



Gurukul Technicals Institute

Computer education · Robotics · Science projects · Delhi

FREE EBOOK

PROGRAMMING AND WEB

C Programming Study Book

The complete study notes, practice questions and glossary for C Programming

INSIDE THIS BOOK

- Structure of a C Program and Data Types
- Operators, printf and scanf
- Making Decisions: if, else and switch
- Loops: for, while and do-while
- Arrays and Strings
- ...and 2 more chapters, practice questions and a glossary

Beginner

Students starting programming in school, college or a diploma

engineergurukul.com

+91 97116 99511

C Programming Study Book

Learn to write, compile and run C programs, from data types, operators and decisions to loops, arrays, strings, functions, pointers, structures and simple file handling.

WHO IT IS FOR	Students starting programming in school, college or a diploma
LEVEL	Beginner
SUBJECT AREA	Programming and web
WHAT IS INSIDE	7 chapters, practice questions with answers, and a glossary

CONTENTS

1	Structure of a C Program and Data Types	3
2	Operators, printf and scanf	5
3	Making Decisions: if, else and switch	8
4	Loops: for, while and do-while	10
5	Arrays and Strings	13
6	Functions and Pointers	16
7	Structures and File Handling Basics	18
	Practice questions	21
	Answers	25
	Glossary	26

Free ebook: you may read, print and share it for learning; it may not be sold or changed. © 2026 Gurukul Technicals Institute. First edition, October 2026. Contact details are on the back cover.

Structure of a C Program and Data Types

What a C program looks like

C is a compiled language. You type the source code into a file ending in `.c`, a compiler such as GCC turns it into machine code, and then you run the program it produces. Here is a complete program:

```
C
#include <stdio.h>

/* Greets the user */
int main(void)
{
    printf("Namaste, Priya!\n");
    return 0;
}
```

Each line has a job:

- `#include <stdio.h>` is a preprocessor directive. It brings in the declarations of the standard input and output functions, such as `printf` and `scanf`.
- `int main(void)` is where execution begins. Every program has exactly one `main` function, and it returns an `int` to the operating system.
- The braces `{` and `}` mark the start and end of the function body.
- Every statement ends with a semicolon.
- `return 0;` reports that the program finished successfully.
- The line above `main` is a comment, which the compiler ignores. Since C99 you can also write a single-line comment that starts with `//`.

C is case-sensitive: `main`, `Main` and `MAIN` are three different names.

With GCC on Linux you compile and run the program like this:

```
TEXT
gcc -std=c11 hello.c -o hello
./hello
```

Variables and identifiers

A variable is a named location in memory that holds a value. You must declare a variable with its type before you use it. Its name, called an identifier, follows three rules:

1. It may contain letters, digits and the underscore.
2. It must not start with a digit.
3. It must not be a keyword such as `int`, `if` or `return`.

So `total_amount` and `price2` are valid names, while `2price` and `net-pay` are not.

Basic data types

Type	Holds	Typical size	Example
<code>char</code>	one character	1 byte	<code>'A'</code>
<code>int</code>	whole numbers	4 bytes	<code>250</code>
<code>float</code>	decimals, about 6 significant digits	4 bytes	<code>99.5f</code>
<code>double</code>	decimals, about 15 significant digits	8 bytes	<code>1499.75</code>

A `char` is exactly 1 byte by definition. The other sizes are what most modern compilers on Windows and Linux use, but the C standard only fixes minimum ranges, so check them with the `sizeof` operator. Modifiers adjust the range: `short` may be smaller than `int`, `long` and `long long` (added in C99) may be larger, and `unsigned` stores only zero and positive values. On Windows compilers, for example, `long` is the same 4 bytes as `int`.

```
c
#include <stdio.h>

int main(void)
{
    int quantity = 3;
    double price = 249.50;
    char section = 'B';
    const double DISCOUNT = 50.0;    /* cannot be changed later */

    printf("Section %c: %d items, bill Rs %.2f\n", section, quantity, quantity *
price - DISCOUNT);
    printf("An int uses %zu bytes here\n", sizeof(int));
    return 0;
}
```

The first line of output is `Section B: 3 items, bill Rs 698.50`. The keyword `const` makes a variable read-only. Older programs often use a `#define` line such as `#define PI 3.14159` instead; the preprocessor replaces the name with the value before compilation.

KEY POINTS

- Every C program starts running at the main function.
- Including `stdio.h` gives your program access to `printf` and `scanf`.
- Statements end with a semicolon, and C is case-sensitive.
- A char is always 1 byte; check the size of other types on your compiler with `sizeof`.
- An identifier may use letters, digits and underscores, must not start with a digit and must not be a keyword.

TRY THESE EXERCISES

1. Type the Namaste program from this lesson into a file named `hello.c` with your own name in it, compile it with `gcc -std=c11 hello.c -o hello` and run it; then delete one semicolon, compile again and copy the error message the compiler prints into your notebook.
2. Write a program that declares one variable each of type `char`, `int`, `float` and `double` with values of your choice, prints all four with `printf`, and prints `sizeof` for each type with the `%zu` specifier; compare the sizes you get with the table in this lesson.
3. Write a bill program for a stationery shop that keeps the price of a pen in a `const double` and the quantity in an `int` and prints the amount with two decimal places; then add a line that tries to give the `const` variable a new value and note the error the compiler reports.

