The 68XXX Machines

Vol. 1, Issue 4, May 1991 Price: $2.75

The 68xxx Machines is published and copyright (C) 1991 by Chatham House Company, RD#1 Box 375, Wyoming, DE 19934. Ph. (302) 492-8511. The editor is James H. DeStafeno. One year USA subscription is $12.50. Canada and Mexico $14.25. All others (surface) $17.25. All major credit cards accepted. Our low prices reflect a 10% cash discount. Please add 10% to credit card orders. Any size display advertising is accepted. The half page rate is $10/issue. Write for other size/duration rates.

Readers are encouraged to contribute letters, articles, programming information and other material related to computers with the 68xx(x) processors; excepting Macs and Amigas. Please send material to the above address. Thank you for your support.

The Editor's Thoughts

By Jim DeStafeno

Yes, we are late. The presses were held to give you a report on the 68XXX action at the Chicago '91 CoCofest. All three major 68K computers were there in action.

The System-IV booth had several machines demonstrating excellent color graphics, and digitalized voice and music. They even had their MS-DOS board running. They were proud to say they have been shipping machines for many months.

Frank Hogg Labs had three demo machines showing fine animated color graphics and text. Frank said he will be shipping before the middle of May.

The MM-1 booth had two demo machines running. At first they too showed graphics and text. However, the graphic colors were very dark and muddy; no better then what they showed at Atlanta. I didn't see the graphics displayed all day Saturday or Sunday.

Saturday afternoon Lonnie Falk of RAINBOW MAGAZINE moderated a question and answer forum between Ed Gresick of delmar's System-IV,

This Issue:

Frank Hogg, and Paul Ward of IMS's MM-1. The highlights were, Paul opened with an apology for having overstated expected shipping date(s) and the MM-1's capability. He didn't state a new date, but did say the MM-1 would only run MS-DOS programs after they are rewritten (ported).

Frank Hogg openly vented his anger toward Paul based on discrediting statements he had made previously. At one point Paul bowed his head to his cupped hands on the table as Frank expressed his anger.

Gresick gave short relaxed answers throughout the three hour session, and asked several pointed questions himself. His main point was that he has been selling operating production hardware and software for many months. The other two didn't question his statements.

Under the "What's New?" thought, "68xxx" has added a page, it has more advertisers and of course subscribers. Also, we are letting some software authors write their own software reviews. We'll see how that goes.

There just isn't room for letters this month; hope we can do it next.

Advertiser's Index

Text Formatting In C

By Bob van der Poel

One of my REAL interests in programming is the development of text editors and formatters. I guess it dates back to the time I got my first computer and wrote my first REAL program--a simple text editor in Basic. Both my abilities and computers have come a long way since that time, but many of the problems in formatting text remain the same.

An interesting topic is that of justifying lines of text. At first look it is pretty simple--just put in extra spaces until the line is long enough to fill the space between the left and right margins. A 'C' language function for this might look like this:

```
/* point to buffer containing text */
justify(buff,currlen,nlen) char *buff;
int currlen; /* current length of the string */
int nlen;
/* "needed" length */
{
   register int t;
   while(currlen<nlen){
      t=0;
      while(t++<currlen-1 && currlen<nlen){
         if(buff[t]==SPACE && buff[t+1]!=SPACE){
            memcpy(&buff[t+1],&buff[t],currlen-t+1);
            currlen++;
            t++;
         }
      }
   }}
```

However, this first effort fails for a number of reasons. First, there should be a comment somewhere stating that 'buff' must be large enough to accomodate the expanded line. Second, what happens if the line contains no spaces? Right--it will loop forever. A flag is needed which will terminate the loop in this situation. Third, the results of a series of lines justified by this routine will not look very nice--the padding is always started at the left side of the line. This last problem is the focus of this article.

I have seen a number of programs which solve the problem of off balance lines by having two justify routines: one which expands from left to right, the other from right to left. These routines are called alternately. If you've seen text justified in this manner you'll probably have noticed that

it, too, looks a bit unbalanced (although it is better that the results of the above function). Not only that, but two routines are a needless complication.

When writing VPRINT I spent a lot of time thinking about a good solution--and the one I finally came up with is both simple to code and generates nice results. My solution is to always do the expansion from left to right, but to start the first pass at a random position in the line. Who said that random number generators are only good for games and simulations?

