EFFECTIVELY TEACHING C IN AN INTRODUCTORY EMBEDDED MICROCONTROLLER COURSE

Jeffrey J. Richardson

Purdue University, West Lafayette, Indiana; Email: jricha14@purdue.edu

ABSTRACT

This paper details an effective approach to teaching the C programming language in an introductory microcontroller course. For the past several years, the Electrical and Computer Engineering Technology department at Purdue University has been teaching the C programming language as the language of choice for the introduction to microcontrollers course. Teaching a high-level language (HLL) in an introductory microcontroller course in preference to an assembly level language is a contrast to institutions with similar programs. The use of a HLL allows the students to concentrate on the material and concepts and not get bogged down in the cryptic details of how to implement a solution in assembly. The difference is that a HLL allows values to be written *directly* into the register instead of moving data through multiple operations with assembly. Just-in-time (JIT) teaching methods are used to introduce new components of the C programming language as needed. The lab experiments are carefully designed to spread out the concepts of the language over the entire course. After a problem is presented, the students are taught *what* must be done to solve the problem and *why*? Once an algorithm has been established and evaluated, the algorithm is drawn graphically in the form of a flowchart. The flowchart is then *easily* translated into the C code required to solve the given problem. A JIT teaching method allows the C language to be taught indirectly and keeps the focus of the course on the true subject matter: microcontroller fundamentals. The techniques outlined in this paper relegate the programming language to a simple tool to implement a solution.

1. INTRODUCTION

Over the last ten years, hundreds of millions of microcontrollers, a microcomputer that contains most of its peripherals and memory inside a single integrated circuit along with the CPU, central processing unit, have been incorporated into products ranging from keyboards to automobile control systems (Ayala, 2000). Originally they were used only for expensive industrial-control applications, but as technology brought down the cost of dedicated microcontrollers, they began to appear in moderately expensive applications such as automobiles, communications and office equipment, and televisions. Today's embedded systems are so inexpensive that they are used in almost every electronic product in our life (Lewis, 2002).

High-Level Languages are rapidly becoming the standard methodology for embedded microcontrollers due to improved time-to-market and simplified maintenance support (Myklebust, N.D.). To keep pace with the changing times and technology, a high-level programming language is now being utilized in the introductory embedded microcontroller course taught as part of the Electrical and Computer Engineering Technology curriculum at Purdue University, West Lafayette.

2. HIGH-LEVEL PROGRAMMING ADVANTAGES

It was the belief in the early days of embedded microcontrollers that the assembly language was the only choice for producing code for applications. The extremely limited on-chip space for data and program code did not appear to be an adequate place for anything developed using a high-level language (Stewart & Miao, 1999). High-level languages (HLL's) historically produce larger code sizes than assembly programs which results in a reduced speed of execution. During the past several years, compilers have emerged on the market that claim to produce code as efficient as assembly (Stewart & Miao, 1999). In addition to more efficient compilers, advancing technology surrounding microcontrollers continues to provide greater amounts of functionality and speed. These increases in technology have led to the almost universal use of high-level languages to program even time-critical tasks that once require assembly language programs (Barnett, Cox, & O'Cull, 2003).

There are several advantages in using HLL's instead of the assembly language when developing microcontroller applications. These advantages include reduced development time, easier maintainability and portability, and easier reuse of code. High-level languages allow programmers to deal with complex objects without worrying about details of the particular processor on which the program is running (Darnell & Margolis, 1991). Writing HLL programs frees the programmer from having to worry about the low-level details of a program (Reisdorph, 1998).

C is a general-purpose high-level programming language, developed without a specific processor in mind and quickly became the language of choice for programmers due to the advantages it has over other high-level languages (Stewart & Miao, 1999).

3. C PROGRAMMING AND EMBEDDED MICROCONTROLLERS

Software development with embedded controllers requires a structured approach to programming. Many embedded systems have to run 24 hours a day, seven days a week, 365 days a year. They cannot be rebooted when something goes wrong! For this reason, good coding practices and thorough testing take on a new level of importance in the realm of embedded microcontrollers (Lewis, 2002). The C language facilitates a structured and disciplined approach to computer program design (Deitel & Deitel, 1992). According to MacKenzie, the advantages of adopting a structured approach to programming include the following: a program's sequence of operations is simple to trace and thus facilitates debugging, structures lend themselves easily to building subroutines or functions, structures are self-

documenting and are easy to describe in flowcharts, and structured programming results in increased programmer productivity (1991).