CHAPTER 2

Operators, printf and scanf

Arithmetic operators

C has five basic arithmetic operators: `+` (add), `-` (subtract), `*` (multiply), `/` (divide) and `%` (remainder).

Two rules catch many beginners:

- When both operands are integers, `/` gives an integer and drops the fractional part. `7 / 2` is `3`, but `7.0 / 2` is `3.5`. Since C99, integer division always truncates toward zero, so `-7 / 2` is `-3`.
- `%` works only with integers. `17 % 5` is `2`, and `n % 2` is `0` exactly when `n` is even.

Multiplication, division and remainder are done before addition and subtraction, so `2 + 3 * 4` is `14`. Use brackets when in doubt: `(2 + 3) * 4` is `20`.

Relational and logical operators

Relational operators compare two values: `==`, `!=`, `<`, `>`, `<=` and `>=`. The result is `1` for true and `0` for false. Logical operators combine conditions:

- `&&` (AND) is true only when both sides are true.
- `||` (OR) is true when at least one side is true.
- `!` (NOT) reverses a condition.

C stops evaluating `&&` as soon as the left side is false, and `||` as soon as the left side is true. This is called short-circuit evaluation.

Assignment, increment and decrement

`=` stores a value. Compound operators such as `+=` and `-=` combine an operation with assignment, so `total += price;` means `total = total + price;`.

The `++` and `--` operators add or subtract 1, and their position matters:

C FRAGMENT

```
int a = 5;
int b = a++; /* b gets 5, then a becomes 6 */
int c = ++a; /* a becomes 7, then c gets 7 */
printf("%d %d %d\n", a, b, c); /* prints 7 5 7 */
```

Printing with printf

`printf` prints text that contains placeholders called format specifiers:

Specifier	Prints	Example output
<code>%d</code>	an <code>int</code>	42
<code>%f</code>	a float or double, 6 decimals	3.500000
<code>%.2f</code>	a decimal rounded to 2 places	3.50
<code>%c</code>	a single character	A
<code>%s</code>	a string	Delhi

Escape sequences control the layout: `\n` starts a new line and `\t` inserts a tab.

Reading input with scanf

`scanf` reads values typed on the keyboard. Put `&` before each ordinary variable so that `scanf` receives its address and can store the value there. To read a double, use `%lf`; `%f` is for float.

```
#include <stdio.h>

int main(void)
{
    int quantity;
    double price;

    printf("Enter quantity and price: ");
    if (scanf("%d %lf", &quantity, &price) != 2) {
        printf("Please enter two numbers\n");
        return 1;
    }
    printf("Total: Rs %.2f\n", quantity * price);
    return 0;
}
```

If the user types `4 125.50`, the program prints `Total: Rs 502.00`. `scanf` returns how many values it read successfully, which is how this program notices bad input.

KEY POINTS

- Dividing one integer by another drops the fractional part, so $7 / 2$ is 3.
- The `%` operator gives the remainder and works only with integers.
- `x++` uses the old value and then adds 1; `++x` adds 1 first.
- In `scanf`, put `&` before ordinary variables and read a double with `%lf`.

TRY THESE EXERCISES

1. Write a program that reads two whole numbers with `scanf` and prints their sum, difference, product, integer quotient and remainder on separate lines; run it with 17 and 5 and check that the quotient is 3 and the remainder is 2.
2. Write a program that reads a temperature in Celsius into a double with `%lf`, converts it to Fahrenheit with $F = C * 9 / 5 + 32$, and prints the result with `%.2f`; test it with 37 and confirm that it prints 98.60.
3. In your notebook, predict the value of $7 / 2$, $-7 / 2$, $17 \% 5$, $2 + 3 * 4$, $(2 + 3) * 4$ and $7.0 / 2$, and the values of `a`, `b` and `c` after `int a = 5; int b = a++; int c = ++a;` then print them all in a program, using `%.1f` for $7.0 / 2$, and compare.

Making Decisions: if, else and switch

The if statement

An `if` statement runs a block only when its condition is true. In C, any non-zero value counts as true and `0` counts as false.

```
C
#include <stdio.h>

int main(void)
{
    int n;
    printf("Enter a whole number: ");
    if (scanf("%d", &n) != 1)
        return 1;

    if (n % 2 == 0)
        printf("%d is even\n", n);
    else
        printf("%d is odd\n", n);
    return 0;
}
```

The `else` part is optional and runs when the condition is false. When a block has more than one statement, enclose it in braces. Many programmers use braces even for a single statement, because it prevents mistakes when lines are added later.

Writing `if (x = 5)` does not compare anything: it stores 5 in `x`, and because 5 is non-zero the condition is always true. Use `==` to compare.

The else-if ladder

To choose between several options, chain the conditions. C checks them from top to bottom and runs only the first block whose condition is true.

```
C FRAGMENT

int balance = -250;

if (balance > 0)
    printf("Credit balance\n");
else if (balance < 0)
    printf("Amount due: Rs %d\n", -balance);
else
    printf("Account settled\n");
```

This prints Amount due: Rs 250. You can also place one `if` inside another, called a nested `if`, and combine conditions with `&&` and `||`, for example `if (age >= 18 && hasLicence)`.

