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

Preliminary Task: please click this link to a google form for Check-up Quiz #1 and answer a simple non-graded check-up quiz about variables. 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 topics on variables and functions in your CCPROG1 class. The objective of the quiz is to determine possible misconception about variables which we intend to clarify and remediate as part of the learning activities. You'll find the quiz answers in the footnote of page 2 of this document.

MODULE 2: Tracing and Visualizing Variables

A variable is essentially a memory space used for storing a value of a certain data type. The four basic data types that you have already learned in CCPROG1 are char, int, float and double. There are three ways to specify variables in a C program, namely by (a) variable declaration, (b) variable definition, (c) as formal parameters in a function definition.

Do you know the difference between variable declaration and a variable definition? The learning activities in this module will provide a visual explanation of the difference between the meaning of the words "declare" and "define" in the context of C Language.

Learning Activity 2.1: Visualizing a single integer variable

Learning Outcome: The student can determine the initial value of a variable.

Instructions:

1. Encode the source code shown in Figure 2.1. Do not use C Tutor for now – we'll do later in step 3 below. You should instead encode, compile and execute the code via the IDE that you usually use in your CCPROG1 class such as DEVCPP or Visual Studio Code (VSC). The code is the same one given in Check-up Quiz #1.

```
1  #include <stdio.h>
2
3  int main() {
4      int x;
5
6      printf("%d\n", x);
7      return 0;
8  }
```

Figure 2.1: Source code encoded in Visual Studio Code (VSC) IDE

2. Please answer the questions below. Refer to the footnote for the expected correct answers[1].

    a. What is the value of variable x just BEFORE Line 6, i.e., before the printf statement?
    b. Using DEV-CPP/VSC/XCode, compile the source code and then run the executable file. What value was displayed by the printf statement?

3. Next, we trace and visualize the same program using C Tutor. Please copy/paste the source code onto the C Tutor edit window area. Thereafter, click on the "Visualize Execution" button. You should have something similar to the screenshot in Figure 2.2 below.

    With C Tutor we can track and visualize the value of a variable throughout the program's execution. As shown in the screenshot, **main** is the function that is currently executing, and that **x** is a local variable inside **main**. It also shows two important information: (a) the data type of **x** is **int**, and (b) that its current value is garbage. The question mark **?** is the symbol used by C Tutor to indicate that a variable currently contains an unknown or garbage value.

