In fact, workers from two different areas of computers and education jumped on the microcomputer bandwagon: those who believe that the best way to use computers is to have the computer control the student and those who believe that the best way to use the computer is to have students control the computer. Nonetheless, the dominant attitude within the BASIC culture is to teach students to program and to have them use some prepared materials, especially simulations. The eclectic nature of the BASIC culture is one of its most appealing aspects and at the same time one of its inhibiting characteristics.

BASIC and microcomputers together have given rise to a particular computer culture, which I refer to as the BASIC computer culture. There are many facets to this culture and many enthusiastic and active supporters. There is no one spokesperson, no one central figure who has given it shape. Of the several possible representatives of this culture I choose Tom Dwyer as an example. His views and theoretical position seem to reflect the dreams and aspirations of this diverse culture.

An Overview

Dwyer wants to reform the process of teaching and learning. He sees computers as providing a way to achieve this reformulation. Dwyer shares with Seymour Papert, Alan Kay, and others (including me) the belief that the best computer learning experiences consist of learning to master the computer. We will see that he differs from Papert and Kay in his model of what constitutes mastery of the computer and what needs to be done to achieve it.

I have introduced into my discussion of the theorists discussed so far some elements from their personal lives. With Dwyer, a significant element is his involvement in flying. Dwyer is a pilot, and flying is an important part of his mental imagery. Dwyer’s central model for learning is derived from this involvement. The crucial moment in learning to fly is the dramatic first solo. The goal of initial instruction for the fledgling pilot is to reach the degree of proficiency needed for solo learning, in which the pilot flies solo alone, taking charge of his or her own learning. Dwyer is struck by this quality of the computer: The learner-programmer comes more quickly to the stage of solo learning than in most other fields.

The realization of Dwyer’s dream of solo learning is shaped by another personal attribute: He is a man of action. In the 1960s Papert and Kay were dreaming of how children would program the computers of the 1980s and 1990s. Dwyer wanted them programming the machines available right then and using the programming languages available right then. This commit-
ment to action determined a major thrust of his work: how to use the BASIC language in an educationally meaningful way.

A final major difference that sets Dwyer apart from Suppes and Davis is his grass roots origin in classroom teaching. For the others, “teaching” has meant teaching in elite universities with an occasional foray into schools. Before Dwyer became a professor of computer science, he was a high school teacher, and this background shows itself in a particularly respectful attitude toward the teacher. The thrust of Suppes’s work is to eliminate the teacher. Davis seeks to define the teacher’s role in the form of well-worked-out scripts. Dwyer recommends a mode of instruction that is, in his own words, “harder on the teacher than on the student.” And, generally, one finds more references to the role of the teacher and models of the teaching process in Dwyer’s writings that in the writings of the others.

At the core of Dwyer’s approach to using computers is the idea of giving the student mastery of the computer: The student should program the computer and use it as an instrument for music and for graphic arts, as a simulator for learning to fly airplanes, as a physics or mathematics laboratory, and so on. The educational value of doing this is bolstered by reference to a list of “educational theorists, including Aristotle, Aquinas, Whitehead, Dewey, and Piaget, on the writings of psychologists like Rogers and Maslow, and on some of the contemporary insights of people like Illich and Papert” (Dwyer 1974, p. 138). From these thinkers Dwyer sees a common view of education as “that which liberates human potential, and thus the person” (Dwyer 1974, p. 138).

Dwyer (1974, p. 138) sees education as “helping people achieve certain kinds of control over their lives”—a “liberating control” that extends not only to the environment in social, physical, and economic aspects but also to themselves, contributing to “internal” control or self-control; and he sees computers in education as a way of creating a learning environment where this can occur, where learning to control a computer connects with all aspects of people’s lives.

The Teaching Process

Dwyer wants to reform the process of teaching and learning. His vision of what is possible is based on his personal experiences in flying airplanes, in programming computers, and in teaching high school. From his flying experiences he captures a metaphor explaining his theory of education: dual mode and solo mode learning. He takes as his model of a teacher a flying instructor whose goal is to “transmit” enough information and experience to his or her students while they are together, in dual mode, to enable the students to solo, to fly alone successfully. Success is determined not by a test but by not crashing and not causing others to crash.

The teacher and the student form a relationship, which Dwyer formalizes. It is a relationship that is a celebration of “good teaching.” The teacher’s role is to convey information and “sensitize” the student into “receiving” the information. Doing this requires active participation by teacher and student. This dual participation becomes clear in learning to fly. The student and the teacher know they are working toward the same end—getting the student to fly by herself. This process requires the student to internalize for herself flying knowledge and experiences. When the student is soloing she is on her own and must rely on what she knows and on debugging techniques developed while under the direct influence of her teacher.

The teacher knows that the student must “appropriate” flying knowledge for herself, and so the teacher must also be a sensitive transmitter and know that it is impossible to impose her own structure on the student. The instructor’s job is to help the student construct her own mental models of how to fly. There will, of course, be shared metaphors, but, whether these mental constructs are identical or different, this constructivist view of the teaching process is particularly conducive to a programming environment.

This view puts the student in control and indeed turns the teacher into a facilitator. Then discussion about teaching and learning styles takes on a different complexion. Whether it is more optimal for a student to use a drill-and-practice routine or a scripted discovery-learning routine can now be tested in a different way because it is with the student’s consent and participation that one method is chosen over another, if either is chosen. Once the student is in control of her own learning, both the teacher and the student are working toward a common goal: liberating the student to reach her potential through education.

Examples of Programming in BASIC

Dwyer’s vision of what can happen to students and teachers in a supportive learning environment has been shared by many teachers and students who have themselves learned to program computers. To show what it means for a young beginning student to program computers I begin with some examples of the most elementary kinds of program and proceed from there to progressively more complex examples. I have, however, taken some liberties in my selection and have chosen some examples from my own work with children in Logo classes in the 1960s. At that time, uppercase typewriter terminals were used as input devices to time-shared computers. These Logo examples circulated among many educational computerists in-
including those using BASIC. Before then, typical introductory sessions involved children writing programs to find the average of five numbers. Unfortunately, writing the program was more complicated than solving the problem by hand. Furthermore, the computer did not provide the child with a deeper insight into the problem.

