

UNIT 5

Input and Output

(3 Hours) — C Programming (CACS151), BCA I/II

5.1 Input/Output Operation

Every useful program needs to communicate with the outside world — it must accept data from the user (input) and display results back to the user (output). In C, input and output are not built into the core language itself; instead, they are provided through a collection of standard library functions declared in the header file `stdio.h` (Standard Input/Output).

C treats input and output as a continuous flow of bytes called a stream. Data flows from an input device (such as a keyboard) into the program through the `stdin` (standard input) stream, and flows from the program to an output device (such as a monitor) through the `stdout` (standard output) stream.

Standard Input/Output Streams



Fig 5.1 — Standard input and output streams

- Input Operation: Reading data supplied by the user or another source into program variables.
- Output Operation: Displaying results, messages, or variable values from the program onto the screen.
- All I/O functions in C are broadly divided into two categories: Formatted I/O and Unformatted I/O.

Classification of I/O Functions in C

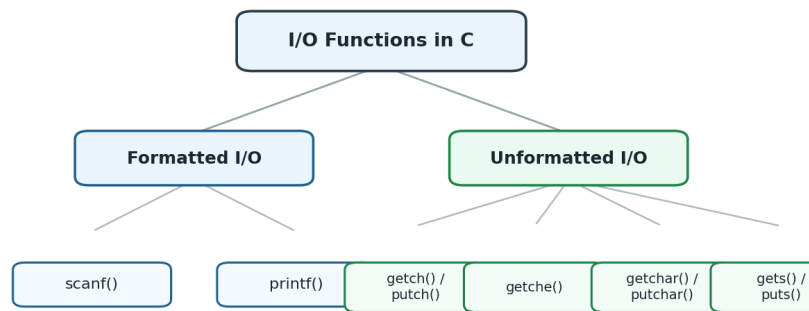


Fig 5.2 — Classification of I/O functions in C

Note: A stream is simply an abstraction — a sequential flow of characters between the program and a device — that allows the same functions to work for the keyboard, screen, or even files.

5.2 Formatted I/O — scanf() and printf()

Formatted I/O functions allow data to be read or displayed in a specific format, using format specifiers that indicate the type of data being handled (e.g. %d for integers, %f for floats, %c for characters). This makes them flexible enough to handle multiple data types and mixed input/output in a single statement.

printf() — Formatted Output

The printf() function displays output on the screen according to a specified format string. It is declared in stdio.h and its general syntax is:

Syntax

```
printf("format string", argument1, argument2, ...);
```

Specifier	Data Type
%d / %i	int
%f	float
%lf	double
%c	char
%s	string

Specifier	Data Type
%u	unsigned int
%x / %X	hexadecimal integer
%o	octal integer
%p	pointer address

Example 5.1 — Using printf()

```
printf_demo.c
#include <stdio.h>
int main()
{
    int age = 20;
    float marks = 88.5;
    char grade = 'A';
    printf("Age: %d\n", age);
    printf("Marks: %.1f\n", marks);
    printf("Grade: %c\n", grade);
    return 0;
}
```

Output

```
Age: 20
Marks: 88.5
Grade: A
```

scanf() — Formatted Input

The scanf() function reads formatted data typed by the user at the keyboard and stores it into program variables. Because scanf() needs to modify the variable's actual memory location, each variable argument (except arrays and strings) must be preceded by the address-of operator (&).

Syntax

```
scanf("format string", &variable1, &variable2, ...);
```

Example 5.2 — Using scanf()

```
scanf_demo.c
#include <stdio.h>
int main()
{
    int roll;
    float percentage;
    printf("Enter roll number and percentage: ");
    scanf("%d %f", &roll, &percentage);
    printf("Roll No: %d, Percentage: %.2f", roll, percentage);
    return 0;
}
```

Explanation

- `&roll` and `&percentage` pass the memory addresses of the variables so `scanf()` can store the entered values directly in them.
- Multiple values can be read in one call by separating format specifiers with spaces, and entering values separated by spaces or newlines at runtime.

Note: A common beginner mistake is forgetting the `&` before a variable name in `scanf()` (except for string/array variables, which already act as addresses) — this causes undefined behaviour or a program crash.

