DOWNLOAD PDF INSIDE THE TOOLBOX USING THINK PASCAL

Chapter 1 : MACINTOSH C PROGRAMMING PRIMER INSIDE THE TOOLBOX USING THINK C TM VOL

Note: Citations are based on reference standards. However, formatting rules can vary widely between applications and fields of interest or study. The specific requirements or preferences of your reviewing publisher, classroom teacher, institution or organization should be applied.

More than 85, lines-per-minute compile speed. Full access to all Windows functions and messages. Built-in assembler for speed and control. Math coprocessor and emulator support. Smart linker to remove unused objects and code. Public, private, and inherited keywords. Open array and string parameters. Faster string and file operations. A complete manual set should include the following: Check the file times of your installation. At the time it appeared March you got it from Borland Inprise if you were a registered user. On some processors they produce garbage -- on some processors they work. The implementors of 7. A solution to avoid this bug with 7. Bug of the bug fixes in 7. Some range and overflow checks with byte and shortint which were implemented incorrectly in both the RTL and TurboVision. GetDir did not return an invalid drive error error 15 when called with a nonexistent drive. There was some sort of problem converting denormals to true zeros when using an coprocessor. White label reads Item: Also includes three K Turbo Debugger 2. Two K Turbo Assembler 2. Produces programs for use with MS-Windows and or higher processors only.

Chapter 2 : Ingemar's guide to Think Pascal d4

WorldCat is the world's largest library catalog, helping you find library materials www.nxgvision.com more â€ºâ€º.

But most programmers were used to the extremely sparse world of the text terminal glass TTY , in which they had to learn only how to read and write text on the terminal, and read and write data to files. If you had a Commodore 64 or Apple II you might know something about bitmapped graphics, but learning to draw on the screen was basically about learning the hardware, not about operating a complicated set of software libraries. Some programming languages had libraries with subroutines for communicating with terminals. For example, in Pascal, there was writeln which would write a line to the terminal and readln which would read. Another thing that may seem odd is the popularity of the Pascal language. Subroutines saved their return addresses, local variables and parameters in statically allocated data structures called activation records, so FORTRAN IV code was non-reentrant, and recursive programming was not practical. Even though much good work was and continued to be done with the language, modern programming thought had turned away from the FORTRAN design. The new way of programming was embodied in the Algol and Algol specifications. I say specifications, because these languages were designed without too much concern for implementation. They were complex and powerful computer languages. It was pretty hard to write a compiler that could handle all the complexities of programs written in the full form of the language. It was also complicated for programmers. Niklaus Wirth was a computer scientist at Stanford University working on languages and compilers. After the Algol specification was released, he became interested in simpler, easier to learn and easier to implement Algol-like languages. He invented several such languages, including Pascal. In , Wirth produced a Pascal compiler written entirely in Pascal. Lisa software, including the operating system, were mostly written in Pascal, and the Lisa Programming Workshop was designed for Pascal programming. The Macintosh operating system was largely written in assembly language to optimize for memory space , but its functions were intended to be called from Pascal. BASIC was a popular language in the microcomputer world. But it was interpreted, not compiled. This sounds like a bad thing, because interpreters are slow. But it was a good thing, because debugging was greatly enhanced in an interpreter. But it was useless. Nobody wanted to program the Macintosh that way. About a year later, Microsoft produced a much better version 2 of Mac Basic. Things were hard enough when you used the language described in the documentation. When you started it, it gave you a text editing window called Untitled in which to write your program, a window called Drawing, and one called Text. It auto-formatted the Pascal code you wrote in the code window, and used immediate changes in typeface to indicate an error if you wrote something that was not syntactically correct. Nowadays this is done using colors, but in the black and white world of the early Macintosh, it was done with type styles bold, outline, etc. More subtle errors not caught this way could be found by picking Check from the Run menu, or by trying to execute your program. The thumbs-down hand pointed to the line with the error. Two other windows, one called Instant, and one called Observe, were available from the Windows menu. In the Instant window, you could write any valid Pascal expression that made sense in the current context of the program, and it would execute. In the Observe window, you could enter the name of variables and observe their values as they changed. These things are all common in advanced programming environments today, but were amazing and unexpected in Because in MacPascal, your program was running within the context of another program the interpreter , you did not have to create resources, initialize menus and windows, or use the Macintosh memory manager. Most of the complexities of the Macintosh Toolbox were removed, at least at first. You could learn one thing at a time. In particular, MacPascal was a testbed for learning Quickdraw graphics. Quickdraw procedures that draw on the screen, like LineTo and FrameRect, would immediately be directed to the Drawing window. Likewise, the standard terminal text functions in Pascal like writeln would immediately put text in the Text window. This meant that standard Pascal programs you might have found in a textbook on programming in Pascal would still work and do sensible things. And it did this by applying user interface tricks nobody had ever seen before. One bad thing about MacPascal: It was copy protected. You had to have the original floppy disk mounted. This was fixed in version 2. The Integrated Development

