รายการอ้างอิง

ภาษาอังกฤษ

1. Boris Beizer, Software testing techniques. Van Nostrand Reinhold, 115 Fifth Avenue, New York. 10003, 1990 ISBM 0-442-20672-0
2. Edward Miller, and William Howden E., Softwqre testing & validation techniques. IEEE No. EHO 180-0, Library of congress No. 81-81431 Order No. 365, IEEE COMPUTER SOCIETY, ISBN 0-8186-0365-8
3. Roger C.F. Shaw, Case studies in systematic software development. Prentice Hall International (UK) Ltd., 66 WoodLane End, HemelHempstead. Hartfordshire HP2 4RG 1990. ISBN 0-13-116088
4. Seymour Lipschutz, Essential computer mathematics. McGraw-Hill, Inc., 1987. ISBN 0-07-099132-4
5. Herbert Schildt, Born to code in C, McGraw-Hill, 2600 Tenth-Street, Berkeley, California 94710, U.S.A. ISBN 0-07-881468-5
6. Hendrix, James E. A small C compiler, 2nd Edition, M & T Publishing, Inc., 501 Galveston Drive, Redwood City, CA 94063-4728, U.S.A. 1990, ISBN 0-13-814724-8
7. Allen I. Holub, Compiler design in C, Prentice-Hall International, Inc. Englewood Cliffs, New Jersey 07632, 1990, ISBN 0-13-155151-5

ภาคผนวก ก.

ตัวอย่างผลการทดลอง และผลลัพธ์ต่างๆ

MODULE CALL FREQUENCY REPORT

Total number of files: 1

TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C

FILE : ftype.c
Total number of module: 30

module name	source code line number	call frequency	percentage
main	38	1	0.00611
write_fp2	187	812	4.95999
skip_space	199	3818	23.32173
isspace_line	209	646	3.94600
handle_space_line	220	23	0.14049
iscomment	228	2869	17.52489
handle_c_comment	236	42	0.25655
is_preprocess_keywords	283	623	3.80551
handle_preprocess_key	296	11	0.06719
is_declare_keywords	308	2827	17.26834
handle_declare_key	325	65	0.39704
is_statement_keywords	339	2762	16.87130
get_next_line	356	681	4.15979
handle_asm_statement	368	0	0.00000
handle_label_statement	377	22	0.13438
handle_jump_statement	406	81	0.49478
handle_statement_if	414	52	0.31763
handle_statement_switch	430	5	0.03054
handle_statement_while	445	14	0.08552
handle_statement_do	461	1	0.00611
handle_statement_for	472	3	0.01833
handle_statement_else	488	1	0.00611
handle_statement_sizeof	498	0	0.00000
handle_string_literal	506	24	0.14660
find_char	535	169	1.03231
handle_single_quote	572	7	0.04276
handle_parenthesis	584	104	0.63527
check_module	628	515	3.14581
write_module	664	30	0.18325
write_dd_path	673	163	0.99566

COMPLETE DD_PATH COVERAGE REPORT

Number of files : 1
Total number of dd_path : 162

TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C

FILE : ftype.c
Total number of Module : 30

MODULE NAME : main

Source code line number : 38
Total number of dd_path : 53

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	42	0	0.00000
dd_path else condition	45	1	0.00035
if	46	0	0.00000
dd_path else condition	49	1	0.00035
if	53	0	0.00000
dd_path else condition	56	1	0.00035
if	57	0	0.00000
dd_path else condition	60	1	0.00035
while	64	646	0.22411
if	66	23	0.00798
dd_path else condition	69	623	0.21613
if	70	11	0.00382
dd_path else condition	73	612	0.21232
while	74	2869	0.99532
if	76	42	0.01457
dd_path else condition	80	2827	0.98075
if	81	65	0.02255
dd_path else condition	84	2762	0.95820
if	85	179	0.06210
if	88	15	0.00520
dd_path else condition	91	164	0.05690
dd_path else condition	99	2583	0.89610
if	101	1444	0.50096
if	104	1323	0.45898
case	106	0	0.00000
case	110	106	0.03677
case	114	1	0.00035

case	117	4	0.00139
case	118	18	0.00624
case	119	70	0.02428
case	122	71	0.02463
case	124	30	0.01041
case	126	35	0.01214
case	130	77	0.02671
if	131	5	0.00173
dd_path else condition	134	72	0.02498
case	136	96	0.03330
if	137	10	0.00347
dd_path else condition	140	86	0.02984
dd_path else condition	143	121	0.04198
dd_path else condition	146	1139	0.39515
if	148	13	0.00451
dd_path else condition	151	1126	0.39064
while	154	4510	1.56462
dd_path after while	160	1126	0.39064
if	161	515	0.17867
if	162	30	0.01041
dd_path else condition	164	485	0.16826
dd_path else condition	166	611	0.21197
if	167	611	0.21197
dd_path else condition	172	0	0.00000
dd_path after while	179	612	0.21232
dd_path after while	180	1	0.00035

MODULE NAME : write_fp2

Source code line number : 187
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	189	812	0.28170
dd_path else condition	195	0	0.00000

MODULE NAME : skip_space

Source code line number : 199
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
while	201	4377	1.51848
dd_path after while	205	3818	1.32455

MODULE NAME : isspace_line

Source code line number : 209
Total number of dd_path : 4

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	211	23	0.00798
dd_path else condition	213	623	0.21613
if	214	0	0.00000
dd_path else condition	216	623	0.21613

MODULE NAME : handle_space_line

Source code line number : 220
Total number of dd_path : 0

dd_path keyword	source code line number	traversal count	percentage of traversal total

MODULE NAME : iscomment

Source code line number : 228
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	230	42	0.01457
dd_path else condition	232	2827	0.98075

MODULE NAME : handle_c_comment

Source code line number : 236
Total number of dd_path : 14

dd_path keyword	source code line number	traversal count	percentage of traversal total
while	242	44	0.01526
while	243	1608	0.55785
dd_path after while	247	44	0.01526
case	251	43	0.01492
if	253	42	0.01457
dd_path else condition	258	1	0.00035
case	260	0	0.00000
if	262	0	0.00000
dd_path else condition	267	0	0.00000
case	269	1	0.00035

if	271	0	0.00000
dd_path else condition	273	1	0.00035
default	275	0	0.00000
dd_path after while	279	42	0.01457

MODULE NAME : is_preprocess_keywords

Source code line number : 283
Total number of dd_path : 4

dd_path keyword	source code line number	traversal count	percentage of traversal total
for	287	8055	2.79447
if	288	11	0.00382
dd_path else condition	291	8044	2.79065
dd_path after for	292	612	0.21232

MODULE NAME : handle_preprocess_key

Source code line number : 296
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	298	0	0.00000
dd_path else condition	300	11	0.00382

MODULE NAME : is_declare_keywords

Source code line number : 308
Total number of dd_path : 6

dd_path keyword	source code line number	traversal count	percentage of traversal total
for	313	69639	24.15940
if	314	65	0.02255
if	317	65	0.02255
dd_path else condition	319	0	0.00000
dd_path else condition	320	69574	24.13685
dd_path after for	321	2762	0.95820

MODULE NAME : handle_declare_key

Source code line number : 325
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total

if	332	11	0.00382
dd_path else condition	334	54	0.01873

MODULE NAME : is_statement_keywords

Source code line number : 339
Total number of dd_path : 6

dd_path keyword	source code line number	traversal count	percentage of traversal total
for	344	37499	13.00928
if	345	179	0.06210
if	348	179	0.06210
dd_path else condition	350	0	0.00000
dd_path else condition	351	37320	12.94718
dd_path after for	352	2583	0.89610

