TUG Lines 51

Communications of the Programmer Resource Organization **October/November 1992**

Perspective

My buddy Brad can always get a laugh. He'll look you straight in the eye and say with great sincerity, "You can trust me. I'm in Real Estate." No one ever takes him seriously. I wonder why...

I remember my college days, when we used to say "Never trust anyone over 30." Those were words to live by - until we turned 30. And then 40. Now I've begun to wonder just who I *can* trust, and I've come to one conclusion: *Watch out for the "experts"* -the people to whom we give trust without them having to earn it.

Consider the "Truth in Advertising" laws here in the U.S. If you feel you can sit back and relax because you're being protected, think again. Here are some quick examples...

Do you know that products like tires, toothpaste and cold remedies are called "parity" products, because it's acknowledged by the Federal Trade Commission that all products within a given category are basically the same? Therefore, all toothpastes are equal. Now comes the *logic of experts:* since the products are all equal, if any one is good, they're *all* good. And if one is the best (since they're all equal), they must *all be the best.* The next time you see an ad for a cough syrup and are told that "tests confirm one cough syrup is best", be aware that statement means absolutely nothing, and that the manufacturer has no requirement to support the claim.

Do you prefer to buy "fresh" chicken at the grocery, or will you settle for "frozen"? According to the Department of Agriculture, chicken meat is still considered "fresh" and not "frozen" if it has been reduced in temperature to no less than 28 degrees Fahrenheit. Apparently physics is not on the required list of subjects for USDA policy makers.

In his book Doublespeak (Harper & Row, 1989), William Lutz describes in detail how we are constantly being deceived by others who use language as their weapon. He covers everything from politics to business; advertising to the food industry. My favorite is his multiple-choice test. In one column you'll find a list of 10 popular products; in the other column are 10 lists of ingredients. The challenge is to match each product with its ingredients. The products include such diverse items as white bread, Cool Whip dessert topping, Gaines Burgers dog food, Pepsodent toothpaste and Preparation H. If you think that sounds ridiculous, you'll be amazed how difficult it is to match them up. Get the book and you'll see what I mean. (I recommend it highly.)

So much for the people trying to sell us products and services. But what about the politicians, who are trying to sell us the promise of a better future?

Along with millions of others, I followed the presidential election campaigns here in the U.S. As part of that I watched portions of the conventions, and all of the debates. What I experienced throughout it all was a sense of disgust, as I witnessed one of the most sickening displays of lies, half-truths and smear tactics I've ever seen. For the debates, one of the candidates brought in more than 200 assistants - *experts* whose only purpose was to make them look good on television. Another candidate brought more than 300 experts! As preparations progressed, these experts concerned themselves with critical tasks like measuring the dressing rooms, trying to get the largest room for their candidate to wait in; negotiating the maximum number of limousines that could be in their entourage; and pumping their candidate with sound bites, mini-speeches, and lessons in body language. When the cameras were turned on, each of the candidates looked good, smelled good, and sounded even better. They fought valiantly to reframe each question asked as

Continued on page 2

In This Issue

About TUG Lines . . .

TUG Lines is published bi-monthly by TUG/Pro, as a benefit of TUG/Pro membership. TUG Lines is not available on newsstands or by subscription independent of membership.

Address/Phone

TUG/Pro
PO Box 1510
Poulsbo, WA 98370
Voice:.... 206/779-9508
FAX: 206/779-8311

Editorial Staff

Technical Staff

Member Services

Contributors

Membership. Membership in TUG/Pro is $75.00 per year in the United States; $85.00 per year in Canada and Mexico; and $99.00 per year elsewhere. All monies must be in US dollars. Each year's membership includes a one-year subscription (six issues) to TUG Lines, six newsletter disks, and discounts on TUG/Pro products, including a special discounted rate for GeTUGether, our annual programmer's conference. For more information, request a copy of our current membership prospectus.

Renewals. The renewal date shown on your TUG Lines label tells you the month and the year by which you must renew if you don't want your membership to lapse. For example, a renewal date of "07/94" means you must renew your membership by June 30, 1994, or your membership will come to an end. You will automatically receive a renewal reminder shortly before your membership lapses. (By the way - the label also tells you the issue number of the last TUG Lines you're scheduled to receive.)

Advertising. We accept a limited amount of advertising, and our rates are very reasonable. For more information, or for our latest advertising rate sheet, contact Carol Taylor at 206/779-9508.

Articles and Submissions. You are encouraged to share your knowledge and experience by contributing articles and reviews for possible publication in TUG Lines. Write for a free copy of our author's guide..

Mailing List. Our membership list is intended for TUG/Pro business only, and it will not be rented, sold or made available to another party or used for any other purpose.

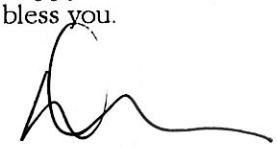
Continued from Page 1

rapidly and smoothly as possible into a prepared mini-speech intended to win the hearts of the American public. But no matter who gets elected, you can bet when it comes time to try to implement them, the promises will have changed.

Are you getting an accurate picture of world and local events from television news or your newspaper? Probably not. Let's take a quick example, and as we do, let's agree on one thing: that to quantify anything absolutely, it must be measured against a standard. In this case, the standard is the collective knowledge of computer viruses you and I possess - for the moment, we're the experts. Okay, remember the reporting that took place just prior to the Michaelangelo virus fiasco? In nearly every case, the story was prefaced by a description of what a computer virus is, as described by the local or national news reporter. Of those you saw or read, how many gave an accurate description of a virus? How many times during those reports did you hear something said that was either inaccurate or downright untrue? Now consider this: *what you're hearing and reading in everyday reporting has that same level of accuracy.* News reporters aren't normally experts on subjects they're assigned; they just pick experts who supply them with their information, be it technical, financial, political or whatever. (If you want a fascinating description of how all of this misinformation happens - and particularly how it happened during the Michaelangelo fiasco - get a copy of "Computer Viruses - The Real Story" from the TUG/Pro library.)

At this point you're probably wondering what any of this has to do with the programming profession. The answer is "nothing" and "everything". In the next issue I'll tell you why, and also why I believe knowing how to pick the experts you trust may be the most important issue you'll face in the remainder of this century.

In the meantime, I wish you a Merry Christmas, a Happy Hanukkah, and a Prosperous New Year. May God bless you.



Don Taylor

P.S. I've got a little gift for you. You'll find its description hidden in the *Top of the Heap* section of this issue...

Global

Product News

Knowledge Dynamics Corporation has announced a major update to their **Install** program. The most significant new feature in this release is Install's new compression algorithm, Reduce, that compresses data approximately 40% better than before. This brings the total average data compression up to approximately 60% for most products. Additionally, version 3.2 has an enhanced script language that can simulate almost any target computer environment (including ones that don't even exist yet like MS-DOS 8.0). It also offers fully general assignment statements; 12 new string-handling functions; on-the-fly modification of environment variables; and the ability to cold or warm boot the end-user's computer.

Another optional feature for Install 3.2 is a new HyperText Help System. This Pop Up help system is a TSR program which allows instant access to most of the Install manual. It requires only 2K of RAM with EMS/XMS and is Norton Guides compatible. HyperText Help lists for $49, but, for a limited time, it is free to people who update to Install 3.2. The list price of Install 3.2 is $249.95. Version 3.2 updates are priced between free and $99, depending on the purchase date. For more information contact Knowledge Dynamics Corporation, PO Box 1558, Canyon Lake, TX 78130, 800/331-2783, 512/964-3994; 512/964-3958 (fax).

MakeFast is a new software product that automatically configures MS-DOS to make your application software as fast as possible. MakeFast searches your hard disk to determine what software you use. It then customizes your system so that your software will make the best use of your hardware, resulting in a faster, more responsive system.

Without specific customization, MS-DOS does a remarkably poor job of utilizing your system's memory. You need to modify your configuration file to obtain the best performance from your system. MakeFast automatically does this for you. It understands all about device drives, disk caches and memory management, and it will maximize the memory available to your programs, providing the type (conventional, expanded or extended) of memory they need for optimal performance. MakeFast combines an expert system with a detailed knowledge-base of application software to do its work. It will detect many of the popular programs, and it can even determine which programs you use most. For instance, it will put the emphasis on optimizing your system for Quattro Pro if it finds QPro on your system, along with a lot of spreadsheet files. For more information contact TDRMSoft, 275 West 96th Street, Suite 10R, New York, NY 10025, 212/865-2719.

Library Notes

The videos from GeTUGether '92 are now available, and in addition, we have several interesting disks - one in particular that should be a lot of fun...

Videos

"Software Craftsmanship" Borland International's ***Zack Urlocker*** talks frankly about software craftsmanship. Who are your users? What is most important to them? How can you meet their needs? What is the difference between features and benefits - and what is the true cost of adding features? Zack teaches the *guiding principles* of software design - those principles that will help you craft the best possible product. These principles include orthogonal design, cohesion, user accommodation, automation, information, platform exploitation and optimization. The bonus disk that accompanies this video contains Zack's routines for using charts in Turbo Pascal for Windows - along with a fun little surprise... *Stock number VID-MIS-027*.

"Validated Data Entry with Turbo Pascal for Windows" TurboPower Software's ***Brian Foley*** discusses the pitfalls of data validation under Windows, using Borland's Turbo Pascal for Windows and their Object Windows Library. Brian covers three different levels of validation, and presents TPW routines that solve many of the problems when working with Windows. He also discusses and demonstrates some of the subtleties of validation, including complex focus changes and handling multiple invalid fields. He then describes how to handle error reporting. The bonus disk that accompanies this video contains the examples used in this session. *Stock number VID-PAS-028*.

