NAME

c2ph, pstruct - Dump C structures as generated from `cc -g -S` stabs

SYNOPSIS

```
c2ph [-dpnP] [var=val] [files ...]
```

OPTIONS

```
Options:

-w wide; short for: type_width=45 member_width=35 offset_width=8
-x hex; short for:  offset_fmt=x offset_width=08 size_fmt=x
size_width=04

-n do not generate perl code  (default when invoked as pstruct)
-p generate perl code         (default when invoked as c2ph)
-v generate perl code, with C decls as comments

-i do NOT recompute sizes for intrinsic datatypes
-a dump information on intrinsics also

-t trace execution
-d spew reams of debugging output

-slist  give comma-separated list a structures to dump
```

DESCRIPTION

The following is the old c2ph.doc documentation by Tom Christiansen <tchrist@perl.com> Date: 25 Jul 91 08:10:21 GMT

Once upon a time, I wrote a program called pstruct. It was a perl program that tried to parse out C structures and display their member offsets for you. This was especially useful for people looking at binary dumps or poking around the kernel.

Pstruct was not a pretty program. Neither was it particularly robust. The problem, you see, was that the C compiler was much better at parsing C than I could ever hope to be.

So I got smart: I decided to be lazy and let the C compiler parse the C, which would spit out debugger stabs for me to read. These were much easier to parse. It's still not a pretty program, but at least it's more robust.

Pstruct takes any .c or .h files, or preferably .s ones, since that's the format it is going to massage them into anyway, and spits out listings like this:

```
 struct tty {
   int                          tty.t_locker                          000
      4
   int                          tty.t_mutex_index                     004
      4
   struct tty *                 tty.t_tp_virt                         008
      4
   struct clist                 tty.t_rawq                            00c
     20
     int                        tty.t_rawq.c_cc                       00c
      4
```

```
    int                     tty.t_rawq.c_cmax               010
     4
    int                     tty.t_rawq.c_cfx                014
     4
    int                     tty.t_rawq.c_clx                018
     4
    struct tty *            tty.t_rawq.c_tp_cpu             01c
     4
    struct tty *            tty.t_rawq.c_tp_iop             020
     4
    unsigned char *         tty.t_rawq.c_buf_cpu            024
     4
    unsigned char *         tty.t_rawq.c_buf_iop            028
     4
  struct clist              tty.t_canq                      02c
    20
    int                     tty.t_canq.c_cc                 02c
     4
    int                     tty.t_canq.c_cmax               030
     4
    int                     tty.t_canq.c_cfx                034
     4
    int                     tty.t_canq.c_clx                038
     4
    struct tty *            tty.t_canq.c_tp_cpu             03c
     4
    struct tty *            tty.t_canq.c_tp_iop             040
     4
    unsigned char *         tty.t_canq.c_buf_cpu            044
     4
    unsigned char *         tty.t_canq.c_buf_iop            048
     4
  struct clist              tty.t_outq                      04c
    20
    int                     tty.t_outq.c_cc                 04c
     4
    int                     tty.t_outq.c_cmax               050
     4
    int                     tty.t_outq.c_cfx                054
     4
    int                     tty.t_outq.c_clx                058
     4
    struct tty *            tty.t_outq.c_tp_cpu             05c
     4
    struct tty *            tty.t_outq.c_tp_iop             060
     4
    unsigned char *         tty.t_outq.c_buf_cpu            064
     4
    unsigned char *         tty.t_outq.c_buf_iop            068
     4
  (*int)()                  tty.t_oproc_cpu                 06c
     4
  (*int)()                  tty.t_oproc_iop                 070
     4
  (*int)()                  tty.t_stopproc_cpu              074
     4
```

```
(*int)()                    tty.t_stopproc_iop                  078
   4
struct thread *             tty.t_rsel                          07c
   4
```

etc.

Actually, this was generated by a particular set of options. You can control the formatting of each column, whether you prefer wide or fat, hex or decimal, leading zeroes or whatever.

