

# Rethinking the First Year Programming Course

William David Lubitz  
Assistant Professor, School of Engineering, University of Guelph  
wlubitz@uoguelph.ca

## Abstract

*The use of microcontrollers in beginning programming classes has been found to increase student learning and enthusiasm. Microcontroller programming has been successfully used in advanced undergraduate courses as a way to reinforce previous learning and to practice design, and in beginning undergraduate courses to introduce basic programming concepts in an engaging and relevant manner. Based on observations in a third year undergraduate course that included both traditional and microcontroller programming, a case is made for using a user-friendly microcontroller, instead of a traditional full-featured language, to teach fundamental programming concepts to novice engineering students.*

## 1. Background

Learning to write programs provides engineering students with practice in logical thinking and problem solving, as well as a useful tool for solving a wide range of engineering problems. Most engineering students take a course on programming in their first year, and learn the fundamentals of a "language" such as C, Fortran or Matlab. However, students often find classes on programming difficult and uninspiring. Typical criticisms include that lectures are difficult to follow, and assignments involve writing code to do an abstract task.

A survey of programming students and instructors [1] confirmed that the most difficult concepts for students to learn are the overall concepts of writing programs, as opposed to specific details of a language. While students can often learn the syntax of individual commands quickly, the skill of combining commands into a program is harder to learn [2].

One proposed solution is to change the programming language and environment that is used to teach beginning programming skills. There are

many options, including mini-languages [3], structured environments [4] and development tools for traditional languages [5].

An interesting possibility is incorporating a robotics component as a means of increasing student interest in programming. Success has been reported with robots that are simulated on a computer such as Karel [6,7]. In physical space, LEGO Mindstorms products, which are a user-friendly modular robotics system, have been used to teach both introductory robotics [8] and beginning programming [9, 10]. Use of the LEGO product has been reported to improve both student learning and enthusiasm [9].

LEGO robots have also been used in more advanced courses as a way to integrate and reinforce concepts from previous courses [11]. This paper describes a similar course that uses the BOE-Bot robot system (Parallax Inc.). Qualitative observations from this course are used as the basis for a proposed introductory programming course utilizing a microcontroller platform.

## 2. Requirements of an Introductory Programming Language

A programming language for teaching introductory programming must include all of the commands and structures needed for students to practice fundamental programming concepts:

- ⇒ Looping and decision structures
- ⇒ Different data and variable types
- ⇒ Input and output capabilities
- ⇒ Subroutines and functions

For a language that may be a student's first experience with programming, desirable attributes also include

- ⇒ Simple and straightforward syntax
- ⇒ Simple or no compiling
- ⇒ Good documentation and examples

⇒ Simple programmer interface

The proprietary programming languages used by some of the more “user-friendly” microcontrollers fit the above criteria well. In addition, they are implemented on a limited resource platform that encourages efficient use of processor time and memory. For example, PBASIC, the programming language used for the BASIC Stamp microcontroller that is used in the BOE-Bot, is modelled after the standard BASIC programming language, with some modifications:

- ⇒ The math command set is reduced due to limitations of memory and processing capability.
- ⇒ Additional commands for input and output from individual microcontroller pins are included.
- ⇒ Additional commands for writing data to and from memory are included.
- ⇒ Commands to print data to a screen (e.g. “print”) are replaced with a “debug” command that is used to send messages from the microcontroller to a PC terminal via an RS-232 or USB serial connection.

The syntax of the language is straightforward (e.g. no semicolons or curly brackets) and easy to read. For example, the following code checks a pushbutton on pin 2 of the microcontroller (“IF (IN2 = 1) THEN...”) and blinks an LED connected to pin 3 if the button is depressed. The state of the button is also transmitted to a PC via the serial connection (using the DEBUG command). It loops about once per second (or 1000 milliseconds).

```
' {$STAMP BS2}
' {$PBASIC 2.5}

DO
    DEBUG ? IN2

    IF (IN2 = 1) THEN
        HIGH 3
        PAUSE 500
        LOW 3
        PAUSE 500
    ELSE
        PAUSE 1000
    ENDIF
LOOP
```

Given that the capabilities (and complexity) of traditional languages far exceed the needs of beginning programmers, the use of a user-friendly microcontroller language for introductory programming should be explored. An initial programming assignment that involves controlling LEDs with switches is also more engaging than getting “Hello World!” to display on a screen.

