"Do I Press Return?"

Liz Levine
Beverly Woolf
Rich Filoramo

Computer and Information Sciences
University of Massachusetts, Amherst

COINS TECHNICAL REPORT 85-50

"Do I Press Return?"

Liz Levine
Beverly Woolf
Rich Filoramo

Computer and Information Sciences
University of Massachusetts, Amherst

ABSTRACT

The introductory programming course at this university attempts to serve some 1500 students each semester. The attrition rate, due in part to the overload on the system and in part to the students' difficulties in "keeping up", has, at times, approached 25%. In response to this situation we have revised and reordered the curriculum for use in an experimental course designed for the novice user. The course is directed toward discovering and addressing the confusions of new programming students. It facilitates our ongoing study of the novice programmers' response to graphics, friendly interface packages and the revised curriculum which includes the teaching of procedures and control structures at the beginning of the course. In studying these responses we have learned some techniques in aiding the novice user to unravel some of the mysteries surrounding the acquisition of programming skills. The course is constantly undergoing development in addition to being in its second semester as a departmental offering. It is detailed in this paper.

1. Introduction

We have designed a laboratory/classroom to explore the needs of the novice programmer and have begun to evaluate and improve the courseware and teaching methods used to instruct the beginnning programmer. We are also investigating the beginner's classroom environment. The motivation for this undertaking stems from the limitations of working on a large time-sharing system with its concurrent poor response time, weak editor facilites and low availability of terminals. The pilot course on the other hand runs on a VAX 780, dedicated to department research, uses a Barco color monitor and a DEC VT125 terminal. The course will soon move to personal computers networked for communicaton and resource sharing.

We have been able to generate integrated hardware and software packages into the curriculum and have culled inspiration from the teaching potential available in the creation of these packages. The curriculum uses these packages to aid the beginning student and to present the rudiments of Pascal programming while allowing the user to apply this knowledge to her own field of interest.

2. Graphics Pascal Packages

The courseware consists of software packages inspired by LOGO, written in Pascal and designed to facilitate the teaching of Procedures at an early stage in the course. The packages, collectively known as the InterActive Ladybug, (I.A.Ladybug), are designed around an icon in the form of a Ladybug which represents the point of a graphics pen on the screen. The students command the Ladybug both interactively and through Pascal programs to Crawl, Turn, Dot (leave a trail of dots), Penup (move without leaving a trail), and Sequence (remember sequences of commands in an interactive mode). The students are first taught to experiment with the Ladybug in her interactive mode. A single key input activates the Ladybug; a distance, angle or color value input following the single key command determines such things as the length or color of a line. A repetition of sequences of commands can also be executed, thus preparing the student to work with iteration and looping constructs. Using the Sequencing option available in this mode, students are able to record Ladybug movements and incorporate these in larger drawings. In this way, students become familiar with breaking up a problem into small pieces and coding each piece.

Following three to four weeks of work with the I.A.Ladybug the students become acquainted with some of the groundwork for Pascal programming techniques and are taught to command the Ladybug from within a program. Once the student becomes proficient in the specifics of Pascal syntax and program development a new and expanded Ladybug package for Pascal loops is introduced. This package provides a chart for graphically "recording" the Ladybug's meal. The Ladybug "eats" aphids which are represented by colorful and randomly shaped icons on the screen. Each "meal" is programmed by the student who adjusts the control variable of a *WHILE, REPEAT,* or *FOR* loop For example two commands used with this package are:

"WHILE (aphid = red) *eataphid"*
"WHILE (aphidweight > 700) *eataphid"*

The first command causes all red aphids to be eaten and the second causes aphids of weight greater than 700 to disappear. The Ladybug becomes proportionally bigger with every insect eaten. Using this package, the student learns to code boolean variables to test the color, size, or distance of an aphid.

3. Results in the Classroom

We find that students are intrigued, albeit timid, by the use of graphics in a computer course. We find, however, that there is a steady level of interest. One reason for this may be the friendliness of the software. Unhampered by opaque error messages and a maze of user manuals, the student is free to explore the system without risk. With little knowledge of the language primitives, the student is, nevertheless, able to witness powerful effects of her efforts.

For example, for his final project, a business student wrote an interactive program to produce a skeleton housing plan which accepted variables for the dimensions of its walls, rooms, and house type.

