เอกสารอ้างอิง

ภาษาไทย

บุญเลิศ เอี่ยมทัศนา, ยืน ภู่วรวรรณ และ สมนึก คีรีโต. โปรแกรมคอมพิวเตอร์ภาษาซี. : ซีเอ็ดยูเคชั่น, กรุงเทพมหานคร, 2529.

ภาษาอังกฤษ

Ammeraal, L. Programs and Data Structures in C. : John Wiley & Son, Great Britain, 1987.

Baron, R.J., and Shapiro, L.G. Data Structures and their Implementation : Van Nostrad Reinhold, New York, 1980.

Esakov, J., and Weiss, T. Data Structures : An Advanced Approach Using C. : Prentice-Hall, USA., 1989.

Folk, M.J., and Zoellick, B. File Structures A Conceptual Toolkit. : Addison-Wesley, Canada, 1987.

Gethin, P. Full Text Information Storage and Retrieval. BRS Information Technologies Research. : March, 1987.

Harbron, T.R. File Systems : Structures and Algorithms. : Prentice-Hall, 1988.

Heaps, H.S. Information Retrieval : Computational and Theoretical Aspects. : Academic Press, USA., 1978.

Kruse, R.L. Data Structures and Program Design. : Prentice-Hall, USA., 1984, 1987.

_____, Leung, B.P., and Tondo, C.L. Data Structures and Program Design in C. : Prentice-Hall, USA., 1991.

Lafore, R. C Programming Using Turbo C++. : The Waite Group, USA., 1990.

Meadow, C.T. The Analysis of Information System : A Programmer's Introduction to Information Retrieval. : John Wiley & Sons, USA., 1967.

Salton, G. Automatic Text Processing : The Transformation, Analysis, and Retrieval of Information by Computer. : Addison-Wesley, USA., 1989.

_____, and McGrill, M.J. Introduction to Modern Information Retrieval. : MaGraw-Hill, Singapore, 1983.

Salzberg, B. File Structures : an Analytic Approach. : Prentice-Hall, 1988.

Tharp, A.L. File Organization and Processing. : John Wiley & Sons, Singapore, 1988.

