Iteration / Loops - while

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

Marcus Birkenkrahe

March 28, 2025

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

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`

The `while` statement

Overview

- The `while` statement has the general form

```
while ( /expression/ ) statement
```

- The `statement` is executed as long as the `expression` is true.

Simple example

- A simple example.

```
while ( i < n ) {  /* controlling expression */
  i = i * 2;       /* loop body */
  }
```

- Parentheses `(...)` around the *expression* are mandatory
- Braces `{ }` are used for multi-line statements
- What does the code in do?
- We can trace what happens:

```
int i = 1, n = 10;
while ( i < n ) {
  i = i * 2;
  printf("%d < %d ?\n", i, n);
 }
```

```
2 < 10 ?
4 < 10 ?
8 < 10 ?
16 < 10 ?
```

- What would the pseudocode look like?

```
While i is smaller than n
  double the value of i
  show i and n
end when i is greater than n
```

- What would a BPMN model look like?

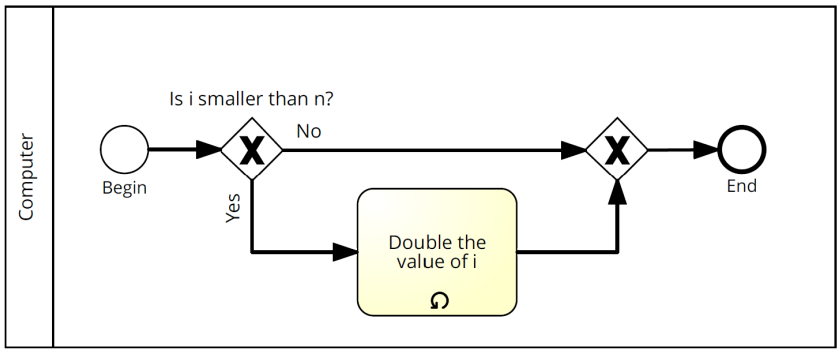

Figure 1: Simple while example

TODO Practice: First loop

- Your turn! Open an editor and enter the starter code:

```
int i = 1, n = ...;

while (i < n) {
  i = i * 2;
  ...
}
```

  1. Set the loop limit `n` outside of the loop to 10.
  2. Insert a `printf` statement in the `while` loop body.
  3. Print the values of `i`, `n` and `i < n` for each iteration.
  4. Sample output:

```
2 < 10 == 1
4 < 10 == 1
8 < 10 == 1
16 < 10 == 0
```

Solution:

```
int i = 1, n = 10;

while (i < n) {
  i = i * 2;
```

```
  printf("%d < %d == %d\n",i,n,i<n);
 }
```

```
2 < 10 == 1
4 < 10 == 1
8 < 10 == 1
16 < 10 == 0
```

Countdown example

- What does the following statement do? What is printed out at the very end?

```
#define N 10
int i = 0;
printf("i = %d\n",i);
while ( i < N ) {   //
  printf("T plus %d and counting\n", i);
  i++;  //
 }
printf("i = %d\n", i); //
```

```
i = 0
T plus 0 and counting
T plus 1 and counting
T plus 2 and counting
T plus 3 and counting
T plus 4 and counting
T plus 5 and counting
T plus 6 and counting
T plus 7 and counting
T plus 8 and counting
T plus 9 and counting
i = 10
```

- Solution (code with comments):

```
#define N 10 // Define loop limit as constant
```

```
int i = 0;  // declaration and definition of loop variable

printf("i = %d\n",i); // print loop variable before loop

while ( i < N ) {   // tests if i is positive
  printf("T plus %d and counting\n", i);  // print i
  i++;  // same as i = i + 1; (executed from the right)
 }
printf("i = %d\n", i); // print loop variable after loop


i = 0
T plus 0 and counting
T plus 1 and counting
T plus 2 and counting
T plus 3 and counting
T plus 4 and counting
T plus 5 and counting
T plus 6 and counting
T plus 7 and counting
T plus 8 and counting
T plus 9 and counting
i = 10
```

- Why are we using `i++` and not `++i` ?[1]
- What would change if we would swap the two statements inside the `while` loop?
- When would the `while` statements be bypassed completely?[2]
- The code could be made more concise (shortened by one line) - can you guess how? Remember what you know about `printf`?

```
#define N 10
```

[1] `i++` is evaluated from the left, while `++i` is evaluated from the right. Both stand for `i = i + 1` , but `i++` assigns the current value of `i` and then adds `1`, while `++i` adds `1` and then assigns the result to `i`. In this case, the result is the same because we don't have any more statements that use `i` but if there were, it would make a difference.