"Assembly Language Optimization Tricks" In this session, ***Bob Falk*** of Falk Data Systems shows how to squeeze the last bit of performance from your code.. Using "hot rod" techniques, Bob reveals how to go beyond classical optimization methods by considering the details of the PC hardware the program is running on. You'll learn how merely minimizing machine cycles in code can actually decrease performance. You'll find out about the various cycle eaters that maybe you've never before considered - DRAM refresh and video memory refresh, for instance (your display adapter can be hogging 90% of the time available for your CPU to access video memory!). You'll find out what you can do about it. And lots of other tricks for increasing your program's performance. *Stock number VID-MIS-029*.

"Debugging Strategies for Frameworks-Based Applications" Borland International's ***Danny Thorpe*** will reveal some deep, dark secrets of debugging applications written with Borland's Object Windows Library and Turbo Vision frameworks. Danny demonstrates how to

classify a problem, and then to bracket it and converge on it. He shows how to determine which messages are involved, and how to use these messages to set program breakpoints. You'll learn what causes bugs to move around in a Windows app, and how you can use "WRITELN debugging" - without the debugger - to track the status of your program's operation. Through the use of some step-by-step examples, Danny demonstrates the techniques he teaches in this session. *Stock number VID-MIS-030*.

"**Writing Software for Multilingual Users**" In this session ***Jim Welsh*** discusses the concepts required to write multilingual software - software that can be configured to provide all its information - menu items, field prompts and help messages - in more than one language. Jim discusses when to use software to translate from one language to another, and when to use a professional translator. He also covers the concerns of programmers, including how best to incorporate several languages into one program, how to deal with the variations in word length between language translations, and how to keep from sacrificing speed. *Stock number VID-MIS-031*.

"**Fun and Fancy String Routines in Turbo Pascal**" Software Science's ***Dr. George Rothbart*** demonstrates five Pascal string routines: *Plurals* - using the standard English grammar rules, this routine will pluralize (nearly) any noun properly; *Wholes and fractions* - converts a real number (such as 93.127) to a whole number and a fraction (93 1/8); *Proper case* - converts a string to words with uppercase initial characters; *Search and replace* - handles substrings of varying length; and *Screen implosion* - this fun routine can make even the dullest text look like a video game! The bonus disk that accompanies this video contains the examples used in this session. *Stock number VID-PAS-032*.

"**Getting the Technical Press to Publicize Your Product**" PC Techniques Magazine's ***Jeff Duntemann*** lays out a plan that will give anyone with a high-tech product a leg up on getting mentioned in new product announcements or even landing a review. Mind you, this is not the typical advice you may have read before. Jeff has for years been a recognized player in the high-tech PC industry, as both an editor and an author. In this session he reveals some jealously guarded secrets - guerilla tactics - for conducting your publicity campaign. And, of course, there's no stopping Jeff's razor-sharp sense of humor... *Stock number VID-MIS-033*.

"**Objects and DLLs in Turbo Pascal for Windows**" TurboPower Software's ***Kim Kokkonen*** introduces the concept of pseudo objects - entities that offer the best of both worlds, giving the benefits of OOP, while meeting the memory model and structure requirements of DLLs for Windows. Kim examines the TPW memory model and the TPW object model, including an analysis of Virtual Method Table and Dynamic Method Table structures to show why a DLL can't export objects, and how a pseudo object's record structure can emulate a VMT. As an example Kim shows how to apply the pseudo object technique to create a large array facility that will manage arrays larger than 64KB. *Stock number VID-PAS-034*.

"**Making Windows 3.1 Work with Networks**" In this session ***Bob Haslanger*** of NET Systems reveals how to do it the easy way. He shares recommended resources for information, along with invaluable "unobtainium" he has gained as a network consultant and a beta tester for Microsoft. You'll find out the real "bare bones" configuration for workstations; the real minimum configuration; why you should have a hard disk in your workstation; how much RAM is enough; why standards are essential; how you can save disk space when installing a network; and suggested CONFIG and AUTOEXEC files to avoid system lockups. Bob also covers the best way to set up a server and its workstations; the fastest way to set up new users; how to troubleshoot the installation; how to maintain the network by adding new users and applications; and recommended tools for system troubleshooting. *Stock number VID-MIS-035*.

"**TPW Objects for Printing Under Windows**" Borland International's ***Danny Thorpe*** demonstrates an easier way to print from a Windows application. He introduces two new objects that form a subset of a group of objects that will be included in a future release of TPW. Danny gives an overview of operations when printing the normal way, contrasting it with the use of printing objects from the OWLPrint library. The bonus disk that accompanies this video contains the OWLPrint library objects presented in this session. *Stock number VID-PAS-036*.

"**Ray Tracing with Turbo Pascal**" Western Kentucky University's ***Bob Crawford*** discusses how to generate realistic graphic objects on the PC, using ray tracing techniques. You'll learn about how light sources are accommodated, how colors are determined, how reflected and normal rays are traced, and much more. Bob includes a step-by-step example of computing local color from ambient, diffuse and specular components, and he demonstrates the effects of "bump maps". Bob also demonstrates the TUGRay program, a Pascal application which performs all the fundamental ray tracing functions. The bonus disk that accompanies this video contains the TUGRay program, a bibliography, and numerous examples presented in this session. *Stock number VID-PAS-037*.

"**Writing User Documentation that Works**" Barn Owl Software's ***Rob Rosenberger*** covers what he calls "guerilla" writing tips - four powerful concepts that will help your user documentation look its best. You'll find out how to reduce the fluff you find in most magazine and newspaper articles - and unfortunately, in most user manuals. With the techniques presented here, you can make your writing more concise, which can reduce both the size of your manuals and your cost to print them. *Stock number VID-MIS-038*.

"**Building on the Simple GUI**" TEGL Systems' ***Richard Tom*** shows you how to create a graphical interface without resorting to Windows. Building on the 16K Graphical User Interface he introduced at GeTUGether '91, Richard adds "Window Dressings" that make the simple GUI look and act even more like Windows. Starting with a quick review of the concepts presented last year, Richard shows how the GUI ties together frames, mouse clicks and "hot spots" on the screen. He then extends the concepts with the addition of string

editing in dialog boxes. He discusses the new facilities - keypress events, pending events and timer events - required by the system. He then demonstrates a tiny application that has an uncanny resemblance to a Windows app - except it only occupies 30K! The bonus disk that accompanies this video contains the 30K GUI examples presented in this session. *Stock number VID-MIS-039*.

"Giving Text Programs a Graphical Look" Rimrock Software's ***Michael Burton*** shows how you can add "spiff" to your applications by - Replacing display fonts with custom fonts; Creating graphic icons for use in text mode; and Changing the mouse cursor. Mike demonstrates the new look offered by these techniques, and then shows how to incorporate them into your own Pascal and C programs. The bonus disk that accompanies this video contains the Pascal and C examples used in this session. *Stock number VID-MIS-040*.

"Lateral Problem Solving Techniques" Have you ever been faced with an "unsolvable" problem? One that had you throwing your hands in the air, shouting "I've tried everything!"? (Which you hadn't, of course.) We've all had similar experiences. But in this session with Midnight Engineering's ***Bill Gates***, you'll be exposed to some different ways of dealing with problems - er, *situations* - that can provide significant advantages. Bill is one of the most creative thinkers around, and according to him, our view of the world has an important impact on our approaches to problem solving. By looking at problems with a unique perspective (as radical as giving up!), we can increase the probability of solving them. As Bill says, "When things are working - that's when to be bold." *Stock number VID-MIS-041*.

"Why Profilers?" 3QC, Inc.'s ***Breck Carter*** of 3QC, Incorporated gives you a step-by-step example of how to optimize a program by analyzing it with a profiler and then changing the most critical portions of the code. He takes a program that displays information about a vendor's products, and by changing only a few lines of code he increases its speed by nearly a third! (Quite a feat, since critical portions of the program were originally coded in assembly language to enhance performance!) If you've ever wondered how - or even *why* - you would use a profiler, this video is for you. *Stock number VID-MIS-042*.

Disks

The Custom Controls Tutorial in an "entry level" introduction to BWCC, covering everything from basic DLLs to multiple DLLs. It's almost a megabyte of material, written by Bob Bourbonnais at Borland. (For more information, see the CScape section of this issue.) *Stock number OW-CPP-WIN-003*.

The Turbo Powered Editor disk has been updated with the latest version of TPE, plus we've added several other goodies. **HelpExe** is a collection of programs for creating, editing, modifying and testing the text files of an

Object Professional (or Turbo Professional) help system. **Heap6** is a collection of Turbo Pascal units and utilities that offers methods for managing and extending the heap. And of course, there's **TPE**, one of the most powerful programmer's editors around. The latest version is 3.4. It has most of the features of editors costing more than $100, and it also has the "four Fs" - it's fast, flexible, fun and freeware. *Stock number UT-MIS-DOS-005*.

Super Disk Utilities is a collection of four powerful shareware programs for working with diskettes. **ANAD** is the compleat diskette utility. Nothing like it anywhere else - scan, edit, repair and copy just about any kind of diskette. Current release supports secondary diskette controllers, FAT editing, and sector dump to DOS file. **CNFmt** is a "pop-up" diskette formatter that allows you to format diskettes while doing more productive things. Supports all current DOS formats. **CopyQ** is a virtual diskette duplicating machine - it formats, copies and verifies all in one pass. Up to four drives supported at once; features color icon-type interface, drive-status sensing (no keyboard entries). Record/playback image files from hard disk, serialize copies, even copy non-DOS formats in "Blind Copy" mode. Compare copies against master diskette or image file. Convert between diskette formats. All standard DOS formats supported. **FormQ** is a mass diskette formatter in the same "no hands" tradition of CopyQM. Formats and verifies faster than anything else around. Supports all standard DOS formats, up to four drives at a time. *Stock number UT-MIS-DOS-020*.

