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

Florante.Salvador@dlsu.edu.ph
College of Computer Studies, De La Salle University

MODULE 1: Getting Familiar with the C Coding Tutor Interface

1.1 What is C Coding Tutor?

C coding tutor is part of a free web-based application named Python Tutor (pythontutor.com) that enables novice programmers to explore, trace, debug and visualize programs written in introductory programming languages such as the C programming language. It was developed and is continuously being maintained by Philip Guo as described in (Guo, 2021).

We will refer to C coding tutor as C Tutor for reason of brevity throughout the remainder of this document. Let's start our exploration by clicking on the following link: https://pythontutor.com/c.html#mode=edit

1.2 C Tutor Interface

Our objective for this module is to become familiar in using the C Tutor interface as shown in Figure 1.1. There is an edit area where you can edit or encode your source code. Each line of code (including an empty line) is identified with a line number. C Keywords such as `int` and `return` are color coded in blue. The default initial screen interface shows four lines of codes with a main function with just one statement, i.e., `return 0`.

Clicking the "Visualize Execution" button shown at the bottom portion of the screen will initiate the visualization of the program execution. Click on the "Visualize Execution" button and see what happens.

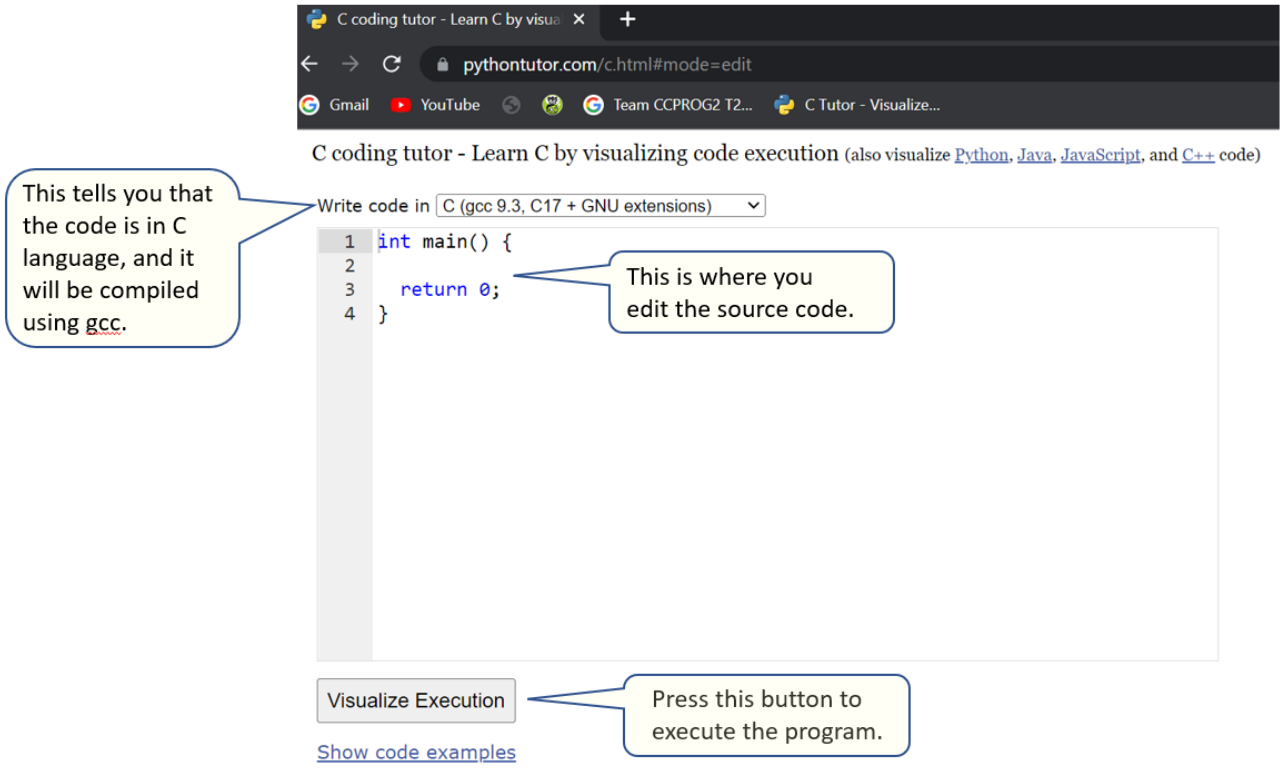

Figure 1.1: Annotated C Tutor initial screen interface

Figure 1.2 shows a screenshot of the interface during program execution. The user is provided the following useful information:

- the line of code that has just been executed – this is visualized with a light green arrow
- the line of code that will be executed next – this is visualized with a red arrow; in the default code, it is line 3 with return 0 that will be executed next
- the name of the function that is currently executing – this is visualized within a box in bluish gray background color; in the default code it is `main` which is shown on the right half of the screen interface.
- the names and current values of the local variables (if there are any); note that in the default code there is no local variable [we'll introduce variables in another example program…]

Notice also that there are four buttons named **First**, **Prev**, **Next** and **Last** which allows you to step forward and backward in the code sequences. The slider bar shown on top of these buttons visualizes the progress of execution. Finally, the current step number and the total number of steps for the entire code are shown just below these buttons. It is shown as Step 1 of 1 in Figure 1.2.

HINT: It is actually the **Prev** and **Next** buttons that you'll need to press often when you do the actual tracing of the program's logic.

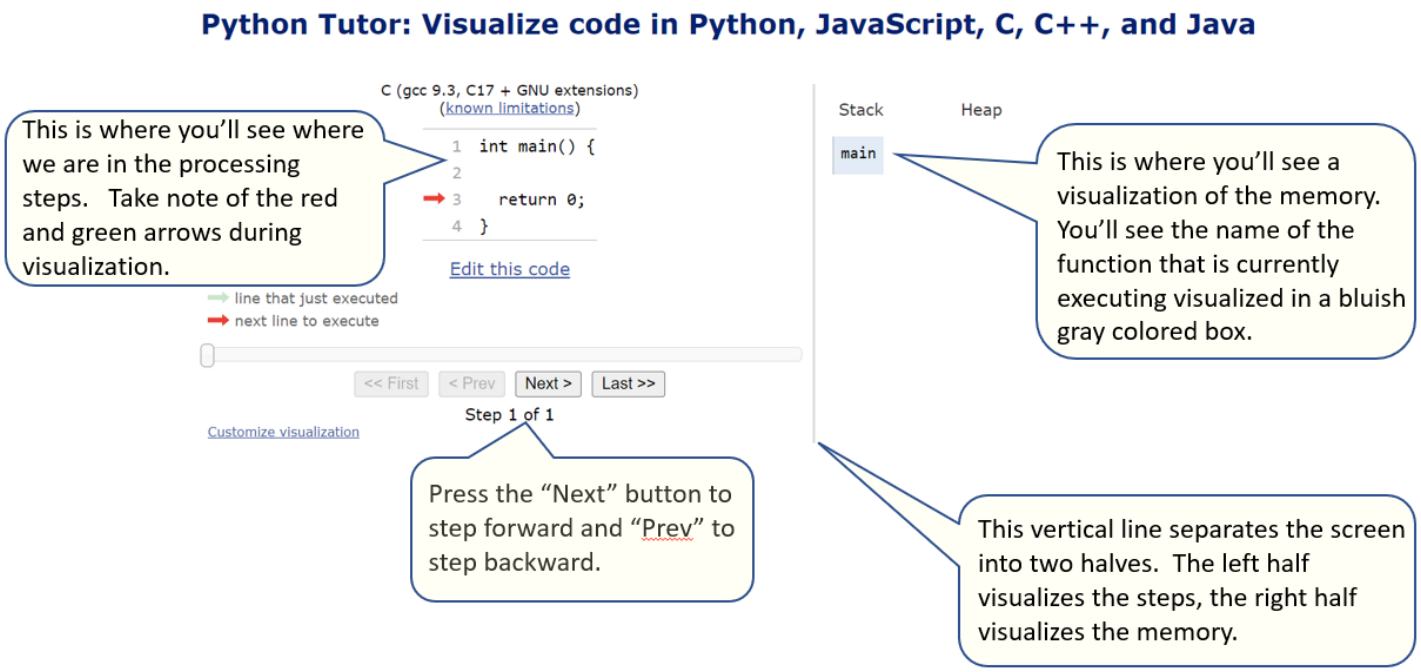

Figure 1.2: Annotated screen Interface during program execution

Learning Activity 1.1: Edit, trace and visualize the default source code.

Learning Outcome: The student can navigate and use the C Tutor interface.

Instructions:

1. Edit the default source code by inserting a printf statement that will simply display “Hello world!” as shown below.

```
1  int main() {
2    printf("Hello world!");
3    return 0;
4  }
```

2. Next, click on “Visualize Execution” button to trace and visualize the program execution. You should see the same screen as shown in Figure 1.3.

