/* ____ *

src2tex* version 2.12 (Sep 9, 1996)

Kazuo Amano† and Shinichi Nomoto‡

* ____ */

/* ____________________ langflag.c ____________________ */

```
#include <stdio.h>
#include "src2tex.h"

#define SkipSpaces1(Ptr) while((*Ptr>'\0')&&(*Ptr<=' '))++Ptr
#define SkipSpaces2(Ptr) while((*Ptr>'\0')&&(*Ptr<=' ')&&(*Ptr!='\n'))++Ptr

int TXT_flag = 0;                       /* TEXT flag                        */
int BAS_flag = 0;                       /* BASIC flag                       */
int CBL_flag = 0;                       /* COBOL flag                       */
int C_flag = 0;                         /* C flag                           */
int F77_flag = 0;                       /* FORTRAN flag                     */
int HTML_flag = 0;                      /* HTML flag                        */
int JAVA_flag = 0;                      /* JAVA flag                        */
int LISP_flag = 0;                      /* LISP flag                        */
int MAKE_flag = 0;                      /* MAKE flag                        */
int PAS_flag = 0;                       /* PASCAL flag                      */
int PERL_flag = 0;                      /* PERL flag                        */
int SH_flag = 0;                        /* SHELL flag                       */
int TCL_flag = 0;                       /* TCL/TK flag                      */

int ASR_flag = 0;                       /* ASIR flag                        */
int MAC_flag = 0;                       /* MACSYMA flag                     */
int MAP_flag = 0;                       /* MAPLE flag                       */
int MAT_flag = 0;                       /* MATHEMATICA flag                 */
int MLAB_flag = 0;                      /* MATLAB flag                      */
int MPAD_flag = 0;                      /* MuPAD flag                       */
int RED_flag = 0;                       /* REDUCE flag                      */

int M_flag = 0;                         /* M-file flag                      */
```

/* ____________________ search a phrase globally ____________________ */

† Department of Mathematics, Josai University, Keyakidai, Sakado, Saitama, 350-02, JAPAN (kamano@po.iijnet.or.jp)

‡ Department of Mathematics, Josai University, Keyakidai, Sakado, Saitama, 350-02, JAPAN (m93093@crescent.josai.ac.jp)

/* This function `search_phrase(buf_ptr, char_ptr)` searches the area

`buf_ptr+0, buf_ptr+1,` $\cdots$, `buf_ptr+(int)(BUFFER_SIZE/2)-1`

and tries to find a phrase stored at `char_ptr`="*phrase*". It returns the following values according to the result of search.

$$\texttt{search\_phrase}() = \begin{cases} 1 & \text{if given phrase exists} \\ 0 & \text{otherwise} \end{cases}$$

The automatic language identification mechanism essentially depends on this function. */

```
int search_phrase(buf_ptr, char_ptr)
int *buf_ptr;
char *char_ptr;
{
    char *c_ptr, c;
    int *b_ptr1, *b_ptr2, flag;

    b_ptr1 = buf_ptr;
    flag = 1;
    while (b_ptr1 != buf_ptr + (int) (BUFFER_SIZE / 2))
      {
          b_ptr2 = b_ptr1;
          SkipSpaces1(b_ptr2);
          c_ptr = char_ptr;
          SkipSpaces1(c_ptr);
          while (*c_ptr != '\0')
            {
                c = (char)*b_ptr2;
                if ((c >= 'A') && (c <= 'Z'))    /* upper case -> lower case */
                    c += 0x20;
                flag = c - *c_ptr;
                if (flag != 0)
                    break;
                ++b_ptr2;
                SkipSpaces1(b_ptr2);
                ++c_ptr;
                SkipSpaces1(c_ptr);
            }
          if (flag == 0)
              break;
          ++b_ptr1;
      }
#ifdef DEBUGGING
    if (flag == 0)
      printf("search_phrase(): phrase \"%s\" exists\n", char_ptr);
    else
      printf("search_phrase(): phrase \"%s\" is not found\n", char_ptr);
#endif
    if (flag == 0)
      return 1;
    else
      return 0;
}
```

/* ______________________________ search two phrases globally ______________________________ */

/* This function `search_two_phrases(buf_ptr, char_ptr1, char_ptr2)` searches the area

`buf_ptr+0, buf_ptr+1`, $\cdots$, `buf_ptr+(int)(BUFFER_SIZE/2)-1`

and tries to find a pair of phrases stored at `char_ptr1`="*phrase1*", `char_ptr2`="*phrase2*". It returns the following values according to the result of search.

$$\texttt{search\_two\_phrases()} = \begin{cases} 1 & \text{if given phrases exist} \\ 0 & \text{otherwise} \end{cases}$$

The automatic language identification mechanism essentially depends on this function. */

```
int search_two_phrases(buf_ptr, char_ptr1, char_ptr2)
int *buf_ptr;
char *char_ptr1, *char_ptr2;
{
    char *c_ptr, c;
    int *b_ptr1, *b_ptr2, i, flag;

    b_ptr1 = buf_ptr;
    flag = 1;
    while (b_ptr1 != buf_ptr + (int) (BUFFER_SIZE / 2))
      {
          b_ptr2 = b_ptr1;
          SkipSpaces1(b_ptr2);
          c_ptr = char_ptr1;
          SkipSpaces1(c_ptr);
          while (*c_ptr != '\0')
            {
                c = (char)*b_ptr2;
                if ((c >= 'A') && (c <= 'Z'))    /* upper case -> lower case */
                    c += 0x20;
                flag = c - *c_ptr;
                if (flag != 0)
                    break;
                ++b_ptr2;
                SkipSpaces1(b_ptr2);
                ++c_ptr;
                SkipSpaces1(c_ptr);
            }
          if (flag == 0)
              break;
          ++b_ptr1;
      }
#ifdef DEBUGGING
    if (flag == 0)
      printf("search_two_phrases(): phrase \"%s\" exists\n", char_ptr1);
    else
      printf("search_two_phrases(): phrase \"%s\" is not found\n", char_ptr1);
#endif
    if (flag != 0)
      return 0;
    for (i = 0 ; i < 40 ; ++i)
      {
          b_ptr2 = b_ptr1;
          SkipSpaces1(b_ptr2);
          c_ptr = char_ptr2;
          SkipSpaces1(c_ptr);
```

```
          while (*c_ptr != '\0')
            {
                c = (char)*b_ptr2;
                if ((c >= 'A') && (c <= 'Z'))    /* upper case -> lower case */
                    c += 0x20;
                flag = c - *c_ptr;
                if (flag != 0)
                    break;
                ++b_ptr2;
                SkipSpaces1(b_ptr2);
                ++c_ptr;
                SkipSpaces1(c_ptr);
            }
          if (flag == 0)
              break;
          ++b_ptr1;
        }
#ifdef DEBUGGING
    if (flag == 0)
      printf("                      phrase \"%s\" exists\n", char_ptr2);
    else
      printf("                      phrase \"%s\" is not found\n", char_ptr2);
#endif
    if (flag == 0)
      return 1;
    else
      return 0;
}
```

/* ———————————— get a phrase locally ———————————— */
/* This function `get_phrase(buf_ptr, char_ptr)` searches the area
`buf_ptr+0`, `buf_ptr+1`, $\cdots$, `buf_ptr+63`
and tries to find an exactly the same phrase stored at `char_ptr=`"*phrase*". It returns the following values according to the result of search.

$$\texttt{get\_phrase}() = \begin{cases} \text{the address of desired phrase} & \text{if given phrase exists} \\ \text{the address of null phrase} & \text{otherwise .} \end{cases}$$

*/

```
int *get_phrase(buf_ptr, char_ptr)
int *buf_ptr;
char *char_ptr;
{
    char *c_ptr, c;
    int *b_ptr1, *b_ptr2, i, flag = 1;
    static int null_phrase[1]={0x00};

    b_ptr1 = buf_ptr;
    for (i=0; i<64; ++i)
      {
          b_ptr2 = b_ptr1;
          c_ptr = char_ptr;
          while (*c_ptr != '\0')
```

```
            {
                c = (char)*b_ptr2;
                if ((c >= 'A') && (c <= 'Z'))    /* upper case -> lower case */
                    c += 0x20;
                flag = c - *c_ptr;
                if (flag != 0)
                    break;
                ++b_ptr2;
                ++c_ptr;
            }
          if (flag == 0)
              break;
          ++b_ptr1;
      }
#ifdef DEBUGGING
    if (flag == 0)
      printf("get_phrase(): phrase \"%s\" exists\n", char_ptr);
    else
      printf("get_phrase(): phrase \"%s\" is not found\n", char_ptr);
#endif
    if (flag == 0)
      return b_ptr1;
    else
      return null_phrase;
}
```

/* ———————————— search line locally ———————————— */

/* This function `search_line(buf_ptr, char_ptr)` searches the area

`buf_ptr+0`, `buf_ptr+1`, $\cdots$, `buf_ptr+255`

and tries to find a line of phrase stored at `char_ptr` =”a *line of phrase*”. It returns the following values according to the result of search.

$$\texttt{search\_line()} = \begin{cases} 1 & \text{if given line of phrase exists} \\ 0 & \text{otherwise .} \end{cases}$$