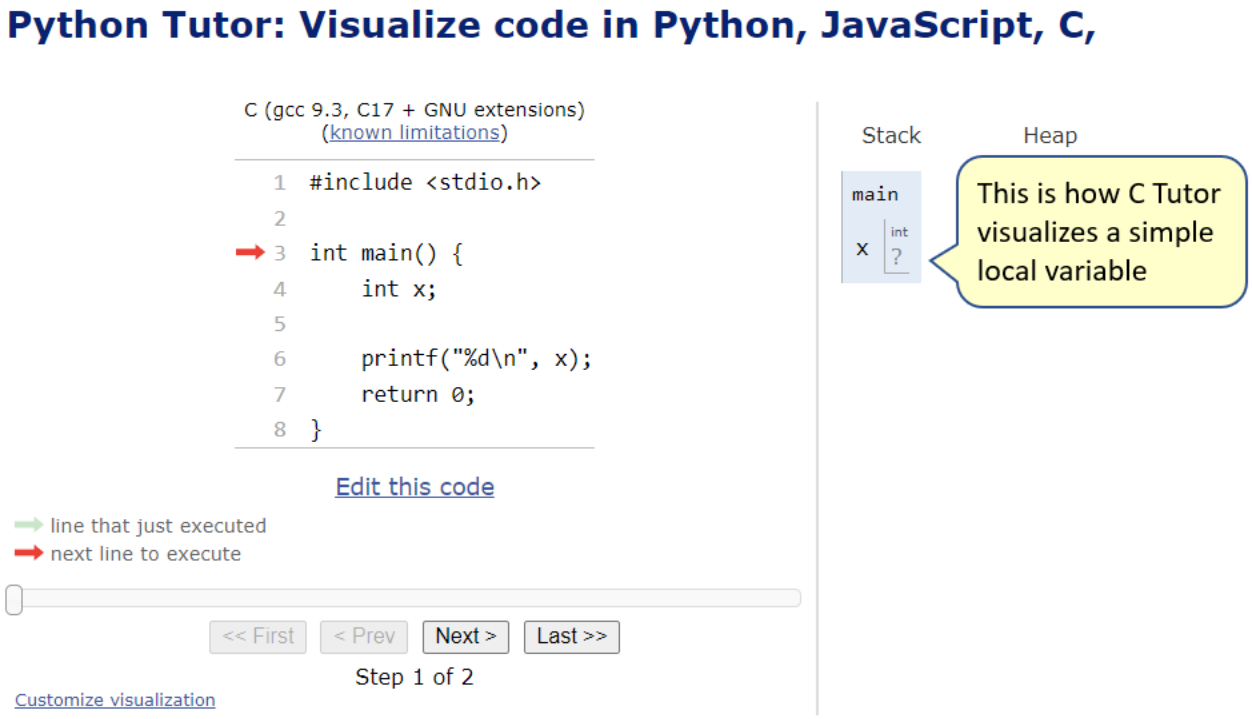

Figure 2.2: Visualization of local variable x; the question mark ? means that the value is garbage or uninitialized

[1] Learning Activity 2.1 (also Check-up Quiz #1) Answers:
(a) Take note that x contains a **garbage value**, not zero! Another way to say this is that x contains a value, but we (the programmer) do not know the actual value, i.e., the **value is unknown**. If you answered "I do not know" for this question, it is actually a correct answer 😊.
(b) In the computer system that I used (author), the program displayed a value of 0. Think of this as just a "coincidence" (in Filipino, the better term is "tsamba lang"). You may see a different number in your own computer system.

.

4. Let's trace the steps. Click on the "Next" button. The red arrow points to Line 6 which means that that printf is the next statement to be executed. Click on "Next" button again. Notice that the printf was not executed by C Tutor. The program stopped executing and C Tutor flagged an error message stating

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

   The key phrase that we should take note of here is move depends on uninitialized value(s) which was caused by variable x having a garbage value. C Tutor prohibited printf from displaying a garbage value because it is considered as a semantic or logical error.

   IMPORTANT NOTE: This is a different behavior compared with running the exe file via DEVCPP/VSC/XCode IDE or in the command line where the program did not stop, and printf displayed a garbage value.

Learning Activity 2.2: Visualizing char, int, float and double variables

Learning Outcome: The student can determine the initial value of a variable.

Instructions:

1. Let's introduce more variables with different data types. Encode the following source code. Again, do not use C Tutor for now.

```
1   #include <stdio.h>
2
3   int main() {
4       char ch;
5       int i;
6       float f;
7       double d;
8
9       printf("%c %d %f %lf\n", ch, i, f, d);
10      return 0;
11  }
```

Figure 2.3: Source code with char, int, float and double variables encoded in Visual Studio Code (VSC) IDE

2. Answer the questions below. Refer to the footnote for the expected correct answers[2].

    a. What are the values of ch, x, f and d just BEFORE Line 9, i.e., before the printf statement?
    b. Using DEV-CPP/VSC/XCode, compile the source code and then run the executable file. What values were displayed by the printf statement?

3. Next, we examine the same program using C Tutor. Please copy the source code that you have encoded and paste it onto the C Tutor edit window area. Thereafter, click on the "Visualize Execution" button and then click on the "Next" button twice. You should have something similar to Figure 2.4 below.

    Look at the visualization on the right side of the screen. It indicates that **main** is the function that is currently executing, and that **ch**, **i**, **f** and **d** are local variables inside **main** and that their values are all garbage.