Example student comments about this course are:

"As an Environmental Design major I realized that I would sooner or later have to work with a computer in design, I decided it was time to know my enemy. The low stress, small scaled classroom with maximum opportunity for student-teacher interaction quickly dispelled my fears and misconceptions."

"Enter the ever loving and humble Ladybug; she was the most disarming factor of my computer fears."

"The use of graphics to learn the rudiments of programming was a key factor in understanding the material. The visual display of our efforts made the whole process far more meaningful."

"Without the first introduction to Ladybug I think that I would have had a much more difficult time grasping the elements of Pascal." It would have been a much less creative process for those of us that do not so readily and immediately think in the logical ways required for writing a program "

Graphic output from programs written by students in the course are shown at the end of the paper.

Graphic and textual feedback from the system to the student is a teaching method which requires further study. The student sees the graphic result of joining several small drawings or earlier designs into one larger display. Textually, she gets practice writing code for the simpler problems and then uses indented modules (Procedures, If/Then, While loops) to include these smaller modules in the final program. The advantage of the dual feedback is that the student is quick to accept and practice this method of problem-solving.

4. Format of the Class

We looked for three things in prospective students 1) a non-technical backround, 2) a willingness to partake in an experimental course and 3) an ability to contribute suggestions and criticisms regarding the construction of the course.

The class uses both traditional and innovative techniques during the three weekly meetings. A short lecture is used to present new material. Central to this lecture is distribution of a set of definitions and manuals designed to aid the user in comprehension of new material. Manuals were written to accompany each phase of the new student's interactions with the system, including the screen editor, the graphics packages, the Pascal compiler, and the system command language. A question session follows the lecture and often Pascal programs with built in errors are displayed and students asked to eliminate the bugs. The last portion of the class is devoted to student experimentation and teacher consultation on any current problems.

4.0.1 Assignments

Assignments were designed so students had some choice in what each program would produce. Assignments were accepted when the user had demonstrated sufficient knowledge of the concept in question. For example a typical assignment might involve designing an interactive program which allowed the user to design her own house or rewriting the simpler software to accomplish a variety of original tasks.

The order in which topics were presented was based on several factors. A primary consideration was the teaching of procedures at an early stage in the curriculum. In fact, procedures were taught as soon as the student was comfortable with the computer system. This was prompted by the belief that an early acquaintance with these tools enables a student to write modular programs at an early point in her career (See, for instance MINDSTORMS, Papert). Students were given an average of three programming assignments a month.

4.0.2 The Curriculum

The curriculum is outlined below:

the operating system
screen editor
files
interactive course work
introduction to Pascal
documentation
Procedures
Pascal programs to command the Ladybug
Variables
Control structures
While, Repeat and For loops
text output
Design and implementation of final projects
applications of computers

The sequence of topics is also motivated by the belief that the user can and should access powerful utilities before she is able to understand their inner mechanics. We equip the student with some of this richness in the form of a flexible editor and learning system so that she may move on to programming concepts without having to grapple with some of the low level tinkering so often inherent in introductory courses. Having these facilites at her disposal enables the new user to move much more rapidly through basic programming constructs. In addition, some of the usefulness of a computer is witnessed at an early stage in contrast with the frustration frequently experienced in such courses.

5. Programming Skills Need Repetitive Reinforcement

We chose not to use formal testing to evaluate students' progress in the course. The decision was part of our effort to discover other methods which might determine whether a student had attained adequate grasp of the material. We also dispensed with fixed due dates for assignments preferring to allow a negotiable window of time during which completed assignments would be accepted. We hoped that by reducing the pressure associated with testing and inflexible due dates the student would give more attention toward learning and retaining the material. Students were graded based on participation in the class, adherence to a regular work schedule and completion of a specific number of assignments including a four week project which they designed and wrote.

The results of this experiment on scheduling assignments produced a nearly anarchic state. Too many students who might otherwise have been able to master the material in a reasonable amount of time were unable to allocate their time efficiently. The most significant reason for this result is the fact that programming skills cannot be learned the night before an assignment is due. Unfortunately, many students complete courses by determining the nature of the requisite assignment material and accomplishing the assignment in a single sitting. This method (and it is a well-honed method), is not conducive to the concrete acquisition of programming skills.