An excellent introductory activity involves graphing; the student writes programs to print keyboard characters in planned patterns. Graphical representations include houses, dogs, people, signs, and words. The programs which generate these figures are remarkably close to the figures themselves, thus acquiring a kind of concreteness that makes them a good first step into programming. The programming style of using PRINT statements in BASIC to make a picture is similar to Logo techniques. For example:

```
Program                                      Result
10 PRINT "DDD    D"                           DDD    D
20 PRINT "DDDD    D"                           DDDDD    D
30 PRINT "    DDDDDDDD"                        DDDDDDDD
40 PRINT "   D   D   D"                        D   D   D
50 PRINT "   D   D   D"                        D   D   D
60 PRINT "   D   D   D"                        D   D   D
70 PRINT "11111111111"                        111111111111
80 END
```

Other examples of figures made in this way (Papert and Solomon 1971, pp. 27–29) are shown in figure 4.1.

For beginners in BASIC, a typical direction to take after graphing pictures is generating tables of numbers using a FOR...NEXT loop. For example (Dwyer and Critchfield 1978, p. 18):

```
10 PRINT "MULT. TABLE FOR 9"
20 FOR K=0 TO 12
30 PRINT K*9
40 NEXT K
50 END
```
RUN
MULT. TABLE FOR 9
0
9
18
27
36
45
54
63
72
81
90
99
108
OK

TO SIMPLE.SEN
TYPE PICK [THAT THE A SOME]
TYPE PICK [COOL WILD GROSS BEAUTIFUL]
TYPE PICK [DONKEY PROF KID]
TYPE PICK [WALKED CLAPED KISSED HUMMED WALKED]
END

Although this project might have been written by a ten-year-old in Logo, in BASIC it would have been too difficult. BASIC was designed to manipulate numbers not words.

The following project (Dwyer and Critchfield 1978, p. 161), which follows a template made up by the programmer, is easier to write in BASIC but would probably be undertaken by a junior or senior high school student.

10 PRINT"THIS PROGRAM CAN HELP YOU BECOME A "
20 PRINT" 'POET' . "
30 PRINT"PLEASE TYPE IN THE FOLLOWING KINDS OF WORDS OR PHRASES"
40 PRINT"AS THE PROGRAM ASKS FOR THEM:"
50 PRINT" NOUN:" ;INPUT NS$
60 PRINT" ADJECTIVE DESCRIBING THE NOUN:" ;INPUT AS$
70 PRINT" THE ADJECTIVE:" ;INPUT A$
80 PRINT" A PREPOSITIONAL PHRASE TELLING"
90 PRINT" WHERE OR WHEN SOMETHING CAN HAPPEN TO YOUR NOUN:" ;INPUT P$
100 PRINT" A VERB:" ;INPUT V$
110 PRINT" AN ADVERB DESCRIBING HOW YOUR NOUN DOES IT:" ;INPUT C$
120 PRINT" ANOTHER ADVERB:" ;INPUT D$
130 PRINT" HERE IS THE 'POEM':" :PRINT
140 PRINT" " ;NS$ 
150 PRINT" " ;AS$ 
160 PRINT" " ;A$ 
170 IF Z$="YES" THEN 40
180 END

Children like playing with words and sentences, and in Logo they might continue this theme. For example, a typical Logo project in the 1960s involved some kind of wordplay. In the following example, a child made up lists of nouns, verbs, adjectives, determiners, and so on. She then used a previously defined procedure to pick a word from a list (Papert and Solomon 1971, p. 28).

THE FUNNY PROF TALKED WHILE THAT COOL KID KISSED...
SOME FUNNY PROF WALKED BUT A BEAUTIFUL KID CLAPED...
A WILD DONKEY KISSED WHILE THE FUNNY PROF CLAPED...
SOME GROSS PROF WALKED ALTHOUGH SOME COOL KID HUMMED...

Here are the procedures:

TO SENGEND
SIMPLE.SEN
TYPE PICK [BUT WHILE ALTHOUGH]
SIMPLE.SEN
PRINT []
END
RUN

THIS PROGRAM CAN HELP YOU BECOME A 'POET'.
PLEASE TYPE IN THE FOLLOWING KINDS OF WORDS OR
PHRASES
AS THE PROGRAM ASKS FOR THEM:
NOUN—? SNOW
ADJECTIVE DESCRIBING THE NOUN—? GRAY
ANOTHER ADJECTIVE—? RAGGED
A PREPOSITIONAL PHRASE TELLING
WHERE OR WHEN SOMETHING CAN HAPPEN TO YOUR NOUN—?
BETWEEN THE BUILDINGS
A VERB—? FALLS
AN ADVERB DESCRIBING HOW YOUR NOUN DOES IT—?
SLOWLY
ANOTHER ADVERB—? UNCERTAINLY

HERE IS THE 'POEM':

THE SNOW
GRAY, RAGGED
BETWEEN THE BUILDINGS
SLOWLY, UNCERTAINLY
FALLS.

WANT TO MAKE ANOTHER 'POEM'? NO

Children like developing CAI programs, such as interactive mathematical sentence generators. We are reminded here of Suppes, but in this case the children create their own drills. The first example of this kind of work occurred in a Logo class from 1968 to 1969. Nonetheless, children later developed this style of project in BASIC because it involves numbers and not words. Children might start such a project by printing the generated sentence. Later, they modify the program so that it asks a user to supply the answer. They then concentrate on responding to the user.

The effect is something like

\[ 7 + 4 = 11 \]
\[ 3 + 2 = 5 \]
\[ 9 + 6 = 15 \]

and so on. A slight modification will cause the computer to print something like

\[ 7 + 4 = ? \]

and wait for a human victim to type something in order to insult him if he fails to give the appropriate answer. For example:

\[ 7 + 4 = ? \]

ELEVEN

IDiot, THE ANSWER IS 11

(Computer)

(Victim)

Even when the [program] has been modified to accept "ELEVEN" we can still tease the victim:

\[ 7 + 4 = ? \]

ELEVEN

DON'T THINK YOU ARE SMART, YOU TOOK MORE THAN 2 SECONDS.