Perhaps the most compelling reason for using any high-level language is to save programmer time. C compilers are supplied with a library of C programs that perform many common math and character-handling tasks. Using these standard-function C programs relieves the programmer of having to write, test, and debug equivalent versions (Ayala, 2000). Although the standard library functions are technically not a part of the C language, they are invariably provided with ANSI C systems (Deitel & Deitel, 1992).

4. EFFECTIVELY TEACHING THE C LANGUAGE

The C programming language is finding its way into more and more curricula and schools every day (Antonakos & Mansfield, 2001). The ECET curriculum does not require a C programming class prior to the introductory microcontroller course. Therefore, the instructor must teach the C language, as it pertains to embedded microcontrollers, as well as microcontroller fundamentals. Teaching the C programming language, or any language, will reduce the amount of lecture time that is devoted to the actual microcontroller fundamentals. Utilizing the following approach can significantly reduce the lecture time required to deliver the programming concepts.

Solving problems using an embedded microcontroller and the C programming language is the terminal objective of the course. To this end, teaching problem solving techniques is a must. It has been the experience of the ECET department at Purdue that algorithm development is the number one challenge faced by the students. To assist the students in learning to solve problems by generating algorithms, it is necessary to work through fully illustrated examples early in the course. Initially, the students are spectators in the algorithm development process but quickly become active participants. Generating algorithms is an essential first step in writing software.

Once an algorithm has been developed, a flowchart is generated. Flowcharts are graphical representations of algorithms and are a vital visual tool in translating the algorithm into software. Psuedo code is also a useful tool for translating algorithms into software, but flowcharts are preferred during the initial stages of the course. When drawn in the proper form, flowcharts allow the algorithm to be easily converted into the corresponding software. Flowcharts also allow the problem to be described in terms of "what must be done" rather than "how it is to be done" and help the student to "think out" a program (Deitel & Deitel, 1992).

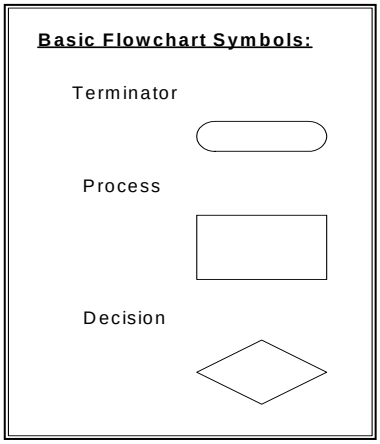

Figure 1 – Basic Flowchart.

Flowcharts are drawn using three special symbols: ovals, diamonds and rectangles (see figure 1). Rectangles are used to indicate any process that must be performed. Decisions are represented with diamonds and ovals are used to indicate the starting and stopping points of the algorithm. Lines with arrowheads, commonly known as flow lines, indicate the order in which the processes and decisions are to be performed.

According to MacKenzie, all programming problems can be solved using three structures: statements, loops, and choices (1991). The completeness of the three structures seems unlikely; but, with the addition of nesting, structures within structures, it is easily demonstrated that any programming problem can be solved using only three structures (MacKenzie, 1991). Statements provide the basic mechanism to do a process. The "loop" is used to repeatedly perform an operation and the "choice" is the programmers "fork in the road" both of which are decisions (MacKenzie, 1991). The C structures represented by the flowchart symbols described earlier, are easily identifiable as seen in the figures 2 and 3.

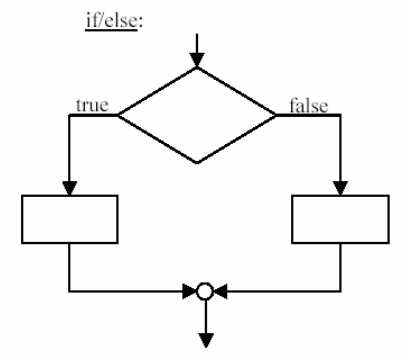

Figure 2 – IF/ELSE.

