CPM Archives

This page contains .ZIP archives of much of my CP/M work done in Draco. Some archives will have a short description and a contents list; others will just have a description. Unfortunately, I have been unable to find the following:

- - source for the "CRT" or forms libraries
- - source for the "config" program
- - Draco source for the database

The CRT-Form.zip archive on my Amiga archive page contains source for the form library that is probably very similar to the CP/M one. The source there for the CRT library will have some similarities, but it hardcodes Amiga escape sequences instead of using configurable values.

Many of the programs I wrote use my "CRT" library, which allows them to do character "graphics" on terminals. This library plays the same role as the "termcap" and "termlib" libraries on Unix and Linux systems. Those programs will be named "XXX.SET" instead of the usual "XXX.COM" used for CP/M programs. They must be processed by the "CONFIG" program on the "Tools.zip" disk, which involves selecting a terminal type to use. That program also needs the "CONFIG.DAT" file which contains a few dozen terminal definitions. If you are running these programs using a CP/M emulator (e.g. "yaze") in an "xterm", either directly on a Unix/Linux system or within the Cygwin environment on Windows, then select the "VT100" terminal. "CONFIG" will read the ".SET" file, find and update the terminal definition information, and write out an "XXX.COM" file which can then be executed.

It is worth describing what the CP/M Draco compiler included:

- - runs in the 64K address space provided (the CP/M "operating system" needed at least 4K and usually 8K)
- - runs in one pass - reads unprocessed source files and writes optimized relocatable object files
- - written in itself, and does not use overlays or paging
- - supports 8, 16 and 32 bit values, plus software floating point
- - supports user-defined operators
- - supports array and struct constants
- - allows run-time retrieval of parameter array sizes
- - allows in-line machine code using the "code" construct
- - has local, file-level and program-global variables/constants
- - supports "new", "free", "pretend", CPU-specific constructs
- - flexible text/binary I/O constructs and file manipulation
- - libraries include terminal-configurable "CRT" library, input forms library
- - supports conditional compilation and constant folding
- - supports compile-time evaluation of some/all of array indexing
- - includes a peep-hole optimizer
- - tracks register values to avoid loads
- - generates near-optimal branch networks for conditions, loops, etc.
- - uses either direct index or binary search for "case" constructs

Distribution Disks

These 3 disks represent a snapshot of Draco, in the form of a 3-disk set providing the Draco compiler and tools, plus some of my games, etc.

The Doc_Src disk contains the documentation for the Draco system and the tools and games provided. File list:

```
Documentation files:

    DRACO.REF - first section of language and utility description
    DRIO.REF - description of Draco's I/O facilities
    OPTYPE.REF - description of Draco's "operator type" facility
    DRUTIL.REF - description of the utility library
    DRCERR.REF - description of all compiler error messages
    DAS.REF - description of the simple Draco assembler
    CRT.REF - description of the CRT routines in CRT.LIB
    DED.DOC - description of DED (Draco EDitor)
    DEDMAP.TXT - template to print to show key bindings in DED
    QUEST.DOC - description of the workings of the Quest game system
    CONFIG.TXT - writeup on the CONFIG CRT configuration program
    ROBOTS.TXT - writeup on the ROBOTS game
    NUCLEAR.TXT - writeup on the NUCLEAR game
    LIFE.TXT - writeup on the LIFE game
    WRUNG.TXT - writeup on the WRUNG game
    SWITCH.TXT - writeup on the SWITCH system
    FUTURE.TXT - notes on possible changes/extensions to Draco

Source files:

    XREF.DRC - the Draco cross referencer
    HEDIT.DRC - hexadecimal file editor
    - several of the traditional benchmark programs
    - several small programs illustrating various points of the language
    - the complete source to the Quest adventure system - just add an
        adventure
```

Two source files provide additional examples. The first, hardware.drc, shows using Draco to do low-level hardware operations, including setting up a periodic interrupt. Note that a "vector" proc in CP/M Draco ends in a return-from-interrupt instruction instead of a return-from-subroutine instruction. Such procs also save/restore all registers on entry/exit. The second example, tree.drc, shows using some of the "higher-level" language constructs - new, free, readln and writeln.

