Tracing and Visualizing C Programs Using C Coding Tutor: Learning Activities for CCPROG1

Preliminary Task: there are two non-graded Check-up Quizzes in this module. Please click this link for Check-up Quiz #4 on conditional expressions, and then click this link for Check-up Quiz #5 on while and for loop. The form is only accessible to a DLSU domain; please log-in to your DLSU account in case you cannot access it the first time around. The quiz assumes that you have already covered the topic on loops (also called iterative or repetitive statements) in your CCPROG1 class. The objective of the quiz is to determine possible misconceptions which we intend to clarify and remediate as part of the learning activities. The answers to Quiz #4 are on page 2, and the answers to Quiz #5 are on the last two pages of this document.

MODULE 5: Tracing and Visualizing Loops

It is assumed that you have covered the topic on loops, i.e., **while** loop, **for** loop and **do while** loop in your CCPROG1 class[1]. A loop requires a conditional expression to be evaluated. A conditional expression usually involves the use of relational operators (i.e., <, >, <=, >=, !=, ==) and logical operators (i.e., !, &&, ||). The loop semantics dictate that as long as the condition is true, the body of the loop will be executed repetitively. The loop will terminate when the conditional expression evaluates to false. If the condition will never evaluate to false, then the loop is said to be an infinite or endless loop.

Learning Activity 5.1: Evaluating the value of a conditional expression

Evaluating a valid conditional expression made up of relational and logical operators will result only into either a 0 or a 1. Let's verify this by printing the values of some representative expressions. Please make sure to answer first Check-up Quiz #4 in case you have not done it yet before proceeding with this activity.

Learning Outcome: The student can evaluate the value of a conditional expression.

Instructions:

1. Encode the source code shown in Figure 5.1. Or if you want to omit the encoding step, you may simply click this link to open a browser window with a pre-encoded program.

```c
#include <stdio.h>

int main() {
    int x = 10, y = 5, z = 2;

    // expressions with relational operators
    printf("1. x < y is %d\n", x < y);
    printf("2. x > y is %d\n", x > y);
    printf("3. x != y is %d\n", x != y);
    printf("4. x / y == z is %d\n", x / y == z);
    printf("5. y / z != z %% y is %d\n", y / z != z % y);

    // expressions with relational and logical operators
    printf("6. x / y == 0 && y / z == 1 is %d\n", x / y == 0 && y / z == 1);
    printf("7. z %% 10 > y || x / z != 1 is %d\n", z % 10 > y || x / z != 1);
    printf("8. !( (x > y && y > z) || (y * z == x) ) is %d\n", !( (x > y && y > z) || (y * z == x) ));
    return 0;
}
```

Figure 5.1: sample program with conditional expressions

[1] We will only cover the while loop and for loop in this module…

2. Which expression do you think will evaluate to 0, i.e., false, and which expression will evaluate to 1, i.e., true? Trace and visualize the execution by clicking on the "Next" button until the end of the program. Pay attention to what happens as you progress through each step.

   A screenshot of the program's complete output is shown in Figure 5.2. Compare and verify your answers to Check-up Quiz #4 with the program output.

