AC 2010-1840: WORK IN PROGRESS: ADOPTION OF CCS0 COMPUTATIONAL METHODS AND CIRCUIT ANALYSIS TECHNIQUES INTO AN INTRODUCTORY PROGRAMMING COURSE FOR ELECTRICAL ENGINEERS

Virgilio Gonzalez, University of Texas, El Paso

Eric Freudenthal, University of Texas, El Paso

Work in progress: Adoption of CCS0 Computational Methods and Circuit Analysis Techniques into an Introductory Programming Course for Electrical Engineers

Abstract

We report on the content and early evaluation of a pilot for a revised introductory programming course for ECE students titled “Software Design I, modified.” (SDIm.) SDIm incorporates pedagogical components from a course developed by our computer science department (CCS0) combined with an introduction to electric circuits and other ECE topics. SDIm is being developed in response to observations from several ECE faculty that many students, who attended the previously-offered courses in introductory C-programming and in computer organization, had struggled with minor programming assignments throughout the ECE curriculum. They also reported that fewer than 20% of students demonstrated mastery of programming in later senior courses.

The CCS0 course employs a simple interpreted programming environment based on “Python.” It uses simple small programs associated with mathematical and physical applications in order to illustrate the concepts of programming techniques. This intervention is based on the hypothesis that students will more quickly learn the fundamentals of programming using CCS0’s pedagogical model and programming environment than with a conventional course in C, and that they will effectively transfer these understandings to the study of C during the second half of the same course. Furthermore, SDIm’s inclusion of projects that examine the dynamic behavior of simple RLC circuits will reinforce key concepts taught in foundational ECE courses.

Introduction and motivation

The University of Texas at El Paso (UTEP) offers bachelor programs in several engineering disciplines and in Computer Science. One problem reported by many faculty members is the limited programming skills that the students acquire through the degree plans. This is more crucial to the Electrical Engineering area where we proposed this intervention. There are several factors that negatively affect our students, including the methodologies used to teach computer languages. The Computer Science department developed an introductory course in programming titled “Computational CS-Zero” (CCS0) or also called “Introductory Computational Systems” ICS[5] used in the entering program[4] that has shown its effectiveness in the past[8, 9, 10]. Therefore, we proposed the modification of our engineering course incorporating some modules from CCS0 and adding more relevance by applying the assignments to the simulation of electric circuits or other physical systems.

Our school is considered a Minority Serving Institution. One characteristic of the entering students is that the majority have not been exposed to any programming skill before attending college. The problem is compounded by the limited number of credits the student must take on the subject. The State Legislature has imposed a limit on the number of credits for an undergraduate program. Consequently, the EE degree plan can only afford to include one mandatory software course. Students that select the Computer Engineering concentration take additional programming courses but the other concentrations rarely do. Therefore, the skill level for most students is not enough to work later on many projects.

The department of Computer Science offers an introductory course with the objective to assist students in developing the skills necessary to succeed in the STEM areas. CCS0's activities are designed to provide analytical challenges typical of STEM professions and to motivate additional inquiry. It exploits programmed systems' lenience at manipulating computation to provide students with a review of foundational mathematical concepts in the context of graphical manipulation such as such as the use of nested for-range statements to enumerate the coordinates of pixels within geometric objects. For the new course we modified the context of the programs to associate them with electric circuit topics and focus the content to the EE students. Previous efforts have suggested to start teaching the programming fundamentals with interpreted environments, such as MATLAB® [11] , Infinity Project[14, 13], and the emphasis on practice in a relevant context[12]. Other discussions about the best beginner computer language show the advantages of using Python over other programming environments[15].

Pedagogical approach

Activities and projects of the prior introductory computing course generally focus on the outcomes of tasks whose programming challenges are frequently more clerical than analytical. While the outcomes of these projects are important, we have concern that the technical tasks have little relevance to engineering applications, causing the students to fail to understand the importance of programming to their intended major. In retrospect, the computer science department's recently developed CCS0 which successfully engages Freshmen in the programming of analytically focused problems. The original programming interface used a rich object oriented (OO) Java AWT toolbox[1]. With this approach, even the design of extremely simple algorithms requires fairly complex access code before anything can be programmed. The assignments examine the mathematics of dynamic systems of relevance to the engineering by applying numerical solutions[7] without overwhelming the students. Some exercises mimic the familiar phenomenon of ball bounce and spring resonance, which are frequently poorly understood, even by students who have completed a semester of college physics[6] . New assignments include programs that simulate and plot voltage and current of an LC oscillator. Students in CCS0 program in "Jython", a variant of Python. It is an easily learned interpreted programming language with expressive syntax.

