SOFTWARE SUPPORT FOR THE TRACING METHOD

Tom M. Warms
tomwarms@psu.edu
Department of Computer Science and Engineering
Abington College
The Pennsylvania State University

Abstract: In this paper, the first version of a program that supports the tracing method is described. The tracing method aids the student in demonstrating his or her understanding of the functioning of a computer program by providing a written record of the effect of each of the program's statements. This program, RandomLinearizer, randomizes the elements of the trace, and the student then selects the elements in the correct order to complete the trace. Formatting of the trace is done by RandomLinearizer. In this paper, the elements of the tracing method are recounted, and some examples of the functioning of RandomLinearizer are provided.

Keywords: Tracing, linearization, C++, programming

Introduction

Tracing a program is a method by which a record can be created of the effect of each of its statements. It is a way for instructors to communicate information about the features of a programming language and of algorithms, and for students to demonstrate their understanding. The first version of RandomLinearizer provides support for programs that have some of these features, such as iterative and recursive functions, and value and reference parameters.

Basic rules for tracing

The name of the function being executed appears above the vertical line. Names of identifiers are placed on the left side of the line and the identifiers' values on the right. Input values are underlined, boxes indicate output, ↵ indicates the RETURN character, and returned values are encircled. Indeterminate values are indicated by '?'. In tracing each statement, the values that resulted from tracing previous statements are available.

Some introductory examples

Figure 1 shows a program to calculate the sum of two input numbers, along with its trace. (Figures 1 – 8 first appeared in [1]. The tracing method was described in [2, 3]; some semantic properties of the method are shown in [4].) Arbitrary input values of 14 and 287 are assumed. Each executable statement in the program is traced in turn. When the user traces the statement in which sum is calculated, num1 and num2 already have values.

The underlined values (input), boxed material (output), and the RETURN character of a trace can be used to predict the contents of the console. From Figure 1, the contents of the console are seen to be as in Figure 2.

Tracing decision structures

The trace of an if or if-else statement contains the word "if" to the left of the vertical line, the Boolean value of the condition across the line, and the result of executing the if-true statement or the if-false statement on succeeding lines. The same holds for tracing switch statements: the value of the selector expression is again written across the vertical line, followed by the trace of the selected clause. In Figure 3, the traces of the indicated segments are given in a form in which variables are provided with assumed values: these values appear above the horizontal line, and the results of tracing the segments appear below the line. Italicized material annotates the trace but is not itself part of the trace.

Tracing looping structures

In tracing looping structures, the name of the structure is written to the left of the vertical line and encircled. In the for loop of Figure 4, n is assumed to be 2 and the input values are assumed to be 12.88 and 7.98: A squiggly line is drawn across the vertical line after each execution of the update clause.

Tracing array operations

The elements of a small array can be shown on one line, as at the beginning of the trace of Figure 5. The array elements can also be shown individually, as on succeeding lines. In the trace, the first three elements of array arr are rotated forward one place, with arr[2] being replacing by arr[0].

In order to determine values for the various array elements after the segment has been executed, one starts at the bottom for the most recent values: arr[0], arr[1], and arr[2] can be read from the starred lines, while the values for arr[3] and arr[4] have not changed from the first line of the trace.

Tracing value-returning functions

When a value-returning function is traced, the trace moves to the right, and the name of the function is written above a continuation of the vertical line. The values of the formal parameters are entered first, followed by a trace of the execution of the statements of the function body. The returned value is then encircled, and the trace returns to the left. Figure 6 shows a program in which a function is used to calculate the hypotenuse of a right triangle. In the trace, input values of 5.0 and 12.0 are assumed, and the function getHypotenuse returns a value of 13.0.

Tracing functions with reference parameters

Any change in the value of a reference parameter is reflected in a change in the value of the corresponding actual parameter. In the trace, a double arrow is used to indicate this relationship. Figure 7 shows a program in which a function, getAnswers, is used to calculate the sum and product of two input values, assumed to be 7 and 13. As control is transferred to getAnswers, the values of the formal parameters, num1 and num2, are shown in the trace to be 7 and 13. The reference parameters, total and prod, take on the indeterminate values of the corresponding

actual parameters sum and product. When the function body is executed, total, and therefore actual parameter sum, is calculated to be 20, the sum of num1 and num2. Then prod, and therefore actual parameter product, becomes the product of these values. Control is returned to the main program, and the values of sum and product are printed.

