Real Programmers Don't Use Pascal

Ed Post
Tektronix, Inc.
P.O. Box 1000  m/s 63-205
Wilsonville, OR 97070
Copyright (c) 1982
(decvax | ucbvax | cbosg | pur-ee | lbl-unix)!teklabs!iddic!evp

Table of Contents

- The Real Programmer At Play
- The Real Programmer's Natural Habitat
- The Future
- Acknowledgments
- References

Introduction

Back in the good old days -- the "Golden Era" of computers, it was easy to separate the men from the boys (sometimes called "Real Men" and "Quiche Eaters" in the literature). During this period, the Real Men were the ones that understood computer programming, and the Quiche Eaters were the ones that didn't. A real computer programmer said things like "DO 10 I=1,10" and "ABEND" (they actually talked in capital letters, you understand), and the rest of the world said things like "computers are too complicated for me" and "I can't relate to computers -- they're so impersonal". (A previous work [1] points out that Real Men don't "relate" to anything, and aren't afraid of being impersonal.)

But, as usual, times change. We are faced today with a world in which little old ladies can get computers in their microwave ovens, 12 year old kids can blow Real Men out of the water playing Asteroids and Pac-Man, and anyone can buy and even understand their very own Personal Computer. The Real Programmer is in danger of becoming extinct, of being replaced by high-school students with TRASH-80s.

There is a clear need to point out the differences between the typical high-school junior Pac-Man player and a Real Programmer. If this difference is made clear, it will give these kids something to aspire to -- a role model, a Father Figure. It will also help explain to the employers of Real Programmers why it would be a mistake to replace the Real Programmers on their staff with 12 year old Pac-Man players (at a considerable salary savings).

Languages

The easiest way to tell a Real Programmer from the crowd is by the programming language he (or she) uses. Real Programmers use FORTRAN. Quiche Eaters use PASCAL. Nicklaus Wirth, the designer of PASCAL, gave a talk once at which he was asked "How do you pronounce your name?". He replied, "You can either call me by name, pronouncing it 'Veert', or call me by value, 'Worth.'" One can tell immediately from this comment that Nicklaus Wirth is a Quiche Eater. The only parameter passing mechanism endorsed by Real Programmers is call-by-value-return, as implemented in the IBM/370 FORTRAN G and H compilers. Real programmers don't need all these abstract concepts to get their jobs done -- they are perfectly happy with a keypunch, a FORTRAN IV compiler, and a beer.

- Real Programmers do List Processing in FORTRAN.
Real Programmers do String Manipulation in FORTRAN.
Real Programmers do Accounting (if they do it at all) in FORTRAN.
Real Programmers do Artificial Intelligence programs in FORTRAN.
If you can't do it in FORTRAN, do it in assembly language.
If you can't do it in assembly language, it isn't worth doing.

Structured Programming

The academics in computer science have gotten into the "structured programming" rut over the past several years. They claim that programs are more easily understood if the programmer uses some special language constructs and techniques. They don't all agree on exactly which constructs, of course, and the examples they use to show their particular point of view invariably fit on a single page of some obscure journal or another -- clearly not enough of an example to convince anyone. When I got out of school, I thought I was the best programmer in the world. I could write an unbeatable tic-tac-toe program, use five different computer languages, and create 1,000 line programs that worked. (Really!) Then I got out into the Real World. My first task in the Real World was to read and understand a 200,000 line FORTRAN program, then speed it up by a factor of two. Any Real Programmer will tell you that all the Structured Coding in the world won't help you solve a problem like that -- it takes actual talent. Some quick observations on Real Programmers and Structured Programming:

- Real Programmers aren't afraid to use GOTOs.
- Real Programmers can write five page long DO loops without getting confused.
- Real Programmers like Arithmetic IF statements -- they make the code more interesting.
- Real Programmers write self-modifying code, especially if they can save 20 nanoseconds in the middle of a tight loop.
- Real Programmers don't need comments -- the code is obvious.
- Since FORTRAN doesn't have a structured IF, REPEAT ... UNTIL, or CASE statement, Real Programmers don't have to worry about not using them. Besides, they can be simulated when necessary using assigned GOTOs.