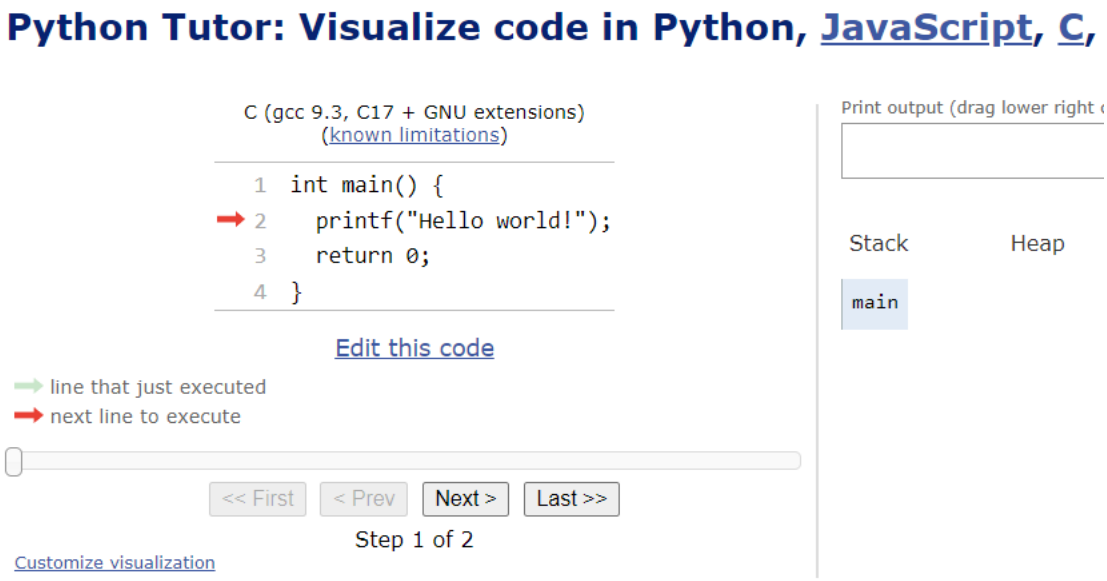

Figure 1.3: Visualization of the steps, i.e., flow of program execution.

Please answer the guide questions in the given sequence below. The expected correct answers are indicated in the footnote[1].

a. Which line will be executed next?
b. Click on the "Next" button to execute the first statement. Where was the printf statement result displayed?
c. Which line will be executed next?
d. How do you go back to the previous step?
e. How do you rewind back to the first step?
f. How do you jump forward to the last step?

With reference to Learning Activity 1.1, we see in Figure 1.4 the following:

- Line 2 was the line that has just executed as indicated by the green arrow
- Line 3 is the next line to execute; it is Step 2 in the program steps
- The printf result is displayed inside the "Print output" box shown on the right side of the screen

Figure 1.4: The result of the printf() statement is displayed in the output box

[1] Learning Activity 1.1 Answers:
(a): Line 2 – as visualized by the red arrow pointing to it. Notice also the printf statement is the first statement, i.e., Step 1 of 2 in the program.
(b) The string "Hello world!" was displayed on the right side of the screen interface inside the box labeled as "Print output". As indicated in the interface, you can drag the lower right corner to increase or decrease the size of the output box. This will be necessary if you have multiple lines of output.
(c) Line 3 as visualized by the red arrow pointing to it. The return statement is the second and the last statement, i.e., Step 2 of 2 in the program. Notice also that the green arrow points to the previous Line, i.e., Line 2 indicating that it was the line that has just executed.
(d) Press the "Prev" button.
(e) Press the "First" button.
(f) Press the "Last" button.

1.3 Errors When Using C Tutor

You may encounter different errors upon clicking the “Visualize Execution” button.

1.3.1 Server Error

A server error can be due to for several possible reasons as indicated in Figure 1.5. If you experience that it takes a long time for the “Visualize Execution” to take effect, it can be due to a large crowd traffic, i.e., many users trying to connect to C Tutor’s server simultaneously which causes a server overload. This is beyond your control. In such a case, please be patient, wait for a few minutes and retry clicking on the “Visualize Execution” again until you get connected.