The switch statement

`switch` compares one integer expression against several constant values. It suits menus well:

```
C
#include <stdio.h>

int main(void)
{
    int choice;
    double a = 12, b = 4;

    printf("1 Add, 2 Subtract, 3 Multiply, 4 Divide: ");
    if (scanf("%d", &choice) != 1)
        return 1;

    switch (choice) {
    case 1:
        printf("Result: %.2f\n", a + b);
        break;
    case 2:
        printf("Result: %.2f\n", a - b);
        break;
    case 3:
        printf("Result: %.2f\n", a * b);
        break;
    case 4:
        printf("Result: %.2f\n", a / b);
        break;
    default:
        printf("Invalid choice\n");
    }
    return 0;
}
```

Remember these rules:

- Case labels must be integer constants, such as `3` or `'A'`. You cannot use a `float` or a string.
- `break` ends the `switch`. Without it, execution falls through into the next case and runs its statements too.
- `default` runs when no case matches. It is optional.

A `switch` is often clearer than a long else-if ladder when you test one variable against fixed values, such as menu numbers or letter codes. For ranges, such as an amount between 1,000 and 5,000 rupees, use `if` statements, because a standard C case label cannot express a range.

The conditional operator

The `?:` operator picks one of two values in a single expression:

C FRAGMENT

```
int a = 8, b = 12;
int max = (a > b) ? a : b;    /* max is 12 */
```

It is handy for short choices, but switch to a full `if` statement when the logic grows.

KEY POINTS

- An non-zero value is true in C, and 0 is false.
- Use `==` to compare; a single `=` assigns a value.
- An else-if ladder runs only the first block whose condition is true.
- Case labels in a switch must be integer or character constants.
- Without `break`, a switch case falls through into the next case.

TRY THESE EXERCISES

1. Write a program that reads a student's percentage into a double with `%lf` and prints a grade with an else-if ladder: A for 75 and above, B for 60 and above, C for 45 and above, D for 33 and above, and Fail below 33; test it with 82, 60, 44.5 and 20.
2. Write a menu-driven calculator with switch that reads two numbers and a choice from 1 to 4, prints the result, prints a warning instead of dividing when the second number is 0, and prints Invalid choice for any other number; then remove one `break` and note how the output changes.
3. Write a program that reads a year and uses `&&` and `||` in one if condition to print whether it is a leap year (divisible by 4 but not by 100, or divisible by 400), and test it with 1900, 2000, 2024 and 2025.

CHAPTER 4

Loops: for, while and do-while

Why loops

A loop repeats a block of code. C has three loops: `for`, `while` and `do-while`.

Choosing a loop

Choose `for` when a counter runs through a known range, `while` when you wait for something to happen, such as a value reaching zero, and `do-while` when the body must run once before the test. Any loop can be rewritten as one of the others, so pick the one that reads most clearly.

The for loop

Use `for` when you know how many times to repeat. Its header has three parts: initialisation, condition and update.

```
c
#include <stdio.h>

int main(void)
{
    for (int i = 1; i <= 10; i++)
        printf("7 x %d = %d\n", i, 7 * i);
    return 0;
}
```

The loop sets `i` to 1, checks `i <= 10` before every round, runs the body, then adds 1 to `i`. Declaring the counter inside the header, as here, is allowed from C99 onwards. All three parts are optional, and `for ( ;; )` is an infinite loop that you leave with `break`.

The while loop

`while` checks its condition before each round, so the body may not run at all. It suits loops where the number of rounds is not known in advance:

```
c
#include <stdio.h>

int main(void)
{
    int n = 4523, sum = 0;

    while (n > 0) {
        sum += n % 10;    /* add the last digit */
        n /= 10;          /* remove the last digit */
    }
    printf("Sum of digits: %d\n", sum);    /* prints 14 */
    return 0;
}
```

The do-while loop

`do-while` checks its condition after the body, so the body always runs at least once:

C FRAGMENT

```
int count = 10;
do {
    printf("count = %d\n", count);
    count++;
} while (count < 5);
```

The condition is false from the start, yet `count = 10` is printed once. This makes `do-while` a natural fit for a menu that must be shown at least once.

break and continue

- `break` ends the loop immediately.
- `continue` skips the rest of the current round and moves on to the next one.

C FRAGMENT

```
for (int i = 1; i <= 10; i++) {
    if (i % 3 == 0)
        continue; /* skip multiples of 3 */
    if (i > 7)
        break; /* stop once we pass 7 */
    printf("%d ", i);
}
/* prints: 1 2 4 5 7 */
```

Nested loops

A loop inside another loop is common for tables and patterns. The inner loop runs completely for every round of the outer loop:

C FRAGMENT

```
for (int row = 1; row <= 3; row++) {
    for (int col = 1; col <= row; col++)
        printf("%d", col);
    printf("\n");
}
```

This prints `1`, then `12`, then `123` on three separate lines.

If a loop never ends, check that something inside it changes the variable its condition depends on.

KEY POINTS

- Use for when the number of rounds is known in advance.
- while checks its condition first, so its body may run zero times.
- do-while checks its condition last, so its body always runs at least once.
- break leaves the loop; continue skips to the next round.

