References

Aho, A. V., Kernighan, B. W., and Weinberger, P. J. The AWK programming language. Reading, MA: Addison-Wesley, 1988.

————, Sethi, R., and Ullman, J. D. Compilers: principles, techniques and tools. Reading, MA: Addison-Wesley, 1986.

Friedman Jr, H. G., and Schreiner, A. T. Introduction to compiler construction with Unix. Englewood Cliffs, NJ: Prentice-Hall, 1985.

Johnson, S. C. Yacc - yet another compiler compiler. In B. W. Kernighan and M. D. McIlroy (eds.), Unix Programmer's Manual, Seventh edition, Volume 2, pp. 353-387. New York: Holt, Rinehart, and Winston, 1979.

Kernighan, B. W., and Pike, R. The Unix programming environment. Englewood Cliffs, NJ: Prentice-Hall, 1984.

————, and Ritchie, D. M. The C programming language. 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1988.

Lesk, M. E., and Schmidt, E. Lex - a lexical analyzer generator. In B. W. Kernighan and M. D. McIlroy (eds.), Unix Programmer's Manual, Seventh edition, Volume 2, pp. 388-400. New York: Holt, Rinehart, and Winston, 1979.

Pratt, T. W. Programming languages: design and implementation. 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1984.

Tremblay, J., and Sorenson, P. G. The theory and practice of compiler writing. New York: McGraw-Hill, 1985.

Appendix A

The AWC-to-C Translator User Manual

A.1 Unpackaging the Source Code

The source code of the AWK-to-C translation system is distributed in a single file named *awc-1.0.tar*, where the number *1.0* is the version number. This file should be copied into an empty directory and then unpackaged by the following command:

```
$ tar xvf awc-1.0.tar
```

which will create many subdirectories and a large number of source files.

A.2 Compilation and Installation

To compile and install the AWK-to-C translation system, do the following steps:

1. Use an editor to edit *Makefile* in the top-level directory to set the values for the *make*'s variables *bindir*, *libdir*, and *includedir* to be the directory names for installing the executable files, the library and the skeleton file, and the header files, respectively. For example, you may set these three variables as follows:

```
prefix = /usr/local
bindir = $(prefix)/bin
libdir = $(prefix)/lib
includedir = $(prefix)/include
```

2. To build the translator and the libawc library, just run the following command:

```
$ make
```

3. If the compilation is successful, then the software is ready to be installed into the system. First, make sure that you have write-permission to the three directories you set in *bindir*, *libdir*, and *includedir*. Then, run the following command:

```
$ make install
```

This will copy the executable files *awc* and *awcc* into the *bindir* directory, the library *libawc.a* and the skeleton file *awcskel.c* into the *libdir* directory, and the header files *awclib.h* and *regex.h* into the *includedir* directory.

A.3 Using the Translator

Using the AWK-to-C translator is very simple. Suppose you want to translate an AWK program named *foo.awk* into a C program named *foo.c*, just type in the command:

```
$ awc foo.awk > foo.c
```

Note that the translator always outputs the generated code to the standard output file.

If you want to tranlate *foo.awk* and then compile the generated C program *foo.c* to produce the executable file *foo* in a single step, just use the shell script *awcc* to do the task:

```
$ awcc foo.awk
```

Once you have the executable program *foo*, running the following command:

```
$ foo -F'\t' data1 v=1 data2
```

should give the same result as using *nawk* to run the AWK source program:

```
$ nawk -F'\t' -f foo.awk data1 v=1 data2
```