*/

```
int search_line(buf_ptr,char_ptr)
int *buf_ptr;
char *char_ptr;
{
  char *c_ptr, c, line1[256], line2[256];
  int *b_ptr, i, flag=1;

  b_ptr = buf_ptr;
  SkipSpaces2(b_ptr);
  c_ptr = char_ptr;
  SkipSpaces2(c_ptr);
  for (i=0; ((i < 255) && (*c_ptr != 0x00)); ++i)
    {
      c = (char)*b_ptr;
      if (c == '\n')
        c = 0x00;
      if ((c >= 'A') && (c <= 'Z'))      /* upper case -> lower case */
        c += 0x20;
```

```
      line1[i] = c;
      line2[i] = *c_ptr;
      ++b_ptr;
      SkipSpaces2(b_ptr);
      ++c_ptr;
      SkipSpaces2(c_ptr);
    }
  line1[i] = 0x00;
  line2[i] = 0x00;
  for (i=0; ((i < 256) && (line2[i] != 0x00)); ++i)
    {
      flag = line1[i] - line2[i];
      if (flag != 0)
        break;
    }
#ifdef DEBUGGING
    if (flag == 0)
      {
        printf("given line is ...\n");
        printf("\"%s\"\n", line1);
        printf("searching a line ...\n");
        printf("\"%s\"\n", line2);
        printf("search_line(): line \"%s\" exists\n", char_ptr);
      }
    else
      {
        printf("given line is ...\n");
        printf("\"%s\"\n", line1);
        printf("searching a line ...\n");
        printf("\"%s\"\n", line2);
        printf("search_line(): line \"%s\" is not found\n", char_ptr);
      }
#endif
  if (flag == 0)
    return 1;
  else
    return 0;
}


/* ____________________ set BASIC flag ____________________ */

void set_bas_flag(buf_ptr)
int *buf_ptr;
{
  if (M_flag != 0)
    return;
  /* search characteristic phrases                                  */
  BAS_flag = 10 * search_phrase(buf_ptr, "' basic");
  BAS_flag += 10 * search_phrase(buf_ptr, "rem basic");
  BAS_flag += search_phrase(buf_ptr, "0 '");
  BAS_flag += search_phrase(buf_ptr, "1 '");
  BAS_flag += search_phrase(buf_ptr, "2 '");
```

```
  BAS_flag += search_phrase(buf_ptr, "3 '");
  BAS_flag += search_phrase(buf_ptr, "4 '");
  BAS_flag += search_phrase(buf_ptr, "5 '");
  BAS_flag += search_phrase(buf_ptr, "6 '");
  BAS_flag += search_phrase(buf_ptr, "7 '");
  BAS_flag += search_phrase(buf_ptr, "8 '");
  BAS_flag += search_phrase(buf_ptr, "9 '");
  BAS_flag += 2 * search_phrase(buf_ptr, "0 rem");
  BAS_flag += 2 * search_phrase(buf_ptr, "1 rem");
  BAS_flag += 2 * search_phrase(buf_ptr, "2 rem");
  BAS_flag += 2 * search_phrase(buf_ptr, "3 rem");
  BAS_flag += 2 * search_phrase(buf_ptr, "4 rem");
  BAS_flag += 2 * search_phrase(buf_ptr, "5 rem");
  BAS_flag += 2 * search_phrase(buf_ptr, "6 rem");
  BAS_flag += 2 * search_phrase(buf_ptr, "7 rem");
  BAS_flag += 2 * search_phrase(buf_ptr, "8 rem");
  BAS_flag += 2 * search_phrase(buf_ptr, "9 rem");
  BAS_flag += search_phrase(buf_ptr, "cls");
  BAS_flag += search_phrase(buf_ptr, "defdbl");
  BAS_flag += search_phrase(buf_ptr, "<>");
  BAS_flag += 2 * search_phrase(buf_ptr, "gosub");
  BAS_flag += 3 * search_phrase(buf_ptr, "then gosub");
  BAS_flag += 3 * search_phrase(buf_ptr, "then print");
  BAS_flag += search_phrase(buf_ptr, "wend");
  BAS_flag += search_phrase(buf_ptr, "input \"");
  BAS_flag += search_phrase(buf_ptr, "data \"");
  BAS_flag += search_phrase(buf_ptr, "print \"");
  BAS_flag += 2 * search_phrase(buf_ptr, "$=\"");
  BAS_flag += 4 * search_phrase(buf_ptr, "left$(");
  BAS_flag += 4 * search_phrase(buf_ptr, "mid$(");
  BAS_flag += 4 * search_phrase(buf_ptr, "right$(");
  BAS_flag += search_phrase(buf_ptr, "dir$(");
  BAS_flag += search_phrase(buf_ptr, "getattr(");
  BAS_flag += search_phrase(buf_ptr, "chdir \"");
  BAS_flag += search_phrase(buf_ptr, "curdir$");
  BAS_flag += search_phrase(buf_ptr, "select case");
  BAS_flag += search_phrase(buf_ptr, "end select");
  BAS_flag += search_phrase(buf_ptr, "exit proc");
  BAS_flag += search_phrase(buf_ptr, "end proc");
  BAS_flag += search_phrase(buf_ptr, "end sub");
#ifdef DEBUGGING
  printf("set_bas_flag(): BAS_flag is set to %d\n", BAS_flag);
#endif
}


/* ——————————————————— set C flag ——————————————————— */

void set_c_flag(buf_ptr)
int *buf_ptr;
{
  /* search characteristic phrases                                    */
  C_flag = 10 * search_phrase(buf_ptr, "/* c */");
```

```
  C_flag += 10 * search_phrase(buf_ptr, "// c");
  C_flag += 10 * search_phrase(buf_ptr, "/* objective-c */");
  C_flag += 10 * search_phrase(buf_ptr, "// objective-c");
  C_flag += 5 * search_phrase(buf_ptr, "// //");
  C_flag += 5 * search_phrase(buf_ptr, "#include <");
  C_flag += 5 * search_phrase(buf_ptr, "#include \"");
  C_flag += 5 * search_phrase(buf_ptr, "#import <");
  C_flag += 5 * search_phrase(buf_ptr, "#import \"");
  C_flag += search_phrase(buf_ptr, "#define");
  C_flag += 4 * (search_phrase(buf_ptr, "#if")
            & search_phrase(buf_ptr, "#endif"));
  C_flag += 5 * search_phrase(buf_ptr, "@implementation");
  C_flag += 5 * search_phrase(buf_ptr, "@interface");
  C_flag += 5 * search_phrase(buf_ptr, "@private");
  C_flag += 5 * search_phrase(buf_ptr, "@protected");
  C_flag += 5 * search_phrase(buf_ptr, "@public");
  C_flag += 5 * search_phrase(buf_ptr, "@selector(");
  C_flag += 5 * search_phrase(buf_ptr, "@def(");
  C_flag += 5 * search_phrase(buf_ptr, "@encode(");
  C_flag += 3 * search_phrase(buf_ptr, "main() {");
  C_flag += 5 * search_phrase(buf_ptr, "main(argc,argv) int argc;");
  C_flag += search_phrase(buf_ptr, "printf(\"");
  C_flag += search_phrase(buf_ptr, "; } }");
  C_flag += search_phrase(buf_ptr, "++");
  C_flag += search_phrase(buf_ptr, "--");
  C_flag += 2 * search_phrase(buf_ptr, "+=");
  C_flag += 2 * search_phrase(buf_ptr, "-=");
  C_flag += 2 * search_phrase(buf_ptr, "*=");
  C_flag += 2 * search_phrase(buf_ptr, "/=");
  C_flag += search_phrase(buf_ptr, ")&&(");
  C_flag += search_phrase(buf_ptr, ")||(");
#ifdef DEBUGGING
  printf("set_c_flag(): C_flag is set to %d\n", C_flag);
#endif
}


/* ------------------------------ set COBOL flag ------------------------------ */

void set_cbl_flag(buf_ptr)
int *buf_ptr;
{
  if (M_flag != 0)
    return;
  /* search characteristic phrases                                              */
  CBL_flag = 10 * search_phrase(buf_ptr, "* cobol");
  CBL_flag += 10 * search_phrase(buf_ptr, "/ cobol");
  CBL_flag += 4 * search_phrase(buf_ptr, "identification division.");
  CBL_flag += 4 * search_phrase(buf_ptr, "program-id.");
  CBL_flag += search_phrase(buf_ptr, "author.");
  CBL_flag += 4 * search_phrase(buf_ptr, "date-written.");
  CBL_flag += 4 * search_phrase(buf_ptr, "date-written.");
  CBL_flag += 4 * search_phrase(buf_ptr, "environment division.");
```

```
  CBL_flag += 4 * search_phrase(buf_ptr, "configuration section.");
  CBL_flag += 4 * search_phrase(buf_ptr, "source-computer.");
  CBL_flag += 4 * search_phrase(buf_ptr, "object-computer.");
  CBL_flag += 4 * search_phrase(buf_ptr, "special-names.");
  CBL_flag += 3 * search_phrase(buf_ptr, "input-output section.");
  CBL_flag += 4 * search_phrase(buf_ptr, "file-contral.");
  CBL_flag += 2 * search_phrase(buf_ptr, "i-o-contral.");
  CBL_flag += 4 * search_phrase(buf_ptr, "data division.");
  CBL_flag += 3 * search_phrase(buf_ptr, "file section.");
  CBL_flag += 4 * search_phrase(buf_ptr, "working-storage section.");
  CBL_flag += 4 * search_phrase(buf_ptr, "procedure division.");
  CBL_flag += search_phrase(buf_ptr, "section.");
  CBL_flag += 4 * search_phrase(buf_ptr, "stop run.");
#ifdef DEBUGGING
  printf("set_cbl_flag(): CBL_flag is set to %d\n", CBL_flag);
#endif
}


/* ———————————————— set FORTRAN flag ———————————————— */

void set_f77_flag(buf_ptr)
int *buf_ptr;
{
  if (M_flag != 0)
    return;
  /* search characteristic phrases                                          */
  F77_flag = 10 * search_phrase(buf_ptr, "c fortran");
  F77_flag += 10 * search_phrase(buf_ptr, "* fortran");
  F77_flag += search_phrase(buf_ptr, "implicit");
  F77_flag += 2 * search_phrase(buf_ptr, "logical*2");
  F77_flag += 2 * search_phrase(buf_ptr, "logical*4");
  F77_flag += 2 * search_phrase(buf_ptr, "integer*2");
  F77_flag += 2 * search_phrase(buf_ptr, "integer*4");
  F77_flag += 2 * search_phrase(buf_ptr, "real*4");
  F77_flag += 2 * search_phrase(buf_ptr, "real*8");
  F77_flag += (search_phrase(buf_ptr, "do")
               & search_phrase(buf_ptr, "continue"));
  F77_flag += (search_phrase(buf_ptr, "subroutine")
               & search_phrase(buf_ptr, "end"));
  F77_flag += 3 * (search_phrase(buf_ptr, "write(")
               & search_phrase(buf_ptr, "format("));
  F77_flag += search_phrase(buf_ptr, "stop end");
  F77_flag += 4 * search_phrase(buf_ptr, ".lt.");
  F77_flag += 4 * search_phrase(buf_ptr, ".le.");
  F77_flag += 4 * search_phrase(buf_ptr, ".eq.");
  F77_flag += 4 * search_phrase(buf_ptr, ".ne.");
  F77_flag += 4 * search_phrase(buf_ptr, ".gt.");
  F77_flag += 4 * search_phrase(buf_ptr, ".ge.");
  F77_flag += 4 * search_phrase(buf_ptr, ".or.");
  F77_flag += 4 * search_phrase(buf_ptr, ".and.");
  F77_flag += 4 * search_phrase(buf_ptr, ".not.");
#ifdef DEBUGGING
```

```
  printf("set_f77_flag(): F77_flag is set to %d\n", F77_flag);
#endif
}


/* ------------------------- set HTML flag ------------------------- */

void set_html_flag(buf_ptr)
int *buf_ptr;
{
  /* search characteristic phrases                                   */
  HTML_flag = 10 * search_phrase(buf_ptr, "<!- html ->/");
  HTML_flag += 10 * search_phrase(buf_ptr, "</a>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</address>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</b>");
  HTML_flag += 10 * search_phrase(buf_ptr, "</base href=");
  HTML_flag += 5 * search_phrase(buf_ptr, "</body>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</caption>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</cite>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</dir>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</dl>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</em>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</h1>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</h2>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</h3>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</h4>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</h5>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</h6>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</head>");
  HTML_flag += 10 * search_phrase(buf_ptr, "</html>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</i>");
  HTML_flag += 10 * search_phrase(buf_ptr, "</img src=");