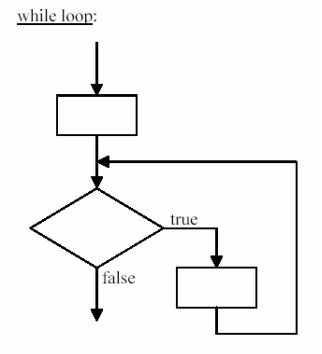

Figure 3 – While Loop.

Drawing a flowchart in the proper form, to be easily translated into C code, may take several iterations. The following rules should be applied when creating flowcharts. The initial flowchart should be kept to a one-page limit. Flowcharts must have a single starting and a single ending point as dictated by structured programming. Complicated processes, sequences of processes and control sequences should initially be represented as a single process. Once a simplified flowchart is created, the simplified processes should be expanded in additional flowcharts created on new pages. The rules for creating structured flowcharts are shown graphically in figure 4.

An embedded microcontroller course cannot cover, nor should it try to cover, the entire C programming language and all of its nuances. The essential C programming topics to solve problems with embedded microcontrollers include basic program structure, character and integer sized variables, the assignment operator, the arithmetic operators, and the bitwise operators. In order to add "intelligence" to the software through decision making, the *if/else* and the *while loop* constructs need to be taught along with the relational operators. In order to create modular software,

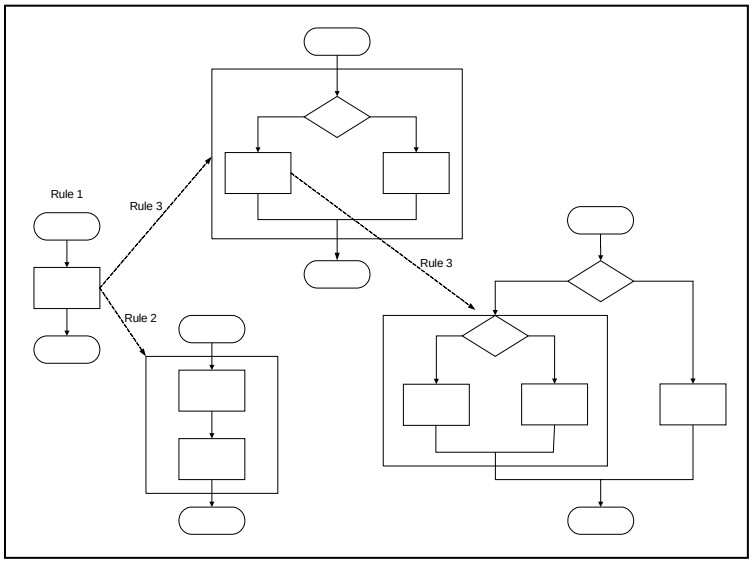

Figure 4 – Flowcharting.

functions should be taught. Arrays should also be taught to assist in creating appropriate, structured code. Topics like *do/while loops* and *for loops* can be included, but are not necessary as those operations can be performed by *while loops*. Additional topics like *pointers* can also be avoided, as they tend to confuse beginning programmers and are not required in an introductory microcontroller course.

Once the number of C language elements has been reduced, a strategic approach must be taken to implement the language. Trying to teach all of the C language elements at once is counterproductive and unnecessary. A carefully designed sequence of laboratory projects will facilitate the scheduling of topics and allow for a just-in-time (JIT) approach for teaching the C language concepts.

```
/*********************
Lab experiment 1 - Jeffrey J. Richardson – Jan. 20, 2005
*********************/
#include <mega16.h>

void main(void)
        {
        DDRA = 0x00;                    // set Port A for Input
        DDRC = 0xFF;                    // set Port C for Output

        while(1)                        // never end
                {
                PORTC = PINA;           // echo value
                }
        }
```

