BY SIMSON L. GARFINKEL

Computer whiz Richard Stallman is determined to make software free—even if he has to transform the industry single-handedly.

Programs to the People

"That's a great program; it's just what I need."

"Want me to make a copy for you?"

ACCORDING to the Software Publisher's Association (SPA), more than half of all programs currently in use are illegal copies. SPA estimates that unauthorized copying costs the software industry nearly $2 billion a year in lost revenue. The crooks aren't just pimply-faced pirates or vendors in Southeast Asia copying programs and shipping them back to the United States. Rather, all of us are to blame—small offices buying one copy of a word processor and using it on two computers, or people copying a program from work for use at home. After all, a copy of a program works as well as the original, so why pay?

Richard M. Stallman, president of the Cambridge-based Free Software Foundation (FSF), believes companies that sell programs give their customers the choice of being criminals or bad neighbors. People can break the law by copying programs for friends, or they can force friends to go and buy their own. "Imagine if somebody was going around your neighborhood saying, 'I will give you all of these wonderful things if you promise not to let your neighbors have them,'" says Stallman. "To many people, that person would be the Devil." Six years ago, when he started the work his foundation supports, his motivation was to be part of a software-sharing community in which people can freely give copies of programs to their friends: "I decided that I was going to do it even if I had to write all the software myself."

What might have been an impossible task for anyone else was just a matter of punching in coding for Stallman, who many consider to be one of the world's greatest and most prolific programmers. Already he has helped create dozens of programming tools, many of them vastly superior to their commercially available counterparts, and broad acceptance by users has convinced several companies, such as Hewlett-Packard and Digital, to include his programs with their computer systems. At the forefront of his achievements is EMACS, a powerful program used by hundreds of thousands of people throughout the world. EMACS lets programmers perform an extensive range of tasks—from editing files to playing games—and they can alter it to their own liking and add their own features.

Moreover, the free-software move-

PHOTO: RICK FREEDMAN

ment Stallman has spearheaded is taking off. He has convinced hundreds of programmers to contribute their time and efforts. Most of the programs these people have produced are small improvements to other free programs that are already available, but others have been substantial projects, conceived, developed, and distributed as free software. Also, FSF has attracted more than $600,000 worth of gifts in cash and computer equipment. And last summer, Stallman was awarded a "genius grant" from the MacArthur Foundation in recognition of his work.

Stallman wants to create a family of free software so good that companies who do not use it could be driven out of business. In the process, he hopes to free computer users and return youthful hacker idealism to the computer world.

He may just do it.

Back to the Source

Programs, which allow computers to be word processors today and electronic spreadsheets or payroll-printers tomorrow, are something like a cross between a cookbook recipe and a mathematical proof. Each line of a program contains a set of instructions for the computer to execute at a certain time; around the instructions are comments that explain how the program works. Programmers call the collection of instructions and comments the "source code," and in the early days of computing, companies almost always provided it with the programs they sold. Programmers read the code to learn how programs worked and modified it to fix problems and add features. They even built new programs by taking parts from old ones and reassembling them.

But as the business of computing exploded in the 1970s and 1980s, companies began restricting access to source code so that competitors couldn't see how a program worked and write their own versions. Richard Stallman thinks that was a big mistake. Making source code available again is key to his free-software movement. He likes to explain why it's so important by telling the story of the first two laser printers at the MIT Artificial Intelligence Laboratory, where he was a researcher from 1971 until 1983.

The laser printers of the mid-1970s were the size of today's compact cars. When Xerox gave the AI lab a Xerox Graphics Printer, the only place for it was in the lab's ninth-floor machine room. Researchers connected the printer to the local area network that the lab was developing, and soon anybody in the building could print a 100-page document by typing in a few commands.

That worked fine, except that sometimes the printer would run out of paper or jam, and dozens of other jobs would pile up. Other times there would simply be a lot of people wanting to print long documents, and the person who needed to print a single page would have to run up and down the stairs or babysit the printer until that page appeared. But since the programmers at the lab had the source code to the program that ran the printer, they could add features that solved these problems. Soon the printer was helping the lab run smoothly. "It would send you a message when your document had actually been printed," recalls Stallman. "It would send you a message if you had anything queued and there was a paper jam."

All this changed in 1978, when Xerox replaced the machine with a new laser printer called a Dover but wouldn't share the printer's source code with the lab. "We wanted to put those features into the Dover program, but we couldn't," Stallman says. Xerox wouldn't put the features into the program either. "So we had to suffer with paper jams that nobody knew about."