(Papert and Solomon 1971, p.30)

Other popular programming projects are games such as NIM, which involves modular arithmetic, and word games based on Hangman. Again I have taken the liberty of selecting examples from my experiences with Logo classes as well as from Dwyer's repertoire. The projects as developed in Logo could be implemented by children younger than those who develop such projects in BASIC.

NIM

DO YOU KNOW HOW TO PLAY NIM?

<NO

HERE ARE THE RULES: YOU WILL BE SHOWN A COLLECTION OF X'S.

YOU MAY REMOVE 1, 2, OR 3. THE PLAYER WHO TAKES
THE LAST X WINS.

THIS IS PROBABLY TOO VAGUE FOR YOU TO UNDERSTAND,

BUT TRY PLAYING AND

I'LL CORRECT YOUR MISTAKES.

ARE YOU READY?

<I AM

OK. NOW TELL ME THE NAME OF THE FIRST PLAYER.

<JON

NOW THE NAME OF THE OTHER PLAYER

<COMPUTER

HOW MANY STICKS DO YOU WANT TO START WITH?
<THIRTY-ONE
I'M A DUMB COMPUTER. TYPE A PROPER NUMERAL.
<31

JON TO PLAY.
THERE ARE 31 STICKS.
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
JON, TAKE 1, 2, OR 3
<3

COMPUTER TO PLAY.
THERE ARE 28 STICKS.
XXXXXXXXXXXXXXXXXXXXXXXXXXX
I TAKE 3

JON TO PLAY.
THERE ARE 25 STICKS.
XXXXXXXXXXXXXXXXXXXXXXXXXX
JON, TAKE 1, 2, OR 3
<3

---

(Papert and Solomon 1970b, p. 14)

Here is Dwyer's version of Hangman (Dwyer and Critchfield 1978, p. 166; © 1978, Addison-Wesley. Reprinted with permission):

RUN
WORD GUESSING GAME
IF YOU GET 8 WRONG GUESSES, THE NOSTER WILL EAT YOU!
WANT MONSTER TO BE VISIBLE? YES

---

GUESS A LETTER? A
---A---

GUESS THE WORD?
NO—TRY ANOTHER LETTER
LETTERS YOU USED:
E, A

---A---

GUESS A LETTER? I
SORRY, NOT IN WORD.
(( ))
( ** ** )
** * **
LETTERS YOU USED:
E, A, I

---A---

GUESS A LETTER? T
SORRY, NOT IN WORD.
(( ))
( ** ** )
** * **
* *
LETTERS YOU USED:
E, A, I, T

---A---

GUESS A LETTER? O

---A---

GUESS THE WORD? PAYLOAD
RIGHT!!! YOU TOOK 5 GUESSES.
WANT ANOTHER WORD
These ideas lead to other more-complex programming projects that elementary school children often use rather than write. For example, in Lemonade the user pretends to run a lemonade stand and, within the constraints of the program’s economic model, makes a profit or suffers a financial loss. The user is asked to price the lemonade and to specify the amount, and the computer does the rest.

Finally, there are some examples of elaborate design plotting programs (figure 4.2).

**History of Project Solo**

Project Solo began in 1969. Terminals connected to a time-shared computer were placed in a large high school. The project concentrated on preparing curriculum materials, developing courses for teachers, and providing expertise on technical issues. From 1972 to 1977, Dwyer ran the Soloworkers Laboratory Project. In this new research phase, the laboratory extended its concerns of interactive computing to exploring various input/output devices including “computer-controlled robots, lunar landers, a computer-controlled pipe organ, a flight simulator, plotters, and color graphics” (Dwyer and Critchfield 1982, p. 8). In 1980 the Solo/NET/works Project began; this project extends the previous work by developing a microcomputer network and software for “running role-playing simulations” and letting users “write their own multi-process simulations” so that the “system could support inventive pedagogies as well as inventive learning” (Dwyer and Critchfield 1982, p. 8).

Although Project Solo work has focused primarily on junior and senior high school students and college students, its use of computers in education has influenced computing activities in elementary schools now, with individual teachers introducing computers to their students as “personal intellectual tools.” The microcomputer has enhanced this possibility of computers becoming personal tools because computer costs have decreased. Because most microcomputers come equipped with the BASIC programming language, a bit of BASIC’s and Project Solo’s histories might shed some light on current activities.

The Project Solo view of computers extends the tradition begun at Dartmouth College, where BASIC was developed in a time-sharing environment in the 1960s. There, every calculus student learned to write six programs in BASIC. John Kemeny, then head of Dartmouth’s math depart-
ment along with Tom Kurtz, designed BASIC specifically to serve the needs of Dartmouth students in this era preceding the availability of microprocessor based hand-held calculators (Kemeny and Kurtz 1967). Project Solo carries the image of computers as personal tools beyond the calculus or algebra classroom to everyone interested in learning to use computers. In schools today this appeal is echoed by many of the active computer users who want to go “beyond CAI” (Critchfield 1979, p. 18). But the model of the project initially follows a pattern of other projects of the 1960s involving high school students, for example, at Bolt, Beranek and Newman, using TELCOMP, and at Dartmouth, using BASIC. Dwyer, who joined the young computer science department of the University of Pittsburgh in 1968, initiated Project Solo shortly thereafter under a grant from the National Science Foundation. Project Solo then involved three high schools in the greater Pittsburgh area.

This and subsequent projects carried on by other researchers in other parts of the country either offered in-service workshops during the year or ran summer institutes for teachers, in which the teachers would learn to program in an algebraic language. In the fall following the summer institute, a number of terminals would be installed in the high schools and the teachers
would then use them to enhance and supplement their regular math or science curricula. The project staff would make on site visits, run in-service workshops, observe the classes, make suggestions, help to prepare materials (new programming projects), explain new programming ideas, etc. The teachers and their students were major collaborators in the project initiating programs of their own out of their own interests. The games, simulations, etc. that owe their origin to high school teachers and students are considerable (See Ahl, 101 BASIC Computer Games and Braun’s Huntington I materials as examples.)

