DT-900
高速ファイルサーチライブラリ

ユーザマニュアル

Version 1.00

CASIO

カシオ計算機株式会社

CASIO

目次

CASIO

I. 概要

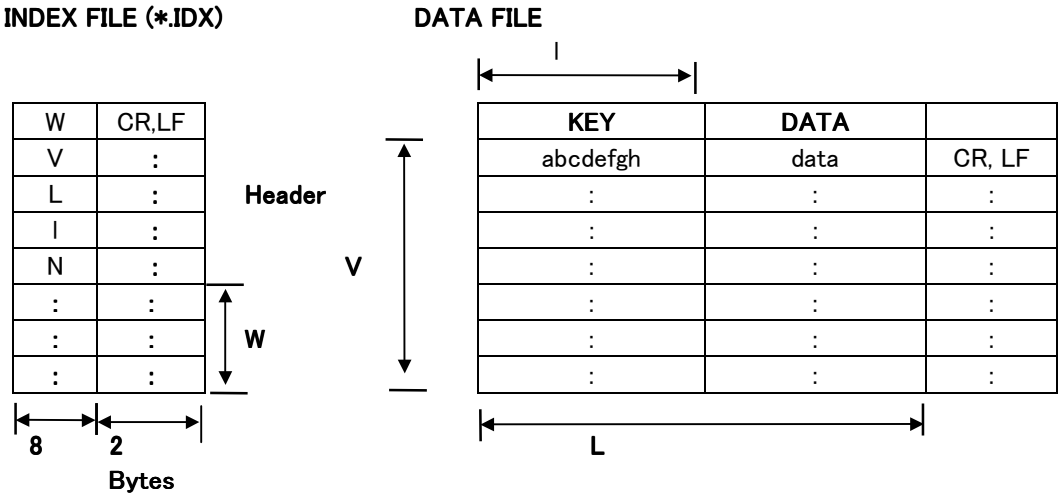
A. はじめに

本高速ファイル検索ライブラリは、HASH法を使用した高速ファイル検索機能セットです。
HASH法とは、キーデータを数値に変換した特別なインデックスファイルを用い、データレコード検索時間を最小限にする技法です。
本ライブラリを用いることにより、特に大容量ファイルを扱う場合、従来の逐次検索に比べ、高速にファイルを検索することができます。

B. 機能

機能名	説明
HashAssign	データファイルからインデックスファイルを生成します。
HashRead	入力されたキーに該当するデータをデータファイルから読み出します。
HashWrite	入力されたキーに該当するデータの内容を書き換えます。
HashAdd	データファイルにデータを追加します。 （同時にインデックスファイルも更新します）

II. ファイル構造



A. 用語説明

1. I : キーフィールドの長さ (Bytes).
2. L : データレコード長 (KEY+DATA)
3. V : 総データレコード数
4. W : 総インデックス数
   インデックスの衝突を避けるために、最低でも
   W≧1.2×V が必要です。
   WをVの2～3倍程度確保すると、より効率の良いファイルが構築できます。
5. N : 使用レコード数

B. HASH関数

キーデータからインデックスを求めるために、HASH関数を使用します。
HASH関数はライブラリ内で自動的に実行されるため、ユーザが意識する必要はありません。
仮にキーが 'abcdefgh' の場合、HASH関数は以下のようになります。
（内部的に、キーは 'abcdefgh00000' のように、自動桁数補正されます）

```
f(x) =  MOD { [square(a*b*c*d) + square(b*c*d*e) + square(c*d*e*f)
              + square(d*e*f*g)  + square(e*f*g*h) + square(f*g*h*0)
              + square(g*h*0*0) + square(h*0*0*0) ]  /  (W - 1) }
```

C. 注意事項

1. データファイルは、上記フォーマットに従い、ユーザ側で作成しなければなりません。
2. データレコードは、データファイルの先頭から順に格納されていなければなりません。
3. キーは、ユニークでなければなりません。
4. インデックスファイルおよびデータファイルのOPEN／CLOSEは、ユーザ側で行ってください。
5. DT900のドライブ（特にBドライブ）は、デバイスの特性上、書き込み速度が遅いためHASHREAD関数以外（特にHASHASSIGN関数）は、PC上で実行することを強く推奨します。

III. HASHライブラリとアプリケーション

本ライブラリは、以下の環境下で動作するものを提供します。

- DT-900
- PC running MS DOS
- PCs running Windows 95.