All you need to be able to use this is a C compiler than generates BSD/GCC-style stabs. The **-g** option on native BSD compilers and GCC should get this for you.

To learn more, just type a bogus option, like **-\?**, and a long usage message will be provided. There are a fair number of possibilities.

If you're only a C programmer, than this is the end of the message for you. You can quit right now, and if you care to, save off the source and run it when you feel like it. Or not.

But if you're a perl programmer, then for you I have something much more wondrous than just a structure offset printer.

You see, if you call pstruct by its other incybernation, c2ph, you have a code generator that translates C code into perl code! Well, structure and union declarations at least, but that's quite a bit.

Prior to this point, anyone programming in perl who wanted to interact with C programs, like the kernel, was forced to guess the layouts of the C structures, and then hardwire these into his program. Of course, when you took your wonderfully crafted program to a system where the sgtty structure was laid out differently, your program broke. Which is a shame.

We've had Larry's h2ph translator, which helped, but that only works on cpp symbols, not real C, which was also very much needed. What I offer you is a symbolic way of getting at all the C structures. I've couched them in terms of packages and functions. Consider the following program:

```
#!/usr/local/bin/perl

require 'syscall.ph';
require 'sys/time.ph';
require 'sys/resource.ph';

$ru = "\0" x &rusage'sizeof();

syscall(&SYS_getrusage, &RUSAGE_SELF, $ru)      && die "getrusage: $!";

@ru = unpack($t = &rusage'typedef(), $ru);

$utime =  $ru[ &rusage'ru_utime + &timeval'tv_sec  ]
+ ($ru[ &rusage'ru_utime + &timeval'tv_usec ]) / 1e6;

$stime =  $ru[ &rusage'ru_stime + &timeval'tv_sec  ]
+ ($ru[ &rusage'ru_stime + &timeval'tv_usec ]) / 1e6;

printf "you have used %8.3fs+%8.3fu seconds.\n", $utime, $stime;
```

As you see, the name of the package is the name of the structure. Regular fields are just their own names. Plus the following accessor functions are provided for your convenience:

```
  struct This takes no arguments, and is merely the number of first-level
elements in the structure.  You would use this for indexing
into arrays of structures, perhaps like this

    $usec = $u[ &user'u_utimer
  + (&ITIMER_VIRTUAL * &itimerval'struct)
  + &itimerval'it_value
  + &timeval'tv_usec
      ];

  sizeof    Returns the bytes in the structure, or the member if
     you pass it an argument, such as

 &rusage'sizeof(&rusage'ru_utime)

  typedef   This is the perl format definition for passing to pack and
     unpack.  If you ask for the typedef of a nothing, you get
     the whole structure, otherwise you get that of the member
     you ask for.  Padding is taken care of, as is the magic to
     guarantee that a union is unpacked into all its aliases.
     Bitfields are not quite yet supported however.

  offsetof This function is the byte offset into the array of that
member.  You may wish to use this for indexing directly
into the packed structure with vec() if you're too lazy
to unpack it.

  typeof Not to be confused with the typedef accessor function, this
one returns the C type of that field.  This would allow
you to print out a nice structured pretty print of some
structure without knoning anything about it beforehand.
No args to this one is a noop.  Someday I'll post such
a thing to dump out your u structure for you.
```

The way I see this being used is like basically this:

```
% h2ph <some_include_file.h  >  /usr/lib/perl/tmp.ph
% c2ph  some_include_file.h  >> /usr/lib/perl/tmp.ph
% install
```

It's a little tricker with c2ph because you have to get the includes right. I can't know this for your system, but it's not usually too terribly difficult.

The code isn't pretty as I mentioned -- I never thought it would be a 1000- line program when I started, or I might not have begun. :-) But I would have been less cavalier in how the parts of the program communicated with each other, etc. It might also have helped if I didn't have to divine the makeup of the stabs on the fly, and then account for micro differences between my compiler and gcc.

Anyway, here it is. Should run on perl v4 or greater. Maybe less.

```
--tom
```