So here is my routine. I've included a rnd() function which is adequate for this job, but I'm afraid it might not be quite as random as one would like. You should also be aware that memcpy() on my compiler (OS-9/68000) handles overlapping memory--other compilers may need to use memmove().

```
justify(buff,currlen,nlen) char buff; int currlen;
int nlen; {
   register int t,flag;

   flag=1;
   t=rnd(currlen-1);

   while(currlen<nlen){
      while(t++<currlen-1 && currlen<nlen){
         if(buff[t]==SPACE && buff[t+1]!=SPACE){
            memcpy(&buff[t+1],&buff[t],currlen-t+1);
            currlen++;
            t++;
            flag++; /* signal something done */
         }
      }
      if(!flag) break;  /* nothing inserted, exit */
      t=flag=0;         /* start next loop at start
of line */
   } }

#include <time.h>

rnd(x) int x;    /* max value o return */

{
   register long temp;

   static union{
      struct sgtbuf tbuf;
      long  lnum[2];
      int flag;
   }seed;

   /* seed the pattern */
   if(!seed.flag) getime(&seed.tbuf);
   seed.lnum[0]^=0x1234567; /* set next pattern */
   seed.lnum[1]^=0x89abcde;
   temp=seed.lnum[0]+seed.lnum[1]; /* get number */
   if(temp<0) temp=-temp; /* make sure >=0 */
   return (int)(temp % x); /* return the number */
```

Bob can be reached in care of The 68xxx Machines or directly at PO Box 355, Porthill, ID, 83853.

Classified Ads

- WANTED - SS-50 equipment. SWT CPU card, also Gimix PIO #28 (30 pin) FDC. Alen B. Gordon, MD / 160 NW 176 St / Miami, FL 33169 / (305) 653-8000.

- SALE - PT68K-2 complete system: 10MHz motherbd w/1M RAM, 20M hard drive, 720K floppy, amber monitor, SK*DOS, RBASIC, EDDI, all manuals, more! Used very little, $800 + shipping. Phil Krieg, 1122 1/2 Bayland, Houston TX 77009. 713/861-0367 eve/wkends.

Turn that old computer equipment into cash with a *68xxx* classified ad. Sale ads are $5 per 50 character line. Wanted ads are just $2.50 per 50 character line.

Review: PAT Text Editor JUST Text Formatter

By R. W. Anderson

Please note, "68xxx" is trying a new review method; letting the program author write the review. Ron gets to go first. I'd like to hear what you think of the idea. For now, when reading, just keep in mind who is doing the writing. ED

PAT is an acronym for Program And Text editor. The design goal was to make an Editor that would be a touch typist's dream. Rather than using the Function keys to get a particular action, PAT uses control keys, <CONT> or <ESC>, followed by a specific character. The command control keys were selected to make a sensible layout, not for their letter name. The idea is to be logical by position. (I freely admit the key layout that controls the cursor movement is not original. I first saw it used on an Apple Editor programmed by by Tom Crosley.)

The use of control keys allows a touch typist to do many functions required by a screen editor by "touch", rather than having to find something like SHIFT F11. I submit, it is far easier for a touch typist to learn to hold the control key with the left little finger and type any valid control key on the keyboard then it is to reach function keys.

One quickly learns to substitute the 4th finger for the little one in typing <CONT-A> or <CONT-Z> for example. PAT groups the control of the cursor motions at the left hand side of the keyboard. It is the position of the keys that is helpful in remembering their function. The center of the cursor motion controls is the D key. Move up one row to the E key and type <CONT-E>, and the cursor moves up one line (never sideways regardless of short or blank lines). Move down a row to the C key to move the cursor down one line, <CONT-C>. To move right, <CONT-F>. <CONT-S> moves the cursor to the left. Get the idea?

Now expanding on the theme, moving two keys to the left, the A key moves the cursor left one tab

position, or one word depending on the mode. Moving to the right two keys from the D key, <CONT-G> moves the cursor to the right by one tab position or one word. <CONT-E> moves the cursor up one line. Moving to the right of E, <CONT-R> moves the text down one screen. The cursor ends up one screen above its previous location in the text. <CONT-V> moves the cursor down one screen in the text. <CONT-T> moves the cursor to the top of the file. <CONT-B> moves the cursor to the bottom of the file.

After the basic cursor moves we have a few more mnemonic keys, <CONT-K> kills a line. <CONT-J> kills a single character, not mnemonic, but of lesser scope moving more toward the D key. Continuing in that direction <CONT-H> backspaces and kills a single character. <ESC-E> deletes text from the cursor to the End of the line. <ESC-J> joins the next line to the current one, and <ESC-S> Splits a line at the cursor, the portion to the right moving down a line.

The cursor can also be moved to, say the 30th column of the 10th line after the last line with text on it and type an X. , PAT puts a CR at the beginning of all the blank lines down to the point and spaces over the 29 spaces, then puts in a X followed by a CR. You don't have to insert the blank lines with successive CRs and you don't have to space out to the 30th column. Some of this was the hardest part of PAT to code, but I felt it to be so far superior to the way many other editors work, it was worth the effort. It is so much more natural to move the cursor to wherever you want to put text, and just put it there. It is as though the screen were a blank piece of paper in a typewriter; you can put text anywhere on the paper.

All of these characters are assigned to their action by a programable ASCII configuration file. It allows you to swap the functions of the keys around to suit your tastes, though all the keys are used so you end up swapping functions rather than changing one or two to something else. There is a terminal configuration file too.

