G isn't C!

LabVIEW and G as a Computing Language Course

Mukkai Krishnamoorthy and Sibylle Schupp,
{moorthy,schupp}@cs.rpi.edu
Department of Computer Science,
Rensselaer Polytechnic Institute,Troy,NY 12180**.**

Abstract

In this paper, we will describe our experiences in teaching an introductory computing languages course using the programming language G with the LabVIEW environment. This course is a one credit course (in a series of one credit courses like LISP, C++, and Java), for students who have already taken an introductory course in computer science. We will describe our experiences in covering graphics oriented programming, debugging aids and data-flow programming paradigms through the words of two typical students. The paper is organized in the form of a dialog between a computer science student and an engineering student.

Act One

A lab, somewhere on campus. At right, a desk. A laptop, scrap paper, empty soda cans. It is dark. Ms. Engi Neer, a sophomore, is staring at the screen of the laptop. Mr. Com Putist, a classmate of hers, rushes in.

Mr. Com Putist: What are you doing here so late?

Ms. Engi Neer: I'm working on my LabVIEW project. We have to calculate the *(stops shortly)* g.c.d. of two numbers. *(Sighs.)* LabVIEW is a cool language. But my instructor is weird. He gives us computer science projects, instead of engineering projects! You're a computer science major. Can you explain why my instructor is giving such projects?

Mr. Com Putist: Do you know what g.c.d. stands for?

Ms. Engi Neer shakes her head.

Mr. Com Putist: G.c.d. means: greatest common divisor. It's a pretty old algorithm, more than 2000 years. Some say it goes back to Euclid, but I'm not sure.[1]

Ms. Engi Neer: Algorithm? What's that? Gcd algorithm? What does an algorithm have to do with programming?

Mr. Com Putist: It's three things. For one, you have to make clear what the input of your program should be. But you also have to state what you expect to get back, the output of your program. Given specified inputs and specified outputs you can think of an algorithm as a set of rules for transforming your inputs into your outputs.

Mr. Com Putist: Hold on, I forgot: you also want to make sure that you actually get the output you're interested in; that is, that your program eventually terminates. So, I'd better say, an algorithm is a set of rules for transforming

[1] The student's doubts are right. There is evidence that Euclid's algorithm (c. 300 B.C.) was known up to 200 years earlier.

your specified inputs into your specified outputs in a finite number of steps.[2]

Ms. Engi Neer: Can you give me an example?

Mr. Com Putist: Why not take the g.c.d. algorithm? At first, we have to specify our input. It would be two positive numbers. *(Thinks for a second.)* Well, our calculus teacher might want us to call them integers, but you know what I mean.

Mr. Com Putist pulls over the laptop, opens an editor and enters the following text.

```
Input:      Two positive numbers, a and b.

Output:     A positive number c, such that c = g.c.d. (a,b)
            (i.e., the largest positive number that divides both a and b).

   Step 1:   If (a mod b = 0) go to Step 3, otherwise go  to Step 2.

   Step 2:  (a,b) = (b, a mod b); goto Step 1.

   Step 3:  Set c:=a; return ||
```

Figure 1: Greatest Common Divisor Algorithm

Ms. Engi Neer: This is cool. I remember my CS-1 instructor lecturing about algorithms in his class. I wish I had paid more attention to his class. Now that I understand the algorithm I can start my programming. Here is my front panel. *(Opens a new front panel, quickly enters two numeric controls and one numeric indicator.)* That's all my users get to know. They don't have to see the details of my program.

Mr. Com Putist: That is phat. Your front panel is the problem specification that we emphasize so much in computer science. One of our professors has preached to us that before writing our programs we have to write our specifications. In LabVIEW it comes for free. This is really nice. *(Pointing at the screen.)* One number looks different from the other two numbers. What's the difference? I also can see only pictures in your specification panel, um, front panel. That's pretty hip. But where's the code?

Ms. Engi Neer: You are asking too many questions. LabVIEW is a graphical language. This isn't like programming in C or C++ or Java. Both specification and connection diagrams are graphical.

Mr. Com Putist: Man, that's spiffy. Come on, do you draw pictures for comments too?

Ms. Engi Neer: Yeah. Yeah. We need to write text for comments. These comments mostly are labels for graphical symbols, sometimes somewhat longer descriptions. But comments are the only textual attribute. For everything else LabVIEW uses graphics attributes. Take, for example, our input. We specified two integers. How can I be sure that LabVIEW treats our input as integers and not, say, as real numbers? It's because of the color. Integers are represented by a blue color, real numbers are represented in an orange color. Coming back to your question, graphics attributes are also used to distinguish between input parameters---we call them controls---and output parameters or, in our language, indicators.