Keeping source code proprietary hurts users in a wide variety of other ways as well. Say a real estate company with an accounting-system program that allows for 10 checking accounts suddenly finds itself in charge of 13 properties. The program may not be able to handle the additional accounts, and if the company doesn't have the source code, it will either have to change accounting practices or find a new program. If the real estate firm lacks the source code, it may not even be able to hire an outside programming firm to make the necessary changes. "It is a monopoly because only one company can provide you with fixes or updates or changes to that program," says Robert J. Chassell, FSF's treasurer. "It's like you bought a car but there was only one mechanic who was permitted to work on it, and he lived in another city. Americans and American law have been against monopolies for years and for good reason—it is bad for both industry and the public."

Although consumers theoretically have the choice of being able to buy a different program, that choice is often illusory. "People have spent money on a specific program, but more significantly, they have become habituated to it," explains Chassell, who was trained as an economist at Cambridge University in England. "The expense of changing to a new program is not buying it: the expense is unlearning one program and relearning a second one."

To add to the cost, most programs store their data files in a format that is not compatible with competing programs. Most people, says Chassell, will put up with two or three major problems with a program rather than make a change.

SIMSON L. GARFINKEL is a free-lance science writer and a doctoral candidate in the MIT Media Laboratory. He is writing a book about Richard Stallman and Project GNU.


DON'T

During the 40 years of the Cold War,

there was always money to end lives but seldom enough to improve the quality of life.

lation programs overseas. The measure would have earmarked $60 million or 16 percent of the total authorization, whichever is less, for the UNFPA. The no-strings-attached multilateral approach of the UNFPA has won the trust and confidence of the entire developing world. For instance, UNFPA provides population assistance to 140 developing countries, while the United States provides bilateral aid to 40. Only through a coordinated, concerted effort encompassing multilateral and private voluntary organizations, as well as bilateral programs, will population growth be significantly slowed.

Time and again, Congress has demonstrated its comprehension of the consequences of overpopulation and its dedication to bringing population into a better balance with global resources. Indeed, Congress showed its concern when it voted to resume the U.S. contribution to the UNFPA before President Bush's veto scotched it. For the sake of poor people throughout the world who are virtually forced to have babies they do not want, regardless of the risk to the lives and health of both the children and their mothers, the United States must not evade its responsibility.

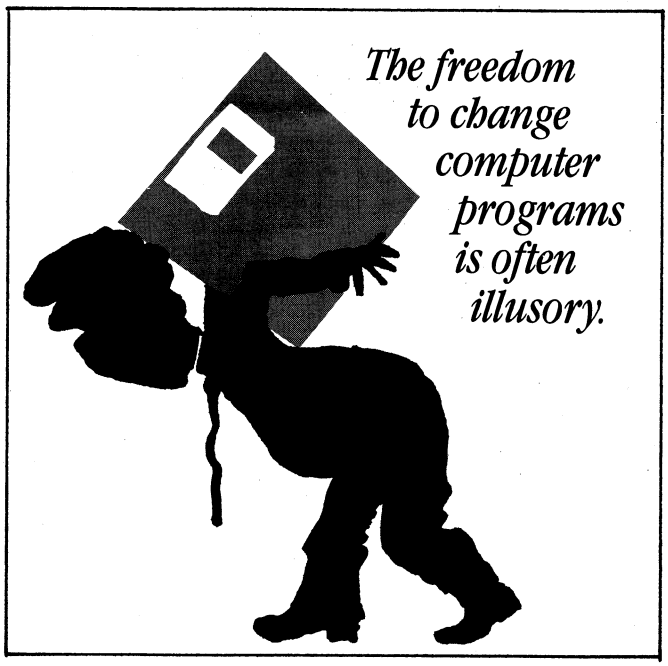
We could demonstrate our resolve not only by increasing population assistance but also by consolidating the 48 separate entities within AID's Office of Population into a centralized organization. The Office of Population operated much more efficiently and effectively in the 1970s, when it was a centralized unit. During that decade, fertility declined more dramatically in several countries receiving U.S. population assistance than it has since. In addition, the president could appoint an ambassador-at-large for population matters, a career diplomat committed to the idea that reining in population growth is essential to a nation's development.