TRY THESE EXERCISES

1. Write a program that prints the multiplication table of a number entered by the user with a for loop, then write a second version that uses a while loop, and check that both print the same ten lines.
2. Write a program that reads a whole number and uses a while loop to reverse it and count its digits, so that 4523 prints 3254 and 4 digits; then wrap the whole program in a do-while loop that asks whether to try another number (1 for yes) after each result.
3. Use nested for loops to print a right-angled triangle of stars five rows high, then change the inner loop so that it prints the number pattern 1, 12, 123, 1234 and 12345 on five separate lines.

CHAPTER 5

Arrays and Strings

Arrays

An array stores several values of the same type in consecutive memory locations, under one name. Each element is reached by its index, which starts at 0.

C FRAGMENT

```
int marks[5] = {78, 85, 62, 90, 71};
printf("%d\n", marks[0]); /* 78, the first element */
printf("%d\n", marks[4]); /* 71, the last element */
```

An array of size 5 has indexes 0 to 4. Some rules to remember:

- The size is fixed when the array is created.
- If you give fewer initial values than the size, the remaining elements are set to 0. So `int a[5] = {1, 2};` holds 1, 2, 0, 0, 0.
- You can leave out the size when you initialise: `int a[] = {4, 8, 15};` has 3 elements.
- C does not check indexes. Using `marks[5]` above is undefined behaviour: it can crash the program or silently damage other data.

Loops and arrays go together:

C

```
#include <stdio.h>

int main(void)
{
    int sales[7] = {1200, 950, 1430, 800, 1600, 2100, 1750};
    int total = 0;

    for (int i = 0; i < 7; i++)
        total += sales[i];

    printf("Weekly sales: Rs %d\n", total);           /* 9830 */
    printf("Daily average: Rs %.2f\n", total / 7.0); /* 1404.29 */
    return 0;
}
```

Dividing by `7.0` instead of `7` keeps the decimal part. A two-dimensional array such as `int table[3][4]`; is an array of rows, and you reach one element with two indexes, `table[row][col]`.

Strings

C has no separate string type. A string is an array of `char` that ends with the null character `'\0'`, which marks where the text stops.

C FRAGMENT

```
char city[10] = "Pune"; /* uses 5 of the 10 places: P, u, n, e and the null
character */
```

So the array needs room for the text plus one more character for the terminator.

Reading a word with `scanf("%s", city)` needs no `&`, because the name of an array already converts to the address of its first element. `%s` stops at the first space, so use `fgets(city, sizeof city, stdin)` to read a whole line. The old `gets` function was removed in C11 because it cannot prevent overflow.

String functions

The header `string.h` provides the common operations:

Function	What it does
<code>strlen(s)</code>	counts the characters, not including the null character
<code>strcpy(dest, src)</code>	copies <code>src</code> into <code>dest</code>
<code>strcat(dest, src)</code>	joins <code>src</code> onto the end of <code>dest</code>
<code>strcmp(a, b)</code>	gives 0 if equal, negative if <code>a</code> sorts first, positive otherwise

```

#include <stdio.h>
#include <string.h>

int main(void)
{
    char first[20] = "Rohan";
    char full[40];

    strcpy(full, first);
    strcat(full, " Mehta");
    printf("%s has %zu characters\n", full, strlen(full));    /* 11 */

    if (strcmp(first, "Rohan") == 0)
        printf("Names match\n");
    return 0;
}

```

You cannot compare strings with `==` or copy them with `=`; use `strcmp` and `strcpy` instead.

KEY POINTS

- Array indexes start at 0, so an array of size n ends at index $n - 1$.
- C does not check array bounds; going outside them is undefined behaviour.
- A string is a char array ending in the null character, so it needs one extra place.
- Compare strings with `strcmp`, which returns 0 when they are equal.

TRY THESE EXERCISES

1. Write a program that reads the marks of five students into an int array and prints the total, the average to two decimal places, the highest mark and the lowest mark.
2. Write a program that reads one word with `scanf %s` into a char array of size 30, prints its length with `strlen`, and then prints the word backwards with a for loop that runs from the last index down to 0.
3. Write a program with two char arrays holding your first name and surname that builds the full name in a third char array large enough for both names and the space (copy the first name in with `strcpy`, then add the space and the surname with `strcat`), and prints it with its length; then use `strcmp` to check whether a password typed by the user matches a stored one.

Functions and Pointers

Writing functions

A function is a named block of code that does one job. Splitting a program into functions makes it easier to read, test and reuse.

```
C

#include <stdio.h>

double lineTotal(int qty, double rate); /* prototype */

int main(void)
{
    printf("Total: Rs %.2f\n", lineTotal(4, 125.5)); /* 502.00 */
    return 0;
}

double lineTotal(int qty, double rate) /* definition */
{
    return qty * rate;
}
```

- The prototype tells the compiler the function's name, return type and parameter types before the function is used.
- The definition contains the body.
- `return` sends a value back to the caller. A function declared `void` returns no value.

Pass by value

C passes arguments by value: the function receives copies, so changing a parameter does not change the caller's variable.

```
C TOP

void change(int n)
{
    n = 50; /* changes only the local copy */
}
```

After `int x = 5; change(x);` the value of `x` is still 5.