The Tools disk contains the Draco compiler, linker, and other tools, along with the various standard Draco libraries. It also contains the "config" program used to configure (for a given terminal type) the included "CMP" and "HEDIT" programs. File list:

```
README.TXT - shareware notice
FILES.TXT - this file
DRACO.COM - the compiler itself (this version of the compiler lacks long
    (prefix 0L) and float constants, but will run in a 56K TPA)
DRCERR.DAT - file of error messages used by the compiler (if this isn't on
    your default drive, error numbers only will be printed by the compiler)
LINK.COM - the link editor
TRRUN.LIB - library of run-time routines (automatically searched)
TRCPM.LIB - library of CP/M interface routines (automatically searched)
CRT.LIB - library of CRT independent I/O routines (programs need configuring)
UTIL.G - include file with some utility declarations
OPDEF.G - include file with operator bits for operator types
CRT.G - include file for CRT independent screen I/O
FORM.G - include file for input forms part of CRT.LIB
DLIB.COM - Draco librarian
DAS.COM - Draco assembler
DDIS.COM - Draco dissassembler
XREF.COM - object file cross-referencer
DED.SET - Draco screen editor (configurable)
CONFIG.COM - program to manipulate terminal definitions and to configure
    programs which use the terminal independent I/O routines (*.SET)
CONFIG.DAT - database of terminal definitions
DCONFIG.COM - program to assign key bindings to Ded commands
CMP.SET - visual binary/text file comparison program (configurable)
HEDIT.SET - hexadecimal file viewer/editor (configurable)
BIGDRACO.COM - version of compiler with long/float constants (needs a
    60K TPA to run). The other version understands types 'float', 'long' and
    'ulong', but only this version understands long constants (> 16 bit or
    with a leading '0L') and float constants.
```

The Others disk contains various games and data files for use with them, along with another copy of the "config" program. The "Wrung" game is vaguely similar to commercial games like "Donkey Kong", except that it does not operate in real-time - it allows you to think carefully about each move before you do it. The "Switch" game system allows the creation of railroad layouts, described via a description language. The layouts can represent real layouts ("emra.swt" describes the layout of the Edmonton Model Railroad Association as it existed in the old Edmonton Gardens), or can be puzzles, to be solved using the minimum number of moves. All of the programs here use only text for their displays. File list:

```
README.TXT - shareware notice
FILES.TXT - this file
CONFIG.COM - program to configure the various .SET files
CONFIG.DAT - database of terminal definitions for use with CONFIG.COM
TWINKLE.SET - mindless demo
WORMS.SET - slightly less mindless demo - see 'worms ?'
GAS.SET - slightly less mindless demo - see 'gas ?'
RAIN.SET - mindless demo
SNOW.SET - Canadian version
NUCLEAR.SET - amusing two player game
ROBOTS.SET - amusing one player game
LIFE.SET - decent version of John Conway's simulation
CYCLES.LIF - save file of several Life cyclic patterns
SHIPS.LIF - save file of some Life spaceships
GUN.LIF - save file of Life glider gun
WRUNG.SET - fairly complex game - can be addictive!
WRUNG.LOG - some scores to try for
SWITCH.SET - complex railroad simulation/game
SWCREATE.COM - layout compiler for Switch
SWMERGE.COM - recording merge program for Switch
*.SWT - layout files for Switch
*.REC - recording files for two of the layouts
*.LOG - score files for some of the layouts
QUEST.SET - CRT adventure system lacking only an adventure
SLAM.SET - beginnings of an interactive language environment
```

Source Archives

File Draco2.3Src.zip contains the sources for version 2.3 of the CP/M Draco compiler. This was the latest version of the Draco compiler for CP/M. Later work done on CP/M was the cross compiler for the Amiga's MC68000 CP/U. Note that assembler sources use the syntax of my "das.com" Draco assembler. That assembler produces relocatable output files that must be linked with my "link.com" linker.

