

UNIT 6

Control Structure

(6 Hours) — C Programming (CACSI51), Year I / Sem II

6.1 Introduction

By default, statements in a C program execute sequentially — one after another, in the exact order they are written. A control structure is a programming construct that changes this default sequential flow, allowing a program to make decisions, repeat a block of statements, or jump to another point in the code.

Control structures are the building blocks that give a program its logic and decision-making ability. Without them, every program would simply be a straight-line list of instructions with no ability to react to different conditions or repeat work.

Note: Control structures are what allow a program to 'think' — to choose between alternatives and to repeat actions efficiently.

6.2 Types of Control Structure

C provides three broad categories of control structures: Branching (selection), Looping (iteration), and Jumping. Each category contains specific statements suited to different situations.

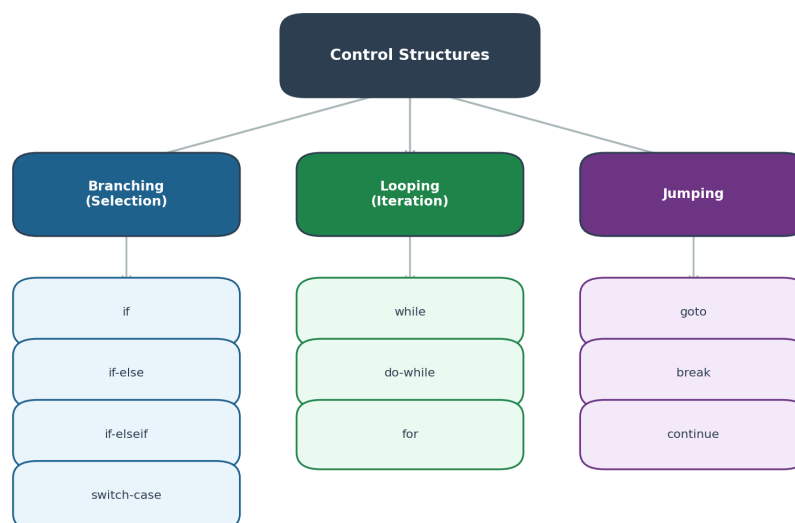


Fig 6.1 — Categories of control structures in C

6.2.1 Branching (Selection) Statements

Branching statements allow a program to choose between two or more paths of execution based on the evaluation of a condition (an expression that evaluates to true or false).

(a) if statement

The simplest branching statement. The block of code inside the if executes only when the condition evaluates to true (non-zero); otherwise it is skipped entirely.

Syntax

```
if (condition) {  
    // statements executed if condition is true  
}
```

Example

```
int age = 20;  
if (age >= 18) {  
    printf("Eligible to vote");  
}
```

Since age (20) is greater than or equal to 18, the message is printed.

(b) if-else statement

Adds an alternative path: when the condition is false, the statements inside the else block execute instead.

Syntax

```
if (condition) {  
    // executes if condition is true  
} else {  
    // executes if condition is false  
}
```

(c) if-elseif ladder

Used when there are multiple mutually exclusive conditions to test in sequence. C evaluates each condition top-to-bottom and executes the block for the first one that is true; if none match, the final else (if present) executes as the default case.

Syntax

```
if (condition1) {  
    // block 1  
} else if (condition2) {  
    // block 2  
} else {  
    // default block  
}
```

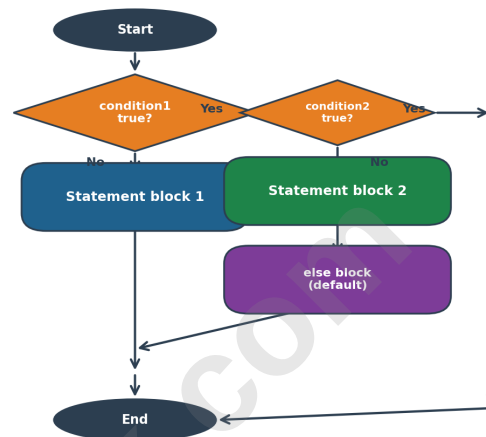


Fig 6.2 — Flowchart of an if / else-if / else ladder

Example — Grading System

```
int marks = 72;  
if (marks >= 90) printf("Grade A+");  
else if (marks >= 75) printf("Grade A");  
else if (marks >= 60) printf("Grade B");  
else printf("Grade C");
```

Output

Grade B (since 72 falls in the 60–74 range)

(d) switch-case statement

The switch statement provides a cleaner alternative to a long if-elseif ladder when a single variable is compared against several constant values. Each matching case label transfers control

to that block; a break statement is needed to prevent execution from 'falling through' into the next case. A default case handles any value that matches no listed case.