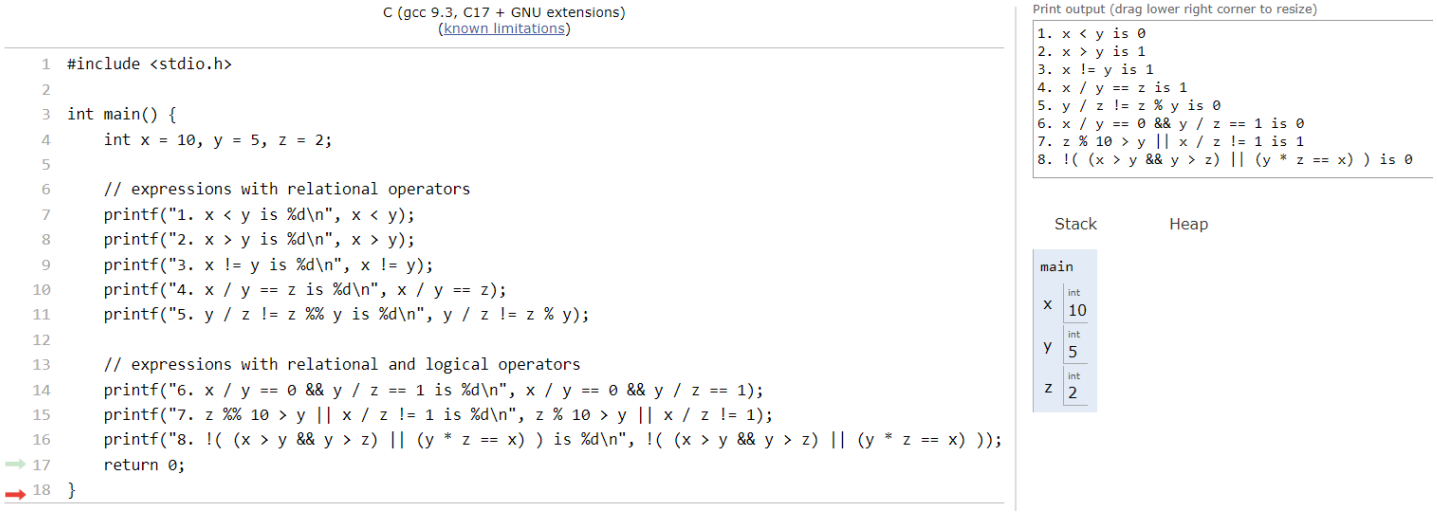

Figure 5.2: Screenshot of the complete trace of the program

Learning Activity 5.2: Tracing a while loop

One of the applications of loops is in generating a series of numbers. For this activity let's trace the logic of a simple program that requires printing of integers from 1 to 5.

Learning Outcome: The student can trace the logic of a **while** loop.

Instructions:

1. Encode the source code shown in Figure 5.3. Or if you want to omit the encoding step, you may simply click this link to open a browser window with a pre-encoded program.

```c
#include <stdio.h>

int main() {
    int ctr;

    ctr = 1;  // initialization
    while (ctr <= 5) { // conditional expression
        // body of the loop
        printf("ctr is %d\n", ctr);
        ctr++; // change of state
    }
    return 0;
}
```

Figure 5.3: while loop that generates/prints a series of integers from 1 to 5

2. Let's see a visualization of the program. Press "Visualize Execution" button, and then press the "Next" button twice. Notice that the initial value of the counter variable named as **ctr** is garbage before executing the code in Line 6. Press the "Next" button again; you'll see that **ctr** gets initialized to 1. Forgetting to initialize a counter variable is one of the common logical errors committed by novice programmers.
3. Click the "Next" button again. The conditional expression **ctr <= 5** will be evaluated as **1 <= 5** which results to a value of 1. Recall that a 1 means true; thus, the body of the loop will be executed next as shown in Figure 5.4 screenshot.

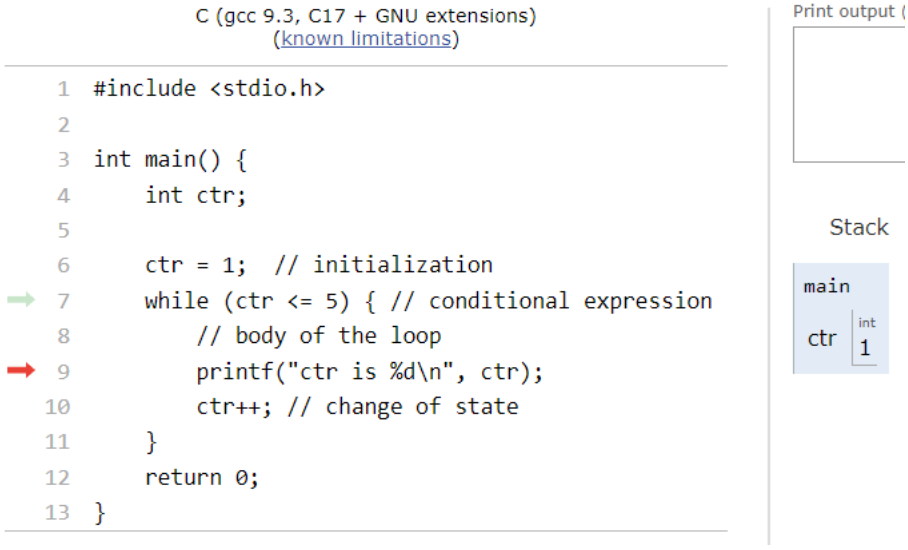

Figure 5.4: When ctr is 1, the condition ctr <= 5 is true; the body of the loop will then be executed