ภาคผนวก

```
#include <process.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#define  M 10
#define  MM (2*M)
#define  NAME 8
#define  MAX 81
#define  MAXWORD 16
#define  MLOC 5
#define  NIL (-1L)
#define  blank 160
#define  space ' '
#define  eoln '\n'
#define  nul '\0'
#define  dirname "DIRECTOR"
#define  extdict ".DIC"
#define  extindex ".INX"
#define  dot '.'
#define  dochead "dh"
#define  paratext "p"
#define  eng_zero 0x30
#define  eng_nine 0x39
#define  first_eng 0x41
#define  last_eng 0x5A
#define  fst_eng 0x61
#define  lst_eng 0x7A
#define  fst_thai 0xA1
#define  lst_thai 0xE4
#define  thai_zero 0x90
#define  thai_nine 0x99

struct   Directory_File
{        char fname[NAME];
         char desc[MAX];
};
typedef  struct  Directory_File  DFIL;

struct   Text_Index
{        long docadd;
         char title[MAX];
};
typedef  struct  Text_Index INDX;

struct   Link_List
{        long begin;
         int  docno;
         int  parano;
         int  wordno;
         long next;
};
typedef  struct  Link_List LINK;
```

```
struct   node
{        int  cnt;
         char key[MM][MAXWORD];
         int  occur[MM];
         long list[MM];
         long ptr[MM+1];
};
typedef  struct node NODE;

NODE  rootnode;
FILE  *fpdir, *fptext, *fpdict, *fpindex, *fpsrc, *fpdes;
int   ctindex, count = 0;
long  start, root = NIL, ptindex, pointer = 0L;

create();
int  opencfile();
insert(char*, long);
int  ins(char*, long, long, char**, int*, long*, long*);
int  binsearch(char*, char*[], int);
search();
int  opensfile();
printlocat(long, int);
printtitle(long, int);
printpara(long, int);
notfound(char*);
appendtext();
printdir();
update();
closefile();
long getdir();
writedir(long, DFIL*);
long getindex();
readindex(long, INDX*);
writeindex(long, INDX*);
rdstindex();
wrstindex();
long getlist();
readlist(long, LINK*);
writelist(long, LINK*);
long  getnode();
readnode(long, NODE*);
writenode(long, NODE*);
rdstart();
wrstart();
error(char*);

main()
{  char ch;
   clrscr();
   while(1)
   {  printf("\n    ะฟท'ศระก\n\n");
      printf("1   วมแฮคฎฮทขแสฟ'ร\n");
      printf("2   ขแทศธขแสฟ'ร\n");
      printf("3   ะผ*เฟขแสฟ'ร\n");
      printf("4   ผ*ฟผ mฮภจะเส ผแฟขแสฟ'ร\n");
```

```
      printf("5   นมะธนม.ค.ฝแฟขแสฟ'ราฮมธะช\n");
      printf("0   ฆธกธมคๆคฮท\n\n");
      printf("     านมฆะระสกขแสขฤฮฟค'เฒแสคกธม : ");
      do
      {  ch = getchar();
         ch = toupper(ch);
      }  while (ch != '1' && ch != '2' && ch != '3' && ch != '4'
               && ch != '5' && ch != '0');
      if (ch == '0')   {  clrscr();  break;  }
      clrscr();
      switch (ch)
      {  case '1' : create();
                    count = 0;  root = NIL;  pointer = 0L;
                    printf("\nกฎฆน.เฟาฆ.ะผะเสคๆคฮทฌเส\n");
                    getch();
                    clrscr();
                    break;
         case '2' : search();
                    clrscr();
                    break;
         case '3' : appendtext();
                    printf("\nกฎฆน.เฟาฆ.ะผะเสคๆคฮทฌเส\n");
                    getch();
                    clrscr();
                    break;
         case '4' : printdir();
                    printf("\nกฎฆน.เฟาฆ.ะผะเสคๆคฮทฌเส\n");
                    getch();
                    clrscr();
                    break;
         case '5' : update();
                    clrscr();
                    break;
      }
   }
}

create()
{  char word[MAXWORD];
   LINK address;
   INDX index;
   long add, ind, fpt;
   int  code, hflag = 0, wflag = 0, ch, m = 0, i = 0, j;
   int  countword = 0, countpara = 0, countdoc;

   code = opencfile();
   if (code == 0)  return;
   if (fptext != NULL)
   {  countdoc = count;
      if (fseek(fptext, pointer, 0))  error("fseek in creat 1");