The goal was to make the other things logical too. Lines longer then the width of the screen, up

to 127 characters, scroll across the screen left to right. There is a solid reverse video status line at the bottom of the screen that shows memory left in the buffer; the mode, INS for Insert Mode; Pmode, Amode or neither; the Line Length; Cursor Position, line and column; first and last lines of a marked block (while it is marked), and a number of other mode flags. It is also the place to enter parameters such as line lengths, tab spacing, or search strings.

There are Help Screens acessed via control keys. If you forget a command, no need to dig out the manual, just type <ESC-H-CR> and choose from a menu of help screens or a return to the edit process.

There are three main modes; Overlay (Over strike) Mode, Insert Mode and AutoIndent Mode, and a couple of sub-modes. PAT's modes allows you to accomplish different general actions. In the Overlay (Over strike) Mode changes whatever is under the cursor to whatever is typed. In the Insert Mode whatever you type is inserted at the cursor and whatever that is after the cursor moves along to the right. In AutoIndent Mode, you can do programs or outlines. A CR puts the cursor under the first non blank character of the previous line. You can tab in or out of one indent level and continue with your outline or program. You can set tabs every Nth column with a simple command or you can set tabs irregularly at any specific columns you like from no columns to every column.

In either overlay or insert mode, you can turn on Paragraph Mode or Pmode as it is called. When you get to the preset line length, the last word or partial word is moved to the next line (a feature called word wrap). You never have to type a CR while entering text except to end a paragraph. In Paragraph Mode you can move the cursor left or right respectively to the first character in the previous or next word. When Paragraph Mode is off, the tabs move the cursor to the next or previous tab position. This is true whether or not AutoIndent Mode is on.

CR is rather benign. It doesn't split a line, but moves the cursor harmlessly to the beginning of the next line. This is true whether you are in the middle of a paragraph at the time or not. PAT presently has the edit buffer set to be 200,000 bytes, 800 sectors. PAT will also edit larger files by spooling the early part to a disk file and pulling in more to the buffer limit. Spooling is not dynamic. That is, once you have pulled in more of a file you can't go back to the early part that has already been written to a disk. To do that, you have to save the file, exit, and start editing again.

PAT has global search and replace commands. They allow you to change one occurrence, selected occurrences, or all occurrences of a particular string to another. There are also block marking features. A marked block appears in reverse video on a monochrome monitor, while a CGA momitor shows the block on a red background. Normal text is on a blue background. A marked block can be copied elsewhere in the text, moved, or deleted.

For systems with an IBM keyboard, I've activated the cursor keypad to work in parallel with the control keys for cursor motions. (I find them convenient when I am looking through a file and not actively entering text.) They are single keystrokes that take the place of control keys. The arrow keys do just what you would expect. PgUp does the same thing as <CONT-R>. PgDn does the same as <CONT-V>. Home and End move the cursor to the left and right end of the current line respectively. INS toggles insert/overlay modes, <CONT-P>. DEL deletes one character at the cursor, <CONT-J>.

For systems with an old serial terminals, I've made another modification. On most serial terminals the ESC key is immediately to the left of the Q key. On the IBM keyboards the TAB key occupies that space and the ESC is off on the upper left of the numeric keypad. I've reassigned the TAB key to be ESC since tab is activated with <CONT-G>. I find it much handier, but on request at time of ordering I'll leave TAB as TAB. IBM interprets the TAB key to send <CONT-I>. In PAT that will insert a blank line at the cursor. In either case the normal ESC key continues to function as ESC.

When running under SK*DOS the backspace key and <CONT-H> activates the backspace function; move the cursor left and delete any character under it. Backspace will go past the beginning of a line and wrap around to the end of the previous line. Unfortunately, in the REX version, some other considerations have made the actual backspace key a cursor left only key rather than a real backspace. You must use <CONT-H> with REX to move and delete.

PAT can reformat ragged paragraphs, putting as many words as will fit on each line, as determined by the line length that you have programmed.

Also, by your command PAT can insert additional spaces to right justify the text. After placing titles, etc. in the proper position, printing can be accomplished by simply LISTing the file to a printer. It will come out spaced as it is on the screen, including justified. However, to cause special location page breaks and to produce page numbers, amoung other things, the eompanion text formatter called JUST must be used.

PAT produces standard ASCII files with no control characters other than CR, and no character with the high order bit ON. Each line is terminated by a CR, rather then a paragraph as just one super long line. That means a standard LIST utility will list a PAT edited file with normal line breakes.

However, normally I use my JUST text formatter. It has formatting control characters which are embeded in the text file when it is ready to print. Using the control characters the formatter signals a printer to change type widths, switch to italics, etc. To use it, you must prepare a printer control file that contains the proper escape sequences to control your specific printer. You simply insert a back slash and a predetermined character to get the action you want. Such as, \i in your text turns the italic font ON, and another \i turns it OFF; \n is for narrow font, \w for wide and \b is for boldface.

Formatting commands begin with a comma at the beginning of a line, also followed by a character; such as, ,C for center the next line and ,p to signal a paragraph start with a program set-

table number of indent spaces. JUST doesn't care whether commands are upper or lower case.