Figure 5 – First Program

Assuming that one laboratory experiment will be conducted each week, then only the *new* C language topics covered in the upcoming lab assignment must be covered prior to the laboratory in lecture. For example, the first laboratory assignment is an introduction to the integrated development environment (IDE) and the available lab hardware. The students require very little C knowledge to complete this experiment. At Purdue, the lab instructors "walk" the students through the creation of a simple C program to "echo" an 8-bit value from one microcontroller port to another (the complete program can be seen in figure 5). Since the program is simple in nature, a lab instructor can explain the entire program in a matter of minutes thus not requiring any lecture time to be devoted to preparing for this laboratory activity. Most of the time in lecture is spent teaching *why* the I/O ports need to be initialized and *what* must be done to accomplish this, not *how* it is done with C. However, the lecture instructor can cover the basic program structure, also in a matter of a couple minutes, to reinforce what has or what will occur in the lab. If the details of the *while(1)* loop are omitted for now, the only other topic to cover is the assignment operator which is intuitive for most students. The *while* loop is covered in detail later in the course. For now, the students need only be concerned with what it does, not how it works.

The second laboratory assignment has the students explore integer math on an 8-bit microcontroller. In order to accomplish this experiment, the students need to be introduced to the concepts of variables and variable size limits. Since the course utilizes an 8-bit, non-floating point processor, the *character* sized (8-bits) and *integer* sized (16-bits) variables are the only two required. Reducing the number of possible variables to two significantly reduces the amount of class time required to deliver the instructional material. The operation and use of the arithmetic operators in C are straightforward and require minimal lecture time. The bulk of the lecture material can focus on applications of using variables and arithmetic operations to solve problems.

The third laboratory experiment has the students monitor the microcontroller ports and variables while performing bitwise operations. Since the students are required to have completed a digital

electronic fundamentals course prior to taking a course in microcontrollers, the bitwise C operators can be quickly and effectively taught by relating them directly to their digital electronic equivalents as seen in figures 6 and 7.

It is not until the fourth laboratory experiment that programs actually make decisions and alter program flow. Since the relational operators are used to make decisions, they must now be taught along with the *if/else* and *while loop* constructs. By eliminating the use of *do/while loops* and *for loops*, lecture time can be kept to a minimum when teaching these control structures. The sequence of processes, relationship of the decision and the program flow, as indicated by the flow lines, determine which construct must be used as seen previously in figures 2 and 3.

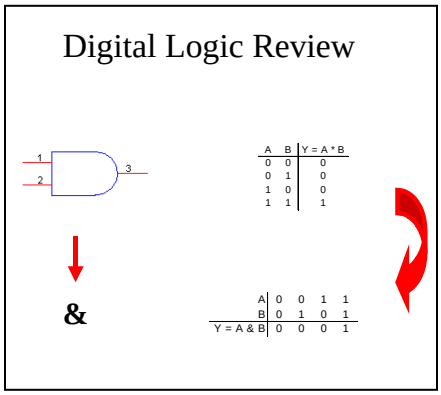

Figure 6 – Bitwise Operators 1.

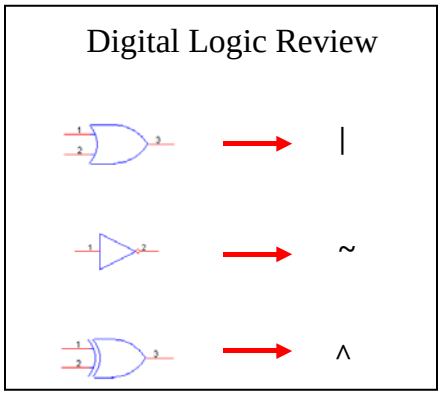

Figure 7 – Bitwise Operators 2.

Functions are not taught until the fifth week of the semester when the programs require operations like the reading of a keypad to be repeated. Arrays are not introduced until the seventh week of the semester when they are used to hold data to be displayed on 7-segment LED's. The concept of arrays can be taught by relating them to the cells located inside an Excel worksheet. This relationship provides the students a visual representation of how the data is organized.