4. Click “Next” again to execute the printf() statement, and then click “Next” again to execute the increment operation **ctr++** which will set the value of ctr to 2. Thereafter, the program logic goes back to the start of the while loop such that the next step is to evaluate again the conditional expression **ctr <= 5** noting that ctr = 2 at this point in time.
5. Let’s move faster by clicking on the “Next” button several more times until the value of ctr becomes 6, and the conditional expression **ctr <= 5** will be evaluated again shown as Step 18 in Figure 5.5. Since ctr = 6, the condition will be evaluated as 6 <= 5 which is equal to 0, i.e., false which will cause the loop to terminate.
6. Click on “Next” again, and you should see that the loop has terminated, and that the next step to execute is return 0. Clicking “Next” one last time will then terminate the program.

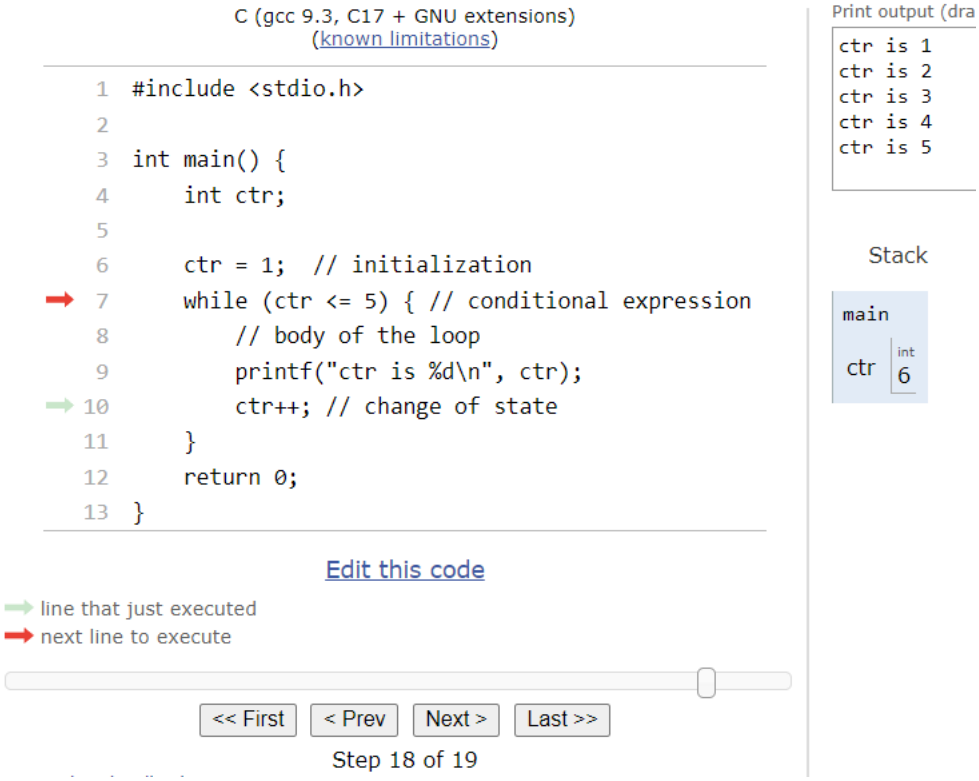

Figure 5.5: When ctr is 6, the condition ctr <= 5 is false causing the loop to terminate

Learning Activity 5.2: Common logical errors by novice programmers – uninitialized counter variable

There are many possible logical errors that a novice programmer will commit when writing programs with a while loop. We'll experience some of the common errors in the succeeding learning activities.

Learning Outcome: The student can trace the logic of a **while** loop that contains a logical error.

Instructions:

1. Edit the previous source code by simply commenting out the initialization code **ctr = 1** as shown in Figure 5.6. In this case, ctr will be uninitialized, i.e., ctr contains a garbage value.

```
1   #include <stdio.h>
2
3   int main() {
4       int ctr;
5
6       // ctr = 1;  // note: ctr contains garbage value!!!
7       while (ctr <= 5) { // conditional expression
8           // body of the loop
9           printf("ctr is %d\n", ctr);
10          ctr++; // change of state
11      }
12      return 0;
13  }
```

Figure 5.6: What will happen when the ctr variable is uninitialized?