PAT versions are available for 6809, MS-DOS and of course 68000 machines; the latter under SK*DOS and REX. The 6809 version only has room for about 29K of text buffer. The others all have up to 200,000 byte buffers. The source file for PAT is about 75K of text, so (except in the 6809 version for which the source is split into multiple files) it can edit its own source file with plenty of room to spare.

In addition to the features mentioned above, there are also bookmarks (a way to mark a place in a file so you can return to it later), block moves, copies, deletes, writing blocks to another file or reading in whole files into the text at the cursor position. Search and Replace features include being able to search for a word ignoring upper/lower case. You can delete a character, a word, a line, a marked block or a specified number of lines. There are express cursor moves beginning - end of line and top - bottom of screen, just to name a few more.

It would be unfair if I didn't mention that PAT does not select fonts and format two or three columns across a page. Nor can it edit two files at the same time. I am presently working on that. The action can be done now but it is a cumberesome. Suppose you want to copy part of file A into file B. First you edit file A and write the part you want to transfer into a sidefile C. Then you exit and edit file B. Now you put the cursor where you want to insert, and input file C. When you are done you might want to delete file C.

PAT is available on disk with complete manual (about 45 pages) in text files on the disk, ready to print out by simply listing the files on an 80 column printer. I'll supply a printed manual for $15 more. Please specify REXDOS or SK*DOS version and the disk formats you can read. I'll throw in JUST which works with "monospaced" type. It has the same sort of configuration file as PAT. You tell it what the control codes are for your printer and it then sends them when it sees commands in the text file.

I am presently not supporting the OS-9 version, of PAT or JUST, but I might be interested in reviving them if there are enough requests. The MS-DOS version presently runs only on color monitors, CGA, EGA or VGA. I would add Monochrome capabilities if there is interest in it as well. If you are interested in the above, write me a letter. I'll try to be flexible.

I supply the terminal configuration file set to video terminals, and include a number of sample ones for various common serial terminals. Of course I'll help you if you have a problem configuring for a particular terminal. It is an ASCII file that can be edited with any editor, which needs not be assembled or compiled. It can handle just about any serial terminal including the ANSI type that requires rather different and longer control codes than other types, (but are more flexible).

- PAT and JUST (including source code) R.W.Anderson, 3540 Sturbridge Ct, Ann Arbor MI 48105, $50 with the manual on the program disk, $15 for printed manual -

Good ideas, sudden realizations, and other such "Ah-Hah!" moments are not always especially original. There are many observations that are "known" by the general populace - at least internally - but they do not allocate much conscious time thinking about them. Consequently, others get the credit for "discovering" these truths. I'm one of those people that just can't resist being out there looking for these subtle truths that are in the "ethers" rapping on the windows of our consciousness.

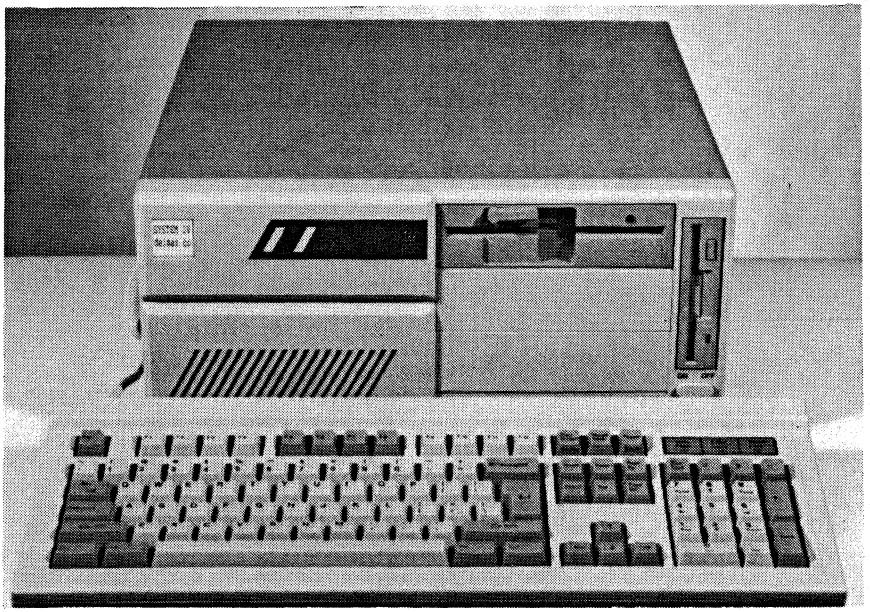
My topic for this month's ramblings falls into such a category mentioned above. My wife and I have agreed that both separately and together, we have discovered the root causes - and therefore, the solution - to over 90% of the crime in America. That's right!

A couple of the major buzzwords making their way through many disciplines today are "preventative" and "proactive". Preventative Medicine teaches doctors to treat patients such that problems can be identified before they get out of hand. Preventative Dentistry is a practice that all but eliminates the need for traumatic visits to the dentist. Be preventative, be proactive. Anticipate problems and act in such a manner that will allow you to escape or minimize the consequences. So we've decided that Law Enforcement agencies should enact laws and work toward prevention of crime before it can happen.