MODULE NAME : get_next_line

Source code line number : 356
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	359	680	0.23591
dd_path else condition	364	1	0.00035

MODULE NAME : handle_asm_statement

Source code line number : 368
Total number of dd_path : 0

dd_path keyword	source code line number	traversal count	percentage of traversal total

MODULE NAME : handle_label_statement

Source code line number : 377
Total number of dd_path : 8

dd_path keyword	source code line number	traversal count	percentage of traversal total
while	383	22	0.00763
while	384	185	0.06418
dd_path after while	388	22	0.00763
if	392	0	0.00000

if	394	0	0.00000
dd_path else condition	397	0	0.00000
dd_path else condition	399	22	0.00763
dd_path after while	403	0	0.00000

MODULE NAME : handle_jump_statement

Source code line number : 406
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	408	81	0.02810
dd_path else condition	410	0	0.00000

MODULE NAME : handle_statement_if

Source code line number : 414
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	417	52	0.01804
dd_path else condition	426	0	0.00000

MODULE NAME : handle_statement_switch

Source code line number : 430
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	433	5	0.00173
dd_path else condition	441	0	0.00000

MODULE NAME : handle_statement_while

Source code line number : 445
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	448	14	0.00486
dd_path else condition	457	0	0.00000

MODULE NAME : handle_statement_do

Source code line number : 461
Total number of dd_path : 0

dd_path keyword	source code line number	traversal count	percentage of traversal total

MODULE NAME : handle_statement_for

Source code line number : 472
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	475	3	0.00104
dd_path else condition	484	0	0.00000

MODULE NAME : handle_statement_else

Source code line number : 488
Total number of dd_path : 0

dd_path keyword	source code line number	traversal count	percentage of traversal total

MODULE NAME : handle_statement_sizeof

Source code line number : 498
Total number of dd_path : 0

dd_path keyword	source code line number	traversal count	percentage of traversal total

MODULE NAME : handle_string_literal

Source code line number : 506
Total number of dd_path : 6

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	510	24	0.00833
while	515	258	0.08951
if	518	21	0.00729
dd_path else condition	522	237	0.08222
dd_path after while	525	24	0.00833
dd_path else condition	532	0	0.00000

MODULE NAME : find_char

Source code line number : 535
Total number of dd_path : 11

dd_path keyword	source code line number	traversal count	percentage of traversal total
while	539	169	0.05863
while	540	516	0.17901
dd_path after while	544	169	0.05863
if	545	169	0.05863
else	550	169	0.05863
case	552	0	0.00000
case	555	0	0.00000
case	558	0	0.00000
if	562	0	0.00000
dd_path else condition	564	0	0.00000
dd_path after while	568	169	0.05863

MODULE NAME : handle_single_quote

Source code line number : 572
Total number of dd_path : 2

dd_path keyword	source code line number	traversal count	percentage of traversal total
do	574	17	0.00590
dd_path after do	578	7	0.00243

MODULE NAME : handle_parenthesis

Source code line number : 584
Total number of dd_path : 12

dd_path keyword	source code line number	traversal count	percentage of traversal total
while	591	220	0.07632
while	592	2080	0.72160
if	593	7	0.00243
dd_path else condition	596	2073	0.71917
dd_path after while	600	220	0.07632
case	602	51	0.01769
case	608	155	0.05377
case	614	11	0.00382
case	617	3	0.00104
if	619	0	0.00000
dd_path else condition	621	3	0.00104
dd_path after while	624	104	0.03608

MODULE NAME : check_module

Source code line number : 628
Total number of dd_path : 14

dd_path keyword	source code line number	traversal count	percentage of traversal total
if	630	63	0.02186
if	633	30	0.01041
while	635	90	0.03122
if	639	30	0.01041
dd_path else condition	642	60	0.02082
if	643	30	0.01041
if	646	0	0.00000
dd_path else condition	648	30	0.01041
dd_path else condition	650	30	0.01041
if	651	30	0.01041
dd_path else condition	656	0	0.00000
dd_path after while	658	0	0.00000
dd_path else condition	659	33	0.01145
dd_path else condition	660	452	0.15681

MODULE NAME : write_module

Source code line number : 664
Total number of dd_path : 0

dd_path keyword	source code line number	traversal count	percentage of traversal total

MODULE NAME : write_dd_path

Source code line number : 673
Total number of dd_path : 0

dd_path keyword	source code line number	traversal count	percentage of traversal total

UNTRAVERSAL DD_PATH COVERAGE REPORT

Number of files : 1
Total number of dd_path : 162

TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C TEST_C

FILE : ftype.c
Total Number of Module : 30

MODULE NAME: main

Source code line number: 38
Total number of dd_path: 53

dd_path keyword	source code line number	traversal count
if	42	0
if	46	0
if	53	0
if	57	0
case	106	0
dd_path else condition	172	0

MODULE NAME: write_fp2

Source code line number: 187
Total number of dd_path: 2

dd_path keyword	source code line number	traversal count
dd_path else condition	195	0