In particular, we found that several students lacked experience with the notion of repetition as a technique for learning. There do exist courses where completion of one assignment is not dependent upon knowledge retained from the previous one; in this particular course that is not the case and levels of knowledge required for each task presume facility with any material utilized up through that point. Inability to recognize this fact led some students to resist starting work early enough to allow for the clarification of bugs. This resulted in completion delays and a misunderstanding about how the final output would look. Clearly, an introductory course in programming needs to present an additional branch of instruction; the novice requires some aid in structuring the studying of this material. Exercises which stress repetition and usage of individual constructs is a possibility. Merely telling the student to look at the problems in the back of the chapter is not the answer.

We suggest, therefore, that a reasonable balance needs to be achieved between a lenient grading policy and one that does not allow for different rates of acquisition. We believe that we have qualitatively, at least, documented the fact that students achieve graduated levels of programming competence in accordance with their ability to both grasp programming concepts and exercise them regularly. These differences in abilites needs to be accounted for in an introductory programming course, and allowed for within reasonable limits when due dates are arranged. Extreme leniency in this regard is not the answer; indeed it appears to hinder rather than help the student to move ahead.

Our intention is to enforce:

1. explicit due dates without the facility to negotiate

2. explicit partial due dates, ie, due dates for an outline, design and meta-language specification

3. evidence from the student that consistent work habits are being developed through completion of small exercise sets in addition to programming assignments

6. Results: The graphics packages

We have organized an introductory programming course utilizing friendly graphics packages to teach various programming constructs. We have found that despite the aforementioned difficulties with work routines and structured completion of assignments, students respond productively to these packages. One observable reason for this is plainly the built in friendliness of the software. Unhampered by opaque error messages and a maze of incomprehensible user manuals, the student is free to explore the system without risk. With little knowledge, the user is able to witness some powerful effects of her efforts. For example a few simple commands to the Ladybug will result in a screen filled with colorful designs. As the user becomes more discriminating in what she wishes to produce, small increments in knowledge allow her to increase the complexity of graphical output. We have thus built into the course an incentive plan enabling the student to envision her future in the course.

We have found that a structured laboratory session is useful for beginning students. Technical snags as well as conceptual hurdles are inevitable for any user; for the new user they can be both baffling and halting. Through weekly sessions, during which the student writes and compiles a program or explores further the intricacies of the I.A.Ladybug the instructor is on hand to answer questions and help sort out problems when they occur. In this way the waiting period until the next meeting is eliminated and the student can move ahead in her work. In addition, the student can bring any chronic problems to the laboratory and demonstrate to the instructor the exact circumstances under which they occur. This is useful in keeping track of both hardware and software glitches.

7. Conclusions

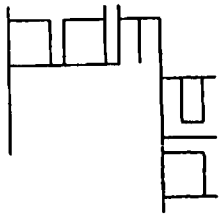
The course is currently in its final semester of experimentation. Final validation as a regular university course will allow us to service both the department and the larger university community.

In the department, we will continue to provide a testbed where an active research community can experiment with friendly programming interfaces and the design of graphics packages for the novice. This research advantage is currently being tapped in the development of a dynamic on-line help system to be used in conjunction with the Ladybug system.

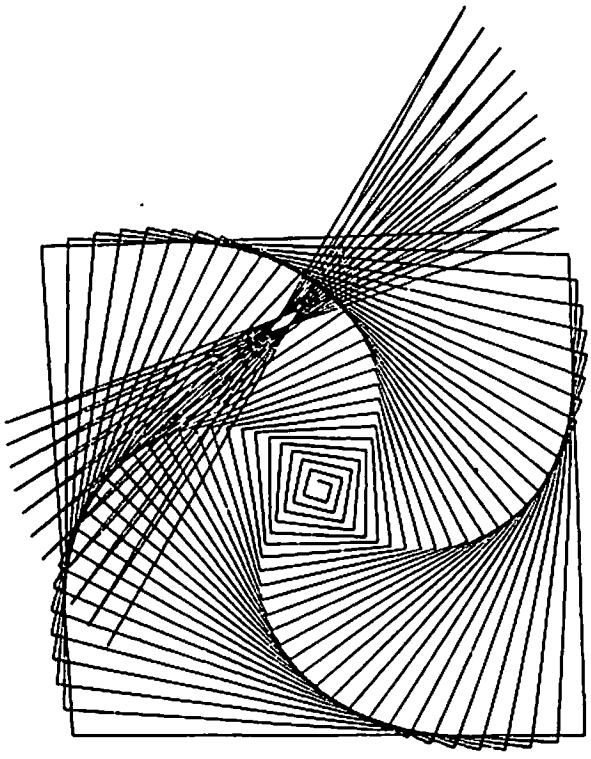
Expansion and development are underway with the aim of servicing 50 students in the spring of 1984. In particular we hope to service those members of the university community who find the larger course inacessible as a learning mode, yet who wish to attain proficiency in a programming language and technique.