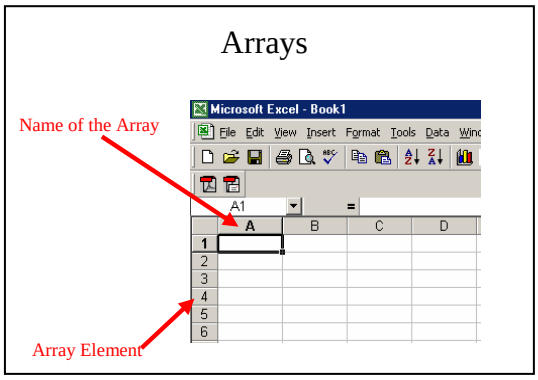

Figure 8 – Arrays

The C programming language can be mysterious and messy while promoting bad programming habits. The problem is that C gives special meaning to many punctuation characters, such as asterisks, plus sign, braces, and angle brackets. Most problems that beginners have with C relate to forgetting punctuation marks (Ayala, 2000). To reduce confusion, programming standards are enforced. These standards include indenting all blocks of code by one tab space and placing the opening and closing braces of a block of code on separate lines. Each block of code for a control structure must be bounded by braces (even single lines) whether required by ANSI C or not. Every line in the program must have a comment unless the operation is obvious. Comment must indicate *what* is being done and *why*. Shortcuts are not allowed in programs expect for the increment and decrement arithmetic operations. Some of these standards can be seen in figure 5. To assist the student in implementing these standards, a

reference sheet should be constructed to provide the students a quick reference for use when translating a flowchart into software. A reference guide will reduce the number of syntax errors and misplaced or missing punctuation marks.

5. CONCLUSION

The C programming language can and is successfully taught in the introductory microcontroller course at Purdue University through an indirect approach. The C language is treated as a tool to solve problems along with algorithms and structured flowcharts. The emphasis is placed on *what* has to be done and *why* through the generation of the algorithm and resulting flowchart. Once an appropriate algorithm and flowchart have been produced, the flowchart can quickly and easily be translated into C code by identifying the corresponding C structures in the flowchart. Implementing a reduced number of C structures along with programming standards promotes good structured programming techniques and assists in identifying missing punctuation marks. Through the careful design of laboratory experiments, the C elements required to generate a solution can be spread across the entire course. The techniques listed in this paper, when applied properly, will lead to an effective method for teaching the C programming language in an introductory embedded microcontroller course.

REFERENCES

Antonakos, J. L. & Mansfield, K. C. (2001). *Structured C for Engineering And Technology.* Prentice Hall, Upper Saddler River, NJ.

Ayala, K. (2000). *The 80251 Microcontroller.* Prentice Hall, Upper Saddle River, NJ.

Barnett, R. H. (1995). *The 8051 Family of Microcontrollers.* Prentice Hall, Englewood Cliffs, NJ.

Barnett, R. H. & Cox, S. A. & O'Cull, L. D. (2003). *Embedded C Programming and the Atmel AVR.* Delmar Learning, Clifton Park, NY.

Bogen, A. & Wollan, V. (N.D.). *AVR Enhanced RISC Microcontrollers.* ATMEL Development Center, Trondheim, Norway.

CodeVisionAVR User Manual *(2001). HP InfoTech S.R.L. Bucharest, Romania.*

Darnell, P. A. & Margolis, P. E. (1991). *C: A Software Engineering Approach.* Springer-Verlag, New York, NY.

Deitel, H. M. & Deitel, P. J. (1992). *C How to Program.* Prentice Hall, Englewood Cliffs, NJ.

Efficient C Coding for AVR (2002). Atmel Corporation. Retrieved June 15, 2002, from: http://www.atmel.com/atmel/acrobat/doc1497.pdf

Holmes, S. (n.d.) *C Programming.* February 5, 2002 from http://www.strath.ac.uk/IT/Docs/Ccourse

Lewis, D. W. (2002). *Fundamentals of Embedded Software.* Where C and Assembly Meet. Prentice Hall, Upper Saddle River, NJ.

MacKenzie, I. S. (1999). *The 8051 Microcontroller.* Prentice Hall, Upper Saddle, NJ.

Myklebust, G. (N.D.). *The AVR Microcontroller and C Compiler Co-Design.* ATMEL Development Center, Trondheim, Norway.

Reisdorph, K. (1998). *Teach Yourself Borland C++ Builder 3 in 14 Days.* Sams Publishing, Indianapolis, IN.
Schultz, T.W. (1998). *C and the 8051.* Hardware, Modular Programming, and Multitasking. Prentice Hall, Upper Saddle, NJ.
Schultz, T.W. (1999). *C and the 8051.* Building Efficient Applications. Prentice Hall, Upper Saddle, NJ.