MODULE NAME: skip_space

Source code line number: 199
Total number of dd_path: 2

dd_path keyword	source code line number	traversal count

-=* No untraversal in this module *=-

MODULE NAME: isspace_line

```
-----------------------------------------------
Source code line number: 209
Total number of dd_path: 4
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------
if                           214                  0

MODULE NAME: handle_space_line
-----------------------------------------------
Source code line number: 220
Total number of dd_path: 0
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------
       -=*   No untraversal in this module   *=-


MODULE NAME: iscomment
-----------------------------------------------
Source code line number: 228
Total number of dd_path: 2
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------
       -=*   No untraversal in this module   *=-


MODULE NAME: handle_c_comment
-----------------------------------------------
Source code line number: 236
Total number of dd_path: 14
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------
case                         260                  0
if                           262                  0
dd_path else condition       267                  0
if                           271                  0
default                      275                  0

MODULE NAME: is_preprocess_keywords
-----------------------------------------------
Source code line number: 283
Total number of dd_path: 4
    dd_path              source code          traversal
    keyword              line number            count
```

```
-----------------------------------------------------------
      -=*   No untraversal in this module   *=-


MODULE NAME: handle_preprocess_key
-----------------------------------------------
Source code line number: 296
Total number of dd_path: 2
    dd_path              source code          traversal
    keyword              line number           count
-----------------------------------------------------------
if                            298                 0

MODULE NAME: is_declare_keywords
-----------------------------------------------
Source code line number: 308
Total number of dd_path: 6
    dd_path              source code          traversal
    keyword              line number           count
-----------------------------------------------------------
dd_path else condition        319                 0

MODULE NAME: handle_declare_key
-----------------------------------------------
Source code line number: 325
Total number of dd_path: 2
    dd_path              source code          traversal
    keyword              line number           count
-----------------------------------------------------------
      -=*   No untraversal in this module   *=-


MODULE NAME: is_statement_keywords
-----------------------------------------------
Source code line number: 339
Total number of dd_path: 6
    dd_path              source code          traversal
    keyword              line number           count
-----------------------------------------------------------
dd_path else condition        350                 0

MODULE NAME: get_next_line
-----------------------------------------------
Source code line number: 356
Total number of dd_path: 2
    dd_path              source code          traversal
    keyword              line number           count
```

```
-----------------------------------------------------
      -=*   No untraversal in this module   *=-

MODULE NAME: handle_asm_statement
-----------------------------------------------
Source code line number: 368
Total number of dd_path: 0
    dd_path              source code          traversal
    keyword              line number            count
-----------------------------------------------------
      -=*   No untraversal in this module   *=-

MODULE NAME: handle_label_statement
-----------------------------------------------
Source code line number: 377
Total number of dd_path: 8
    dd_path              source code          traversal
    keyword              line number            count
-----------------------------------------------------
if                          392                  0
if                          394                  0
dd_path else condition      397                  0
dd_path after while        403                  0

MODULE NAME: handle_jump_statement
-----------------------------------------------
Source code line number: 406
Total number of dd_path: 2
    dd_path              source code          traversal
    keyword              line number            count
-----------------------------------------------------
dd_path else condition      410                  0

MODULE NAME: handle_statement_if
-----------------------------------------------
Source code line number: 414
Total number of dd_path: 2
    dd_path              source code          traversal
    keyword              line number            count
-----------------------------------------------------
dd_path else condition      426                  0

MODULE NAME: handle_statement_switch
-----------------------------------------------
Source code line number: 430
```

```
Total number of dd_path: 2
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------
dd_path else condition        441                  0

MODULE NAME: handle_statement_while
-----------------------------------------------
Source code line number: 445
Total number of dd_path: 2
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------
dd_path else condition        457                  0

MODULE NAME: handle_statement_do
-----------------------------------------------
Source code line number: 461
Total number of dd_path: 0
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------
       -=*   No untraversal in this module   *=-


MODULE NAME: handle_statement_for
-----------------------------------------------
Source code line number: 472
Total number of dd_path: 2
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------
dd_path else condition        484                  0

MODULE NAME: handle_statement_else
-----------------------------------------------
Source code line number: 488
Total number of dd_path: 0
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------
       -=*   No untraversal in this module   *=-


MODULE NAME: handle_statement_sizeof
-----------------------------------------------
Source code line number: 498
```

```
Total number of dd_path: 0
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------------
       -=*   No untraversal in this module   *=-


MODULE NAME: handle_string_literal
-----------------------------------------------
Source code line number: 506
Total number of dd_path: 6
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------------
dd_path else condition        532                  0

MODULE NAME: find_char
-----------------------------------------------
Source code line number: 535
Total number of dd_path: 11
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------------
case                          552                  0
case                          555                  0
case                          558                  0
if                            562                  0