Appendix B

A Yacc Grammar for AWK Used by the Parser

```
start
      : opt_nls program opt_nls
      ;
program
      : rule
      | program rule
      ;
rule
      : awk_begin action
      | awk_end action
      | awk_begin statement_term
      | awk_end statement_term
      | pattern action
      | action
      | pattern statement_term
      | function_prologue function_body
      ;
awk_begin
      : TOK_BEGIN
      ;
awk_end
      : TOK_END
      ;
function_prologue
      : TOK_FUNCTION func_name
        '(' opt_param_list r_paren opt_nls
      ;
function_body
      : l_brace statements r_brace opt_semi
      ;
```

```
func_name
      : FUNC_CALL
      | NAME
      ;
pattern
      : exp
      | exp comma exp
      ;
regexp
      : '/' REGEXP '/'
      ;
action
      : l_brace statements r_brace opt_semi
      | l_brace r_brace opt_semi
      ;
statements
      : statement
      | statements statement
      ;
statement_term
      : nls
      | semi opt_nls
      ;
```

```
statement
      : semi opt_nls
      | l_brace r_brace
      | l_brace statements r_brace
      | if_statement
      | While condition opt_nls statement
      | Do opt_nls statement
        TOK_WHILE '(' exp r_paren opt_nls
      | TOK_FOR '(' opt_exp semi opt_exp semi
        opt_exp r_paren opt_nls statement
      | TOK_FOR '(' NAME TOK_IN NAME r_paren opt_nls
        statement
      | TOK_BREAK statement_term
      | TOK_CONTINUE statement_term
      | TOK_NEXT statement_term
      | TOK_EXIT opt_exp statement_term
      | TOK_CLOSE '(' exp r_paren statement_term
      | print '(' expression_list r_paren
        output_redir statement_term
      | print opt_rexpression_list
        output_redir statement_term
      | TOK_RETURN opt_exp statement_term
      | TOK_DELETE NAME '[' expression_list ']'
        statement_term
      | exp statement_term
      ;
print
      : TOK_PRINT
      | TOK_PRINTF
      ;
if_statement
      : If condition opt_nls statement
      | If condition opt_nls statement Else opt_nls
        statement
      ;
```

```
If
      : TOK_IF
      ;
Else
      : TOK_ELSE
      ;
While
      : TOK_WHILE
      ;
Do
      : TOK_DO
      ;
condition
      : '(' exp r_paren
      ;
nls
      : NEWLINE
      | nls NEWLINE
      ;
opt_nls
      :
      | nls
      ;
input_redir
      :
      | '<' simp_exp
      ;
output_redir
      :
      | '>' exp
      | APPEND_OP exp
      | '|' exp
      ;
```

```
opt_param_list
      :
      | param_list
      ;
param_list
      : NAME
      | param_list comma NAME
      ;
opt_exp
      :
      | exp
      ;
opt_rexpression_list
      :
      | rexpression_list
      ;
rexpression_list
      : rexp
      | rexpression_list comma rexp
      ;
opt_expression_list
      :
      | expression_list
      ;
expression_list
      : exp
      | expression_list comma exp
      ;
```

```
exp
        : variable ASSIGNOP exp
        | exp MATCHOP exp
        | regexp
        | '!' regexp
        | exp '?' exp ':' exp
        | exp '|' TOK_GETLINE opt_variable
        | TOK_GETLINE opt_variable input_redir
        | '(' expression_list r_paren TOK_IN NAME
        | exp TOK_IN NAME
        | exp TOK_AND exp
        | exp TOK_OR exp
        | exp RELOP exp
        | exp '<' exp
        | exp '>' exp
        | simp_exp
        | exp exp
        ;
rexp
        : variable ASSIGNOP rexp
        | rexp MATCHOP rexp
        | regexp
        | '!' regexp
        | rexp '?' rexp ':' rexp
        | TOK_GETLINE opt_variable input_redir
        | rexp TOK_IN NAME
        | rexp TOK_AND rexp
        | rexp TOK_OR rexp
        | rexp RELOP rexp
        | simp_exp
        | rexp rexp
        ;
```

```
simp_exp
      : non_post_simp_exp
      | post_inc_dec_exp
      | simp_exp '^' simp_exp
      | simp_exp '*' simp_exp
      | simp_exp '/' simp_exp
      | simp_exp '%' simp_exp
      | simp_exp '+' simp_exp
      | simp_exp '-' simp_exp
      ;
non_post_simp_exp
      : '!' simp_exp
      | '(' exp r_paren
      | TOK_BUILTIN '(' opt_expression_list r_paren
      | TOK_LENGTH '(' opt_expression_list r_paren
      | TOK_LENGTH
      | FUNC_CALL '(' opt_expression_list r_paren
      | INCREMENT variable
      | DECREMENT variable
      | TNUMBER
      | TSTRING
      | '-' simp_exp
      | '+' simp_exp
      ;
post_inc_dec_exp
      : variable INCREMENT
      | variable DECREMENT
      | variable
      ;
opt_variable
      :
      | variable
      ;
```

```
variable
      : NAME
      | NAME '[' expression_list ']'
      | '$' non_post_simp_exp
      | '$' variable
      ;
l_brace
      : '{' opt_nls
      ;
r_brace
      : '}' opt_nls
      ;
r_paren
      : ')'
      ;
opt_semi
      :
      | semi
      ;
semi
      : ';'
      ;
comma
      : ',' opt_nls
      ;
```