Mr. Com Putist: This is very interesting. It all makes sense. Now I want to know more. How to go about

[2] Apparently, he has studied D. Knuth, The Art of Programming, volumes 1 and 2.

implementing my algorithm?

Act Two

Mr. Com Putist and Ms. Engi Neer are jointly hunched over the laptop, Ms. Engi Neer is composing a block diagram and wiring objects. After a short while she leans back, satisfied.

Ms. Engi Neer: Here is my connection diagram.

Mr. Com Putist: I do not understand anything here. What are these wires from one block to another?

Ms. Engi Neer: This is in the spirit of circuit diagrams you see in electrical wiring diagrams. When two blocks are wired, that means data flows from one block to the other block.

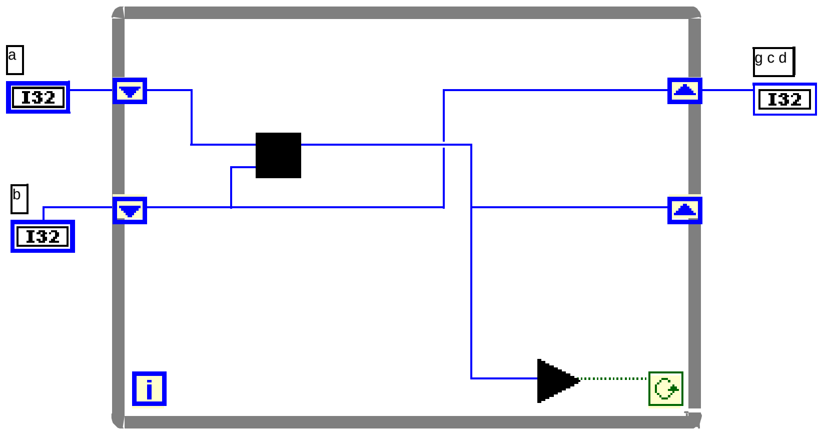

Figure 2: Greatest Common Divisor (g.c.d.) Connection Diagram

Mr. Com Putist: Aha! Your language is certainly weird. *(Laughing.)* No wonder a weird professor is teaching that course.

Ms. Engi Neer: Man, don’t talk bull of the language or my professor. You will soon appreciate good things in life.

Mr. Com Putist: *(Towards the screen, rather for himself.)* Hmm, my programming language professor taught me about procedural languages and data flow languages. I hated any abstract ideas then.

Ms. Engi Neer: Tell me about all those ideas!

Mr. Com Putist: You know the language C? Or C++ or Fortran? These languages are procedural (control flow) languages. Essentially, a program is a sequence of statements, one processed after the other. Data flow languages are different. In data flow languages, a statement executes as soon as it receives all its inputs.

Ms. Engi Neer: And?

Mr. Com Putist: Data flow programs can run in parallel; that is, two, maybe more statements or blocks can execute at the same time. With procedural languages, it’s really not so easy to get parallel programs.

Ms. Engi Neer: *(Interrupting.)* You mean to tell me that LabVIEW programs are graphical and based on data flow programming concepts?

Mr. Com Putist: Yep. Hats off to your language and your professor!

Ms. Engi Neer: Let me show you some more of my connection windows. Maybe we'll both learn something new.

Mr. Com Putist: That's good. Now I understand the wires. I even understand the color of these wires. I also notice that you have a "while" loop construct. LabVIEW seems to have the best of both worlds. Wait a minute. *(Remembers.)* I remember my programming language professor mentioning that there are no states in the programs.

Ms. Engi Neer: States of a program? I'm an engineer, I know the state of a flip-flop or a register---but hold on, we do have shift registers in LabVIEW. They do what you're asking for, yes. They maintain states.

Mr. Com Putist: Man! I'm truly flabbergasted.

Ms. Engi Neer: And I am impressed that you can use such complicated words like flabbergasted. You know that we scientists and engineers get a bad rap that we have a poor command over English.

Mr. Com Putist: I know, I know---that's all myth. We can do anything that we want and more than that; we can be the best in anything that we want to do. Coming back to states and shift registers, can you explain what you have done?

Ms. Engi Neer: Easy enough. I store the initial values of *a* and *b* in the incoming shift register. I compute new quantities for *a* and *b* (as you described in the three steps) and store the new value in the outgoing shift register. When the loop is executing for the second time, the outgoing shift register gets shifted into the incoming shift register. Now you understand why they are called shift registers.