dd_path else condition        564                  0

MODULE NAME: handle_single_quote
-----------------------------------------------
Source code line number: 572
Total number of dd_path: 2
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------------
       -=*   No untraversal in this module   *=-


MODULE NAME: handle_parenthesis
-----------------------------------------------
Source code line number: 584
Total number of dd_path: 12
    dd_path              source code          traversal
    keyword              line number            count
-------------------------------------------------------------
if                            619                  0
```

```
MODULE NAME: check_module
-------------------------------------------------
Source code line number: 628
Total number of dd_path: 14
    dd_path                source code          traversal
    keyword                line number            count
-----------------------------------------------------------
if                              646                  0
dd_path else condition          656                  0
dd_path after while             658                  0

MODULE NAME: write_module
-------------------------------------------------
Source code line number: 664
Total number of dd_path: 0
    dd_path                source code          traversal
    keyword                line number            count
-----------------------------------------------------------
       -=*   No untraversal in this module   *=-


MODULE NAME: write_dd_path
-------------------------------------------------
Source code line number: 673
Total number of dd_path: 0
    dd_path                source code          traversal
    keyword                line number            count
-----------------------------------------------------------
       -=*   No untraversal in this module   *=-
```

```
#include <alloc.h>
#include <stdio.h>
#define  TRUE   1
#define  FALSE  0
struct database_type {
     int file_count;
     int module_total;
     int dd_path_total;
     struct file_type *file_list;
};
struct file_type {
     char *name;
     int name_len;
     int num_of_module;
     struct module_type *module_list;
     struct file_type *next;
};
struct module_type {
     char *name;
     int name_len;
     int line_number;
     int num_dd_path;
     long call_frequency;
     struct dd_path_type *dd_path_list;
     struct module_type *next;
};
struct dd_path_type {
     char *keyword;
     int keyword_len;
     int line_number;
     long traversal_freq;
     struct dd_path_type *next;
};
struct database_type HEAD_DATA;

int read_DB_file( void )
{
   FILE *fp1;
   struct file_type    *ft_temp,  *ft_temp2;
   struct module_type  *mt_temp,  *mt_temp2;
   struct dd_path_type *dpt_temp, *dpt_temp2;
   int    i, j, k, ft_len, mt_len, dpt_len;

   if ( (fp1 = fopen("TEST_C.DBF", "rb")) == NULL) {
      printf("Can't open file : TEST_C.DBF");
      return(FALSE);
   }
```

```
ft_len  = sizeof(struct file_type);
mt_len  = sizeof(struct module_type);
dpt_len = sizeof(struct dd_path_type);
if (fread(&HEAD_DATA, sizeof(struct database_type), 1, fp1) == 0) {
   printf("Can't read file : TEST_C.DBF");
   fclose(fp1);
   return(FALSE);
}
for ( i = 0; i < HEAD_DATA.file_count; i++ ) {
   if ( (ft_temp = malloc(ft_len)) == NULL ) {
      printf("Can't allocate memory");
      fclose(fp1);
      return(FALSE);
   }
   if ( fread(ft_temp, ft_len, 1, fp1) == 0 ) {
      printf("Error structure on TEST_C.DBF");
      fclose(fp1);
      return(FALSE);
   }
   ft_temp->name = malloc(ft_temp->name_len + 1);
   fread(ft_temp->name, ft_temp->name_len + 1, 1, fp1);
   ft_temp->next = NULL;
   if ( i == 0 ) {
      HEAD_DATA.file_list = ft_temp;
   } else {
      ft_temp2->next = ft_temp;
   }
   ft_temp2 = ft_temp;
   for ( j = 0; j < ft_temp2->num_of_module; j++ ) {
      if ( (mt_temp = malloc(mt_len)) == NULL ) {
         printf("Can't allocate memory");
         fclose(fp1);
         return(FALSE);
      }
      if ( fread(mt_temp, mt_len, 1, fp1) == 0 ) {
         printf("Error structure file: TEST_C.DBF");
         fclose(fp1);
         return(FALSE);
      }
      mt_temp->name = malloc(mt_temp->name_len + 1);
      fread(mt_temp->name, mt_temp->name_len+1, 1, fp1);
      mt_temp->next = NULL;
      if ( j == 0 ) {
         ft_temp2->module_list = mt_temp;
      } else {
         mt_temp2->next = mt_temp;
      }
```

```
            mt_temp2 = mt_temp;
            for ( k = 0; k < mt_temp2->num_dd_path; k++ ) {
               if ( (dpt_temp = malloc(dpt_len)) == NULL ) {
                  printf("Can't allocate memory");
                  fclose(fp1);
                  return(FALSE);
               }
               if ( fread(dpt_temp, dpt_len, 1, fp1) == 0 ) {
                  printf("Error structure file: TEST_C.DBF");
                  fclose(fp1);
                  return(FALSE);
               }
               dpt_temp->keyword = malloc(dpt_temp->keyword_len + 1);
               fread(dpt_temp->keyword,dpt_temp->keyword_len+1,1,fp1);
               dpt_temp->next = NULL;
               if ( k == 0 ) {
                  mt_temp2->dd_path_list = dpt_temp;
               } else {
                  dpt_temp2->next = dpt_temp;
               }
               dpt_temp2 = dpt_temp;
            }
         }
      }
      fclose(fp1);
      return(TRUE);
   }

   int write_DB_file( void )
   {
      FILE *fp1;
      struct file_type    *ft_temp;


      struct module_type  *mt_temp;
      struct dd_path_type *dpt_temp;
      int    i, j, k, ft_len, mt_len, dpt_len;

      if ( (fp1 = fopen("TEST_C.DBF", "wb")) == NULL) {
         printf("Can't open file : TEST_C.DBF");
         return(FALSE);
      }

      ft_len   = sizeof(struct file_type);
      mt_len   = sizeof(struct module_type);
      dpt_len  = sizeof(struct dd_path_type);
      if (fwrite(&HEAD_DATA,sizeof(struct database_type),1,fp1) == 0) {
```

```
            printf("Can't write file : TEST_C.DBF");
            fclose(fp1);
            return(FALSE);
         }
         ft_temp = HEAD_DATA.file_list;
         for ( i = 0; i < HEAD_DATA.file_count; i++ ) {
            if ( fwrite(ft_temp, ft_len, 1, fp1) == 0 ) {
               printf("Error write structure on TEST_C.DBF");
               fclose(fp1);
               return(FALSE);
            }
            fwrite(ft_temp->name, ft_temp->name_len + 1, 1, fp1);
            mt_temp = ft_temp->module_list;
            for ( j = 0; j < ft_temp->num_of_module; j++ ) {
               if ( fwrite(mt_temp, mt_len, 1, fp1) == 0 ) {
                  printf("Error structure file: TEST_C.DBF");
                  fclose(fp1);
                  return(FALSE);
               }
               fwrite(mt_temp->name, mt_temp->name_len + 1, 1, fp1);
               dpt_temp = mt_temp->dd_path_list;
               for ( k = 0; k < mt_temp->num_dd_path; k++ ) {
                  if ( fwrite(dpt_temp, dpt_len, 1, fp1) == 0 ) {
                     printf("Error structure file: TEST_C.DBF");
                     fclose(fp1);
                     return(FALSE);
                  }
                  fwrite(dpt_temp->keyword, dpt_temp->keyword_len + 1, 1, fp1);
                  dpt_temp = dpt_temp->next;
               }