SeekEasy (version 7.6) is a file-searching, information-finding program that's uniquely easy to use. It uses "fuzzy-matching" logic to let you find information stored on your disk without having to worry about specific filenames, the exact wording of what you're looking for, word-order, capitalization, exact spelling, etc. Note: this program only reads files - it will not alter files. SeekEasy can search any file for the desired text - word-processor files, database files, even EXE and COM files. It will search a single file, a specified group of files, all the files in a directory, all the files in a directory and its "child" directories, or all files on a given drive. It will search floppies, hard disks, RAM-disks, etc. *Stock number UT-MIS-DOS-021*.

The Pseudographics Font Masticator Disk contains two unusual shareware programs from Rimrock Software. **DFE** (version 1.04) is a graphics-based editor for DOS code page display files (.CPI) or raw EGA/VGA font files. DFE allows the user to create custom fonts that can be loaded into the display adapter and used in place of the normal display font. DFE includes a set of utilities that provide useful capabilities such as loading fonts to the display, mapping fonts and converting fonts to ASCII. The ASCII files may be incorporated directly into programmer's C source files to give any program a custom look.

Have you ever wanted to dress up the text-based programs you create with graphic images, without actually going through the hassle of rewriting your program for a graphics display mode? With icons created by the **Rimrock Icon Editor** (version 1.04), you can create 'pseudo-graphic' images that can be displayed on an EGA or VGA display in text mode. RIE allows you to graphically create a 16 bit by 32 bit icon, which is stored in a C source file. This information can then be used as 8 characters of 8 lines, which can be loaded as user-defined characters into the display adapter. *Stock number UT-MIS-DOS-022*.

SnagIt (version 2.0) is *the* screen capture and print shareware for Microsoft Windows 3.x. With SnagIt, Windows users can capture an entire screen, a portion of the screen or a single window or icon. SnagIt sends the captured area to a printer, the Windows clipboard or a file. These images can be pasted into other Windows applications such as word processors and desktop publishing programs. This is an ideal tool for anyone who needs to document MS Windows programs (we're using it to do TUG Lines). *Stock number UT-MIS-WIN-023*.

The Sound Blister Disk may be a pain in Microsoft's side, but it's sure a lot of fun for the rest of us. It's the "Poor Man's Sound Blaster Board" (it's really a driver) that enables you to play those Win 3.1 sound WAV files through the PC's speaker. Imagine the stillness of the night being broken by Curly (of 3 Stooges fame) shouting "I'm tryin' to think, but nuttin' happens!" through your PC speaker. It's a riot! The only problem is - nothing productive gets done on the computer any more. (The whole truth: Microsoft didn't release this driver because they found it wouldn't run on *every PC ever made*. But it's run beautifully on every one we've tried, so we're making it available to you.) Complete with a ton of sound files, including "I'm tryin' to think...". *Stock number UT-MIS-WIN-024*.

PC Techniques Disk *16* contains the source code for routines published in Volume 3, Number 4 (the October/November 1992 issue). *Stock number RS-MIS-MIS-815*.

When Ordering Disks

Use the order form that accompanies this issue. Please note that stock numbers ending with "I5" are in 5.25", 360K format. Stock numbers ending with "I3" are 3.5", 1.44M format.

Product Insights

Windows Programming Editors Part 3 - ED for Windows

Dave Chowning
Vancouver, BC Canada

At A Glance...

Product: ED

Version: 1.0
Author: Neville Franks
Soft As It Gets, Melbourne, Australia

Vendor: LifeBoat Software
1163 Shrewsbury Avenue
Shrewsbury, New Jersey 07702-9949
(800) 447-1955 (orders only)
(908) 389-0037 (voice)
(908) 389-9227 (FAX)

Price: $269.00

I obtained a copy of ED from Bill French at the Borland World Tour. It is appropriately called ED for Windows because there was previously an ED for DOS. This tells us it is a port. Remember that Premia Corporation boasts that its CodeWright editor was designed from the ground up as a Windows app. So what is different about a 'ported' app?

Neville Franks has written and spoken about porting DOS apps to Windows. He says that a port from DOS to Windows falls between lots of hackwork and a total redesign and rewrite. According to Neville, DOS programs are usually written as though they were the only programs running. A DOS app waits for keyboard input, meanwhile typing up all a computer's resources: keyboard, screen, memory, CPU, disks, etc. Neville adds that the program is in control, not the user. Windows apps are different in that they share resources, are user driven or event driven. According to Neville, the Windows program relinquishes total control to Windows and responds to commands from Windows and/or the user.

ED for DOS was both a C/Pascal programming editor and a strong database editor. The powerful database features are both the strong and unique points of the Windows version. It would seem to me that from version 1.0 there was an emphasis on retaining the powerful DOS database features, and that some basic Windows features were not included. Some Windows features not available are Windows CUA keyboard support, text drag and drop, DLL support, and printing.

Installation

During installation ED did not place all its files in its own directory; instead ED placed its INI file, EDRUN.EXE and EDRUN.PIF in the Windows directory. The installed program required 2.3 MB for all its files.

Interface and Help

When you first start ED for Windows you see a toolbar that uses color well and adds text to many of the icons, making the toolbar easy for first time users. The cursor is narrow and full height for insert, and a full height block for overwrite. This makes it easy to remember what mode you are working in before you overtype a hard to remember function with parameters!

There is a status line at the bottom of the screen, but I was unable to read it, since the text was black on dark blue. ED, like MS Publisher, is reading pre-set Windows colors: shadow and background color of buttons for the display of the status line. The status line displays INS mode, W for word wrap, time and date, line and column, and filename. There is also T icon which removes or displays the toolbar. Clicking on the line or column areas will pop up dialog boxes for goto line or goto column. Clicking on the INS area will toggle the mode from insert to overwrite.

Help is available with the F1 key in both text editing and in the dialog boxes, but not in the menus. It was very frustrating to try to use a help file that referred to ED functions as *K_something_or_other*. I don't want to learn ED's underlying macro language before I can use the help system.

When the cursor is over a key or reserved word, you can view SDK help by pressing F1, then clicking on the word when the cursor becomes a large question mark. Or you can simply right-click on the word and view SDK help directly. If you are using a language other than C, you can set the SDK help to another HLP file.

ED has a keymap that adheres to the CUA/SAA standard. Menu access and exit is via the F10 and ESC keys. The standard ALT+letter keys also brings up specific pull down menus. The menu system includes access to most of ED's features.

On the surface, ED appears to have a very logical and solid interface, until you begin to look for accelerator or short cut or hot keys. Outside of the default CUA keys, you are confronted with the most convoluted CTRL+letter+letter formats you possibly have seen since the early days of Wordstar.

Compiling

It was easy to set up the Turbo Pascal for Windows 1.5 compiler to work within ED. Starting from the main menu through a series of dialogs: click on TOOL, ALL PROGRAMS, PASCAL, and Turbo Pascal, and EDIT. A dialog box allowed me to enter the DOS command line, or TPCW and <NAME> for directory, path and current filename. TPCW is close in error checking to Microsoft C so I chose that for error tracing. With this I was able to compile successfully both my test program and find errors in my "error" program.

When I clicked on the compile icon on the Toolbar icon, I still had to deal with a Program dialog box from which I had to scroll to the file extension, then choose RUN. The icon should directly run the compiler set up for the extension of the current file. Unfortunately, I could not find a RUN or execute command, so I was unable to run or test my compiled program from within ED. A serious drawback to an otherwise fine editor.

Documentation

The ED for Windows manual contains the best introduction to using a Windows software I have ever seen. You are walked through the entire environment from status lines to toolbars, from file and buffers to working with windows. The introduction and reference guide are in the same 300+ page manual. The detailed Table of Contents and index are well organized for both new and experienced users. There are appendices with screens of Menu Maps, Quick Reference Guide, Templates for code editing, and a Quick Reference for C. In addition to the standard README file and ED4W.REL (which describes the new release), there are several ASCII files describing templates, operator precedence, ASCII table, and the API function guide.

Language Support

In its compiler setup, ED supports C, C++, dBASE, Clipper, DataFlex, Pascal, and COBOL. ED does not have language specific color coding for reserved or key words, comments, and code. ED has a powerful implementation of brace/control/structure/object matching, which extends to BEGIN...END and other control structures. Additionally, there is a DBformat macro command that reformats dBASE/Clipper code.

Customization

With ED you can interactively map the keyboard and create macros, as well as edit the configuration file. ED has a number of ready to load key maps: Brief, Norton, QEdit, Wordstar, and the default CUA/SAA. Although you cannot customize ED's icon toolbar, you can customize the menus. There is no DLL support. ED has a C-like macro language and compiler with which you can extend ED's functionality.

Summary

If you are in the market for a powerful Windows Editor with a professional look, you won't go wrong with ED, especially if you are a C programmer. ED can be easily adapted for Turbo Pascal as well. ED also provides language support for database programmers (dBASE, Clipper, and DataFlex). ED's abilities for dBASE code reformatting and brace/object matching of control structures are unique among Windows editors.

ED for Windows is only release 1.0. Neville called me from Australia and discussed some of the changes that he plans for version 1.1, which include printing, text drag and

drop, and Windows CUA keyboard. He assured me that when he gets a release ready, he will ship it over. When that happens, we will keep you updated. Even though ED is 1.0, it still has a great deal of maturity due to the port from DOS.

Dave Chowning is the TUG/Pro Database Editor and gives college lectures to programming classes on programming editors.

An "Almost Free" Video?!

