# Iteration / for loops

CSC100 / Introduction to programming in C/C++ - Spring 2025

Marcus Birkenkrahe

April 12, 2025

## Contents



## 1 README

- This script introduces C looping structures.
- This section is based on chapter 4 in Davenport/Vine (2015) and chapter 6 in King (2008).

- Practice workbooks, input files and PDF solution files in GitHub

## 2 Loops

- A **loop** is a statement whose job is to repeatedly execute over some other statement (the **loop body**).
- Every loop has a **controlling expression**.
- Each time the loop body is executed (an **iteration** of the loop), the controlling expression is evaluated.
- If the expression is **TRUE** (has a value that is non-zero), the loop continues to execute.
- C provides three iteration statements: `while`, `do`, and `for`

## 3 The `for` statement

- The `for` statement has the general form

  ```
  for ( /expr1 ; expr2 ; expr3/ ) /statement/ ;
  ```

- Here, `expr1`, `expr2` and `expr3` are expressions.

## 4 Simple example: counting down

- You recognize the familiar countdown program - except that the `for` loop includes initialization, condition and counting down all in one go.
- Pseudocode:

```
// DECLARE i as integer
// FOR i starting at 5 down to 1 DO
      // PRINT i
// END FOR

int i; // DECLARE i as integer
// FOR i starting at 5 down to 1 DO
for ( i = 5; i > 0; i-- )  {
  printf("%d ", i); // PRINT i
 } // END FOR
```

```
5 4 3 2 1
```

## 5 Practice: counting up

- The pseudocode for the countdown is here: onecompiler.com/c/43dkfh69u - turn it into a program that counts up from 0 to 9.

  First copy the code for the countdown and run it, then ADD the pseudocode to count up below it, then run both. Sample output:

  ```
  5 4 3 2 1
  0 1 2 3 4 5 6 7 8 9
  ```

### Solution:

`https://onecompiler.com/c/43dkg79h7`

1. Pseudocode:

   ```
   // DECLARE i as integer
   // FOR i starting at 0 up to 9 DO
       // PRINT i
   // END FOR
   ```

2. Source code:

   ```
   int i; // DECLARE i as integer
   // FOR i starting at 0 up to 9 DO
   for (i = 0; i <= 9; i++) {
     printf("%d ",i); // PRINT i
   } // END FOR
   ```

   ```
   0 1 2 3 4 5 6 7 8 9
   ```

3. You don't need to define two loop variables if you declare them **inside** the `for` statement

   ```
   /* Countdown from 5 to 1 */

   // FOR i starting at 5 down to 1 DO
   ```

```
for (int i = 5; i > 0; i-- )  {
  printf("%d ", i); // PRINT i
 } // END FOR
puts("");

/* Counting up from 0 to 9 */

// FOR i starting at 0 up to 9 DO
for (int i = 0; i <= 9; i++) {
  printf("%d ",i); // PRINT i
 } // END FOR


5 4 3 2 1
0 1 2 3 4 5 6 7 8 9
```

4. The `for {...}` loop is a protected area, defining so-called **local scope** variables (whose memory is erased when the loop is finished.

## 6 Swapping `for` and `while`

- `for` loops can be replaced by `while` loops and vice versa:

  ```
  expr1;
  while (expr2) {
    statement
    expr3;
    }
  ```

  Becomes:

  ```
  for (expr1; expr2; expr3;) {
    statement
  }
  ```

- Studying the equivalent `while` loop can yield important insights: you remember what happened when we swapped the postfix for a prefix operator in the countdown `while` loop. Rewriting this program as a `for` loop, we get:

```
int i = 3;  /* expr1 */
while ( i > 0 /* expr2 */) {
  printf("while: %d\n", i-- /* expr3 */ );
 }

for (int i = 3; i > 0; i--) { // expr 1; expr2; expr 3;
  printf("for: %d\n",i);
 }
```

```
while: 3
while: 2
while: 1
for: 3
for: 2
for: 1
```

- Notice that we can re-use `i` in the `for` loop.

## 7 Practice : Swapping `while` and `for`

Run and then convert the program below into a while loop using the code block below. The program should count from 3 to 9.

```
// FOR i from up to 9 DO
for ( int i = 3; i < 10 ; ) {
  printf("for: %d\n", i++); // PRINT "for: " + i
} // END FOR
```

```
for: 3
for: 4
for: 5
for: 6
for: 7
for: 8
for: 9
```

Convert this to a `while` loop (start with pseudocode).