Data structures have also gotten a lot of press lately. Abstract Data Types, Structures, Pointers, Lists, and Strings have become popular in certain circles. Wirth (the above-mentioned Quiche Eater) actually wrote an entire book [2] contending that you could write a program based on data structures, instead of the other way around. As all Real Programmers know, the only useful data structure is the Array. Strings, Lists, Structures, Sets -- these are all special cases of arrays and can be treated that way just as easily without messing up your programming language with all sorts of complications. The worst thing about fancy data types is that you have to declare them, and Real Programming Languages, as we all know, have implicit typing based on the first letter of the (six character) variable name.

Operating Systems

What kind of operating system is used by a Real Programmer? CP/M? God forbid -- CP/M, after all, is basically a toy operating system. Even little old ladies and grade school students can understand and use CP/M.

Unix is a lot more complicated of course -- the typical Unix hacker never can remember what the PRINT command is called this week -- but when it gets right down to it, Unix is a glorified video game. People don't do Serious Work on Unix systems: they send jokes around the world on UUCP-net and write adventure games and research papers.

No, your Real Programmer uses OS/370. A good programmer can find and understand the description of the IJK305I error he just got in his JCL manual. A great programmer can write JCL without referring to the manual at all. A truly outstanding programmer can find bugs buried in a 6 megabyte core dump without using a hex calculator. (I have actually seen this done.)

OS is a truly remarkable operating system. It's possible to destroy days of work with a single misplaced space, so alertness in the programming staff is encouraged. The best way to approach the system is through a keypunch. Some people claim there is a Time Sharing system that runs on OS/370, but after careful study I have come to the conclusion that they were mistaken. Programming Tools

What kind of tools does a Real Programmer use? In theory, a Real Programmer could run his programs by keying them into the front panel of the computer. Back in the days when computers had front panels, this was actually done occasionally. Your typical Real Programmer knew the entire bootstrap loader by memory in hex, and toggled it in whenever it got destroyed by his program. (Back then, memory was memory -- it didn't go away when the power went off. Today, memory either forgets things when you don't want it to, or remembers things long after they're better forgotten.) Legend has it that Seymour Cray, inventor of the Cray I supercomputer and most of Control Data's computers, actually toggled the first operating system for the CDC7600 in on the front panel from memory when it was first powered on. Seymour, needless to say, is a Real Programmer.

One of my favorite Real Programmers was a systems programmer for Texas Instruments. One day, he got a long distance call from a user whose system had crashed in the middle of saving some important work. Jim was able to repair the damage over the phone, getting the user to toggle in disk I/O instructions at the front panel, repairing system tables in hex, reading register contents back over the phone. The moral of this story: while a Real Programmer usually includes a keypunch and lineprinter in his toolkit, he can get along with just a front panel and a telephone in emergencies.

In some companies, text editing no longer consists of ten engineers standing in line to use an 029 keypunch. In fact, the building I work in doesn't contain a single keypunch. The Real Programmer in this situation has to do his work with a "text editor" program. Most systems supply several text editors to select from, and the Real Programmer must be careful to pick one that reflects his personal style. Many people believe that the best text editors in the world were written at Xerox Palo Alto Research Center for use on their Alto and Dorado computers [3]. Unfortunately, no Real Programmer would ever use a computer whose operating system is called SmallTalk, and would certainly not talk to the computer with a mouse.

Some of the concepts in these Xerox editors have been incorporated into editors running on more reasonably named operating systems -- EMACS and VI being two. The problem with these editors is that Real Programmers consider "what you see is what you get" to be just as bad a concept in Text Editors as it is in Women. No, the Real Programmer wants a "you asked for it, you got it" text editor -- complicated, cryptic, powerful, unforgiving, dangerous. TECO, to be precise.

It has been observed that a TECO command sequence more closely resembles transmission line noise than readable text [4]. One of the more entertaining games to play with TECO is to type your name in as a command line and try to guess what it does. Just about any possible typing error while talking with TECO will probably destroy your program, or even worse -- introduce subtle and mysterious bugs in a once working subroutine.