      while(1)
      {  ch = getc(fptext);
         if (ch == dot)   hflag = 1;
         if ((ch >= eng_zero) && (ch <= eng_nine) || (ch >= first_eng)
            && (ch <= last_eng) || (ch >= fst_eng) && (ch <= lst_eng)
```

```
   || (ch >= fst_thai) && (ch <= lst_thai)
   || (ch >= thai_zero) && (ch <= thai_nine))
{
   wflag = 1;
   word[i++] = ch;
}
if ((ch == blank) || (ch == space) || (ch == eoln) || (ch == EOF))
{  word[i] = nul;
   if (hflag)
   {  /* If document header, store address of document
         and document title in index file.            */
      /* Document title is the first line of document
         and must be less than 80 chars.              */

      if (strncmp(word,dochead,2) == 0)
      {  index.docadd = ftell(fptext);
         address.begin = ftell(fptext);
         fpt = index.docadd;
         while (((index.title[m++] = getc(fptext)) != eoln)
               && (m < MAX-1));
         index.title[m] = nul;
         ind = getindex();
         writeindex(ind, &index);
         countdoc++;
         countword = 0;
         countpara = 0;
         printf("\nะสกาฮมด·เ %d จะเสะมะเสค : %s\n",countdoc,index.title);
         for (m = 0; m < MAX; m++) index.title[m] = space;
         m = 0;

         /* Seek backward to the beginning of document title
            to scan word of title.   */
         if (fseek(fptext, fpt, 0))  error("fseek in creat 2");
      }

      if (strncmp(word,paratext,1) == 0)
      {  address.begin = ftell(fptext);
         countpara++;
         countword = 0;
      }
      hflag = 0;  wflag = 0;
   }  /* end if (hflag) */

   if (wflag)
   {  /* If word, store word in B-Tree
         and store address in link list. */

      countword++;
      address.docno = countdoc;
      address.parano = countpara;
      address.wordno = countword;
      address.next = NIL;
      add = getlist();
      writelist(add,&address);
      insert(word,add);
```

```
                wflag = 0;
            }  /* end if (wflag) */

            i = 0;
            for (j = 0; j < strlen(word); j++)   word[j] = space;
            if (ch == EOF)   break;
         }  /* end if (blank) */
      }  /* end while */

      /*  Update the first record of index file and dictionary file. */
      count = countdoc;
      pointer = ftell(fptext);
      wrstindex();
      wrstart();
   }
   closefile();
}

int  opencfile()
{  DFIL direct;
   char filename[NAME], sname[NAME+4], descript[MAX];
   long d;
   int  dup = 0;

   if ((fpdir = fopen(dirname, "rb+")) == NULL)
      fpdir = fopen(dirname, "wb+");
   printf("\nก ้าาเจะเส฿ฝแฟฃนสฟ๋รฃนสฃฦฮฟ : ");
   scanf("%s",filename);
   if ((fptext = fopen(filename, "rb")) == NULL)
      {  printf("\nFile not found\n");
         return 0;
      }
   printf("\n");
   printf("้าเฃ๊สก๋ธฮภาะแฑ฿ะก'เภฦกะธ฿ฝแฟฃนสฟ๋รฃนสฃฦฮฟ (฿ฟเะก๋ฑ 80 ฆะฦสะกฦม)\n");
   flushall();
   gets(descript);
   printf("\n");

   while (fread(&direct, sizeof(DFIL), 1, fpdir) != 0)
      if (strcmp(filename, direct.fname) == 0)
      {  dup = 1;
         break;
      }
   if (!dup)
   {  strcpy(direct.fname, filename);
      strcpy(direct.desc, descript);
      d = getdir();
      writedir(d, &direct);
   }
   strcpy(sname, filename);
   strcat(sname, extdict);
   fpdict = fopen(sname, "rb+");
   if (fpdict == NULL)
   {  fpdict = fopen(sname, "wb+");
      wrstart();
```

```
   }
   else  rdstart();

   strcpy(sname, filename);
   strcat(sname, extindex);
   if ((fpindex = fopen(sname, "rb+")) == NULL)
   {  fpindex = fopen(sname, "wb+");
      wrstindex();
   }
   else  rdstindex();
   return 1;
}

insert(char *word, long add)
{  /*  Driver function for node insertion, called only in the