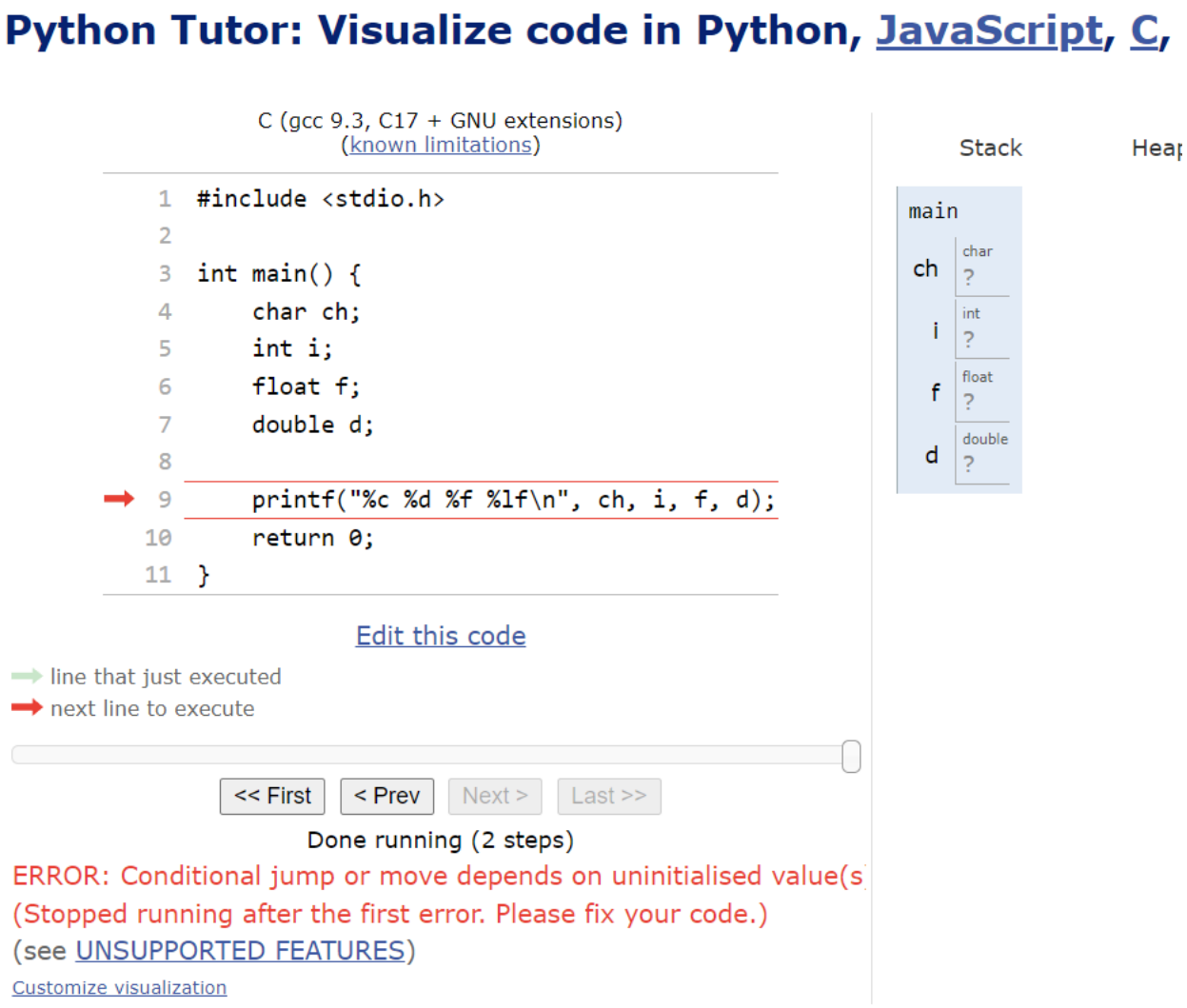

Figure 2.4: Visualization of uninitialized variables ch, i, f and d

[2] Learning Activity 2-1 Answers:
(a) Variables ch, i, f and d each contain a **garbage value**, not zero!
(b) In the computer system that I used, the program displayed 660 0.000000 0.000000. Note that there is an invisible character before the first digit 6. These are all garbage values.

.

Learning Activity 2.3: Fixing an uninitialized variable logical error

Accessing a garbage value of an initialized variable is a logical error. You must first initialize a variable by assigning a value to it.

Learning Outcome: The student can fix an uninitialized variable logical error.

Instructions:

1. Encode and edit the incomplete source code shown in Figure 2.5 to initialize the four variables. The comments serve as guide on what to use as initial values. 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      char ch;
5      int i;
6      float f;
7      double d;
8      // Fill in the blanks with the correct initialization code
9      ______; // initialize ch to 'A'
10     ______; // initialize i to 123
11     ______; // initialize f to 4.56
12     ______; // initialize d to 3.1416;
13     printf("%c %d %f %lf\n", ch, i, f, d);
14
15     // Update the values of the variables
16     ch++;
17     i--;
18     f = f/2;
19     d = d * 2;
20     printf("%c %d %f %lf\n", ch, i, f, d);
21
22     return 0;
23 }
```

Figure 2.5: Incomplete source code for this activity

2. Next, trace and visualize the execution of the program by clicking on the "Visualize Execution" button and clicking on the "Next" button several times until all steps are completed.

   Take special note of the visualization of the variables as you step through the program execution. You will see that the values of the variables change due to the initializations in Lines 9 to 12, and the operations done in Lines 16 to 19.

   A screenshot of the values just before executing the first and the second printf statements are shown in Figures 2.6 and 2.7 respectively.

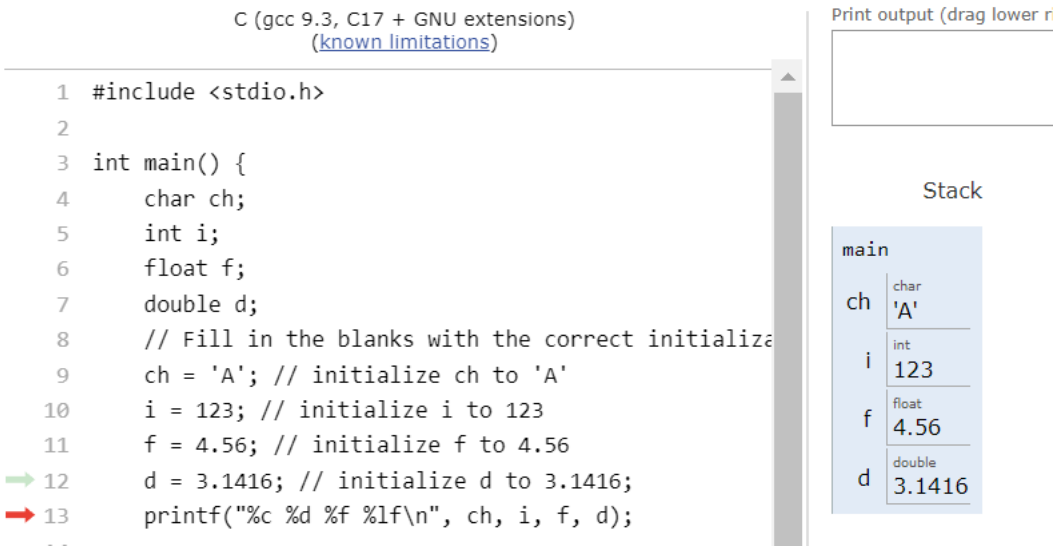

Figure 2.6: Values of the variables before executing Line 13

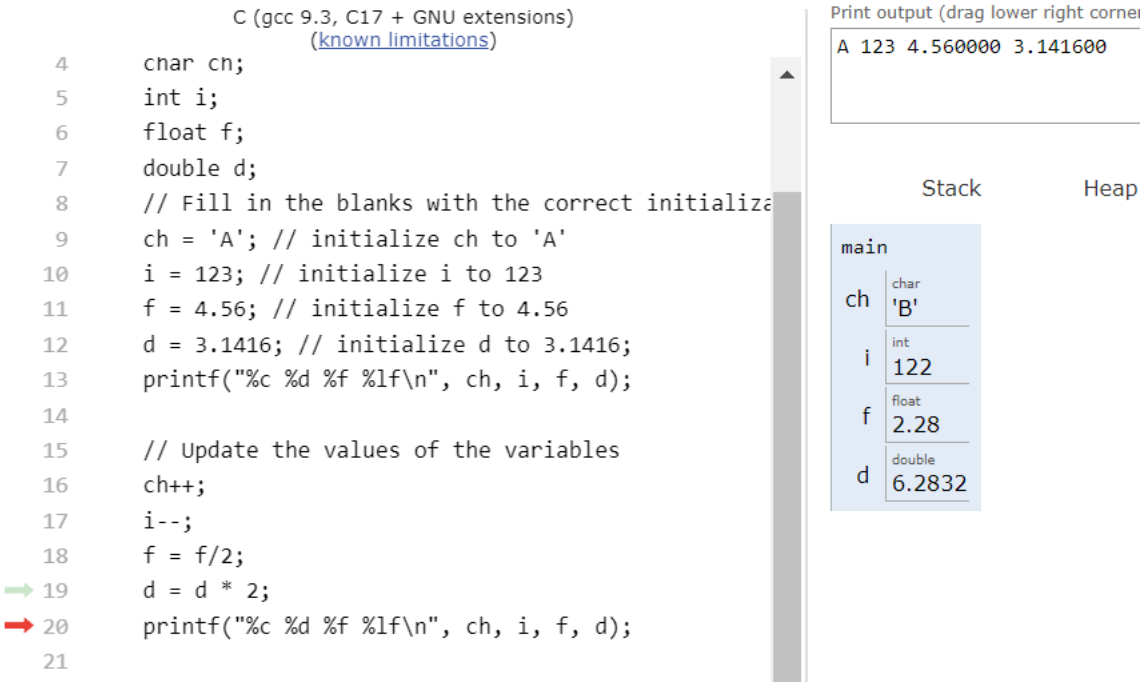

Figure 2.7: Values of the variables before executing Line 20

Learning Activity 2.4: Defining a variable

It is assumed that you learned the difference between a variable **declaration** from a variable **definition** in your CCPROG1 class. We'll trace a program and see a visualization of a variable definition in this learning activity.

Learning Outcome: The student can define a variable.

Instructions:

1. Encode the source code shown in Figure 2.8. 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      // Lines 5 to 8 are examples of variable definitions
5      char ch = 'A';
6      int i = 123;
7      float f = 4.56;
8      double d = 3.1416;
9
10     printf("%c %d %f %lf\n", ch, i, f, d);
11
12     // Update the values of the variables
13     ch++;
14     i--;
15     f = f/2;
16     d = d * 2;
17     printf("%c %d %f %lf\n", ch, i, f, d);
18
19     return 0;
20 }
```