Ordinary variables declared inside a function are local: they exist only while the function runs and other functions cannot see them, so two functions can both use a variable called `i` without any clash. A variable declared outside every function is global and visible to all of them, but heavy use of globals makes a program hard to follow.

Recursion

A function that calls itself is recursive. It needs a base case that stops the calls:

C TOP

```
long long factorial(int n)
{
    if (n <= 1)
        return 1;           /* base case */
    return n * factorial(n - 1); /* recursive case */
}
```

`factorial(5)` returns 120.

Pointers

A pointer is a variable that stores the address of another variable. Two operators work with pointers:

- `&x` gives the address of `x`.
- `*p` gives the value stored at the address held in `p`. This is called dereferencing.

C FRAGMENT

```
int x = 10;
int *p = &x;           /* p holds the address of x */
printf("%d\n", *p);    /* 10 */
*p = 25;               /* changes x through the pointer */
printf("%d\n", x);     /* 25 */
```

In a declaration, the star means "pointer to", so `p` above is a pointer to `int`. Pointers let a function change the caller's variables:

C

```
#include <stdio.h>

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main(void)
{
    int x = 3, y = 8;
    swap(&x, &y);
    printf("x = %d, y = %d\n", x, y); /* x = 8, y = 3 */
    return 0;
}
```

Pointers and arrays

In almost every expression an array name converts to a pointer to its first element (`sizeof` is the main exception: it gives the size of the whole array), and `a[i]` means exactly the same as `*(a + i)`. That is why a function that receives an array can change the caller's elements.

Give a pointer that points nowhere the value `NULL`, and never dereference a pointer that has not been set to a valid address.

KEY POINTS

- A prototype declares a function's name, return type and parameters before it is used.
- C passes arguments by value, so a function works on copies.
- The `&` operator gives an address; the star operator reads or writes the value at an address.
- Passing addresses lets a function such as `swap` change the caller's variables.
- A recursive function must have a base case.

TRY THESE EXERCISES

1. Write a function `int isEven(int n)` that returns 1 for an even number and 0 for an odd one, put its prototype above `main`, and call it in a loop to print whether each number from 1 to 10 is even or odd.
2. Write a recursive function that returns the factorial of `n` and a second recursive function that returns the sum of the digits of a number, each with a clear base case; check that `factorial(5)` gives 120 and that the digit sum of 4523 is 14.
3. Type the `swap` function from this lesson, then write a version called `swapCopy` that takes two plain `int` parameters instead of pointers; call both on `x = 3` and `y = 8` and print `x` and `y` after each call to show that only the pointer version changes them.

CHAPTER 7

Structures and File Handling Basics

Structures

A structure groups related variables of different types under one name. Each variable inside it is called a member.

```
struct Student {
    int roll;
    char name[30];
    float percentage;
};
```

This defines a new type, `struct Student`. You create variables of it and reach the members with the dot operator:

```
struct Student s1 = {101, "Ananya Iyer", 88.5f};
s1.roll = 102;
printf("%d %s %.1f\n", s1.roll, s1.name, s1.percentage); /* 102 Ananya Iyer 88.5 */
```

An array of structures holds a set of records, such as `struct Student batch[40];`, and `batch[0].name` is the name in the first record. When you have a pointer to a structure, use the arrow operator: `p->roll` is short for `(*p).roll`. You can also give the type a shorter name with `typedef struct Student Student;`, after which `Student s2;` works.

A structure can be passed to a function like any other variable, and it is copied just as an `int` is. For a large structure, pass its address instead: this avoids the copy and lets the function update the original record.

File handling basics

Data in variables disappears when the program ends; files keep it. Every file operation in C uses a `FILE` pointer from `stdio.h`:

1. Open the file with `fopen(name, mode)`.
2. Check that the result is not `NULL`, which means the file could not be opened.
3. Read or write.
4. Close it with `fclose`, which also writes out any data still waiting in memory.

Mode	Meaning
"r"	read; fails if the file does not exist
"w"	write; creates the file, or erases it if it already exists
"a"	append; creates the file if needed and adds to the end

Add a plus sign for both reading and writing, as in `"r+"`, and `b` for binary files, as in `"rb"`.

```

#include <stdio.h>

struct Item {
    char name[20];
    int qty;
    double rate;
};

int main(void)
{
    struct Item items[3] = {
        {"Notebook", 4, 45.0},
        {"Pen", 10, 12.5},
        {"Geometry box", 1, 180.0}
    };

    FILE *fp = fopen("bill.txt", "w");
    if (fp == NULL) {
        printf("Could not create bill.txt\n");
        return 1;
    }
    for (int i = 0; i < 3; i++)
        fprintf(fp, "%s,%d,%.2f\n", items[i].name, items[i].qty, items[i].qty *
items[i].rate);
    fclose(fp);

    char line[80];
    fp = fopen("bill.txt", "r");
    if (fp == NULL)
        return 1;
    while (fgets(line, sizeof line, fp) != NULL)
        printf("%s", line);
    fclose(fp);
    return 0;
}

```

The program writes three lines, `Notebook,4,180.00`, `Pen,10,125.00` and `Geometry box,1,180.00`, then reads them back and prints them.