2. Compile and run the program, NOT in C Tutor for now, but in your DEV-CPP or VSC IDE. What did you get as a result?

   Note that depending on the platform you're using, there's a probability that you will get the same result as in the previous learning activity, i.e., values of 1 to 5 will be printed. At best this is "tsamba"!

3. This time, try to execute the program in C Tutor (click this link to open a browser window with a pre-encoded program). What do you think will happen? Click on the "Next" button twice.

   C Tutor stopped executing at Line 7 because **ctr** was uninitialized. Recall that we already experienced the same logical error way back in Module 1. C Tutor flags an error message stating

   ERROR: Conditional jump or move depends on uninitialised value(s)
   (Stopped running after the first error. Please fix your code.)

   Remember: do NOT forget to initialize your loop's counter variable

Learning Activity 5.3: Common logical errors by novice programmers – incorrect conditional expression

Another common logical error that novice programmer commit is specifying an incorrect loop conditional expression.

Learning Outcome: The student can trace the logic of a **while** loop that contains a logical error.

Instructions:

1. Edit Line 6 of the previous source code by uncommenting **ctr = 1** in Line 6, i.e., the initialization will be part of the code again. Afterwards, edit Line 7 be removing the equal sign in the condition. The edited conditional expression should now be **ctr < 5**. The edited code should be the same as in Figure 5.7. You may also simply click this link to open a browser window with a pre-encoded program.

```
1   #include <stdio.h>
2
3   int main() {
4       int ctr;
5
6       ctr = 1;
7       while (ctr < 5) { // logical error: incorrect condition!
8           // body of the loop
9           printf("ctr is %d\n", ctr);
10          ctr++; // change of state
11      }
12      return 0;
13  }
```

Figure 5.6: What will happen when the loop conditional expression is incorrect?

2. Execute the program in C Tutor by clicking on "Visualize execution" and "Next" button several times until the last step. Make sure to keenly observe the visualization as you go through each step.

   Recall that the original intent of the program is to print the integers from 1 to 5. What did you get as program output?

   Notice that the program printed integers from 1 to 4 only. There was no output corresponding to integer 5. The root cause of this logical error is due to an incorrect loop condition. Removing the equal sign in the original condition caused the loop condition to become false when ctr was set to 5, i.e., 5 < 5 is 0 (false).

3. Edit the source code again. This time, change the condition to **ctr < 6**. Will this work? Visualize and trace the program to see if it does.

   Remember: make sure that the loop's conditional expression is correct.

Learning Activity 5.4: Common logical errors by novice programmers – infinite loop

An infinite loop is a loop that never terminates. Take note that if the program is required to terminate at some point in time, but if it doesn't – a possible cause of the logical error is an infinite loop.

Learning Outcome: The student can trace the logic of a **while** loop that contains a logical error.

Instructions:

1. Edit the previous source code by changing the condition back to **ctr <= 5**, and then by commenting out Line 10, i.e., by removing the **ctr++** statement from the program's logic as shown in Figure 5.7.

```
1   #include <stdio.h>
2
3   int main() {
4       int ctr;
5
6       ctr = 1;  // initialization
7       while (ctr <= 5) { // conditional expression
8           // body of the loop
9           printf("ctr is %d\n", ctr);
10          // ctr++; // logical error: NO change of state!
11      }
12      return 0;
13  }
```

Figure 5.7: What will happen when ctr++ is removed from the program logic?

2. Compile and run the program, NOT in C Tutor for now, but in your DEV-CPP or VSC IDE. What did you get as a result?

   You should get a program that repetitively prints "ctr is 1" infinitely. The logical error was caused by the removal of **ctr++**. Since ctr was initialized to 1, and it was no longer changed via **ctr++**, the loop's conditional expression **ctr <= 5** will always be true; thus, the infinite loop.

3. This time execute the program in C Tutor (click this link to open a browser window with a pre-encoded program). What do you think will happen? Click on "Visualize execution" button. Notice that the program appears to be non-responsive for a few seconds, and then it flags a message at the bottom portion of the screen stating as shown in the screenshot in Figure 5.8.