[2] The loop body will not be entered if the expression tests out as false, i.e. if `i` is zero or negative. Try that!

```
int i = 0;
printf("i = %d\n",i);
while ( i < N ) {
  printf("T plus %d and counting\n", i++);
 }
printf("i = %d\n",i);

i = 0
T plus 0 and counting
T plus 1 and counting
T plus 2 and counting
T plus 3 and counting
T plus 4 and counting
T plus 5 and counting
T plus 6 and counting
T plus 7 and counting
T plus 8 and counting
T plus 9 and counting
i = 10
```

- Note that in the concise version, it makes a difference if we use `i--` or `--i`. Try it!

TODO Practice: Countdown

- Your turn! The program below counts down from `i=N` and prints both the counter variable and the end value.

  1. Enter the starter code:

     ```
     #define N 10
     int i = ...;
     printf("i = %d\n", i);
     while ( ... ) {
       printf("T minus %d and counting\n", i);
       ...
         }
     printf("i = %d\n", i);
     ```

  2. Fix the loop variable definition and the condition, and add a compound operator `i--` for counting down from `i=10`. Run the code.

3. Change the operator to `--i` and check if there's a difference.
4. Create a more concise version of the code by pulling the counting statement into the `printf` statement. Change the compound operator from `i--` to `--i`.

Solution

- Completed code:

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


i = 10
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
i = 0
```

- More concise code:

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

```
i = 10
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
i = 0
```

Infinite loops

- If the controlling expression always has a non-zero value, the `while` statement will not terminate.
- For example in a game a loop would have a statement like `while(1)` because this condition is always true - until the player enters 'quit'
- The compiler does not check this. This program has to be stopped manually - in the online editor it runs out of memory after a few thousand lines or so:

```
while (1)
  puts("Endless...\n");
```

- To stop infinite loops from within, you need to provide `break`, `goto` or `return` statements ("controlled jump").

TODO Practice: Infinite loop

- Your turn! Complete a simple practice exercise under "Infinite loops" in the practice file.
- Create a program that runs forever:

```
while(...) {
  puts("Endless...");
}
```

- Run it and see what happens.

TODO Exercise: Printing a table of squares

Problem

- Compute the squares of all integers from `1` to `n`.
- Print `n` and its square as a table of `n` rows
- Sample output for `n=10`.

```
 1          1
 2          4
 3          9
 4         16
 5         25
 6         36
 7         49
 8         64
 9         81
10        100
```

- Challenge: Enter number of rows to print (via command-line). Sample output for `N=10`:

```
Enter number of rows: 10
       1          1
       2          4
       3          9
       4         16
       5         25
       6         36
       7         49
       8         64
       9         81
      10        100
```

Solution

Summing numbers (Home assignment)

Problem

- Input a series of integers via the command line.

- Compute the sum of the integers.
- Sample output:

```
Enter integers (0 to terminate). 8 23 71 5 0
The sum is 107
```

Solution

- Scan numbers one after the other.
- The program should exit when a `0` is scanned.
- To sum, we can use the compound operator `+=`
- Pseudocode:

```
// Purpose: Sum a series of integers
Input: series of integers
Output: sum of all integers

Begin:
   // declare and initialize variables
   // scan first integer

   while integer non-zero
     sum integer
     scan next integer

   print the sum
End
```

- Generate test input file:

```
echo 8 23 71 5 0 > ../data/sum_input
cat ../data/sum_input
```

- Code:

```
// declaration and definition
int n, sum = 0;
// get user input and first number
```

```
puts("Enter integers (0 to terminate): ");
scanf("%d", &n);  printf("%d ", n); // need non-0 number to start

// test if number entered is non-zero
while ( n != 0 ) {
  // sum = sum + n
  sum += n;
  scanf("%d", &n); printf("%d ", n);
 }

printf("\nThe sum is %d\n", sum);
```

- There are two identical calls to `scanf`, because we need a non-zero number to enter the `while` loop in the first place.

Solutions

1. Counting up from 1 to 5:

```
for(int j=1;j<=5; j++)
  printf("%d and counting\n",j);
```

```
1 and counting
2 and counting
3 and counting
4 and counting
5 and counting
```

2. Converting `for` loop into `while` loop:

```
int i = 3;
while(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
```

3. Summing numbers (convert `do while` to `for`):

```
int n, sum = 0;

scanf("%d", &n);

for ( ; n != 0; ) {
  sum += n;
  scanf("%d", &n);
 }

printf("The sum is %d\n", sum);
```

References

- Davenport/Vine (2015) C Programming for the Absolute Beginner (3ed). Cengage Learning.
- Kernighan/Ritchie (1978). The C Programming Language (1st). Prentice Hall.
- King (2008). C Programming - A modern approach (2e). W A Norton.
- Orgmode.org (n.d.). 16 Working with Source Code [website]. URL: orgmode.org