You've heard it all before. After some heinous crime, most people's first reaction is "Now who could have done a terrible thing like that?" But after years of reading and watching the daily news, the answer, it seems, is simple. Over 90% of all heinous violent crime can be attributed to persons falling into one or more of three categories. #1 They keep a diary; #2 they have a distinctive tattoo burned somewhere into their body; or #3 they are the sole beneficiary of the decedent's monster life insurance policy.

Diaries, tattoos, and life insurance should be outlawed! If

SYSTEM IV



SYSTEM IV - The 68000 Computer already serving customers here and abroad!

- Expandable using readily available low-cost cards
- Flexible enough to tailor to your requirements
- Versatile enough to use OS-9/68000, OS-9/6809, MS-DOS and other Operating Systems
- Powerful - outperforms other machines in its' price class

THE BEST OF ALL WORLDS

1 - A Multi-User Multi-Tasking System
2 - Optional Plug-in Board for MS-DOS
3 - Optional Emulator/Interpreter for OS-9/68000 Software
4 - Will support Other Operating Systems

Just call, FAX, or drop us a line. We will respond promptly with complete pricing and specifications.

DELMAR CO - serving its' customers since 1975.

Terminal Systems . . . . from $999.00 Console Systems . . . . from $1,149.00

Assembled boards and kits available

delmar co

Middletown Shopping Center - PO Box 78 - Middletown, DE 19709

302-378-2555 FAX 302-278-2556

OS9/68000 SOFTWARE

M6809 - 6809 Emulator/Interpretor $ 99.95

M6809 emulates the user mode OS9 Level II operating system and interprets OS9-6809 binary object modules under OS9-68000.

QUICK ED - Screen Editor and Text Formatter $275.00

A high quality documentation tool and program editor ideally suited to laser printer users. Uses function and cursor keys on any terminal, configurable per user. Micro-justifies mixed proportional text. Automatic table of contents generation and user-definable macros and commands. Handles an unlimited number of fonts. Drives any printer. Ideal for multi-user systems. Available on a 30-day try before-you-buy basis.

FLEXELINT V4.00 - The C Source Code Checker $495.00

Flexelint finds quirks, idiosyncracies, glitches and bugs in C programs. 60 options control checking by symbol name or error number. Checks included intermodule inconsistencies, definition and usage of variables, structures, unions and arrays, indentation, case fall-through, type conversion, printf and scanf format string inconsistencies, and suspicious semi-colons. A must for all serious C programmers.

DISASM_OS9 - OS-9/68K Disassembler $250.00

This high-speed, three-pass 68000 disassembler can also handle the 68010 and 68020. It intelligently decodes module headers and produces symbol information that can be repeatedly edited and passed through the disassembler allowing iterative disassembly. The system libraries are read to supply symbols.

IMP - Intelligent Make Program $250.00

IMP does everything you wished Microware's Make would do, and a great deal more. It is well-behaved, consistent, and extremely flexible. It has a built-in C-like preprocessor and has comprehensive debugging facilities. Rules can be user-defined, and make files for jobs other than assembly-language or C compilation are easily constructed.

WINDOWS - C Source Code Windowing Library $250.00

This C source code library package supports multiple overlapping windows displayed on one character-based terminal screen. It supports window headers and footers, and pop-up windows. Windows may be moved, panned, written to while off-screen, etc.

PROFILE - User State Program Profiler $270.00

Designed to profile user-state programs. Profile effectively samples a traced execution building statistical information as it goes. It reads symbol table modules to give a function-by-function account of the time spent during execution. The user may "zoom-in" on a function to find a smaller range of addresses where time is being spent.

PC9 - MS-DOS to OS-9 Windowing System $350.00

PC9 allows an MS-DOS computer to be used as a terminal to multiple processes on a remote OS-9 system linked by a single serial cable. Each OS-9 process is displayed through a resizable, moveable window on the PC screen. Terminal emulation facilities support uMACS and other screen editors and provide a programmable PC keyboard. Access to PC disk drives is also avilable through the OS-9 unified I/O system, giving disk capability to ROM based OS-9 systems.

delmar co

Middletown Shopping Center - PO Box 78 - Middletown, DE 19709

302-378-2555 FAX 302-378-2556

that sounds too harsh, think about it for a while. Diary keepers (a la Oswald, Sirhan, Hinkley, etc.) are responsible for nearly all political assassinations in our recent history. Monster Whole Life, Double Indemnity life insurance is the smoking gun behind God knows how many murders. And tattoos! .. Well, you know it doesn't take an Einstein to realize that for three years nearly every criminal profiled on America's Most Wanted has sported an odd tattoo of one shape or another.

So it's time to be proactive! Prevent crime before it happens! Outlaw these killer practices and monitor those that indulge in them. You will go a long way toward cleaning up crimes of all sorts ravaging the country today.