Appendix C

A Translation Example

This Appendix presents an example of AWK-to-C translation. The AWK program *fmt.awk* and the corresponding, translator-generated C program *fmt.c* are listed below:

Listing of fmt.awk

```
 1:  # fmt.awk - formatter with right justification
 2:
 3:  BEGIN { blanks = sprintf("%60s", " ") }
 4:  /./   { for (i = 1; i <= NF; i++) addword($i) }
 5:  /^$/  { printline("no"); print "" }
 6:  END   { printline("no") }
 7:
 8:  function addword(w) {
 9:      if (cnt + size + length(w) > 60)
10:          printline("yes")
11:      line[++cnt] = w
12:      size += length(w)
13:  }
14:
15:  function printline(f,    i, nb, nsp, holes) {
16:      if (f == "no" || cnt == 1) {
17:          for (i = 1; i <= cnt; i++)
18:              printf("%s%s", line[i], i < cnt ? " " : "\n")
19:      } else if (cnt > 1) {
20:          dir = 1 - dir        # alternate side for extra blanks
21:          nb = 60 - size       # number of blanks needed
22:          holes = cnt - 1      # holes
23:          for (i = 1; holes > 0; i++) {
24:              nsp = int((nb-dir) / holes) + dir
25:              printf("%s%s", line[i], substr(blanks, 1, nsp))
26:              nb -= nsp
27:              holes--
28:          }
29:          print line[cnt]
30:      }
31:      size = cnt = 0
32:  }
```