You may have noticed an ad in the last couple of issues for a video called "Computer Viruses - The Real Story". If so, you may have wondered why we would be offering a video on viruses, supposedly aimed at non-technical people, to a group of professional programmers.

Good question.

When I asked Rob Rosenberger to do the video this summer, it was with a specific intent. I knew what Rob was doing, and I wanted the material available to the public. Basically, the presentation is the same one Rob has given probably 100 times - to groups small and large (including the military). It contains technical information that many of us already know (along with a few things I wasn't aware of). But in addition, there is vital information that *everyone* should be aware of. But you'll never hear it on the 6 o'clock news. (In fact, it contains information the mass media would probably rather you didn't know.) It's true *guerilla video*.

"The Real Story" is finding its way into companies, where it is being used to train personnel who use computers. That's great, but I want to make this info available to a lot more people. If I could afford to give it away, I would. But I can't.

But if you live in the U.S. or Canada, you can do a small favor that will help a lot of people, and in the process you can get a copy of "The Real Story" - *almost free*. On page 20 of this issue, you'll find an ad for the video, aimed at the general public (and at the usual non-TUG/Pro member price). Just copy that ad, and take it to your local library. Request that they buy a copy for the library system. If they do, you (and a lot of other people in your area) will have free access to the vital information on this video.

Thanks.

Pascal

Top of the Heap

Don Taylor

When the UPS driver showed up with my long-awaited copy of Borland Pascal 7.0, I'll admit I was a bit skeptical. After all, Borland C++ 3.0 with Frameworks came in a box the size of an overnight suitcase, and it weighed 14 pounds. But the BP7 box was only about half the size of its BC++ brother. How were the contents of this demure little box going to stand up to that kind of power?

I carefully lifted the box onto the scale. *Fifteen pounds!* Lordy, lordy - I think it's a *record!!*

More than that, though: Borland Pascal 7.0 is the most powerful Pascal compiler Borland has ever shipped.

The Compiler

BP7 is actually *four* compilers, with three Integrated Development Environments: The Borland Pascal compiler, with a DOS IDE that runs in DOS protected mode and creates DOS real mode, Windows or DOS protected mode applications (protected mode means you can create EXEs larger than 640K without overlays); Borland Pascal for Windows, with a Windows IDE that creates DOS real mode, Windows, or DOS protected mode applications; Turbo Pascal, a DOS real mode IDE that creates DOS real mode applications only; and BPC, the command line compiler, which runs in protected mode and can create applications in any of the three modes.

Several of the features of TPW 1.5 have made their way into the DOS IDEs, including colored syntax highlighting. There are Object Browsers, undo and redo, and a whole host of other features. There are additions to the language; new units; enhancements to the run-time library; new tools formerly available only with BC++ (WinSight, which lets you view messaging while Windows programs are running, and WinSpector, which lets you see what went wrong with a program after it's dead).

The Frameworks

BP7 includes both Turbo Vision and ObjectWindows Library. TV is now at version 2.0, and it includes some new capabilities. Both TV and OWL share some capabilities, like the new data validation objects (they even share the same code!). Controls within dialog boxes now have an associated pointer to a validation object. To add validation (filter, range, lookup, string lookup and picture validators are available), just set the pointer to the object. Source code for both frameworks is included in the package.

The Documentation

The weight of the package is mainly due to the more than *3,500 pages* of documentation in the box. In all, there are 11 manuals: the User's Guide, the Language Guide, the Programmer's Reference, the Tools and Utilities Guide, the ObjectWindows Programming Guide, the Turbo Vision Programming Guide, the Resource Workshop User's Guide, the Tools and Utilities Guide, the Turbo Assembler User's Guide, the Turbo Assembler Quick Reference Guide, the Turbo Debugger User's Guide, and the Turbo Profiler User's Guide.

As you can see, there aren't complete sets of manuals for the various versions of the compiler. Instead, the manuals combine to give an efficient overall picture of the product's capabilities. Information has been arranged so it's no longer necessary to have every manual spread out on the desk all the time (my experience with TPW 1.5). The TV and OWL guides have been rewritten and are greatly improved over previous versions. The single most appreciated improvement - a set of tables for each library object, showing in one place the fields and methods available, which ancestor added them, and who (if anyone) overrode them.

The only down side I've seen in the documentation is the lack of a printed API reference with the package. This information is available through the online help system, but I prefer a printed version. The book is available at extra cost from Borland; if you have TPW 1.5, hang onto the Windows Reference Guide.

The Experience

I have for some time been meaning to write my first Win app. But for whatever reason, I've never found the energy or the time. But when BP7 arrived, I was just at the right point. I needed a utility to massage ASCII text files in ways that it would make it easier and more accurate to import them into a word processor under Windows (in this case, Word for Windows 2.0).

I couldn't afford full-time on the project, but over the period of four and a half days, I went from opening the box to completing my first Win app. It's called ParaGraf, and it comes complete with dialog boxes, buttons, check boxes - even its own icon. I won't say it wasn't frustrating at times, but I didn't suffer massive hair loss either. All in all, it was a pretty enjoyable experience.

The Recommendation

The list price of BP7 is $495. Considering everything you get, that's not bad - and at a projected street price of

about $350, it's a bargain. Borland is offering upgrades from TP6 and TPW 1.5 for $149.95. My recommendation is this: Just Do It. If you plan to make your living programming in Pascal, you'll be able to write better programs. And you'll probably make a better living.

The Gift

You may have guessed that I'm going to make you a present of a copy of ParaGraf. And you're partially right. On the Newsletter Disk, you'll find a compiled version, along with the full source, plus resource files in both source and compiled versions. It's a useful utility (I'm only on page 11 of this issue, and I've already used it almost a dozen times, just doing this one job). Have fun.

Now for what Paul Harvey might call The Rest of the Story. Many times in the past, I've written code and dumped it into the public domain. We all do it; it's part of what we do to repay those who went before us. But I'm also here to benefit you directly. So here's what I'm going to do...

I'm retaining full rights to ParaGraf. It does not belong to the public domain. Your copy belongs only to you. Everyone in the world isn't going to have a copy. Now for the good part. I'm giving you a license to make as many copies of ParaGraf as you like - you can sell them or give them away. You won't ever owe me a penny for it. All I ask is that you keep my copyright notice on the About dialog and print it on any labels or documentation you create for it.

Let me be clear: your license enables you to make as many copies and distribute them as you like. But you don't have the rights to pass that same privilege onto someone else. They can't reproduce the copy they receive and distribute copies.

Okay, think about this for a moment. If you have a product for sale, you can use it as an added bonus. If you're producing shareware, you can use it as an incentive to get people to register. If you want to get a little aggressive about marketing, you can actually *make money* on this (hey - you can even pay for your TUG/Pro membership dues!). Depending on how you apply it, this one program could pay for your membership several times over.

If you like this idea, let me know. Perhaps we can do more of them in the future.

Inside the BGI: `GetImage`

Bob Crawford
Bowling Green, Kentucky

I have been asked to write a series of articles on image processing, BGI drivers, and the like. I wonder a bit about the wisdom of this, since these are not something I know a lot about. On the other hand, it certainly is an opportunity to learn something new, so I guess I will give it a fling. Being only an "expert in the making" (at best), I'm going to start with some very elementary stuff - a look at the way Turbo stores images when you grab them off the screen with `GetImage`. Once we understand this, we may be able to do a bit of image manipulation right in the native BGI format. Much of the stuff I'm going to tell you is undocumented - but it is fairly easy to guess, because there is no compression going on. Nevertheless, you should be aware that you are being fed *deductions*, not "facts".

Inside `ImageSize`

As you know, before you call `GetImage`, you have to have a buffer of an appropriate size to store the image, and you are supposed to use the `ImageSize` function to figure out how big this buffer should be (and it can't be larger than 64K). The first item of business is to see where the values returned by `ImageSize` come from. Of course, a call like

```
ImageSize(50, 50, 150, 83)
```

will return different values for different modes. It all depends on the number of colors available - 2, 4, 16, or 256. If you have only 2 colors, a single bit can store all the information needed about a pixel. With 4 colors, 2 bits are needed for each pixel. A pixel on a 16 color screen takes 4 bits, and one on a 256 color screen requires 8 bits. If we let `BitsPerPixel` be a variable representing the number of bits needed for each pixel in the image, it turns out that a rectangular region of the screen `Width` pixels wide and `Height` pixels tall will take

```
6 + Height * BitsPerPixel * ((Width + 7) div 8)
```

bytes for its storage. This is the value returned by `ImageSize`.

Here is a little rationale for the formula. The summand of 6, which does not vary with the number of colors, comes from the fact that Turbo reserves 3 words (6 bytes) for its own use. The statement in the manual is "The first two words ... store the width and height of the region. The third word is reserved." We'll see about that, but in any case, a total of 6 bytes is reserved. To understand where the rest of the expression comes from, we need a couple of general principles. First, storage for the image is going to be done scanline by scanline. Second, for each scanline, the storage is going to be done in byte-size

chunks (of course), even if this means that there are a few extra bits left over. The question is - how many bits are needed? Suppose for the moment we are in the 2-color mode, so each pixel corresponds to a single bit. Since the region is `Width` pixels wide, when we group the bits into bytes, the number of bytes we need is almost `Width/8` - but not quite. What we actually need is the smallest integer greater than or equal to the fraction `Width/8` -- the thing that is usually called the Ceiling of `Width/8`. That is precisely what the computation `(Width + 7) div 8` does. If `BitsPerPixel` is greater than 1, we are going to need `BitsPerPixel` times as many bytes as we require in the 2-color case, which accounts for the `BitsPerPixel` factor. Finally then we need

```
BitsPerPixel * ((Width + 7) div 8)
```

bytes for each of the `Height` scan lines, which yields the final formula. (See the `ImgSize` program on the Newsletter Disk for a verification of the formula.)