Review: Whimsical

By R. W. Anderson

I've been using Whimsical both at work and at home for several years. The manual (which is large and quite complete) contains a history of the language. Therefore I won't go into that here, save to say that it was developed in New Zealand by John Spray while he was attending the University of Auckland. Now JOhn and I work together, for the same company, here in Michigan.

Whimsical is a fairly complete Pascal like language that compiles to machine language. It is available in SK*DOS or REXDOS versions. With the Pascal like syntax you can do most anything you can do in assembler. Also, if you want to use some assembler code in the middle of a procedure or perhaps write a whole procedure in assembler, it is easy to do so.

Whimsical is "tuned" to the architecture of the 68000. It takes full advantage of all the

registers. In any given procedure you can specify register variables. If there are any spare registers the compiler will generate code to use them. If not, it will tell you they are not available. The compiler "knows" at the time of compiling the register allocation and can tell you whether or not you can use a register variable. Unlike the process using 'C', you know whether you are really getting register variables or not.

Another nice thing, compiling is done in a single pass. Any unresolved forward references are retained in a buffer, and are "fixed". That is, the code generator generates a branch or jump that was given an address of 0 (as many bytes as required for the particular jump instruction). When the compiler knows the destination address it goes back and overlays the 0 with the actual destination address. If the jump code has already been written to disk, the compiler generates a load record that overlays the jump when the program is loaded from the disk. Because of the single pass nature, large programs can be compiled very quickly.

Whimsical compiles several times faster than 'C'. 'C' compiled one test I made in 121 seconds, while Whimsical took only 7; and that was on my old slow PT68K-1A. In addition, the 'C' version took 25 disk sectors, 6063 bytes of object code. The Whimsical version took only 3 sectors, 701 bytes.

Though Whimsical is mostly Pascal like, it It is less wordy. It also has some of the niceties of 'C'. It allows the use of curly braces for BEGIN and END as 'C' does. I emphasize the word ALLOW. You can use BEGIN and END if you prefer (I do). It allows the fast shortcut assignment operators of 'C', i.e. +=, -=, *= and /= for changing the value of a variable as in INDEX +=3; (rather than the usual INDEX := INDEX+3;). Statements are free form, terminated by a semicolon as in Pascal. A line break (CR) has no more significance than a space, in a Whimsical statement.

The language has a very useful pointer data type, more like a 'C' pointer than the one in Pascal. The syntax is much less confusing

than that of 'C':

```
char pointer p;

p := address port;
[p] := 'c';
```

(The syntax is close to Motorola's for the 6809 indirect addressing mode in assembler.) The pointer is defined by using the word POINTER in the definition. The variable name used by itself on the left side of an assignment statement must be assigned an address. The keyword address insures this. When the variable name is enclosed in square brackets it becomes the pointer. The statement says "set the variable pointed at by p to the ASCII code for the character 'c'". In the above example PORT might have been assigned a location when it was declared as in:

```
char port($FF07FF);
```

Whimsical allows the user to specify the address of a variable when it is declared. The parentheses in the above assignment holds the Hexadecimal address. A pointer in Whimsical can have an offset. It can be used to access a group of variables (perhaps declared as an array and perhaps not).

In the above example, address port+1 might be a control register for a parallel port. We can access that with our pointer p by using something like:

```
[p,1] := strobe;
```

where strobe might be a declared constant that turns on a handshake for an external device etc. The ,1 is an index operator that takes into account the size in bytes of the variable accessed by the pointer. It means "the next data item". In the case of a pointer to longint, for example, it will access the variable four bytes past the one pointed at by p. Data can be accessed sequentially by using a post increment or pre decrement operator on the pointer:

```
ch := [p+]; or ch := [-p];
```

Successive execution of this statement in a loop will move the pointer through the data forward or backwards respectively. A pointer can also be incremented using the ",=" operator:

```
p ,=3;
```

That would increment the pointer by three items of the variable type for which P was declared a pointer. In the case of longint, again p would be incremented by 3*4 or 12 bytes. (I didn't mean to give the impression here that pointers can't have meaningful names. p could just as well be data pointer or pointer to the value in question if you like. In fact all variable names in Whimsical can be up to 80 characters long. Only the first 32 are significant so if you use very long names you must be careful to put the distinguishing difference within the first 32 characters.

Whimsical is very particular about type mismatches in arithmetic expressions. For example you can't multiply an integer by a shortint and assign the result to an integer. If you try to do that you will get a "TYPE MISMATCH" error. You must convert the shortint to an integer by means of the EXTEND operation:

```
INT_VAL := INT_NUM * EXTEND SHORTINT_NUM;
```

Whimsical has a wealth of variable types:

Single Byte	Two Bytes	Four Bytes
byte	word	long
shortint	integer	longint
char		
real		
boolean		

Byte, word, and long are unsigned types that use hexadecimal values. The various length integers are signed and require decimal values. Char, of course expects a character value and real a floating point decimal. Types of the same length (byte, shortint, and char for example) may be converted using the CHAR conversion operator, just like the type cast operation in 'C'. If, for example, you try to assign a hexadecimal value such as $0d (or the value of a byte variable) to a

char variable, Whimsical will issue a "Type Mismatch" error. Type casts work between variable types of the same size. Data types are converted to the next larger type using "extend" and to the next smaller using "trim". You must be careful not to trim a variable containing significant bits in the upper half or the program will produce incorrect results. Conversions are made between longint and real using "float" to convert to real and "round" or "trunc" to convert from real to longint.

```
REALVAR := FLOAT EXTEND EXTEND
SHORTINT $3F;
```

Realvar contains the value 63.0. Of course it would be silly not to assign that value directly, but suppose rather than $3F we had wanted to convert the value of a byte variable to a real.