   You can actually still press the "Next" button many times, but the program will just print "ctr is 1" again, and again, and again…

   Remember: make sure to update the counter such that the condition will become false at some point in time.

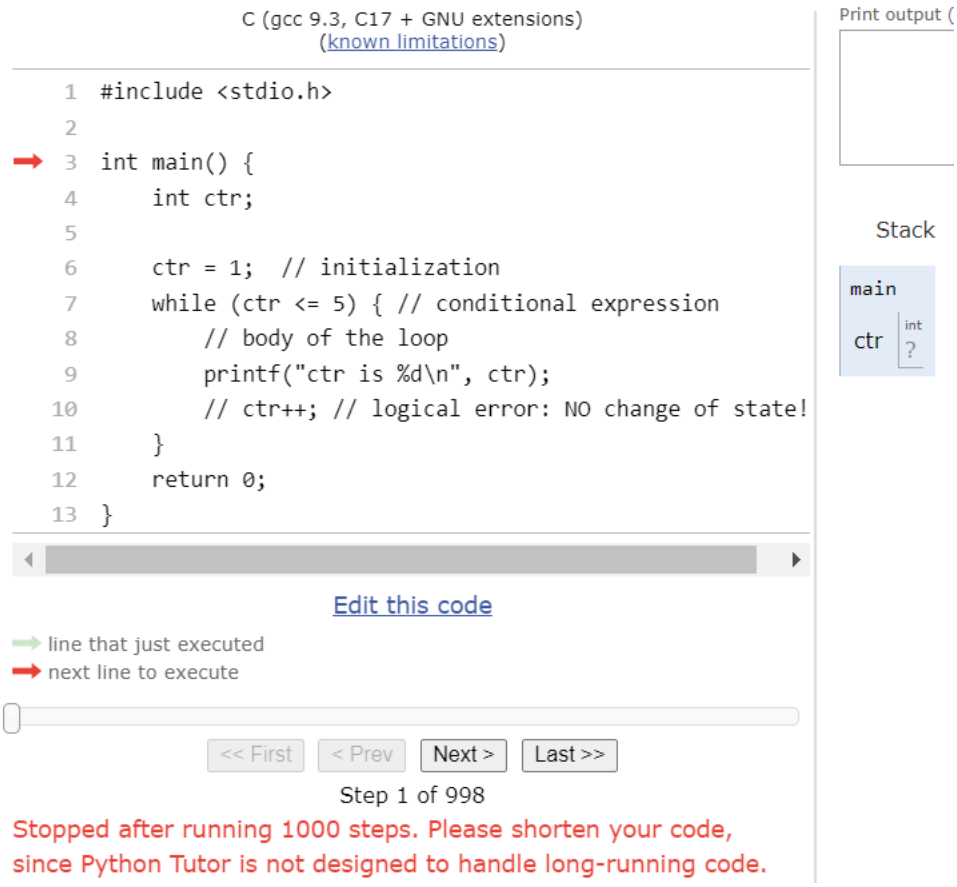

Figure 5.8: C Tutor flags a message caused by an infinite loop logical error even BEFORE clicking on the “Next” button

Learning Activity 5.5: Common logical errors by novice programmers – incorrect statement sequencing

Changing the sequencing of statements will affect a program's logic; in the worst-case scenario, it will cause a logical error.

Learning Outcome: The student can trace the logic of a **while** loop that contains a logical error.

Instructions:

1. Edit the previous source code by interchanging the codes in Lines 9 and 10, i.e., the increment ctr++ should come first before the printf() statement as shown in Figure 5.8. Alternatively, you may also simply click this link to open a browser window with a pre-encoded program.

```
1   #include <stdio.h>
2
3   int main() {
4       int ctr;
5
6       ctr = 1;  // initialization
7       while (ctr <= 5) { // conditional expression
8           // logical error: incorrect sequencing
9           ctr++; // change of state
10          printf("ctr is %d\n", ctr);
11      }
12      return 0;
13  }
```

Figure 5.8: What will happen if we interchange the sequence of the statements in Lines 9 and 10?

2. Execute the program in C Tutor by clicking on "Visualize execution" and "Next" button several times until the last step. Make sure to keenly observe the visualization as you go through each step.