### 3. Prior Experience: ME 175

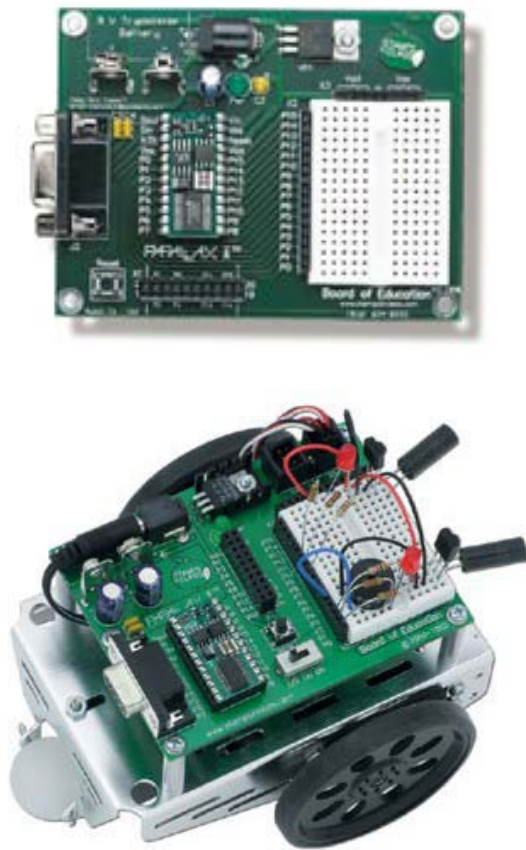
At California State University, Sacramento, a course titled ME 175 “Computer Applications in Mechanical Engineering” is a required course for third year Mechanical Engineering students in an ABET accredited four year program. Course prerequisites include an introductory programming course that used Matlab. Most students had also used Microsoft Excel spreadsheet software, primarily for plotting data in laboratory exercises. Typical course enrolment was 25 students.

The primary goal of ME 175 is to teach numerical methods, programming and introductory robotics using a curriculum built around three primary tools: the Matlab “programming language”, spreadsheet software (Microsoft Excel) and the “BOE-Bot”, a small autonomous wheeled robot built around a BASIC Stamp microcontroller (Fig. 1). PBASIC, the programming language used for the BASIC Stamp, is modelled after the standard BASIC programming language, with some modifications.

The author taught this course in the fall semesters of both 2002 and 2003. Student evaluation was based on laboratory and homework assignments, weekly quizzes, a midterm exam, and presentation and demonstration of a design project utilizing the BOE-Bot. Course evaluation consisted of a department-given student survey at semester end, and an anonymous survey at the midpoint of the course administered by the instructor for diagnostic purposes.

In addition to regular homework assignments, students complete microcontroller assignments and a design-based student-developed final project that involves programming the BASIC Stamp and configuring the BOE-Bot for a student-defined application. The course is very popular with students, especially the laboratory sessions and the final project.

A strong effort was made to immediately integrate lecture material (primarily on programming and numerical methods) to homework and laboratory assignments. A typical homework assignment involved implementing a solution in Excel and Matlab to a practical problem using a numerical method just discussed in lecture. Initial laboratory assignments were focussed on learning PBASIC and how to interface electronic sensors to the BASIC Stamp. Later assignments included programming the Stamp to perform tasks involving decision-making, input and output, sensing and math. Programming lectures attempted to teach overall programming concepts using tools such as pseudo-code and flow charts. Examples were given in Matlab, Excel and PBASIC whenever possible to reinforce common concepts.