Real numbers are four byte IEEE standard floating point numbers using an 8 bit exponent and 24 bit signed mantissa. The result is seven digit precision for floating point. Suppose we want to get the exponent portion of a real variable into a byte variable. The number format is SEEEEEEEEEMMMMMMMMMMMMMMMMMMMM where S is the Sign bit, E is the exponent byte and M is the "mantissa" of the number. Whimsical has a way to access the individual words or bytes of one of the larger data types. Looking at the above, we have:

```
   BYTE[3]     BYTE[2]     BYTE[1]
BYTE[0]

   SEEEEEEE EMMMMMMM MMMMMMMM MMM-
MMMMM

   ^---WORD[1]-----^^---WORD[0]---
---^   Now we can get the exponent
by
the following:

   EXP := BYTE[1] (WORD[1] REALVAR
<< 1);
```

That is, the exponent is obtained by shifting the high order word of the real number to the left one place and taking the upper byte of the result. Not only can you access individual bytes of a variable, but you can manipulate them by means of the bitwise and "&", or "", not "", or exclusive

---

or "^" operators. For example, you could remove the high order bit from a character variable (usually the parity bit) by using:

```
ch := char (byte ch & $7f);
```

The assignment operators also work with bitwise functions so that could be shortened to:

```
ch &= char $7f;
```

Another of the real strengths of Whimsical is its ability to use precompiled modules. You may split out a group of procedures and make them a module that can be precompiled. I did that with my scientific functions package. The module is easily included in a program:

```
module scipack=code from "1.w/scipack"
```

The module construction contains a header section that names all the procedures (and variables) that are "public", i.e. those accessible from the program or other modules. The rest of the module is private. That is, procedures and variables not declared in the public section are local to the module. A precompiled module is automatically given the extension .PCM by Whimsical. A complete program is given the extension .COM. Whimsical also works with uncompiled "include" files. You can split a large program into a number of files, and the main one can include others, which are pulled in as needed; just as though they were part of the main file.

Whimsical can work in conjunction with PAT, an editor. (See PAT review on other pages in this issue. ED) WHIM FILENAME +A invokes the connection. If Whimsical detects an error, it calls PAT and passes it the file line number and column number for the error pointer, and an error message. PAT loads and goes to the offending line. The cursor ends up where Whimsical detected the error, and the status line of PAT contains the error message. Of course precompiled modules are not accessible to this error mechanism, but "include" files are.

Whimsical will happily instruct PAT to edit an "include" file. When you fix the error and exit PAT the compiler tries to compile

the code again. On a successful compile, the process ends and a .COM output file is saved to disk. If you find you have gotten in too deeply and want to quit the process before you get through all of the errors, you can abort the process at the present state of errors fixed by quitting the editor. Fixes up to that time have been saved.

Whimsical allows two kinds of comments. All text after a double dash on a line (up to the end of the line) is considered to be a comment (as in Ada). Text enclosed in the delimiters /* and */ is considered to be a comment as well (as in 'C'). A comments can extend over several lines.

The Ada type comment is nice for adding a comment after a statement or for "commenting out" a single statement. The 'C' type is great for section headings and the like, or for commenting out a large number of lines. Comments of both types may be nested, so a large portion of a procedure or program can be commented out even if it contains comment lines.

Whimsical ignores capitalization within the program. That is, variables named PartNumber, partnumber or PARTNUMBER are the same to Whimsical. Some programmers like to use capitals to separate mulitple words in a variable name as in the first example above. Some (myself included) prefer underlines to separate the words.

Procedures that return a value must be typed. That is a procedure that returns a character must be defined as:

```
char procedure whatever; begin
   dosomething;
   return charvariable; end;
```

Procedures that return a value must have an explicit return statement. Execution of a return statement ends the procedure. You can make use of this fact to simplify logic in a procedure:

```
boolean procedure vowel(char ch);
begin
   if ch='a' or ch='e' or ch='i'
or ch='o' or ch='u'
      then return true;
   return false; end;
```

In this case if a vowel is found the "then return true" is executed and the next line "return

false;" is not. If the condition is not met, the THEN is not executed so the return false line IS.