Inside the `GetImage` Buffer

Having taken care of the `ImageSize` question and gotten a glimpse of the general approach to image storage being taken by the BGI, let's turn to a few details - and in the details are hidden some surprises. The Turbo documentation states in several places that the third word of the buffer used by `GetImage` is "reserved". *Not!* The storage of the bitmap begins with word 3 (byte 5). The very last word of the buffer is, as far as I can tell, always unused, and in a couple of modes some additional space also appears to be unused.

In the 2, 4, and 256 color modes, the storage that begins with byte 5 is a simple linear map. For example, in the 4 color mode, suppose one line of your image contains five pixels having colors 0, 1, 2, 3, 1. To get the storage used by this line, we convert these values to 2 bit binary quantities, line them up, and then pad out with 0s until the total number of bits is a multiple of 8:

```
00 01 10 11 01 00 00 00
```

Note that 2 bytes are required for this 5-pixel line. For a multi-line image, the single-line representations are just laid end-to-end, following the width and height information. Suppose our little example had two 5-pixel lines. Then the total storage actually required is the 4 bytes for the width and height, together with 4 bytes for the actual pixel data. This total of 8 bytes is two bytes short of the value of 10 reported for such an image by `ImageSize`. One of those (I suppose the last one) is the reserved byte mentioned in the manuals. The other one - who knows? I suppose trying to account for it would complicate the formula for `ImageSize`, without ever actually saving much space.

The 16 color mode is a bit different, since it follows the bit-plane organization of the EGA and VGA. To get an explicit (and simple) example of the storage involved, I used the following little procedure to draw a tiny picture in the upper left corner of the screen:

```
procedure DrawPicture;
var
  x : integer;
begin { DrawPicture }
  for x := 0 to 4 do
    begin
      PutPixel(x, 0, x);
      PutPixel(x, 1, 15 - x);
    end;
end; { DrawPicture }
```

The layout of color numbers produced by this is

```
 0   1   2   3   4
15  14  13  12  11.
```

I then saved this to disk and used a little dump program (`ImgView` - on the disk) to look at the interesting part of the storage (beginning with byte 5):

```
0    8   48   80  248  240  200  168  232  247
```

Now, just where do these numbers come from? Let's take a look at the first row. Since there are only 5 pixels, append three 0s to the end of our list of color numbers to give a total number of pixels divisible by 8. Now write the 4-bit binary expansion of each number under it in a column, with the most significant bit at the top:

```
0    1    2    3    4    0    0    0

0    0    0    0    0    0    0    0
0    0    0    0    1    0    0    0
0    0    1    1    0    0    0    0
0    1    0    1    0    0    0    0
```

Now regard each row as a binary number:

```
00000000 ------>   0
00001000 ------>   8
00110000 ------>  48
01010000 ------>  80
```

Lo and behold! There are the first four bytes of the image. The second row (padded to 15, 14, 13, 12, 11, 0, 0, 0) gives rise to the next four bytes in a similar manner. The final two bytes are unused. What is going on, of course, is the four bits in a color number in this mode are being used to represent the four bit-planes in the EGA/VGA board - they can be sloppily thought of as Intensity, Red, Green, and Blue.

I don't know about you, but this kind of complexity is enough to deter me from trying to do any kind of serious image manipulation using this native BGI format. It did not, however, deter me from a little fooling around, the results of which are to be found in the `BGIImg` unit and the `ImgFlip` program on disk. The `BGIImg` unit provides a `TImage` type with routines to flip and invert the image (along with a few more obvious methods). These routines are just toys - the format is built for efficient getting and putting of screen images and not much else. Next time we will get a bit more serious.

Bob Crawford *is a professor in the Computer Sciences Department at Western Kentucky University, and a contributing editor of TUG Lines on a variety of subjects. He can be reached on CompuServe at 71121,777.*

Object Morphing: State Programming with Turbo Pascal Part 1

Paul Gallagher
Mt. Waverley, Victoria, Australia

It seems these days that any Turbo Pascal programmer worth his salt is "into" objects. We spend our time building object hierarchies fanning out every which way. If you're like me though, it's not long before you start feeling that the object paradigm and syntax we have to work with are perhaps not as good at representing real world problems as we first thought.

Roots, Branches, Language Limitations and Hacks

The problem I keep running into is summarised by the truism: "tug at any tree long enough, and you'll soon find out that it has roots as well as branches!". As much as I like Object Pascal, the current language specification restricts you to "growing" object functionality (by inheritance) in one dimension (a limitation found in all O-O languages that I am aware of). However, when we set about modeling real world objects, it would often be nice to have an *elegant* way of extending an object's function in many independent dimensions.

What do I mean? Say you wanted to write a truly generic communications object hierarchy. One facet of the problem is to build the "Protocol Stack(s)" which would control how the communicated data is interpreted. A "basic" protocol object could be defined, and descendants could handle progressively more advanced protocols. The "protocol" concept grows nicely in the one dimension. A second dimension to the problem is the actual hardware interface used to send/receive the data. You'll want a range of network cards for example, perhaps RS232, and the ability to inherit basic functionality and add new interface drivers at a later date.

Conceptually (see Figure 1), the protocol stack sits on top of the interface driver, but to descend the protocol stack from the driver object means you've just blown away all the opportunities for "generically" extending the driver's functionality. You've killed the real world object's second dimension!

What we do, of course, is model the two dimensions as separate classes. The "fudging" (or the "big hack" if you prefer) occurs when attempting to link the dimensions while still maintaining a generic, extensible design. Commonly, the "primary dimension" (in this case the Protocol Stack), will have a field which is a pointer to the base class of the "dependent dimension", in this case the Driver class. Initialising an instance of the primary

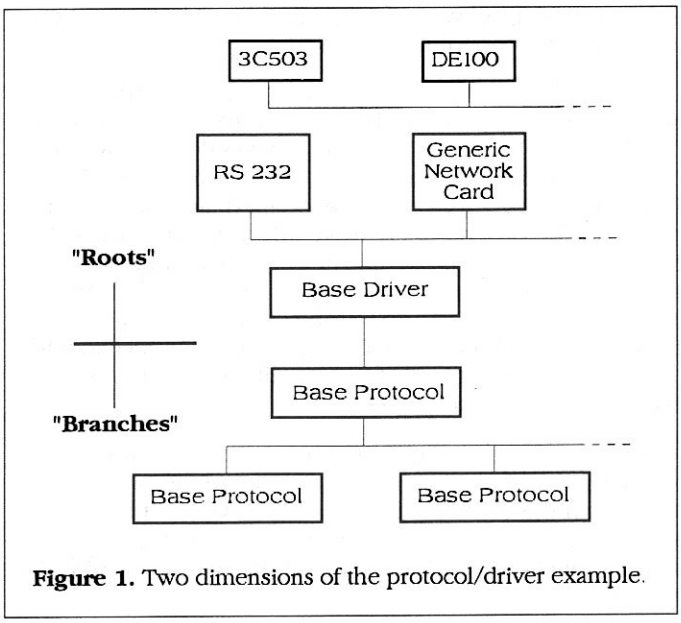


Figure 1. Two dimensions of the protocol/driver example.

dimension would involve passing it a handle to the desired object from the dependent dimension (e.g., we would give a TCP protocol stack a 3C503 driver). This does work, but there is still an ugly bit of code out there that does something like this:

```
function MakeMeADriver(driverType : byte):
                          PBaseDriver;
begin
 case driverType of
  c_RS232 : MakeMeADriver:=new(PRS232Driver,init);
  c_3C503 : MakeMeADriver:=new(P3C503Driver,init);
  c_DE100 : MakeMeADriver:=new(PDE100Driver,init);
  ...
 end;
end;
```

Why is this ugly? Consider maintenance of the code. If you or someone else adds a new driver, the mechanism for selecting the appropriate value of `driverType` may well be abstracted enough to not require modification, but going back and revising the `MakeMeADriver` function is unavoidable. Definitely not good object oriented practice. Extending the capabilities of an application should only involve adding code blocks, data or object definitions - *not* adding or changing lines of code in existing code blocks.

Utopian Dream #1: Can we instantiate a base object, and say "today, I would like you to be a ... `driverType` object" - like this:

```
function  MakeMeADriver(driverType : word) :
                          PBaseDriver;
var driver : PBaseDriver;
begin
 driver:=new(PBaseDriver,init);
{ change my type to something descended
  from TBaseDriver }
 driver^.Morph(driverType);
 MakeMeADriver:=driver;
end;
```

What if we wanted to change the network interface during the lifetime of the protocol object? It could be done (as I will discuss in Parts 2 and 3), but why would we need to do it? After all, someone's hardly going to come along and change our network card while we're executing, are they?

Let's turn the problem upside down. Consider the driver class as the "primary dimension", and the protocol stacks as the "dependent dimension". It is quite possible, even *probable*, that the protocol required may change during a communications session.

In Figure 2 I've represented a "custom" protocol as an object called `MyProtocol`. This fictional protocol works in two modes: "fast" or "reliable". Perhaps the fast mode uses larger block sizes and less rigorous error correction. The modes are represented by descendants of `MyProtocol` which over-ride key methods. The real world situation we want to model probably requires the protocol implementation to switch from fast mode to reliable mode, based on the proportional number of errors encountered or some other measure of line quality. We want a `MyProtocol` object that can seamlessly change from normal state, to fast mode, to reliable, back and forth as required.

The key point here is that we want objects to be able to change *themselves*, not rely on some external factor to determine how and when the object should be retyped.