In order to permit students to focus on analytical tasks and algorithm design, typical programming projects in CCS0 or SDIm are surprisingly short – typically four to ten lines of Python code that include numerical iteration controlled by for-range statements. In order to provide visceral understandings of program behavior, output is generally graphical, involving the direct manipulation of pixels within an RGB image addressed using Cartesian coordinates. Subsequent labs first examine the plotting of simple mathematical functions, which are later extended to explore the simulation and mathematical simulation of familiar physical phenomena such as ballistics and resonance. Previous work describes the rationale and initial modules introducing the programming environment.

A common tool used in the analysis of physical systems is the simulation using numerical methods and we commonly teach the methods in courses at the junior or senior level. This approach assumes the student knows the analytical methods for modeling of a system, such as an electric circuit or a mechanical device. With this new course, we present a simple model for the devices and simulate the dynamic behavior by using simple incremental changes. Students get simultaneously an insight on the physical system and apply simple programming methods.

Current course objectives and structure

EE 2372 is an introduction to software design with a structured computer language that focuses on the construction of programs consisting of multiple functions residing in multiple files. It covers program creation and top-down-design, basic elements and operations, modular program construction, and the use of programming tools such as make files. It introduces object oriented programming techniques. The prerequisites are: EE 1305 or CS 1401 with a minimum grade of "C" or better. Course topics are:

A. Introduction, and structure, compilation and execution of C program. (4 hours)
B. Variables, data types and arrays. (3 hours)
C. Operators and expressions. (4 hours)
D. Assignment statements and flow of control statements. (4 hours)
E. Input and output statements. (3 hours)
F. Function definitions and function calls. (6 hours)
G. Structure programming techniques and programming tools. (3 hours)
H. Pointer definition and use. (5 hours)
I. Derived data structures. (5 hours)
J. File I/O. (3 hours)
K. Introduction to C++. (2 hours)

Proposed new content modules

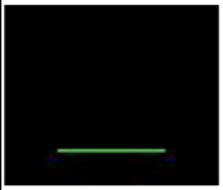
The introduction of new concepts will take place during the first period. That will enable the students to begin using programming techniques quickly and applying it to practical uses. The critical changes will be the first few weeks, and in preparing the right reference sheets so that students can gain momentum quickly. The later periods of the semester will cover the original material at a faster pace. The proposed new content is:

A. Introduction, installation of python environment (Jython) and interface. (1 hours)
B. Simple plots using arithmetic, iterations and "if-then" statements. (2 hours)
C. Use of summation and linear operations applied to image transformations. (2 hours)
D. Functions and example of class definition. (2 hours)
E. Application to simulation of simple mechanical systems. (2 hours)
F. Application to simulation of basic electric circuits. (3 hours)
G. Structure, compilation and execution of C program. (3 hours)
H. Variables, data types and arrays. (3 hours)
I. Operators and expressions. (2 hours)
J. Assignment statements and flow of control statements. (2 hours)

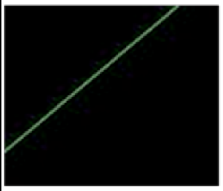
K. Input and output statements. (3 hours)
L. Function definitions and function calls. (3 hours)
M. Structure programming techniques and programming tools. (3 hours)
N. Pointer definition and use. (3 hours)
O. Derived data structures. (3 hours)
P. File I/O. (3 hours)
Q. Introduction to C++. (2 hours)