Tracing recursive functions

When control is transferred to a recursive function, the trace moves to the right, and the first value that is entered is that of the return address (RET). The rest of the trace is as before. In the program in Figure 8, a recursive function, getLarge, is used to calculate the largest element in an array. The return address marked a is in the top-level call, at the point where the result of the calculation is assigned to large. The return address marked b is in getLarge, and is at the point where the result of the recursive call is assigned to tempLarge. In the trace, the first 3 array elements are assumed to have values of 12, 28, and 15, and nElts is assumed to equal 3.

The software support

For each program in a variety of C++ programs that are encountered in the introductory computer science sequence, RandomLinearizer displays the program as well as an input set and a scrambled list of the elements that go into its trace. Each time the user clicks on the appropriate element, that element is copied to the other side of the screen, where a complete trace of the program is unfolding. In Figure 9, a program to read two integers and calculate and print the larger one is shown, along with the elements in scrambled order that comprise the trace.

The user is expected to click on those elements in the correct order. Each time the user succeeds, the element is copied to the right panel. In Figure 10, the user has clicked correctly on five panels, but not on the sixth, bringing forth a prompt from RandomLinearizer.

When the user has correctly completed the trace, RandomLinearizer presents the possibilities for continuing the session (Figure 11). The contents of the console window are also displayed.

In the program of Figure 12, a function, getSum, calculates the sum of two input values. The trace is shown after the user has successfully clicked on the elements that trace the input statement. In this example, the option is invoked in which trace elements that are no longer needed are dropped from the left panel. The completed trace, along with the contents of the console window, are shown in Figure 13.

In figure 7 above, the trace of a program that uses a function to calculate the sum and product of two integers is displayed. The RandomLinearizer version of that program before the trace unfolds is shown in Figure 14, and when the trace is complete in Figure 15.

Future work

Succeeding versions of RandomLinearizer will implement additional features of C++ and other programming languages, as well as a wider array of programming techniques. They will provide a variety of methods for the user to enter input and display his or her expertise, and a robust method for instructors to supplement the data that is built into the program.

References

[1] Warms, T.M. and Drobish, R., “Tracing the Execution of Computer Programs – Report on a Classroom Method,” Proceedings of the Spring 2007 ASEE Mid-Atlantic Section Conference, Newark, April 2007.

[2] Warms, Tom M., "Tracing the Execution of C++ Programs," *Inroads - the SIGCSE Bulletin*, Vol. 33, No. 4, (2001), 64-67.

[3] Warms, Tom M., " The Power of Notation: Modeling Pointer Operations," *Inroads - the SIGCSE Bulletin*, Vol. 37, No. 2, (2005), 41-45.

[4] Warms, T. M., "The Semantics of Tracing: Transitivity of Reference," Proceedings of FECS'07 — The 2007 International Conference on Frontiers in Education: Computer Science and Computer Engineering, Las Vegas, June 2007, pp 302 – 307.

Figures

```
// This program prompts the user for two integers, and calculates
// and prints their sum
#include <iostream>
using namespace std;
int main()
{
      int num1, num2, sum;
      // Prompt for input
      cout << "Enter two integers:";
      cin >> num1 >> num2;
      // Calculate sum
      sum = num1 + num2;
      // Print result
      cout << "The sum of " << num1 << " and " << num2
            << " is " << sum << endl;
      return 0;
}
```

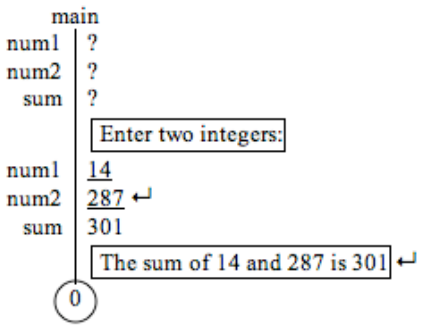


Figure 1

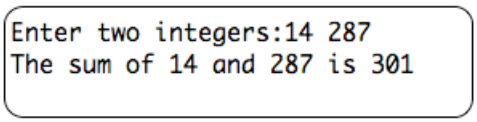
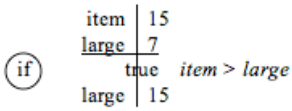