File RunTime.zip contains the various source files for the runtime system for Draco. Some of the files:

```
fltarith.drc - start of software floating point arithmetic (ignore)
lngarith.das - assembler source for 32 bit arithmetic
lngio.drc - Draco source for I/O of 32 bit values
malloc.drc - Draco source for storage allocation/free
rtconv.drc - Draco source for 8 and 16 bit input/output conversions
rtio.drc - Draco source for input/output utilites ("read" and "write" constructs)
rtpars.drc - access to command-line parameters
stdio.drc - Draco source for utility procs
trcpm.das - assembler source for CP/M interface stubs
trrun.das - assembler source for run-time system
util.drc - Draco source for more utility routines
```

File Utils1.zip contains miscellaneous system sources. Some of the files:

```
bit32.drc - looks like a different interface to 32 bit arithmetic
cmp.drc - file compare program
das.drc - Draco assembler (for Intel 8080 CPU)
ddis.drc - Draco disassembler (for Intel 8080 CPU)
dlib.drc - Draco library builder
dump.drc - Draco .rel file dumper
entab.rig - ancient tab-inserter (note the ".rig" suffix!)
find.drc - find strings in files
```

```
link.drc - Draco linker
pr.drc - program to print stuff on an Epson MX-80 printer
wc.drc - character/word/line counter
```

File Utils2.zip contains more miscellaneous sources. Some of the files:

```
del.drc - get rid of CP/M files with lower-case letters in their names
find.drc - find string in files
flin.drc - floating point input
hedit.drc - visual hexadecimal file editor
ncmp.drc - side-by-side file text/binary file compare program
nmkerr.drc - create the DRCERR.DAT error message file
strings.drc - show text strings in binary files
uudecode.drc, uuencode.drc - simple standard for encoding binaries as text
wator.drc - the Wator world of sharks and fish from Scientific American (1984)
xref.drc - cross-referencer program for Draco .rel files
```

Here is a random source file that isn't written in Draco, and doesn't run on CP/M. It is half of a communication protocol, written in my QC language, which allowed binary and text file transfers between the Michigan Terminal System running on an IBM mainframe and a CP/M system running my "call" modem program. It has no real relevance here other than as a curiosity. The "call" program *was* written in Draco for CP/M. It almost became a part of the software distribution for the Franklin Ace computer (a machine which included Apple-II compatible hardware along with a Z-80 CPU to run CP/M). This is the only example that I think I still have that shows QC's "lproc"s, which ran in the context of their containing caller.

Here is an archive of the sources to my Draco editor, Ded. This was a multi-file editor usable for programming, documentation, and most general text editting work. The editor occupied about 18K of RAM, leaving lots of space for in-memory copies of the files being edited. Ded was configurable as to the terminal it was running on using the "config" program in some of the above archives. It was also configurable as to its command key bindings using the "dconfig" progam whose source is included here. Documentation for Ded is on the first "distribution" disk, above.

One of the projects that a partner and I did was to write a text formatting program. This is something along the lines of the "nroff" program on UN*X sytems. We also had experience with the *FMT and *TEXTFORM programs on the MTS system. These programs operate as the non-WSIWYG part of things like Microsoft Word. They take an input file which contains the text to be formatted along with embedded commands and tags that describe how the text is to be formatted. Our program, which we called "Doc", supported "diversions" and a fairly full programming language which we used to implement page headers/footers, footnotes, tables of contents, indexes, etc. The doc.zip archive contains the Draco source to that program, along with standard macro libraries, test files, etc. Note that "doc" does not support multiple fonts - the concept didn't really exist on CP/M systems, since output was on a text-only terminal or on early printers which didn't support multiple fonts.