### Solution

1. Pseudocode:

```
// SET i = 3
// WHILE i smaller than 10
   // PRINT "while: " + i
// END WHILE
```

2. Source code:

```
int i = 3;// SET i = 3
// WHILE i smaller than 10
while (i < 10) {
  printf("while: %d\n",i++); // PRINT "while: " + i
} // END WHILE
```

```
while: 3
while: 4
while: 5
while: 6
while: 7
while: 8
while: 9
```

## 8 `for` statement patterns

- `for` loops are best when counting up or down

| PATTERN / IDIOM | CODE |
|---|---|
| Counting up from `0` to `n-1` | `for ( i = 0; i < n; i++ )` |
| Counting up from `1` to `n` | `for ( i = 1; i <= n; i++ )` |
| Counting down from `n-1` to `0` | `for ( i = n-1; i >= 0; i-- )` |
| Counting down from `n` to `1` | `for ( i = n; i > 0; i-- )` |

  - Counting up loops rely on `<` and `<=`, while counting down loops rely on `>` and `>=` operators.
  - Note that the controlling expression does **not** use `==` but `=` instead - we're not computing Boolean/truth values (`==`) but we're assigning beginning numerical values (`=`).

– The following is cool (but also dangerous): you can initialize the counting variable inside the first expression:

```
// int i; // SET loop counter i
for ( int i = 3 ; i > 0 ; i--) {
  printf("T minus %d and counting\n", i);
 }

T minus 3 and counting
T minus 2 and counting
T minus 1 and counting
```

– Even better: You can keep using the same loop variable this way:

```
for ( int i = 3 ; i > 0 ; i--) {
  printf("%d ", i);
 }; puts("\nNew loop - same counter variable:");
for ( int i = 3 ; i > 0 ; i--) {
  printf("%d ", i);
 }; puts("\nNew loop - same counter variable:");
for ( int i = 3 ; i > 0 ; i--) {
  printf("%d ", i);
 }

3 2 1
New loop - same counter variable:
3 2 1
New loop - same counter variable:
3 2 1
```

## 9 Omitting expressions

- Some `for` loops may not need all 3 expressions, though the separators `;` must all three be present
- If the **first** expression is omitted, no initialization is performed before the loop is executed:

```
int i = 3;
for ( ; i > 0 ; --i) {
  printf("T minus %d and counting\n", i);
 }
```

```
T minus 3 and counting
T minus 2 and counting
T minus 1 and counting
```

- If the **third** expression is omitted, the loop body is responsible for ensuring that the value of the 2nd expression eventually becomes false so that the loop ends (just like in `while` and `do while`):

```
for (int i = 3 ; i > 0 ; ) {
  printf("T minus %d and counting\n", i--);
 }
```

```
T minus 3 and counting
T minus 2 and counting
T minus 1 and counting
```

- If the **first** and **third** expressions are omitted, the resulting loop is nothing but a `while` statement in disguise:

```
int i = 3;
for ( ; i > 0 ; )
  printf("T minus %d and counting\n", i--);
```

```
T minus 3 and counting
T minus 2 and counting
T minus 1 and counting
```

- The `while` version is clearer and to be preferred:

```
int i = 10;
while ( i > 0 ) {
  printf("T minus %d and counting\n", i--);
 }
```

```
T minus 10 and counting
T minus 9 and counting
T minus 8 and counting
```

```
T minus 7 and counting
T minus 6 and counting
T minus 5 and counting
T minus 4 and counting
T minus 3 and counting
T minus 2 and counting
T minus 1 and counting
```

- If the **second** expression is missing, it defaults to a `TRUE` value so that the `for` loop will cause an infinite loop:

```
int i;
for (  ; ;  ) {
  printf("T minus %d and counting\n", i--);
 }
```

## 10 Practice omitting expressions from `for` loop statements

- I've omitted the third expression in the code block below. When you run the block you will realize that it does not end.
- Fix the error **without** changing the controlling expressions so that you can see the countdown from 10 to 1 as output!

```
for ( int i = 10 ; i > 0 ; )
  printf("%d ",i);
```

### Solution

- You can move the decrementing expression into the `printf` statement:

```
for ( int i = 10 ; i > 0 ; )
  printf("%d ",i--);


10 9 8 7 6 5 4 3 2 1
```

## 11 Example: Printing a table of squares

- The program below can be condensed by converting its `while` loop to a `for` loop:

```
int i, n;

printf("This program prints a table of squares.\n");
printf("Enter number of entries in table: ");
scanf("%d", &n); printf("%d\n", n);

for ( i = 1; i <= n; i++) printf("%10d%10d\n", i, i * i);
```

- Inputfile

```
echo "5" > ../data/square1_input
cat ../data/square1_input
```

- In **??**, all three expressions are controlled by the variable `i` for initialization, testing, and updating. However, **there is no requirement that they be related in any way**: the version **??** of the same program demonstrates this:

```
int i; // testing variable
int n; // upper bound constant
int odd; // incrementing variable
int square; // initialization variable

printf("This program prints a table of squares.\n");
printf("Enter number of entries in table: ");
scanf("%d", &n); printf("%d\n", n);

i   = 1;
odd = 3;
puts("         i    square       odd");
puts("------------------------------");

for ( square = 1; i <= n; odd += 2) {
  printf("%10d%10d%10d\n", i, square, odd);
  ++i;
```

```
  square += odd;
 }
```

- The `for` statement in the code block initializes one variable (`square`), tests another (`i`), and increments a third (`odd`).

  `i` is the number to be squared, `square` is the square of `i`, and `odd` is the odd number that must be added to the current square to get the next square (without having to multiply anything).