Why Should Students Learn C?
Many students moan and groan when they enter a computer programming class. One student compared her programming fears with other students' fears of geometry, calculus or test taking. Often students ask why they should learn programming in the first place. This question can be answered from two perspectives: research and practical language benefits. A discussion of the research involving computer programming in general and the specific benefits of programming in C follows.

Examining the Research
There are very few studies which deal specifically with the benefits of learning C programming; however, general research on computer programming abounds. The most studied programming languages have been Logo, Pascal, and BASIC. Research on computer programming examined its cognitive benefits including improvements in problem solving, critical thinking, planning, and transfer. The general body of research in this area is mixed. Both positive and negative findings have been reported. The positive results are listed in the next paragraphs.

Programming increased student problem solving ability when skills learned from programming were combined with teacher guidance, direct instruction, and intentionally applied to other domains or areas (Lehner & Smith, 1986; Kay & Mayer, 1986; Mann, 1986; Clements & Gullo, 1984; McCoy & Orey, 1988). Lehrer and Smith examined the instructional method and the teacher's role in helping students transfer thinking skills; they found that direct and explicit teacher guidance fostered the development of these skills. Direct and mediated instruction helps students successfully acquire problem solving and transfer skills (Clement, C., Kurland, D.M., Mawby, R., & Pea, R.D. 1986; Emilovich & Miller 1986; Gavier, 1988; Saidman, 1981; Swan, 1989).

Kay and Mayer (1988) offered, "Our results indicate that programming can serve as a vehicle for learning general thinking skills, but only if those skills are learned and practiced in the programming environment and their applicability to other domains is recognized" (p. 73). According to this statement, programming activities have merit when the skills learned are practiced and applied.

Mann (1986) compared the problem-solving abilities of eighth graders instructed in Logo with the problem-solving abilities of eighth-graders who did not receive Logo instruction. He found that the Logo group improved its problem-solving ability as measured by performance on the non-verbal battery of a cognitive abilities test.

In a 1984 study by Clements and Gullo, improvements in reflective and divergent thinking, language fluency, and the ability to give directions were noted. Horton and Ryba (1986) indicated positive findings from a study they conducted. A Logo and non-Logo group were studied. Although small sample sizes were used, students in the Logo group seemed to outperform students in the non-Logo group in several areas: writing directions, estimating designs, and performing block design tasks.

McCoy and Orey (1988) conducted a study to determine whether programming in BASIC for one semester increased middle and high school students' problem-solving ability. Their work indicated that after one semester of BASIC programming instruction students' problem-solving scores increased.

Practical Benefits of Learning C
C is rapidly becoming "the language of choice" in industry. It is a powerful and flexible programming language that can be used for many purposes. Several word processing packages and graphics programs on the market are written in C, and many companies are investing in C because of its portability, modular nature, structured programming features, and applicability to many different types of problems.

Portability. The ability to write programs for one platform and then use those same programs on a totally different system or platform with little or no modification is portability. This feature saves time and money. Programs can be written and debugged once and then "ported" to different environments without restructuring the code or making major changes to it. This is an extremely valuable characteristic—especially with large, complex programs that would otherwise require an additional investment in programmer time and company cost. This also makes the program usable in a much shorter timeframe.

Modularity. Modularity is a decomposition strategy for programs that are large and complex. This technique is based on the Parnas Principle of 1971 which says that only one design decision should be expressed in a program module, function or procedure (MacLennan, 1987). Modularity simplifies programming and makes it possible for many independent aspects of a program to be "worked on" simultaneously by different members of a programming team. Each separate module is given only the information it needs in order to perform its task. C promotes modularity through the use of predefined functions stored in library files. The use of "built-in" functions simplifies programming tasks, shortens program development, and makes maintenance easier.

Modularity in C is also accomplished by having program compilation and linkage take place separately. During linkage, the programmer's object code (created from compilation of the source program) is combined with code from C Libraries and start-up code (similar to Job Control Language).