Our software and curriculum packages are being expanded to include records, files and arrays. We are thus able to treat all the concepts contained in a traditional introductory programming course. Yet, we believe that by using graphics packages and the results of this research effort, we will have provided an opportunity for the novice to attain problem solving proficiency without facing the obstacles inherent in larger more text orientated curriculums. We trust that our results for both the department and the community will prove useful in meeting their needs creatively and efficiently.

We gratefully acknowledge the initial work on this course by Jeff Bonar, and the progrmming assistance of Sallie Blackburn and Geoff Brown.

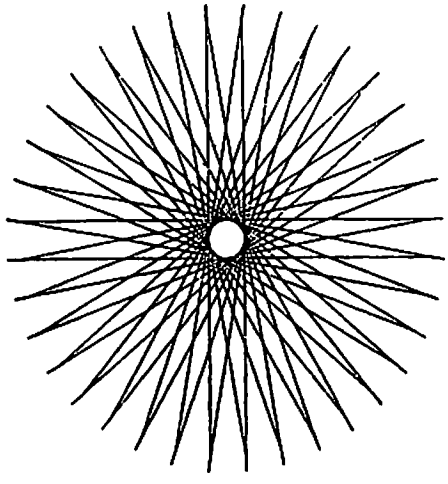


Pamela's Customized Signature.



"A Squiral made with a WHILE loop"

Kemon's Customized Signature.



A Sun made with a REPEAT loop.

"Do I Press Return?"

Liz Levine
Beverly Woolf
Rich Filoramo

Computer and Information Sciences
University of Massachusetts, Amherst

COINS TECHNICAL REPORT 85-50

"Do I Press Return?"

Liz Levine
Beverly Woolf
Rich Filoramo

Computer and Information Sciences
University of Massachusetts, Amherst

ABSTRACT

The introductory programming course at this university attempts to serve some 1500 students each semester. The attrition rate, due in part to the overload on the system and in part to the students' difficulties in "keeping up", has, at times, approached 25%. In response to this situation we have revised and reordered the curriculum for use in an experimental course designed for the novice user. The course is directed toward discovering and addressing the confusions of new programming students. It facilitates our ongoing study of the novice programmers' response to graphics, friendly interface packages and the revised curriculum which includes the teaching of procedures and control structures at the beginning of the course. In studying these responses we have learned some techniques in aiding the novice user to unravel some of the mysteries surrounding the acquisition of programming skills. The course is constantly undergoing development in addition to being in its second semester as a departmental offering. It is detailed in this paper.

1. Introduction

We have designed a laboratory/classroom to explore the needs of the novice programmer and have begun to evaluate and improve the courseware and teaching methods used to instruct the beginnning programmer. We are also investigating the beginner's classroom environment. The motivation for this undertaking stems from the limitations of working on a large time-sharing system with its concurrent poor response time, weak editor facilites and low availability of terminals. The pilot course on the other hand runs on a VAX 780, dedicated to department research, uses a Barco color monitor and a DEC VT125 terminal. The course will soon move to personal computers networked for communicaton and resource sharing.

We have been able to generate integrated hardware and software packages into the curriculum and have culled inspiration from the teaching potential available in the creation of these packages. The curriculum uses these packages to aid the beginning student and to present the rudiments of Pascal programming while allowing the user to apply this knowledge to her own field of interest.