Mr. Com Putist: Yeah. Shift registers are indeed nice abstract concepts. You can even write simple recursive programs using shift registers.

Ms. Engi Neer: Ah—the R word. Our CS-1 instructor mentioned something about recursion. You mean to tell me that I can compute Fibonacci numbers? (You know what a Fibonacci sequence is---just kidding---you computer science types always like the Fibonacci sequence.)

Mr. Com Putist: That is easy. *(A few mouse clicks later.)* Here is the diagram. Even though Fibonacci is not a tail recursive function you can compute it using shift registers. (Once you'll been enlightened you'll know about various kinds of recursion like head recursion and other kinds.)

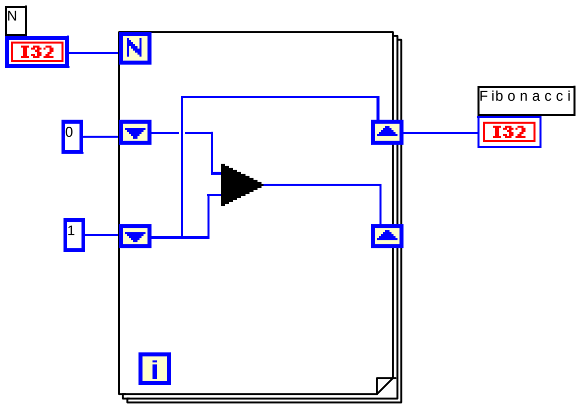

Figure 3: Fibonacci Connection Diagram

Ms. Engi Neer: Cool. You are good and fast.

Mr. Com Putist: Usually our programs are full of bugs and we spend most of our time debugging our implementation. I wonder how do you debug in LabVIEW.

Ms. Engi Neer: Glad you asked. The debugging aids are nifty. We can watch the execution as it proceeds. Also, we can place probes at various points to know its values. Debugging is graphical too.

Mr. Com Putist: Cool!! Truly you are blessed to have such a nice environement! Hmm. My programming language professor told us that if we understand the concepts then learning a new language is not that difficult. Anyway, I do enjoy the LabVIEW environment. It is interesting and challenging. Now, do you have the notion of procedural abstraction in LabVIEW?

Ms. Engi Neer: Pro what?

Mr. Com Putist: Give me a break. I'll go and get some food for us.

Act Three

Mr. Com Putist and Ms. Engi Neer sitting at the desk. Each with a huge bag of potato chips.

Ms. Engi Neer: You asked me whether we have procedural abstraction in LabVIEW?

Mr. Com Putist: Right. Is it possible to write subroutines or functions? That's what really makes a programming language powerful.

Ms. Engi Neer: Certainly it is possible to write functions in LabVIEW. We call them SubVIs. Once a SubVI gets created, anyone can use it. Just as a black box. All a user has to know is which inputs the SubVI expects and what it returns. We can even provide a short description for a SubVI (or as you computer scientists might say, a functional specification).

Mr. Com Putist: Wait a minute. What does the "VI" in "SubVI" mean?

Ms. Engi Neer: Virtual instrument.

Mr. Com Putist: I probably could have guessed. Now, how do I create a VI?

Ms. Engi Neer: Mr. Com Putist! This you can answer on your own! There are textbooks, an online manual, an online tutorial---not the worst, if you ask me---don't they teach you how to study independently?

Mr. Com Putist: *(Embarrassed, but not too much.)* Well.

Ms. Engi Neer: Look it up! SubVIs save me a lot of time. The next time someone gives me two positive integers and wants me to compute their g.c.d., I'll just use the SubVI we did today. I have to implement it once and that's it.

Mr. Com Putist: That's how it should be, I agree. Now, what if they don't give you two positive integers but, say, two Gaussian integers? You know what I mean, this imaginary domain? There is a g.c.d. algorithm, too. If I had to explain …*(Waiting for encouragement. Ms. Engi Neer graciously nods.)*

Mr. Com Putist: I could use the very same algorithm we talked about earlier. It works exactly the same, for our positive numbers and for Gaussian integers . All I have to do is to replace "positive number" by "Gaussian integer." Can you plug in Gaussian integers in your SubVI?

Ms. Engi Neer: *(Shocked.[3])* No, I can't! I would have to start from scratch. I would have to build an entire new SubVI.

Mr. Com Putist: Too bad. LabVIEW does not support generic programming.