Structured Programming. C promotes structure in program design by requiring header files at the beginning of programs,
grouping function statements within braces, and requiring a main function that is terminated with a return statement. Execution always begins with main function and then continues to other functions. C allows its programmers to create program designs which do not use the "GOTO" statement for branching, and permits programmers to design code with one entry and one exit. A typical C program looks like the following example:

```c
/* Programmer Comments */
/* Programmer Identification */
/* Program and File Name & path */
/* Program Purpose */

#include <stdio.h>
#include <conio.h>

main() /* Required Function */
{
    float i=0, sum=0, int_numb=0; /* Declarations */
    float average(float sum, float i); /* Function decl */
    float avg;
    clrscr();
    printf("\t\tType any number and press enter key.\n");
    printf("\t\tWhen finished, type q to quit.\n\n\n\n");

    while (scanf("%f", &int_numb) == 1)
        {
        sum+=int_numb; /* Sum user entries */
        i++;
        /* Count the number of */
        printf("\t\t"); /* User entries */
        printf("\n\n\n\t\t\tSum: %f Number of entries: %f\n", sum, i);
        printf("\t\tThe average is %0.2f\n", average(sum, i));
        
        return 0;
    
    float average(float sum, float i)
    {
    float avg;
    avg=sum/i; /* Compute and return */
    return avg; /* avg to main */
    }
}
```

This may look like a bunch of "hocus pocus," but it is actually a program that computes the average of a series of user-entered numbers. The main function declares or defines three integer variables and one floating-point variable. It also declares a function named average which accepts two arguments (sum and i), and returns a real value. A prompt asks the user to enter a number, and as long as the user is entering numeric values a total is kept of all numbers entered. Once the user stops entering numeric values, the average function is called to calculate and return the average of all entries.

**Applicability to Many Problems.** C, unlike Cobol or Fortran, can be effectively used for many different problems. It can perform extensive character manipulation often needed for Cobol reports involving inventory, payroll, and accounts receivable information. C (like Fortran) also can provide a user with the precision needed for scientific computation—formulas, equations, and the use of other numbering systems—octal, binary, and hexadecimal. Low level bit manipulation also is possible with C. This allows a programmer to control hardware components and simulate some operating system tasks.

**How Should Students Learn C?**

**Practical Tips on Teaching C Programming**

Programming generally makes many cognitive demands of programmers—planning, learning language features, designing, and acquiring procedural skills (Linn, 1985). Instruction in C can address these demands if it is well-organized, thorough, encourages active student participation, presents a psychologically safe learning environment that is varied, and presents many opportunities for student learning. The seven tips listed below can make programming instruction easier.

**Tip 1: References and Resources**

A good text for C Programming is one that is clearly written, contains a listing of further sources of information, and presents examples that are closely tied to the operating environment the student will use. Clear illustrations, up-to-date material (using the latest ANSI standards), positive reviews, document formatting (type, fonts should contribute to communication with the reader), and sample coding should correspond with your objective—helping students learn to use C. All illustrations, and headings should help your students visualize and organize new concepts and their application. Practice problems, "what if" scenarios and programming exercises should be included in the text. Examples of such texts on C programming are: *New C Primer Plus* by Waite and Prata (1993) and *Teach Yourself C in 21 Days* by Aitken and Jones (1992). Any text you choose should start at a low level and work up to more complex tasks. For example, variables and variable types should be presented before arrays and linked lists. Remember, foundation is important! Give students background information on which they can build.

Resources are another issue. We have found it helpful to use an integrated development environment in our instruction. This type of atmosphere makes compilation and linkage easy, because students enter key strokes necessary instead of a series of commands. Development is easier because students use a built in editor and menus in a Windows-type environment. They also create executable files which can be run from DOS.

**Tip 2: Lots of "Hands On"**

You know the old adage, "practice makes perfect." We think an even better one is "perfect practice makes perfect." We give students worksheets at each class period. These sheets contain programming problems, exercises debugging parts of programs, questions regarding principles we are studying and interpretation dialogues where students tell us the result of a programming segment. Students are supposed to desk-check their inter-
pretation dialogues and write or type their responses. After this, they check their written responses by executing the program. This gives them feedback and allows them to analyze and evaluate worksheet code. Often they spot errors quickly, and find that they are actually learning C.

**Tip 3: PLEASANT ENVIRONMENT**
Remove environmental stress. Make students feel comfortable. Encourage students to ask and answer questions. Also, allow students to work together on small group projects. We often let groups of three students take a program and dissect it. Each group member has a role. One will “desk check” the program, one will compile the program, and one will present the group’s explanation of the program to the class. Students seem to enjoy these group problem-solving ventures. They encourage collaboration, and they make students responsible for explanations and instruction within the groups.