       function create.  Most of the work is delegated to 'ins'.
   */
   long  tnew, u, getnode();
   int   code, onew;
   char  *wordnew;
   long  anew;

   code = ins(word, add, root, &wordnew, &onew, &anew, &tnew);
   if (code)  return;    /*  Code = 1 or 2 successful insert.  */

   /*  Code = 0 : first key is inserted,
            or :  new root node is created.   */

   u = getnode();
   rootnode.cnt = 1;
   strcpy(rootnode.key[0],wordnew);
   rootnode.occur[0] = onew;
   rootnode.list[0] = anew;
   rootnode.ptr[0] = root;
   rootnode.ptr[1] = tnew;
   root = u;
   writenode(u, &rootnode);
}

int  ins(char *word, long add, long t, char **y, int *v, long *x, long *u)
{  /* Insert word in B-tree with root t. If not completely
      successful, the integer *y and the pointer *u
      remain to be inserted.
      Returned value :
          0 if insertion not completely successful,
          1 if insertion successful,
          2 if word is already present in B-tree.  */

   long tnew, getnode(), p_final, *p, a_final, *a, anew, a_temp;
   int  c, i, j, *n, code, *o, o_final, onew;
   char *wordnew, k_final[MAXWORD], *k[MM];
   NODE nod, newnod;
   LINK add_temp;

   /* Examine whether t is a pointer field in a laef :  */
```

```
if (t == NIL)
{  *u = NIL;
   *y = word;
   *v = 1;
   *x = add;
   return 0;
}
readnode(t, &nod);
n = &nod.cnt;
o = nod.occur;
a = nod.list;
p = nod.ptr;
for (c = 0; c < MM; c++)   k[c] = nod.key[c];

/* Select pointer p[i] and try to insert word in
   the subtree of which p[i] is the root ;      */
/* "i" is the position to insert new item. */

i = binsearch(word, k, *n);

if (i < *n && strcmp(word,k[i]) == 0)
{  /* Duplicate key, create link list of word's location. */

   o[i]++;
   readlist(a[i], &add_temp);
   a_temp = a[i];
   while (add_temp.next != NIL)
   {  a_temp = add_temp.next;
      readlist(add_temp.next, &add_temp);
   }
   add_temp.next = add;
   writelist(a_temp, &add_temp);
   writenode(t, &nod);
   return 2;
}
code = ins(word, add, p[i], &wordnew, &onew, &anew, &tnew);
if (code)  return code;

/* Insertion in subtree did not completely succeed;
   try to insert wordnew and tnew in the current node :   */

if (*n < MM)
{  i = binsearch(wordnew, k, *n);
   for (j = *n; j > i; j--)   /*  Shift item one position to the right. */
   {  strcpy(k[j],k[j-1]);
      o[j] = o[j-1];
      a[j] = a[j-1];
      p[j+1] = p[j];
   }
   strcpy(k[i],wordnew);     /*  Insert new item at ith position. */
   o[i] = onew;
   a[i] = anew;
   p[i+1] = tnew;
   ++*n;
```

```
   writenode(t, &nod);
   return 1;
}

/* The current node was already full, so split it. Pass item k[m] in
   the middle of the augmented sequence back through parameter y,
   so that it can move upward in the tree. Also, pass a pointer to
   the newly created node back through u. Return 0, to report that
   insertion was not completed:
*/

if (i == MM)                    /* Insert new item to k_final. */
{  strcpy(k_final,wordnew);
   o_final = onew;
   a_final = anew;
   p_final = tnew;
}
else
{  /* Copy the last item to k_final, and shift item one position
      to the right to insert new item at ith position. */

   strcpy(k_final,k[MM-1]);
   o_final = o[MM-1];
   a_final = a[MM-1];
   p_final = p[MM];
   for (j = MM-1; j > i; j--)
   {  strcpy(k[j],k[j-1]);
      o[j] = o[j-1];
      a[j] = a[j-1];
      p[j+1] = p[j];
   }
   strcpy(k[i],wordnew);        /*  Insert new item at the ith position. */
   o[i] = onew;
   a[i] = anew;
   p[i+1] = tnew;
}
/*  Move the middle item upward to insert in the parent node. */
/*  And split current node into two node.                     */

strcpy(*y,k[M]);
*v = o[M];
*x = a[M];
*n = M;
*u = getnode();
newnod.cnt = M;