Figure 2

```
if (item > large)
      large = item;
```




```
switch (party)
{
      case 'D': numDs++;
break;
      case 'R': numRs++;
break;
      default: numIs++;
break;
}
```

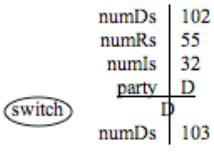


Figure 3

```
double sum = 0.0;
for (int j = 0; j < n; j++)
[
        double cost;
        cout << "Cost: $";
        cin >> cost;
        sum += cost;
}
cout << "Total cost = $" << fixed
        << showpoint << setprecision(2)
        << sum << endl;
```

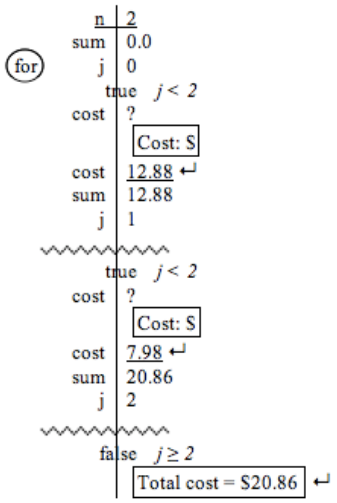


Figure 4

```
int temp = arr[0];
for (int j = 0; j < nElts - 1; j++)
        arr[j] = arr[j + 1];
arr[nElts – 1] = temp;
```

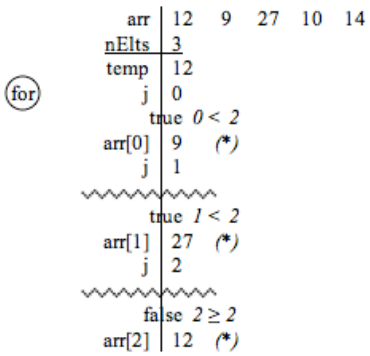


Figure 5

```
// Program to prompt the user for the legs of a right
// triangle and calculate and print the hypotenuse
#include <iostream>
#include <cmath>
using namespace std;
double getHypotenuse (double leg1, double leg2);
int main()
{
      double leg1, leg2;

      // Prompt for legs of triangle
      cout << "Enter legs of right triangle:";
      cin >> leg1 >> leg2;

      // Calculate hypotenuse
      double hypotenuse = getHypotenuse(leg1, leg2);

      // Print hypotenuse
      cout << "Hypotenuse = "  << hypotenuse << endl;
      return 0;
}

double getHypotenuse (double x1, double x2)
// getHypotenuse calculates the hypotenuse of a
// right triangle whose legs are x1 and x2
{
      double sumOfSquares = pow(x1, 2.0) + pow(x2, 2.0);
      double hyp = sqrt(sumOfSquares);
      return hyp;
}
```

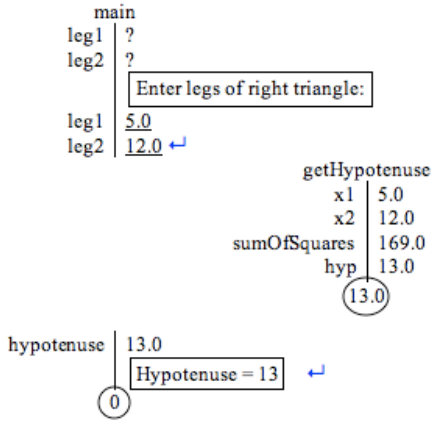


Figure 6

```
// Program to prompt user for two integers and calculate
// and print their sum and product.
#include <iostream>
using namespace std;
void getAnswers (int num1, int num2, int &total, int &prod);
int main()
{
      int x1, x2, sum, product;

      // Prompt for input
      cout << "Enter two integers:";
      cin >> x1 >> x2;

      // Call function to do the calculations
      getAnswers(x1, x2, sum, product);

      // Print results
      cout <<  x1 << " + " << x2 << " = " << sum < < endl;
      cout <<  x1 << " x " << x2 << " = " << product < < endl;
      return 0;
}

void getAnswers (int num1, int num2, int &total, int &prod)
// Function getAnswers calculates the sum and product of
// its first two parameters and places the answers
// in the third and fourth parameters
{
      total = num1 + num2;
      prod = num1 * num2;
}
```