2. Graphics Pascal Packages

The courseware consists of software packages inspired by LOGO, written in Pascal and designed to facilitate the teaching of Procedures at an early stage in the course. The packages, collectively known as the InterActive Ladybug, (I.A.Ladybug), are designed around an icon in the form of a Ladybug which represents the point of a graphics pen on the screen. The students command the Ladybug both interactively and through Pascal programs to Crawl, Turn, Dot (leave a trail of dots), Penup (move without leaving a trail), and Sequence (remember sequences of commands in an interactive mode). The students are first taught to experiment with the Ladybug in her interactive mode. A single key input activates the Ladybug; a distance, angle or color value input following the single key command determines such things as the length or color of a line. A repetition of sequences of commands can also be executed, thus preparing the student to work with iteration and looping constructs. Using the Sequencing option available in this mode, students are able to record Ladybug movements and incorporate these in larger drawings. In this way, students become familiar with breaking up a problem into small pieces and coding each piece.

Following three to four weeks of work with the I.A.Ladybug the students become acquainted with some of the groundwork for Pascal programming techniques and are taught to command the Ladybug from within a program. Once the student becomes proficient in the specifics of Pascal syntax and program development a new and expanded Ladybug package for Pascal loops is introduced. This package provides a chart for graphically "recording" the Ladybug's meal. The Ladybug "eats" aphids which are represented by colorful and randomly shaped icons on the screen. Each "meal" is programmed by the student who adjusts the control variable of a *WHILE*, *REPEAT*, or *FOR* loop For example two commands used with this package are:

"WHILE (aphid = red) *eataphid"*
"WHILE (aphidweight > 700) *eataphid"*

The first command causes all red aphids to be eaten and the second causes aphids of weight greater than 700 to disappear. The Ladybug becomes proportionally bigger with every insect eaten. Using this package, the student learns to code boolean variables to test the color, size, or distance of an aphid.

3. Results in the Classroom

We find that students are intrigued, albeit timid, by the use of graphics in a computer course. We find, however, that there is a steady level of interest. One reason for this may be the friendliness of the software. Unhampered by opaque error messages and a maze of user manuals, the student is free to explore the system without risk. With little knowledge of the language primitives, the student is, nevertheless, able to witness powerful effects of her efforts.

For example, for his final project, a business student wrote an interactive program to produce a skeleton housing plan which accepted variables for the dimensions of its walls, rooms, and house type.

Example student comments about this course are:

"As an Environmental Design major I realized that I would sooner or later have to work with a computer in design, I decided it was time to know my enemy. The low stress, small scaled classroom with maximum opportunity for student-teacher interaction quickly dispelled my fears and misconceptions."

"Enter the ever loving and humble Ladybug; she was the most disarming factor of my computer fears."

"The use of graphics to learn the rudiments of programming was a key factor in understanding the material. The visual display of our efforts made the whole process far more meaningful."

"Without the first introduction to Ladybug I think that I would have had a much more difficult time grasping the elements of Pascal." It would have been a much less creative process for those of us that do not so readily and immediately think in the logical ways required for writing a program "

Graphic output from programs written by students in the course are shown at the end of the paper.

Graphic and textual feedback from the system to the student is a teaching method which requires further study. The student sees the graphic result of joining several small drawings or earlier designs into one larger display. Textually, she gets practice writing code for the simpler problems and then uses indented modules (Procedures, If/Then, While loops) to include these smaller modules in the final program. The advantage of the dual feedback is that the student is quick to accept and practice this method of problem-solving.

4. Format of the Class

We looked for three things in prospective students 1) a non-technical backround, 2) a willingness to partake in an experimental course and 3) an ability to contribute suggestions and criticisms regarding the construction of the course.

The class uses both traditional and innovative techniques during the three weekly meetings. A short lecture is used to present new material. Central to this lecture is distribution of a set of definitions and manuals designed to aid the user in comprehension of new material. Manuals were written to accompany each phase of the new student's interactions with the system, including the screen editor, the graphics packages, the Pascal compiler, and the system command language. A question session follows the lecture and often Pascal programs with built in errors are displayed and students asked to eliminate the bugs. The last portion of the class is devoted to student experimentation and teacher consultation on any current problems.

4.0.1 Assignments

Assignments were designed so students had some choice in what each program would produce. Assignments were accepted when the user had demonstrated sufficient knowledge of the concept in question. For example a typical assignment might involve designing an interactive program which allowed the user to design her own house or rewriting the simpler software to accomplish a variety of original tasks.