These, like so many other research projects, tend to attract and involve the “top” teachers and the “top” students, although anecdotes abound about mediocre students who are fired up by the computer’s presence. Nonetheless, students under the influence of these computing environments do things they would not otherwise do. This immediate and dramatic demonstration of student productivity and creativity reinforces Dwyer’s view that computer technology offers a heuristic methodology for “liberating the human potential, and thus the person.”

These projects demonstrate that the computer under human control is a potentially powerful human tool. It can be used in a variety of ways for different purposes. This potential can be realized by anyone, not only by experts at universities or in industry. What is needed is to put the tool in the hands of the people and then to give the people suggestions of things that have already been done. Out of these programs the people will create new ones for their own personal use or will reappropriate some of those created by others for their own personal amusement, edification, and pleasure.

Theory and Practice: An Inherent Contradiction

Up to this point I have concentrated on presenting Dwyer’s attractive educational philosophy. I turn next to a discussion of some of the difficulties of the task of implementing these ideas. In doing this, I have tried to avoid ad hoc criticisms and concentrate on a fundamental and almost universal dilemma of educational innovation. The innovator tends to be—and Dwyer certainly is—action oriented. But reality sets a trap for the revolutionary activist: The most utopian visions of the future can be undermined when the means to achieve them are borrowed from the systems they wish to replace. Dwyer himself warns of this danger:

attempting to innovate with supportive systems that don’t begin to match the sophistication of the human learner should be viewed as a betrayal, not a consequence, of a humanistic approach to education. (Dwyer 1971a, p. 100)

But I believe that Dwyer falls into a fundamental inconsistency in failing to take his own advice: By adopting the easily available BASIC as the computer language, he undermines his own utopian vision. Understanding how this happens requires a closer look at the large community of BASIC users and at the language itself.

BASIC and Its Community

As microcomputers equipped with BASIC began to find their way into people’s lives, there was an increased sense of personal power to turn the computer into a tool for work and play. A proliferation of magazines and primers appeared in bookstores, on newsstands, and in computer stores directed at this wide community of users, telling them what they could do and how they could get started. Most of these magazines and books described programs written in BASIC. Educational software packages made available for the new generation of computers was often written in BASIC. These software packages ranged from drill-and-practice materials in the style of Suppes to lessons derived from Plato-like materials. There was also a large number of simulations and games developed by teachers, researchers, and children in various BASIC programming environments.

Some of the people behind the primers, programs, and other published materials available by 1980 have been outspoken advocates of computers in education since the 1960s, when they used BASIC on time-shared computers. Another common element in this culture was (at least theoretically) a feeling that the ideas and programs—the materials, the courseware—were created or could be created by teachers in the field and students in the classroom or at home. (For example, Ahl’s 101 BASIC Computer Games, an annotated collection of programs, many of which were written by high school students. The most popular programs in the collection, Hamurabi and Lunar Lander were written by Ahl (1973, 1978).)

Other Activists in the BASIC Culture

I would like to mention some other people who have been outspoken advocates of computing activities in schools: Ludwig Braun, Bob Albrecht, David Ahl, and Arthur Luehrmann.

Ludwig Braun, a professor at the New York Institute of Technology, became known to the education world when he developed a series of simulation materials (Huntington I and II) in the late 1960s and early 1970s. The Huntington Computer Project began while Braun was at the Polytechnic Institute of Brooklyn and was funded by the National Science Foundation. The project engaged high school students in developing and using simulation programs written in BASIC for biology, chemistry, earth science,
mathematics, and social studies. These materials were widely distributed by Digital Equipment Corporation to run on their PDP-8 (the educational computer of the 1970s). The Huntington materials were used extensively in junior and senior high schools (Huntington Computer Project 1971).

A typical example of the Huntington simulations is POLUT, a program on water pollution. In this program the user can vary five different parameters: the kind of body of water, the water temperature, the kind of waste, the rate at which the waste is dumped per day into the water, and the type of treatment of the waste. The parameter options are well defined; the program does not have to make theories. It computes a table of figures and/or a graph displaying for each day the oxygen content and the waste content of the water until the conditions become stable and there are no new observable effects.

Braun contrasts sharply with Dwyer in the extent to which his writing emphasizes the technology as such rather than the principles of educational philosophy. Braun is a “technologist”; he sees computers as the advanced technology of the information age we live in and believes deeply in their capacity to be harnessed to improve educational productivity. He talks about how student dropout rates will decrease and student test scores will increase as computers provide instruction and attention for individual students. He believes in using computers to improve education in traditionally measurable ways through standardized testing. He is not challenging the ongoing educational enterprise but rather seeking ways to deliver it. He is currently developing CAI materials on personal computers for standard high school algebra (Braun 1980).

Braun shares with Dwyer a belief in the computer and a sense of immediacy—of using computers and programming languages that are available now. Like Dwyer, he believes that the computer can potentially meet the diverse needs of teachers and students and that doing so requires developing materials. The first difference between them is seen in their ideas of what constitutes “materials.” Braun’s materials are tightly defined and clearly related to specific traditional subject matter. Dwyer’s “materials” tend to be environments in which students take their own paths. This difference is rooted in their respective views of educational philosophy. Dwyer sees the computer as a way of liberating people to reach their potential in a humanistic tradition based on realizing one’s own creative powers through learning to control and not be controlled by computers. Braun takes a more technocratic view of using computers to educate people according to current societal demands.

Braun, like Suppes, recognizes the impact of computers on his own intellectual life. For example, in his public presentations he talks about the positive effects on him, a nonmusician, of his own computer-generated music programs. But, like Suppes, he does not see that sharing these experiences with thousands of students and teachers is either possible or practical with respect to cost and productivity. Thus in his proselytizing Braun appears to be a revolutionary—he wants to see everyone using computers—but in his educational philosophy, like Suppes, he tends to conservatism. Suppes and Braun have a large following for their past work. Both favor materials that are self-contained and intended to teach the students largely independently of teachers present. Moreover, neither the teachers nor the students are required to learn anything about computers except how to turn them on and off.