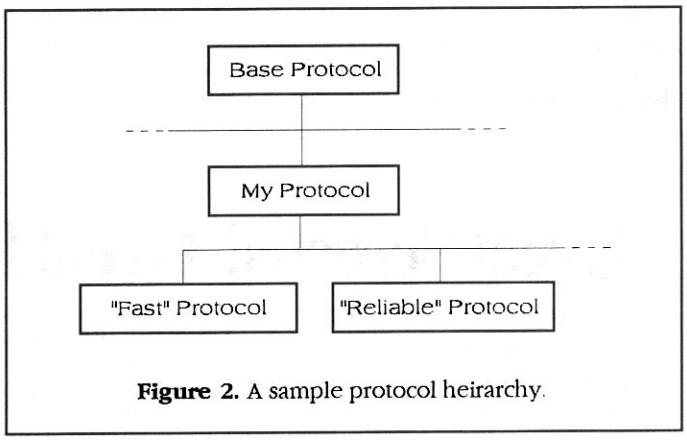


Figure 2. A sample protocol heirarchy.

As far as application components that use the "chameleon" object should be concerned - it's business as usual. Nothing's changed.

Utopian Dream #2: We want an object instance to be able to dynamically retype itself as required by changing circumstances, without affecting its "owner".

If we could get our objects to behave in this way, we've gone beyond object orientation, beyond event-driven paradigms ... into the realm of *state programming*.

In Part 2 we will explore our Utopian Dream, and we will discover a mechanism that enables us to realise a new form of inheritance that lets us actually *change* the functionality of objects - not just add to it.

***Paul Gallagher** has been a Turbo Pascal fan since version 3. He works in PC/Mac Support for BHP Research - Melbourne Laboratories. Between playing with computers at work and playing with computers at home, he finds some time to make music and beer. You can reach him at PO Box 731, Mt. Waverley, Victoria 3149 Australia, or through internet mail to paulg@resmel.bhp.com.au.*

C/C++

The C Scape

Tim Gentry

Why is it that programming languages become such a bone of contention between programmers? In some recent issues of a major computer magazine (which shall remain nameless) several letters were published espousing the virtues of <language X> over the inadequacies of <language Y> and that programmers who use <language Y> are either stupid or foolish. This is arrogant rubbish. Different people use different languages in different ways for different purposes. I personally used to use Pascal and thought it a marvelous language. My Pascal skills were dulled, however, by two years of maintenance coding in COBOL and when I had the chance to change I took up C - and never looked back. Was that because I thought that C was inherently "better" than Pascal? Not at all. C merely matched up better to the way my mental processes worked than did Pascal. And when C++ rolled around I saw that it was closer to the way I *wanted* to think and program, so I made the switch.

All of these changes were not driven by some inane concept that any given language is superior to another. They were merely driven by the fact that C (for example) was better *for me* than was Pascal. That's the key concept: Programming language FOO is better than programming language BAR only because I can churn out faster, better, and more understandable code more efficiently using FOO than I can using BAR. If BAR works better for you then absolutely no attempt should be made to force you to use FOO or to try to convince you that you're making some sort of grave mistake - and there should be no "stigma" at all for using BAR instead of the "superior" FOO.

"Language Bigotry" has another, more subtle side to it as well. All too often programmers try to "make it work" in one language when another language or tool would make life so much easier - all because they feel some sort of dedication to their favorite language. To those programmers I say: "Get a clue!" Languages don't have feelings - they can't be offended. If it would be easier or more efficient to develop your user interface using Actor (or something similar) then do it! The customers who pay for your services couldn't care less what programming language you used to develop their product. They only care that it was developed in the shortest possible time and that it does the job they need.

The article by Alan Thomas and Pete Rothermel in this issue would seem to violate this concept and in some respects it does. I initially didn't want to include it because I didn't want to contribute to the language wars. But it does, however, contain a number of insights into C++'s concepts and design issues, so I present it here for you. I only hope that it doesn't generate a flurry of letters trying to rebut one or more points made within the article.

Of course, all of this rambling leaves the obvious question: Do I think that C is better than COBOL? Um... well... - I refuse to answer that question on the grounds that it may incriminate me. Stop me quick, before I start tearing the tags from the sofa cushions...

***Tim Gentry** is a professional C++ programmer for Boeing Computer Services. When he isn't programming you'll probably find him on his motorcycle, looking for the most twisted route from FOO to BAR.*

Library Notes

It has been brought to my attention that the Borland Custom Controls disk that we put in the library in the last issue could use some sort of companion "entry level" disk. Your wish is my command. This time I'm adding a 1.44M disk containing about a megabyte of custom control tutorial, starting from the very basics. The entire package was developed by Bob Bourbonnais at Borland, and covers:

- Basic DLL's,
- Basic custom controls,
- Passing data in custom controls, and
- Multiple custom controls in DLL's.

The executables on this disk require DLL's that come with Borland's language products, including BWCC.DLL. Except for BWCC.DLL these files are not included on the library disk, and the version of these DLL's already residing on your system will be used.

Here's an update to the annoying RCS behavior I mentioned in the last issue. For those of you who missed it last time RCS started life under UNIX, where filenames can be umpty-squat characters long. The RCS delta files (where all the revision changes are stored) were simply given the name of the original file with ",v" appended to the filename. This behavior is mirrored in the DOS version, which limits RCS to working with files with 1-character extensions. The problem appears to be in the file RCSFNMS.C, in the function `pairfilenames()`. This function generates the RCS filename based on the original source filename and should be modified to work differently under DOS. The behavior I'm looking for is to convert the source extension using the mask "._V". This means that if I have a file with no extension the RCS extension will be "._V". A one-character extension will become ".X_V", a two-character extension will become ".XXV", and a three-character extension will have its last character replaced by "V". This mirrors the behavior of PVCS, and should work just fine under DOS. I'll have a patch in the next issue - unless one of you beats me to it.

A Dissenter's Opinions on C++

Alan Thomas and Pete Rothermel

[Editor's note: Alan Thomas wrote TUG/Pro with the following viewpoints on C++. In the interest of fair coverage we took Alan's observations to a C++ and OOP instructor, Pete Rothermel, for his analysis and rebuttal. Alan has not been given a chance to respond (that could go on forever...), and both sides comments appear uncut and unrehearsed. Alan's side of the conversation is in normal print, while Pete's appears in *italics*. The author's biographies appear at the end of the article. I've included this in this month's column not to defend C++, but rather to show some of the important points that Pete brings out in regard to C++ design. My attitude (and TUG/Pro's as well) is that programmers should use the tools and languages that get the job done - you'll find no language bigotry among the TUG/Pro staff. Well, OK - maybe just the tiniest bit... ;-)]

In the interest of saving reading time for C++ fans, let me begin by saying that C++ is a dangerous tool with too much power for most problems. Like other dangerous tools, C++ is perfectly safe in the hands of a skilled user, but the costs associated with becoming a "skilled" user are significantly higher than for another object oriented language - Turbo Pascal v5.5+.

The cleanest explanation of the difference between Turbo Pascal and C++ is that Turbo Pascal is a language for programmer/analysts while C++ is for programmers. P/A's combine people and computer skills... Programmers combine computer skills with computer skills. Programmer/Analysts often wear ties ... Programmer's may not even own one. Programmers understand system internals with little effort ... Programmer/Analysts think of interrupts as things which break up conversations with application users.

C is a "Programmer's" language; Turbo Pascal is a Programmer/Analyst's language.

I don't really follow any of this. I'm not quite sure from the above discussion if I'm a Programmer or a Programmer/Analysts. What about software engineers and computer scientists? My experience on software projects is that such labels often restrict the pay and opportunities for certain team members and do not contribute anything to the team's success.

Let's inject some reality here. Someone proficient in any language can do very good work on just about any problem domain (provided they know enough about that domain).

"The hard part of software development is the manipulation of its essence due to the inherent complexity of the problem, rather than the accidents of its mapping into a particular language which are due to temporary imperfections in our tools that are rapidly being corrected." - FP Brooks

When we talk about "favorite" programming languages, what we're talking about is a personal choice closely akin to that of spouse or significant other. We generally don't know how to talk about these things and we almost always get emotional when we try.

Its easy to get carried away. If you try to stay calm though it can be done.

C++ supports both strong typing and function prototypes. Both features protect me, my project budgets, and my users against my favorite errors: type mismatches and ridiculous pointer operations.

You will probably find these features in most of the newer C compilers as well. The looooooooong awaited ANSI C definition has function prototypes and strong type checking features that are similar though not exactly the same as C++.

For that reason, I welcomed C++ as a useful response to my criticisms of C. I've been working with C (Microsoft 5.1) and later C++ (Borland C++) for just over a year. Along the way I've gotten advice from mediocre, good, and great C programmers; and I've picked their brains about the languages as tools.

What is it about C++ that leads me to think unkindly of the language? It could be that I'm a wimp ... by now several "real" programmers have already stopped reading this piece after forming exactly that conclusion.

Operating overloading is a "benefit" of C++. From a maintenance perspective, operator loading requires additional time to analyze and understand a module's expected and actual behaviors. That it takes longer to understand is bad enough; that operator overloading may completely confuse the poor schmuck who has to maintain the module is even worse. Maintenance is a fact of life in application development; requirements change and whether they change quickly or slowly, bugs will crawl out of the code to gnaw on users and operators.

I sense at this point some confusion on some basic Object-Oriented concepts. I'm going to lecture a little bit here but please bear with me. I'll get back to why operator overloading helps reduce maintenance costs.

OPERATIONS are functions or transformations applied to or by OBJECTS. All the OBJECTS of a particular CLASS (type) share the same OPERATIONS. Many CLASSES may have the same OPERATION as part of their behavior. A METHOD is a specific CLASS's implementation of an OPERATION.