for (j = 0; j < M-1; j++)     /*  Copy the second half to the newnode. */
{  strcpy(newnod.key[j],k[j+M+1]);
   newnod.occur[j] = o[j+M+1];
   newnod.list[j] = a[j+M+1];
   newnod.ptr[j] = p[j+M+1];
}
newnod.ptr[M-1] = p[MM];
strcpy(newnod.key[M-1],k_final);  /*  Copy item from k_final to newnode. */
newnod.occur[M-1] = o_final;
newnod.list[M-1] = a_final;
```

```
   newnod.ptr[M] = p_final;
   writenode(t, &nod);
   writenode(*u, &newnod);
   return 0;
}

int  binsearch(char *word, char *a[], int n)
{  /* Search array a[0], a[1], ..., a[n-1] for word.          */
   /* Return value : 0 if word <= a[0], n if word > a[n-1],  */
   /*                or r, where  a[r-1] < word <= a[r]      */
   /* Shift right 1 bit is the same as divide by 2           */

   int i, left, right;
   if (strcmp(word,a[0]) <= 0)   return 0;
   if (strcmp(word,a[n-1]) > 0)  return n;
   left = 0; right = n-1;
   while (right - left > 1)
   {  i = (right + left) >> 1;
      if (strcmp(word,a[i]) <= 0)  right = i;  else left = i;
   }
   return right;
}

search()
{  int  i, n, c, found, fsl, fwd, code, count;
   char word[MAXWORD], str[MAX], *k[MM], *ch;
   NODE nod;
   long t;

   code = opensfile();
   if (code == 0)
   {  printf("\nกดพิมพ์ไฟล์ที่จะเรียกดูหรือสร้างใหม่\n");
      getch();
      return;
   }
   count = 0;
   printf("\n");
   while (1)
   {  code = 0;  fwd = 0;
      t = root;  found = 0;
      printf("%d : ",++count);
      flushall();
      gets(str);

      /*  "str" may be command or word,
          if str is command it must be ".p" or ".q",
          and command ".p" must be   .p lo/word
                                  or .p ti/word
                                  or .p pa/word
          if str is not command it must be word.     */

      ch = str;
      while (strncmp(ch, nul, 1) != 0)
      {  if (strncmp(ch, " ", 1) != 0)  break;
         ch = ch+1;
```

```
}
if (ch[0] == dot)
{  if (ch[1] == 'q')  break;
   switch (ch[1])
   {  case 'p' : fsl = 0;
                 ch = ch+2;
                 while (strncmp(ch, nul, 1) != 0)
                 {  if (strncmp(ch, " ", 1) != 0)  break;
                    ch = ch+1;
                 }
                 if (strncmp(ch, "lo", 2) == 0)  code = 1;
                 if (strncmp(ch, "ti", 2) == 0)  code = 2;
                 if (strncmp(ch, "pa", 2) == 0)  code = 3;
                 ch = ch+2;
                 while (strncmp(ch, nul, 1) != 0)
                 {  if (strncmp(ch, "/", 1) == 0)  { fsl = 1; break; }
                    ch = ch+1;
                 }
                 ch = ch+1;
                 if ((strlen(ch) != 0) && (code) && (fsl))
                 {  fwd = 1;
                    while (strncmp(ch, nul, 1) != 0)
                    {  if (strncmp(ch, " ", 1) != 0)  break;
                       ch = ch+1;
                    }
                    strcpy(word, ch);
                 }
                 else  printf("\nขววาะเค.ฟเค'กฒแสด\n\n");
                 break;
      default :  printf("\nขววาะเค.ฟเค'กฒแสด\n\n");
   }
}
else
   if (strlen(ch) != 0)
   {  fwd = 1;
      strcpy(word, ch);
   }
if (fwd)
{  while (t != NIL)        /* Traverse B-Tree to search word. */
   {  readnode(t, &nod);
      for (c = 0; c < MM; c++)   k[c] = nod.key[c];
      n = nod.cnt;
      i = binsearch(word, k, n);
      if (i < n && strcmp(word,k[i]) == 0)
      {  found = 1;
         switch (code)
         {  case 1 : printlocat(t,i);
                     break;
            case 2 : printtitle(t,i);
                     break;
            case 3 : printpara(t,i);
                     break;
           default : readnode(t, &nod);
                     printf("\n%s นมธกญ %d ขมะนค",nod.key[i],