The real educational impact of computers, as Dwyer and others, such as Albrecht and Ahl, see it, comes from personally using them—programming them. Bob Albrecht, whose readers know him as the Dragon, focuses on elementary education. He started a computer storefront, a drop-in center (in Menlo Park, California, in the early 1970s), in which computing power was made available to anyone dropping in. He also began publishing computer magazines for children, teachers, and hobbyists. He has helped to start several magazines, for example, People’s Computer Company and its replacement, Recreational Computing; Computers and Calculators; and Dr. Dobbs’ Journal. One of his BASIC manuals has been adopted for many of the popular microcomputers. Albrecht has written columns in his and other magazines. He has dedicated many years to trying to give people computing power in any form for personal purposes.

Another activist who has approached computers and education through magazine power and thus realized national distribution is David Ahl. Ahl, founder and editor-in-chief of Creative Computing, designed the magazine to appeal to the home computer and hobbyist audience. The magazine, which ceased publishing in December 1985, was large and wide ranging in levels of sophistication. Creative Computing was directed toward people looking for software, courseware, and gameware in print, on tape, on disk, etc. and for “off the shelf” computer hardware. Its goal was to infuse people with an enthusiasm and a sense of belonging to a movement—the personal computer movement—to something larger than themselves. Ahl himself has probably introduced more novices to computing than any other person in the country, as he has been a speaker or a workshop leader or an exhibitor at most personal computer shows around the country.

In the 1980s, a strong spokesperson for BASIC in schools has been Arthur Luehrmann, who had actively used BASIC in courses at Dartmouth. He has been a promoter of standardizing BASIC and of introducing structured BASIC. He left Dartmouth to become director of computing activities at the Lawrence Hall of Science at Berkeley, California, where he expanded this public access place to reach out to schools in the area. He then founded
Computer Literacy, a company that produces textbooks. Luehrmann suggests that BASIC be introduced in the school curriculum in the seventh grade. He has also developed material for Pascal. Luehrmann writes for popular computer magazines in support of programming as an important societal skill (Luehrmann 1980, 1983, 1984).

Thus Dwyer, Ahl, Albrecht, Luehrmann, and others in the BASIC culture have a more romantic and revolutionary vision of the computer’s potential impact on the educational process than either Braun or Suppes do. For Dwyer and some of the others in the BASIC culture, BASIC was the language of choice because it was the language available. As Logo, Pascal, LISP, and other languages become available, their language of choice is in flux.

In what follows I offer some criticisms of BASIC as a language for learning and as a carrier of powerful computational ideas.

**BASIC**

BASIC was designed in the early 1960s by John Kemeny and Tom Kurtz. Kemeny saw the need at Los Alamos and other government projects for fast computing by scientists and engineers, extending the capabilities of slide rules in much the same way that programmable calculators do now. The commands or key words needed for this kind of activity were few. The intention was to maximize the communication between human and computer, man-machine symbiosis, by restricting what was asked of either of them. Arithmetic and trigonometric functions were needed. Stored programs were needed for repeated calculations and error detecting. A control structure for repeating parts of the program was needed (FOR loop). A way of using pieces of the program at certain times was needed (branching with GOTO). Some text manipulation (string handling) was added as a feature but the primary focus was on labeling and performing calculations. BASIC was designed for a specific audience to replace FORTRAN. The audience was to be calculus students, scientists, or engineers who needed to compute complex series of calculations repeatedly.

In the early 1960s Kemeny negotiated for Dartmouth a business agreement with General Electric, on whose computer the Dartmouth Time-Sharing System (DTSS) was built. The time-sharing system was designed around BASIC as the programming language. Digital Equipment Corporation and other computer companies began to offer BASIC on their computers. Kemeny and others at Dartmouth established BASIC groups such as Project COMPUTe and later CONDUIT now centered at the University of Iowa. The user community grew.

There are now committees to standardize BASIC. CONDUIT offers programs written in BASIC as annotated packages for university courses.

The Minnesota Educational Computing Consortium (MECC) offers "debugged" programs in BASIC for the Apple and other microcomputers for elementary and secondary school children and teachers. Computer stores sell different dialects of BASIC and programs written in them to consumers. I mention these business arrangements and organizational support mechanisms because they are part of what has helped institutionalize and spread BASIC and some of the considerations people use in joining the community of BASIC users. Nonetheless, BASIC was designed in reaction to batch-processing FORTRAN and was intended to be as efficient as possible in terms of computer considerations of space and time. Human time was cheap by comparison, and yet part of BASIC’s cost effectiveness lay in its specificity and accessibility. It was targeted to be a powerful programmable slide rule.

BASIC is advertised as a language that has few primitives (commands or key words) and so can be learned quickly. Students can quickly learn to use it as a desk calculator. They could convert algorithms they already understand in their math or science courses to BASIC programs. But what of the people who do not understand the math and science algorithms? What of the people who want to use the computer for nonnumeric programming? In partial response to these questions and partially as a reflection of other views of how to harness the computer, people have been developing simulations in which there is less emphasis on the user becoming proficient at programming and more emphasis on developing convivial tools for the user to explore a particular problem domain—simulations of economic models in a more sophisticated way than, for example, the Huntington materials. For many microcomputer enthusiasts the way to build these environments is to have a cluster of computers networked to one another and to a central database. Dwyer, for example, began to develop simulation environments that his students could explore. The students could also change and extend the scope of the simulation by using extended BASIC. Dwyer’s Solo/NET/works Project endeavors to develop a multicomputer system to support "inventive learning" (Dwyer and Critchfield 1982).