For this reason, Real Programmers are reluctant to actually edit a program that is close to working. They find it much easier to just patch the binary object code directly, using a wonderful program called SUPERZAP (or its equivalent on non-IBM machines). This works so well that many working programs on IBM systems bear no relation to the original FORTRAN code. In many cases, the original source code is no longer available. When it comes time to fix a program like this, no manager would even think of sending anything less than a Real Programmer to do the job -- no Quiche Eating structured programmer would even know where to start. This is called "job security". Some programming tools NOT used by Real Programmers:
The Real Programmer At Work

Where does the typical Real Programmer work? What kind of programs are worthy of the efforts of so talented an individual? You can be sure that no real Programmer would be caught dead writing accounts-receivable programs in COBOL, or sorting mailing lists for People magazine. A Real Programmer wants tasks of earth-shaking importance (literally!).

- Real Programmers work for Los Alamos National Laboratory, writing atomic bomb simulations to run on Cray I supercomputers.
- Real Programmers work for the National Security Agency, decoding Russian transmissions.
- It was largely due to the efforts of thousands of Real Programmers working for NASA that our boys got to the moon and back before the Russkies.
- The computers in the Space Shuttle were programmed by Real Programmers.
- Real Programmers are at work for Boeing designing the operating systems for cruise missiles.

Some of the most awesome Real Programmers of all work at the Jet Propulsion Laboratory in California. Many of them know the entire operating system of the Pioneer and Voyager spacecraft by heart. With a combination of large ground-based FORTRAN programs and small spacecraft-based assembly language programs, they are able to do incredible feats of navigation and improvisation -- hitting ten-kilometer wide windows at Saturn after six years in space, repairing or bypassing damaged sensor platforms, radios, and batteries. Allegedly, one Real Programmer managed to tuck a pattern-matching program into a few hundred bytes of unused memory in a Voyager spacecraft that searched for, located, and photographed a new moon of Jupiter.

The current plan for the Galileo spacecraft is to use a gravity assist trajectory past Mars on the way to Jupiter. This trajectory crosses within 80 +/- 3 kilometers of the surface of Mars. Nobody is going to trust a PASCAL program (or PASCAL program) for navigation to these tolerances.

As you can tell, many of the world's Real Programmers work for the U.S. Government -- mainly the Defense Department. This is as it should be. Recently, however, a black cloud has formed on the Real Programmer horizon. It seems that some highly placed Quiche Eaters at the Defense Department decided that all Defense programs should be written in some grand unified language called "ADA" (Dijkstra, DoD). For a while, it seemed that ADA was destined to become a language that went against all the precepts of Real Programming -- a language with structure, a language with data types, strong typing, and semicolons. In short, a language designed to cripple the creativity of the typical Real Programmer. Fortunately, the language adopted by DoD has enough interesting features to make it approachable -- it's incredibly complex, includes methods for messing with the operating system and rearranging memory, and Edgar Dijkstra doesn't like it [8]. (Dijkstra, as I'm sure you know, was the author of "GoTos Considered Harmful" -- a landmark work in programming methodology, applauded by Pascal Programmers and Quiche Eaters alike.) Besides, the determined Real Programmer can write FORTRAN programs in any language.

The real programmer might compromise his principles and work on something slightly more trivial than the destruction of life as we know it, providing there's enough money in it. There are several Real Programmers building video games at Atari, for example. (But not playing them -- a Real Programmer knows how to beat the machine every time: no challenge in that.) Everyone working at LucasFilm is a Real Programmer. (It would be crazy to turn down the money of fifty million Star Trek fans.) The proportion of Real Programmers in Computer Graphics is somewhat lower than the norm, mostly because nobody has found a use for Computer Graphics yet. On the other hand, any Computer Graphics is done in FORTRAN, so there are a fair number people doing Graphics in order to avoid having to write COBOL programs.

The Real Programmer At Play

Generally, the Real Programmer plays the same way he works -- with computers. He is constantly amazed that his employer actually pays him to do what he would be doing for fun anyway (although he is careful not to express this opinion out loud). Occasionally, the Real Programmer does step out of the office for a breath of fresh air and a beer or two. Some tips on recognizing real programmers away from the computer room:

- At a party, the Real Programmers are the ones in the corner talking about operating system security and how to get around it.
- At a football game, the Real Programmer is the one comparing the plays against his simulations printed on 11 by 14 fanfold paper.
- At the beach, the Real Programmer is the one drawing flowcharts in the sand.
- A Real Programmer goes to discos to watch the light shows.
- At a funeral, the Real Programmer is the one saying "Poor George. And he almost had the sort routine working before the coronary."
- In a grocery store, the Real Programmer is the one who insists on running the cans past the laser checkout scanner himself, because he never could trust keypunch operators to get it right the first time.