**Tip 4: EMPHASIZE DOCUMENTATION**
Good documentation leaves a trail within the program for those who were not directly involved in the program’s design or implementation. (This is so important that we always stress this in class.)

Documentation should be placed at the top of each program in the header, and documentation should be interspersed (via blocks of user comments) within the body of the program. Here are some of the things that should be included in header documentation: a description of the program’s tasks, the complete name of the program (including path specification), input and output created, and special programming features, such as special functions used and dependent data. Documentation within the body of the program should be clearly written and should describe the purpose of programming segments. Establish documentation standards for all programs that will be graded.

**Tip 5: PROGRAMMING TEMPLATES**
Templates are essentially examples of programs you will have your students write. Give students many examples which illustrate a variety of programming tasks and programming features. Encourage students to execute and document the examples. This helps familiarize students with reading, examining, and modifying existing programs. A lot can be learned in this way. Build on these templates by having your students incorporate templates into new programming assignments. Templates could be very useful for sorting, manipulating strings and creating files.

**Tip 6: METHODOLOGY**
Help students formulate a plan for all programming activities. Emphasize the steps or “how to’s” which might eliminate or reduce student confusion or misconceptions. Many aspects of programming involve “tried and tested” methods. Teach those that apply to your students. As an example, the programming process itself is comprised of many steps, from analyzing the problem to debugging the code written. Each of these steps emphasizes planning and organization. Help students formulate plans of action, so that they can move from one phase of the problem to another. Another example is the testing process. There are several methods of testing: black box, top down, bottom up. If you teach your students techniques like these, they will have tools they can use to help them solve some potential problems.

**Tip 7: PRACTICAL EXAMPLES**
This is helpful when teaching data structures. We try to use arrays and pointer notation on common elements. We find it helpful to use visual representations of common occurrences and problems. Examples are a flowchart of the trip from home to school, a flowchart of making a telephone conversation, an assortment of paper money in our pocket, linked lists of courses required. Sometimes pictures of “every day” events and items give students a better idea of the underlying programming concepts.

**CONCLUSION**
Research suggests students should study C programming because it is cognitively challenging, and it is a good way to help students construct or build new knowledge. It also encourages students to transfer what they have learned to other domains. Effective pedagogy in C can be fostered by following these practical tips: use “good” references, emphasize hands on activities, create a pleasant learning environment, emphasize documentation, and use programming templates, systematic methods and practical examples.

**REFERENCES**
cational Research Association, San Francisco, CA.

**CONTRIBUTORS:**

Demetria Ennis has worked as a Computer Analyst at Louisiana State University and a programmer at International Business Machines. She is presently an assistant professor at the University of North Texas. She is interested in computer programming and problem solving, educational applications of Artificial Intelligence, and teacher effectiveness.

Willie Ennis III has worked at Southern University and is presently employed as an Assistant Professor at Southeastern Louisiana University in Hammond, LA. His interests include hypermedia, video production, educating with technology, and visual literacy. Dr. Ennis has traveled extensively in Asia and Hong Kong where he pursues joint research projects on visual literacy. Addresses: 1200 Dallas Drive #928, Denton, TX 76205; ph. 817/665-3536; fax 817/565-2185; e-mail: ennis@unt.edu or ennis@coe.unt.edu.

iste@oregon.uoregon.edu

The wide world of Internet. It's growing larger and making our world smaller. If you're not already an Internet user, you probably want to be one.

As you prepare to keyboard your way into the Internet fast lane, ISTE can be a big help to you. We've been busy adding Internet books to our mix of publications available to educators.

The ISTE Internet collection includes the following titles:

- **Way of the Ferret—Finding Educational Resources on the Internet**
- **Hands-On Internet—A Beginning Guide for PC Users**
- **How the Internet Works**
- **How to Use the Internet**
- **NetPower—Resource Guide to Online Computer Services**
- **Realizing the Information Future—The Internet and Beyond, Second Edition**
- **The Internet Companion—A Beginner's Guide to Global Networking**
- **The Mac Internet Tour Guide—Cruising the Internet the Easy Way**
- **The PC Internet—Cruising the Internet the Easy Way**
- **The Windows Internet Tour Guide—Cruising the Internet the Easy Way**
- **Zen and the Art of the Internet—A Beginner's Guide**

These books are valuable keys for getting your computer started and on its way via the information superhighway.