               mt_temp = mt_temp->next;
            }
            ft_temp = ft_temp->next;
         }
         fclose(fp1);
         clear_HEAD_DATA();
         return(TRUE);
}

int count_module(int f_th, int m_th)
{
         struct file_type   *ft_temp;
         struct module_type *mt_temp;

         if (HEAD_DATA.file_list == NULL) {
            if ( read_DB_file() == FALSE ) {
               printf("ERROR : database file "); getch();
```

```
            return (FALSE);
         }
      }
      ft_temp = HEAD_DATA.file_list;
      while ((f_th > 1) && (ft_temp != NULL)) {
         ft_temp = ft_temp->next;
         f_th--;
      }
      if ((ft_temp != NULL) && (ft_temp->module_list != NULL)) {
         mt_temp = ft_temp->module_list;
         while ((m_th > 1) && (mt_temp != NULL)) {
            mt_temp = mt_temp->next;
            m_th--;
         }
         if (mt_temp != NULL) {
            mt_temp->call_frequency++;
            return(TRUE);
         }
      }
}

int count_DD_Path(int f_th, int m_th, int ddp_th)
{
   struct file_type    *ft_temp;
   struct module_type  *mt_temp;
   struct dd_path_type *dpt_temp;

   if (HEAD_DATA.file_list == NULL) {
      if ( read_DB_file() == FALSE ) {
         printf("ERROR : database file "); getch();
         return (FALSE);
      }
   }
   ft_temp = HEAD_DATA.file_list;
   while ((f_th > 1) && (ft_temp != NULL)){
      ft_temp = ft_temp->next;
      f_th--;
   }
   if ((ft_temp != NULL) && (ft_temp->module_list != NULL)) {
      mt_temp = ft_temp->module_list;
      while ((m_th > 1) && (mt_temp != NULL)) {
         mt_temp = mt_temp->next;
         m_th--;
      }
      if ((mt_temp != NULL) && (mt_temp->dd_path_list != NULL)) {
         dpt_temp = mt_temp->dd_path_list;
         while ((ddp_th > 1) && (dpt_temp != NULL)) {
```

```
            dpt_temp = dpt_temp->next;
            ddp_th--;
         }
         if (dpt_temp != NULL) {
            dpt_temp->traversal_freq++;
            return(TRUE);
         }
      }
   }
}

int clear_HEAD_DATA(void)
{
   struct file_type    *ft_temp1,  *ft_temp2;
   struct module_type  *mt_temp1,  *mt_temp2;
   struct dd_path_type *dpt_temp1, *dpt_temp2;

   ft_temp1 = HEAD_DATA.file_list;
   while (ft_temp1 != NULL) {
      ft_temp2 = ft_temp1->next;
      mt_temp1 = ft_temp1->module_list;
      while (mt_temp1 != NULL) {
         mt_temp2 = mt_temp1->next;
         dpt_temp1 = mt_temp1->dd_path_list;
         while (dpt_temp1 != NULL) {
            dpt_temp2 = dpt_temp1->next;
            free(dpt_temp1->keyword);
            free(dpt_temp1);
            dpt_temp1 = dpt_temp2;
         }
         free(mt_temp1->name);
         free(mt_temp1);
         mt_temp1 = mt_temp2;
      }
      free(ft_temp1->name);
      free(ft_temp1);
      ft_temp1 = ft_temp2;
   }
   HEAD_DATA.file_count    = 0;
   HEAD_DATA.module_total  = 0;
   HEAD_DATA.dd_path_total = 0;
   HEAD_DATA.file_list     = NULL;
}
```

```
int insert_probe                ( char *file_name );
int write_fp2                   ( void );
void skip_space                 ( void );
int isspace_line                ( char *str_temp );
int handle_space_line           ( void );
int iscomment                   ( char *str_temp );
int handle_c_comment            ( void );
int is_preprocess_keywords      ( unsigned char *str_temp );
int handle_preprocess_key       ( int key_no );
int is_declare_keywords         ( unsigned char *str_temp );
int handle_declare_key          ( int key_no );
int is_statement_keywords       ( unsigned char *string_buf );
int get_next_line               ( void );
int handle_asm_statement        ( void );
int handle_label_statement      ( void );
int handle_jump_statement       ( void );
int handle_statement_if         ( void );
int handle_statement_switch     ( void );
int handle_statement_while      ( void );
int handle_statement_do         ( void );
int handle_statement_for        ( void );
int handle_statement_else       ( void );
int handle_statement_sizeof     ( void );
int handle_string_literal       ( void );
int find_char                   ( char c );
int handle_single_quote         ( void );
int handle_parenthesis          ( void );
int check_module                ( void );

int preprocess_keywords_flag = FALSE;
unsigned char *preprocess_keywords[] =
   {  "#define\0",  "#elif\0",   "#else\0",    "#endif\0",
      "#error\0",   "#ifdef\0",  "#ifndef\0",  "#if\0",
      "#include\0", "#line\0",   "#pragma\0",  "#undef\0",
      "#\0"
   };
enum {   P_DEFINE,  P_ELIF,  P_ELSE,   P_ENDIF,
         P_ERROR,   P_IFDEF, P_IFNDEM, P_IF,
         P_INCLUDE, P_LINE,  P_PRAGMA, P_UNDEF,
         P_PRE,     MAX_PREPROCESS_KEYWORDS
 };

int declare_keywords_flag = FALSE;
unsigned char *declare_keywords[] =
   {  "typedef\0",  "extern\0",    "static\0",   "auto\0",
      "register\0", "void\0",      "char\0",     "short\0",
      "int\0",      "long\0",      "float\0",    "double\0",
```

```
      "signed\0",   "unsigned\0",  "const\0",    "volatile\0",
      "struct\0",   "union\0",     "enum\0",     "cdecl\0",
      "pascal\0",   "interrupt\0", "near\0",     "far\0",
      "huge\0"
   };
enum { TYPEDEF,  EXTERN,    STATIC,  AUTO,
       REGISTER, VOID,      CHAR,    SHORT,
       INT,      LONG,      FLOAT,   DOUBLE,
       SIGNED,   UNSIGNED,  CONST,   VOLATILE,
       STRUCT,   UNION,     ENUM,    CDECL,
       PASCAL,   INTERRUPT, NEAR,    FAR,
       HUGE,     MAX_DECLARE_KEYWORDS
 };

unsigned char *statement_keywords[] =
   {  "asm\0",    "case\0",     "default\0", "if\0",
      "switch\0", "while\0",    "do\0",      "for\0",
      "goto\0",   "continue\0", "break\0",   "return\0",
      "else\0",   "sizeof\0"
   };
enum {   ASM,    CASE,     DEFAULT, IF,
         SWITCH, WHILE,    DO,      FOR,
         GOTO,   CONTINUE, BREAK,   RETURN,
         ELSE,   SIZEOF,   MAX_STATEMENT_KEYWORDS,
         MODULE
 };
int (*handle_statement_keywords[])( void ) =
   { handle_asm_statement,       /* asm      */
     handle_label_statement,     /* case     */
     handle_label_statement,     /* default  */
     handle_statement_if,        /* if       */
     handle_statement_switch,    /* switch   */
     handle_statement_while,     /* while    */
     handle_statement_do,        /* do       */
     handle_statement_for,       /* for      */
     handle_jump_statement,      /* goto     */
     handle_jump_statement,      /* continue */
     handle_jump_statement,      /* break    */
     handle_jump_statement,      /* return   */
     handle_statement_else,      /* else     */
     handle_statement_sizeof,    /* sizeof   */
   };
```

```
#include <io.h>
#include <dir.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <alloc.h>
#include <stdlib.h>

#include "define.h"
#include "dbase.h"
#include "global.h"
#include "ftype.h"

int line_of_module;
int probe_module  = FALSE;
char module_name[ MAX_S ];

int line_number  = 0;
unsigned long sum = 0;

int nest_brace_p = 0;
int nest_brace[ MAX_S ];

int statement_key_no;
int statement_line_no;
unsigned char variable_name[256];
unsigned char string_literal[256];
unsigned char file_tcc[256];
unsigned char file_mdc[256];
unsigned char *buf_temp;

char drive[MAXDRIVE];
char dir[MAXDIR];
char file[MAXFILE];
char ext[MAXEXT];

/*-------------------------------------------*/
