representable. Truncation toward zero means that the sign of `a % b` follows the sign of `a`. Division or `%` by zero is undefined behaviour.

Pointers, arrays and strings in C

1-D ARRAY INDEXING / POINTER ARITHMETIC

$$\text{Addr}(a[i]) = \text{Base}(a) + i \times \text{sizeof}(T)$$

For any array `a` with element type T. The same value is the address `a + i`. C arrays always start at index 0, and memory is byte-addressable.

2-D ROW-MAJOR ADDRESS (C ARRAYS)

$$\text{Addr}(a[i][j]) = \text{Base}(a) + (i \times C + j) \times \text{sizeof}(T)$$

The array is declared `T a[R][C]`. C is the number of COLUMNS (the row length), i is the row index and j is the column index, both starting at 0. The number of rows R does not appear.

3-D ROW-MAJOR ADDRESS

$$\text{Addr}(a[i][j][k]) = \text{Base}(a) + (i \times Y \times Z + j \times Z + k) \times \text{sizeof}(T)$$

The array is declared `T a[X][Y][Z]`, with all indices starting at 0. The first dimension X does not appear.

GENERAL ROW-MAJOR ADDRESS WITH ARBITRARY INDEX BOUNDS

$$\text{Addr}(A[i][j]) = \text{Base} + \big((i - l_1)(u_2 - l_2 + 1) + (j - l_2)\big) \times w$$

Use this for a language-neutral array A[l1..u1][l2..u2] with element size w bytes, where Base is the address of A[l1][l2]. For column-major storage, swap the roles of the two indices: Base + ((j − l2)(u1 − l1 + 1) + (i − l1)) × w. For C arrays, l1 = l2 = 0 and this reduces to the formula above.

STORAGE FOR `CHAR T[] = "...";` (SIZE LEFT OUT)

$$\text{sizeof}(t) = n + 1$$

n is the number of characters in the literal, which contains no embedded `'\0'`. The +1 is the terminator. So `char t[] = "GATE";` gives `sizeof(t)` = 5 while `strlen(t)` = 4. If a size is written, the array has that size: `char t[10] = "GATE";` has `sizeof(t)` = 10, with the unused bytes set to `'\0'`. `char t[4] = "GATE";` is legal C but stores NO terminator.

MEMBER OFFSET (ALIGNMENT RULE)

$$\text{offset}(m) = \text{ceil}\left(\frac{\text{end of previous member}}{\text{align}(m)}\right) \times \text{align}(m)$$

Apply it member by member, in declaration order. 'End of previous member' is that member's offset plus its size, and it is 0 for the first member. align(m) is the alignment of m's type, equal to its size under the convention above.

STRUCT TOTAL SIZE (WITH TRAILING PADDING)

$$\text{sizeof}(\text{struct}) = \text{ceil}\left(\frac{E}{A_{\max}}\right) \times A_{\max}$$

E is the end offset of the last member (its offset plus its size). A_max is the largest alignment among all members.

UNION SIZE

$$\text{sizeof}(\text{union}) = \text{ceil}\left(\frac{S_{\max}}{A_{\max}}\right) \times A_{\max}$$

S_max is the size of the largest member and A_max is the largest alignment among all members.

Recursion: tracing recursive programs

CALL COUNT: SINGLE (LINEAR) RECURSION

$$C(n) = C(n-1) + 1, \quad C(0) = 1 \;\Rightarrow\; C(n) = n + 1$$

For a function that makes exactly one recursive call, on $n-1$, and stops at `n == 0` without a further call (for example, a factorial with base case `n == 0`). $C(n)$ counts the initial call and the base-case call. If the base case is at `n == 1` instead (with $n \geq 1$), the count is n. Always check where the base case fires.

CALL COUNT: HALVING RECURSION

$$C(n) = C\left(\lfloor n/2 \rfloor\right) + 1 \;(n \geq 1), \quad C(0) = 1 \;\Rightarrow\; C(n) = \lfloor \log_2 n \rfloor + 2 \;\text{ for } n \geq 1$$

For a function that makes one recursive call on `n / 2` (integer division) and stops at `n == 0`, like `bin` above. The calls that print or do work (those with $n \geq 1$) number $\lfloor \log_2 n \rfloor + 1$, which is the number of bits of n. The extra call is the base case `n == 0`.

CALL COUNT: NAIVE FIBONACCI (TWO CALLS, ON N-1 AND N-2)

$$T(n) = T(n-1) + T(n-2) + 1, \quad T(0) = T(1) = 1 \;\Rightarrow\; T(n) = 2F_{n+1} - 1$$

For a function with base cases `n == 0` and `n == 1` that makes no further calls, and otherwise calls itself on $n-1$ and $n-2$. The $+1$ counts the current call. F_k is the standard Fibonacci sequence $F_0 = 0,\ F_1 = 1,\ F_k = F_{k-1} + F_{k-2}$. Of the $2F_{n+1} - 1$ calls, F_{n+1} are base-case calls (leaves) and $F_{n+1} - 1$ make further calls. The count grows as $\Theta(\varphi^n)$, where $\varphi = (1 + \sqrt{5})/2 \approx 1.618$.

UNROLLING SINGLE RECURSION ON N-1

$$T(n) = T(n-1) + c, \quad T(0) = c_0 \;\Rightarrow\; T(n) = c_0 + n\,c = \Theta(n)$$

Holds when each non-base call does a constant amount of work c and makes exactly one recursive call on $n-1$, and the base case at $n = 0$ costs c_0. Substitute $T(n-1) = T(n-2) + c$ repeatedly: $T(n) = T(n-k) + k\,c$, then put $k = n$.

TWO RECURSIVE CALLS ON N-1 (TOWER OF HANOI MOVES)

$$M(n) = 2M(n-1) + 1, \quad M(0) = 0 \;\Rightarrow\; M(n) = 2^n - 1$$

Unrolling gives $M(n) = 2^k M(n-k) + (2^k - 1)$; put $k = n$. The same shape counts calls when the base case is `n == 0` and is itself counted: $C(n) = 2C(n-1) + 1,\ C(0) = 1 \Rightarrow C(n) = 2^{n+1} - 1$.

FROM IAE ACADEMY

Preparing for GATE CS?

IAE Academy teaches GATE CS in Ballygunge, Kolkata, and online. Counselling is free: ask us about batches, fees or your preparation.

FREE COUNSELLING

Chat on WhatsApp

Call +91 84204 36524

Open every day, 10:30 am to 7:30 pm.

Scan to chat on WhatsApp

THE GATE CS COURSE

Classroom batches in Ballygunge and online batches. Fees and details are on the course page.

Scan: course page

See the course

FREE ON IAEACADEMY.IN

Notes and solved previous-year questions for Programming in C, topic by topic.

Scan: the notes

Open the notes

Found this useful? Send it to a friend preparing for GATE.

Send to a friend on WhatsApp

IAE Academy free study material. The formulas come from the free Programming in C notes on iaeacademy.in. Prepared and published by IAE Academy, Kolkata. Free to read and free to share, unaltered and with attribution.

IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE coaching in Kolkata since 2012 · EC, EE, IN, CS and IT, DA, BT and CE · in the classroom and online

227A Rash Behari Ave, Ballygunge, Kolkata 700019

+91 84204 36524 · iaeacademy.in · Updated 29 September 2026

Get directions