5.3 Unformatted I/O

Unformatted I/O functions handle a single character or a string at a time, without requiring a format specifier. They are simpler and faster than formatted I/O but offer less flexibility, since they cannot handle mixed data types in one call.

getch() and putch()

`getch()` and `putch()` are character I/O functions declared in `conio.h` (available mainly on Turbo C / DOS-based compilers).

- `getch()`: Reads a single character from the keyboard without echoing (displaying) it on the screen, and does not wait for the Enter key.
- `putch()`: Displays a single character on the screen.

Example 5.3 — Using `getch()` and `putch()`

getch_demo.c

```
#include <stdio.h>
#include <conio.h>
int main()
{
    char ch;
    printf("Press any key: ");
    ch = getch();          /* key not shown on screen */
    printf("\nYou pressed: ");
    putchar(ch);           /* now display it manually */
    return 0;
}
```

getche()

`getche()` is similar to `getch()`, but it echoes (displays) the character on the screen as soon as it is typed. It also does not require the Enter key to be pressed.

Example 5.4 — Using `getche()`

getche_demo.c

```
#include <stdio.h>
#include <conio.h>
int main()
{
    char ch;
    printf("Enter a character: ");
    ch = getche();          /* character IS echoed on screen */
    printf("\nYou entered: %c", ch);
    return 0;
}
```

getchar() and putchar()

`getchar()` and `putchar()` are standard, portable character I/O functions declared in `stdio.h` (unlike `getch/getche/putch`, which are compiler-specific extensions found in `conio.h`).

- `getchar()`: Reads a single character from the keyboard. It waits until the Enter key is pressed, and the character is echoed automatically.
- `putchar()`: Displays a single character on the screen.

Example 5.5 — Using `getchar()` and `putchar()`

```

getchar_demo.c
#include <stdio.h>
int main()
{
    char ch;
    printf("Enter a character: ");
    ch = getchar();
    printf("You entered: ");
    putchar(ch);
    return 0;
}

```

gets() and puts()

gets() and puts() are used to read and display an entire line of text (string) rather than a single character.

- gets(): Reads a line of text, including embedded spaces, from the keyboard until the Enter key is pressed, and stores it in a character array.
- puts(): Displays a string on the screen followed automatically by a newline.

Example 5.6 — Using gets() and puts()

```

gets_demo.c
#include <stdio.h>
int main()
{
    char name[50];
    printf("Enter your full name: ");
    gets(name);
    printf("Hello, ");
    puts(name);
    return 0;
}

```

Note: gets() is considered unsafe in modern C because it does not check array bounds and can cause a buffer overflow if the input exceeds the array's size. It has been removed from the C11 standard library and replaced with the safer fgets() function.

5.4 Formatted vs. Unformatted I/O — Comparison

Basis	Formatted I/O	Unformatted I/O
Format specifier	Required (%d, %f, %c, %s, etc.)	Not required

Basis	Formatted I/O	Unformatted I/O
Data handled	Any data type, mixed in one call	Usually a single character or string
Functions	scanf(), printf()	getch(), getche(), putch(), getchar(), putchar(), gets(), puts()
Header file	stdio.h	conio.h (getch/getche/putch) or stdio.h (getchar/putchar/gets/puts)
Speed	Comparatively slower (parses format string)	Faster (direct character/string handling)
Typical use	Reading/printing structured, mixed data	Simple character or line-based input/output

Key Takeaways

- ✓ C handles input and output through library functions declared in stdio.h and conio.h, not through built-in language keywords.
- ✓ I/O in C is treated as a stream of characters flowing via stdin (input) and stdout (output).
- ✓ Formatted I/O (scanf, printf) uses format specifiers and can handle multiple data types in one call.
- ✓ scanf() requires the address-of operator (&) for ordinary variables so it can store values directly in memory.
- ✓ Unformatted I/O functions (getch, getche, putch, getchar, putchar, gets, puts) work with a single character or string, without format specifiers.
- ✓ getch() does not echo the typed character, while getche() and getchar() do.
- ✓ gets() is unsafe due to lack of bounds checking and has been deprecated in favour of fgets() in modern C.

— End of Unit 5 —