int main( int argc, char *argv[] )
{
   int key_no, n, ctemp;


read_DB_file();

count_module(1, 1);
   if (argc < 2) {
count_DD_Path(1, 1, 1);
```

```
      printf(" use -> test filename [p:pause]\n");

write_DB_file();
      return( FALSE );
   }

else {
count_DD_Path(1, 1, 2);
}
if ((fp1 = fopen(argv[1], "rb")) == NULL) {
count_DD_Path(1, 1, 3);

      printf("Can't open file : %s\n", argv[1]);

write_DB_file();
      return( FALSE );
   }

else {
count_DD_Path(1, 1, 4);
}
n = fnsplit( argv[1], drive, dir, file, ext );
   fnmerge( file_tcc, drive, dir, file, ".TCC" );
   fnmerge( file_mdc, drive, dir, file, ".MDC" );
   if ((fp2 = fopen( file_tcc, "wb")) == NULL) {
count_DD_Path(1, 1, 5);

      printf("Can't open temp file : %s\n", file_tcc );

write_DB_file();
      return( FALSE );
   }

else {
count_DD_Path(1, 1, 6);
}
if ((fp3 = fopen( file_mdc, "wb")) == NULL) {
count_DD_Path(1, 1, 7);

      printf("Can't open temp file : %s\n", file_mdc );

write_DB_file();
      return( FALSE );
   }


else {
```

```
count_DD_Path(1, 1, 8);
}
line_number = line_of_module = statement_line_no = 0;

   while (get_next_line() > 0) {
count_DD_Path(1, 1, 9);

  skip_space();
      if ( isspace_line(buf_temp) == TRUE ) {
count_DD_Path(1, 1, 10);

         handle_space_line();
         continue;
      }

else {
count_DD_Path(1, 1, 11);
}
if ((key_no = is_preprocess_keywords(buf_temp)) > -1) {
count_DD_Path(1, 1, 12);

 handle_preprocess_key( key_no );
         continue;
      }

else {
count_DD_Path(1, 1, 13);
}
while ( *buf_temp != NULL ) {
count_DD_Path(1, 1, 14);

 skip_space();
 if ( iscomment(buf_temp) == TRUE ) {
count_DD_Path(1, 1, 15);

buf_temp++; buf_temp++;
            handle_c_comment();
            continue;
 }

else {
count_DD_Path(1, 1, 16);
}
if ( (key_no = is_declare_keywords(buf_temp)) > -1 ) {
count_DD_Path(1, 1, 17);

            handle_declare_key( key_no );
```

```
            continue;
          }

else {
count_DD_Path(1, 1, 18);
}
if ((key_no = is_statement_keywords(buf_temp)) > -1) {
count_DD_Path(1, 1, 19);

            statement_key_no  = key_no;
statement_line_no = line_number;
            if ( (nest_brace_p > 0) && (probe_module == FALSE) ) {
count_DD_Path(1, 1, 20);

               write_module( fp3 );
               probe_module = TRUE;
            }

else {
count_DD_Path(1, 1, 21);
}
buf_fp2[ buf_fp2_p ] = NULL;
            strcat( buf_fp2, statement_keywords[key_no] );
            buf_temp  += strlen( statement_keywords[key_no] );
            buf_fp2_p += strlen( statement_keywords[key_no] );

            handle_statement_keywords[key_no]();
            continue;
 }

else {
count_DD_Path(1, 1, 22);
}
/* is delimiter */
         if ( strchr("[](){}*,:=;#\r", *buf_temp) ) {
count_DD_Path(1, 1, 23);

            buf_fp2[ buf_fp2_p ] = *buf_temp;
            buf_fp2_p++;
            if ( nest_brace_p > 0 ) {
count_DD_Path(1, 1, 24);

               switch (*buf_temp) {
                  case OPEN_BRACE:
count_DD_Path(1, 1, 25);

                     printf(" error: open_brace");
```

```
                    getch();

write_DB_file();
                    return( FALSE );
                 case CLOSE_BRACE:
count_DD_Path(1, 1, 26);

                    nest_brace_p--;
                    ctemp = nest_brace[ nest_brace_p ];
                    switch( ctemp ) {
                       case DO    :
count_DD_Path(1, 1, 27);

                          buf_fp2_p--;
                          find_char(SEMI_COLON);
                       case FOR   :
count_DD_Path(1, 1, 28);

                       case WHILE:
count_DD_Path(1, 1, 29);

                       case IF    :
count_DD_Path(1, 1, 30);

                          write_fp2();
                          write_dd_path( fp3 );
                       case ELSE  :
count_DD_Path(1, 1, 31);

                          break;
                       case MODULE:
count_DD_Path(1, 1, 32);

                          probe_module = FALSE;
                       case SWITCH:
count_DD_Path(1, 1, 33);

                          break;
                    }
                    break;
                 case EQUAL :
count_DD_Path(1, 1, 34);

                    if ( probe_module == FALSE ) {
count_DD_Path(1, 1, 35);

                       write_module( fp3 );
```

```
                        probe_module = TRUE;
                    }

else {
count_DD_Path(1, 1, 36);
}
break;
                    case OPEN_PAREN :
count_DD_Path(1, 1, 37);

 if ( probe_module == FALSE ) {
count_DD_Path(1, 1, 38);

                        write_module( fp3 );
                        probe_module = TRUE;
 }

else {
count_DD_Path(1, 1, 39);
}
break;
                }
            }

else {
count_DD_Path(1, 1, 40);
}
buf_temp++;
            continue;
        }

else {
count_DD_Path(1, 1, 41);
}
/* is string_literal */
 if ( *buf_temp == '"' ) {
count_DD_Path(1, 1, 42);

            handle_string_literal();
            continue;
 }

else {
count_DD_Path(1, 1, 43);
}
/* assume it is variable */
 n = 0; /* temp ..................... */
```

```
 while ( !strchr(" [](){}*,:=;#\r\n", *buf_temp) ) {
count_DD_Path(1, 1, 44);

variable_name[n] = *buf_temp; n++;
variable_name[n] = NULL;
            buf_fp2[ buf_fp2_p ] = *buf_temp;
            buf_fp2_p++;
            buf_temp++;
 }
count_DD_Path(1, 1, 45);

         if ( n > 0 ) {
count_DD_Path(1, 1, 46);

if ( check_module() == TRUE ) {
count_DD_Path(1, 1, 47);

                strcpy( module_name, variable_name );
}

else {
count_DD_Path(1, 1, 48);
}
continue;
 }

else {
count_DD_Path(1, 1, 49);
}
if ( *buf_temp == NEWLINE ) {
count_DD_Path(1, 1, 50);

            buf_fp2[ buf_fp2_p ] = NEWLINE;
            buf_fp2_p++;
            write_fp2();
            break;
         }


else {
count_DD_Path(1, 1, 51);
}
/* skip DELIMITER */    /* this is use temp */
         buf_fp2[ buf_fp2_p ] = *buf_temp;
         buf_fp2_p++;
         buf_temp++;
```

```
      }
count_DD_Path(1, 1, 52);

   }
count_DD_Path(1, 1, 53);

   fclose(fp1);
   fclose(fp2);
   fclose(fp3);

write_DB_file();
}

/*-----------------------------------------*/
int write_fp2 ( void )
{

count_module(1, 2);
   if ( buf_fp2_p > 0 ) {
count_DD_Path(1, 2, 1);

      buf_fp2[ buf_fp2_p ] = NULL;
  fprintf( fp2, "%s", buf_fp2 );
      buf_fp2_p  = 0;
      buf_fp2[ buf_fp2_p ] = NULL;
      return ( TRUE );
   }

else {
count_DD_Path(1, 2, 2);
}
return ( FALSE );
}
/*-----------------------------------------*/
void skip_space(void)
{  /* skip space in *buf_temp and copy white space to buf_fp2 */

count_module(1, 3);
   while ((*buf_temp == SPACE) || (*buf_temp == TAB)) {
count_DD_Path(1, 3, 1);

      buf_fp2[ buf_fp2_p ] = *buf_temp;
      buf_fp2_p++;
      buf_temp++;
   }
count_DD_Path(1, 3, 2);
```

```
   buf_fp2[ buf_fp2_p ] = NULL;
}
/*-------------------------------------------*/
int isspace_line( char *str_temp )
{

count_module(1, 4);
   if ( (*str_temp == CR) && (*(str_temp+1) == NEWLINE) ) {
count_DD_Path(1, 4, 1);

      return( TRUE );
   }

else {
count_DD_Path(1, 4, 2);
}
if ( (*str_temp == NEWLINE) && (*(str_temp+1) == CR) ) {
count_DD_Path(1, 4, 3);

      return( TRUE );
   }

else {
count_DD_Path(1, 4, 4);
}
return( FALSE );
}
/*-------------------------------------------*/
int handle_space_line( void )
{