Figure 2.8: Source code with variable definitions

2. Next, trace and visualize the execution of the program by clicking on the "Visualize Execution" button and clicking on the "Next" button several times until all steps are completed. Take note what happens after executing each line of variable definition from Lines 5 to 8.

Learning Activity 2.5: scanf is NOT supported in C Tutor

BEWARE: There are many unsupported features in C Tutor. One that is quite limiting for novice C programmers is that it does NOT support the scanf function. Let's experience what happens when there is a scanf() in the program.

Instructions:

1. Encode the source code shown in Figure 2.9. 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.

```c
#include <stdio.h>

int main() {
    int x;

    scanf("%d", &x); // note: scanf() is NOT supported in C Tutor
    printf("%d", x);
    return 0;
}
```

Figure 2.9: Source code with scanf()

2. Next, trace and visualize the execution of the program. Click on the "Visualize Execution" button, and then click on the "Next" button 3 times while observing what happens during each step. While it may seem that it got through the scanf step, it was not executed. Click the "Next" button the third time. At this point, C Tutor stops execution and flags an error message as shown in Figure 2.10. The nature of the error is exactly the same as what you already encountered in Learning Activity 2.1.

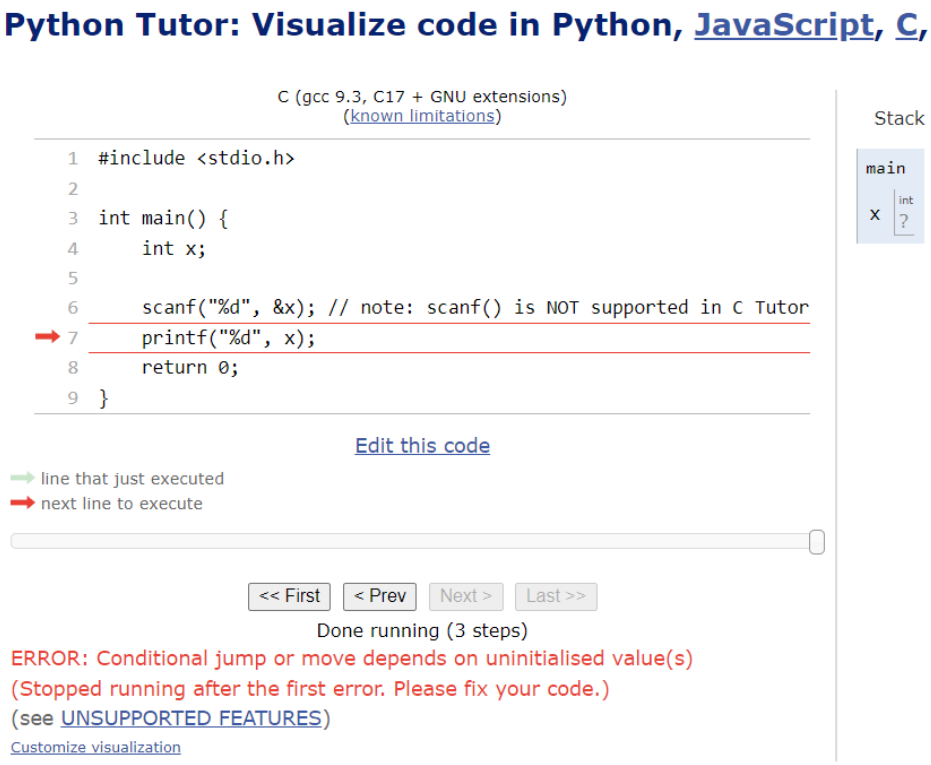

Figure 2.10: Error message due to unsupported scanf()

Bottomline, please avoid using scanf in C Tutor. The recommended alternative is to replace it with a variable initialization using an assignment operator. For example, replace scanf("%d", &x); with say x = 1; to be able to go through the program tracing and visualization and avoid an uninitialized value(s) error.

--- End of Module 2. Up Next: Tracing and visualizing functions. --