  HTML_flag += 5 * search_phrase(buf_ptr, "</kbd>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</menu>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</ol>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</strong>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</table>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</title>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</tr>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</tt>");
  HTML_flag += 5 * search_phrase(buf_ptr, "</ul>");
#ifdef DEBUGGING
  printf("set_html_flag(): HTML_flag is set to %d\n", HTML_flag);
#endif
}


/* ------------------------- set JAVA flag ------------------------- */

void set_java_flag(buf_ptr)
int *buf_ptr;
{
  /* search characteristic phrases                                   */
```

```
  JAVA_flag = C_flag;
  JAVA_flag += 10 * search_phrase(buf_ptr, "/* java */");
  JAVA_flag += 10 * search_phrase(buf_ptr, "// java");
  JAVA_flag += 10 * search_phrase(buf_ptr, "synchronized");
  JAVA_flag += 10 * search_phrase(buf_ptr, "instanceof");
  JAVA_flag += 10 * search_phrase(buf_ptr, "import java.");
  JAVA_flag += 5 * search_phrase(buf_ptr, "@author");
  JAVA_flag += 5 * search_phrase(buf_ptr, "@version");
#ifdef DEBUGGING
  printf("set_java_flag(): JAVA_flag is set to %d\n", JAVA_flag);
#endif
}


/* ———————————————————— set LISP flag ———————————————————— */

void set_lisp_flag(buf_ptr)
int *buf_ptr;
{
  if (M_flag != 0)
    return;
  /* search characteristic phrases                                   */
  LISP_flag = 10 * search_phrase(buf_ptr, "; lisp");
  LISP_flag += 10 * search_phrase(buf_ptr, "; scheme");
  LISP_flag += 5 * search_phrase(buf_ptr, ")))))");
  LISP_flag += 4 * search_phrase(buf_ptr, "(car");
  LISP_flag += 4 * search_phrase(buf_ptr, "(cdr");
  LISP_flag += 2 * search_phrase(buf_ptr, "(cons");
  LISP_flag += search_phrase(buf_ptr, "(list");
  LISP_flag += 4 * search_phrase(buf_ptr, "(setq");
  LISP_flag += 4 * search_phrase(buf_ptr, "(set!");
  LISP_flag += 4 * search_phrase(buf_ptr, "(lambda");
  LISP_flag += search_phrase(buf_ptr, "(def");
  LISP_flag += 4 * search_phrase(buf_ptr, "(defun");
  LISP_flag += search_phrase(buf_ptr, "(define");
  LISP_flag += search_phrase(buf_ptr, "(let");
  LISP_flag += search_phrase(buf_ptr, "(+");
  LISP_flag += 3 * search_phrase(buf_ptr, "(/");
  LISP_flag += 3 * search_phrase(buf_ptr, "(cond (");
  LISP_flag += search_phrase(buf_ptr, "(if (");
#ifdef DEBUGGING
  printf("set_lisp_flag(): LISP_flag is set to %d\n", LISP_flag);
#endif
}


/* ———————————————————— set MAKE flag ———————————————————— */

void set_make_flag(buf_ptr)
int *buf_ptr;
{
  int *b_ptr;
```

```
  if (M_flag != 0)
    return;
  /* search MAKE comment area                                              */
  b_ptr = buf_ptr;
  while (b_ptr != buf_ptr + (int) (BUFFER_SIZE / 2))
    {
      if ((*b_ptr == '\n')
          && (*(b_ptr + 1) == '#')
          && ((*(b_ptr + 2) == '\t') || (*(b_ptr + 2) == ' ')))
        ++MAKE_flag;
      ++b_ptr;
    }
  /* search characteristic phrases                                         */
  MAKE_flag += 10 * search_phrase(buf_ptr, "# makefile");
  MAKE_flag += 5 * search_phrase(buf_ptr, "all:");
  MAKE_flag += 5 * search_phrase(buf_ptr, "install:");
  MAKE_flag += 5 * search_phrase(buf_ptr, "install.man:");
  MAKE_flag += 5 * search_phrase(buf_ptr, "depend:");
  MAKE_flag += 5 * search_phrase(buf_ptr, "clean:");
  MAKE_flag += 3 * search_phrase(buf_ptr, "lint:");
  MAKE_flag += 3 * search_phrase(buf_ptr, "diff:");
  MAKE_flag += search_phrase(buf_ptr, "cc=");
  MAKE_flag += search_phrase(buf_ptr, "$(cc)");
  MAKE_flag += search_phrase(buf_ptr, "objs=");
  MAKE_flag += search_phrase(buf_ptr, "obj=");
  MAKE_flag += search_phrase(buf_ptr, "srcs=");
  MAKE_flag += search_phrase(buf_ptr, "src=");
  MAKE_flag += search_phrase(buf_ptr, "missing=");
  MAKE_flag += search_phrase(buf_ptr, "optimize=");
  MAKE_flag += search_phrase(buf_ptr, "parser=");
  MAKE_flag += search_phrase(buf_ptr, "flags=");
  MAKE_flag += 2 * search_phrase(buf_ptr, "ldflags=");
  MAKE_flag += 2 * search_phrase(buf_ptr, "libdir=");
  MAKE_flag += 2 * search_phrase(buf_ptr, "bindir=");
  MAKE_flag += 2 * search_phrase(buf_ptr, "destdir=");
  MAKE_flag += 2 * search_phrase(buf_ptr, "mandir=");
  MAKE_flag += 2 * search_phrase(buf_ptr, "docdir=");
  MAKE_flag += 5 * search_phrase(buf_ptr, "@echo");
  MAKE_flag += 5 * search_phrase(buf_ptr, "@-echo");
#ifdef DEBUGGING
  printf("set_make_flag(): MAKE_flag is set to %d\n", MAKE_flag);
#endif
}


/* ------------------------ set PASCAL flag ------------------------ */

void set_pas_flag(buf_ptr)
int *buf_ptr;
{
  int *b_ptr;

  if (M_flag != 0)
```

```
    return;
  /* search PASCAL comment area                                          */
  b_ptr = buf_ptr;
  while (b_ptr != buf_ptr + (int) (BUFFER_SIZE / 2))
    {
      if ((*b_ptr == '\n')
          && (*(b_ptr + 1) == '(')
          && (*(b_ptr + 2) == '*')
          && ((*(b_ptr + 3) == ' ') || (*(b_ptr + 3) == '*')))
        ++PAS_flag;
      ++b_ptr;
    }
  /* search characteristic phrases                                       */
  PAS_flag += 10 * search_phrase(buf_ptr, "{ pascal }");
  PAS_flag += 10 * search_phrase(buf_ptr, "(* pascal *)");
  PAS_flag += 2 * search_phrase(buf_ptr, "*)(*");
  PAS_flag += 3 * (search_phrase(buf_ptr, "program")
             | search_phrase(buf_ptr, "procedure"));
  PAS_flag += (search_phrase(buf_ptr, "begin")
             & search_phrase(buf_ptr, "end"));
  PAS_flag += (search_phrase(buf_ptr, "const")
             & search_phrase(buf_ptr, "var"));
  PAS_flag += 3 * search_phrase(buf_ptr, "keypressed(");
  PAS_flag += search_phrase(buf_ptr, "blockread(");
  PAS_flag += search_phrase(buf_ptr, "blockwrite(");
  PAS_flag += 4 * search_phrase(buf_ptr, "readln(");
  PAS_flag += 4 * search_phrase(buf_ptr, "writeln(");
  PAS_flag += search_phrase(buf_ptr, "write('");
  PAS_flag += search_phrase(buf_ptr, ":=");
#ifdef DEBUGGING
  printf("set_pas_flag(): PAS_flag is set to %d\n", PAS_flag);
#endif
}


/* ________________________ set PERL flag ________________________ */

void set_perl_flag(buf_ptr)
int *buf_ptr;
{
  int *b_ptr;

  if (M_flag != 0)
    return;
  /* search PERL comment area                                            */
  b_ptr = buf_ptr;
  while (b_ptr != buf_ptr + (int) (BUFFER_SIZE / 2))
    {
      if ((*b_ptr == '\n')
          && (*(b_ptr + 1) == '#')
          && ((*(b_ptr + 2) == '\t') || (*(b_ptr + 2) == ' ')))
        ++PERL_flag;
      ++b_ptr;
```

```
    }
  /* search characteristic phrases                                              */
  PERL_flag += 10 * search_phrase(buf_ptr, "# perl");
#ifdef UNIX
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/bin/perl");
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/usr/bin/perl");
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/usr/local/bin/perl");
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/bin/jperl");
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/usr/bin/jperl");
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/usr/local/bin/jperl");
#else
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\bin\\perl");
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\usr\\bin\\perl");
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\usr\\local\\bin\\perl");
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\bin\\jperl");
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\usr\\bin\\jperl");
  PERL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\usr\\local\\bin\\jperl");
#endif
  PERL_flag += 2 * search_phrase(buf_ptr, ".+");
  PERL_flag += 2 * search_phrase(buf_ptr, ".-");
  PERL_flag += 3 * search_phrase(buf_ptr, ".*");
  PERL_flag += 2 * search_phrase(buf_ptr, "./");
  PERL_flag += 3 * search_phrase(buf_ptr, ".\\");
  PERL_flag += 3 * search_phrase(buf_ptr, ".**");
  PERL_flag += 3 * search_phrase(buf_ptr, ".^");
  PERL_flag += 3 * search_phrase(buf_ptr, "~=");
  PERL_flag += 3 * search_phrase(buf_ptr, "=~");
  PERL_flag += 3 * search_phrase(buf_ptr, ".=");
  PERL_flag += 4 * search_phrase(buf_ptr, ".'");
  PERL_flag += 4 * search_phrase(buf_ptr, "/^");
  PERL_flag += 3 * search_phrase(buf_ptr, "/\\");
  PERL_flag += search_phrase(buf_ptr, "print \"");
  PERL_flag += search_phrase(buf_ptr, "<>");
  PERL_flag += 4 * search_phrase(buf_ptr, "while(<");
  PERL_flag += 4 * search_phrase(buf_ptr, "$_");
  PERL_flag += 3 * search_phrase(buf_ptr, "$.");
  PERL_flag += 4 * search_phrase(buf_ptr, "@_");
  PERL_flag += 2 * search_phrase(buf_ptr, "__end__");
#ifdef DEBUGGING
  printf("set_perl_flag(): PERL_flag is set to %d\n", PERL_flag);
#endif
```

```
}


/* ———————————————— set SHELL flag ———————————————— */

void set_shell_flag(buf_ptr)
int *buf_ptr;
{
  int *b_ptr;

  if (M_flag != 0)
    return;
  /* search SHELL comment area                                          */
  b_ptr = buf_ptr;
  while (b_ptr != buf_ptr + (int) (BUFFER_SIZE / 2))
    {
      if ((*b_ptr == '\n')
          && (*(b_ptr + 1) == '#')
          && ((*(b_ptr + 2) == '\t') || (*(b_ptr + 2) == ' ')))
        ++SH_flag;
      ++b_ptr;
    }
  /* search characteristic phrases                                      */
  SH_flag += 10 * search_phrase(buf_ptr, "# shell");
#ifdef UNIX
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/bin/sh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/usr/bin/sh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/bin/csh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/usr/bin/csh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/bin/ksh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/usr/bin/ksh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/bin/tcsh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/usr/bin/tcsh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/usr/local/bin/tcsh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/bin/zsh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/usr/bin/zsh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/usr/local/bin/zsh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/bin/bash");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/usr/bin/bash");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "/usr/local/bin/bash");
#else
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "\\sh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "\\csh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "\\ksh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "\\tcsh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "\\zsh");
  SH_flag += 10 * search_two_phrases(buf_ptr, "#!", "\\bash");
#endif
#ifdef DEBUGGING
  printf("set_shell_flag(): SH_flag is set to %d\n", SH_flag);
#endif
}
```