count_module(1, 5);
   fprintf( fp2, "%s", buf_fp1 );
   buf_fp2_p = 0;
   buf_fp2[ buf_fp2_p ] = NULL;
   return( TRUE );
}
/*-------------------------------------------*/
int iscomment( char *str_temp )
{

count_module(1, 6);
   if ( (*str_temp == SLASH) && (*(str_temp+1) == STAR) ) {
count_DD_Path(1, 6, 1);

      return( TRUE );
   }
```

```
else {
count_DD_Path(1, 6, 2);
}
return( FALSE );
}
/*-----------------------------------------*/
int handle_c_comment( void )
{
   int m, n = 1;


count_module(1, 7);
   buf_fp2[ buf_fp2_p ] = '/';  buf_fp2_p++;
   buf_fp2[ buf_fp2_p ] = '*';  buf_fp2_p++;
   while ( n > 0 ) {
count_DD_Path(1, 7, 1);

      while ( !strchr("*/\n", *buf_temp) ) {
count_DD_Path(1, 7, 2);

         buf_fp2[ buf_fp2_p ] = *buf_temp;
         buf_fp2_p++;
         buf_temp++;
      }
count_DD_Path(1, 7, 3);

      buf_fp2[ buf_fp2_p ] = *buf_temp;
      buf_fp2_p++;
      switch( *buf_temp ) {
         case STAR :
count_DD_Path(1, 7, 4);

            buf_temp++;
            if ( *buf_temp == SLASH ) {
count_DD_Path(1, 7, 5);

               buf_fp2[ buf_fp2_p ] = *buf_temp;
               buf_fp2_p++;
               buf_temp++;
               n--;
            }

else {
count_DD_Path(1, 7, 6);
}
break;
         case SLASH:
```

```
count_DD_Path(1, 7, 7);

            buf_temp++;
            if ( *buf_temp == STAR ) {
count_DD_Path(1, 7, 8);

               buf_fp2[ buf_fp2_p ] = *buf_temp;
               buf_fp2_p++;
               *buf_temp++;
               n++;
            }

else {
count_DD_Path(1, 7, 9);
}
break;
         case NEWLINE:
count_DD_Path(1, 7, 10);

            write_fp2();
            if ( get_next_line() < 1 ) {
count_DD_Path(1, 7, 11);

               return( FALSE );
            }

else {
count_DD_Path(1, 7, 12);
}
break;
         default:
count_DD_Path(1, 7, 13);

            /* test default case */
            break;
      }
   }
count_DD_Path(1, 7, 14);

   return( TRUE );
}
/*-------------------------------------------*/
int is_preprocess_keywords(unsigned char *str_temp)
{
   int i;
```

```
count_module(1, 8);
   for (i=0; i<MAX_PREPROCESS_KEYWORDS; i++) {
count_DD_Path(1, 8, 1);

      if ( !strncmp(str_temp, preprocess_keywords[i],
            strlen(preprocess_keywords[i])) ) {
count_DD_Path(1, 8, 2);

 return(i);
      }

else {
count_DD_Path(1, 8, 3);
}
}
count_DD_Path(1, 8, 4);

   return(-1);
}
/*-------------------------------------------*/
int handle_preprocess_key( int key_no )
{

count_module(1, 9);
   if ( key_no < 0 ) {
count_DD_Path(1, 9, 1);

      return( FALSE );
   }

else {
count_DD_Path(1, 9, 2);
}
fprintf( fp2, "%s", buf_fp1 );
   preprocess_keywords_flag = key_no;
   buf_fp2_p = 0;
   buf_fp2[ buf_fp2_p ] = NULL;
   return( TRUE );
}
/*-------------------------------------------*/
int is_declare_keywords( unsigned char *str_temp )
{
   int i;
   char ctemp;


count_module(1, 10);
```

```
   for (i=0; i<MAX_DECLARE_KEYWORDS; i++) {
count_DD_Path(1, 10, 1);

      if ( !strncmp(str_temp, declare_keywords[i],
            strlen(declare_keywords[i])) ) {
count_DD_Path(1, 10, 2);

         ctemp = *(str_temp + strlen(declare_keywords[i]));
 if ( strchr(" {\t", ctemp) ) {
count_DD_Path(1, 10, 3);

            return(i);
         }

else {
count_DD_Path(1, 10, 4);
}
}

else {
count_DD_Path(1, 10, 5);
}
}
count_DD_Path(1, 10, 6);

   return(-1);
}
/*-------------------------------------------*/
int handle_declare_key( int key_no )
{

count_module(1, 11);
   buf_fp2[ buf_fp2_p ] = NULL;
   strcat( buf_fp2, declare_keywords[key_no] );
   buf_temp  += strlen( declare_keywords[key_no] );
   buf_fp2_p += strlen( declare_keywords[key_no] );
   declare_keywords_flag = key_no;
   if ( nest_brace_p > 0 ) {
count_DD_Path(1, 11, 1);

      find_char(';');
   }

else {
count_DD_Path(1, 11, 2);
}
/*      find_semicolon(); */
```

```
   return( TRUE );
}
/*-------------------------------------------*/
int is_statement_keywords( unsigned char *str_temp )
{
   int i;
   char ctemp;


count_module(1, 12);
   for (i=0; i<MAX_STATEMENT_KEYWORDS; i++) {
count_DD_Path(1, 12, 1);

      if ( !strncmp(str_temp, statement_keywords[i],
           strlen(statement_keywords[i])) ) {
count_DD_Path(1, 12, 2);

         ctemp = *(str_temp + strlen(statement_keywords[i]));
 if ( strchr(" (:;\t", ctemp) ) {
count_DD_Path(1, 12, 3);

            return(i);
         }

else {
count_DD_Path(1, 12, 4);
}
}

else {
count_DD_Path(1, 12, 5);
}
}
count_DD_Path(1, 12, 6);

   return(-1);
}
/*-------------------------------------------*/
int get_next_line( void )
{

count_module(1, 13);
   buf_temp = fgets(buf_fp1, MAX_S, fp1);
   if (buf_temp != NULL) {
count_DD_Path(1, 13, 1);

      line_number++;
```

```
/*      printf( "%s", buf_temp );
*/
      return( line_number );
   }

else {
count_DD_Path(1, 13, 2);
}
return( FALSE );
}
/*-------------------------------------------*/
int handle_asm_statement( void )
{

count_module(1, 14);
   fprintf( fp2, "%s", buf_fp1 );
   buf_fp2_p = 0;
   buf_fp2[ buf_fp2_p ] = NULL;
   *buf_temp = NULL;
   return( TRUE );
}
/*-------------------------------------------*/
int handle_label_statement( void )
{  /* case , default */
   int m;

/*   find_colon( buf_temp ); */


count_module(1, 15);
   while ( *buf_temp != NULL ) {
count_DD_Path(1, 15, 1);

  while ( !strchr(":\n", *buf_temp) ) {
count_DD_Path(1, 15, 2);

         buf_fp2[ buf_fp2_p ] = *buf_temp;
         buf_fp2_p++;
         buf_temp++;
      }
count_DD_Path(1, 15, 3);

      buf_fp2[ buf_fp2_p ] = *buf_temp;
      buf_fp2_p++;
      buf_temp++;
      if ( *buf_temp == NEWLINE ) {
count_DD_Path(1, 15, 4);
```

```
        write_fp2();
        if ( get_next_line() > 0 ) {
count_DD_Path(1, 15, 5);

           skip_space();
           continue;
        }

else {
count_DD_Path(1, 15, 6);
}
return( FALSE );
      }

else {
count_DD_Path(1, 15, 7);
}
write_fp2();
      write_dd_path( fp3 );
  return( TRUE );
   }
count_DD_Path(1, 15, 8);

}
/*-----------------------------------------*/
int handle_jump_statement( void )
{

count_module(1, 16);
   if ( find_char(';') > 0 ) {
count_DD_Path(1, 16, 1);

      return( TRUE );
   }

else {
count_DD_Path(1, 16, 2);
}
return( FALSE );
}
/*-----------------------------------------*/
int handle_statement_if( void )
{

count_module(1, 17);
   skip_space();
   if ( *buf_temp == OPEN_PAREN ) {
```

```
count_DD_Path(1, 17, 1);

      handle_parenthesis();
      skip_space();
      find_char( OPEN_BRACE );
      write_fp2();
      write_dd_path( fp3 );
      nest_brace[nest_brace_p] = IF;
      nest_brace_p++;
      return( TRUE );
   }

else {
count_DD_Path(1, 17, 2);
}
return(FALSE);
}
/*-----------------------------------------*/
int handle_statement_switch( void )
{

count_module(1, 18);
   skip_space();
   if ( *buf_temp == OPEN_PAREN ) {
count_DD_Path(1, 18, 1);

      handle_parenthesis();
      skip_space();
      find_char( OPEN_BRACE );
      write_fp2();
      nest_brace[nest_brace_p] = SWITCH;
      nest_brace_p++;
      return( TRUE );
   }

else {
count_DD_Path(1, 18, 2);
}
return(FALSE);
}
/*-----------------------------------------*/
int handle_statement_while( void )
{

count_module(1, 19);
   skip_space();
   if ( *buf_temp == OPEN_PAREN ) {
```

```
count_DD_Path(1, 19, 1);