Environment - Lightspeed Pascal As happens with all good teachers, the stuff MacPascal was teaching became obvious after a little while. Everybody wanted to make native Mac programs that could run outside an interpreter. Lightspeed Pascal introduced the rest of the features that we now associate with modern Integrated Development Environments. A project file contained all the object code and preference information everything but the source code and finished application associated with the project. Opening the project file would initialize Lightspeed Pascal with the information for that project, ready for more coding and compiling. It was reasonable to expect that there would be a price to be paid for this. Without the interpreter, I would expect to still have syntax checking, but what about the Drawing and Text windows? Amazingly, Lightspeed Pascal could take a MacPascal source file, and after you created a project file for it compile and execute it exactly like MacPascal. The Text and Drawing windows were still there. In the compiled version of the program, those windows could still be used, so simple programs did not need to create their own windows. And, the Instant and Observe windows were still there, and still worked the same as always. Breakpoints were little stop signs in the left margin of the source code window. There was one new window, and it was an eye-opener. It was called LightsBug. This was an obvious reference to MacsBug, the object level debugger. It showed the values of all the registers, and could provide a dump of the contents of arbitrary regions of the heap. It lost nothing in the process, and most of what it gained could be seen in the LightsBug window. In addition to showing registers and the heap, it was an inspector for all local and global variables, could examine data structures, by their field names, and included a stack trace in the upper left corner. By this time , Niklaus Wirth had collaborated with Apple in the creation of an object-oriented Pascal variant, called Object Pascal. Apple developed a class library in this language, called MacApp, inspired by Smalltalk All things cool become uncool eventually, and once everybody knew Pascal well, it quit being the language we wanted to learn. Now we wanted to program in C. Pascal and C are very close relatives. They are both simplified Algol-like languages. They look different, but if you can write a Pascal compiler, you can write one for C. The first usable version of Windows was still 9 years away. Unix programmers were writing their source code in vi and compiling it on the command line. I guess a lot of them still do. The same way that the Macintosh invented the user interface for computer users, THINK Technologies invented the user interface for programmers. The project file, source level debugger, variable browser, these are the user-illusion that programmers expect today.

Chapter 3 : Download [PDF] Macintosh C Programming Primer Inside The Toolbox Using Think C â€“ Fodr

*Macintosh Pascal Programming Primer: Inside the Toolbox Using Think Pascal [Dave Mark, Cartwright Reed] on www.nxgvision.com *FREE* shipping on qualifying offers. This book shows programmers new to the Macintosh how to use the powerful Toolbox, resources, and the Macintosh interface to create applications with the distinctive Macintosh look and feel.*