The Real Programmer's Natural Habitat

What sort of environment does the Real Programmer function best in? This is an important question for the managers of Real Programmers. Considering the amount of money it costs to keep one on the staff, it's best to put him (or her) in an environment where he can get his work done.

The typical Real Programmer lives in front of a computer terminal. Surrounding this terminal are:

- Listings of all programs the Real Programmer has ever worked on, piled in roughly chronological order on every flat surface in the office.
- Some half-dozen or so partly filled cups of cold coffee. Occasionally, there will be cigarette butts floating in the coffee. In some cases, the cups will contain Orange Crush.
- Unless he is very good, there will be copies of the OSJCL manual and the Principles of Operation open to some particularly interesting pages.
- Taped to the wall is a line-printer Snoopy calendar for the year 1969.
- Strewen about the floor are several wrappers for peanut butter filled cheese bars -- the type that are made pre-stale at the bakery so they can't get any worse while waiting in the vending machine.
- Hiding in the top left-hand drawer of the desk is a stash of double-stuff Oreos for special occasions.
- Undersneath the Oreos is a flow-charting template, left there by the previous occupant of the office. (Real Programmers rewrite programs, not documentation. Leave that to the maintainence people.)

The Real Programmer is capable of working 30, 40, even 50 hours at a stretch, under intense pressure. In fact, he prefers it that way. Bad response time doesn't bother the Real Programmer -- it gives him a chance to catch a little sleep between compiles. If there is not enough schedule pressure on the Real Programmer, he tends to make things more challenging by working on some small but interesting part of the problem for the first nine weeks, then finishing the rest in the last week, in two or three 50-hour marathons. This not only impresses the hell out of his manager, who is despairing of ever getting the project done on time, but creates a convenient excuse for not doing the documentation. In general:
The Future

What of the future? It is a matter of some concern to Real Programmers that the latest generation of computer programmers are not being brought up with the same outlook on life as their elders. Many of them have never seen a computer with a front panel. Hardly anyone graduating from school these days can do hex arithmetic without a calculator. College graduates these days are soft -- protected from the realities of programming by source level debuggers, text editors that count parentheses, and "user friendly" operating systems. Worst of all, some of these alleged "computer scientists" manage to get degrees without ever learning FORTRAN! Are we destined to become an industry of Unix hackers and Pascal programmers?

From my experience, I can only report that the future is bright for Real Programmers everywhere. Neither OS/370 nor FORTRAN show any signs of dying out, despite all the efforts of Pascal programmers the world over. Even more subtle tricks, like adding structured coding constructs to FORTRAN have failed. Oh sure, some computer vendors have come out with FORTRAN 77 compilers, but every one of them has a way of converting itself back into a FORTRAN 66 compiler at the drop of an option card -- to compile DO loops like God meant them to be.

Even Unix might not be as bad on Real Programmers as it once was. The latest release of Unix has the potential of an operating system worthy of any Real Programmer -- two different and subtly incompatible user interfaces, an arcane and complicated teletype driver, virtual memory. If you ignore the fact that it's "structured", even C programming can be appreciated by the Real Programmer: after all, there's no type checking, variable names are seven (ten? eight?) characters long, and the added bonus of the Pointer data type is thrown in -- like having the best parts of FORTRAN and assembly language in one place. (Not to mention some of the more creative uses for #define.)

No, the future isn't all that bad. Why, in the past few years, the popular press has even commented on the bright new crop of computer nerds and hackers (3) and.

Acknowledgments

I would like to thank Jan E., Dave S., Rich G., Rich E. for their help in characterizing the Real Programmer, Heather B. for the illustration, Kathy E. for putting up with it, and atlavsdS:mark for the initial inspiration.

References


Shoot yourself in the foot THE PROGRAMMER'S QUICK GUIDE TO THE LANGUAGES

The proliferation of modern programming languages (all of which seem to have stolen countless features from one another) sometimes makes it difficult to remember what language you're currently using. This handy reference is offered as a public service to help programmers who find themselves in such a dilemma.