Listing of fmt.c

```
 1:  #include <stdio.h>
 2:  #include <string.h>
 3:  #include <stdlib.h>
 4:  #include <stdarg.h>
 5:  #include <math.h>
 6:  #include <sys/types.h>
 7:  #include <time.h>
 8:  #include <setjmp.h>
 9:  #include "awklib.h"
10:
11:  /*
12:   * The following defininitions correspond to AWK built-in variables.
13:   * The initial values for these variables rely on the rule of C that
14:   * external variables are guarateed to be initialized to zero. (K&R, 2nd Ed.
15:   * p.85)  Thus, all these Vartype objects should have initial value of
16:   * { V_UNDEF, 0.0, (char *) NULL, (ArrElem **) NULL } if not explicitly
17:   * initialized.
18:   *
19:   * Note that it's important that V_UNDEF be defined as 0.
20:   */
21:  VarType field[MAXFIELD + 1];    /* field variables $0 - $MAXFIELD */
22:  VarType ARGV_AWK;
23:  VarType ARGC_AWK;
24:  VarType ENVIRON_AWK;
25:  VarType FILENAME_AWK;
26:  VarType FNR_AWK =
27:  {V_NUM, 0.0, (char *) NULL, (ArrElem **) NULL};
28:  VarType FS_AWK;
29:  VarType NF_AWK =
30:  {V_NUM, 0.0, (char *) NULL, (ArrElem **) NULL};
31:  VarType NR_AWK =
32:  {V_NUM, 0.0, (char *) NULL, (ArrElem **) NULL};
33:  VarType OFMT_AWK;
34:  VarType OFS_AWK;
```

```
35:  VarType ORS_AWK;
36:  VarType RLENGTH_AWK;
37:  VarType RS_AWK;
38:  VarType RSTART_AWK;
39:  VarType SUBSEP_AWK;
40:
41:  static void initVar P((int argc, char **argv));
42:
43:  char *progname;              /* bare program name, extracted from argv[0] */
44:  char *effectiveFS;           /* effective FS string actually used to split fields */
45:  Regexp effectiveFSre;        /* regular expression for effective FS string */
46:  Boolean varList_filled = FALSE; /* TRUE, iff varList[] has been filled */
47:  int exitStatus = 0;          /* exit status for the program */
48:  Boolean inENDaction = FALSE;  /* TRUE, iff within the END action */
49:  jmp_buf sjbuf;               /* for setjmp call in main */
50:
51:
52:  /* current status of ARGV array, initial values are for index, eof and fp */
53:  struct argvStatType argvStat =
54:  {0, (FILE *) NULL};
55:
56:  /*------------------------------------------------------------------------*
57:   * initVar() - set initial value of some variables
58:   *------------------------------------------------------------------------*/
59:  static void
60:  initVar(argc, argv)
61:       int argc;
62:       char **argv;
63:  {
64:      /*
65:        extern int getopt P((int argc, char **argv, char *opts));
66:       */
67:      extern char *basename P((char *filespec));
68:
69:      extern int optind;
70:      extern char *optarg;
```

```
71:
72:     int i, ch;
73:     char sindex[10];
74:
75:     reSetup();              /* Do some setups about regular expression */
76:
77:     strAssign(&FILENAME_AWK, A_STR, "-");
78:     strAssign(&FS_AWK, A_STR, " ");
79:     strAssign(&OFMT_AWK, A_STR, "%.6g");
80:     strAssign(&OFS_AWK, A_STR, " ");
81:     strAssign(&ORS_AWK, A_STR, "\n");
82:     strAssign(&RS_AWK, A_STR, "\n");
83:     strAssign(&SUBSEP_AWK, A_STR, "\034");
84:
85:     setFS();                /* must be called everytime FS or RS gets new value */
86:
87:     /* value for ARGV[0] */
88:     strAssign(array(&ARGV_AWK, A_STR, "0"), A_STR, argv[0]);
89:     progname = basename(argv[0]);
90:
91:     /* process command-line options */
92:     while ((ch = getopt(argc, argv, "F:")) != EOF)
93:     {
94:         switch (ch)
95:         {
96:         case 'F':
97:             strAssign(&FS_AWK, A_STR, optarg);
98:             setFS();
99:             break;