/* ——— set TCL/TK flag ——— */

```
void set_tcl_flag(buf_ptr)
int *buf_ptr;
{
  int *b_ptr;

  if (M_flag != 0)
    return;
  /* search TCL/TK comment area                                        */
  b_ptr = buf_ptr;
  while (b_ptr != buf_ptr + (int) (BUFFER_SIZE / 2))
    {
      if ((*b_ptr == '\n')
          && (*(b_ptr + 1) == '#')
          && ((*(b_ptr + 2) == '\t') || (*(b_ptr + 2) == ' ')))
        ++TCL_flag;
      ++b_ptr;
    }
  /* search characteristic phrases                                     */
  TCL_flag += 10 * search_phrase(buf_ptr, "# tcl/tk");
#ifdef UNIX
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/bin/tclsh");
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/usr/bin/tclsh");
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/usr/local/bin/tclsh");
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/bin/wish");
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/usr/bin/wish");
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "/usr/local/bin/wish");
#else
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\bin\\tclsh");
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\usr\\bin\\tclsh");
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\usr\\local\\bin\\tclsh");
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\bin\\wish");
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\usr\\bin\\wish");
  TCL_flag += 10
    * search_two_phrases(buf_ptr, "#!", "\\usr\\local\\bin\\wish");
#endif
  TCL_flag += 2 * search_phrase(buf_ptr, "tcl_");
  TCL_flag += 2 * search_phrase(buf_ptr, "tk_");
  TCL_flag += 2 * search_phrase(buf_ptr, "button .");
  TCL_flag += 2 * search_phrase(buf_ptr, "pack .");
```

```
  TCL_flag += 2 * search_phrase(buf_ptr, "destroy .");
  TCL_flag += 2 * search_phrase(buf_ptr, "entry .");
  TCL_flag += 2 * search_phrase(buf_ptr, "label .");
  TCL_flag += 2 * search_phrase(buf_ptr, "bind .");
  TCL_flag += 2 * search_phrase(buf_ptr, "radiobutton .");
  TCL_flag += 2 * search_phrase(buf_ptr, "scale .");
  TCL_flag += 2 * search_phrase(buf_ptr, "frame .");
  TCL_flag += search_phrase(buf_ptr, "menu .");
  TCL_flag += 2 * search_phrase(buf_ptr, "menubutton .");
  TCL_flag += 2 * search_phrase(buf_ptr, "focus .");
  TCL_flag += 5 * search_phrase(buf_ptr, "tk_menuBar .");
  TCL_flag += 2 * search_phrase(buf_ptr, "geometry .");
  TCL_flag += search_phrase(buf_ptr, "title .");
  TCL_flag += 2 * search_phrase(buf_ptr, "tkwait");
  TCL_flag += 2 * search_phrase(buf_ptr, "blink .");
#ifdef DEBUGGING
  printf("set_tcl_flag(): TCL_flag is set to %d\n", TCL_flag);
#endif
}


/* ------------------------ set ASIR flag ------------------------ */
/* Before calling this function set_asr_flag() we have to call set_c_flag() in advance. */

void set_asr_flag(buf_ptr)
int *buf_ptr;
{
  if (M_flag != 0)
    return;
  /* search characteristic phrases */
  ASR_flag = C_flag;
  ASR_flag += 10 * search_phrase(buf_ptr, "/* asir */");
  ASR_flag += search_phrase(buf_ptr, "fujitsu lab");
  ASR_flag += search_phrase(buf_ptr, "computer algebra");
  ASR_flag += search_phrase(buf_ptr, "algebraic");
  ASR_flag += search_phrase(buf_ptr, "symbolic");
  ASR_flag += search_phrase(buf_ptr, "ctrl(\"");
  ASR_flag += search_phrase(buf_ptr, "} def");
  ASR_flag += 2 * search_phrase(buf_ptr, "*/ def");
  ASR_flag += search_phrase(buf_ptr, "pari(");
  ASR_flag += search_phrase(buf_ptr, "setprec(");
  ASR_flag += search_phrase(buf_ptr, "newvect(");
  ASR_flag += search_phrase(buf_ptr, "newmat(");
  ASR_flag += search_phrase(buf_ptr, "functor(");
  ASR_flag += search_phrase(buf_ptr, "funargs(");
  ASR_flag += search_phrase(buf_ptr, "rtostr(");
  ASR_flag += search_phrase(buf_ptr, "strtov(");
  ASR_flag += search_phrase(buf_ptr, "which(\"");
  ASR_flag += search_phrase(buf_ptr, "plotover(");
  ASR_flag += search_phrase(buf_ptr, "conplot(");
  ASR_flag += search_phrase(buf_ptr, "ifplot(");
  ASR_flag += search_phrase(buf_ptr, "grm(");
  ASR_flag += search_phrase(buf_ptr, "hgr(");
```

```
  ASR_flag += search_phrase(buf_ptr, "hgrm(");
  ASR_flag += search_phrase(buf_ptr, "newalg(");
  ASR_flag += search_phrase(buf_ptr, "defpoly(");
  ASR_flag += search_phrase(buf_ptr, "ratint(");
  if (ASR_flag == C_flag)
    ASR_flag = 0;
#ifdef DEBUGGING
  printf("set_asr_flag(): ASR_flag is set to %d\n", ASR_flag);
#endif
}


/* ------------------------------ set MACSYMA flag ------------------------------ */

void set_mac_flag(buf_ptr)
int *buf_ptr;
{
  /* search characteristic phrases                                                */
  MAC_flag = 10 * search_phrase(buf_ptr, "/* macsyma */");
  MAC_flag += 10 * search_phrase(buf_ptr, "/* maxima */");
  MAC_flag += 5 * search_phrase(buf_ptr, "?round");
  MAC_flag += 5 * search_phrase(buf_ptr, "?truncate");
  MAC_flag += 5 * search_phrase(buf_ptr, "::= ");
  MAC_flag += search_phrase(buf_ptr, ":=");
  MAC_flag += search_phrase(buf_ptr, "cabs(");
  MAC_flag += 2 * search_phrase(buf_ptr, "declare(");
  MAC_flag += 5 * search_phrase(buf_ptr, "define_variable(");
  MAC_flag += search_phrase(buf_ptr, "diff(");
  MAC_flag += search_phrase(buf_ptr, "eigenvals(");
  MAC_flag += search_phrase(buf_ptr, "eigenvects(");
  MAC_flag += 5 *search_phrase(buf_ptr, "eval_when");
  MAC_flag += 3 * search_phrase(buf_ptr, "gendiff(");
  MAC_flag += search_phrase(buf_ptr, "graph2(");
  MAC_flag += search_phrase(buf_ptr, "graph3d(");
  MAC_flag += 3 * search_phrase(buf_ptr, "file_");
  MAC_flag += 3 * search_phrase(buf_ptr, "grobner_basis");
  MAC_flag += 5 * search_phrase(buf_ptr, "mode_check_");
  MAC_flag += 5 * search_phrase(buf_ptr, "mode_declare(");
  MAC_flag += search_phrase(buf_ptr, "plot2(");
  MAC_flag += search_phrase(buf_ptr, "plot3d(");
  MAC_flag += 3 * search_phrase(buf_ptr, "resolvante_");
  MAC_flag += 5 * search_phrase(buf_ptr, "setup_autoload(");
  MAC_flag += search_phrase(buf_ptr, "solve(");
  MAC_flag += search_phrase(buf_ptr, "sum(");
  MAC_flag += search_phrase(buf_ptr, "taylor(");
  MAC_flag += search_phrase(buf_ptr, "taylor_");
  MAC_flag += search_phrase(buf_ptr, "tr_");
  MAC_flag += 3 * search_phrase(buf_ptr, "undiff(");
  MAC_flag += 5 * search_phrase(buf_ptr, "with_stdout(");
  MAC_flag += 3 * search_phrase(buf_ptr, "writefile(");
#ifdef DEBUGGING
  printf("set_mac_flag(): MAC_flag is set to %d\n", MAC_flag);
#endif
```

```
}


/* ———————————————— set MAPLE flag ———————————————— */

void set_map_flag(buf_ptr)
int *buf_ptr;
{
  int *b_ptr;

  /* search MAPLE comment area                                    */
  b_ptr = buf_ptr;
  while (b_ptr != buf_ptr + (int) (BUFFER_SIZE / 2))
    {
      if ((*b_ptr == '\n')
          && (*(b_ptr + 1) == '#')
          && ((*(b_ptr + 2) == '\t') || (*(b_ptr + 2) == ' ')))
        ++MAP_flag;
      ++b_ptr;
    }
  /* search characteristic phrases                                */
  MAP_flag += 10 * search_phrase(buf_ptr, "# maple");
  MAP_flag += search_phrase(buf_ptr, "##");
  MAP_flag += search_phrase(buf_ptr, "###");
  MAP_flag += 2 * search_phrase(buf_ptr, "####");
  MAP_flag += search_phrase(buf_ptr, "university of waterloo");
  MAP_flag += search_phrase(buf_ptr, "computer algebra");
  MAP_flag += search_phrase(buf_ptr, "algebraic");
  MAP_flag += search_phrase(buf_ptr, "symbolic");
  MAP_flag += search_phrase(buf_ptr, "mvcal");
  MAP_flag += search_phrase(buf_ptr, "daub");
  MAP_flag += search_phrase(buf_ptr, "calcp");
  MAP_flag += search_phrase(buf_ptr, "calcplot");
  MAP_flag += search_phrase(buf_ptr, "posets");
  MAP_flag += search_phrase(buf_ptr, "coxeter");
  MAP_flag += search_phrase(buf_ptr, "matthews");
  MAP_flag += search_phrase(buf_ptr, "casa");
  MAP_flag += search_phrase(buf_ptr, "macro(");
  MAP_flag += search_phrase(buf_ptr, "with(");
  MAP_flag += 4 * search_phrase(buf_ptr, "fi;");
  MAP_flag += 4 * search_phrase(buf_ptr, "od;");
  MAP_flag += 4 * search_phrase(buf_ptr, "error(`");
  MAP_flag += 3 * search_phrase(buf_ptr, "`)");
  MAP_flag += search_phrase(buf_ptr, "='");
  MAP_flag += 3 * search_phrase(buf_ptr, "=`");
  MAP_flag += 3 * search_phrase(buf_ptr, "`=");
  MAP_flag += search_phrase(buf_ptr, ":=");
  MAP_flag += 3 * search_phrase(buf_ptr, "`:=");
  MAP_flag += 3 * search_phrase(buf_ptr, ":=`");
  MAP_flag += search_phrase(buf_ptr, "<>");
  MAP_flag += search_phrase(buf_ptr, "proc(");
  MAP_flag += 2 * search_phrase(buf_ptr, "readlib(");
  MAP_flag += 4 * search_phrase(buf_ptr, "readlib(`");
```

```
  MAP_flag += search_phrase(buf_ptr, "eigenvals(");
  MAP_flag += search_phrase(buf_ptr, "eigenvects(");
  MAP_flag += search_phrase(buf_ptr, "simplify(");
  MAP_flag += search_phrase(buf_ptr, "sum(");
  MAP_flag += search_phrase(buf_ptr, "diff(");
  MAP_flag += search_phrase(buf_ptr, "int(");