During the 40 years of the Cold War, somehow there was always money to end lives but seldom enough to improve the quality of life. Even given the high priority placed on reducing the national debt, the United States can certainly afford more than the current $300 million for humanitarian family-planning aid in a foreign assistance budget totalling $15.5 billion. While such aid is frequently overlooked or ignored, there is no better hope for global stability and security than establishing universally available, voluntary family-planning services, slowing the rate of population growth, and reducing the destruction of the world's resources. ■

The Rise of UNIX

Until recently, people had the same problems switching between computers made by different companies that they have today switching between different application programs. The problem had to do with the operating system, the master control program that orchestrates the functions a computer performs: every computer had a different one, and all of them were incompatible. Computers made by IBM used an operating system called VM, while those made by Prime used PRIMOS. Digital Equipment Corp. had a variety of different operating systems—sometimes more than one for each computer that it sold.

For the hardware manufacturers, this was good business, because even if a company lost its competitive edge, it would still have a captive base of users who would have to keep buying its computers to run their old programs. And these users could be counted on to pay just about anything the company asked. From the users' point of view, this state of affairs was simply a fact of life. It added to costs, and there was nothing they could do about it. But for computer researchers, such "closed systems" were a nightmare. If someone developed a program on one computer, those who had other kinds of machines had no access to any of the research that person had done.



Today, "open systems," which let users mix hardware and software components built by different vendors, are changing the computer industry. Compatibility makes more services and products available, while competition cuts prices. Open systems are, in fact, central to Stallman's mission to liberate software, though he can hardly be credited with originating the idea. At its core is a special operating system called UNIX and a programming language called C, both developed at Bell Labs in the 1970s.

UNIX, a pet project of AT&T researchers Ken Thompson and Dennis Ritchie, evolved into a programmer's dream. The system was composed of compact programs called tools, each of which performed a single function. By putting tools together, programmers could do complicated things. The operating system mimicked the way that programmers think. C, the programming language UNIX programs were written in, was created by Ritchie expressly to make them "portable"—that is, able to run on different computers. And unlike other portability schemes under development at the time, C was designed to be sleek, simple, and fast.

Nevertheless, problems remained. UNIX, which the AT&T researchers had developed on DEC computers, handled data a little differently than the operating system on IBM computers—which meant that UNIX programs, even if they were scrupulously written in C, didn't always work on an IBM. The Honeywell operating system was a little different still, creating a whole new set of obstacles. Programs that worked on one machine would mysteriously fail on others.

Then somewhere around 1976 Thompson and Ritchie made a breakthrough. They decided that although writing their programs in C was certainly a good idea, it wasn't enough. What they really needed to do, they reflected, was to move UNIX itself—after all, an operating system is just another program, and users could simply run UNIX instead of the system the computer manufacturer had supplied. It was a radical idea in an age when every computer and its particular operating system seemed to be inextricably linked.

By this time, UNIX had become more than just a research curiosity. As early as 1973, some 25 Bell Labs computers were running it, and the operating system soon spread outside of the telephone company. By 1977, more than 500 sites were using it, 125 of them at universities, among them the University of California at Berkeley.

UNIX took a new turn at Berkeley that shows just how much can be done when a source code remains available to users. Like other schools, Berkeley had paid $400 for a tape that included the complete source code to the operating system. But instead of merely running UNIX, two bright graduate students, Bill Joy and Chuck Haley, started making changes. In 1977, Joy sent out 30 free copies of the Berkeley Software Distribution (BSD) UNIX, a collection of programs and modifications to the UNIX system.

ILLUSTRATIONS: ART SPIKOL

Over the next six years, the BSD UNIX grew into an operating system of its own that had significant advantages over AT&T's. For example, a programmer using the BSD UNIX could switch between multiple programs running at the same time. AT&T's UNIX allowed the names on files to be no more than 14 letters long, but on Berkeley's they could stretch out to 255 letters. Berkeley also developed software to connect many UNIX computers together using high-speed networks. If there had been a popularity contest between the two systems, the BSD UNIX would have won hands down. And Berkeley never charged more than a modest duplication fee for its software.

Yet Berkeley didn't make a dent in AT&T's sales: since the university's system was based on UNIX, anybody who wanted to run it first had to purchase a source-code license for UNIX from AT&T. What's more, the company was beginning to realize the true value of the operating system it had spawned. In 1977, a commercial source-code license for UNIX cost $17,000, but by 1981, that price had jumped to $43,000.