100:        case '?':
101:            fatal("unknown option");
102:            break;
103:        }
104:    }
105:    argc -= optind;
106:    argv += optind;
```

```
107:
108:        /* value for ARGC */
109:        numAssign(&ARGC__AWK, argc + 1);
110:
111:        /* values for ARGV[1], ARGV[2], ... , ARGV[ARGC-1] */
112:        for (i = 1; i <= argc; ++i)
113:        {
114:            sprintf(sindex, "%d", i);
115:            xstrAssign(array(&ARGV__AWK, A_STR, sindex), A_STR, argv[i - 1]);
116:        }
117:    }
118:
119:
120:    /*-------------------------------------------------------------------------*
121:     * cleanup - do some housekeeping cleanup (supposedly before dying)
122:     *-------------------------------------------------------------------------*/
123:    void
124:    cleanup()
125:    {
126:        closeAll();                 /* close all i/o files and/or pipes */
127:    }
128:
129:            /* the rest of generated code comes here */
130:
131:    char regExp_0[] =
132:    {'.', '\0'};
133:    char regExp_1[] =
134:    {'^', '$', '\0'};
135:    VarType line__AWK;
136:    VarType dir__AWK;
137:    VarType cnt__AWK;
138:    VarType size__AWK;
139:    FnType *printline__AWK(int nArg,...);
140:    FnType *addword__AWK(int nArg,...);
141:    VarType i__AWK;
142:    VarType blanks__AWK;
```

```
143:  Regexp regExp_list[2];
144:
145:  void
146:  initVarList()
147:  {
148:      (void) installVar("line", &line_AWK);
149:      (void) installVar("dir", &dir_AWK);
150:      (void) installVar("NR", &NR_AWK);
151:      (void) installVar("RLENGTH", &RLENGTH_AWK);
152:      (void) installVar("RSTART", &RSTART_AWK);
153:      (void) installVar("RS", &RS_AWK);
154:      (void) installVar("cnt", &cnt_AWK);
155:      (void) installVar("ENVIRON", &ENVIRON_AWK);
156:      (void) installVar("FNR", &FNR_AWK);
157:      (void) installVar("size", &size_AWK);
158:      (void) installVar("OFS", &OFS_AWK);
159:      (void) installVar("ARGC", &ARGC_AWK);
160:      (void) installVar("ORS", &ORS_AWK);
161:      (void) installVar("ARGV", &ARGV_AWK);
162:      (void) installVar("SUBSEP", &SUBSEP_AWK);
163:      (void) installVar("NF", &NF_AWK);
164:      (void) installVar("OFMT", &OFMT_AWK);
165:      (void) installVar("FS", &FS_AWK);
166:      (void) installVar("FILENAME", &FILENAME_AWK);
167:      (void) installVar("i", &i_AWK);
168:      (void) installVar("blanks", &blanks_AWK);
169:  }
170:
171:  void
172:  main(argc, argv)
173:       int argc;
174:       char **argv;
175:  {
176:      int getlineFlag;
177:
178:      initVar(argc, argv);
```

```
179:      switch (setjmp(sjbuf))
180:      {
181:      case JMP_exitMark:
182:          goto exitMark;
183:      case JMP_ENDaction:
184:          goto ENDaction;
185:      }
186:      {
187:          strAssign(&blanks__AWK, A_STRP, aSprintf(A_STR, "%60s", A_STR, " "));
188:      }
189:      while ((getlineFlag = dfGetline(NULL)) == 1)
190:      {
191:          if (isTrue(A_NUM, (AWKFLOAT) matchop(A_STR, varSval(field), A_STR, regExp_0, 0)))
192:          {
193:              for (numAssign(&i__AWK, (AWKFLOAT) 1); isTrue(A_NUM, relopVV(REL_LE, &i__AWK,
    &NF__AWK)); postinc(&i__AWK))
194:                  freeFnType(addword__AWK(1, A_VAR, faddr(varIval(&i__AWK))));
195:          }
196:          if (isTrue(A_NUM, (AWKFLOAT) matchop(A_STR, varSval(field), A_STR, regExp_1, 1)))
197:          {
198:              freeFnType(printline__AWK(1, A_STR, "no"));
199:              print(A_STR, NULL, OUT_DEFAULT, 1, A_STR, "");
200:          }
201:        nextMark:;
202:      }
