C Programming: Loops

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

January 16, 2026

Topics Covered

1. Introduction to Loops
2. while Loop
3. do-while Loop
4. for Loop
5. for Loop Variations
6. break Statement
7. continue Statement
8. Nested Loops
9. Common Loop Patterns
10. Summary

What are Loops?

- Repeatedly execute a block of code
- Continue until a condition becomes false
- Avoid code repetition
- Essential for iterative tasks

Types of Loops in C:

1. `while` loop - condition checked before execution
2. `do-while` loop - condition checked after execution
3. `for` loop - compact loop with initialization

Loop Control:

- `break` - exit loop immediately
- `continue` - skip to next iteration

while Loop - Syntax

Syntax:

```c
while (condition) {
  // code to execute
  // update condition variable
}
```

Flow:

- Check condition first
- If true, execute block
- Repeat until condition is false
- May never execute if condition is false initially

Program 1: Basic while Loop

```c
#include <stdio.h>
int main() {
  int i = 1;
  printf("Counting 1 to 5:\n");
  while (i <= 5) {
    printf("%d ", i);
    i++;
  }
  printf("\nDone!\n");
  return 0;
}
```

Output:

```
Counting 1 to 5:
1 2 3 4 5
Done!
```

Explanation:

- i starts at 1
- Loop runs while i ¡= 5
- i incremented each iteration
- Exits when i becomes 6

Program 2: while Loop - Sum of Numbers

```c
#include <stdio.h>
int main() {
  int n = 5, i = 1, sum = 0;
  printf("Sum of 1 to %d:\n", n);
  while (i <= n) {
    sum += i;
    printf("Adding %d, sum=%d\n",
           i, sum);
    i++;
  }
  printf("\nTotal: %d\n", sum);
  return 0;
}
```

Output:

```
Sum of 1 to 5:
Adding 1, sum=1
Adding 2, sum=3
Adding 3, sum=6
Adding 4, sum=10
Adding 5, sum=15

Total: 15
```

Note:

- Accumulates sum
- Shows each step

Program 3: while Loop - Factorial

```c
#include <stdio.h>
int main() {
  int n = 5, i = 1;
  int factorial = 1;
  printf("Factorial of %d:\n", n);
  while (i <= n) {
    factorial *= i;
    printf("%d! = %d\n", i, factorial);
    i++;
  }
  return 0;
}
```

Output:

```
Factorial of 5:
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
```

Explanation:

- Multiplies 1*2*3*4*5
- Shows each step
- $5! = 120$

Program 4: while Loop - Reverse Print

```c
#include <stdio.h>
int main() {
  int i = 10;
  printf("Countdown:\n");
  while (i > 0) {
    printf("%d ", i);
    i--;
  }
  printf("\nBlastoff!\n");
  return 0;
}
```

Output:

```
Countdown:
10 9 8 7 6 5 4 3 2 1
Blastoff!
```

Note:

- Counts down from 10
- Loop decrements i
- Stops when i becomes 0

do-while Loop - Syntax

Syntax:

```
do {
  // code to execute
  // update condition variable
} while (condition);
```

Flow:

- Execute block first
- Then check condition
- Repeat if condition is true
- Executes at least once (key difference from while)

Key Point: Note the semicolon after while(condition)

Program 5: Basic do-while Loop

```c
#include <stdio.h>
int main() {
  int i = 1;
  printf("Counting with do-while:\n");
  do {
    printf("%d ", i);
    i++;
  } while (i <= 5);
  printf("\nDone!\n");
  return 0;
}
```

Output:

```
Counting with do-while:
1 2 3 4 5
Done!
```

Explanation:

- Executes body first
- Then checks condition
- Same result as while here

Program 6: do-while vs while - Key Difference

```c
#include <stdio.h>
int main() {
  int i = 10;
  printf("while loop:\n");
  while (i < 5) {
    printf("This won’t print\n");
  }
  printf("while done\n\n");
  i = 10;
  printf("do-while loop:\n");
  do {
    printf("This prints once!\n");
  } while (i < 5);
  printf("do-while done\n");
  return 0;
}
```

Output:

```
while loop:
while done

do-while loop:
This prints once!
do-while done
```

Key Difference:

- while: checks first, may not execute
- do-while: executes once minimum

Program 7: do-while - Menu System

```c
#include <stdio.h>
int main() {
  int choice;
  int count = 0;
  do {
    printf("\nMenu:\n");
    printf("1. Option 1\n");
    printf("2. Option 2\n");
    printf("3. Exit\n");
    choice = (count == 0) ? 1 :
             (count == 1) ? 2 : 3;
    printf("Choice: %d\n", choice);
    count++;
  } while (choice != 3);
  printf("Exiting...\n");
  return 0;
}
```

