The '`-a`' switch sets the name of the routine used by the `advise` facility (see page 74).

The '`-d`' switch directs *pepy* to ignore any action statements in the input.

The '`-h`' switch enables additional heuristics when *pepy* generates a printer.

The '`-o`' switch sets the name of the output file, which is normally derived from the name of the input file. The distinguished file "`-`" can be used to force the use of the standard output.

If the '`-o`' switch is not used, then `-b␣prefix` can be used to direct *pepy* to produce multiple output files, each starting with `prefix`.

The '`-P`' switch directs *pepy* not to include *cpp*-type line number information in the output. This is useful for debugging *pepy* (gasp!).

The '`-p`' switch directs *pepy* to remove the action states from the input file and print the resulting information on the standard output. This typically results in a file suitable for pretty-printing.

The '`-r`' switch enables the generation of "robust" code. If this option is given, then the *C* code generated will not check for unknown or extraneous objects. This is used for extensibility purposes.

Normally, *pepy* prints the name of each type as it works. The '`-s`' switch disables this behavior.

Finally, the '`-S`' switch sets the initial section-processing mode for *pepy*. The default is `-S␣DECODE`, which causes *pepy* to generate a decoder. Other values are `-S␣ENCODE` to generate an encoder; or, `-S␣PRINT` to generate a printer. The '`-A`' switch is used to set the initial mode to generate encoder, decoder, and printer routines.

6.6.3 Makefiles

By convention, input files to *pepy* have the extension `.py`. The rules for *make*(1) are:

```
.SUFFIXES:      .py .c .o

.py.o:;         pepy $(PYFLAGS) $<
                $(CC) $(CFLAGS) -c $*.c
                -rm -f $*.c

.py.c:;         pepy $(PYFLAGS) $<
```

6.6.4 Grinding

Figure 6.7 contains an entry for the *vgrindefs*(5) file, which may be used by the *vgrind*(1) or *tgrind*(1) facility. While this entry is not perfect, it has proven to be adequate for most purposes; e.g., the examples in this chapter were pretty-printed using this entry.

In addition, if you want to get an ASN.1 specification by removing all of the *C* augmentations from the *pepy* input file, use the '`-p`' switch. For example,

```
pepy -p < myfile.py
```

can be used to view the ASN.1 specification in the file `myfile.py`.

6.7 An Example

Let's consider how one might construct a pretty-printer for the personnel records described in Figure 6.1 on page 61. (Of course, earlier in Section 6.5 the preferred method of generating pretty-printers was described.)

First, we start the module off by defining the usual prefactory *C* code as verbatim actions. This is shown in Figure 6.8. Next, we define the `PersonnelRecord` types. The rest of the types are then defined. This is shown in Figures 6.9 and 6.10 respectively. Finally, the usual closing *C* code is included as verbatim actions, as shown in Figure 6.11.

```
pepy|asn1|asn.1|ASN1|ASN.1|x409|x.409|X409|X.409:\
:pb=^\d?\p\d?\:\:\=:tl:\
:bb=\dBEGIN\d|{|\[|\(:be=\dEND\d|}|\]|\):\
:cb=--:ce=--|$:sb="|':se="|':\
:kw=ANY APPLICATION BEGIN BIT BITSTRING BOOLEAN\
    CHOICE COMPONENTS DEFAULT DEFINITIONS END\
    EXTERNAL FALSE IDENTIFIER IMPLICIT INTEGER\
    NULL OBJECT OCTET OCTETSTRING OF OPTIONAL\
    PRIVATE SEQUENCE SET STRING TRUE UNIVERSAL\
MACRO NOTATION TYPE VALUE\
ABSENT BY COMPONENT DEFINED ENCRYPTED\
    ENUMERATED EXPLICIT EXPORTS FROM IMPORTS\
    INCLUDES MIN MAX PRESENT REAL SIZE TAGS\
    WITH\
ARGUMENT BIND ERROR ERRORS LINKED OPERATION\
    PARAMETER RESULT UNBIND\
ABSTRACT AS CONSUMER INVOKES OPERATIONS PAIRED\
    PORT PORTS RECURRING REFINE SUPPLIER\
    VISIBLE [C] [S]\
ATTRIBUTE CLASS CONTAIN EMPTY EQUALITY FOR\
    MATCHES MAY MULTI MULTIPLE MUST NUMBER\
    SINGLE SUBCLASS SUBSTRINGS SYNTAX USES\
CONTEXT ELEMENT ELEMENTS INITIATOR REMOTE SERVICE\
    SYNTAXES\
OBJECT-TYPE SYNTAX ACCESS STATUS\
DECODER ENCODER PRINTER SECTIONS %{ %} [[ ]] << >>:
```

Figure 6.7: Grind Entry for pretty-printing PEPY programs

```
-- personnel record pretty-printer

PERSONNEL DEFINITIONS ::=

%{
#define DEBUG

#include <stdio.h>

                                                                    10
#define ps_adios(ps, f) \
        adios (NULLCP, "%s:  %s", (f), ps_error ((ps) -> ps_errno))


static char *myname = "prpp";


void    adios ();

                                                                    20
static char *givenName;
static char *initial;
static char *familyName;
static char *Title;
static char *Date;


static char *months[] = {
   "January", "February", "March", "April", "May", "June",
   "July", "August", "September", "October", "November", "December"     30
};

...
```