                               nod.occur[i]);
```

```
                    printf("\n\n");
                } /* end switch */
                break;
             }
             t = nod.ptr[i];
          }
          if ((t == NIL) && (!found))  notfound(word);
       }
    }
    closefile();
}

int opensfile()
{  char filename[NAME], sname[NAME+4];

   printf("\nกาวเจะเส.ฝแพขแสฬ'ร : ");
   scanf("%s",filename);
   if ((fptext = fopen(filename, "rb")) == NULL)
   {  printf("\nCan not open %s file or File not found\n", filename);
      return 0;
   }

   strcpy(sname, filename);
   strcat(sname, extdict);
   if ((fpdict = fopen(sname, "rb")) == NULL)
   {  printf("\nCan not open  DICTIONARY file or File not found\n");
      return 0;
   }
   rdstart();

   strcpy(sname, filename);
   strcat(sname, extindex);
   if ((fpindex = fopen(sname, "rb")) == NULL)
   {  printf("\nCan not open INDEX file or File not found\n");
      return 0;
   }
   rdstindex();
   pointer = ftell(fpindex);
   return 1;
}

printlocat(long  t, int i)
{  NODE nod;
   LINK add;
   int count;

   readnode(t, &nod);
   printf("\nฉ๊ว.ศทเค : ศฬฮกะรขะสกาฮม ศฬฮกะรขฒสท ศฬฮกะรขซ๊\n");
   add.next = nod.list[i];
   count = 0;
   printf("\n\t");
   do
   {  readlist(add.next, &add);
      count++;
      if (count > MLOC)  {  printf("\n\t");  count = 0;  }
```

```
      printf("%d %d %d ; ", add.docno, add.parano, add.wordno);
   } while(add.next != NIL);
   printf("\n\n");
}

printtitle(long t, int i)
{  NODE nod;
   LINK add;
   INDX index;
   long offset;
   int  dup = 0;

   readnode(t, &nod);
   add.next = nod.list[i];
   printf("\n");
   do
   {  readlist(add.next, &add);
      if (add.docno != dup)
      {  offset = pointer + (add.docno - 1) * sizeof(INDX);
         readindex(offset, &index);
         printf("เอกสารหมายเลข %d ชื่อเรื่องคือ : %s\n",add.docno, index.title);
         dup = add.docno;
      }
   } while(add.next != NIL);
   printf("\n");
}

printpara(long t, int i)
{  NODE nod;
   LINK add;
   int  n, ch, j = 0, ddup = 0, pdup = 0;
   char word[MAX], ans;

   readnode(t, &nod);
   add.next = nod.list[i];
   do
   {  readlist(add.next, &add);
      if ((add.docno != ddup) || (add.parano != pdup))
      {  if (fseek(fptext, add.begin, 0))  error("fseek in printpara");
         for (n = 0; n < strlen(word); n++)  word[n] = space;
         j = 0;
         printf("\n");
         while(1)
         {  ch = getc(fptext);
            if ((ch != eoln) && (ch != EOF) && (j <= MAX-MAXWORD))
               word[j++] = ch;
            else
            {  if (j > MAX-MAXWORD)
               while (1)
               {  if ((ch == blank) || (ch == space) ||
                      (ch == eoln) || (ch == EOF))  break;
                  word[j++] = ch;
                  ch = getc(fptext);
               }
               word[j] = nul;
```

```
            if ((strncmp(word+1, dochead, 2) == 0) ||
                (strncmp(word+1, paratext, 1) == 0))  break;
            printf("%s\n",word);
            for (n = 0; n < strlen(word); n++)  word[n] = space;
            j = 0;
          }
          if (ch == EOF)   break;
        } /* end while */
        printf("\nพบแสดงอีกหรือไม่ [Y/N] : ");
        do
        { ans = getchar();
          ans = toupper(ans);
        } while (ans != 'Y' && ans != 'N');
        if (ans == 'N')  break;
      }
      ddup = add.docno;
      pdup = add.parano;
    } while(add.next != NIL);
    printf("\n");
}

notfound(char *word)
{ printf("\n%s ไม่มีอยู่ในแฟ้ม\n\n", word);
}

appendtext()
{ char  string[MAX], filename[NAME];

   printf("\nท่านจะเสนอแฟ้มข้อมูลที่ต้องการผนวก : ");
   scanf("%s",filename);
   if ((fpsrc = fopen(filename,"rb+")) == NULL)