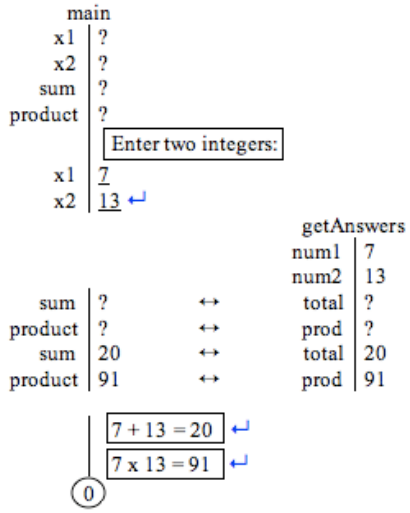


Figure 7

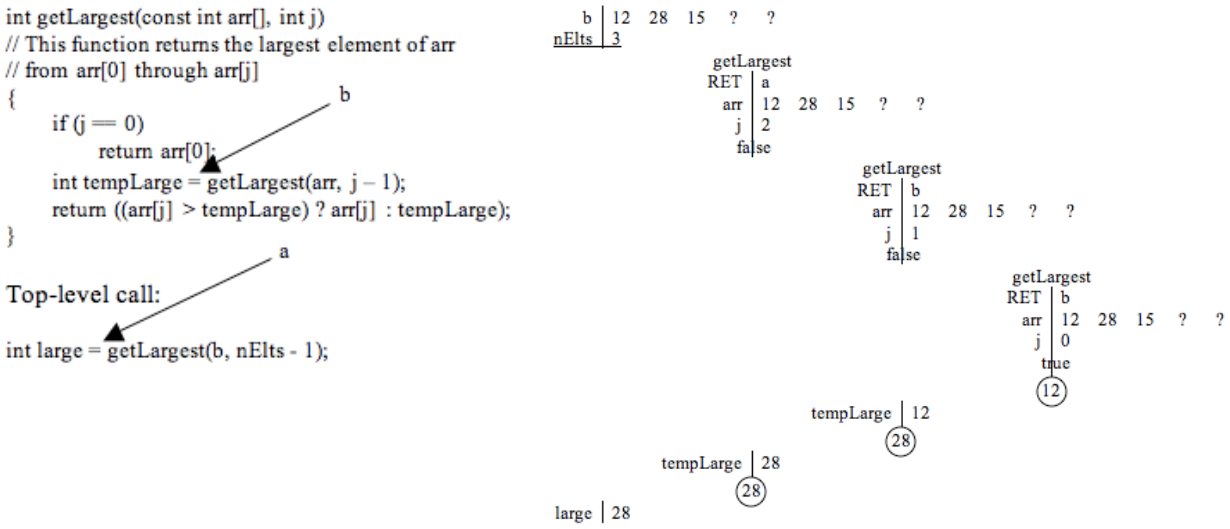