  MAP_flag += search_phrase(buf_ptr, "solve(");
  MAP_flag += search_phrase(buf_ptr, "draw(");
  MAP_flag += search_phrase(buf_ptr, "plot(");
  MAP_flag += search_phrase(buf_ptr, "plot3d(");
  MAP_flag += search_phrase(buf_ptr, "animate(");
  MAP_flag += search_phrase(buf_ptr, "animate3d(");
  MAP_flag += 4 * search_phrase(buf_ptr, "save `");
#ifdef DEBUGGING
  printf("set_map_flag(): MAP_flag is set to %d\n", MAP_flag);
#endif
}


/* ------------------------- set MATHEMATICA flag ------------------------- */

void set_mat_flag(buf_ptr)
int *buf_ptr;
{
  int *b_ptr;

  /* search MATHEMATICA comment area                                        */
  b_ptr = buf_ptr;
  while (b_ptr != buf_ptr + (int) (BUFFER_SIZE / 2))
    {
      if ((*b_ptr == '\n')
          && (*(b_ptr + 1) == '(')
          && (*(b_ptr + 2) == '*')
          && ((*(b_ptr + 3) == '\t') || (*(b_ptr + 3) == ' ')))
        ++MAT_flag;
      ++b_ptr;
    }
  /* search characteristic phrases                                          */
  MAT_flag += 10 * search_phrase(buf_ptr, "(* mathematica *)");
  MAT_flag += 2 * search_phrase(buf_ptr, "*)(*");
  MAT_flag += search_phrase(buf_ptr, "wolfram research");
  MAT_flag += search_phrase(buf_ptr, "computer algebra");
  MAT_flag += search_phrase(buf_ptr, "algebraic");
  MAT_flag += search_phrase(buf_ptr, "symbolic");
  MAT_flag += 3 * search_phrase(buf_ptr, "beginpackage[");
  MAT_flag += 3 * search_phrase(buf_ptr, "endpackage[");
  MAT_flag += 3 * search_phrase(buf_ptr, "begin[");
  MAT_flag += 3 * search_phrase(buf_ptr, "module[");
  MAT_flag += search_phrase(buf_ptr, "block[");
  MAT_flag += 3 *search_phrase(buf_ptr, "end[");
  MAT_flag += search_phrase(buf_ptr, "on[");
  MAT_flag += search_phrase(buf_ptr, "off[");
  MAT_flag += search_phrase(buf_ptr, "array[");
```

```
  MAT_flag += search_phrase(buf_ptr, "table[");
  MAT_flag += search_phrase(buf_ptr, "vectorq[");
  MAT_flag += search_phrase(buf_ptr, "matrixq[");
  MAT_flag += search_phrase(buf_ptr, "list[");
  MAT_flag += 2 * search_phrase(buf_ptr, "evaluate[");
  MAT_flag += search_phrase(buf_ptr, "function[");
  MAT_flag += 4 * search_phrase(buf_ptr, "if[");
  MAT_flag += 4 * search_phrase(buf_ptr, "switch[");
  MAT_flag += 4 * search_phrase(buf_ptr, "do[");
  MAT_flag += 4 * search_phrase(buf_ptr, "while[");
  MAT_flag += 4 * search_phrase(buf_ptr, "for[");
  MAT_flag += 4 * search_phrase(buf_ptr, "break[");
  MAT_flag += 4 * search_phrase(buf_ptr, "continue[");
  MAT_flag += 4 * search_phrase(buf_ptr, "return[");
  MAT_flag += 4 * search_phrase(buf_ptr, "label[");
  MAT_flag += 4 * search_phrase(buf_ptr, "goto[");
  MAT_flag += 2 * search_phrase(buf_ptr, "sum[");
  MAT_flag += 2 * search_phrase(buf_ptr, "product[");
  MAT_flag += 3 * search_phrase(buf_ptr, "expand[");
  MAT_flag += 2 * search_phrase(buf_ptr, "factor[");
  MAT_flag += 2 * search_phrase(buf_ptr, "simplify[");
  MAT_flag += 2 * search_phrase(buf_ptr, "limit[");
  MAT_flag += 2 * (search_phrase(buf_ptr, "d[")
              | search_phrase(buf_ptr, "dt["));
  MAT_flag += 3 * search_phrase(buf_ptr, "integrate[");
  MAT_flag += 2 * search_phrase(buf_ptr, "solve[");
  MAT_flag += search_phrase(buf_ptr, "series[");
  MAT_flag += search_phrase(buf_ptr, "show[");
  MAT_flag += search_phrase(buf_ptr, "plot[");
  MAT_flag += search_phrase(buf_ptr, "plot3d[");
#ifdef DEBUGGING
  printf("set_mat_flag(): MAT_flag is set to %d\n", MAT_flag);
#endif
}


/* ________________ set MATLAB flag ________________ */

void set_mlab_flag(buf_ptr)
int *buf_ptr;
{
  int *b_ptr;

  /* search MATLAB or OCTAVE comment area                                    */
  b_ptr = buf_ptr;
  while (b_ptr != buf_ptr + (int) (BUFFER_SIZE / 2))
    {
      if ((*b_ptr == '\n')
          && ((*(b_ptr + 1) == '#') || (*(b_ptr + 1) == '%'))
          && ((*(b_ptr + 2) == '\t') || (*(b_ptr + 2) == ' ')))
        ++MLAB_flag;
      ++b_ptr;
    }
```

```
  /* search characteristic phrases                                          */
  MLAB_flag += 10 * search_phrase(buf_ptr, "# matlab");
  MLAB_flag += 10 * search_phrase(buf_ptr, "% matlab");
  MLAB_flag += 10 * search_phrase(buf_ptr, "# octave");
  MLAB_flag += 10 * search_phrase(buf_ptr, "% octave");
  MLAB_flag += search_phrase(buf_ptr, "# mathworks");
  MLAB_flag += search_phrase(buf_ptr, "% mathworks");
  MLAB_flag += 3 * search_phrase(buf_ptr, "endfor");
  MLAB_flag += search_phrase(buf_ptr, "function[");
  MLAB_flag += 3 * search_phrase(buf_ptr, "endfunction");
  MLAB_flag += 3 * search_phrase(buf_ptr, "endwhile");
  MLAB_flag += 2 * search_phrase(buf_ptr, "end end");
  MLAB_flag += search_phrase(buf_ptr, "plot(");
  MLAB_flag += search_phrase(buf_ptr, "plot3d(");
  MLAB_flag += 2 * search_phrase(buf_ptr, "surf(");
  MLAB_flag += 2 * search_phrase(buf_ptr, "shg");
  MLAB_flag += 2 * search_phrase(buf_ptr, "clg");
  MLAB_flag += search_phrase(buf_ptr, "gplot");
  MLAB_flag += search_phrase(buf_ptr, "gsplot");
  MLAB_flag += 3 * search_phrase(buf_ptr, "casesen");
  MLAB_flag += 3 * search_phrase(buf_ptr, "edit_history");
  MLAB_flag += 3 * search_phrase(buf_ptr, "run_history");
  MLAB_flag += 2 * search_phrase(buf_ptr, ".+");
  MLAB_flag += 2 * search_phrase(buf_ptr, ".-");
  MLAB_flag += 3 * search_phrase(buf_ptr, ".*");
  MLAB_flag += 2 * search_phrase(buf_ptr, "./");
  MLAB_flag += 3 * search_phrase(buf_ptr, ".\\");
  MLAB_flag += 3 * search_phrase(buf_ptr, ".**");
  MLAB_flag += 3 * search_phrase(buf_ptr, ".^");
  MLAB_flag += 3 * search_phrase(buf_ptr, "~=");
  MLAB_flag += 4 * search_phrase(buf_ptr, ".'");
  MLAB_flag += search_phrase(buf_ptr, "<>");
  MLAB_flag += search_phrase(buf_ptr, "printf(\"");
  MLAB_flag += 4 * search_phrase(buf_ptr, "printf('");
  MLAB_flag += 4 * search_phrase(buf_ptr, "disp('");
  MLAB_flag += search_phrase(buf_ptr, "error(\"");
  MLAB_flag += 4 * search_phrase(buf_ptr, "error('");
  MLAB_flag += 3 * search_phrase(buf_ptr, "nargin");
  MLAB_flag += 3 * search_phrase(buf_ptr, "nargout");
  MLAB_flag += 4 * search_phrase(buf_ptr, "(:");
  MLAB_flag += 4 * search_phrase(buf_ptr, ":)");
  MLAB_flag += search_phrase(buf_ptr, "zeros(");
  MLAB_flag += 3 * search_phrase(buf_ptr, "rot90");
  MLAB_flag += 3 * search_phrase(buf_ptr, "fliplr");
  MLAB_flag += 3 * search_phrase(buf_ptr, "flipud");
  MLAB_flag += search_phrase(buf_ptr, "diag");
  MLAB_flag += 3 * search_phrase(buf_ptr, "tril");
  MLAB_flag += 3 * search_phrase(buf_ptr, "triu");
  MLAB_flag += 2 * search_phrase(buf_ptr, "title('");
  MLAB_flag += 3 * search_phrase(buf_ptr, "gtext(");
  MLAB_flag += 3 * search_phrase(buf_ptr, "ginput(");
  MLAB_flag += search_phrase(buf_ptr, "input(");
#ifdef DEBUGGING
```

```
  printf("set_mlab_flag(): MLAB_flag is set to %d\n", MLAB_flag);
#endif
}


/* ________________________ set MuPAD flag ________________________ */

void set_mpad_flag(buf_ptr)
int *buf_ptr;
{
  int *b_ptr;

  /* search MuPAD comment area                                        */
  b_ptr = buf_ptr;
  while (b_ptr != buf_ptr + (int) (BUFFER_SIZE / 2))
    {
      if ((*b_ptr == '\n')
          && (*(b_ptr + 1) == '#')
          && ((*(b_ptr + 2) == '\t') || (*(b_ptr + 2) == ' ')))
        ++MPAD_flag;
      ++b_ptr;
    }
  /* search characteristic phrases                                    */
  MPAD_flag += 10 * search_phrase(buf_ptr, "# mupad #");
  MPAD_flag += search_phrase(buf_ptr, "##");
  MPAD_flag += search_phrase(buf_ptr, "###");
  MPAD_flag += 2 * search_phrase(buf_ptr, "####");
  MPAD_flag += search_phrase(buf_ptr, "computer algebra");
  MPAD_flag += search_phrase(buf_ptr, "algebraic");
  MPAD_flag += search_phrase(buf_ptr, "symbolic");
  MPAD_flag += 5 * search_phrase(buf_ptr, "end_case");
  MPAD_flag += 5 * search_phrase(buf_ptr, "end_for");
  MPAD_flag += 5 * search_phrase(buf_ptr, "end_if");
  MPAD_flag += 3 * search_phrase(buf_ptr, "parbegin");
  MPAD_flag += 5 * search_phrase(buf_ptr, "end_par");
  MPAD_flag += 5 * search_phrase(buf_ptr, "end_proc");
  MPAD_flag += 5 * search_phrase(buf_ptr, "end_repeat");
  MPAD_flag += 3 * search_phrase(buf_ptr, "seqbegin");
  MPAD_flag += 5 * search_phrase(buf_ptr, "end_seq");
  MPAD_flag += 5 * search_phrase(buf_ptr, "end_while");
  MPAD_flag += search_phrase(buf_ptr, ":=");
  MPAD_flag += search_phrase(buf_ptr, "diff(");
  MPAD_flag += search_phrase(buf_ptr, "int(");
  MPAD_flag += search_phrase(buf_ptr, "solve(");
  MPAD_flag += search_phrase(buf_ptr, "plot2d(");
  MPAD_flag += search_phrase(buf_ptr, "plot3d(");
  MPAD_flag += search_phrase(buf_ptr, "simplify(");
  MPAD_flag += search_phrase(buf_ptr, "sum(");
#ifdef DEBUGGING
  printf("set_mpad_flag(): MPAD_flag is set to %d\n", MPAD_flag);
#endif
}
```