**Critiquing BASIC as a Carrier of Powerful Ideas**

Microcomputers offer users more computing power than what most people had on large time-shared computers. Television sets are used for displaying conversations with the computer and also for displaying video games with moving pictures. Students want to write their own programs to make video games or display pictures. In a procedural language such as Logo, students can apply problem-solving techniques in the style of Polya—breaking
problems into simpler components, doing what you know how to do, and so on. These pieces cannot be done procedurally in BASIC because BASIC is not a procedural language. The pieces cannot be debugged, named, and then forgotten until they are needed, as they can be in Logo, or proceduralized, as they can be in Pascal and Logo. In BASIC, pieces of code are referred to by line number, and the flow of the program can be altered by conditional statements; but only one program can be in the work space at any given time, and adding new features to the language is not possible for a beginning programmer.

Dwyer wants novices to be able to use computers in sophisticated ways. Developing the necessary programming expertise to build a large simulation in BASIC, for example, might take too long. Thus Dwyer would like to move toward developing kits or environments that users interact with, extend, or use to build larger programs. Because it is not procedural, BASIC does not make building such kits for users easy.

There is another problem for classroom teachers that is causing many of them to look toward the development of materials. Teachers are finding BASIC difficult to use with their students. Sometimes the attitude is expressed by saying that perhaps, after all, programming is something most people will not be good at or like to do. Often this happens when teachers have not been given a chance to use the computer themselves in an exploratory fashion. Often it happens when teachers think of ways for thirty children to share one computer. Often it happens when teachers are asked to integrate the computer into their regular math classroom activities. Then children are asked to translate algorithms from their math texts to a computer language. The problem here is that if the child does not already know the algorithm, working on the computer does not help much. The computer is an incentive, but often with this kind of problem the child does not know if the program is doing its job.

One of the problems teachers have in teaching children to program is that they emphasize learning the vocabulary and grammar of a programming language instead of emphasizing the process of exploring ideas. For the most part, teachers are not experienced in introducing computational ideas to the children. They do not have in mind examples of programming activities or experience in helping children to pursue their own interests and in using the language to express their own ideas. These problems are more apparent in BASIC programming environments because BASIC is an algebraic, non-procedural language; the language does not lend itself to thinking about building user tools in it. It does not allow users to add easily new key words to those already in the language.

Some difficulties in BASIC programming come from its algebraic nature and from the kinds of examples for which it is helpful, such as MULT.TABLE. There are many directions that a programmer can take, but there is still the question of how a child becomes a solo programmer. These projects, whether they are written in BASIC or Logo, have a common flaw for beginning students: They are confined to translating algorithms into a typewriter environment. Dwyer responded to this problem in his laboratory by attaching devices to the computer for plotting or making music. Dwyer worked primarily with junior and senior high school students and college students within the context of doing algebra or physics; the deficiencies of the algebraic language BASIC pale in comparison with the alienated mathematics of classroom algebra without BASIC programming.

Thus in Dwyer’s discussions there is a conflict between the heuristic procedures and strategies that exist in the head and the programs that are actually written in BASIC. This conflict is being felt within the BASIC community, and there have been several efforts to structure BASIC and make it procedural (Kemeny and Kurtz 1985). The dialects of BASIC popular on microcomputers and in use in schools and homes today are not procedural. The limitations of BASIC as a language for learning about powerful ideas in computer science are being felt. In colleges and universities Pascal, which is a procedural language, is becoming popular. Some believe that the way around the problem is to create more laboratory materials—games, quizzes, drills, etc. Others, like Seymour Papert and Alan Kay, believe that a powerful language for learning should be developed in which children can then grow.

The creators of BASIC, Kemeny and Kurtz, have become vocal critics of BASIC as it has evolved on microcomputers. At Dartmouth during the 1970s BASIC underwent several transformations. But it is the early version of BASIC that was used as a model for microcomputers. Kemeny and Kurtz criticize these implementations for “tiny computers, so that severe compromises had to be made”: Some of these compromises were ugly and violated our design principles for the language. But people got used to them, and many programs were written in these poor versions of BASIC. (Kemeny and Kurtz 1985, p. 55)

Furthermore, they contend that different implementations take advantage of the special hardware features of different computers so that “no two of these implementations are compatible.” The authors go on to say that they “are concerned that a generation of students is growing up learning Street BASIC”:

An apt description, we believe. Vernacular street talk varies from location to location and year to year, and is full of vulgarisms not to be used in polite surroundings.
Unfortunately, the same is true for BASIC. (Kemeny and Kurtz 1985, p. 56)

They also feel "that this is directly relevant to the problem that whereas schools now have hardware, educational software lags far behind."

There have been devastating criticisms of BASIC in the literature. Unfortunately, as it applies to Street BASIC, we agree with them. (Kemeny and Kurtz 1985, p. 56)

The strongly expressed disapproval of microcomputer dialects of BASIC by its creators has come at a time when they have prepared their Structured BASIC for microcomputers; it is called True BASIC. Programmers, implementers of software written in microcomputer dialects of BASIC, dismiss these criticisms. The education community cannot. Street BASIC is becoming the language taught in junior high; it is sandwiched between Logo, which is taught in elementary school, and Pascal, which is taught in high school.

Talking about Dartmouth’s BASIC in the early 1970s, Kemeny and Kurtz say:

As we entered the decade of the seventies, there was scant warning about the computing revolutions brewing. We had settled back to enjoy our new version of BASIC the Sixth. Little did we realize the impact that fancy graphics and structured programming would have by the end of the decade. And there was little hint that within a dozen years we would have computers sitting on our desks that would be as powerful as the huge central time-sharing system that served us so well. (Kemeny and Kurtz 1985, p. 39)

This statement seems a bit ludicrous in light of the Logo work going on in Cambridge, Massachusetts, in the late 1960s and early 1970s.

**Conflicts in Conveying Powerful Ideas**

In this section I discuss a particular programming project (Dwyer 1977b; Dwyer and Critchfield 1978; Critchfield 1979). The program raises important issues about the effects of BASIC programming on good programming style and on making use of powerful ideas from computer science.