Output:

```
Menu:
1. Option 1
2. Option 2
3. Exit
Choice: 1

Menu:
1. Option 1
2. Option 2
3. Exit
Choice: 2

Menu:
1. Option 1
2. Option 2
3. Exit
Choice: 3
Exiting...
```

for Loop - Syntax

Syntax:

```c
for (initialization; condition; update) {
  // code to execute
}
```

Flow:

1. Execute initialization once
2. Check condition
3. If true, execute body
4. Execute update
5. Go to step 2

Equivalent while loop:

```c
initialization;
while (condition) {
  // code to execute
  update;
}
```

Program 8: Basic for Loop

```c
#include <stdio.h>
int main() {
  int i;
  printf("for loop 1 to 5:\n");
  for (i = 1; i <= 5; i++) {
    printf("%d ", i);
  }
  printf("\n\nfor loop 10 to 1:\n");
  for (i = 10; i >= 1; i--) {
    printf("%d ", i);
  }
  printf("\n");
  return 0;
}
```

Output:

```
for loop 1 to 5:
1 2 3 4 5

for loop 10 to 1:
10 9 8 7 6 5 4 3 2 1
```

Note:

- Compact syntax
- All loop control in one line
- Most common loop type

Program 9: for Loop - Even Numbers

```c
#include <stdio.h>
int main() {
  int i;
  printf("Even numbers 1-20:\n");
  for (i = 2; i <= 20; i += 2) {
    printf("%d ", i);
  }
  printf("\n\nOdd numbers 1-20:\n");
  for (i = 1; i <= 20; i += 2) {
    printf("%d ", i);
  }
  printf("\n");
  return 0;
}
```

Output:

```
Even numbers 1-20:
2 4 6 8 10 12 14 16 18 20

Odd numbers 1-20:
1 3 5 7 9 11 13 15 17 19
```

Note:

- Custom increment: $i += 2$
- Different start values

Program 10: for Loop - Multiplication Table

```c
#include <stdio.h>
int main() {
  int n = 7, i;
  printf("Multiplication table of %d:\n",
         n);
  for (i = 1; i <= 10; i++) {
    printf("%d x %d = %d\n",
           n, i, n * i);
  }
  return 0;
}
```

Output:

```
Multiplication table of 7:
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

for Loop Variations

Standard form:

```c
for (i = 0; i < 10; i++) { }
```

Variations:

- Multiple initializations: `for (i=0, j=10; ...)`
- Multiple updates: `for (...; i++, j--)`
- Empty parts: `for (;;)` - infinite loop
- No initialization: `for (; i<10; i++)`
- No update: `for (i=0; i<10;)`
- Declare in loop: `for (int i=0; i<10; i++)`

Program 11: Multiple Variables in for

```c
#include <stdio.h>
int main() {
  int i, j;
  printf("Two counters:\n");
  for (i=1, j=10; i<=5; i++, j--) {
    printf("i=%d, j=%d, sum=%d\n",
           i, j, i+j);
  }
  return 0;
}
```

Output:

```
Two counters:
i=1, j=10, sum=11
i=2, j=9, sum=11
i=3, j=8, sum=11
i=4, j=7, sum=11
i=5, j=6, sum=11
```

Note:

- Two variables: i, j
- i increments, j decrements
- Sum stays constant

Program 12: Infinite Loop with Break

```c
#include <stdio.h>
int main() {
  int count = 0;
  printf("Infinite loop demo:\n");
  for (;;) {
    printf("Count: %d\n", count);
    count++;
    if (count >= 5) {
      printf("Breaking out!\n");
      break;
    }
  }
  printf("Loop exited\n");
  return 0;
}
```

Output:

```
Infinite loop demo:
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
Breaking out!
Loop exited
```

Note:

- `for (;;)` is infinite
- break exits the loop

Program 13: Variable Declared in for Loop

```c
#include <stdio.h>
int main() {
  printf("C99 style for loop:\n");
  for (int i = 1; i <= 5; i++) {
    printf("%d ", i);
  }
  printf("\n\nAnother loop:\n");
  for (int i = 10; i > 5; i--) {
    printf("%d ", i);
  }
  printf("\n");
  return 0;
}
```

Output:

```
C99 style for loop:
1 2 3 4 5

Another loop:
10 9 8 7 6
```

Note:

- Variable i declared in loop
- Scoped to loop only
- C99 feature
- Can reuse name i

break Statement

Purpose:

- Exit loop immediately
- Skip remaining iterations
- Continue with code after loop
- Works with while, do-while, for

Syntax:

```c
while (condition) {
  if (some_condition) {
    break;  // exit loop
  }
}
```

Program 14: break - Find First Multiple