Syntax

```
switch (expression) {  
    case value1:  
        // statements  
        break;  
    case value2:  
        // statements  
        break;  
    default:  
        // statements if no case matches  
}
```

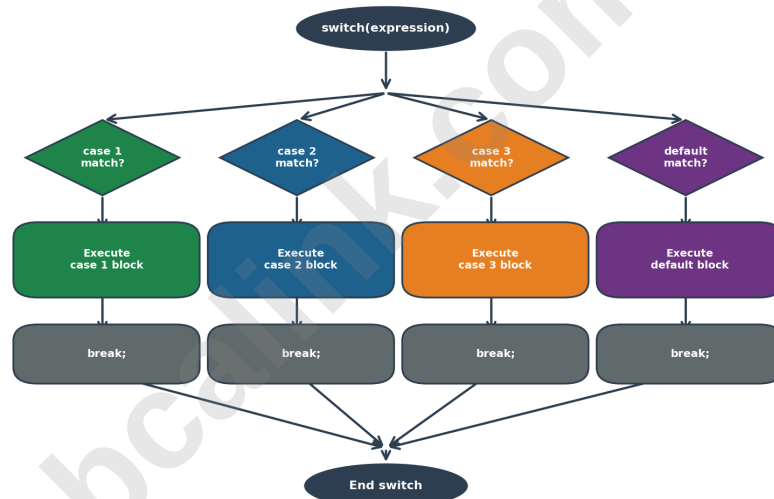


Fig 6.3 — Flowchart of a switch-case statement

Example — Day of the Week

```
int day = 3;  
switch (day) {  
    case 1: printf("Monday"); break;  
    case 2: printf("Tuesday"); break;  
    case 3: printf("Wednesday"); break;  
    default: printf("Invalid day");  
}
```

Output

Wednesday

Note: Omitting 'break;' causes execution to fall through into the following case(s) — a common source of bugs.

Statement	Best Used For
if	A single condition with one optional action
if-else	Two mutually exclusive outcomes
if-elseif	Several ranges or ordered conditions
switch-case	One variable matched against many fixed constant values

6.2.2 Looping (Iteration) Statements

Looping statements repeat a block of statements multiple times until a specified condition becomes false. C provides three looping constructs: while, do-while, and for.

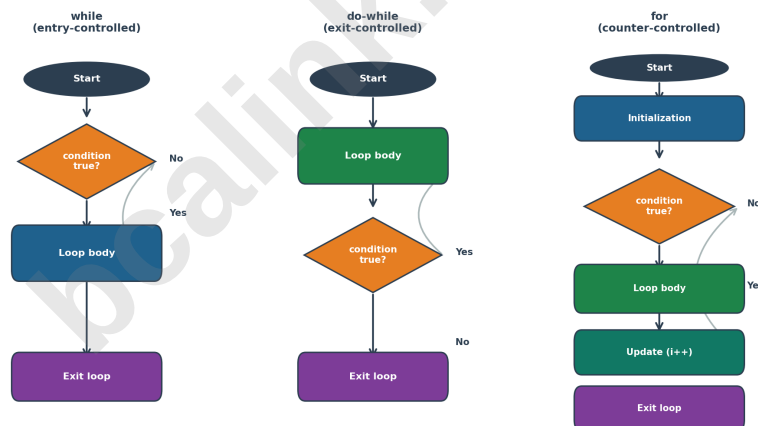


Fig 6.4 — while, do-while, and for loop control flow

(a) while loop

An entry-controlled loop — the condition is tested before each iteration. If the condition is false the very first time, the loop body never executes.