Educational source-code licenses for UNIX were still under $1,000, so many universities bought the AT&T license, put the system that went along with it on the shelf, and ordered the BSD UNIX from Berkeley. But the businesses that were turning to UNIX couldn't justify spending tens of thousands of dollars just for a source code. Instead, they spent the few hundred dollars AT&T charged for versions of UNIX that didn't include the code. These firms couldn't make changes or see how programs were written, but they could still write their own applications.

Software War

Back at MIT, Richard Stallman and the AI lab had had their own brush with commercializing software—with very different results. In the late 1970s, the lab was peopled with students, professors, and staff that had drifted in during their high school or college days and never left. This tightly knit community of hackers seemed to live for programming alone. In many ways, what united them was that the lab had built its own computer, the Lisp Machine, and a whole new operating system designed for AI applications.

Progress in developing software for the Lisp Machine was swift: whenever somebody discovered a bug, it was fixed. If people wanted to add a feature to a program—make it do something useful that it hadn't done before—they went right ahead.

Encouraged by the academic success of their machine, a group of hackers left the lab in 1980 to set up a company to commercialize the computer. They called it Lisp Machine Inc. (LMI). Soon a second group left and set up a company called Symbolics. Both companies licensed the Lisp Machine operating system from MIT, and a clause in their contracts specified that any improvements they made had to be returned to the Institute. So although competition between the two companies was fierce, they shared everything they learned. Any time anyone made an advance, everyone in the embryonic industry benefited. The hackers at the AI lab saw the cooperation between Symbolics, LMI, and MIT as a model for software development.

Then in 1982, Symbolics' lawyers reread their licensing agreement with MIT and discovered that while they had to give any new software they created back to the Institute, they didn't have to grant MIT the right to redistribute those ideas. Programmers at Symbolics developed a new feature for the operating system and refused to let MIT share it with LMI. Although the feature wasn't in itself a major advance, Symbolics' new policy was the death knell to software sharing.

"Stallman and I went into a crash mode," recalls Richard Greenblatt, the Lisp Machine's inventor. They refused to accept Symbolics' terms, and decided to reinvent the company's new feature for themselves. "We hacked around the clock for two solid weeks and finally put a comparable feature into the MIT sources."

For the following two years, Stallman took every improvement that Symbolics' programmers made and rewrote it for the operating system used by MIT and LMI. Programs that took Symbolics months to write he would rewrite in a matter of days. The only reason he did it, he says, was to punish Symbolics for breaking its promise to share software. He called it "the war."

But while he fought the war, Stallman's beloved AI lab fell apart. All the old hackers slowly left, siphoned off by LMI and Symbolics. "Machines would break and there was no one to fix them anymore—they had to be turned off and abandoned," he remembers. "It was a society that could no longer keep itself going. I was the last one who could keep it going, but I couldn't, because one person wasn't enough."

He also came to realize that his fight had little significance. The evolution of computer systems had bypassed the Lisp Machine, which was too specialized and expensive to produce. Stallman saw that the real enemy was not Symbolics but the entire software industry that was restricting access to source code.

In 1984, he decided that it was time to start a counterattack: "Instead of continuing to punish those who had destroyed the old software-sharing community, I wanted to start a new one." He quit his job at MIT. More than anything else, he didn't want a repeat of the Lisp Machine debacle—spending years on a project just to have it pulled out from under him and licensed to a company on MIT's terms. Then he sat down and started the task of building a new operating system.

Richard Stallman's mission to liberate all software began at MIT's Artificial Intelligence Laboratory back in 1982, when fellow hackers reneged on their tacit promise to share their ideas. Today, the movement he has spearheaded has taken off.

What's GNU?

He called his brainchild GNU, a recursive acronym meaning GNU's Not UNIX.

As early as 1984, UNIX appeared to be on its way to becoming the operating system of the future. It was taking over the computer research world and making strong inroads in commercial computer systems. Versions of it were already available for most computers—from microcomputers to supercomputers—and engineers were rapidly adapting it to others. UNIX could even run on the lowly IBM PC. Stallman reasoned that a free version of the operating system, written completely from scratch, would have a large user base eager to accept it.

But GNU would *not* be UNIX, even though all GNU software would also run on UNIX. Most significantly, the source code for any GNU program would be available to anyone who wanted it, and people would be able to freely redistribute their own copies of the software—both identical copies for friends and enhanced copies, like Berkeley's version of the original UNIX.