Figure 8

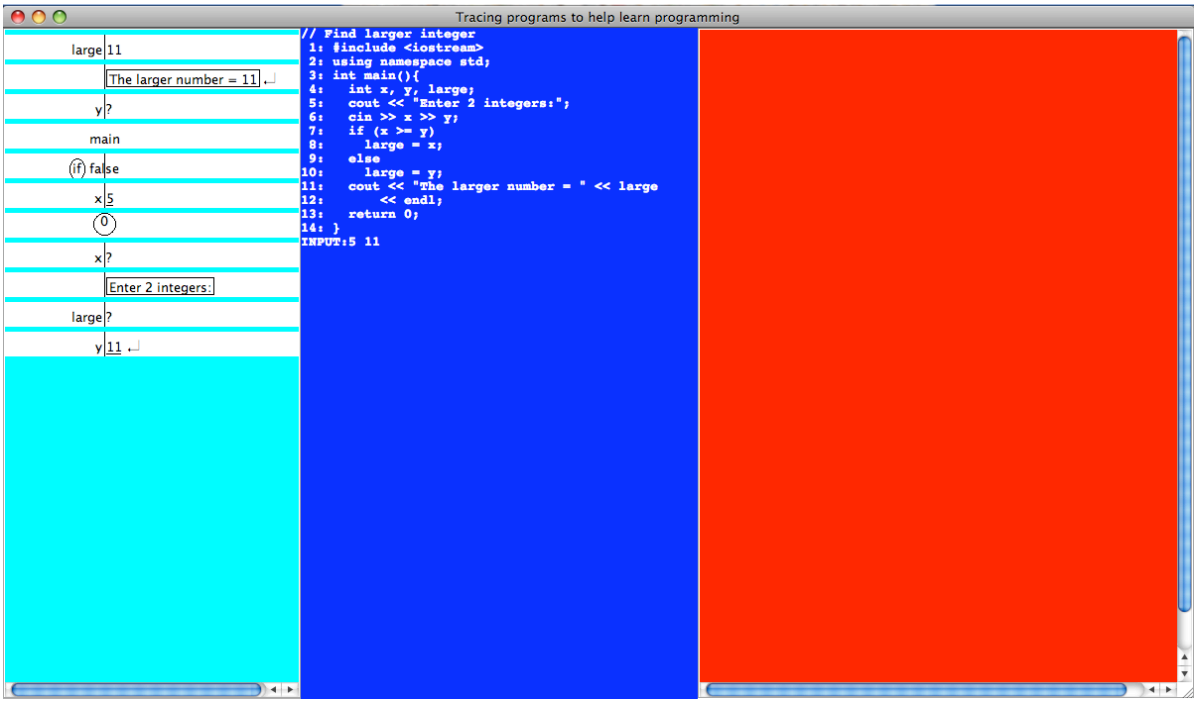


Figure 9

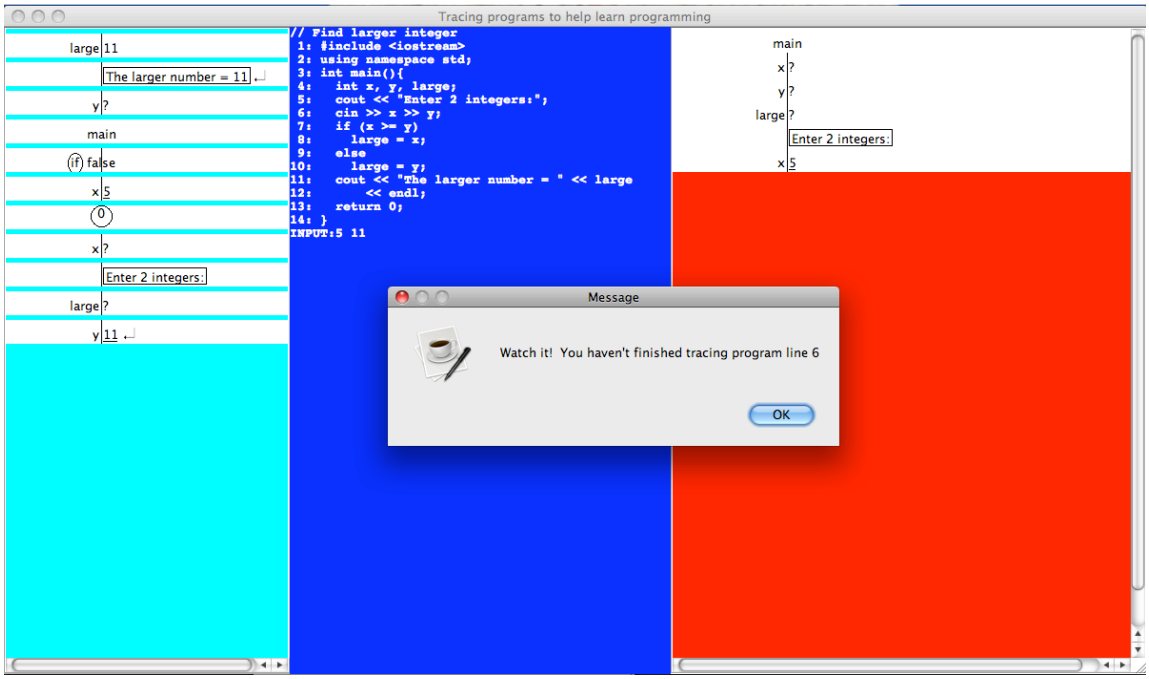