TASK: Shoot yourself in the foot.

C: You shoot yourself in the foot.

C++: You accidentally create a dozen instances of yourself and shoot them all in the foot. Providing emergency medical assistance is impossible since you can't tell which are bitwise copies and which are just pointing at others and saying, "That's me, over there."

FORTRAN: You shoot yourself in each toe, iteratively, until you run out of toes, then you read in the next foot and repeat. If you run out of bullets, you continue with the attempts to shoot yourself anyways because you have no exception-handling capability.

Pascal: The compiler won't let you shoot yourself in the foot.

Ada: After correctly packing your foot, you attempt to concurrently load the gun, pull the trigger, scream, and shoot yourself in the foot. When you try, however, you discover you can't because your foot is of the wrong type.

COBOL: Using a COLT 45 HANDGUN, AIM gun at LEG.FOOT, THEN place ARM.HAND.FINGER on HANDGUN.TRIGGER and SQUEEZE. THEN return HANDGUN to HOLSTER. CHECK whether shoelace needs to be re-tied.

LISP: You shoot yourself in the appendage which holds the gun with which you shoot yourself in the appendage which holds the gun with which you shoot yourself in the appendage which holds the gun which with which you shoot yourself in the appendage which holds the gun which with which you shoot yourself in the appendage which holds ...

FORTH: Foot in yourself shoot.

Prolog: You tell your program that you want to be shot in the foot. The program figures out how to do it, but the syntax doesn't permit it to explain it to you.
BASIC: Shoot yourself in the foot with a water pistol. On large systems, continue until entire lower body is waterlogged.

Visual Basic: You'll really only appear to have shot yourself in the foot, but you'll have had so much fun doing it that you won't care.

HyperTalk: Put the first bullet of gun into foot left of leg of you. Answer the result.

Motif: You spend days writing a UIL description of your foot, the bullet, its trajectory, and the intricate scrollwork on the ivory handles of the gun. When you finally get around to pulling the trigger, the gun jams.

APL: You shoot yourself in the foot, then spend all day figuring out how to do it in fewer characters.

SNOBOL: If you succeed, shoot yourself in the left foot. If you fail, shoot yourself in the right foot.

Unix:

```bash
% ls  
foot.c foot.h foot.o toe.c toe.o  
% rm *.o  
rm:.o no such file or directory  
% ls  
```

Concurrent Euclid: You shoot yourself in somebody else's foot.

370 JCL: You send your foot down to MIS and include a 400-page document explaining exactly how you want it to be shot. Three years later, your foot comes back deep-fried.

Paradox: Not only can you shoot yourself in the foot, your users can, too.

Access: You try to point the gun at your foot, but it shoots holes in all your Borland distribution diskettes instead.

Revelation: You're sure you're going to be able to shoot yourself in the foot, just as soon as you figure out what all these nifty little bullet-thingies are for.

dBase: You buy a gun. Bullets are only available from another company and are promised to work so you buy them. Then you find out that the next version of the gun is the one that is scheduled to actually shoot bullets.

PL/I: After consuming all system resources including bullets, the data processing department doubles its size, acquires two new mainframes and drops the original on your foot.

Assembler: You try to shoot yourself in the foot, only to discover you must first invent the gun, the bullet, the trigger, and your foot.

You crash the OS and overwrite the root disk. The system administrator arrives and shoots you in the foot. After a moment of contemplation, the administrator shoots himself in the foot and then hops around the room rabidly shooting at everyone in sight.

Modula2: After realizing that you can't actually accomplish anything in this language, you shoot yourself in the head.

Computer Humor

Article: 9806 of alt.folklore.computers
Path: duke!mcnc!stanford.edu!morrow.stanford.edu!DecWRL!zaphod.mps.ohio-state.edu!van-bc!ubc-cs!unixg.ubc.ca!kakwa.ucs.ucalgary.ca!acs.ucalgary.ca!jsbell
From: jsbell@acs.ucalgary.ca (Joshua Bell)
Newsgroups: alt.folklore.computers
Subject: Murphy's Laws...
Message-Id: <1991Sep26.212135.16840@acs.ucalgary.ca>
Date: 26 Sep 91 21:21:35 GMT
Organization: The University of Calgary, Alberta, Canada
Lines: 67
Status: RO

Okay, pulled of out our old Multics Jokes Forum...:

This list seems to be even more comprehensive than /usr/games/fortune's contributions to the discussion...