`fprintf` and `fscanf` work like `printf` and `scanf` but take the file pointer as their first argument. `fgets` reads one line at a time and returns `NULL` at the end of the file.

KEY POINTS

- A structure groups members of different types into one new type.
- Use the dot operator with a structure variable and the arrow operator with a pointer to one.
- Always check that `fopen` did not return `NULL` before using the file.
- Mode "w" erases an existing file, while "a" adds to the end of it.
- Close every file with `fclose` when you have finished with it.

TRY THESE EXERCISES

1. Define a struct Student with roll number, name and percentage, read three students from the keyboard into an array of structures, and print the record with the highest percentage; then write a function void addGrace(struct Student *p) that adds 2 to the percentage using the arrow operator and call it for the first student.
2. Write a program that asks for an item name, quantity and rate and adds one line to a file named sales.txt using fopen with mode "a" and fprintf; run it three times and open sales.txt in a text editor such as Notepad to check that all three lines are there.
3. Write a program that opens sales.txt with mode "r", prints a message and stops if fopen returns NULL, and otherwise reads the file line by line with fgets, prints each line with a line number in front and closes it with fclose; then change the writing program to mode "w", run it once, and note what happened to the old lines.

CHECK YOURSELF

Practice questions

Choose the best answer for each question. The answers, with a short explanation, are in the next chapter.

1. Where does a C program begin executing?

- a) At the first line of the source file
- b) In the first function written in the file
- c) In the main function
- d) Inside the stdio.h header file

2. What is the value of sizeof(char) in C?

- a) 2, on every compiler
- b) 4 on 32-bit systems and 8 on 64-bit systems
- c) It depends on the character stored
- d) 1, on every compiler

3. Which of these is a valid C identifier?

- a) total_amount
- b) 2ndPrice
- c) float
- d) net-salary

4. Which header file must be included to use printf and scanf?

- a) string.h
- b) math.h
- c) stdio.h
- d) stdlib.h

5. What does the statement return 0; at the end of main usually tell the operating system?

- a) The program ended with an error
- b) The program finished successfully
- c) The main function returns no value
- d) The program should start again

6. What is the value of $17 \% 5$ in C?

- a) 2
- b) 3
- c) 3.4
- d) 5

7. If a and b are int variables holding 7 and 2, what is the value of a / b ?

- a) 3.5
- b) 3
- c) 4
- d) 1

8. What does this code print? `int a = 5; int b = a++; printf("%d %d", a, b);`

- a) 5 6
- b) 6 6
- c) 5 5
- d) 6 5

9. Which format specifier should scanf use to read a value into a double variable?

- a) %f
- b) %d
- c) %lf
- d) %c

10. Why is the & sign written before n in `scanf("%d", &n)`?

- a) It converts the typed text into an integer before storing it
- b) It prints the current value of n before reading a new one
- c) It passes the address of n, so scanf can store the typed value in n
- d) It makes n a constant so its value cannot change later

11. What does this code print? `int x = 0; if (x = 5) printf("yes"); else printf("no");`

- a) no
- b) yes
- c) Nothing is printed
- d) It does not compile

12. In a switch statement, what happens when the matching case has no break statement?

- a) Execution continues into the statements of the following cases
- b) The program stops with a run-time error at that case
- c) The whole switch is skipped and the next statement runs
- d) The default case runs instead of the matching case

13. Which of these cannot be used as a case label in a C switch statement?

- a) An integer constant such as 3
- b) A character constant such as 'A'
- c) A floating-point value such as 2.5
- d) An enumeration constant

14. What is the value of max after `int a = 8, b = 12; int max = (a > b) ? a : b;` runs?

- a) 12
- b) 8
- c) 1
- d) 20

15. Which value of a condition does C treat as false?

- a) 1
- b) -1
- c) Any negative number
- d) 0

16. How many times does the body of `for (int i = 0; i < 5; i++)` run?

- a) 4
- b) 5
- c) 6
- d) It never stops

17. Which statement always executes its body at least once?

- a) The while loop
- b) The for loop
- c) The if statement
- d) The do-while loop

18. What does continue do inside a loop?

- a) Skips the rest of the current round and moves to the next round
- b) Ends the loop immediately and carries on after it
- c) Restarts the loop from its very first round
- d) Pauses the loop until the user presses a key

19. What does this code print? `int i = 1; while (i < 20) i = i * 2; printf("%d", i);`

- a) 16
- b) 20
- c) 64
- d) 32

20. What does this code print? `for (int i = 10; i > 0; i -= 3) printf("%d ", i);`

- a) 10 7 4 1
- b) 10 7 4
- c) 10 7 4 1 -2
- d) 7 4 1

21. In the declaration `int a[10];` what is the index of the last element?

- a) 10
- b) 11
- c) 9
- d) 1

22. How many char places does an array need to store the string "India", including its terminator?

- a) 5
- b) 6
- c) 7
- d) 4

23. What does strcmp return when the two strings it compares are equal?

- a) 1
- b) 0
- c) -1
- d) The length of the strings

24. What does this code print? `int a[5] = {1, 2}; printf("%d", a[3]);`

- a) 2
- b) An unpredictable value
- c) It does not compile
- d) 0

25. Why does `scanf("%s", name)` not need an & when name is a char array?

- a) Strings in C do not have addresses
- b) The array name already converts to the address of its first element
- c) scanf adds the & automatically for every variable
- d) %s reads only a single character