Figure 6.8: Pretty-Printer initialization

```
...

/*    MAIN */

/* ARGSUSED */

main (argc, argv, envp)
int       argc;
char  **argv,
     **envp;                                                    10
{
    PE        pe;
    PS        ps;

    myname = *argv;
    if ((ps = ps_alloc (std_open)) == NULLPS)
            adios (NULLCP, "ps_alloc loses");
    if (std_setup (ps, stdin) == NOTOK)
            ps_adios (ps, "std_setup");
                                                               20
    for (;;) {
            if ((pe = ps2pe (ps)) == NULLPE)
                if (ps -> ps_errno)
                        ps_adios (ps, "ps2pe");
                else
                        break;

            (void) parse_PERSONNEL_PersonnelRecord (pe, 1, NULLIP, NULLVP,
                        NULLCP);
                                                               30
            pe_free (pe);
    }

    ps_free (ps);

    exit (0);
}

%}
                                                               40
...
```

Figure 6.8: Pretty-Printer initialization (continued)

```
...

BEGIN

PersonnelRecord
        %{ char *Title; %}
        ::=
        [APPLICATION 0]
           IMPLICIT SET {
                   Name                                                    10
                   %{
                         printf ("Name:   %s %s.   %s\n",
                                 givenName, initial, familyName);
                   %},

                title[0]
                   IA5String
                   [[s Title]]
                   %{ printf ("Title:   %s\n", Title); %},
                                                                           20
                   EmployeeNumber,

                dateOfHire[1]
                   Date
                   %{ printf ("Date of Hire:   %s\n", Date); %},

                nameOfSpouse[2]
                   Name
                   %{
                         printf ("Name of Spouse:   %s %s.   %s\n",        30
                                 givenName, initial, familyName);
                   %},

                [3]
                   %{
                         PE      cc = prim2seq ($$);
                         if (cc)
                            printf ("Number of Children:   %d\n",
                                    cc -> pe_cardinal);
                   %}                                                      40
                   IMPLICIT SEQUENCE OF ChildInformation
                   DEFAULT {}
           }
           %{ printf ("--------\n"); %}
...
```

Figure 6.9: Pretty-Printer top-level

```
...

ChildInformation ::=
        SET {
                Name
                %{
                   printf ("\nChild Information\nName:  %s %s.  %s\n",
                         givenName, initial, familyName);
                %},

           dateofBirth[0]
                Date
                %{ printf ("Date of Birth:  %s\n", Date); %}
        }

Name ::=
        [APPLICATION 1]
           IMPLICIT SEQUENCE {
                givenName
                   IA5String [[s givenName]],

                initial
                   IA5String [[s initial]],

                familyName
                   IA5String [[s familyName]]
           }

EmployeeNumber ::=
        [APPLICATION 2]
           IMPLICIT INTEGER
           %{ printf ("Employee Number:  %d\n", $$); %}

...
```

Figure 6.10: Pretty-Printer type definitions

```
...

Date ::=
        [APPLICATION 3]
          IMPLICIT IA5String -- YYYYMMDD --
          [[s Date]]
          %{
                int year, month, day;
                static char buffer[BUFSIZ];
                                                                    10
                if (sscanf (Date, "%4d%2d%2d", &year, &month, &day) != 3
                        || year < 1900 || month < 1 || month > 12
                        || day < 1 || day > 31)
                   adios (NULLCP, "bad date format:  %s", Date);
                (void) sprintf (buffer, "%02d %s %4d", day,
                              months[month - 1], year);
                Date = buffer;
          %}

END                                                                 20

...
```

Figure 6.10: Pretty-Printer type definitions (continued)

```
...

%{
/*    DEBUG */

#ifdef  DEBUG
testdebug (pe, s)
register PE pe;
register char *s;
{                                                                    10
    static int  debug = OK;
    char   *cp;
    register PS     ps;

    switch (debug) {
        case NOTOK:
            break;

        case OK:
            if ((debug = (cp = getenv ("PEPYDEBUG")) && *cp ? atoi (cp)      20
                        : NOTOK) == NOTOK)
                break;
            fprintf (stderr, "%s made with %s\n", myname, pepyid);

        default:
            fprintf (stderr, "%s\n", s);

            if ((ps = ps_alloc (std_open)) == NULLPS)
                return;
            if (std_setup (ps, stderr) != NOTOK)                         30
                (void) pe2pl (ps, pe);
            fprintf (stderr, "--------\n");
            ps_free (ps);
            break;
    }
}
#endif

...
```

Figure 6.11: Pretty-Printer finalization

```
...

/*    ERRORS */

#include <varargs.h>


/* VARARGS2 */

void        adios (what, fmt, va_alist)
char   *what,
       *fmt;
va_dcl
{
    advise (what, fmt, va_alist);

    _exit (1);
}


/* VARARGS2 */

void        advise (what, fmt, va_alist)
char   *what,
       *fmt;
va_dcl
{
    char    buffer[BUFSIZ];

    asprintf (buffer, what, fmt, va_alist);

    (void) fflush (stdout);

    fprintf (stderr, "%s:  ", myname);
    (void) fputs (buffer, stderr);
    (void) fputc ('\n', stderr);

    (void) fflush (stderr);
}

%}
```

Figure 6.11: Pretty-Printer finalization (continued)