__________Anything Below This Line is Not My Fault___________________

[0367] (50 lines) Donovan.Service 03/13/85 1013.5 mdt Wed Jokes
Subject: Murphy
I looked and did not find these in this meeting, so here goes:

A few laws of computer programming:

1. Any given program, when running, is obsolete.
2. Any given program costs more and takes longer.
3. If any program is useful, it will have to be changed.
4. If a program is useless, it will have to be documented.
5. Any given program will expand to fill all available memory.
6. The value of a program is proportional to the weight of its output.
7. Program complexity always grows until it exceeds the capability of the programmer who must maintain it.
8. If a test installation functions perfectly, all subsequent systems will malfunction.
9. Job control cards that positively cannot be arranged in improper
10. If the input editor has been designed to reject all bad input, an
ingenious idiot will discover a method to get bad data past it.
11. Profanity is the one language all programmers know best.
12. Adding manpower to a late software project makes it later.
13. A carelessly planned project takes three times longer to complete
than expected; a carefully planned project takes only twice as long.
14. (Lubarsky's Law of Cybernetic Entomology) There is always one more
bug.
15. It is impossible to make any program foolproof because fools are
so ingenious.
16. When things are going well, something will go wrong.
17. When things just can't get any worse, they will.
18. Anytime things appear to be going well, you have overlooked something.
19. Test functions and their tests should be reproducible -- they should
all fail in the same way.
20. If it looks easy, it's tough.
21. If it looks tough, it's damn near impossible.
22. You always find any bug in the last place you look.
23. Anything can be made to work if you fiddle with it long enough.
24. A terminal usually works better if you plug it in. 25. If all else
fails, read the documentation.
26. If you do not understand a particular word in a piece of technical
writing, ignore it. The piece will make perfect sense without it.
27. No matter how much you do, you'll never do enough.
28. What you don't do is always more important than what you do do.
29. Procrastination avoids boredom; one never has the feeling that there
is nothing important to do.
30. Always leave room to add an explanation if it doesn't work out.
31. No amount of genius can overcome a preoccupation with detail.
32. Nothing is impossible for a man who doesn't have to do it himself.
33. If builders built buildings the way programmers write programs,
then the first woodpecker than came along would destroy civilization.
34. Programmers will act rational when all other possibilities have
been exhausted.

---[0367]---

Ten Commandments for Stress Free Programming

1. Thou shalt not worry about bugs.
   Bugs in your software are actually special features.
2. Thou shalt not fix abort conditions.
   Your user has a better chance of winning state lottery than getting
   the same abort again.
3. Thou shalt not handle errors.
   Error handing was meant for error prone people, neither you or your
   users are error prone.
4. Thou shalt not restrict users.
   Don't do any editing, let the user input anything, anywhere, anytime. That is being very user friendly.
5. Thou shalt not optimize.
   Your users are very thankful to get the information, they don't worry about speed and efficiency.
6. Thou shalt not provide help.
   If your users can not figure out themselves how to use your software than they are too dumb to deserve the benefits of your software anyway.
7. Thou shalt not document.
   Documentation only comes in handy for making future modifications. You made the software perfect the first time, it will never need modifications.
8. Thou shalt not hurry.
   Only the cute and the mighty should get the program by deadline.
9. Thou shalt not revise.
   Your interpretation of specs was right, you know the users' requirements better than them.
10. Thou shalt not share.
    If other programmers needed some of your code, they should have written it themselves.