Figure 10

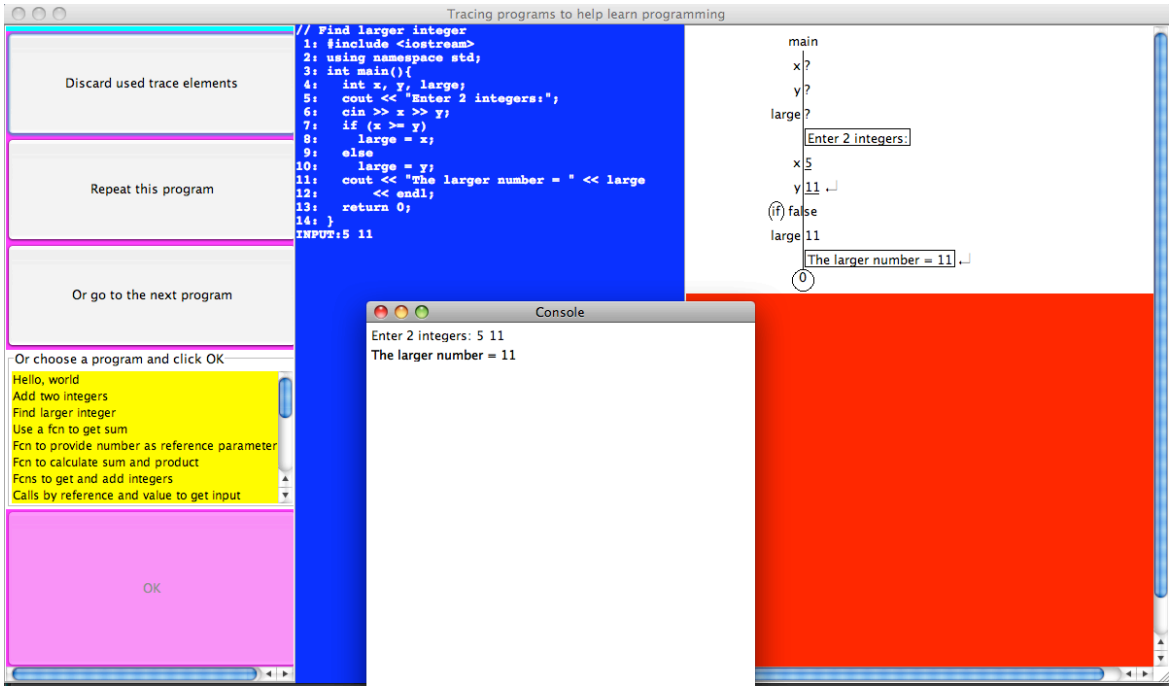


Figure 11

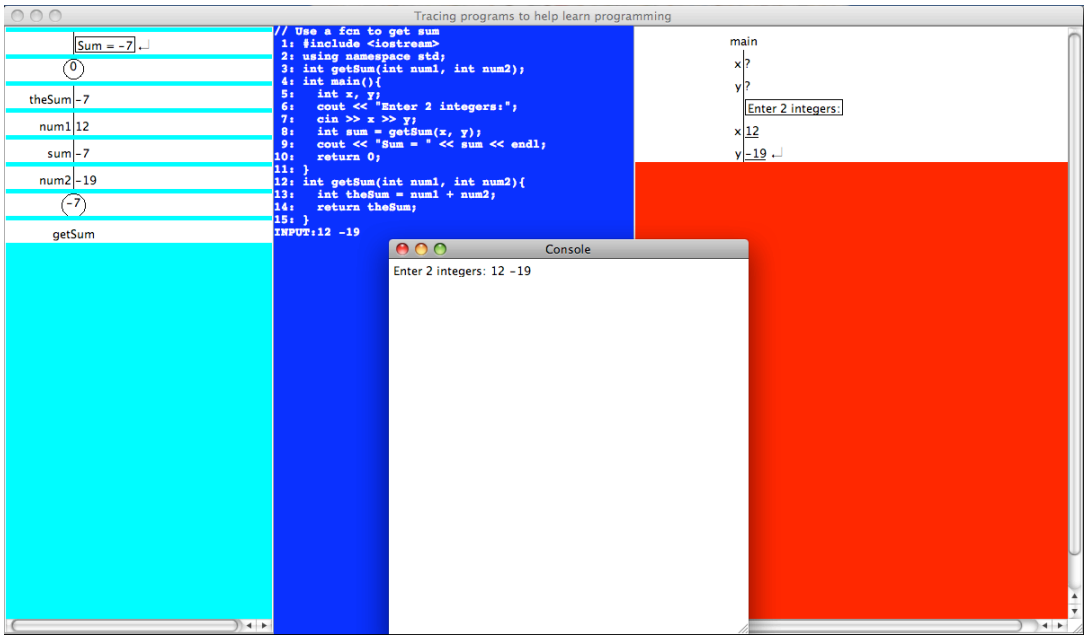