Most of the software only has minor incompatibilities with System 7. You can probably get by with version 3. Hopefully, I can show some of you why you should upgrade to version 4. Upgrading to version 4. The old car, depending on the model, still provides adequate transportation it gets you there most if not all of the time. You can create an alias of the project file and place it anywhere. The original resource file must still reside in the same folder as the original project. My favorite is the Instant Project. Instant Project helps you anytime you create a new project from scratch. If your projects are small, you probably create new projects more often. In that case, you will especially like Instant Project if you create a lot of projects. A new folder is created with your project and a source file already installed in the project. The source file is already set up with program, begin and end. Adding several files to a project is easier too. Hold down the option key and look in the project menu. Any kind of project can have multiple segments. There is a new and improved version of the AppleEdit DA that is also bit clean. The uses clause is now supported in units used by other units. If your unit uses other units, any unit that uses your unit also uses those units automatically. Debugging has improved too. When you use LightsBug you can open a new LightsBug window while the first one is still open. Searching is faster and easier with multiple file searches. Compiling is faster now too up to 60, lines per minute on a Macintosh IIci. The profiler is easier to use under version 4. IM volumes I-III have been the base standard since the beginning with only minor modifications, although major additions have been made from time to time volumes IV-VI. Programmers that follow IM will find that their code will run now and still run in the future. Even after you are familiar with IM and the standard routines that you like to use, it will be only a matter of time before you will need to consult IM for more detailed information. IV on your left, vol. V on a chair behind you. If you can find room for vol. It should not be used as a Macintosh programming reference. JEP is limited to the subjects covered in its tutorial. Symantec must have realized that there was still a need for a good detailed reference to Macintosh programming. Reference covers IM volumes I-V with a few exceptions. Reference is much more than that. The Copy Template item in the Edit menu allows you to copy a template of a particular function or procedure. You can choose to copy the template in either C or Pascal format. This is very useful when you are using Reference with your compiler. Each page of the reference includes a description and usually an example of the use of each routine. The examples are very good about suggesting ways to make use of each routine. The big problem is that there is no way to copy any part of the examples or description. The page can be printed, but it would be more useful to be able to paste the examples directly in source code to try them out. The descriptions could also be used to help document your source code. Each page can be conveniently printed individually, so that you can study each one off-screen. It would be easier to print them if several pages could be printed at once, but you have to go to each page and then print the page. Besides being able to look up any Macintosh Toolbox routine by trap name or by manager, THINK Reference includes hypertext links to let you jump directly to related subjects. Hyperlinks appear as underlined words. Bold underlined hyperlinks usually refer to Toolbox routines. Plain underlined hyperlinks refer to data structures, variables or fields of records. The capability to jump to other Reference pages is what makes Reference most useful. Clicking on one of the buttons brings up a pop up menu. As you can see, it is easy to move around freely from page to page. For smaller configurations, it would have been desirable to have a smaller database of just the templates with the ability to copy templates. Programmers with larger configurations would like to see more included in the Reference such as descriptions of C and Pascal libraries such as the ANSI library in C or the Runtime. It would also be desirable to have the ability to update the Reference with their own notes. Overall, the deficiencies I mentioned are minor compared to the overall usefulness of the database. Maybe some improvements will be made in the next version. Reference database requires 3. For more information, contact:

Chapter 4 : CodeProject - CodeProject

Share your thoughts on Macintosh Pascal Programming Primer: Inside the Toolbox Using Think Pascal (Macintosh Pascal programming primer). Write a review This book is very easy to read, especially for a tech guide.