   { printf("\nfile not found\n");
     return 0;
   }
   printf("\nท่านจะเสนอแฟ้มข้อมูลที่จะผนวก : ");
   scanf("%s",filename);
   if ((fpdes = fopen(filename,"rb")) == NULL)
   { printf("\nfile not found\n");
     return 0;
   }
   if (fseek(fpsrc, 0L, 2))   error("fseek in appendtext");
   flushall();
   while (fgets(string,MAX,fpdes) != NULL)
        fputs(string,fpsrc);
   fclose(fpsrc);
   fclose(fpdes);
}

printdir()
{ DFIL  direct;
   if ((fpdir = fopen(dirname, "rb+")) == NULL)
     fpdir = fopen(dirname, "wb+");
   printf("\nมีอักษรที่ขึ้นต้นด้วยตัวใดบ้างที่ต้องการ\n\n");
   while (fread(&direct, sizeof(DFIL), 1, fpdir) != 0)
     printf("\t%s : %s\n",direct.fname, direct.desc);
```

```
   printf("\n");
   fclose(fpdir);
}

update()
{  DFIL  direct;
   FILE  *fptem;
   char  filename[NAME], descript[MAX], *ch, str[MAX];
   int   dup, count;
   long  d;

   count = 0;
   printf("\n");
   while (1)
   {  printf("%d : ",++count);
      flushall();
      gets(str);

      /*  command must be ".p" or ".i", or ".d",or ".q"
              ".p" : print content of directory
              ".i" : insert new database name into directory
              ".d" : delete database name from directory
              ".q" : quit
      */
      ch = str;
      while (strncmp(ch, nul, 1) != 0)
      {  if (strncmp(ch, " ", 1) != 0)  break;
         ch = ch+1;
      }
      if (ch[0] == dot)
      {  if (ch[1] == 'q')  {  clrscr();  break;  }
         switch (ch[1])
         {  case 'p' : printdir();
                       break;

            case 'i' : dup = 0;
                       if ((fpdir = fopen(dirname, "rb+")) == NULL)
                          fpdir = fopen(dirname, "wb+");
                       printf("\nกาาเจะเส.ฝนแฟซนแสฟ'ร : ");
                       scanf("%s",filename);
                       while (fread(&direct, sizeof(DFIL), 1, fpdir) != 0)
                          if (strcmp(filename, direct.fname) == 0)
                          {  dup = 1;
                             printf("\nFile has been already inserted\n\n");
                             break;
                          }
                       if (!dup)
                       {  printf("\n");
                          printf("าาเขวสก*ธฮภาะแท๛ะก'เภฤกะธ.ฝนแฟซนแสฟ'ร");
                          printf("(.ฟเะก*ฑ 80 ฒะฤสะกฦม)\n");
                          flushall();
                          gets(descript);
                          printf("\n");
                          strcpy(direct.fname, filename);
                          strcpy(direct.desc, descript);
```

```
                    d = getdir();
                    writedir(d, &direct);
                }
                fclose(fpdir);
                break;

        case 'd' : if ((fptem = fopen("DIRECTOR.BAK", "rb")) != NULL)
                    system("del DIRECTOR.BAK");
                if ((fptem = fopen("DIRECTOR", "rb")) == NULL)
                {  printf("\nDIRECTORY file not found\n\n");
                    break;
                }
                system("ren DIRECTOR, DIRECTOR.BAK");
                if ((fptem = fopen("DIRECTOR.BAK", "rb")) != NULL)
                fpdir = fopen(dirname, "wb+");
                printf("\nกำหนดชื่อแฟ้มข้อมูลที่ต้องการลบ : ");
                scanf("%s",filename);
                printf("\n");
                while (fread(&direct, sizeof(DFIL), 1, fptem) != 0)
                    if (strcmp(filename, direct.fname) != 0)
                        fwrite(&direct, sizeof(DFIL), 1, fpdir);
                fclose(fptem);
                fclose(fpdir);
                break;

        default :  printf("\nขออภัยเครื่องพิมพ์ไม่พร้อมใช้งาน\n\n");
      }  /* end switch */
    } /* end ch[0] = dot */