**Figure 1. Above: A “Board of Education” (BOE), which includes a BASIC Stamp microcontroller, power conditioning and a breadboard. Below: A “BOE-Bot” robot. (Parallax Inc.)**

#### 4. Observations

The laboratory sessions were invaluable for determining student understanding of course material, and for observing student approaches to problem solving. During the course of interacting with students completing assignments and projects in the laboratory sessions, several things became apparent:

The use of a “conventional” programming language, spreadsheet software, and a user-friendly microcontroller programming language (Matlab, Excel and PBASIC respectively) in the same course appears to be fairly unique. It was observed that if students had difficulty mastering the basic principles of using one of these tools, they had similar problems with the others. For example, if students had difficulty writing a Matlab program to solve a problem, they usually also had difficulty setting up a spreadsheet solution.

Once a set of skills has been taught, it is important that those skills are practised regularly. While all students had taken an introductory programming course, it came a year before in the course schedule, and the students had not needed the programming skills in any subsequent courses. As a result, many students needed to relearn fundamental programming concepts.

While traditional programming languages include vast capabilities, they also bring additional complexities for beginning students. For example, students experienced some difficulties that were unique to Matlab, such as inadvertently defining an array of numbers as a row vector instead of a column vector, and having math operations on that array/vector fail. With students still learning basic concepts, such as iterating through an array of values, this added complexity to problems that did not contribute to learning fundamental knowledge.

There were also several complaints that some Matlab programming assignments (e.g. write a program to solve an ordinary differential equation using the Runge-Kutta algorithm) were “busy work” or “pointless” because Matlab contains pre-written functions to do these operations.

A majority of students in the course were not comfortable writing programs, and a few students went to considerable lengths to avoid having to type in program code during laboratory exercises, instead “cutting and pasting” text from online example programs. This resulted in a few students having repeated problems with their programs that could have been fixed by making minor changes, such as changing a number in the code, but the students were reluctant to make these changes.

When debugging programs, following the order of execution of a program (i.e., “reading the code”) was very difficult for many students. Interestingly, Winslow [2] reported in 1996 that novice programmers typically take a “line-by-line” approach to programming, however this was not the experience in ME 175 in 2002/3. A possible explanation is that while younger students tend to have spent considerable time using computers, it has almost always been with GUIs, instead of the “command line” environment of earlier generations. This makes it more difficult to transition to text-based programming in which commands are executed sequentially. These students may not be as comfortable interacting with a computer in a “text-based” or “command line” manner as older individuals whose initial exposure to computers predates GUI operating systems. Also, younger students often expressed a belief that text-based programming is not state-of-the-art, and therefore, less useful for them to learn.

In general, students experiencing difficulty with programming were often overwhelmed by the apparent complexity of the programming language, and found that the methods of interaction with the computer were very different from their previous computer experience. Also, many students seemed to have retained few skills from their prior (“traditional”) programming course, and generally felt that it had “not been [a] worthwhile” class.

## 5. Proposed Introductory Programming Course

Based on these observations, a case will be made here for using a new approach to teach introductory programming to engineering students.

Instead of using a full-featured “traditional” programming language, the introductory programming course is structured around a microcontroller with a relatively “user-friendly” instruction set, such as the BASIC Stamp produced by Parallax Inc. The microcontroller used should have an instruction set that includes simplified mathematics, logical comparisons, loops and subroutines. It should also be easy to interface simple devices (e.g. switches for inputs, LEDs for outputs) for students who have not yet taken a circuits course. For teaching programming, the input devices (e.g. switches) and output devices (e.g. LEDs, small speakers) would be pre-assembled in a standard pattern on all of the microcontrollers. As student skills increased, they would be able to add additional hardware to extend the capabilities of their microcontroller.

Course lectures would follow the topics and sequence of a traditional programming course, except programs would be written and tested using the microcontroller. Lectures would be fully integrated with laboratory sessions so that students could apply the lecture material immediately.

Fundamental concepts of computing like managing input and output, binary math, memory usage and the need for efficient algorithms would be much more relevant to students than if they were programming on a PC. Additionally, the students would learn to use a practical tool (the microcontroller) that could be used in future projects or courses, and completing microcontroller projects would give the students a more engaging experience than traditional programming assignments.