/* ——————————— set REDUCE flag ——————————— */

```
void set_red_flag(buf_ptr)
int *buf_ptr;
{
  int *b_ptr;

  if (M_flag != 0)
    return;
  /* search REDUCE comment area                                          */
  b_ptr = buf_ptr;
  while (b_ptr != buf_ptr + (int) (BUFFER_SIZE / 2))
    {
      if ((*b_ptr == '\n')
          && (*(b_ptr + 1) == '%')
          && ((*(b_ptr + 2) == '\t') || (*(b_ptr + 2) == ' ')))
        ++RED_flag;
      ++b_ptr;
    }
  /* search characteristic phrases                                       */
  RED_flag += 10 * search_phrase(buf_ptr, "% reduce");
  RED_flag += search_phrase(buf_ptr, ". hearn");
  RED_flag += search_phrase(buf_ptr, "computer algebra");
  RED_flag += search_phrase(buf_ptr, "algebraic");
  RED_flag += search_phrase(buf_ptr, "symbolic");
  RED_flag += search_phrase(buf_ptr, "arnum");
  RED_flag += search_phrase(buf_ptr, "compact");
  RED_flag += search_phrase(buf_ptr, "excalc");
  RED_flag += search_phrase(buf_ptr, "gentran");
  RED_flag += search_phrase(buf_ptr, "orthovec");
  RED_flag += search_phrase(buf_ptr, "specfn");
  RED_flag += search_phrase(buf_ptr, "fide");
  RED_flag += search_phrase(buf_ptr, "physop");
  RED_flag += search_phrase(buf_ptr, "reacteqn");
  RED_flag += search_phrase(buf_ptr, "rlfi");
  RED_flag += search_phrase(buf_ptr, "ghyper");
  RED_flag += search_phrase(buf_ptr, "linalg");
  RED_flag += search_phrase(buf_ptr, "ncpoly");
  RED_flag += search_phrase(buf_ptr, "xideal");
  RED_flag += search_phrase(buf_ptr, "zeilberg");
  RED_flag += search_phrase(buf_ptr, "out \"");
  RED_flag += search_phrase(buf_ptr, "begin scalar");
  RED_flag += (search_phrase(buf_ptr, "off echo")
               | search_phrase(buf_ptr, "on echo"));
  RED_flag += (search_phrase(buf_ptr, "on rational")
               | search_phrase(buf_ptr, "off rational"));
  RED_flag += (search_phrase(buf_ptr, "on complex")
               | search_phrase(buf_ptr, "off complex"));
  RED_flag += (search_phrase(buf_ptr, "on round")
               | search_phrase(buf_ptr, "off round"));
  RED_flag += (search_phrase(buf_ptr, "on factor")
               | search_phrase(buf_ptr, "off factor"));
```

```
  RED_flag += (search_phrase(buf_ptr, "on nat")
             | search_phrase(buf_ptr, "off nat"));
  RED_flag += search_phrase(buf_ptr, ":=");
  RED_flag += search_phrase(buf_ptr, "part(");
  RED_flag += search_phrase(buf_ptr, "factorize(");
  RED_flag += search_phrase(buf_ptr, "remainder(");
  RED_flag += search_phrase(buf_ptr, "mat((");
  RED_flag += 2 * search_phrase(buf_ptr, "then<<");
  RED_flag += 2 * search_phrase(buf_ptr, "do<<");
  RED_flag += 2 * search_phrase(buf_ptr, "repeat<<");
  RED_flag += 2 * search_phrase(buf_ptr, "else<<");
  RED_flag += search_phrase(buf_ptr, "df(");
  RED_flag += search_phrase(buf_ptr, "int(");
  RED_flag += search_phrase(buf_ptr, "defint(");
  RED_flag += search_phrase(buf_ptr, "solve(");
  RED_flag += search_phrase(buf_ptr, "taylor(");
  RED_flag += search_phrase(buf_ptr, "groebner(");
  RED_flag += search_phrase(buf_ptr, "odesolve(");
  RED_flag += search_phrase(buf_ptr, "root(");
  RED_flag += search_phrase(buf_ptr, "plot(");
  RED_flag += search_phrase(buf_ptr, "linineq(");
#ifdef DEBUGGING
  printf("set_red_flag(): RED_flag is set to %d\n", RED_flag);
#endif
}


/* ------------------------------ set TEXT flag ------------------------------ */
/* The purpose of this function is to decide whether or not the input file is a plain text file or a src2tex
format file.                                                                   */

void set_txt_flag(buf_ptr)
int *buf_ptr;
{
  int brace_counter = 0;        /* brace counter                               */
  int dollar_counter = 0;       /* single dollar sign counter                  */
  int ddollar_counter = 0;      /* double dollar sign counter                  */
  int i, perl_flag, tmp_flag, *ptr, *mark_ptr;
  char c_1, c0, c1, c2;

  /* prepare for special treatment of perl                                     */
  tmp_flag = PERL_flag;
  set_perl_flag(buf_ptr);
  perl_flag = PERL_flag;
  PERL_flag = tmp_flag;

  /* initialize parameters and start checking                                  */
  TXT_flag = 1;
  ptr = buf_ptr;
  while (ptr != buf_ptr + (int)(BUFFER_SIZE / 2))
    {
      /* increase counters if it is necessary                                  */
      if (ptr == buf_ptr)
```

```
        c_1 = 0x00;
      else
        c_1 = *(ptr - 1);
      c0 = *ptr;
      c1 = *(ptr + 1);
      c2 = *(ptr + 2);
      ++ptr;
      if ((c_1 != '\\') && (c0 == '{'))
        ++brace_counter;
      if ((c_1 != '\\') && (c0 == '}'))
        --brace_counter;
      if ((c_1 != '$') && (c0 == '$') && (c1 != '$'))
        {
          mark_ptr = ptr;
          for(i=0 ; i < FORMULA_LEN_MAX; ++i)
            {
              ++mark_ptr;
              if(((char)*(mark_ptr -1) != '$') && ((char)*mark_ptr == '$'))
                break;
            }
          if(((char)*(mark_ptr -1) != '$') && ((char)*mark_ptr == '$'))
            {
              ++dollar_counter;
              dollar_counter %= 2;
#ifdef DEBUGGING
              printf("non-display style mathematical formula of the length ");
              printf("%d\n", i + 1);
#endif
            }
          else
            dollar_counter = 0;
        }
#ifdef LATEX
      if ((c0 == '\\') && (c1 == '('))
        {
          mark_ptr = ptr;
          for(i=0 ; i < FORMULA_LEN_MAX; ++i)
            {
              ++mark_ptr;
              if(((char)*mark_ptr == '\\') && ((char)*(mark_ptr + 1) == ')'))
                break;
            }
          if(((char)*mark_ptr == '\\') && ((char)*(mark_ptr + 1) == ')'))
            {
              dollar_counter = 1;
#ifdef DEBUGGING
              printf("non-display style mathematical formula of the length ");
              printf("%d\n", i);
#endif
            }
          else
            dollar_counter = 0;
        }
```

```
      if ((c0 == '\\') && (c1 == ')'))
        dollar_counter = 0;
      if ((c0 == '\\') && ((c1 == '[') || (c1 == ']')))
        {
          ++ddollar_counter;
          ddollar_counter %= 2;
        }
#else
      if ((c0 == '$') && (c1 == '$') && (c2 != '$'))
        {
          ++ddollar_counter;
          ddollar_counter %= 2;
        }
#endif

      /* set or unset TXT_flag                                                  */
      if (brace_counter > 0)
        {
          /* If there exits an {\ then src2tex thinks that input file is written in TeX.  */
          if ((c_1 == '{') && (c0 == '\\')
              && (((c1 >= 'a') && (c1 <= 'z'))
                  || ((c1 >= 'A') && (c1 <= 'Z')))
              && (((c2 >= 'a') && (c2 <= 'z'))
                  || ((c2 >= 'A') && (c2 <= 'Z'))))
            {
              TXT_flag = 0;
              break;
            }
          if ((c_1 == '{') && (c0 == '\\') && (c1 == ' '))
            {
              TXT_flag = 0;
              break;
            }
        }
      if ((dollar_counter == 1) || (ddollar_counter == 1))
        {
          /* check escape sequence                                                */
          if ((c0 == '\\')
              && (((c1 >= 'a') && (c1 <= 'z'))
                  || ((c1 >= 'A') && (c1 <= 'Z')))
              && (((c2 >= 'a') && (c2 <= 'z'))
                  || ((c2 >= 'A') && (c2 <= 'Z'))))
            {
              TXT_flag = 0;
#ifdef DEBUGGING
              printf("set_txt_flag(): TeX escape sequence is found\n");
#endif
            }
          /* If _ and ^ are not used properly, then src2tex thinks that
             input file is a plain text, i.e., not a TeXt.                        */
          if (((c0 == '_') || (c0 == '^')) && (perl_flag == 0))
            {
              if ((c_1 <= ' ') || (c1 <= ' '))
```

```
                {
                  TXT_flag = 1;
#ifdef DEBUGGING
                  printf("set_txt_flag(): TeX syntax error #1\n");
                  printf("                %c%c%c%c...\n", c_1, c0, c1, c2);
#endif
                  break;
                }
              if (c1 == '{')
                {
                  TXT_flag = 0;
                  break;
                }
              if (((c1 >= '0') && (c1 <= '9'))
                  || ((c1 >= 'a') && (c1 <= 'z')))
                {
                  if (((c2 >= '0') && (c2 <= '9'))
                      || ((c2 >= 'a') && (c2 <= 'z')))
                    {
                      TXT_flag = 1;
#ifdef DEBUGGING
                      printf("set_txt_flag(): TeX syntax error #2\n");
                      printf("                %c%c%c%c...\n", c_1, c0, c1, c2);
#endif
                      break;
                    }
                  else
                    {
                      TXT_flag = 0;
                    }
                }
            }
        }
    }
  /* Any news article or mail is considered as a text file.                */
  if ((search_phrase(buf_ptr, "from:") != 0)
      && (search_phrase(buf_ptr, "newsgroups:") != 0)
      && (search_phrase(buf_ptr, "subject:") != 0)
      && (search_phrase(buf_ptr, "date:") != 0)
      && (search_phrase(buf_ptr, "organization:") != 0)
      && (search_phrase(buf_ptr, "path:") != 0))
    TXT_flag = 1;
  if ((search_phrase(buf_ptr, "from:") != 0)
      && (search_phrase(buf_ptr, "subject:") != 0)
      && (search_phrase(buf_ptr, "date:") != 0)
      && (search_phrase(buf_ptr, "to:") != 0)
      && (search_phrase(buf_ptr, "return-path:") != 0))
    TXT_flag = 1;
#ifdef DEBUGGING
  printf("set_txt_flag(): TXT_flag is set to %d\n", TXT_flag);
#endif
}
```