The order in which topics were presented was based on several factors. A primary consideration was the teaching of procedures at an early stage in the curriculum. In fact, procedures were taught as soon as the student was comfortable with the computer system. This was prompted by the belief that an early acquaintance with these tools enables a student to write modular programs at an early point in her career (See, for instance MINDSTORMS, Papert). Students were given an average of three programming assignments a month.

4.0.2 The Curriculum

The curriculum is outlined below:

the operating system
screen editor
files
interactive course work
introduction to Pascal
documentation
Procedures
Pascal programs to command the Ladybug
Variables
Control structures
While, Repeat and For loops
text output
Design and implementation of final projects
applications of computers

The sequence of topics is also motivated by the belief that the user can and should access powerful utilities before she is able to understand their inner mechanics. We equip the student with some of this richness in the form of a flexible editor and learning system so that she may move on to programming concepts without having to grapple with some of the low level tinkering so often inherent in introductory courses. Having these facilites at her disposal enables the new user to move much more rapidly through basic programming constructs. In addition, some of the usefulness of a computer is witnessed at an early stage in contrast with the frustration frequently experienced in such courses.

5. Programming Skills Need Repetitive Reinforcement

We chose not to use formal testing to evaluate students' progress in the course. The decision was part of our effort to discover other methods which might determine whether a student had attained adequate grasp of the material. We also dispensed with fixed due dates for assignments preferring to allow a negotiable window of time during which completed assignments would be accepted. We hoped that by reducing the pressure associated with testing and inflexible due dates the student would give more attention toward learning and retaining the material. Students were graded based on participation in the class, adherence to a regular work schedule and completion of a specific number of assignments including a four week project which they designed and wrote.

The results of this experiment on scheduling assignments produced a nearly anarchic state. Too many students who might otherwise have been able to master the material in a reasonable amount of time were unable to allocate their time efficiently. The most significant reason for this result is the fact that programming skills cannot be learned the night before an assignment is due. Unfortunately, many students complete courses by determining the nature of the requisite assignment material and accomplishing the assignment in a single sitting. This method (and it is a well-honed method), is not conducive to the concrete acquisition of programming skills.

In particular, we found that several students lacked experience with the notion of repetition as a technique for learning. There do exist courses where completion of one assignment is not dependent upon knowledge retained from the previous one; in this particular course that is not the case and levels of knowledge required for each task presume facility with any material utilized up through that point. Inability to recognize this fact led some students to resist starting work early enough to allow for the clarification of bugs. This resulted in completion delays and a misunderstanding about how the final output would look. Clearly, an introductory course in programming needs to present an additional branch of instruction; the novice requires some aid in structuring the studying of this material. Exercises which stress repetition and usage of individual constructs is a possibility. Merely telling the student to look at the problems in the back of the chapter is not the answer.

We suggest, therefore, that a reasonable balance needs to be achieved between a lenient grading policy and one that does not allow for different rates of acquisition. We believe that we have qualitatively, at least, documented the fact that students achieve graduated levels of programming competence in accordance with their ability to both grasp programming concepts and exercise them regularly. These differences in abilites needs to be accounted for in an introductory programming course, and allowed for within reasonable limits when due dates are arranged. Extreme leniency in this regard is not the answer; indeed it appears to hinder rather than help the student to move ahead.

Our intention is to enforce:

1. explicit due dates without the facility to negotiate

2. explicit partial due dates, ie, due dates for an outline, design and meta-language specification

3. evidence from the student that consistent work habits are being developed through completion of small exercise sets in addition to programming assignments

6. Results: The graphics packages

We have organized an introductory programming course utilizing friendly graphics packages to teach various programming constructs. We have found that despite the aforementioned difficulties with work routines and structured completion of assignments, students respond productively to these packages. One observable reason for this is plainly the built in friendliness of the software. Unhampered by opaque error messages and a maze of incomprehensible user manuals, the student is free to explore the system without risk. With little knowledge, the user is able to witness some powerful effects of her efforts. For example a few simple commands to the Ladybug will result in a screen filled with colorful designs. As the user becomes more discriminating in what she wishes to produce, small increments in knowledge allow her to increase the complexity of graphical output. We have thus built into the course an incentive plan enabling the student to envision her future in the course.