26. What does the & operator give when written before a variable, as in `&x`?

- a) The value stored in x
- b) The size of x in bytes
- c) The logical AND of x with itself
- d) The memory address of x

27. If p was declared as `int *p` and then set with `p = &x`, which expression gives the value of x through p?

- a) &p
- b) p
- c) &x
- d) *p

28. What does this program print? void change(int n) { n = 50; } int main(void) { int n = 5; change(n); printf("%d", n); return 0; }

- a) 5
- b) 50
- c) 0
- d) 55

29. What is a function prototype in C?

- a) The first call made to a function, usually inside main
- b) A function that takes no parameters and returns no value
- c) A declaration that gives a function's name, return type and parameter types before it is used
- d) A copy of a function that the compiler makes for faster calls

30. If a is an int array, which expression means the same as a[2]?

- a) *(a + 2)
- b) *a + 2
- c) &a + 2
- d) a + 2

31. Which operator accesses a member of a structure through a pointer to that structure?

- a) .
- b) &
- c) ->
- d) ::

32. What happens when fopen opens a file that already exists in "w" mode?

- a) The old contents are kept and new data is added after them
- b) Its old contents are erased and writing starts from the beginning
- c) The file is opened for reading only, and writes are refused
- d) fopen fails and returns NULL because the file already exists

33. What does fopen return when it cannot open the requested file?

- a) -1
- b) NULL
- c) EOF
- d) The name of the file

34. Which mode adds new lines to the end of an existing file without losing what is already in it?

- a) "w"
- b) "r"
- c) "a"
- d) "rb"

35. Given struct Student { int roll; char name[30]; } s; which statement sets the roll member of s to 12?

- a) s.roll = 12;
- b) s->roll = 12;
- c) Student.roll = 12;
- d) roll.s = 12;

Answers

1. (c) **In the main function.** Execution always starts at main, wherever it appears in the file; other functions run only when they are called.
2. (d) **1, on every compiler.** The C standard defines `sizeof(char)` as exactly 1; the sizes of the other types can vary between compilers.
3. (a) **total amount.** Identifiers may contain letters, digits and underscores but cannot start with a digit, cannot be a keyword such as `float`, and cannot contain a hyphen.
4. (c) **stdio.h.** `stdio.h` (standard input and output) declares `printf`, `scanf` and the file functions.
5. (b) **The program finished successfully.** By convention a return value of 0 from main means success; a non-zero value signals a problem.
6. (a) **2.** The `%` operator gives the remainder: 17 divided by 5 is 3 with remainder 2.
7. (b) **3.** When both operands are integers, the division is done in integers and the fractional part is dropped, so `7 / 2` gives 3.
8. (d) **6 5.** With postfix `a++`, `b` receives the old value 5 and then `a` is increased to 6.
9. (c) **%lf.** In `scanf`, `%f` is for float and `%lf` is for double; using the wrong one stores the value incorrectly.
10. (c) **It passes the address of n, so scanf can store the typed value in n.** `scanf` needs to know where to put the value, so it is given the address of the variable with the `&` operator.
11. (b) **yes.** `x = 5` is an assignment, not a comparison; its value is 5, which is non-zero, so the condition is true and "yes" is printed.
12. (a) **Execution continues into the statements of the following cases.** Without `break`, control falls through and keeps running the statements of the next cases until a `break` or the end of the switch.
13. (c) **A floating-point value such as 2.5.** Case labels must be integer constant expressions; character and enumeration constants are integers, but 2.5 is a floating-point value.
14. (a) **12.** `a > b` is false, so the conditional operator gives the value after the colon, which is `b`, 12.
15. (d) **0.** In C only zero is false; every non-zero value, positive or negative, is treated as true.
16. (b) **5.** `i` takes the values 0, 1, 2, 3 and 4; when `i` becomes 5 the condition `i < 5` is false and the loop ends.
17. (d) **The do-while loop.** `do-while` tests its condition after running the body, so the body runs once even if the condition is false from the start.
18. (a) **Skips the rest of the current round and moves to the next round.** `continue` jumps straight to the next round of the loop (for a `for` loop, to its update step); `break` is the statement that ends the loop.
19. (d) **32.** `i` doubles through 2, 4, 8, 16 and 32; at 32 the condition `i < 20` is false, so the loop stops and 32 is printed.
20. (a) **10 7 4 1.** `i` starts at 10 and drops by 3 each round: 10, 7, 4, 1; the next value, -2, fails `i > 0`, so it is not printed.
21. (c) **9.** Indexes start at 0, so an array of 10 elements uses indexes 0 to 9.