Procedures that return nothing, have no type. They are simply started by the word "procedure". Procedures have parameters passed by value unless the keyword "ref" is used, in which case a pointer is passed. (Usage is identical to the VAR keyword in Pascal). Arrays are passed by means of a pointer to the first element in the array. The syntax is just like that of Pascal. Variable declarations, however follow the syntax of 'C', with the type coming before the variable name:

```
char array stuff[19];    -- an array of dimension 19

byte array ($12000)[1000];  -- an array at address $12000

char pointer where_zit;      -- a pointer

char CR = char $0d; -- a CONSTANT declaration (no ":")

longint size := 100000; -- an initialized variable

integer a,
        b,
        another;
```

The modifiers "array" or "pointer" follow the type and precede the variable name. Array dimensions (and array references) use square brackets. Presently only singly dimensioned arrays are allowed. Variables declared at a fixed address use parentheses for the address. Arrays at fixed addresses are declared as shown. Arrays may be indexed with a longint variable, and so may be as large as memory permits. Parameters are passed enclosed in parentheses.

You also would want to be told about the write statement too. It is very flexible. The value of variables of all types, but boolean, can be output by using:

```
integer k=17;   -- a constant declaration

write "the value of k is: ",k,"^M^J";
```

That might be a bit of a puzzle. The literal string is printed, then the value of K. Since K is an integer, a decimal value is printed. The last item tells Whimsical to output control M and control J (the carriage return and linefeed sequence). Another way would be to write the hexadecimal values :

```
write "the value of k is: ",k,char $0d,char $0a;
```

As indicated previously, the hex values for linefeed and CR must be "cast" to type CHAR. (We want to output the code as though it were a character. We don't want to print "$0d".)

The write statement is very valuable in debugging a program because it will write the value of a real in "sensible format" without a format specification.

It will write values of byte, word, long variables as hexadecimal values. About the only thing it won't do directly is write the value of a boolean variable "true" or "false". However, in such a case, all you do is:

```
if boolvar then write "true" else write "false";
```

The companion to the write statement is the read statement. Read may be used to input values from the terminal as in:

```
shortint number;

   write "Input a number from 1 to 100: ";
   read number;
```

Whimsical has easy file handling as well. A file is defined as:

```
file of char infile, outfile;
```

The file is associated with a directory filename easily. It may be done by means of a string, or if the filename is a parameter on the command line as in:

```
dosomething    datafile.txt newdat.txt
```

Now the simple statements:

```
assign infile;
assign outfile;
```

will associate infile with datafile.txt and outfile with newdat.txt. You can use a literal string or a character array to assign the file too:

```
    assign infile to "1.datafile.txt";
    assign infile to name_of_in_file; -- a char array
```

The files need to be opened. Assuming infile exists and outfile is to be created and written to:

```
open infile;
create outfile;
```

Now we can read from the input file and write to the output file using:

```
read from infile variablename;
write to outfile variable;
```

Read and Write can handle multiple variables to and from files. If you use file of char, all variables are expected to be in ASCII form and outputs are converted to ASCII. This procedure greatly simplifies looking at data files to see what might be wrong when debugging. If you are to write an output file containing only, say, real numbers, the file will be much smaller if you declare it as a file of real. You will then have to write a program to read the data and print it for you.

Some of the available branch control structures are:

```
while condition do
begin
   statements;
end;

do
   statements;
until condition;
```

Condition, in both cases is a boolean expression or variable. Boolean expressions use the keywords "and", "or", "xor", "not". They may be parenthesized as necessary for correct boolean algebraic evaluation.

The for/next style of counted loop is not yet implemented but may be easily constructed using an index variable:

```
integer n;
n:=0;
while n < 12 do begin
   statements;
   n+=1;
end;
```

The variable need not be a integer, and it need not be changed by 1. The value of the index variable may be used within the loop. The loop terminates when the while condition fails, in this case when N is reached. Such a loop would execute 12 times, with N having the values 0 through 11.

xxxCase statements are allowed on the value of any byte sized variable or expression. Whimsical also allows an ELSE clause. Unlike 'C', a break statement is not required after each case. The syntax is like that of Pascal.

A DO or a While loop can be exited conditionally by using an "if condition then exit" statement. Such an exit is to the line immediately following the loop structure in which the exit is found. That is, you can only exit one nesting level per exit statement. The statement after the current loop, to which exit brings execution, can be another "if condition then exit" statement, etc. The exit statement is very useful at times.

So as you can see Whimsical is a many feartured language that is efficient and therefore fast. I use it most everyday, and highly recommend it.

[John Spray, 304 Hartman Ln., Saline MI 48176, $75. Personal checks will be accepted, but not credit cards. Please specify SK*DOS or REXDOS version and disk format desired. If you have both 5.25" disks and 3.5" please specify 5.25. There might be a delay before 3.5" can be supplied.]

"The 68xxx Machines"
The Chatham House Company
RD#1 Box 375
Wyoming, DE 19934 USA

- Address Correction Requested -

WILMINGTON, DE
PM
13 MAY
1991

U.S. POSTAL SERVICE
OFFICIAL SPONSOR
OLYMPIC GAMES

- First Class -