Quotes

*Programs must be written for people to read, and only incidentally for machines to execute.*

- Abelson & Sussman, *SICP*, preface to the first edition

*Premature optimization is the root of all evil (or at least most of it) in programming.*

- Donald Knuth

*Greenspun's Tenth Rule of Programming: any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp.*

- Phil Greenspun

*Some may say Ruby is a bad rip-off of Lisp or Smalltalk, and I admit that. But it is nicer to ordinary people.*

- Matz, LL2

*I object to doing things that computers can do.*

- Olin Shivers
"Inside every fat person is a thin person screaming to get out."
- Richard Simmons

"Dealing with failure is easy: Work hard to improve. Success is also easy to handle: You've solved the wrong problem. Work hard to improve."
- Alan Perlis

Most papers in computer science describe how their author learned what someone else already knew.
- Peter Landin

"The only way to learn a new programming language is by writing programs in it."
- Kernighan and Ritchie

"If I had a nickel for every time I've written "for (i = 0; i < N; i++)" in C I'd be a millionaire."
- Mike Vanier

"Language designers are not intellectuals. They're not as interested in thinking as you might hope. They just want to get a language done and start using it."
- Dave Moon

"The key to performance is elegance, not battalions of special cases."
- Jon Bentley and Doug McIlroy

"Don't worry about what anybody else is going to do. The best way to predict the future is to invent it."
- Alan Kay

**Todd's Humor Archive The Charge of the Code Brigade**

Half a Mb, half a Mb,
Half a Mb farther.
Churning out code, fixing bugs:
The six coders.
"Forward, the Code Brigade,
Aim for the ship date," he said.
Churning out code, fixing bugs:
The six coders.

"Forward, the Code Brigade!"
Was their a one dismayed?
Well though the coders knew
Some bits were rotten.
Their's not to feel surprise,
Their's not to close their eyes,
Their's but to see sunrise.
Churning out code, fixing bugs:
The six coders.

Bugs to right of them,
Bugs to left of them,
Bugs in front of them,
    Crashes and freezes!
Stack traces so bizarre,
Pointers to near and far,
Zero dereferenced there,
Heap scrambled, frantic now.
Churning out code, fixing bugs:
The six coders.

**Computer Programming Humor Quotes:**

- "Programming graphics in X is like finding sqrt(pi) using Roman numerals."
  - Henry Spencer

- "Never put off until run time what you can do at compile time."

- BASIC programmers never die, they GOSUB and don't RETURN.
Real programmers are surprised when the odometers in their cars don't turn from 99,999 to A0000.

FORTRAN is not a language. It's a way of turning a multi-million dollar mainframe into a $50 programmable scientific calculator.

C is almost a real language. Even the name sounds like it's gone through an optimizing compiler. Get rid of all of those stupid brackets and we'll talk.

Any sufficiently advanced bug is indistinguishable from a feature.

Programming is 10% science, 25% ingenuity and 65% getting the ingenuity to work with the science.

Science is to computer science as hydrodynamics is to plumbing.

We don't really understand it, so we'll give it to the programmers.

COBOL programmers understand why women hate periods.

Computer interfaces and user interfaces are as different as night and 1.

The human mind ordinarily operates at only ten 10% of its capacity, the rest is overhead for the operating system.

A computer scientist is someone who fixes things that aren't broken.

The computer is mightier than the pen, the sword, and usually the programmer.

Programming is an art form that fights back.

After a number of decimal places, who cares?

"Virtual" means never knowing where your next byte is coming from.

If at first you don't succeed, you must be a programmer.

"It's 5:50 a.m., Do you know where your stack pointer is?"

If God had intended humans to program, we would be born with serial I/O ports.

There are two ways to write error-free programs; only the third one works.

You never finish a program, you just stop working on it.

Deliver yesterday, code today, think tomorrow.

PL/1, "the fatal disease", belongs more to the problem set than to the solution set.

Yea, though I walk through the valley of the shadow of APL, I shall fear no evil, for I can string six primitive monadic and dyadic operators together.

You can't make a program without broken egos.

**Fortran purity test [rec.humor.funny]**

Question 1: The forthcoming Fortran standard [now F90] is important because:

a) It will make it easier for people to write useful Fortran programs  
b) It will give compiler vendors more work to do and products to sell  
c) It will make it easier for people to teach others how to write Fortran  
d) It will serve as the ideal application language for Windows 3.0  

... ... ...