/* —————— initialize language and text flags —————— */
/* In this function we just see the suffix of input file name and determine language type. For further details, it would be better to read the following definition of function `init_lang_flag()`. */

```
void init_lang_flag(cptr)
char *cptr[];
{
  char *ptr, c0, c1, c2, c3, c4;

  /* set language flag according to the file name */
  ptr = cptr[0];
  c0 = *ptr;
  c1 = *++ptr;
  c2 = *++ptr;
  c3 = *++ptr;
  c4 = *++ptr;
  if (((c0 == 'M') || (c0 == 'm'))              /* MAKE                    */
      && ((c1 == 'A') || (c1 == 'a'))
      && ((c2 == 'K') || (c2 == 'k'))
      && ((c3 == 'E') || (c3 == 'e'))
      && ((c4 == 'F') || (c4 == 'f')))
    MAKE_flag = 1;
  /* search the suffix of file name                                       */
  for (ptr = cptr[0]; *ptr != '\0'; ++ptr);
  for (; (*ptr != '.') && (ptr != cptr[0]); --ptr);
  /* set language flags according to the suffix                           */
  if (*ptr == '.')
    {
      c1 = *++ptr;
      c2 = *++ptr;
      c3 = *++ptr;
      c4 = *++ptr;
      if (((c1 == 'A') || (c1 == 'a'))          /* TEXT                    */
          && ((c2 == 'W') || (c2 == 'w'))
          && ((c3 == 'K') || (c3 == 'k')))
        TXT_flag = 1;
      if (((c1 == 'B') || (c1 == 'b'))
          && ((c2 == 'A') || (c2 == 'a'))
          && ((c3 == 'T') || (c3 == 't')))
        TXT_flag = 1;
      if (((c1 == 'C') || (c1 == 'c'))
          && ((c2 == 'O') || (c2 == 'o'))
          && ((c3 == 'N') || (c3 == 'n'))
          && ((c4 == 'F') || (c4 == 'f')))
        TXT_flag = 1;
      if (((c1 == 'D') || (c1 == 'd'))
          && ((c2 == 'O') || (c2 == 'o'))
          && ((c3 == 'C') || (c3 == 'c')))
        TXT_flag = 1;
      if (((c1 == 'L') || (c1 == 'l'))
          && ((c2 == 'E') || (c2 == 'e'))
          && ((c3 == 'T') || (c3 == 't')))
        TXT_flag = 1;
```

```
if (((c1 == 'L') || (c1 == 'l'))
    && ((c2 == 'O') || (c2 == 'o'))
    && ((c3 == 'G') || (c3 == 'g')))
  TXT_flag = 1;
if (((c1 == 'S') || (c1 == 's'))
    && ((c2 == 'E') || (c2 == 'e'))
    && ((c3 == 'D') || (c3 == 'd')))
  TXT_flag = 1;
if (((c1 == 'S') || (c1 == 's'))
    && ((c2 == 'Y') || (c2 == 'y'))
    && ((c3 == 'S') || (c3 == 's')))
  TXT_flag = 1;
if (((c1 == 'T') || (c1 == 't'))
    && ((c2 == 'E') || (c2 == 'e'))
    && ((c3 == 'X') || (c3 == 'x')))
  TXT_flag = 1;
if (((c1 == 'T') || (c1 == 't'))
    && ((c2 == 'X') || (c2 == 'x'))
    && ((c3 == 'T') || (c3 == 't')))
  TXT_flag = 1;
if ((c1 == 'B') || (c1 == 'b'))           /* BASIC                          */
  BAS_flag = 1;
if (((c1 == 'V') || (c1 == 'v'))
    && ((c2 == 'B') || (c2 == 'b')))
  BAS_flag = 1;
if (((c1 == 'C') || (c1 == 'c'))          /* C                              */
    && (c2 != 'B') && (c2 != 'b')
    && (c2 != 'O') && (c2 != 'o')
    && (c2 != 'S') && (c2 != 's'))
  C_flag = 1;
if (((c1 == 'V') || (c1 == 'v'))
    && ((c2 == 'C') || (c2 == 'c')))
  C_flag = 1;
if ((c1 == 'H') || (c1 == 'h'))
  C_flag = 1;
if (((c1 == 'C') || (c1 == 'c'))          /* COBOL                          */
    && ((c2 == 'B') || (c2 == 'b'))
    && ((c3 == 'L') || (c3 == 'l')))
  CBL_flag = 1;
if (((c1 == 'C') || (c1 == 'c'))
    && ((c2 == 'O') || (c2 == 'o'))
    && ((c3 == 'B') || (c3 == 'b')))
  CBL_flag = 1;
if (((c1 == 'A') || (c1 == 'a'))          /* ASIR                           */
    && ((c2 == 'S') || (c2 == 's'))
    && (((c3 >= '0') && (c3 <= '9'))
        || ((c3 >= 'A') && (c3 <= 'Z'))
        || ((c3 >= 'a') && (c3 <= 'z'))))
  ASR_flag = 1;
if ((c1 == 'F') || (c1 == 'f'))           /* FORTRAN                        */
  F77_flag = 1;
if (((c1 == 'H') || (c1 == 'h'))          /* HTML                           */
    && ((c2 == 'T') || (c2 == 't'))
```

```
    && ((c3 == 'M') || (c3 == 'm'))
    && ((c4 == 'L') || (c4 == 'l')))
  HTML_flag = 1;
if (((c1 == 'J') || (c1 == 'j'))          /* JAVA                              */
    && ((c2 == 'A') || (c2 == 'a'))
    && ((c3 == 'V') || (c3 == 'v'))
    && ((c4 == 'A') || (c4 == 'a')))
  JAVA_flag = 1;
if (((c1 == 'L') || (c1 == 'l'))          /* LISP                              */
    && ((c2 == 'S') || (c2 == 's')))
  LISP_flag = 1;
if (((c1 == 'S') || (c1 == 's'))
    && ((c2 == 'C') || (c2 == 'c')))
  LISP_flag = 1;
if (((c1 == 'E') || (c1 == 'e'))
    && ((c2 == 'L') || (c2 == 'l')))
  LISP_flag = 1;
if (((c1 == 'P') || (c1 == 'p'))          /* PASCAL                            */
    && (c2 != 'L') && (c2 != 'l')
    && (c2 != 'R') && (c2 != 'r'))
  PAS_flag = 1;
if (((c1 == 'T') || (c1 == 't'))
    && ((c2 == 'P') || (c2 == 'p')))
  PAS_flag = 1;
if (((c1 == 'P') || (c1 == 'p'))          /* PERL                              */
    && ((c2 == 'L') || (c2 == 'l')))
  PERL_flag =1;
if (((c1 == 'P') || (c1 == 'p'))
    && ((c2 == 'R') || (c2 == 'r'))
    && ((c3 == 'L') || (c3 == 'l')))
  PERL_flag = 1;
if (((c1 == 'P') || (c1 == 'p'))
    && ((c2 == 'H') || (c2 == 'h')))
  PERL_flag =1;
if (((c1 == 'P') || (c1 == 'p'))
    && ((c2 == 'M') || (c2 == 'm')))
  PERL_flag =1;
if (((c1 == 'S') || (c1 == 's'))          /* SHELL                             */
    && ((c2 == 'H') || (c2 == 'h')))
  SH_flag = 1;
if (((c1 == 'C') || (c1 == 'c'))
    && ((c2 == 'S') || (c2 == 's'))
    && ((c3 == 'H') || (c3 == 'h')))
  SH_flag = 1;
if (((c1 == 'K') || (c1 == 'k'))
    && ((c2 == 'S') || (c2 == 's'))
    && ((c3 == 'H') || (c3 == 'h')))
  SH_flag = 1;
if (((c1 == 'Z') || (c1 == 'Z'))
    && ((c2 == 'S') || (c2 == 's'))
    && ((c3 == 'H') || (c3 == 'h')))
  SH_flag = 1;
if (((c1 == 'T') || (c1 == 't'))
```

```
      && ((c2 == 'C') || (c2 == 'c'))
      && ((c3 == 'S') || (c3 == 's'))
      && ((c4 == 'H') || (c4 == 'h')))
    SH_flag = 1;
  if (((c1 == 'T') || (c1 == 't'))          /* TCL/TK                     */
      && ((c2 == 'C') || (c2 == 'c'))
      && ((c3 == 'L') || (c3 == 'l')))
    TCL_flag = 1;
  if (((c1 == 'T') || (c1 == 't'))
      && ((c2 == 'K') || (c2 == 'k')))
    TCL_flag = 1;
  if (((c1 == 'M') || (c1 == 'm'))          /* MACSYMA, MAXIMA            */
      && ((c2 == 'A') || (c2 == 'a'))
      && ((c3 == 'C') || (c3 == 'c')))
    MAC_flag = 1;
  if (((c1 == 'M') || (c1 == 'm'))
      && ((c2 == 'A') || (c2 == 'a'))
      && ((c3 == 'X') || (c3 == 'x')))
    MAC_flag = 1;
  if (((c1 == 'M') || (c1 == 'm'))          /* MAPLE                      */
      && ((c2 == 'A') || (c2 == 'a'))
      && ((c3 == 'P') || (c3 == 'p')))
    MAP_flag = 1;
  if (((c1 == 'M') || (c1 == 'm'))
      && ((c2 == 'P') || (c2 == 'p'))
      && ((c3 == 'L') || (c3 == 'l')))
    MAP_flag = 1;
  if (((c1 == 'M') || (c1 == 'm'))          /* MATHEMATICA                */
      && ((c2 == 'A') || (c2 == 'a'))
      && ((c3 == 'T') || (c3 == 't')))
    MAT_flag = 1;
  if (((c1 == 'M') || (c1 == 'm'))
      && ((c2 == 'M') || (c2 == 'm'))
      && ((c3 == 'A') || (c3 == 'a')))
    MAT_flag = 1;
  if (((c1 == 'M') || (c1 == 'm'))          /* MATLAB, OCTAVE             */
      && ((c2 == 'L') || (c2 == 'l')))
    MLAB_flag = 1;
  if (((c1 == 'M') || (c1 == 'm'))
      && ((c2 == 'T') || (c2 == 't'))
      && ((c3 == 'L') || (c3 == 'l')))
    MLAB_flag = 1;
  if (((c1 == 'O') || (c1 == 'o'))
      && ((c2 == 'C') || (c2 == 'c'))
      && ((c3 == 'T') || (c3 == 't')))
    MLAB_flag = 1;
  if (((c1 == 'M') || (c1 == 'm'))          /* MuPAD                      */
      && ((c2 == 'U') || (c2 == 'u')))
    MPAD_flag = 1;
  if (((c1 == 'R') || (c1 == 'r'))          /* REDUCE                     */
      && ((c2 == 'E') || (c2 == 'e'))
      && ((c3 == 'D') || (c3 == 'd')))
    RED_flag = 1;
```

```
      if (((c1 == 'R') || (c1 == 'r'))
          && ((c2 == 'D') || (c2 == 'd'))
          && ((c3 == 'C') || (c3 == 'c')))
        RED_flag = 1;
      if ((c1 == 'M') || (c1 == 'm'))           /* M-file                                   */
        M_flag = 1;
    }
#ifdef DEBUGGING
  printf("init_lang_flag():\n");
  printf("TXT_flag  =%d\n", TXT_flag);
  printf("BAS_flag  =%d\n", BAS_flag);
  printf("C_flag    =%d\n", C_flag);
  printf("CBL_flag  =%d\n", CBL_flag);
  printf("F77_flag  =%d\n", F77_flag);
  printf("HTML_flag =%d\n", HTML_flag);
  printf("JAVA_flag =%d\n", JAVA_flag);
  printf("LISP_flag =%d\n", LISP_flag);
  printf("MAKE_flag =%d\n", MAKE_flag);
  printf("PAS_flag  =%d\n", PAS_flag);
  printf("PERL_flag =%d\n", PERL_flag);
  printf("SH_flag   =%d\n", SH_flag);
  printf("TCL_flag  =%d\n", TCL_flag);
  printf("ASR_flag  =%d\n", ASR_flag);
  printf("MAC_flag  =%d\n", MAC_flag);
  printf("MAP_flag  =%d\n", MAP_flag);
  printf("MAT_flag  =%d\n", MAT_flag);
  printf("MLAB_flag =%d\n", MLAB_flag);
  printf("MPAD_flag =%d\n", MPAD_flag);
  printf("RED_flag  =%d\n", RED_flag);
  printf("M_flag    =%d\n", M_flag);
#endif
}
```