Figure 12

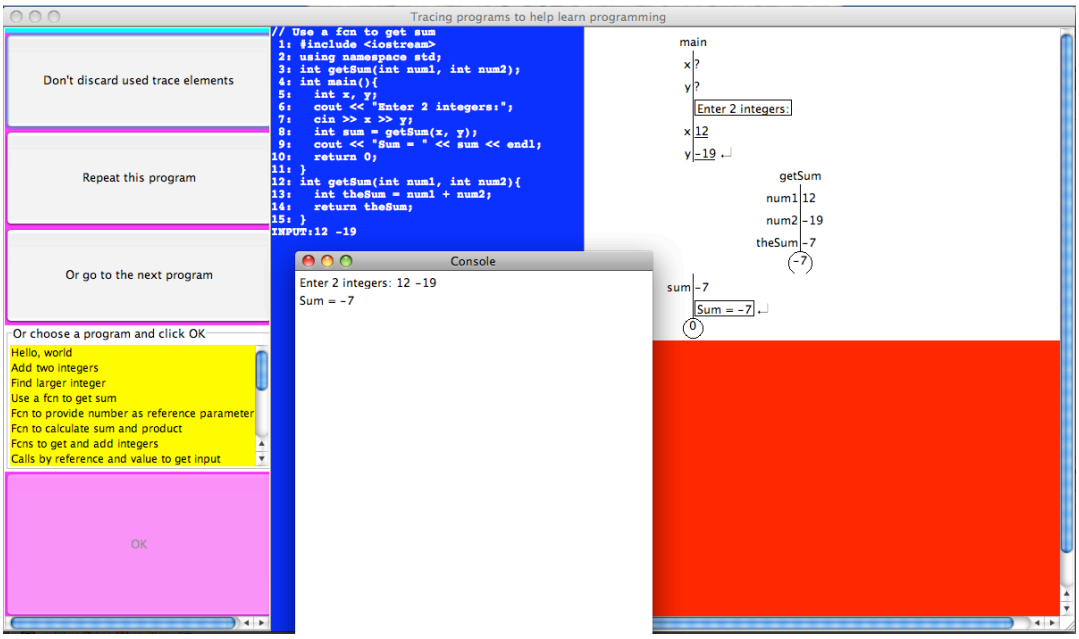


Figure 13