    else  printf("\nขออภัยเครื่องพิมพ์ไม่พร้อมใช้งาน\n\n");
  } /* end while */
}

closefile()
{  flushall();
   fclose(fpdir);
   fclose(fptext);
   fclose(fpdict);
   fclose(fpindex);
}

long  getdir()
{  long d;
   DFIL direct;
   if (fseek(fpdir, 0L, 2))  error("fseek in getdir");
   d = ftell(fpdir);
   writedir(d, &direct);
   return d;
}

writedir(long d, DFIL *direct)
{  if (fseek(fpdir, d, 0))   error("fseek in writedir");
   if (fwrite(direct, sizeof(DFIL), 1, fpdir) == 0)
      error("fwrite in writedir");
}
```

```
long  getindex()
{  long i;
   INDX index;
   if (fseek(fpindex, 0L, 2))  error("fseek in getindex");
   i = ftell(fpindex);
   writeindex(i, &index);
   return i;
}

readindex(long i, INDX *index)
{  if (fseek(fpindex, i, 0))   error("fseek in readindex");
   if (fread(index, sizeof(INDX), 1, fpindex) == 0)
      error("fread in readindex");
}

writeindex(long i, INDX *index)
{  if (fseek(fpindex, i, 0))   error("fseek in writeindex");
   if (fwrite(index, sizeof(INDX), 1, fpindex) == 0)
      error("fwrite in writeindex");
}

rdstindex()
{  if (fseek(fpindex, 0L, 0))  error("fseek in rdstindex");
   if (fread(&ctindex, sizeof(int), 1, fpindex) == 0)
      error("fread in rdstindex 1");
   count = ctindex;
   if (fread(&ptindex, sizeof(long), 1, fpindex) == 0)
      error("fread in rdstindex 2");
   pointer = ptindex;
}

wrstindex()
{  ctindex = count;
   ptindex = pointer;
   if (fseek(fpindex, 0L, 0))  error("fseek in wrstindex");
   if (fwrite(&ctindex, sizeof(int), 1, fpindex) == 0)
      error("fwrite in wrstindex 1");
   if (fwrite(&ptindex, sizeof(long), 1, fpindex) == 0)
      error("fwrite in wrstindex 2");
}

long  getlist()
{  long a;
   LINK address;
   if (fseek(fpdict, 0L, 2))  error("fseek in getlist");
   a = ftell(fpdict);
   writelist(a, &address);
   return a;
}

readlist(long a, LINK *address)
{  if (fseek(fpdict, a, 0))   error("fseek in readlist");
   if (fread(address, sizeof(LINK), 1, fpdict) == 0)
      error("fread in readlist");
}
```

```
writelist(long a, LINK *address)
{  if (fseek(fpdict, a, 0))   error("fseek in writelist");
   if (fwrite(address, sizeof(LINK), 1, fpdict) == 0)
      error("fwrite in writelist");
}

long  getnode()
{  long t;
   NODE nod;
   if (fseek(fpdict, 0L, 2))   error("fseek in getnode");
   t = ftell(fpdict);
   writenode(t, &nod);  /* Reserve space on disk */
   return t;
}

readnode(long t, NODE *pnode)
{  if (t == root)
   {  *pnode = rootnode;
      return;
   }
   if (fseek(fpdict, t, 0))   error("fseek in readnode");
   if (fread(pnode, sizeof(NODE), 1, fpdict) == 0)
      error("fread in readnode");
}

writenode(long t, NODE *pnode)
{  if (t == root)    rootnode = *pnode;
   if (fseek(fpdict, t, 0))   error("fseek in writenode");
   if (fwrite(pnode, sizeof(NODE), 1, fpdict) == 0)
      error("fwrite in writenode");
}

rdstart()
{  if (fseek(fpdict, 0L, 0))   error("fseek in rdstart");
   if (fread(&start, sizeof(long), 1, fpdict) == 0)
      error("fread in rdstart");
   readnode(start, &rootnode);
   root = start;
}

wrstart()
{  start = root;
   if (fseek(fpdict, 0L, 0))   error("fseek in wrstart");
   if (fwrite(&start, sizeof(long), 1, fpdict) == 0)
      error("fwrite in wrstart");
   if (root != NIL)   writenode(root, &rootnode);
}

error(char *str)
{  printf("\nError : %s\n",str);
   exit(1);
}
```

ประวัติผู้เขียน

นางสาว พรทิพย์ บัวสาม เกิดวันที่ 28 มิถุนายน พ.ศ. 2506 ที่อำเภอ ท่าศาลา จังหวัด นครศรีธรรมราช สำเร็จการศึกษาปริญญาตรีวิทยาศาสตรบัณฑิต สาขาคณิตศาสตร์ ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ ในปี 2529 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ. 2531 ปัจจุบันรับราชการที่ วิทยาลัยครูเพชรบุรี อำเภอเมือง จังหวัดเพชรบุรี