Examples of activities

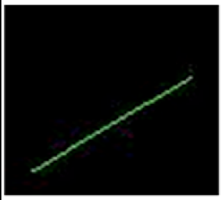
Section B, simple plots: This module is imported directly from the CCS0 course. The objective is to demonstrate the use of summation to draw sloped lines and the computation of slope. As illustrated by the top two images and programs from Figure 1, initial projects extend the concept of iteration introduced to draw horizontal lines the first module to draw sloped lines using summation. A row variable is initialized at column zero. A constant "step size" value is repeatedly summed into the row variable increasing its value at a linear rate that is graphically depicted. Students characterize the effect of various step sizes and initial row values. Later exercises lead students to derive the meaning of their generalization as slope and y-intercept. As illustrated by the image and program at the bottom of Figure 1, students are subsequently challenged to draw lines that connect designated points (say, to draw a geometric shape). To do this, students must derive the step size from the desired change in row and column using division. We observe that most attending the class can recall "y=mx+b" as an equation for a line, but nonetheless, initially have trouble computing step size; even math-phobic students enrolled in non-STEM programs and are visibly delighted when they derive an approach to determining step size (slope) using division.



```
pic = Raster((80,80))
...
row = 15
for col in range(0,pic.width):
  pic.setRGB((col, row), green)
```



```
row, step = 15, 0.5
for col in range(0,pic.width):
  pic.setRGB((col, row), green)
  row = row + 1
```



```
first, last = (10, 10), (70, 50)
run = last[0]-first[0]
rise = last[1]-first[1]
step, row = float(rise)/run, first[1]
for col in range(first[0], last[0]):
    pic.setRGB((col, row), green)
    row += step
```

Figure 1, Module 2 examples

Section D, Functions: Functions are split among two lectures, the first one can introduce the function abstraction, say in the definition of a function that will draw a line, that has a problem. For example, it can't deal with the parameters being in the wrong order (say, end is to the left of start) and not dealing with vertical lines well. A homework could be to: (1) write a function that deals with vertical lines well, and then (2) create another function that takes arguments in any order and handles vertical lines correctly by calling the correct helper function with arguments in the right order. The homework could then require them to use these functions to traverse a sufficiently complicated maze to require all these features.

The next lecture then will plot a line with negative **y**, which could be used to introduce them to PosNegGraph or CenteredGraph - and show them how it works. There is no need to have them write their own class, but instead just show them that classes+functions can be used to create powerful abstractions that they can subsequently exploit while ignoring the internal magic.

Section F, Application to basic electric circuits: After learning the basic concepts of iterations and graphic manipulation, the students started applying the programming techniques to simulate basic mechanical systems in the module 5, then module 6 expand the concepts to the solution of RLC circuits. In Figure 2 we show a program to solve a simple RC circuit with the given initial conditions. The student can get an insight on the exponential decay. Also the students can change the parameters and see the effect on time and amplitude of the responses. In the same module, a second order LC circuit shows how it behaves as a simple oscillator (Figure 3).

```
img = PosNegGraph((300,100))
R, C= 2e3 , 500e-6 # Ohms, Farads
V = 100.0  # Initial value
I = V/R   # Initial value
dt  = 10e-3  # Time step size
for x in range (0, img.width) :
  img.setRGB((x,1000*I),white) # Current in mA
  img.setRGB((x,V),green)
  dv=-(I/C)*dt  # Voltage step
  V= V + dv     # New voltage
  I= V / R      # New current
```

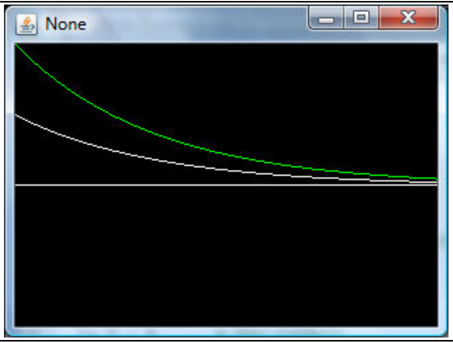


Figure 2, RC example

```
img = PosNegGraph((300,100))
V, I= 0 , 50.0e-3 # initial conditions
C,L = 10e-9, 1e-3 # Farad, Henry
dt  = 1e-7 # Time step size
for x in range (0, img.width) :
  img.setRGB((x,1000*I),white)# scale mA
  img.setRGB((x,V),green)
  dv=(I/C)*dt
  di=-(V/L)*dt
  V=V+dv
  I=I+di
```

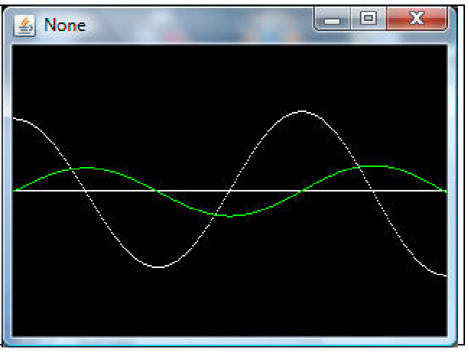


Figure 3, LC example

Evaluation and results

The course is being offered for the first time in the spring semester 2010. There are two sections offered in the semester, therefore we plan to use one as a control because the instructor will follow the original approach and in the other section we will introduce our modifications.