The program consists of five nested FOR loops. The suggestion is made that this problem "is also related to the important idea of tree structures" (Dwyer and Critchfield 1978, p. 60. Here is the statement of the problem:

**THE HOT DOG PROBLEM**

Suppose you’re running the hotdog stand at your next club picnic, and you decide to post a computer printout showing how to order all the possible combinations by number. Let’s assume that there are only YES/NO decisions allowed for hot dog, bun, mustard, mayonnaise, and catsup. To discourage overindulgence, we’ll also print a calorie count for each combination.

The way to think about this problem is to picture what’s called a decision tree.

One way to generate a tree structure in BASIC is to use nested FOR loops, one for each level. Our tree will have five levels (one for each ingredient) so there will be five FOR loops. Here’s how all the paths through our five-level tree can be tabulated with a BASIC program.

```
LIST

10 PRINT " " DOG BUN MUST. MAYO. CATSUP"
15 LET X=1
20 FOR H = 0 TO 1
30 FOR B = 0 TO 1
40 FOR M = 0 TO 1
50 FOR Y = 0 TO 1
60 FOR C = 0 TO 1
70 PRINT"#":K:"": ";
80 PRINT H:" "B:" M:" Y:" C;
90 PRINT "CALORIES=":H*140+B*120+M*20+Y*100+C*30
95 LET K=K+1
100 NEXT C
110 NEXT Y
120 NEXT M
```
This program, as an example of a decision tree, is misleading. One reason for using a tree structure is to take advantage of its dynamic nature. The user might not know or care initially about the size of the structure, that is, whether there are two levels or five levels. What the user cares about is the relationship among the data and a description of how you get from one path or node of the tree to another.

In this problem, the choices are independent of one another. Prior decisions do not influence the current decisions. For example, all the nodes at level 2 represent decisions about buns, and level 3 nodes represent mustard, regardless of whether a bun was chosen or not. The paths are constrained.

There is an additional sense of artificiality to the problem: The program depends on the fact that this binary tree is balanced. That is, its left side or branch is the same as its right branch. But one of the powerful ideas behind tree structures are that they do not have to be balanced. The programmer can abstractly describe how to traverse a tree knowing only that it is a binary tree.

The general characteristic of trees is that they are hierarchical representations of data. There is a logical ordering to the information contained in the tree. Thus it matters when the program makes a choice, unlike the problem posed here. It really does not matter which item is chosen first or second, etc. There is an assumption that the ordering of hot dog, bun, mustard, mayo, and catsup is important to the problem, but the outcome is the same. All possible combinations will be printed. This leads to a further confusion when the “decision tree” idea is introduced. A decision tree is used to help find optimal paths, but here we are not told to do that.

In this well-intentioned attempt to relate the BASIC control structure of FOR loops to tree structures, several issues are raised about the relationship between programming and ideas from computer science to enhance problem solving. FOR loops cannot “generate a tree structure.” They can simulate certain results. Following paths in a tree requires two actions—going down a branch from a node and going back up the branch to the node (backtracking) so that a new branch of the tree can be followed. FOR loops are unidirectional; the flow of the program can go up or down but not both.

The essentially recursive nature of trees is also ignored. By definition, a tree is composed of other trees or subtrees. Thus the program should be able to operate from anywhere in the tree, but doing this requires backtracking, and FOR loops do not allow that. This program always starts at the topmost or root node and then follows a path to its terminal node. Thus, although the program prints out thirty-two paths and their respective calorie counts and although these paths could be traced out on a tree, there is no mathematical connection between the program and the ideas it is meant to represent. These are two parallel but different processes. The program never back-
tracks. In other words the program goes outside of the intended data representation; it does not trace a path and then retrace as it generates new branches but instead computes thirty-two separate routes one at a time.

Although the program does not make use of tree structures, Dwyer does. In his mind he solves the problem using a binary tree. For Dwyer, a computer scientist, procedural thinking is part of his "heuristic strategies." In Dwyer's mind and in his planning with pencil and paper, procedures are definite entities, but in BASIC they do not exist.

In programming cultures like those of LISP, Pascal, and Logo, in which procedures and hierarchical structures have been given concrete identity, programmers find powerful metaphors in tree searches and in recursive processes. There is a tendency to anthropomorphize, to look at control mechanisms among procedures and within the flow of procedures in terms of actors or demons, or other creatures resident in the computer capable of giving advice, passing data, receiving data, activating procedures, changing procedures, etc. As we will see in chapter 5, this is a phenomenon of the Logo computer culture, and it is common to other computer cultures in which recursive and procedural thinking is embedded in the language. It is this aspect that Davis finds so attractive and useful in applying to mathematical behavior. In the Logo culture the intention is to share these and other powerful ideas with children as they develop.

Programming style covers a large area, but the one of most concern here is not readability of code but extensibility and clarity of thought, that is, whether or not the programmer can build on what he or she has done. This might take the form of adapting a program to other purposes or extending the programming idea to enhance everyday problem-solving activities. One idea that comes immediately to mind is giving objects names so that they can be referred to easily; this extends itself to using programs to create other programs, to being able to think of a program as a piece of a job, a subprocess, etc. Another idea involves communicating between programs, passing data—again something that is possible to do in procedural languages. Another powerful idea having to do with organizing thoughts is thinking of different kinds of information structures—data structures such as simple lists, arrays, stacks, queues, list structure of different kinds, tree structures, and even the programs themselves.

Procedural languages such as Pascal and Logo give programmers an opportunity to think about these structures and to create their own and use them. BASIC, on the other hand, is not procedural and does not have list structures; its objects are organized linearly as simple lists or strings or in multidimensional arrays. But clever programming can often bring about the same effect as a program written in Logo or Pascal. The previous discussion of a BASIC programming project shows some of the difficulties that a

restricted programming language can create and that good programmers almost automatically compensate for; that is, they find ways to code around or reconfigure the problem.