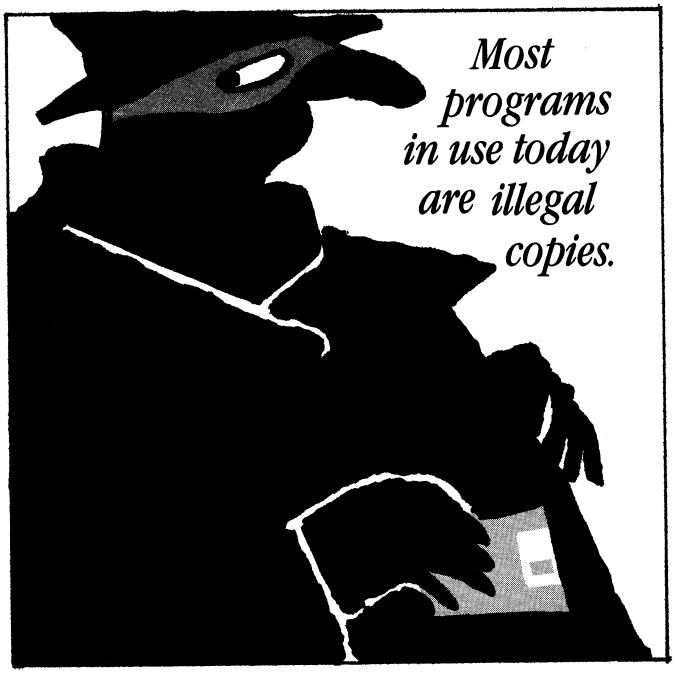
Stallman's main worry was that some company would take the operating system he wrote, make some changes, and then say that their "improved" programs were separate inventions and proprietary. To prevent that, he invented a new kind of licensing agreement, the "Copyleft," which lets people do anything they want with the software except restrict others' right to copy it. As Stallman says, "Forbidding is forbidden." The Copyleft furthermore requires that anybody who distributes a GNU program make its source code available for a nominal fee. And if any piece of a Copylefted program is included into another program, the entire resulting program is Copylefted.

Although Stallman expected that other programmers would eventually help him out with his project, at first he was on his own. When he discovered that nobody else had been assigned to his old office at the AI lab, he started sneaking back at night: he needed a computer to write GNU, and the machines at the lab were available. Soon he was working there days as well. Patrick H. Winston, the AI lab's director, knew about it, but he didn't say anything, since he saw Stallman's resignation as largely symbolic. If Stallman was going to continue writing good programs that other people in the lab could use, Winston wasn't about to tell the 13-year veteran to leave.

PHOTO: L. BARRY HETHERINGTON

Within a year, Stallman's first program was out: GNU EMACS, which edits programs and does a much better job of it than the standard editor that comes with UNIX. EMACS is so powerful that people can use it to write programs, try them out, read electronic mail, browse through online documentation, find programming mistakes with the help of a debugger (also written by Stallman), and even play games. Programmers immediately saw the caliber of the promised GNU software and shared the program with their friends.

And then, just as Stallman had hoped, they started fixing his bugs and adding new features. The hard thing about writing a major program like EMACS, he explains, is starting it. Once the first version is available, people play with it and easily make substantial contributions. By producing just one crop of free software, Stallman bootstrapped a movement that has grown in momentum as the software has improved. Today hundreds of significant subsystems for EMACS have been contributed from around the world, and programmers have adapted it to more than 50 different kinds of computers. It runs on everything from desktop microcomputers to Cray supercomputers.



The success of EMACS led Stallman to found the Free Software Foundation, which gives a tax deduction to companies and individuals who want to contribute to Project GNU. Stallman describes it as "a charity for writing computer programs," and from that perspective, it has been highly successful, receiving $267,782 in donations in 1989 alone. The foundation also earned $330,377 from the sale of manuals and computer tapes containing GNU programs. Moreover, Stallman and the other FSF programmers no longer sneak around to use the AI lab's computers, since they have a fleet of high-performance workstations donated by Hewlett Packard, Thinking Machines, Sony, and even Bell Laboratories. Companies have donated cash as well, and paid for technical staff to spend a year in Cambridge working with Stallman.

The foundation uses the money it garners to pay its staff of fourteen, which includes nine programmers and three technical writers. Even though Stallman works for free, he doesn't expect everybody else to do the same. Nevertheless, FSF programmers earn only $25,000 a year, which is one-half to one-third the salary they would command on the open market. Paying low wages lets FSF take on more staff members, and it guarantees that they're all committed to the cause.