At one point, a bunch of folks running under the MTS operating system at the University of Alberta decided they wanted to write a multi-user game system. The folks running the mainframe said they would create a shared memory segment that could be used for interprocess communications. After lots of talk, I decided to see what I could do on CP/M. I came up with a game system (written in Draco of course) that could be the framework for something bigger. It used my CRT library to do full-screen text input/output. The top half of the screen was used for a small character-based overhead view of a map, plus a status area. The bottom half was a command area with a few lines of history.

I called my little system "Quest". Draco sources for it are here. This was my first try after Six/Fant at English language command parsing. It worked OK, but wasn't as extensive as I eventually got with my AmigaMUD system. The test system had a small world, a couple of objects to play with, and a dozen or so simple commands. I showed the sources to the rest of the crew, but nothing ever came of it.

Another project of mine was to explore creating a highly interactive, language-specific text editor. The language it was to be specific to was a lot like Draco, of course. For lack of anything more meaningful, I called it "SLAM", which was an acronym for "Structured LAnguage for Microprocessors". I'm not sure what my final goal was. A friend tried it out a bit and said it got in the way of code entry too much (think of "electric-C" mode in Gnu emacs). Sources are here: Slam.zip.

Explore

The next two archives represent another large chunk of my time. I had on my CP/M system a graphics card from Godbout electronics. It was very low resolution and very low colour choice compared to modern displays, but it was *graphics*!!! :-) I had seen bits of the Ultima games on a friend's Apple-II computer and wanted to do something like that. The result was my "Explore" game system and "Amelon" world for it.

Explore.zip is the source for the Explore system, including its interactive world creation program. For some reason, there is a simple Mandelbrot set program in the archive. Amelon.zip is the source for the Amelon world.

Explore did not use any overlays or paging, so floppy disk I/O was minimal. Memory was the main constraint on the system. This lead to a strange setup where I made the Draco linker create an extra output file which gave the addresses of all of the procs in the program it had linked. That file could then be given to another run of the linker, and the linker would use those addresses to resolve references from another program being linked. This allowed, for example, sharing of the Draco runtime system between the Explore program and world code of the world being run.

Explore had two graphics modes - a "grid" mode and a "maze" mode. In grid mode, the display showed an array of rectangular tiles representing various terrain types: water, grass, forest, mountain, desert, etc. The icon for the character blinked on and off over the current tile. Movement was restricted to the 4 orthogonal directions.

In maze mode, graphics output was a rough 3-D drawing of the current view of a level of a maze that the character is in. This would include doorways and joining passages. If an object or "monster" was present, it would be shown over top of the maze view. Since I am by no means an artist, the monster images were unbelievably poor. The character could move forward or turn left/right/around, along with the usual combat, spell use, etc..

In both modes, any text output was shown on the normal CP/M terminal. In the graphics mode that gave me a good set of colours, the resolution was simply too low to reasonably do text output on the graphics card. In my case this involved a lot of head turning between the terminal and my television set.

Typical keyboard input was acted upon immediately - e.g. using the numeric keypad on the terminal for movement. In some situations, prompts asked for longer input.

The world creation program displayed interactive views using the graphics card, along with prompts, errors, etc. on the CP/M terminal. This made world creation fairly straightforward, other than the programming necessary. Programming for the world was done in Draco, and the Explore program would load that code (from "amelon.cod") during startup. Selected Draco global variables in the world sources could be save/restored as part of the game save/restore features.

The Amelon world took quite a few hours to play through. It had a main world grid of the island of Amelon, and several smaller grids representing the various cities and towns on the island. Many of the cities and towns had one or more underground mazes to be explored, including one that went underwater to a smaller island. The world provided weapons, armour, two kinds of simple magic, a map, a horse, a ship and a magic carpet. Completing the game required winning a game of tic-tac-toe against the computer. It was played within one of the larger mazes in the world.