   Recall again that the original intent of the program is to print the integers from 1 to 5. What did you get as program output?

   Notice that the program printed integers from 2 to 6. There was no output corresponding to integer 1, and there was an extraneous output for integer 6. The root cause of the logical error is the incorrect sequencing of the statements in Lines 9 and 10.

   Remember: make sure that the sequencing of the statements in the body of the loop is correct.

Learning Activity 5.6: Tracing a for loop

All C programs written with a **while** loop can be rewritten and changed to a **for** loop. The **for** loop is actually the more frequently used C programmers in codes that require counting or generation of a sequence of numbers. The reason for this is that the initialization, the condition, and the counter's change of state (can all) appear neatly on the same line as the **for** keyword.

Let's generate the numbers 1 to 5 again, this time using a **for** loop.

Learning Outcome: The student can trace the logic of a **for** loop.

Instructions:

1. Encode the source code shown in Figure 5.9. Or if you want to omit the encoding step, you may simply click this link to open a browser window with a pre-encoded program.

```
1   #include <stdio.h>
2
3   int main() {
4       int ctr;
5
6       for (ctr = 1; ctr <= 5; ctr++) {
7           // body of the loop
8           printf("ctr is %d\n", ctr);
9       }
10      return 0;
11  }
12
```

Figure 5.9: **for** loop that generates/prints a series of integers from 1 to 5

2. Let's see a visualization of the program. Press "Visualize Execution" button, and then press the "Next" button several times until the program terminates. The execution sequence will be the same with what you have already experienced in Learning Activity 5.2 above.

   The matching curly brackets in Lines 6 and 9 are actually unnecessary since there is only one statement in the body of the **for** loop, i.e., the printf statement in Line 8.

Learning Activity 5.7: Common logical errors by novice programmers – incorrect semicolon in a for loop

When to place a semicolon symbol in a code can prove to be a struggle to novice programmers. A semicolon when placed immediately after the right parenthesis in a loop causes a major logical error that can be difficult to find and debug.

Learning Outcome: The student can trace the logic of a **for** loop that contains a logical error.

Instructions:

1. Edit the previous code by inserting a semicolon symbol immediately after the right parenthesis as shown in Figure 5.10 or click this link to open a browser window with a pre-encoded program.

```
1   #include <stdio.h>
2
3   int main() {
4       int ctr;
5       // logical error: incorrect semicolon in Line 6
6       for (ctr = 1; ctr <= 5; ctr++);  {
7           // body of the loop
8           printf("ctr is %d\n", ctr);
9       }
10      return 0;
11  }
```

Figure 5.10: incorrect semicolon in Line 6

2. Let's see a visualization of the program. Press "Visualize Execution" button, and then press the "Next" button several times until the program terminates.

   Did the program produce the integers 1 to 5?

   The semicolon after the right parenthesis in Line 6 completely changes the program logic causing it to produce a totally different, i.e., incorrect result. This extraneous semicolon caused the body of the loop to become empty, i.e., there is nothing to execute as part of the loop body. The left and right curly brackets together with the printf statement are in fact no longer part of the **for** loop (don't be misled by the indentation), and treated as codes outside and AFTER the for loop.

   Remember: never put an extraneous semicolon after the right parenthesis in a while and a for loop.

--- End of Module 5. Up Next: Tracing and visualizing nested loops. –

The answers to Check-up Quiz #5 are shown in the footnote on this page and on the next page[2] .

[2] Q1: The program below will print integer values from 1 to 5. How many times will the conditional expression **ctr <= 5** be evaluated?
A1: 6 – the first 5 times the condition is true, the 6th time the condition is false

Q2: The program below will print integer values from 1 to 5. What will be the final value of **ctr** right after the loop and just before executing the **return 0** statement?
A2: 6
Q3: A novice programmer wrote the program shown below with the objective of printing integer values from 1 to 5. BUT there may be a logical error because it does not work as intended when executed in some platform. What do you think is the cause of the logical error?
A3: The program output is unknown because ctr was not initialized.
Q4: What do you think is the cause of the logical error?
A4: The program will print the ctr values from 1 to 4 only because the loop condition is incorrect.
Q5: What do you think is the cause of the logical error?
A5: The program will print a ctr value of 6 only because of an incorrect semicolon in Line 6.