As an example consider the two classes SQUARE and CIRCLE. Both the classes have the operation ROTATE as part of their behavior. Class SQUARE will have a somewhat complicated rotation METHOD involving coordinate transformations as part of its class definition. Class CIRCLE will have a rotation METHOD that does nothing. If the rotate operation is applied to an OBJECT of type SQUARE, the complicated SQUARE class's rotation method is invoked. If the rotate operation is applied to an OBJECT of type CIRCLE, the CIRCLE rotation method that does nothing is invoked.

All this seems Logical as long as the class's method logically maps to the operation that it is implementing. If the CIRCLE class's rotate method erased the CIRCLE then things would be very confusing but developers usually don't do such things.

As a second example consider the class Complex that models complex numbers that have both real and imaginary parts (Like FORTRAN built-in type COMPLEX). There is a mathematically correct way to add two complex numbers. We would expect the Complex class definition to include a add method that implements addition of two complex numbers. The operations subtract, multiply and divide would also have methods.

```
Complex a , b , c ;
c = a.add(b) ;
```

This code means that we apply the add (addition) operation to the object "a" with the argument object "b" and assign the result to object "c". As long as the maintainer understands the class Complex mathematically this is fairly clear. Operator overloading allows the developer to overload the "+" operator so that the compiler will call a class's method for addition when the plus operator is applied to an object of that class.

```
c = a + b ;   // this replaces c = a.add(b)
```

This code is a little cleaner with operator overloading. It does require that the maintainer know the behavior of class Complex but they probably should not be doing much with any code if they do not know the classes that are used within it. This also assumes that the developer did not do anything malicious like overloading the "-" operator to do division.

Which is really better: "a.add(b)" or "a+b" ?

My OO/Smalltalk/Flavors/C++ side of my personality likes "a.add(b)" which reads somewhat like "send an add message with an argument of object b to the object a". My Engineering/Math side likes good old "a+b". Operator overloading can significantly reduce maintenance in large expressions using our Complex class.

```
c = (a + b) / e - b*(g-k*l);
```

vs.

```
c =
(a.add(b)).div(e.sub(b.mult(g.sub(k.mult(l))))));
```

Objects and methods can be made accessible to virtually everybody nobody else, or to friends, associates, and to unindicted co-conspirators. What does this do for maintenance?

If the classes in an application are designed so that the implementation details are hidden in the private section and the methods that support the operations (behaviors) of the class are in the public section several important advantages are gained. The C++ class construct with its access modifier provides support for implementing encapsulation and data hiding but the designer of the classes must come up with the design of the classes.

One advantage gained by encapsulation and data hiding is that these kind of classes tend to be rather independent of other classes. These Abstract Data Types (ADTs) are easier to put on the self as reusable components.

Another advantage of ADTs is the limitation of the amount of code that needs to be debugged when you have a problem with the class. During the execution of the application you notice that an object of an ADT has a "bad" value for one of its data attributes. The value of the object's data members can only have been set within one of the member functions for that class because all the data members are in the private section. If any regular function or any member function of another class had tried to access private data members the compiler would have issued an error. This means we have a bug in a member function or we are possibly passing bad data in the arguments to a member function. In any case we have a good start in any investigation. In large systems with thousands of functions we may have saved an enormous debugging effort for the maintainer of the code.

A third advantage has to due with enhancement of maintained software. Lets assume for a moment that we find a much better way of implementing the details of an ADT but it requires that many of the data members change type, are no longer needed, or additional data members are needed. This will probably cause some of the member functions to break. When we rewrite the member functions to provide the same interface to the class we may or may not to change the arguments to some member functions. If the arguments to all member functions remain the same user's of the ADT only need to recompile and link without changing their code to get the improvements. If changes were made in any member function's argument-list the compiler will pin-point those lines of the user's code that need to be modified. If the data of the ADT class was not private the amount of modifications required to user's code may have been very extensive. How many times have you been told you could not improve something because it would break too much other code?

The access modifier cannot be changed without exposing the application to unexpected side-effects whose behaviors may be completely unrelated to the change made ... as viewed by the maintenance programmer. Stated differently, changes in classs "openness" can lead to unexpected and un-obvious problems.

The result of changing the access modifiers of classes should not cause any unexpected problems. If something was accessible because it was in the public section and you move it into the private section you may get a lot of compiler errors that should be very specific about what happened. If you weaken the data hiding of a class by moving implementation details

into the public section you will not get any errors. The problem here is that user of the class may start accessing data members directly rather than via member functions. If this happens you will have lost the advantages that were discussed above.

Inheritance means that core concepts can be reused by descendant objects.

Inheritance is the sharing of attributes and operations among classes based on a hierarchical relationship. A class can be defined broadly then redefined into successively finer subclasses. Each derived class (subclass) incorporates or inherits all the properties of its base class and then adds its own unique properties.

This factoring out of common properties can greatly reduce repetition within designs and code. Maybe more important is that grouping together of classes into hierarchies is something that we seem to do naturally as humans. A base class and a derived class have an association between themselves that is often referred to as an "is-a" relationship. Inheritance is an Object-Oriented theme. It is not really in the structured analysis/design world.

Inheritance also means that bugs and defeats in a method can be propagated throughout an application which doesn't explicitly incorporate the offending method at all.

To me it means that a lot of duplicated code is removed. The more places that code is duplicated, the more chances that an error will occur. Also if any maintenance is required in a method of the base class it needs to be done in just one place. If a bug is suspected in a base class the encapsulation and data hiding features of well designed classes will limit the scope of the debugging effort to member functions of the base class. Once a bug is fixed in a base class it is fixed in all the derived classes as well. The testing effort of software can be very difficult and expensive. Using inheritance to increase code reusability can greatly reduce these costs and increase software reliability by deriving new classes from well-tested mature classes.

Inheritance means that our source code module is intimate with many faceless modules, and who can tell where the code has been, who it's been with, or what was done at that time?

Your source code is also intimate with all the underlying system calls that get made at level below your application code. Does it make you nervous that you have not reviewed the source code for these system calls? I hope not. You probably expect that the code was thoroughly tested and bugs will be fixed as they are reported.

Inheritance complicates rather than simplifies maintenance. A defect fixed in a primitive object has been fixed throughout the family tree of that object... except where the original defect was overwritten by a descendant - or was masked by an intentional or unintentional side effect. How is a maintenance programmer going to figure this out?

The importance is shifted from source code to object/classes and associations between and hierarchies of the classes. As an analyst learns the semantics of the classes and understands how the classes are related, maintenance of the software becomes easier.

Finally, how long does it take to feel that one has mastered a language? For me, 15 months of part-time involvement with Turbo Pascal was sufficient. Thirteen months with C and C++ have convinced me that I'll never feel as confident about C++.

This is a very individual thing. Often maintenance wizards don't get enough time between putting out fires to really get to use the language. A friend of mine had been maintaining a large C program for 3 years when she surprised me by saying that she wanted to take an entry level C class. The C program was an old FORTRAN program that had been modified with minimum changes to make it through the C compiler. She had spent all her time maintaining this monster and she wasn't sure that she remembered how to use pointers.

Where are the benefits that go with those liabilities? Are the benefits applicable to application domains or to just tools and building blocks?

There is enough "Programmer" in me that I'll stay with C++, but the selection is driven by emotion rather than by rational thought.

Another motivating factor may be money. C++ is probably not my favorite programming language (I think Algol68 was but I don't remember) or even my favorite Object-Oriented programming language. But it dwarfs Smalltalk/Eiffel/Flavors/Lisp/ObjectPascal in number of companies that are using it and is gaining popularity daily. I just don't see that many job listings that require Turbo Pascal skills but there are a lot that require C and C++ experience.

Alan Thomas
Plano, Texas
CompuServe 72570,1116

Pete Rothermel
Seattle, Washington
Internet: pete@luey.ca.boeing.com

Alan's opinions are the result of just over a year's work with C and C++, a little over two year's work with object-oriented design and implementation, five years of Pascal experience, and 17 years working in and around large-systems application maintenance and development.

Pete works for Boeing Computer Services on the REDARS distributed image database system using UNIX and C++. He also teaches C++ and object-oriented design in his copious spare time.

Database

Covering the Bases

Dave Chowning

The Borland World Tour - 1992

Most of us received the flyers advertising the World Tour. "Let the experts teach you how to use 100% of your software's power in just one day!" We were assured the Tour was designed to give us a quick start over learning by hand on manuals, increase our productivity, advance our careers, eliminate the need for outside consultants, become database development experts.

I attended the session in Vancouver, Canada, along with professionals from all types of "shops": MIS, small micro shops, independent consultants and developers, in-house programmers. The sessions were excellent, covering all the things programmers needed. The sessions were intensive classroom lectures, with instructors using computers in front with overheads.

An overview of the sessions:

- Turbo Pascal for Windows included the mechanics of OOP, ObjectWindows, graphics and resources.
- Introduction to C++ included creating types, inheritance and virtual functions.
- Advanced topics in C++ included templates, multiple inheritance, assignment and initialization.
- ObjectVision - covering the fundamentals of visual programming with Decision Tree Logic, database links, and extended Windows capabilities.
- Paradox for DOS - PAL Application Development from an introduction to PAL to procedures, libraries, and multi-user issues.
- Paradox for Windows - ObjectPAL Programming and Development from migrating from PAL to ObjectPAL to programming with ObjectPAL and debugging Paradox for Windows Applications.