The pre-survey focused on the students' background experience and self-perception on their expertise level.

Pre-survey questions:

```
1. Classification
What is your current classification?
    a)  Senior
    b)  Junior
    c)  Sophomore
    d)  Freshmen

2. Previous Experience
Did you have any programming experience before graduating from High School (such as Basic, Logo,
C , robotics, etc)?
1. YES                2. NO

3. High school training
Did your High school offer formal programming training?

1. YES                2. NO
```

For all the remaining questions the responses range from 1 through 5, where 1 is no knowledge at all and 5 being an expert.

```
4. Arithmetic1
BEFORE taking this class, what was your knowledge level about computer arithmetic expressions?

5. Variables 1
BEFORE taking this class, what was your knowledge level about computer variables?

6. Compilation1
BEFORE taking this class, what was your knowledge level about program compilation, environment
and execution?

7. I/O 1
BEFORE taking this class, what was your knowledge level about instructions related to display,
graphics and data capture ?

8. Functions 1
BEFORE taking this class, what was your knowledge level about computer function definition and
use?

9. Pointers1
BEFORE taking this class, what was your knowledge level about computer pointers?

10. Structure 1
BEFORE taking this class, what was your knowledge level about computer programming structure and
techniques?

11. General1
BEFORE taking this class, what was your general knowledge level about computer programming?
```

The degree plan requires the students to take 128 credits in total and every academic year they need an average of 32 to move to the next level. 52% of the students reported they were classified as sophomores, 30% as juniors, 13% as seniors, and only 5% as freshmen. However most of the students already covered most of their Liberal Arts core and non-major courses before changing into Electrical Engineering. Additionally, only 22% reported previous programming exposure and for 78% this was the first time they have received formal training.

The students were asked about the previous knowledge about the different outcome areas for the class, including the use of variables, compilation, pointers, etc. Their responses are illustrated in Figure 4. As expected, most of them had just heard about the topics but did not know what they meant.

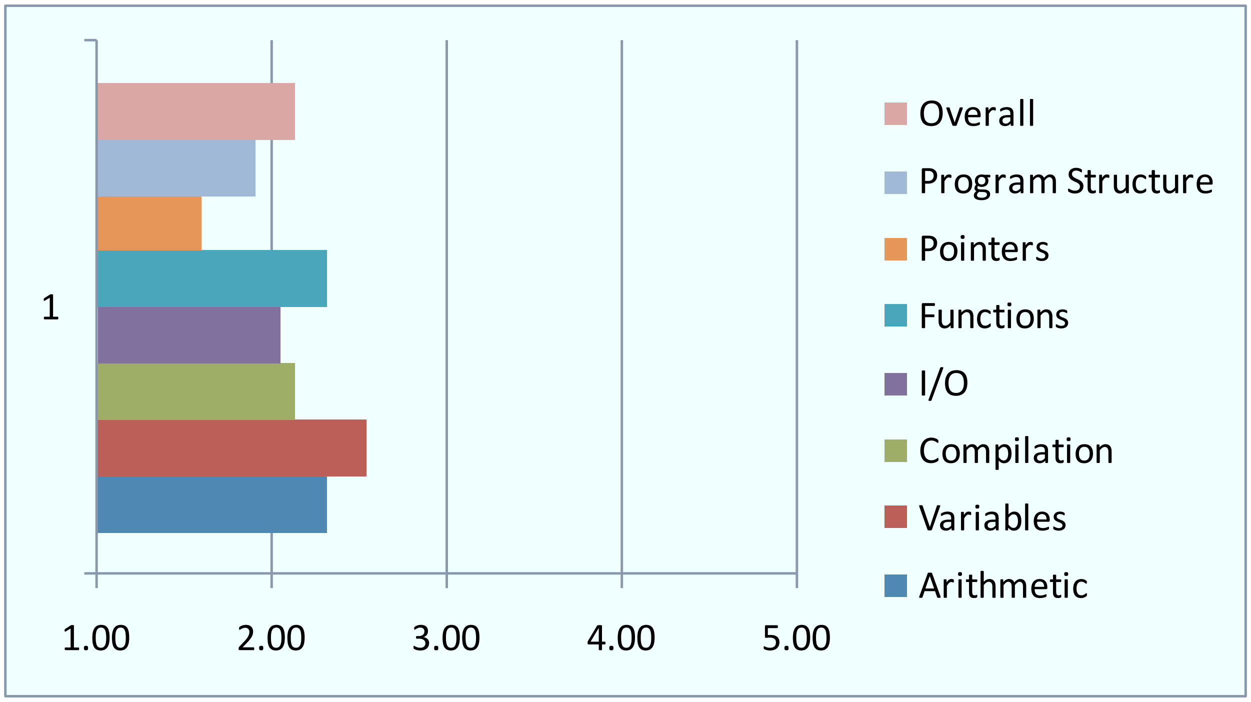


Figure 4, Level of knowledge about programming topics before taking the calass (1 min - 5 max)