Eclecticism

The eclectic nature of the BASIC culture is reflected in the belief that computers can be anyone’s tool for any number of purposes. This can be achieved by adding appropriate gadjetry (hardware to the computer so that it can be extended to music, art, physics, etc. and by debugging programs by either removing bugs or coding around them. There seems to be a gap that this eclecticism does not resolve. How do we introduce children and other novices to this learning environment? What theoretical framework can we operate from? That is, as teachers, what insights into children’s acquisition of knowledge does this learning environment offer?)

Dwyer’s teaching strategies are deeply tied to one principle: respect for the individual. This principle is supported in different ways. For example, in dual mode learning the teacher respects the uniqueness of the individual and does not try to impose his or her own way of doing things because it does not help the learner; instead the teacher helps the student develop his or her own method.

This process can be approached in three different ways that ideally are intertwined. The three approaches are transmittal, experiential, and creative. Transmittal techniques, like those used by Suppes, are totally unacceptable by themselves, according to Dwyer. As one kind of technique at the disposal of the student and thus in the student’s control, it is often useful; but learning takes place experientially. Thus the teacher helps by extending the learning environment into different domains, often by adding new “gadgets.” Writing programs and using them are all part of experiential learning. Creative mode sets in through debugging programs, understanding why they do not work, and then fixing them so that they do work.

For most people, inventive solo learning requires both a supportive social environment, and a supportive physical environment. This means the right kind of guidance, instruction, encouragement, criticism, written materials, time, space, and equipment. (Dwyer and Critchfield 1982, p. 8)

Dwyer concentrates in his discussions on learning on the process of teaching. Dwyer sees the mind as a black box and therefore as something that cannot really be discussed. He has absorbed the ideas of Newell and Simon, Minsky and Papert and others whose theoretical work on thinking has led to new insights into the mind and thus to how knowledge is represented and
how it is acquired. Dwyer believes that there are internal mental structures that are built up by the individual in some way. Teachers using these three different approaches help the students to build their own internal models. Dwyer does not go further.

On this point he differs from Papert and in this difference lies the key distinction between their learning environments. Dwyer’s is built on techniques and gadgets and adapting tools to education, thus following in an eclectic tradition. Papert’s environment is embedded in a theoretical foundation that has its roots in the study of thinking and which molds a computer environment into an instrument for children to use in their intellectual development.

Some Concluding Remarks

I agree with the spirit of what Dwyer says and much of what he envisions, but I have been arguing that his work is blighted by a basic inconsistency. There is an inconsistency in Dwyer’s vision between the environment he wants and the tools with which he chooses to construct this environment. He uses tools created for other purposes than the one at hand, but he chooses them because they are “on the shelf,” and so he does not heed his own advice: “Although I would be the last to advocate designing educational systems by engineering-like formulae, it seems to me that attempting to innovate with supportive systems that don’t begin to match the sophistication of the human learner should be viewed as a betrayal, not a consequence, of a humanistic approach to education” (Dwyer 1971a, p. 100).

For example, in programming he adopts the BASIC language because it is easily available. The significance of this comes from the limited degree of control over the computational medium given by BASIC. In fact, for most children BASIC is so inaccessible that no control is given. BASIC is a primitive programming language that challenges expert programmers to “code around” its limitations. Of course, there is a sense of personal accomplishment when the programmer breaks through these limitations, but the point is that the language that might be easy to learn is hard to use and shuts out aspects of this environment for many people, including elementary school children.

Similarly in the arts Dwyer’s colleague Critchfield writes about “computer experience in the manner of artistic creation” (Critchfield 1979, p. 18). The tools given to the student in this computer experience are a color TV connected to the computer with a light-sensitive pen by which the computer can light up a screen position when the pen points to it. The student uses the pen in conjunction with a program that places points on the screen and then draws lines from those points to ones chosen by the student. The student can select different colors for each of the lines. Thus the student can draw ellipses, circles, sine waves, straight lines, etc. “There are endless possibilities” (Critchfield 1979, p. 21). An alternative to the student is to write his or her own programs using the BASIC commands PLOT or POKE, thus lighting up individual points on the screen.

There is a clear conflict here between intention and the tools at hand. Is this a genuine artistic experience? What kind of art can students achieve with a small microcomputer? Even the art obtained with the best computers is limited. Even at the leading edge of computer technology applied to art, the products are, at best, just beginning to qualify as art. Thus Nicholas Negroponte, a leading figure in aesthetic uses of the computer, a professor of architecture, and a specialist in computer graphics has spoken rather critically of work in computers and art.

Rarely have two disciplines joined forces seemingly to bring out the worst in each other as have computers and art. A mixture of mathematical exercises has predominated in the search for ways to use computers in general and computer graphics in particular for the purpose of achieving a new art form, or simply art, or both. The symmetry and periodicity of the Lissajous figures (easily generated curves on TV screens), transformations into and out of recognizable patterns, and the happenstance of stochastic processing epitomize the current palette of gadgetry used by either the playful computer scientist or the inquiring artist in the name of art. While their intentions may be good, the results are predominantly bad art and petty programming. (Negroponte 1979, p. 21)

Negroponte’s position is, perhaps, extreme; in the 1980s we have begun to see a significant improvement in the quality of computer-generated pictures. The computer graphics used in making films and video are becoming powerful artistic tools. But one thing is certain: In order to produce any approximation to a genuine artistic experience, more was needed than could be provided by a microcomputer experience of the sort “on the shelf” before 1983, when Apple introduced its Macintosh. Macintosh offered users an opportunity to use a painting program similar to the ones built by Alan Kay and his colleagues at the Xerox Palo Alto Research Center in the 1970s. The paint program on the Macintosh does not have color, but it allows the user to mix gray-scaled patterns in a variety of ways.

This discussion points to something else, and that is computer graphics is a multidimensional medium that is visually attractive and intellectually compelling. It has led to the spread of a new mathematics, one based on the computer, of which Papert’s turtle geometry is one example (discussed in chapter 5).
Nonetheless, the gap between the realization of personalizing the computer's power and the tools offered by which to do it is not easily closed in the BASIC computer environment. Often the programmer becomes inured to the difficulties and more enthusiastic over small victories than warranted, thus building bridges that can be seen only by the programmer, not by other people in the environment.