A “traditional language” such as C, Matlab or Fortran would be introduced in the latter part of the course when the limitations of the microcontroller’s language became apparent to the students. The microcontroller would then be configured as a basic

data acquisition system, and the traditional programming language would be used to analyze the data. The programming skills developed using the microcontroller would be transferred to the new traditional language, while the students learned skills such as numerical solutions and file handling that do not translate as readily to the microcontroller.

Ideally, a second course built around the traditional language and numerical methods would follow. Labs for this course would use the microcontroller as a data collection device, with the data being analyzed using the more powerful traditional language. Programming skills should also be used regularly in design or problem-solving exercises throughout the undergraduate curriculum to reinforce the skills learned in dedicated programming classes. This is especially true for non-computer science students, who only (or never) realize how useful their programming skills could be after they have become rusty.

## 6. Conclusions

Microcontroller programming has been successfully used in advanced undergraduate courses to reinforce previously learned programming skills, as well as in beginning undergraduate courses to introduce basic programming concepts. The use of robotics and microcontrollers to teach fundamental programming concepts can increase student enthusiasm for a subject that novice students often find abstract and difficult.

Observations of a third year undergraduate course that included traditional and microcontroller programming, as well as a robotics design component, revealed that students have greater difficulty with general programming concepts, such as planning an algorithm to solve a problem, than they do with learning the syntax and details of a specific language.

A case is made for using a user-friendly microcontroller, instead of a traditional full-featured language, as the chief platform to teach fundamental programming concepts to novice engineering students. The microcontroller would be pre-configured with inputs (e.g. buttons) and outputs (e.g. LEDs), and students would learn fundamental concepts such as algorithm design, variables and data storage, input and output, and efficient program design and memory usage, by writing programs for the microcontroller. The students would transition to a traditional programming language as their skills developed and limitations of the microcontroller began to become evident. However, the microcontroller would continue to be used as a data collection device.

## 7. References

- [1] Lahtinen, E., Ala-Mutka, K., Jarvinen, H-M. A Study of the Difficulties of Novice Programmers. ACM SIGCSE Bulletin, Vol. 37, Iss. 3, 2005.
- [2] Winslow, L. Programming Pedagogy – A Psychological Overview. SIGCSE Bulletin, Vol. 28, Iss. 3. 1996.
- [3] Brusilovsky, P. et al. Mini-Languages: a Way to Learn Programming Principles. Education and Information Technologies. Vol. 2, pp. 65-83. 1997.
- [4] Powers, K. Through the Looking Glass: Teaching CS0 with Alice. SIGCSE Bulletin, Vol. 39, No. 1, pp. 213-217. 2007.
- [5] Gómez-Albarrán, M. The Teaching and Learning of Programming: A Survey of Supporting Software Tools. Computer Journal. Vol. 48, No. 2, pp. 130-144. 2005.
- [6] Becker, B. Teaching CS1 with Karel the Robot in Java. SIGCSE Bulletin, Vol. 33, No. 1, pp. 50-54. 2001.
- [7] Xinogalos, S., Satratzemi, M., Dagdilelis, V. An Introduction to Object-Oriented Programming with a Didactic Microworld: objectKarel. Computers & Education, Vol. 47, pp. 148-171. 2006.
- [8] Lau, K., Tan, H, Erwin, B., Petrovic, P. Creative Learning in School with LEGO® Programmable Robotics Products. 29<sup>th</sup> ASEE/IEEE Frontiers in Education Conference. San Juan, Puerto Rico. Nov. 10-13, 1999.
- [9] Andrews, W. The Qualitative Impact of Using LEGO MINDSTORMS Robots to Teach Computer Engineering. IEEE Transactions on Education, Vol. 46, No. 1. 2003.
- [10] Barnes, D. Teaching Introductory Java Through LEGO MINDSTORMS models. ACM SIGCSE Bulletin, Vol. 34, Iss. 1, 2002.
- [11] Garcia, M., Patterson Mc-Neill, H. Learning How to Develop Software Using the Toy LEGO MINDSTORMS. ACM SIGCSE Bulletin, Vol. 34, Iss. 3, 2002.