We expect to have the post-survey graph at the end of the course showing the change during this intervention. It will be reported during the conference.

Conclusions

This work-in-progress project report describes a modified course designed to introduce students to computer programming including an intense hands-on introduction to Python, C and electric circuits. Continuing evaluation of introductory programming offerings at UTEP has motivated evolutions in curriculum, course objectives, and evaluation strategies. Interestingly, the resulting course, which engages students in "computational reasoning," integrates both programming and mathematics, and is engaging students with weak math skills. Results from early evaluation efforts are encouraging and have lead to adoptions into other areas. We anticipate that students who attend the Electrical Engineering Software Design I course will have a better understanding of the programming techniques and the possible applications. This complements the University's efforts to improve the learning of our students.

Acknowledgements

This report is based on work supported by the National Science Foundation through grants CNS-0540592, IIS-0829683, and DUE-0717877. Any opinions, findings, and conclusions or recommendations expressed in the paper are those of the authors and do not necessarily reflect the views of the NSF.

Bibliography

1. Guzdial, Computing and Programming with Python, a Multimedia Approach, Prentice Hall, 2006.
2. Guzdial, Design Process for a Non-Majors Computing Course, Proc.36th ACM Technical Symposium on Computer Science Education (SIGCSE), ACM, 2005.
3. Guzdial, Narrating Data Structures: The Role of Context in CS2, The Journal of Educational Resources in Computing (JERIC), ACM, 2008.
4. Eric Freudenthal, Mary K. Roy and Ann Q. Gates, Work in Progress – The Synergistic Integration of an Entering Students Program with an Engaging Introductory Course in Programming, Proc, Frontiers in Education, Fall, 2009..
5. Eric Freudenthal, Mary K. Roy, Alexandria Ogrey, Tanja Magoc, and Alan Siegel, A Computational Introduction to Computer Science, Proc. Annual Symposium of the Special Interest Group on Computer Science Education (ACM SIGCSE), 2010.
6. Hestenes, Wells, and Swackhamer, Force Concept Inventory, The Physics Teacher, Vol. 30, March 1992, pp 141-158.
7. Kalman, Elementary Mathematical Models, Mathematical Association of America (Press), 1997.
8. Siegel and Freudenthal, Experiments in teaching an engaging and demystifying introduction to algorithms: Installment 1: Huffman Codes, UTEP Computer Science Technical Report UTEP-CS-09-12, April 2009.
9. Thiry, Barker, and Hug, CAHSI Evaluation Progress Report, The Computing Alliance for Hispanic Serving Institutions, 2009, http://cahsi.cs.utep.edu/Portals/0/2008InterimEvaluationReport.pdf
10. Suskavcevic, Kosheleva, Gates, and Freudenthal, Preliminary Assessment of Attitudes towards Mathematics for a Non-STEM Section of Computational Computer Science Zero, UTEP CS Technical Report UTEP-CS-09-13, May 2009
11. Herniter, M.E., D.R. Scott, and R. Pangasa. Teaching programming skills with MATLAB. 2001. Albuquerque, NM, United states: American Society for Engineering Education.
12. Huet, I., et al. New challenges in teaching introductory programming courses: a case study. in Frontiers in Education, 2004. FIE 2004. 34th Annual. 2004.
13. Attia, J.O. Increasing electrical and computer engineering enrollment: A multi-faceted approach. 2007. Milwaukee, WI, United states: Institute of Electrical and Electronics Engineers Inc.
14. he Infinity Project: Engineering education for today's classroom. 2010 [cited 2010; Available from: http://www.infinity-project.org/.
15. Fish, S. Thoughts about the Best Introductory Language. 2006 [cited 2010 1/08/2010]; Available from: http://www.shlomifish.org/philosophy/computers/education/introductory-language/.