A Programming Coup

In the workstation and minicomputer market, GNU has already caught on strong. Many computer companies that sell UNIX-based systems—including Convex Computer Corp., which makes mini-supercomputers, and DEC—already include GNU software as part of their standard operating-system distribution. Data General and NeXT, Inc., the billion-dollar startup of Apple Computer's founder Steve Jobs, use GNU as the basis of their workstation line. About the only territory that remains untouched by GNU—and by UNIX as well—is the personal-computer market: the UNIX that runs on the IBM PC often costs more than $1,000 for a usable configuration. But the situation is due to change. As soon as the core of GNU is operational, something that Stallman expects before the end of 1991, GNU software will run on any personal computer based on the Intel 386 microprocessor—what is quickly becoming the standard machine—for free.

If EMACS made the computer world suspect that Project GNU was a force to be reckoned with, what clinched the matter was Stallman's second GNU program, something called the GNU C Compiler (GCC). Compilers are those critical programs that translate source code into "machine code," or language that a machine can use. But not all compilers are equal. Given the same source code, different compilers will produce different machine code. A certain compiler may generate machine code that is more efficient than another's, or it may make mistakes, so that its machine code doesn't work properly.

Stallman knew that he had to write a good C compiler; otherwise people wouldn't want to use it. But he didn't intend to write one of the best. Because it is free software, GCC simply *became* one of the best. Stallman

implemented ideas that had been in textbooks for years, and then, since the compiler was distributed with the source code, programmers all around the world helped make it better.

Today the machine code GCC generates is more reliable than that from other commercially available compilers. The reason, say its users, is that people who discover bugs can figure out the fixes themselves by looking through the source code. All the bug reports—and the fixes—end up back on Stallman's workstation. New releases of the compiler come out nearly every month instead of every year, as is the case with most commercial software.

GCC can also generate code for more than 11 different kinds of microprocessors, while most commercial compilers are tailored to a specific microprocessor. Before Stallman wrote GCC, nobody believed a compiler that generated code for more than one kind of machine could be efficient, but Stallman's compiler is efficient indeed: it consistently produces machine code that runs 20 to 30 percent faster than the code from other commercially available compilers.

"The only way for other commercial compilers to continue to exist in the face of GCC is to offer features that GCC does not," says Don Seeley, a senior systems programmer at the University of Utah. "The many vendors whose compilers are not even current with old technology will lose. New compilers must be at least as good as GCC, or the market won't accept them."

It was rave reviews like Seeley's that convinced Ralph W. Hyver, who now manages Hewlett-Packard's Information Architecture Group, to give FSF a $100,000 cash grant and another $350,000 in equipment. Helping Stallman made sense, says Hyver, because many of the research groups that Hewlett-Packard was supporting were using GNU software. The company was also using GNU programs internally.

Another convert is NeXT. All of the software that it delivers with its computers is compiled with GCC. "The issue for us had nothing to do with proprietary versus non-proprietary," says Bud Tribble, NeXT's vice-president of software engineering. "We benchmarked many compilers, and found the GCC code produced to be excellent. The internal structure of GCC was also very clean and allowed us to extend it in several ways. If there had been another 'non-free' compiler that was better, we probably would have used it instead."

Conflicting Definitions of Freedom

Nevertheless, other companies have been reluctant to use GNU software. Some have spent millions of dollars developing their own C compilers and may feel threatened by a compiler Stallman developed essentially by himself. Engineers at Sun Microsystems, for example, refuse to even talk about GCC anymore. "They have all spoken with people about GCC in the past and believe that comparing our compilers with GCC quickly becomes a fairly unproductive philosophical discussion," says Erica Vener, a spokesperson for the company. "Bottom line, Sun is in the business of selling the products it develops."

But ironically, it is probably the Copyleft, more than anything else, that is preventing more widespread adoption of GCC and other GNU programs. Most companies aren't comfortable with the idea of selling a program only to have the customer turn around and make a copy for a friend. And they don't like the requirement that the source code be made available to anybody who asks for it.

At Berkeley, UNIX developer Mike Karels says that the software he writes is actually more free than Stallman's. Since the mid-1980s, Karels and the other researchers at Berkeley's Computer Systems Research Group (CSRG) have been working to isolate their programs from AT&T's. And it has paid off. By now, a "significant fraction" of their code has been "written from scratch," Karels notes. Berkeley gives those programs away to companies that do not have AT&T source-code licenses and imposes essentially no restrictions. The companies, in other words, may modify and resell the software without providing the source code to their customers.