I attended Mike Irwin's dBASE IV "Programming", although I desperately wanted to attend Cary Prague's "dBASE Design Surfaces" which was scheduled at the same time. (I had read both men's writing in magazines - and Cary's books as well - during my growth as a database programmer.) Michael Irwin had just retired from the Washington D.C. police department after 20 years' service. He has relocated to the Cincinnati area to work as a consultant. Michael is a long time dBASE expert. I also attended Bill French's sessions on the dBASE for Windows compiler. Bill is vice-president of Global Technologies in Denver, Colorado. In addition to a number of dBASE utilities, Bill is best known as the creator of dBrief for the Brief editor. He has written a similar add-on product for Multi Edit called Evolve, which will be reviewed in *Covering the Bases* in the next issue..

The dBASE IV sessions, "Programming" by Michael Irwin and "Design Surfaces" by Cary Prague, included a variety of programming and interface issues: complex indexes and QBE, screen forms and data validation, coding tricks with UDFs, new commands in version 1.5, and a sneak preview of the Borland Desktop. Mike's business-like yet easy going approach made it a pleasure to spend the day in his seminar.

The sessions on dBASE For Windows SDK and Compiler was a clarion call to DOS database programmers to get into dBASE IV and TPW or TC++W. Bill French covered the introduction to the pre-beta Windows compiler and SDK, new language syntax and extensions, and the new debugger. At each and every turn Bill was referring to tools and concepts that those of us who attend GeTUGether have been hearing for 3 years: get into Windows, get into OOP, and start using the tools - IDE, Resource Workshop, and Debugger. These tools will be both similar and essential for database programming under Windows.

If you missed this year's Borland World Tour, don't miss the '93 tour. Get after your boss now, or start saving for it. It is worth it. We're not talking about good-feeling-we-are-it togetherness here, we are talking techniques and code and tools and tips and jobs and futures.

Software Update

Quik Reports for Windows version 1.1 is out and 2.0 should be released by the time you get this issue. Release 1.1 is called Crystal Reports and provides Paradox 4.0 compatibility, word wrap for memo fields, the ability to select a printer from inside the report writer (rather than going to the Windows print manager), the ability to print to file, and enhanced formulas with global variables.

With Release 2.0 you will be able to add bitmap graphics (BMP and PCX) to reports, along with borders and shadows around fields. Line drawing will be added. Objects created in OLE can be added to reports. Extensions to the report engine DLL allow developers to write their own functions and add them to the formula language. A rule and status bar have been added to the interface. Another feature developers will like is the report compiler that allows you to provide users with an icon to create a report. There is also a SQL Edition which provides access to SQL Server, Oracle Server, and Gupta SQLBase, without any knowledge of SQL.

Dave Chowning is a professional database programmer and instructor who resides in Vancouver, British Columbia Canada.

Calling All Authors!

If you have written an application using C or Pascal with CodeBase, Paradox engine, TOPAZ, or a SQL library; ObjectVision, with your own functions in a DLL; a third party DLL product with ObjectVision; OLE or DDE with Quik Reports and ObjectVision; SQL with dBASE or Paradox files; or code generators or CASE tools for database systems, we want to hear from you! If you're not comfortable writing for publication, we can help you by co-writing the article!

Contact TUG/Pro at the editorial address, or contact the database editor directly at Box 101 Station A, Vancouver, BC Canada V6C 2L8, 604/434-1348.

The Personal Touch

Don Taylor
Poulsbo, Washington

Most of us technical entrepreneurs tend to develop product delivery and support systems designed for efficiency. After all, a customer shouldn't have his or her time wasted (and neither should we), right?

Couple that effect with the time pressures we all face and the fact that many entrepreneurial endeavors are one- or two-person shops, and the logical result is the installation of a message machine on the business phone line. Customers can call in 24 hours a day, they can hear what you have to say, and they can leave a message - perhaps even an order for your product. It's easy, it's cheap and most of all, it's efficient.

A lot of small companies have installed inexpensive voice mail systems that enable them to sound like the "big boys". Callers can be presented with an entire menu of "press a number" options that offer a wide variety of information, and they can even leave a voice message. Once again, this is very efficient.

But efficiency isn't necessarily all your customer is looking for. Certainly, today's buyer doesn't want to waste their time, whether they're a corporate buyer ordering a gross of imbedded systems boards or an individual ordering a takeout pizza. With the hassles we all face, with the incompetence we must all deal with, none of us needs one more challenge today. We all desire a measure of control over our situation. We all want to feel like we're being heard.

Our customers and clients are no different. They may like having several pizza selections to choose from, but they especially appreciate having the option to pick exactly the toppings for their pizza. You can measure in seconds the amount of time it takes for them to find and explore the "options" or "preferences" selection on the menu of a new piece of software. And when these people call you on the phone with a question, a problem or an order, they may well see your message machine or voice mail as a stumbling block and not a solution.

Maybe you've heard the story of the man who cut his right hand and was encouraged by his wife to go to the hospital. As he entered the emergency room, he was faced with two doors. Over one, the sign said "Over 50"; over the other was a sign that said "Under 50". Since he was over 50, he walked through the appropriate door. He found himself faced with two more doors: "Upper Body" and "Lower Body". He decided his hand constituted an upper body injury, so he went through that door, finding two more doors, "Internal" and "External". His injured hand was definitely an external injury, so through that door he walked, only to find - you guessed it - two more doors, marked "Major" and "Minor". He looked at his hand, decided it was a minor injury, and walked though the door, which turned out to be an exit door that put him back out in the parking lot. When he got home, his wife asked him, "Did you get help at the hospital?" He thought for a moment. "No," he said. "But they sure are well organized." What he wanted was service. What he got was a *system*.

Many people experience a similar frustration with menued voice mail. Having to choose, on the fly, whether you should "Press 3 for customer service" or "Press 4 for product support" is not always an easy choice. If your customer base is international, where callers may speak English as a second language, you may be creating serious problems for yourself.

You can have an excellent product. But if you don't present it with excellence, you may be seriously limiting its success. With good marketing, you can have spectacular results with an adequate product. But with poor marketing, even an outstanding product can be a dismal failure. Can you remember the last time you recommended a McDonald's hamburger to someone for any reason other than to fill their stomach? Yet McDonald's serves food to more than *nine million customers* each and every day. (Personally, I prefer Wendy's).

Perhaps you've got a lock on your niche market, and you have no competition to worry about. Maybe your customers are the exception, and they're perfectly content to communicate with your machine. Consider this: a recent survey showed 66% of the customers lost to competition are typically due to the feeling that the business they had been dealing with had become indifferent to them. *Sixty-six percent!*

When you design the systems necessary to operate your business, consider what they're like for your customer. Are you making them feel heard, or are you just recording their message? Are you giving them a measure of control, or are you trying to force them to jump through hoops?

If possible, have a personable human being answer your phone during business hours. It can improve communications tremendously. The real reason behind many "complaint" calls is a need for the person to vent their frustrations, and many times that's enough. Once they understand that you're genuinely interested in helping, you're well on your way to a solution - and the

next sale.

If you can't afford an employee to handle incoming calls, consider hiring a telephone answering service. Be very picky when choosing an answering service, though: there are a few good ones, a number that are adequate, and multitudes that don't deserve your time. Make several calls to prospective answering services, posing as a customer and requesting information. See how your call is treated on different days and at different times during the day. If at all possible, make cost your lowest priority when making your decision; a good service can make money for you. No matter who you decide on, make sure you call them periodically to see how they're handling your business.

Like many, you may decide to have your spouse answer the phone for you. This can work well in some situations (depending on the people involved), but there are a couple of serious pitfalls. You've got to remember that your phone line is the lifeline of your business, and you should make sure calls are handled in a businesslike manner. If you want your business to grow, you don't want your business phone calls treated the same way your spouse might handle a personal call from a friend. Even though the phone may be ringing in your basement, the caller should have no reason to believe they're not calling a suite in an office building. And whatever you do, don't let the caller hear your children in the background, or - heaven forbid! - let them answer the business phone!!

If you must resort to a message machine or voice mail system, periodically listen to your system and give it a merciless review. Forget that you may be listening to your own voice, and listen for two important things. First, listen to the content of your outgoing message. Remember, the customer couldn't care less about your product or services. What's utmost in their mind is that they have a problem, and they're looking for a solution. Does your message talk about your company and how wonderful and responsive it is? Or does it talk about the solutions you offer, how you are dedicated to working with customers, and the options they have to choose from?

If you're creating your messages "off the top of your head", it's likely you're not talking solutions. Sit down and write a script that makes customers feel they've called the right place. Then go back and listen to your original message again, this time listening for performance. Close your eyes and don't even listen to the words. Does it sound like it was recorded for Joe's Plumbing Service? Does it sound like Joe himself? Does it (honestly, now) sound like someone was reading a script? Your message should give the impression that your business is absolutely professional, that your personnel are competent, and that you care about your customers.

If your message doesn't convey that feeling, consider having someone else record the message for you. Maybe you have a friend who does community theater, or someone who is an excellent public speaker. Spin down the radio dial and check out the announcers on the local radio stations. Many are willing to record short announcements for a minimal cost ($25 or less).

Always remember, first impressions last. As you design or revamp the systems necessary to operate your business and make it grow, make certain you include those which give your customers reassurance they are in control and that their concerns are being heard.

The *personal* touch.

(By the way, Dave Thomas - founder of the Wendy's restaurant franchise - has written a really good book on marketing for any business. It's called Dave's Way, and you can get a copy of the paperback version at most Wendy's restaurants for only 95 cents, a true bargain - and the proceeds from the book go to the adoption cause. If you've gotten the impression from his commercials that Dave is some sort of bumbling fool, don't believe it. He's the same guy who turned a white-haired man using his Cadillac to deliver bags of spices to restaurants, into the multimillionaire head of the Colonel Sanders' Kentucky Fried Chicken empire.)

Have you sent in your Pro/Source data file yet?