**Structured Programmer's Soliloquy**

SP or not SP -- that is the question:  
Whether 'tis nobler in the mind to suffer  
The rules and exceptions of outrageous FORTRAN  
Or to take arms against a sea of transfers  
And by structuring end them. To code -- to test  
No more; and by a test to say we end  
The heartache, and the thousand natural mistakes  
That FORTRAN is heir to. 'Tis a consummation  
Devoutly to be wish'd. To code -- to test.  
To test -- perchance to bomb: aye, there's the rub!  
For in that test of code what bugs may come  
When we have shuffled of this FORTRAN code,  
Must give us pause. There's the respect  
that makes calamity of so long lists. [??]  
For who would bear the whips and scorns of time-sharing  
Th' operating systems wrong, the computer's crash,  
The pangs of despis'd code, the turnaround's delay,
The insolence of compilers, and the spurns
That patient coding of FORTRAN takes
When he himself might his quietus make
with PL/I?  Who would this FORTRAN Bear,
To grunt and sweat under a weary language,
But that the dread of something after FORTRAN
The undiscover'd country, from whose bourne
No programmer returns -- puzzles the will,
And makes us rather bear those ills we have
Than fly to others that we know not of?
Thus conscience does make cowards of us all,
And thus the native hue of resolution
Is sicklied o'er with the pale cast of thought,
And enterprises of great pith and moment
With this regard their currents turn away
And lose the name of action.

- Henry Kleine and Philip H. Roberts
April DATAMATION

Cobol

"The use of COBOL cripples the mind; its teaching should therefore be regarded as a criminal offense."  — E.W. Dijkstra (1930—2002).

"COBOL programs are an exercise in Artificial Inelegance."

"If a group of N persons implements a COBOL compiler, there will be N-1 passes. Someone in the group has to be the manager."

"Shots through the Terminator's vision show Motorola 6502 microprocessor assembler code; the 6502 chip is the main CPU for the Apple II computer. Other code visible is written in COBOL."  — From the IMDB trivia.

"COBOL programmers understand why women hate periods."

Ada, Fortran, assembler, perl, brainf**k and whatnot...

"Some languages are designed to solve a problem. Others are designed to prove a point."  — Bell Labs saying.

"Do you program in Assembly?" she asked. "NOP", he said.

"You can tell how far we have to go, when FORTRAN is the language of supercomputers."  — Steven Feiner

"FORTRAN was the language of choice for the same reason that three-legged races are popular."  — Ken Thompson, "Reflections on Trusting Trust"

"In the good old days physicists repeated each other's experiments, just to be sure. Today they stick to FORTRAN, so that they can share each other's programs, bugs included."  — E.W. Dijkstra (1930—2002).

"Consistently separating words by spaces became a general custom about the tenth century A.D., and lasted until about 1957, when FORTRAN abandoned the practice."  — Sun FORTRAN Reference Manual.

"FORTRAN is not a flower but a weed — it is hardy, occasionally blooms, and grows in every computer."  — A.J. Perlis.

"Any language that will allow you to define the number 4 as a word that places the number 3 on the stack can be a frightening weapon."  — About Forth.

"Pascal in./ A programming language named after a man who would turn over in his grave if he knew about it."

"BASIC programmers never die, they GOSUB and don't RETURN."

"Real programmers don't write in BASIC. Actually, no programmers write in BASIC after reaching puberty."

"When Roman engineers built a bridge, they had to stand under it while the first legion marched across. If programmers today worked under similar ground rules, they might well find themselves getting much more interested in Ada !"  — Robert Dewar, President Ada Core Technologies.

"Q: If anyone knows of a book that is the functional equivalent of "The Idiot's Guide to C" for the Ada language, please send me the title and author. A: Idiots don't use Ada. Idiots only use C or derivations."  — David Weller.

"Epigram: Ada is the 400-pound gorilla of programming languages."

"The problem about all graphical programming languages is that when your project becomes complex, not only will you have spaghetti code, but it will actually look like spaghetti too."

"Two languages implementing the same idea must, on pain of death, use different terms."  — Feldman's Law of Programming Terminology.

"If Python is executable pseudocode, then perl is executable line noise."

Programming Languages are Like Cars
Etc

Society


Quotes


Bulletin:


History:


Classic books:


Most popular humor pages:
The Last but not Least Technology is dominated by two types of people: those who understand what they do not manage and those who manage what they do not understand ~Archibald Putt. Ph.D