All the parameters to NewWindow are a bit confusing, but you will usually read windows from resources, which is easier. I have no intention of providing a complete manual here, but here are some notes on things that are not entirely intuitive: You must have a project file to compile anything. The project file describes what source and library files the compiler should use. Debug, lets you step, set breakpoints etc in the unit. Disable DVR for maximum speed, enable them for maximum safety. I believe N only affects the size. Debugging with range and overflow checking on can be a lot easier than without them! The "Run options" dialog is very important! There, you can select a resource file to use, and to copy into the application when building to disk, and there you set the memory partition and stack size. Note that the memory partition when building to disk is not set there, but by a "SIZE" resource. Having fun with the debugger Have you tried the debugger yet? It consists of at least four parts: The Lightsbug window may seem a bit cluttered. The most useful part is the variable list, but the other parts are worth exploring. You can click-and-drag variables to the icons down the left side to access some special features. The top three icons swich between variable view, register view and heap view. You will mostly use the variable view, although the other two are in no way useless! All are explained if you click on them! The bottom part of the window shows the memory as raw hex numbers. Lightsbug is just one of them. The innocent-looking Observe and Instant are extremely powerful and useful, and the editor windows, where you single-step and set breakpoints, is perhaps the most vital one. Note, however, that power should be used with a little care. Be careful with putting expressions in Observe that use pointers or handles that may become invalid, or expressions that allocate memory! Here are some specific non-obvious debugger features: You must check "stops in" to be able to set breakpoints. You set breakpoints by clicking "stop signs" in the left side of the editor window. When debugging, the little icon in the upper right corner can be clicked to stop execution at any time. Command-shift-period has the same effect. You can get more than one Lightsbug window! Observe will re-evaluate all expressions typed in it if you select Observe from the Debug menu. The same holds for Instant. There are a few pitfalls, problems that you can run into after a while. You sometimes get error messages about "segment too large". If you do, click the pyramid icon in the project window to switch to "segment view". There, you can drag files between segments. Generally speaking, try to keep you segments below 32k! You can debug Drag Manager receive handlers, however. However, if you can assemble a test harness so that your program sends Apple Events to itself, or your test harness can send events to your program from another machine, Think Pascal will pass any events that it receives on to the running program. There is one infamous problem that you should be aware of: This can leave a beginner rather stumped and frustrated. The likely problems are the following: Inside Think Pascal, the port is set to "Drawing" by default. When running stand-alone, that default is no longer valid, so you must make sure that you SetPort to some valid window. Think Pascal initializes the Toolbox automatically. If you do that yourself too, you may get problems. If you give your program plenty of memory in Run options Finally, there is one problem that all development system has. If you write outside arrays, or follow invalid pointers, you will get unpredictable errors, errors that will change whenever your program changes. Finally, if the compiler is crashing on you for no apparent reason: Are you using MacOS 8. Are you still using 4. If not, do it! Learning the Pascal language I will not teach you Pascal here. To learn the language, you can just buy a textbook. There is also some resources on the net. A pretty nice, although not Mac-specific, on-line course is available here. Note that there are some bugs in the links, with mixing "htm" and "htlm" extensions incorrectly, but you will figure it out. There is also a link to another Pascal course at Pascal Central. Apart from learning the Pascal language, Mac programming require some specific knowledge. But, there is a way, and it is free: That book, and its demos, deal with CodeWarrior, but the differences are not too big. Here are some places to get them: My source-code archive also by ftp. Pascal Central , truly worth the name with links to most net resources of interest for a Pascal programmer. In particular, check its source-code section. Peter

DOWNLOAD PDF INSIDE THE TOOLBOX USING THINK PASCAL

Lewis has made a large bunch of reusable modules available. Leo Possajennikov has made a number of small freeware games available with Think Pascal source-code. Want to play music in your TP application? Almost any Pascal source for the Mac is of interest for you. There is one little problem though: You will find demos for TP 4. You will need to make some changes to make it work. It will be easiest with TP 4. A very simple animation demo I wrote is MicroAnimationDemo. It demonstrates sprite animation, and is about as simple as it gets. The link above is to a version that works with TP 4. To see how I adapted it to TP 4. Some demos will work with no changes at all. The demo KbdLts required a few changes. Some changes in DeskBus. Painted Black is a program that required some changes to reflect that problem. Here are some modified demos:

Chapter 5 : Dave Mark | Open Library

Macintosh Pascal Programming Primer Inside the Toolbox with Think Pascal. Author: Dave Mark and Cartwright Reed Publisher: Addison-Wesley,

Chapter 6 : Macintosh C Programming Primer Inside The Toolbox Using Think C â€“ PDF Download Site

"The book wastes no time with peripheral issues but goes straight to the matter of writing real Mac applications and making the best use of the Mac Toolbox and resources." - MacWEEK " The Primer's clear, friendly style, combined with an exceptional layout, numerous illustrations, annotated examples, and highlighted tips, make this a lush.

Chapter 7 : Google Earth Toolbox - File Exchange - MATLAB Central

Move the contents of THINK Pascal d4 (my modified interfaces) into THINK Pascal d4 Folder:THINK Pascal d4 - that is, move my interfaces to the folder where the compiler is located. Trash the empty THINK Pascal d4 folder.

Chapter 8 : Mac GUI :: Mac Pascal

This book shows programmers new to the Macintosh how to use the powerful Toolbox, resources, and the Macintosh interface to create applications with the distinctive Macintosh look and feel, using examples in THINK PASCAL

Chapter 9 : Borland Pascal, Turbo Pascal Version Information

Books by Dave Mark, Learn C on the Macintosh, Learn C on the Macintosh, Macintosh programming primer, Ultimate Mac programming, Macintosh Pascal programming primer, Inside the Toolbox using THINK Pascal, MacworldÂ® Ultimate MacÂ® ProgrammingÂ¿, MacÂ® Programming FAQsÂ¿.