Ms. Engi Neer: Besides, I don't think we have these Gaussian integers. Overall, we don't have many types. We have real numbers, obviously, and Booleans. And strings, tables, *(slows down)* list and rings---which, frankly, I've never used---and, of course, arrays and clusters. I don't see how I could construct Gaussian integers using them.

Mr. Com Putist: Built-in data types only. *(Destroyed.[4])* No double "O" P either?!

Ms. Engi Neer: What do you mean? You're making me nervous. Are you saying that LabVIEW is not a real language since it isn't what you want it to be? G isn't C!

Mr. Com Putist: Good point, I'm just teasing you. I took programming languages! My professor was saying over and over again that there is room for more than one programming paradigm on this planet. And that a language can't be good if it doesn't restrict itself.[5] What would you say, what's the strength of LabVIEW?

Ms. Engi Neer, without saying a word, takes the laptop and goes to the back of the room. There are many machines, devices, measuring instruments. Ms. Engi Neer connects the laptop with a small box. She opens a network connection, reads in datalog files, imports pictures. With increasing speed she builds more and more programs. Mr. Com Putist watches her, fascinated.

Act Four

In the back of the lab, a printer and a plotter are running continuously. The network server is overloaded. Printouts all over. Ms. Engi Neer and Mr. Com Putist are discussing a huge connection diagram with 57 SubVIs, 11 iteration statements, 25 case statements, and 18 shift registers.

Mr. Com Putist: I forgot. What is the semantics of *(points to a section of the diagram)*?

Ms. Engi Neer: Hmm. I forgot too. I would have to see again the details of *(points to a smaller section within the first section)*. Wasn't it what we did over there? *(Wants to point, but cannot find the place.Keeps on searching.)* I wish there were less information on this diagram.

Mr. Com Putist: That's what you're saying now. I know you. One second later, you want to have the full picture and you would complain if the details were gone.

Ms. Engi Neer: That's the problem. I want both, the details and the whole program. Even worse, sometimes I am interested in particular structures only and I wouldn't mind at all if the rest simply won't be there. Well, I think, one can't have everything.

Mr. Com Putist: Maybe I can help you. There are tools out there to cope with large data sets. I know of tools for the presentation of graphs. You can select certain parts, rearrange them, and zoom them in or out. It's neat. And it

---

[3] The second author believes in generic programming.

[4] The first author preaches object-oriented programming.

[5] The converse statement does not hold true.

helps you to structure and visualize information.

Ms. Engi Neer: Sounds great. How do I install this tool?

Mr. Com Putist: It's not yet there. We could try to modify what is available. But I would have to understand what exactly your needs are.

Ms. Engi Neer: And I would have to understand what exactly these tools can do. Why don't we get together to develop our own tool?

Both: Let's get together to develop our own tool! Let's get together to develop our own tool!

Outside, the sun is rising. The lab becomes filled. A new day on campus. Mr. Com Putist and Ms. Engi Neer look at each other, exhausted but lucky.

Mr. Com Putist: Now that we know what to do, let's *(takes a breath, continues, jointly with Engi Neer)* go to sleep.

Curtain.

Acknowledgments

The authors like to thank National Instruments and in particular Jeff Kodosky and Ravi Marawar for their support. We also want to acknowledge the chairman of the computer science department, Professor Frank Luk, for introducing LabView on campus and our colleagues, Professors David Musser, Robert McNaughton, Madhav Prabhu, David Spooner for feedback, suggestions, and improvements. Further suggestions came from Joyce Pritchard and Terry Hayden. Raju Krishnamoorthy checked for authenticity. The first author also likes to thank his Teaching Assistants June Zhu and Adeel Malik for their help and the students of his class for their curiosities.

References

[1] D.Knuth, The Art of Programming, Volumes.1 and 2, Addison Wesley 1997.

[2] M. Krishnamoorthy, A. Suess, M. Onghena, F. Oxaal, and T. Spencer, "Improvements to GraphPack: A System to Manipulate Graphs and Digraphs, Computational Support for Discrete Mathematics, (Edited by N. Dean and G. Shannon), American Mathematical Society, pp. 279-296.

[3] R.H. Bishop. Learning with LabVIEW, Addison Wesley, 1998.

[4] C.T. Wu and T.A. Norman, An Introduction to Programming: An Object-Oriented Approach with C++, McGraw-Hill, 1998.

M. Krishnamoorthy taught *Programming in LabVIEW* (Fall 1998, Spring 1999). S. Schupp used LabVIEW in *Programming Languages* (Spring 1999).

.