        handle_parenthesis();
        skip_space();
        find_char( OPEN_BRACE );
        write_fp2();
        write_dd_path( fp3 );
        nest_brace[nest_brace_p] = WHILE;
        nest_brace_p++;
        return( TRUE );
    }

else {
count_DD_Path(1, 19, 2);
}
return(FALSE);
}
/*-------------------------------------------*/
int handle_statement_do( void )
{

count_module(1, 20);
    skip_space();
    find_char( OPEN_BRACE );
    write_fp2();
    write_dd_path( fp3 );
    nest_brace[nest_brace_p] = DO;
    nest_brace_p++;
    return( TRUE );
}
/*-------------------------------------------*/
int handle_statement_for( void )
{

count_module(1, 21);
    skip_space();
    if ( *buf_temp == OPEN_PAREN ) {
count_DD_Path(1, 21, 1);

        handle_parenthesis();
        skip_space();
        find_char( OPEN_BRACE );
        write_fp2();
        write_dd_path( fp3 );
        nest_brace[nest_brace_p] = FOR;
        nest_brace_p++;
        return( TRUE );
```

```
   }

else {
count_DD_Path(1, 21, 2);
}
return(FALSE);
}
/*-------------------------------------------*/
int handle_statement_else( void )
{

count_module(1, 22);
   skip_space();
   find_char( OPEN_BRACE );
   write_dd_path( fp3 );
   nest_brace[nest_brace_p] = ELSE;
   nest_brace_p++;
   return( TRUE );
}
/*-------------------------------------------*/
int handle_statement_sizeof( void )
{

count_module(1, 23);
   skip_space();
   handle_parenthesis();
   skip_space();
   return( TRUE );
}
/*-------------------------------------------*/
int handle_string_literal( void )
{
   int n;


count_module(1, 24);
   if ( *buf_temp == '"' ) {
count_DD_Path(1, 24, 1);

      buf_fp2[ buf_fp2_p ] = *buf_temp;
      buf_fp2_p++;
      buf_temp++;
      n = 0;
      while ( *buf_temp != '"' ) {
count_DD_Path(1, 24, 2);

         string_literal[n] = *buf_temp;
```

```
            n++;
            if (*buf_temp == '\\') {
count_DD_Path(1, 24, 3);

               buf_temp++;
               string_literal[n] = *buf_temp;
               n++;
            }

else {
count_DD_Path(1, 24, 4);
}
string_literal[n] = NULL;
            buf_temp++;
         }
count_DD_Path(1, 24, 5);

         buf_fp2[ buf_fp2_p ] = NULL;
         strcat( buf_fp2, string_literal );
         buf_fp2_p  += strlen( string_literal );
         buf_fp2[ buf_fp2_p ] = '"';
         buf_fp2_p++;
         buf_temp++;
      }

else {
count_DD_Path(1, 24, 6);
}
}
/*-----------------------------------------*/
int find_char( char c )
{
   int n = 1;


count_module(1, 25);
   while ( n > 0 ) {
count_DD_Path(1, 25, 1);

      while ( (!strchr("\"\n", *buf_temp)) && (*buf_temp != c) ) {
count_DD_Path(1, 25, 2);

         buf_fp2[ buf_fp2_p ] = *buf_temp;
         buf_fp2_p++;
         buf_temp++;
  }
count_DD_Path(1, 25, 3);
```

```
  if ( *buf_temp == c ) {
count_DD_Path(1, 25, 4);

 buf_fp2[ buf_fp2_p ] = *buf_temp;
 buf_fp2_p++;
 buf_temp++;
 n = 0;

count_DD_Path(1, 25, 5);
  } else {
 switch( *buf_temp ) {
           case SINGLE_QUOTE:
count_DD_Path(1, 25, 6);

              handle_single_quote();
              break;
           case DOUBLE_QUOTE:
count_DD_Path(1, 25, 7);

              handle_string_literal();
              break;
case NEWLINE:
count_DD_Path(1, 25, 8);

              buf_fp2[ buf_fp2_p ] = *buf_temp;
              buf_fp2_p++;
              write_fp2();
              if ( get_next_line() < 1 ) {
count_DD_Path(1, 25, 9);

                 return( FALSE );
              }

else {
count_DD_Path(1, 25, 10);
}
break;
        }
  }
   }
count_DD_Path(1, 25, 11);

   return( TRUE );
}
/*-------------------------------------------*/
int handle_single_quote( void )
{
```

```
count_module(1, 26);
   do {
count_DD_Path(1, 26, 1);

      buf_fp2[ buf_fp2_p ] = *buf_temp;
      buf_fp2_p++;
      buf_temp++;
   } while ( *buf_temp != SINGLE_QUOTE );
count_DD_Path(1, 26, 2);

   buf_fp2[ buf_fp2_p ] = *buf_temp;
   buf_fp2_p++;
   buf_temp++;
}
/*-------------------------------------------*/
int handle_parenthesis( void )
{
   int n = 1;


count_module(1, 27);
   buf_fp2[ buf_fp2_p ] = *buf_temp;
   buf_fp2_p++;
   buf_temp++;
   while ( n > 0 ) {
count_DD_Path(1, 27, 1);

  while ( !strchr("()\"\n", *buf_temp) ) {
count_DD_Path(1, 27, 2);

if ( *buf_temp == SINGLE_QUOTE ) {
count_DD_Path(1, 27, 3);

handle_single_quote();
continue;
}

else {
count_DD_Path(1, 27, 4);
}
buf_fp2[ buf_fp2_p ] = *buf_temp;
         buf_fp2_p++;
         buf_temp++;
      }
count_DD_Path(1, 27, 5);

      switch( *buf_temp ) {
```

```
         case OPEN_PAREN :
count_DD_Path(1, 27, 6);

buf_fp2[ buf_fp2_p ] = *buf_temp;
buf_fp2_p++;
            buf_temp++;
            n++;
            break;
         case CLOSE_PAREN:
count_DD_Path(1, 27, 7);

buf_fp2[ buf_fp2_p ] = *buf_temp;
buf_fp2_p++;
            buf_temp++;
            n--;
            break;
 case DOUBLE_QUOTE:
count_DD_Path(1, 27, 8);

            handle_string_literal();
            break;
         case NEWLINE:
count_DD_Path(1, 27, 9);

            write_fp2();
            if ( get_next_line() < 1 ) {
count_DD_Path(1, 27, 10);

               return( FALSE );
            }

else {
count_DD_Path(1, 27, 11);
}
break;
      }
   }
count_DD_Path(1, 27, 12);

   return( TRUE );
}
/*-------------------------------------------*/
int check_module(void)
{

count_module(1, 28);
   if ( nest_brace_p == 0 ) {
```

```
count_DD_Path(1, 28, 1);

  line_of_module = line_number;
  skip_space();
      if ( *buf_temp == OPEN_PAREN ) {
count_DD_Path(1, 28, 2);

         handle_parenthesis();
 while ( TRUE ) {
count_DD_Path(1, 28, 3);

skip_space();
buf_fp2[ buf_fp2_p ] = *buf_temp;
buf_fp2_p++;
if ( *buf_temp == '\r' ) {
count_DD_Path(1, 28, 4);

buf_temp++;
continue;
}

else {
count_DD_Path(1, 28, 5);
}
if ( *buf_temp == '\n' ) {
count_DD_Path(1, 28, 6);

                buf_temp++;
                write_fp2();
                if (get_next_line() < 1 ) {
count_DD_Path(1, 28, 7);

                    return( TRUE );
}

else {
count_DD_Path(1, 28, 8);
}
continue;
}

else {
count_DD_Path(1, 28, 9);
}
if ( *buf_temp == OPEN_BRACE ) {
count_DD_Path(1, 28, 10);
```

```
buf_temp++;
nest_brace[ nest_brace_p ] = MODULE;
                nest_brace_p++;
                return( TRUE );
}

else {
count_DD_Path(1, 28, 11);
}
return( FALSE );
        }
count_DD_Path(1, 28, 12);

      }

else {
count_DD_Path(1, 28, 13);
}
}

else {
count_DD_Path(1, 28, 14);
}
return( FALSE );
}
/*-----------------------------------------------*/
int write_module( FILE *fp )
{
   m_count++;

count_module(1, 29);
   ddp_count = 0;
   fprintf( fp, "\r\n%2d: count_module(%s, %d)\r\n",
            m_count, module_name, line_of_module );
   return( m_count );
}
/*-----------------------------------------------*/
int write_dd_path( FILE *fp )
{
   ddp_count++;

count_module(1, 30);
   fprintf( fp, "     %2d: COUNT_DD_PATH(%d, %s, %d)\r\n",
ddp_count, ddp_count, module_name, statement_line_no );
   return( ddp_count );
}
```

ประวัติผู้เขียน

นายวิทยา ตรีนิติกุล เกิดเมื่อวันที่ 15 เมษายน พ.ศ. 2506 ที่เขตสัมพันธวงศ์ จังหวัดกรุงเทพมหานคร สำเร็จการศึกษา ปริญญาวิทยาศาสตร์บัณฑิต (คณิตศาสตร์) จากภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ มหาวิทยาลัยรามคำแหง ปีการศึกษา 2528 และเข้าศึกษาต่อปริญญาโท สาขาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2531