The Explore system and Amelon world were eventually ported to the Amiga. That version had better graphics abilities, and had the text and graphics on the same physical screen. My monsters in maze mode looked even worse in higher resolution!

Database

Another project that my partner and I did was to create a database system in the style of relational databases. The first version, I wrote in Draco, and no longer have the sources to. It was later translated to C, where it ran on MS-DOS and Xenix.

DB.zip appears to be the latest version of the C sources for the database system. DBUtil.zip was accompanying utilities and documentation.

My partner and I did a program using parts of it called "OmniDial". It was intended to be used as a lookup system for local events, etc. People wanting to find stuff would call up the operators, who would use our software to find and update the various events. We had the system basically working, and I have the sources for it. However, it never actually launched, and so we stopped work on it.

My partner got a contract to implement a tracking system for a "hotshot" company - a company which did quick deliveries out to the oilfields. I don't recall the details, but I believe it used the database system, with some multi-user locking added on top of the system by another fellow, to run this project on a Xenix system (a Unix variant). Real money actually came in for that project.

dBase-II

Early in my CP/M work, I was contacted by a fellow who wanted some custom business systems created. I did three of them with him. He did all of the customer interaction and going around getting things going. I did the programming, using the dBase-II system. I won't name the three companies involved, but we were paid for the work, and I know at least one of them used the system for several years. The largest of the systems included user accounts, invoicing, billing, and inventory control. This included the creation of several custom reports.

To get me started with this, another fellow who first pointed me at this work gave me the sources to an application of his for a local shipping company. The companies I did my stuff for were a refrigeration supply/service company, a pump service company, and a janitorial supply company.

For those interested Wikipedia has a page on the history of the dBase system.

Switch

I've been a fan of trains since I was a small child. One of the projects I did on CP/M reflects that. The "switch" program is a railroad layout simulator. It uses character graphics to show some or all (depending on how large the layout is relative to the size of the terminal screen) of the track in a layout, along with engines and cars on the track. The user can scroll around in larger layouts. The user can toggle the turnout ("switch") directions, move engines and uncouple cars. Like with most model trains, cars and engines automatically couple when they touch.

Documentation on "switch", "swmerge" and "swcreate" is in the "SWITCH.REF" file in the "Doc_Src.zip" archive. Several of the provided layouts (".swt" files, with source in ".sws" files) are "switching puzzles". The idea behind a switching puzzle is to get all of the cars and engines from their initial positions to their target positions using the fewest number of moves. This can be quite tricky. Also included is file "EMRA.SWT" which is a representation of an old model railroad layout owned by the Edmonton Model Railroad Association. There is recording file "EMRA.REC" which can be played within "switch" to show a simple run of that layout.

Layout "SAWBY.SWT" is an example that real train engineers occasionally encounter. It happens when two trains meet head-on on a single piece of track with a "passing siding", but the siding is not long enough for either train. How do the trains get past each other? Try it yourself before playing the "SAWBY.REC" file.

The layouts included are: BASIC, EMRA, MAINA, MAIND, MAINM, MANY, MESSY, SAWBY, SMALLA, SMALLD, SMALLM and TRICKY. The "xxxA" layouts model the arrival of a freight train at a switching yard. Layouts "xxxD" model departure and "xxxM" model mixed arrival/departure. Score files are included for BASIC, MAINA, MAIND, MAINM, MESSY, SMALLA, SMALLD, SMALLM and TRICKY. Layout source files are included for all 12 layouts.

To make things somewhat easier, here is a fully setup CP/M disk file for use with the "yaze" emulator. Save it to a Unix/Linux directory; put a copy of the yaze.boot file in that directory, then run yaze. Make sure you are in an xterm - switch is pre-configured for that. In yaze, mount the disk as "mount a switch.disk". Then start CP/M with "go". You should get the "A>" CP/M prompt. Type "switch" to run switch. It has some built-in help. On exit from "switch", type "sys quit" to exit from the yaze emulator.

Up Home