```c
#include <stdio.h>
int main() {
  int i;
  printf("Find first number > 50\n");
  printf("divisible by 7:\n\n");
  for (i = 51; i <= 100; i++) {
    if (i % 7 == 0) {
      printf("Found: %d\n", i);
      break;
    }
  }
  printf("Loop ended at i=%d\n", i);
  return 0;
}
```

Output:

```
Find first number > 50
divisible by 7:

Found: 56
Loop ended at i=56
```

Explanation:

- Loop starts at 51
- First multiple of 7 is 56
- break exits immediately
- i retains value 56

Program 15: break - Search in Loop

```c
#include <stdio.h>
int main() {
  int target = 7, i;
  int found = 0;
  printf("Searching for %d:\n",
         target);
  for (i = 1; i <= 10; i++) {
    printf("Checking %d\n", i);
    if (i == target) {
      found = 1;
      break;
    }
  }
  if (found) {
    printf("\nFound at position %d\n",
           i);
  }
  return 0;
}
```

Output:

```
Searching for 7:
Checking 1
Checking 2
Checking 3
Checking 4
Checking 5
Checking 6
Checking 7

Found at position 7
```

Note:

- Stops when found
- Saves iterations

continue Statement

Purpose:

- Skip rest of current iteration
- Jump to next iteration
- Loop continues running
- Works with while, do-while, for

Syntax:

```c
for (i = 0; i < 10; i++) {
  if (some_condition) {
    continue;  // skip to next iteration
  }
  // this code skipped if continue executed
}
```

Program 16: continue - Skip Odd Numbers

```c
#include <stdio.h>
int main() {
  int i;
  printf("Even numbers 1-10:\n");
  for (i = 1; i <= 10; i++) {
    if (i % 2 != 0) {
      continue;
    }
    printf("%d ", i);
  }
  printf("\n");
  return 0;
}
```

Output:

```
Even numbers 1-10:
2 4 6 8 10
```

Explanation:

- If i is odd, continue
- printf skipped for odd
- Only even numbers print

Program 17: continue - Skip Multiples

```c
#include <stdio.h>
int main() {
  int i;
  printf("Numbers 1-20\n");
  printf("(skip multiples of 3):\n");
  for (i = 1; i <= 20; i++) {
    if (i % 3 == 0) {
      continue;
    }
    printf("%d ", i);
  }
  printf("\n");
  return 0;
}
```

Output:

```
Numbers 1-20
(skip multiples of 3):
1 2 4 5 7 8 10 11 13 14 16 17 19 20
```

Note:

- Skips 3, 6, 9, 12, 15, 18
- continue jumps to i++
- Loop continues

Program 18: break vs continue

```c
#include <stdio.h>
int main() {
  int i;
  printf("With continue:\n");
  for (i = 1; i <= 10; i++) {
    if (i == 5) continue;
    printf("%d ", i);
  }
  printf("\n\nWith break:\n");
  for (i = 1; i <= 10; i++) {
    if (i == 5) break;
    printf("%d ", i);
  }
  printf("\n");
  return 0;
}
```

Output:

```
With continue:
1 2 3 4 6 7 8 9 10

With break:
1 2 3 4
```

Difference:

- continue: skips 5, continues
- break: stops at 5, exits loop

Nested Loops

Definition:

- Loop inside another loop
- Inner loop executes completely for each outer iteration
- Can nest any loop type

Execution:

- Outer loop: 1 iteration
- Inner loop: all iterations
- Outer loop: next iteration
- Inner loop: all iterations again

Common Uses:

- 2D arrays, matrices
- Pattern printing
- Nested data structures

Program 19: Nested Loop - Rectangle Pattern

```c
#include <stdio.h>
int main() {
  int i, j;
  int rows = 4, cols = 6;
  printf("Rectangle pattern:\n");
  for (i = 1; i <= rows; i++) {
    for (j = 1; j <= cols; j++) {
      printf("* ");
    }
    printf("\n");
  }
  return 0;
}
```

Output:

```
Rectangle pattern:
* * * * * *
* * * * * *
* * * * * *
* * * * * *
```

Explanation:

- Outer loop: 4 rows
- Inner loop: 6 cols each row
- Total: 24 stars

Program 20: Nested Loop - Multiplication Table

```c
#include <stdio.h>
int main() {
  int i, j;
  printf("Multiplication table:\n");
  printf("   ");
  for (i = 1; i <= 5; i++) {
    printf("%4d", i);
  }
  printf("\n");
  for (i = 1; i <= 5; i++) {
    printf("%2d:", i);
    for (j = 1; j <= 5; j++) {
      printf("%4d", i * j);
    }
    printf("\n");
  }
  return 0;
}
```

Output:

```
Multiplication table:
      1   2   3   4   5
 1:   1   2   3   4   5
 2:   2   4   6   8  10
 3:   3   6   9  12  15
 4:   4   8  12  16  20
 5:   5  10  15  20  25
```

Note:

- Outer: rows (i)
- Inner: columns (j)

Program 21: Nested Loop - Triangle Pattern

```c
#include <stdio.h>
int main() {
  int i, j;
  printf("Triangle pattern:\n");
  for (i = 1; i <= 5; i++) {
    for (j = 1; j <= i; j++) {
      printf("* ");
    }
    printf("\n");
  }
  return 0;
}
```

Output:

```
Triangle pattern:
*
* *
* * *
* * * *
* * * * *
```

Explanation:

- Row 1: 1 star
- Row 2: 2 stars
- Row i: i stars
- Inner loop limit is i

Program 22: Nested Loop - Number Pattern

```c
#include <stdio.h>
int main() {
  int i, j;
  printf("Number pattern:\n");
  for (i = 1; i <= 5; i++) {
    for (j = 1; j <= i; j++) {
      printf("%d ", j);
    }
    printf("\n");
  }
  return 0;
}
```

Output:

```
Number pattern:
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Note:

- Each row prints 1 to i
- j is the printed value

Program 23: Nested with break

```c
#include <stdio.h>
int main() {
  int i, j;
  printf("Break in nested loop:\n");
  for (i = 1; i <= 3; i++) {
    printf("Outer i=%d: ", i);
    for (j = 1; j <= 5; j++) {
      if (j == 3) {
        break;
      }
      printf("%d ", j);
    }
    printf("\n");
  }
  return 0;
}
```

Output:

```
Break in nested loop:
Outer i=1: 1 2
Outer i=2: 1 2
Outer i=3: 1 2
```

Important:

- break exits inner loop only
- Outer loop continues
- Each row prints 1 2

Program 24: Sum and Average

```c
#include <stdio.h>
int main() {
  int n = 5, i;
  int sum = 0;
  float avg;
  printf("Numbers: ");
  for (i = 1; i <= n; i++) {
    printf("%d ", i);
    sum += i;
  }
  avg = (float)sum / n;
  printf("\n\nSum: %d\n", sum);
  printf("Average: %.2f\n", avg);
  return 0;
}
```

Output:

```
Numbers: 1 2 3 4 5

Sum: 15
Average: 3.00
```

Pattern:

- Accumulate sum in loop
- Calculate average after
- Type cast for float division

Program 25: Find Maximum

```c
#include <stdio.h>
int main() {
  int i, max;
  int nums[] = {34, 12, 89, 5, 67};
  int size = 5;
  max = nums[0];
  printf("Numbers: ");
  for (i = 0; i < size; i++) {
    printf("%d ", nums[i]);
    if (nums[i] > max) {
      max = nums[i];
    }
  }
  printf("\n\nMaximum: %d\n", max);
  return 0;
}
```

Output:

```
Numbers: 34 12 89 5 67

Maximum: 89
```

Pattern:

- Initialize max to first element
- Compare each element
- Update if larger found

Loop Types - Summary

Loop	When to Use	Min Executions
while	Unknown iterations Condition-based	0
do-while	At least once Menu systems	1
for	Known iterations Counter-based	0

Loop Control:

- **break**: Exit loop immediately
- **continue**: Skip to next iteration

Best Practices

1. **Choose the right loop** for the task
2. **for loop** for known iterations
3. **while loop** for unknown iterations
4. **do-while** when at least one execution needed
5. **Avoid infinite loops** - ensure condition becomes false
6. **Use meaningful variable names** (not just i, j, k)
7. **Indent nested loops** properly
8. **Use break/continue** judiciously
9. **Avoid modifying** loop counter inside loop body
10. **Test edge cases** (empty, single item)

Common Mistakes

1. **Off-by-one errors**: `i < n` vs `i <= n`
2. **Infinite loops**: Forgetting to update counter
3. **Wrong initialization**: Starting at wrong value
4. **Semicolon after for/while**: `for(;;);` { }
5. **Modifying counter**: Changing i inside loop
6. **break/continue scope**: Only affects nearest loop
7. **do-while semicolon**: Forgetting ; after while()
8. **Nested loop confusion**: Which loop does break exit?

Practice Exercises

Try these programs:

1. Print Fibonacci series up to n terms
2. Check if a number is prime
3. Find GCD of two numbers
4. Print reverse of a number
5. Count digits in a number
6. Print all prime numbers between 1 and 100
7. Print Floyd's triangle
8. Calculate power without pow() function
9. Print diamond pattern with stars
10. Find sum of digits of a number