203:      if (getlineFlag == -1)
204:      {
205:          char err[80];
206:          sprintf(err, "error reading the input file '%s'", varSval(&FILENAME__AWK));
207:          fatal(err);
208:      }
209:    ENDaction:
210:      inENDaction = TRUE;
211:      {
212:          freeFnType(printline__AWK(1, A_STR, "no"));
213:      }
```

```
214:    exitMark:
215:      cleanup();
216:      exit(exitStatus);
217:  }
218:
219:
220:  FnType *
221:  addword__AWK(int nArg,...)
222:  {
223:      va_list arg;
224:      int i, type;
225:      FnType *retVal = (FnType *) xmalloc(sizeof(FnType));
226:      VarType w__AWK =
227:      {V_UNDEF, 0.0, NULL, NULL};
228:      VarType *lvarAddr[1] =
229:      {&w__AWK};
230:      VarType *parmAddr[1] =
231:      {NULL};
232:
233:      va_start(arg, nArg);
234:      for (i = 0; i < nArg; ++i)
235:      {
236:          switch (type = va_arg(arg, int))
237:          {
238:          case A_NUM:
239:              numAssign(lvarAddr[i], va_arg(arg, AWKFLOAT));
240:              break;
241:          case A_STR:
242:          case A_STRF:
243:              strAssign(lvarAddr[i], type, va_arg(arg, char *));
244:              break;
245:          case A_VAR:
246:              parmAddr[i] = va_arg(arg, VarType *);
247:              varAssign(lvarAddr[i], parmAddr[i]);
248:              break;
249:          case A_FUNC:
```

```
250:                fnAssign(lvarAddr[i], va_arg(arg, FnType *));
251:                break;
252:            default:
253:                fatal("invalid parameter type");
254:            }
255:        }
256:        va_end(arg);
257:        if (isTrue(A_NUM, relopCC(REL_GT, A_NUM, A_NUM, ((varNval(&cnt__AWK)) +
    (varNval(&size__AWK))) + (length(A_STR, varSval(&w__AWK))), (AWKFLOAT) 60)))
258:            freeFnType(printline__AWK(1, A_STR, "yes"));
259:        varAssign(array(&line__AWK, A_STRF, string(preinc(&cnt__AWK))), &w__AWK);
260:        numAssign(&size__AWK, (varNval(&size__AWK)) + (length(A_STR, varSval(&w__AWK))));
261:        {
262:            retVal->type = A_NUM;
263:            renewVar(&w__AWK);
264:            for (i = 0; i < nArg; ++i)
265:                if (lvarAddr[i]->aval)
266:                    if (parmAddr[i])
267:                        parmAddr[i]->aval = lvarAddr[i]->aval;
268:                    else
269:                        freeArr(lvarAddr[i]->aval);
270:            for (i = nArg; i < 1; ++i)
271:                freeArr(lvarAddr[i]->aval);
272:            return retVal;
273:        }
274:    }
275:
276:    FnType *
277:    printline__AWK(int nArg,...)
278:    {
279:        va_list arg;
280:        int i, type;
281:        FnType *retVal = (FnType *) xmalloc(sizeof(FnType));
282:        VarType f__AWK =
283:        {V_UNDEF, 0.0, NULL, NULL};
284:        VarType i__AWK =
```

```
285:        {V_UNDEF, 0.0, NULL, NULL};
286:        VarType nb__AWK =
287:        {V_UNDEF, 0.0, NULL, NULL};
288:        VarType nsp__AWK =
289:        {V_UNDEF, 0.0, NULL, NULL};
290:        VarType holes__AWK =
291:        {V_UNDEF, 0.0, NULL, NULL};
292:        VarType *lvarAddr[5] =
293:        {&f__AWK, &i__AWK, &nb__AWK, &nsp__AWK, &holes__AWK};
294:        VarType *parmAddr[5] =
295:        {NULL, NULL, NULL, NULL, NULL};
296:
297:        va_start(arg, nArg);
298:        for (i = 0; i < nArg; ++i)
299:        {
300:            switch (type = va_arg(arg, int))
301:            {
302:            case A_NUM:
303:                numAssign(lvarAddr[i], va_arg(arg, AWKFLOAT));
304:                break;
305:            case A_STR:
306:            case A_STRF:
307:                strAssign(lvarAddr[i], type, va_arg(arg, char *));
308:                break;
309:            case A_VAR:
310:                parmAddr[i] = va_arg(arg, VarType *);
311:                varAssign(lvarAddr[i], parmAddr[i]);
312:                break;
313:            case A_FUNC:
314:                fnAssign(lvarAddr[i], va_arg(arg, FnType *));
315:                break;
316:            default:
317:                fatal("invalid parameter type");
318:            }
319:        }
320:        va_end(arg);
```

```
321:     if (isTrue(A_NUM, (AWKFLOAT) (isTrue(A_NUM, relopVC(REL_EQ, &f__AWK, A_STR, "no")) ||
     isTrue(A_NUM, relopVC(REL_EQ, &cnt__AWK, A_NUM, (AWKFLOAT) 1)))))
322:     {
323:         for (numAssign(&i__AWK, (AWKFLOAT) 1); isTrue(A_NUM, relopVV(REL_LE, &i__AWK,
     &cnt__AWK)); postinc(&i__AWK))
324:             aPrintf(A_STR, NULL, OUT_DEFAULT, A_STR, "%s%s", A_VAR, array(&line__AWK,
     A_STR, varSval(&i__AWK)), A_FUNC, isTrue(A_NUM, relopVV(REL_LT, &i__AWK, &cnt__AWK)) ?
     toFnType(A_STR, " ") : toFnType(A_STR, "\n"));
325:     }
326:     else if (isTrue(A_NUM, relopVC(REL_GT, &cnt__AWK, A_NUM, (AWKFLOAT) 1)))
327:     {
328:         numAssign(&dir__AWK, ((AWKFLOAT) 1) - (varNval(&dir__AWK)));
329:         numAssign(&nb__AWK, ((AWKFLOAT) 60) - (varNval(&size__AWK)));
330:         numAssign(&holes__AWK, (varNval(&cnt__AWK)) - ((AWKFLOAT) 1));
331:         for (numAssign(&i__AWK, (AWKFLOAT) 1); isTrue(A_NUM, relopVC(REL_GT, &holes__AWK,
     A_NUM, (AWKFLOAT) 0)); postinc(&i__AWK))
332:         {
333:             numAssign(&nsp__AWK, ((AWKFLOAT) (long) ((((varNval(&nb__AWK)) -
     (varNval(&dir__AWK)))) / (varNval(&holes__AWK)))) + (varNval(&dir__AWK)));
334:             aPrintf(A_STR, NULL, OUT_DEFAULT, A_STR, "%s%s", A_VAR, array(&line__AWK,
     A_STR, varSval(&i__AWK)), A_STRP, SubStr(3, A_STR, varSval(&blanks__AWK), (int) ((AWKFLOAT)
     1), varIval(&nsp__AWK)));
335:             numAssign(&nb__AWK, (varNval(&nb__AWK)) - (varNval(&nsp__AWK)));
336:             postdec(&holes__AWK);
337:         }
338:         print(A_STR, NULL, OUT_DEFAULT, 1, A_STR, varSval(array(&line__AWK, A_STR,
     varSval(&cnt__AWK))));
339:     }
340:     numAssign(&size__AWK, numAssign(&cnt__AWK, (AWKFLOAT) 0));
341:     {
342:         retVal->type = A_NUM;
343:         renewVar(&f__AWK);
344:         renewVar(&i__AWK);
345:         renewVar(&nb__AWK);
346:         renewVar(&nsp__AWK);
347:         renewVar(&holes__AWK);
```

```
348:            for (i = 0; i < nArg; ++i)
349:                if (lvarAddr[i]->aval)
350:                    if (parmAddr[i])
351:                        parmAddr[i]->aval = lvarAddr[i]->aval;
352:                    else
353:                        freeArr(lvarAddr[i]->aval);
354:            for (i = nArg; i < 5; ++i)
355:                freeArr(lvarAddr[i]->aval);
356:            return retVal;
357:        }
358:    }
```

BIOGRAPHY

Mr. Chalermsak Chatdorkmaiprai was born on December 5, 1960 at Amphoe Phranakorn, Bangkok and had his secondary-school education at Suankularb Vidhayalai School. He received a Bachelor of Science Degree in Physics from Chulalongkorn University in 1983 and continued his study towards a Master Degree in Computer Science at the same university in 1990. During his graduate study, he received a U.D.C. scholarship from the Ministry of University Affairs and also a scholarship from C&C Education Foundation. After his graduation, he will join the faculty staff at Department of Computer Engineering, Kasetsart University.