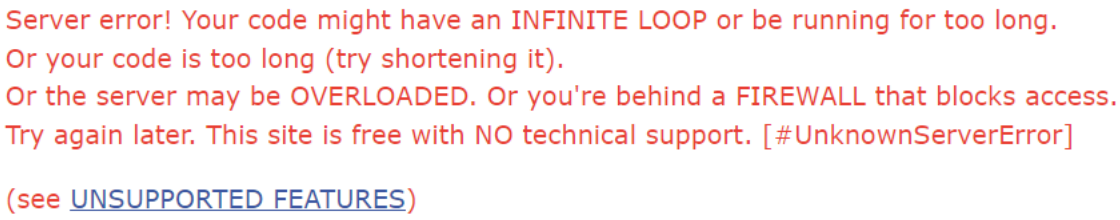

Figure 1.5: Screenshot of a server error message

1.3.2 Syntax Error

A syntax error will happen if there is a failure in following the syntax of the C language, such as incorrect names, forgetting to place a required symbol such as a curly bracket, or a semicolon. In such a case, you’ll need to fix first the syntax error, then retry clicking the “Visualize Execution” button again.

Let’s deliberately commit a syntax error to see what actually happens in C Tutor. Edit the source code by removing the semicolon associated with the printf statement in Line 2. Thereafter click “Visualize Execution”. In this case, C Tutor flags an error message error: expected ‘;’ before ‘return’ as shown in the screenshot in Figure 1.6. Moreover, an “x” mark in red background color is shown in Line 2 to visually indicate that this is where the offending code is located.

Figure 1.6: An example of a syntax error due to a missing semicolon in Line 2

Fix the error by encoding the missing semicolon in Line 2. Thereafter, click the "Visualize Execution" button again. A pop-up window, as shown in Figure 1.7, will appear as user feedback that a syntax error was fixed. At this point in time, simply click on the "Close" button to close the pop-up window and commence with the program execution.

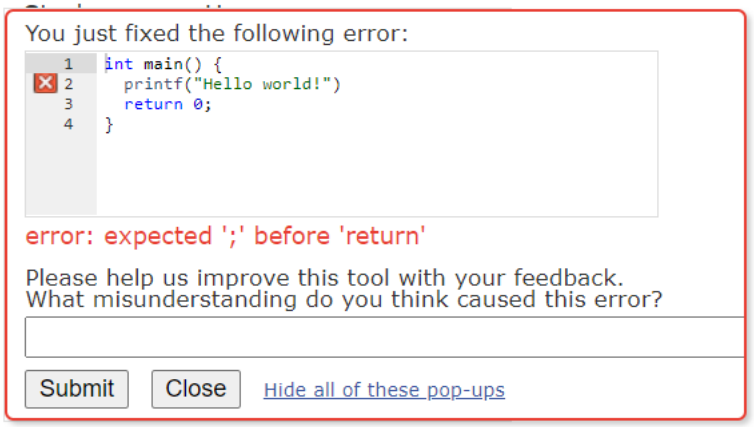

Figure 1.7: User feedback window

TIP: some users may find it easier to encode, compile and fix the syntax errors in a source code via an IDE like Dev-CPP, VSC or XCode rather than doing these inside C Tutor. Once the code is free of syntax errors, the code can just be copy/pasted onto C Tutor's edit window and proceed with the visualization.

1.3.3 Semantic Error

A semantic error will occur during run-time if there is an error in the logic of the program. Common logical errors committed by novice programmers can be due to different reasons such as incorrect sequencing of statements, using the values of uninitialized variables, incorrect conditions, passing incorrect parameters, among other things. In such a case you'll need to debug your code, i.e., find first which line causes a semantic error, fix it accordingly, and then try clicking the "Visualize Execution" button again to determine if the fix worked.

We'll explore codes with semantic errors and discuss how to debug them in other modules.

1.3.4 Unsupported Feature Error

Please take note that there are MANY features that are not supported in C Tutor. One key example is that a **scanf** statement cannot be processed in a C Tutor code. For testing purposes, you'll need to instead initialize a variable via assignment operator. We'll explore an example code in the next module. For now, please click this link and do an initial browsing of the list and description of unsupported features; page 3 of the online document describes the limitations specific to C language.

Reference:

Philip Guo. 2021. Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia. In Proceedings of the UIST '21: The 34th Annual ACM Symposium on User Interface Software and Technology. Association for Computing Machinery, New York, NY, USA, 1235–1251 https://dl.acm.org/doi/10.1145/3472749.3474819

--- End of Module 1. Up Next: Tracing and visualizing variables. ---