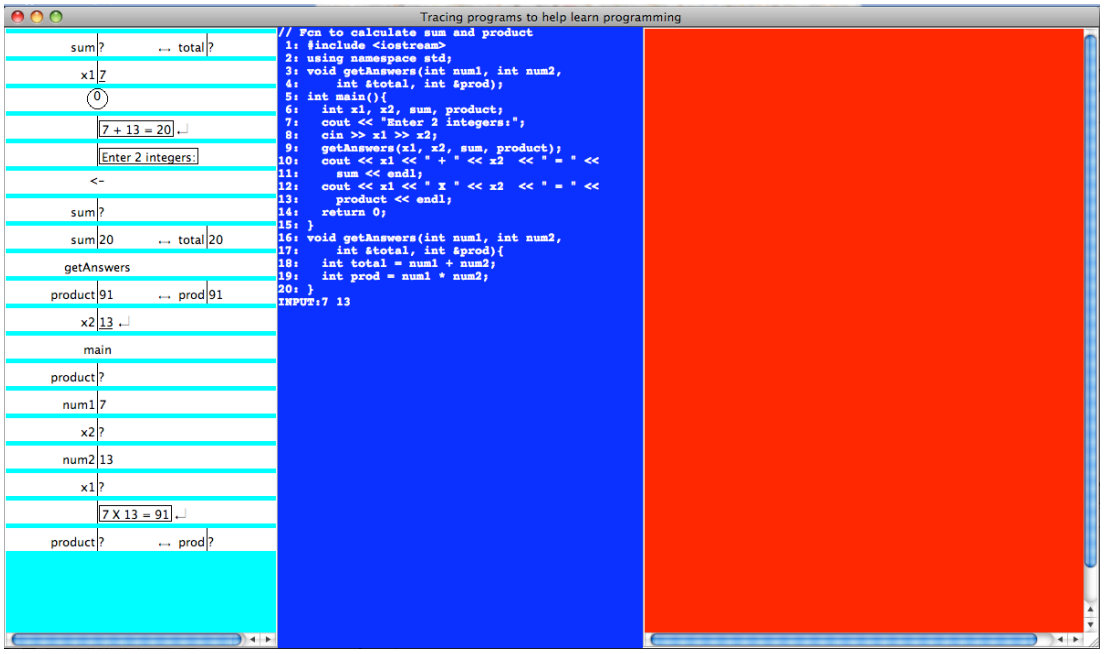


Figure 14

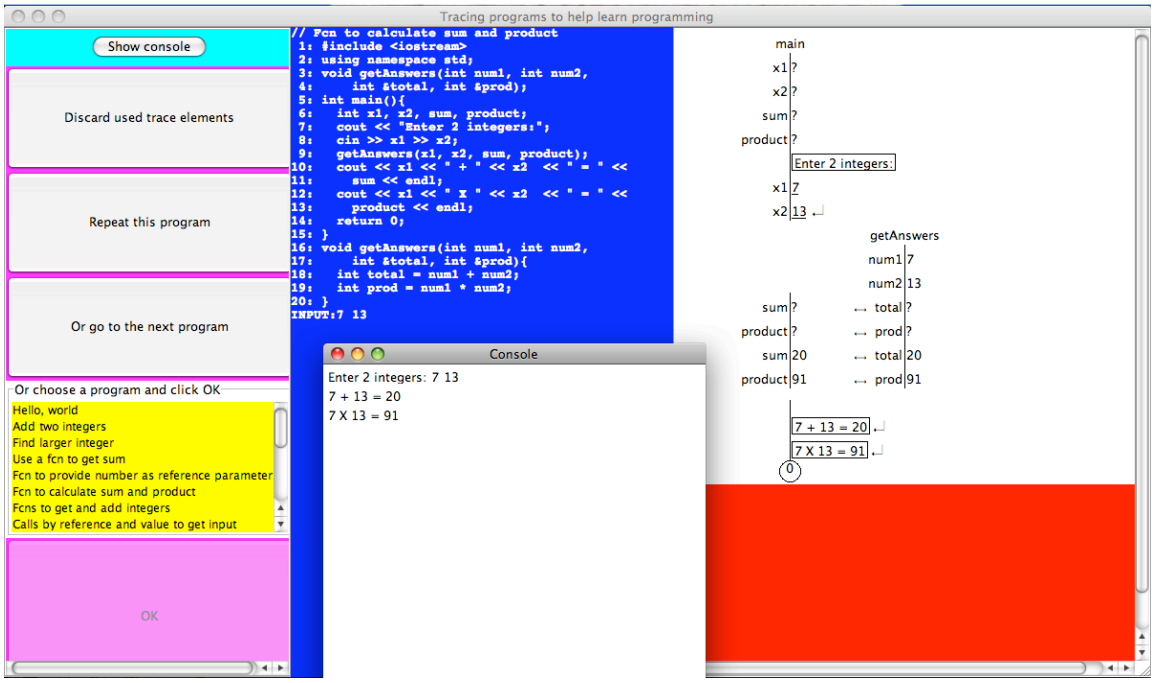


Figure 15