We have found that a structured laboratory session is useful for beginning students. Technical snags as well as conceptual hurdles are inevitable for any user; for the new user they can be both baffling and halting. Through weekly sessions, during which the student writes and compiles a program or explores further the intricacies of the I.A.Ladybug the instructor is on hand to answer questions and help sort out problems when they occur. In this way the waiting period until the next meeting is eliminated and the student can move ahead in her work. In addition, the student can bring any chronic problems to the laboratory and demonstrate to the instructor the exact circumstances under which they occur. This is useful in keeping track of both hardware and software glitches.

7. Conclusions

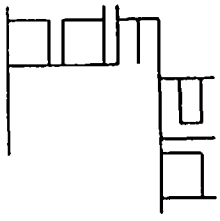
The course is currently in its final semester of experimentation. Final validation as a regular university course will allow us to service both the department and the larger university community.

In the department, we will continue to provide a testbed where an active research community can experiment with friendly programming interfaces and the design of graphics packages for the novice. This research advantage is currently being tapped in the development of a dynamic on-line help system to be used in conjunction with the Ladybug system.

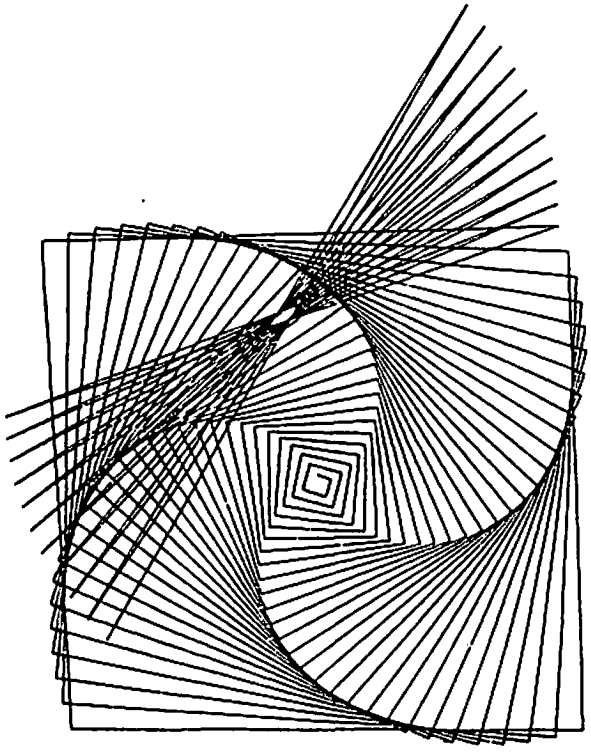
Expansion and development are underway with the aim of servicing 50 students in the spring of 1984. In particular we hope to service those members of the university community who find the larger course inacessible as a learning mode, yet who wish to attain proficiency in a programming language and technique.

Our software and curriculum packages are being expanded to include records, files and arrays. We are thus able to treat all the concepts contained in a traditional introductory programming course. Yet, we believe that by using graphics packages and the results of this research effort, we will have provided an opportunity for the novice to attain problem solving proficiency without facing the obstacles inherent in larger more text orientated curriculums. We trust that our results for both the department and the community will prove useful in meeting their needs creatively and efficiently.

We gratefully acknowledge the initial work on this course by Jeff Bonar, and the progrmming assistance of Sallie Blackburn and Geoff Brown.

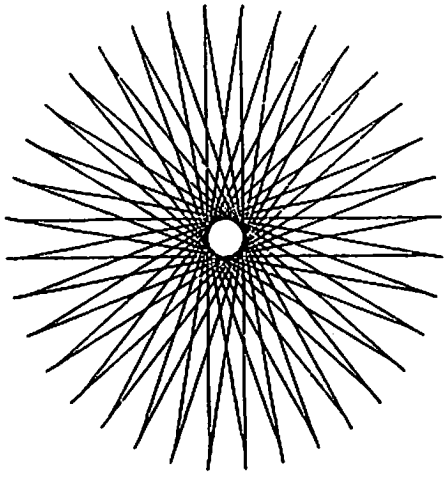


Pamela's Customized Signature.



"A Squiral made with a WHILE loop"

Kemon's Customized Signature.



A Sun made with a REPEAT loop.