22. (b) 6. "India" has 5 letters, and every C string needs one more place for the null character that ends it.
23. (b) 0. strcmp returns 0 for equal strings, a negative value if the first sorts earlier and a positive value if it sorts later.
24. (d) 0. When an array is given fewer initial values than its size, the remaining elements are set to zero, so a[3] is 0.
25. (b) **The array name already converts to the address of its first element.** An array name used as an argument becomes a pointer to its first element, which is already the address scanf needs.
26. (d) **The memory address of x.** The unary & is the address-of operator; it gives the location of x in memory.
27. (d) *p. Writing the star before a pointer dereferences it, giving the value stored at the address it holds; p alone is the address itself.
28. (a) 5. C passes arguments by value, so change works on a copy; the n in main is still 5.
29. (c) **A declaration that gives a function's name, return type and parameter types before it is used.** A prototype such as double lineTotal(int qty, double rate); lets the compiler check calls to a function that is defined later.
30. (a) *(a + 2). a[i] is defined as *(a + i); the option without brackets adds 2 to the first element instead of moving two elements along.
31. (c) ->. p->member is the arrow operator, short for (*p).member; the dot operator is used with a structure variable itself.
32. (b) **Its old contents are erased and writing starts from the beginning.** "w" creates a new file or truncates an existing one to zero length; use "a" to keep the contents and add to the end.
33. (b) NULL. fopen returns a null pointer on failure, which is why a program should check for NULL before using the file.
34. (c) "a". Append mode "a" writes at the end of the file and creates the file if it does not exist.
35. (a) s.roll = 12;. A member of a structure variable is reached with the dot operator; the arrow operator is only for pointers to structures.

WORD LIST

Glossary

Array

A fixed-size group of values of the same type stored in consecutive memory locations under one name, with each element reached by an index that starts at 0.

break

A statement that immediately ends the innermost loop or switch that contains it, so execution continues with the statement after that loop or switch.

Compiler

A program, such as GCC, that translates C source code into machine code the computer can run.

Conditional operator

The ?: operator, which chooses one of two values in a single expression depending on whether a condition is true or false.

Data type

The kind of value a variable can hold, such as char, int, float or double, which also decides how much memory the variable uses.

Dereferencing

Using the star operator on a pointer to read or change the value stored at the address the pointer holds.

Escape sequence

A backslash followed by a character that stands for something hard to type inside a string, such as \n for a new line or \t for a tab.

FILE pointer

A pointer of type FILE * returned by fopen, which a program passes to functions such as fprintf, fgets and fclose to work with an open file.

Format specifier

A placeholder starting with % in a printf or scanf string, such as %d for an int or %lf for reading a double, that says what type of value goes in that place.

Function prototype

A declaration of a function's name, return type and parameter types placed before the function is used, so that the compiler can check every call to it.

Header file

A file ending in .h, such as stdio.h or string.h, that holds declarations of functions and types and is brought into a program with #include.

Identifier

The name given to a variable, function or other item, which may use letters, digits and underscores but must not start with a digit or be a keyword.

Keyword

A word reserved by the C language, such as int, if, while or return, which cannot be used as an identifier.

Local variable

An ordinary variable declared inside a function, which exists only while that function runs and cannot be seen by other functions, unlike a global variable declared outside every function.

main function

The function where every C program starts running; it returns an int to the operating system, with 0 meaning the program finished successfully.

Null character

The character '\0' that marks the end of a string in C, which is why a char array needs one more place than the text it holds.

NULL pointer

A pointer holding the special value NULL to show that it points to no valid object; fopen, for example, returns NULL when it cannot open a file.

Pass by value

The way C passes arguments to a function, giving it copies of the values so that changing a parameter does not change the caller's variable.

Pointer

A variable that stores the memory address of another variable, declared with a star, as in `int *p`.

Preprocessor directive

A line beginning with `#`, such as `#include` or `#define`, that the preprocessor handles before the code is compiled.

Recursion

A technique in which a function calls itself, which needs a base case that stops the calls, as in a function that works out a factorial.

sizeof

An operator that gives the size in bytes of a type or variable, such as `sizeof(int)`, and whose result is printed with the `%zu` format specifier.

String

In C, an array of `char` that ends with the null character, handled with functions from `string.h` such as `strlen`, `strcpy`, `strcat` and `strcmp`.

Structure

A user-defined type, declared with the keyword `struct`, that groups related members of different types, such as a student's roll number, name and percentage, under one name.

Undefined behaviour

What happens when a program breaks a rule of the C standard, such as using an array index outside its bounds, where the result is unpredictable and may be a crash or silently damaged data.



Gurukul Technicals Institute

Computer education institute in Delhi, with authorised member centres, robotics classes and science projects, working models and DIY kits.

COURSES

Computer & Office Skills · Accounting & Business Software · Programming & Web Development · Design & Digital Skills · Hardware & Networking · Robotics & Coding · Other Programmes

PROJECTS AND KITS

Ready-made school projects · DIY and STEM kits · Robotics and coding · Components and spares · Engineering projects

CONTACT

Head office: A-10, Chief Commissioner Colony, opposite Rana Pratap Bagh, Delhi 110007

Phone: 011 2721 1512 · **Mobile and WhatsApp:** +91 97116 99511

Email: engineergurukul@yahoo.co.in

Website: engineergurukul.com · **Instagram:** [@science_projects_club](https://www.instagram.com/science_projects_club)

More free ebooks for computer courses, robotics and science projects: engineergurukul.com/resources/ebooks/

Free to read, print and share. Not for sale.