/* ———————————————— set language flags ———————————————— */
/* If no language flags are set to 1, then `set_lang_flag()` tries to determine language type by reading input source program carefully. And also, `set_lang_flag()` decides whether or not the user knows the grammar of TeX. */

```
void set_lang_flag(buf_ptr)
int *buf_ptr;
{
    char *cptr;
    int flag, max_flag = 0;

    /* plain text file? or src2tex format file?                  */
    /* If all language flags are still equal to 0, then they are  */
    /* determined by searching reserved words of each language.   */
    if (TXT_flag == 0)
      set_txt_flag(buf_ptr);
    flag = BAS_flag;
    flag += C_flag;
    flag += CBL_flag;
    flag += F77_flag;
```

```
    flag += HTML_flag;
    flag += JAVA_flag;
    flag += LISP_flag;
    flag += MAKE_flag;
    flag += PAS_flag;
    flag += PERL_flag;
    flag += SH_flag;
    flag += TCL_flag;
    flag += ASR_flag;
    flag += MAP_flag;
    flag += MAC_flag;
    flag += MAT_flag;
    flag += MLAB_flag;
    flag += MPAD_flag;
    flag += RED_flag;
    flag += TXT_flag;
    /* set language flags and max_flag                                    */
    if (flag == 0)
      {
#ifdef LATEX
        fprintf(stderr,"src2latex: ");
#else
        fprintf(stderr,"src2tex: ");
#endif
        fprintf(stderr, "trying to identify language ");
        set_bas_flag(buf_ptr);
        fprintf(stderr, ".");
        if (BAS_flag > max_flag)
          max_flag = BAS_flag;
        set_c_flag(buf_ptr);
        fprintf(stderr, ".");
        if (C_flag > max_flag)
          max_flag = C_flag;
        set_cbl_flag(buf_ptr);
        fprintf(stderr, ".");
        if (CBL_flag > max_flag)
          max_flag = CBL_flag;
        set_f77_flag(buf_ptr);
        fprintf(stderr, ".");
        if (F77_flag > max_flag)
          max_flag = F77_flag;
        set_html_flag(buf_ptr);
        fprintf(stderr, ".");
        if (HTML_flag > max_flag)
          max_flag = HTML_flag;
        set_java_flag(buf_ptr);
        fprintf(stderr, ".");
        if (JAVA_flag > max_flag)
          max_flag = JAVA_flag;
        set_lisp_flag(buf_ptr);
        fprintf(stderr, ".");
        if (LISP_flag > max_flag)
          max_flag = LISP_flag;
```

```
set_make_flag(buf_ptr);
fprintf(stderr, ".");
if (MAKE_flag > max_flag)
  max_flag = MAKE_flag;
set_pas_flag(buf_ptr);
fprintf(stderr, ".");
if (PAS_flag > max_flag)
  max_flag = PAS_flag;
set_perl_flag(buf_ptr);
fprintf(stderr, ".");
if (PERL_flag > max_flag)
  max_flag = PERL_flag;
set_shell_flag(buf_ptr);
fprintf(stderr, ".");
if (SH_flag > max_flag)
  max_flag = SH_flag;
set_tcl_flag(buf_ptr);
fprintf(stderr, ".");
if (TCL_flag > max_flag)
  max_flag = TCL_flag;
set_asr_flag(buf_ptr);
fprintf(stderr, ".");
if (ASR_flag > max_flag)
  max_flag = ASR_flag;
set_mac_flag(buf_ptr);
fprintf(stderr, ".");
if (MAC_flag > max_flag)
  max_flag = MAC_flag;
set_map_flag(buf_ptr);
fprintf(stderr, ".");
if (MAP_flag > max_flag)
  max_flag = MAP_flag;
set_mat_flag(buf_ptr);
fprintf(stderr, ".");
if (MAT_flag > max_flag)
  max_flag = MAT_flag;
set_mlab_flag(buf_ptr);
fprintf(stderr, ".");
if (MLAB_flag > max_flag)
  max_flag = MLAB_flag;
set_mpad_flag(buf_ptr);
fprintf(stderr, ".");
if (MPAD_flag > max_flag)
  max_flag = MPAD_flag;
set_red_flag(buf_ptr);
fprintf(stderr, ".");
if (RED_flag > max_flag)
  max_flag = RED_flag;
fprintf(stderr, " done\n");
/* normalize language flags                                    */
if (BAS_flag == max_flag)
  BAS_flag = 1;
else
```

```
  BAS_flag = 0;
if (C_flag == max_flag)
  C_flag = 1;
else
  C_flag = 0;
if (CBL_flag == max_flag)
  CBL_flag = 1;
else
  CBL_flag = 0;
if (F77_flag == max_flag)
  F77_flag = 1;
else
  F77_flag = 0;
if (HTML_flag == max_flag)
  HTML_flag = 1;
else
  HTML_flag = 0;
if (JAVA_flag == max_flag)
  JAVA_flag = 1;
else
  JAVA_flag = 0;
if (LISP_flag == max_flag)
  LISP_flag = 1;
else
  LISP_flag = 0;
if (MAKE_flag == max_flag)
  MAKE_flag = 1;
else
  MAKE_flag = 0;
if (PAS_flag == max_flag)
  PAS_flag = 1;
else
  PAS_flag = 0;
if (PERL_flag == max_flag)
  PERL_flag = 1;
else
  PERL_flag = 0;
if (SH_flag == max_flag)
  SH_flag = 1;
else
  SH_flag = 0;
if (TCL_flag == max_flag)
  TCL_flag = 1;
else
  TCL_flag = 0;
if (ASR_flag == max_flag)
  ASR_flag = 1;
else
  ASR_flag = 0;
if (MAC_flag == max_flag)
  MAC_flag = 1;
else
  MAC_flag = 0;
```

```
    if (MAP_flag == max_flag)
      MAP_flag = 1;
    else
      MAP_flag = 0;
    if (MAT_flag == max_flag)
      MAT_flag = 1;
    else
      MAT_flag = 0;
    if (MLAB_flag == max_flag)
      MLAB_flag = 1;
    else
      MLAB_flag = 0;
    if (MPAD_flag == max_flag)
      MPAD_flag = 1;
    else
      MPAD_flag = 0;
    if (RED_flag == max_flag)
      RED_flag = 1;
    else
      RED_flag = 0;
  }
/* If sum of all language flags > 1,                                              */
/* i.e., if we cannot identify language type, then we reset all language flags to 0.  */
flag = BAS_flag;
flag += C_flag;
flag += CBL_flag;
flag += F77_flag;
flag += HTML_flag;
flag += JAVA_flag;
flag += LISP_flag;
flag += MAKE_flag;
flag += PAS_flag;
flag += PERL_flag;
flag += SH_flag;
flag += TCL_flag;
flag += ASR_flag;
flag += MAC_flag;
flag += MAP_flag;
flag += MAT_flag;
flag += MLAB_flag;
flag += MPAD_flag;
flag += RED_flag;
if (flag > 1)
  {
    BAS_flag = 0;
    C_flag = 0;
    CBL_flag = 0;
    F77_flag = 0;
    HTML_flag = 0;
    JAVA_flag = 0;
    LISP_flag = 0;
    MAKE_flag = 0;
    PAS_flag = 0;
```

```
        PERL_flag = 0;
        SH_flag = 0;
        TCL_flag = 0;
        ASR_flag = 0;
        MAC_flag = 0;
        MAP_flag = 0;
        MAT_flag = 0;
        MLAB_flag = 0;
        MPAD_flag = 0;
        RED_flag = 0;
      }
    /* Here, we simply identify ASIR with C ; actually, we don't use ASR_flag in text2tex.c file */
    C_flag += ASR_flag;
#ifdef DEBUGGING
    printf("set_lang_flag():\n");
    printf("TXT_flag  =%d\n", TXT_flag);
    printf("BAS_flag  =%d\n", BAS_flag);
    printf("C_flag    =%d\n", C_flag);
    printf("CBL_flag  =%d\n", CBL_flag);
    printf("F77_flag  =%d\n", F77_flag);
    printf("HTML_flag =%d\n", HTML_flag);
    printf("JAVA_flag =%d\n", JAVA_flag);
    printf("LISP_flag =%d\n", LISP_flag);
    printf("MAKE_flag =%d\n", MAKE_flag);
    printf("PAS_flag  =%d\n", PAS_flag);
    printf("PERL_flag =%d\n", PERL_flag);
    printf("SH_flag   =%d\n", SH_flag);
    printf("TCL_flag  =%d\n", TCL_flag);
    printf("ASR_flag  =%d\n", ASR_flag);
    printf("MAC_flag  =%d\n", MAC_flag);
    printf("MAP_flag  =%d\n", MAP_flag);
    printf("MAT_flag  =%d\n", MAT_flag);
    printf("MLAB_flag =%d\n", MLAB_flag);
    printf("MPAD_flag =%d\n", MPAD_flag);
    printf("RED_flag  =%d\n", RED_flag);
#endif
    /* display title, version and conversion type */
#ifdef LATEX
    fprintf(stderr,"src2latex: version ");
#else
    fprintf(stderr,"src2tex: version ");
#endif
    fprintf(stderr,VERSION);
#ifdef LATEX
    cptr = "(TEXT->LaTeX)";
    if (BAS_flag != 0)
      cptr = "(BASIC->LaTeX)";
    if (C_flag != 0)
      cptr = "(C,C++,OBJECTIVE-C->LaTeX)";
    if (CBL_flag != 0)
      cptr = "(COBOL->LaTeX)";
    if (HTML_flag != 0)
      cptr = "(HTML->LaTeX)";
```

```
    if (JAVA_flag != 0)
      cptr = "(JAVA->LaTeX)";
    if (LISP_flag != 0)
      cptr = "(LISP,SCHEME->LaTeX)";
    if (MAKE_flag != 0)
      cptr = "(MAKE->LaTeX)";
    if (F77_flag != 0)
      cptr = "(FORTRAN->LaTeX)";
    if (PAS_flag != 0)
      cptr = "(PASCAL->LaTeX)";
    if (PERL_flag != 0)
      cptr = "(PERL->LaTeX)";
    if (SH_flag != 0)
      cptr = "(SHELL->LaTeX)";
    if (TCL_flag != 0)
      cptr = "(TCL/TK->LaTeX)";
    if (ASR_flag != 0)
      cptr = "(ASIR->LaTeX)";
    if (MAC_flag != 0)
      cptr = "(MACSYMA,MAXIMA->LaTeX)";
    if (MAP_flag != 0)
      cptr = "(MAPLE->LaTeX)";
    if (MAT_flag != 0)
      cptr = "(MATHEMATICA->LaTeX)";
    if (MLAB_flag != 0)
      cptr = "(MATLAB,OCTAVE->LaTeX)";
    if (MPAD_flag != 0)
      cptr = "(MuPAD->LaTeX)";
    if (RED_flag != 0)
      cptr = "(REDUCE->LaTeX)";
#else
    cptr = "(TEXT->TeX)";
    if (BAS_flag != 0)
      cptr = "(BASIC->TeX)";
    if (C_flag != 0)
      cptr = "(C,C++,OBJECTIVE-C->TeX)";
    if (CBL_flag != 0)
      cptr = "(COBOL->TeX)";
    if (HTML_flag != 0)
      cptr = "(HTML->TeX)";
    if (JAVA_flag != 0)
      cptr = "(JAVA->TeX)";
    if (LISP_flag != 0)
      cptr = "(LISP,SCHEME->TeX)";
    if (MAKE_flag != 0)
      cptr = "(MAKE->TeX)";
    if (F77_flag != 0)
      cptr = "(FORTRAN->TeX)";
    if (PAS_flag != 0)
      cptr = "(PASCAL->TeX)";
    if (PERL_flag != 0)
      cptr = "(PERL->TeX)";
    if (SH_flag != 0)
```

```
      cptr = "(SHELL->TeX)";
    if (TCL_flag != 0)
      cptr = "(TCL/TK->TeX)";
    if (ASR_flag != 0)
      cptr = "(ASIR->TeX)";
    if (MAC_flag != 0)
      cptr = "(MACSYMA,MAXIMA->TeX)";
    if (MAP_flag != 0)
      cptr = "(MAPLE->TeX)";
    if (MAT_flag != 0)
      cptr = "(MATHEMATICA->TeX)";
    if (MLAB_flag != 0)
      cptr = "(MATLAB,OCTAVE->TeX)";
    if (MPAD_flag != 0)
      cptr = "(MuPAD->TeX)";
    if (RED_flag != 0)
      cptr = "(REDUCE->TeX)";
#endif
    fprintf(stderr," %s \n", cptr);
}
```