Throughout the 1980s, CSRG developed a set of programs for networking computers. Firms bought the software, sometimes altered the source code and add-

ed features as they saw fit, and marketed the finished product. Today nearly every UNIX manufacturer sells a version of the Berkeley networking software, and some companies have even placed the programs into integrated circuits that are used inside IBM personal computers. Karels says none of that would have happened if Berkeley had required that the networking source code be made available to customers: companies would have been frightened away by the idea that they would somehow lose their competitive edge. And he adds that many users aren't interested in seeing the source code anyway.

Unfortunately, Berkeley's terms also mean that customers who buy Karels's programs from vendors have to rely on the vendors for bug fixes. This matters the most with security problems. In 1988, for instance, the infamous computer worm written by Robert T. Morris got through a hole in Berkeley's network mail program and shut down thousands of computers across the country. The fix, like many security-related fixes, required changing a single line of the mail program, and it was distributed over the network within a few hours after the worm had been stopped. But it was useful only to those schools and businesses that had the source code. Others had to get new versions of the mail program from their vendors, some of whom took more than a month to distribute them.

"We have been pushing for vendors to ship source code for security-critical functions," Karels says. But vendors haven't complied.

The Question of Support

Advocates of FSF believe it is precisely because of the Copyleft that GNU software will eventually dominate the computer industry. And, they say, by voting with their checkbooks, people are already forcing manufacturers to abandon their proprietary operating systems. Given the opportunity to use free software, many computer users might soon refuse to purchase anything else.

The pressure will become even more intense once FSF follows through on its plan to produce a spreadsheet program for workstations and advanced PCs that competes with Lotus's best-selling 1-2-3. Although at first the GNU spreadsheet will lack many of the features of 1-2-3, they will surely be added over time. Soon the only competitive advantage of 1-2-3 will be its name.

But who would pay for programmers to eat if all software were free? The same people that are now, says Stallman. Most programs are written for internal use, not for resale, and that will continue, he argues. A company that pays a programmer to write a word processor for drawing up reports and other such applications shouldn't care if that program is shared with another company—especially if the second company gives bug fixes and improvements back. GNU software will make programmers more productive, since they won't have to write each new application from scratch, Stallman points out. He's looking toward a future in which companies that sell computer programs earn their money not by using the copyright law to prevent people from making copies, but by offering services like support and training. If you had a personal computer, for example, you would pay company programmers to add extra features or help you use the ones already provided.

Naturally, not everyone is enthusiastic about the idea. "It is nice to say that we should just sell support and give away the software, but why?" asks Tom Lemberg, vice-president of Cambridge-based Lotus Development Corp. "The way our economic system works is that people who create value are able to get value by selling it."

Other critics note that in fact product support for GNU software has been lacking so far—and that this could prevent businesses from wholeheartedly adopting the programs. "Digital supports people in mass quantities," says Jon Hall, one of Digital's product managers for ULTRIX Workstation Software. "Thousands of customers at one time. Some of the customers are not even computer literate, much less UNIX literate." He contends that Digital can provide that level of support only by charging for its software and using the copyright system to prevent people from making their own copies.

But companies that exclusively supported free software would have lower costs. Michael Tiemann, who wrote a compiler for the G++ programming language, is banking on that idea: last January he founded Cygnus Support, a firm that writes, sells, and supports Copylefted software. Tiemann believes that wholesale adoption of GNU programs will be inevitable once there's a company willing to sign its name on the dotted line, charge an annual fee, and guarantee to fix any bugs and answer any questions a customer might have. Cygnus is that company.

In its first year of operation, Cygnus signed over a million dollars in support contracts. One of the clients is Intel, which needed a C compiler for a new microprocessor that it has developed. "They want to ship GCC as their standard compiler, but companies that they sell to are concerned that it is not a supported product. So they contracted with us to do the support for it," says David Wallace, another Cygnus founder. "We are also starting to get calls from people whose potential clients are telling them 'if it doesn't run the GNU software, we are not going to buy your hardware,'" he adds.

Wallace acknowledges that it will take years to wean the computer industry away from proprietary software. Yet he maintains that Stallman isn't just a fluke programmer, and that GCC is not just a lucky success. "The free-software part isn't a gimmick," he points out. "It is the very thing that makes the software so good." ■