Syntax & Example

```
int i = 1;
while (i <= 5) {
    printf("%d ", i);
    i++;
}
```

Output

1 2 3 4 5

(b) do-while loop

An exit-controlled loop — the loop body executes first, and the condition is tested afterward. This guarantees the body runs at least once, regardless of the condition.

Syntax & Example

```
int i = 1;
do {
    printf("%d ", i);
    i++;
} while (i <= 5);
```

Output

1 2 3 4 5 (identical here, but the body would still run once even if i started at 6)

(c) for loop

A counter-controlled loop that combines initialization, condition-testing, and updating into a single compact statement. It is the most commonly used loop when the number of iterations is known in advance.

Syntax & Example

```
for (int i = 1; i <= 5; i++) {
    printf("%d ", i);
}
```

Output

1 2 3 4 5

Loop	Control Type	Minimum Executions	Typical Use
while	Entry-controlled	0	Unknown iteration count, condition checked first
do-while	Exit-controlled	1	Body must run at least once (e.g. menus)
for	Counter-controlled	0	Known/fixed number of iterations

6.2.3 Jumping Statements

Jumping statements alter the normal flow of control by transferring execution to another part of the program, either unconditionally or from within a loop/switch.

(a) goto statement

Transfers control unconditionally to a labeled statement anywhere within the same function. Because it can make code difficult to trace and debug, its use is generally discouraged in modern structured programming, except in limited cases like breaking out of deeply nested loops.

Example

```
int i = 1;
start:
    if (i > 5) goto end;
    printf("%d ", i);
    i++;
    goto start;
end:
```

(b) break statement

Immediately terminates the nearest enclosing loop (while, do-while, for) or switch statement, transferring control to the statement right after it.

Example

```
for (int i = 1; i <= 10; i++) {
    if (i == 5) break;
    printf("%d ", i);
}
```

Output

1 2 3 4 (loop stops as soon as i equals 5)

(c) continue statement

Skips the remaining statements in the current iteration and jumps directly to the next iteration's condition check (or update step, in a for loop), without exiting the loop.

Example

```
for (int i = 1; i <= 5; i++) {  
    if (i == 3) continue;  
    printf("%d ", i);  
}
```

Output

1 2 4 5 (3 is skipped, but the loop continues)

Statement	Effect
goto	Unconditional jump to a labeled statement
break	Exits the nearest enclosing loop or switch entirely
continue	Skips to the next iteration of the nearest enclosing loop

6.3 Nested Control Structure

A nested control structure occurs when one control structure is placed entirely inside another — for example, an if statement inside a loop, a loop inside another loop, or a switch inside an if. Nesting allows a program to express more complex logic by combining simple structures.

When loops are nested, the inner loop completes all of its iterations for each single iteration of the outer loop. Nested structures are common in tasks such as printing patterns, processing two-dimensional data (matrices), and searching within grouped data.

Example — Nested for loop (multiplication table)

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

Output

```
1 2 3  
2 4 6  
3 6 9
```

The outer loop (i) runs 3 times; for each i, the inner loop (j) runs fully from 1 to 3.

Example — Nested if inside a loop

```
for (int n = 1; n <= 5; n++) {  
    if (n % 2 == 0) {  
        printf("%d is even\n", n);  
    } else {  
        printf("%d is odd\n", n);  
    }  
}
```

Note: Excessive nesting (typically beyond 2–3 levels) reduces readability; consider breaking deeply nested logic into separate functions.

Key Takeaways

- ✓ Control structures alter the default sequential flow of a program: Branching, Looping, and Jumping.
- ✓ Branching (if, if-else, if-elseif, switch) lets a program choose between alternative paths.
- ✓ switch-case needs 'break' after each case to prevent unintended fall-through.
- ✓ while is entry-controlled; do-while is exit-controlled and always runs at least once; for is counter-controlled.
- ✓ goto performs an unconditional jump; break exits a loop/switch; continue skips to the next iteration.
- ✓ Nested control structures place one structure inside another to express more complex logic.
- ✓ In nested loops, the inner loop completes all its iterations for every single iteration of the outer loop.

— *End of Unit 6* —