以下にHASHライブラリとアプリケーションの関係を記します。

	DT900 Lib	PC 16 Bit Lib	PC 32 Bit Lib
DT-900	✓	×	×
MS DOS ｱﾌﾟﾘｹｰｼｮﾝ			
Small model	×	✓	×
Medium model	×	✓	×
Large model	×	✓	×
MS DOS ｱﾌﾟﾘｹｰｼｮﾝ (Win 3.11下で動作)			
Small ﾓﾃﾞﾙ	×	✓	×
Medium ﾓﾃﾞﾙ	×	✓	×
Large ﾓﾃﾞﾙ	×	✓	×
Windows 95 ｺﾝｿｰﾙ ｱﾌﾟﾘｹｰｼｮﾝ	×	×	✓
Windows 3.1, 3.11 ｱﾌﾟﾘｹｰｼｮﾝ (16 Bit)			
Small ﾓﾃﾞﾙ	×	×	×
Medium ﾓﾃﾞﾙ	×	×	×
Large ﾓﾃﾞﾙ	×	×	×
Windows 95 ｱﾌﾟﾘｹｰｼｮﾝ	×	×	×
ﾗｲﾌﾞﾗﾘ格納ﾃﾞｨﾚｸﾄﾘ	¥DT900	¥PC16BIT	¥PC32BIT

Ⅳ.　HASHライブラリ（DOS用および DT-900用）

本章では、以下のライブラリに関して説明します。

- DT-900用ライブラリ
- PC 16 Bit用ライブラリ
- PC 32 Bit用ライブラリ

A. iHashAssign

1. 概要

本関数は、データファイルからインデックスファイルを作成します。

2. コーリングシーケンス

```
int  iHashAssign (FILE* DataFilePointer,
                  FILE* IndexFilePointer,
                  long lI,
                  long lV,
                  long lL,
                  long lW1);
```

DataFilePointer :

fopen関数によりオープンされたデータファイルポインタ。
本関数を使用する前に、データファイルは以下のようにオープンされていなければなりません。

```
#include <stdio.h>
FILE *DataFilePointer;
DataFilePointer = fopen (char* file,char* mode);
  file     : データファイル名
  mode   : ファイルアクセスモード ("r")
```

IndexFilePointer:

fopen関数によりオープンされたインデックスファイルポインタ。
本関数を使用する前に、データファイルは以下のようにオープンされていなければなりません。

```
#include <stdio.h>
FILE *IndexFilePointer;
IndexFilePointer = fopen (char* file,char* mode);
  file     : インデックスファイル名
  mode   : ファイルアクセスモード ("w+")
```

lI, lV, lL, lW1:

これらの値は、インデックスファイルのヘッダ部に格納され、以下のような意味を持ちます。

lI	:データファイルのキーフィールド長
lV	:データファイルの総レコード数 ただし、将来データ数が増える見込みがある場合は、現在のデータファイルの実際の総レコード数よりも大きい値を指定します。
lL	:データファイルのレコード長(キー＋データ)
lW1	:インデックスファイルの総インデックス数(ヘッダ情報除く)

（通常 、lW1 >= lV * 1.2 上記変数は、**long** 型です。）

実行時に必要なメモリ（単位＝バイト）:

DT-900	PC（16ビット）	PC（32ビット）
490＋2×キー長	460＋2×キー長	442＋2×キー長

リターン値:

意味	リターンコード
正常終了	0 (HASH_OPERATIONOK)
データファイルリードエラー	102 (DATA_NOFILEREAD)
データファイル未オープン	105 (DATA_NOFILEOPEN)
データファイルシークエラー	111 (DATA_NOFILESEEK)
インデックスファイルリードエラー	202 (IDX_NOFILEREAD)
インデックスファイルライトエラー	203 (IDX_NOFILEWRITE)
インデックスファイル未オープン	205 (IDX_NOFILEOPEN)
インデックスファイルシークエラー	211 (IDX_NOFILESEEK)
実行メモリ不足	320 (HASH_NOMEMORY)
パラメータエラー	400 (PARAM_INVALID)

B. iHashRead

1. 概要

本関数は、入力キーデータに対応するデータを検索します。

2. コーリングシーケンス

```
int iHashRead (char *pszKey,
               FILE *DataFilePointer,
               FILE *IndexFilePointer,
               char  *pszBuff);
```

pszKey:

検索するキーデータのポインタ。
キーデータの末尾には、NULL文字を入れて下さい。
キーデータ長が、データファイル内の実際のキーデータ長と一致しない場合は、エラーとなります。

DataFilePointer :

fopen関数によりオープンされたデータファイルポインタ。
本関数を使用する前に、データファイルは以下のようにオープンされていなければなりません。

```
#include <stdio.h>
FILE *DataFilePointer;
DataFilePointer = fopen (char* file,char* mode);
  file    : データファイル名
  mode    : ファイルアクセスモード ("r")
```

IndexFilePointer:

fopen関数によりオープンされたインデックスファイルポインタ。
本関数を使用する前に、データファイルは以下のようにオープンされていなければなりません。

```
#include <stdio.h>
FILE *IndexFilePointer;
IndexFilePointer = fopen (char* file,char* mode);
  file    : インデックスファイル名
  mode    : ファイルアクセスモード ("r")
```

pszBuff:

入力されたキーと一致するデータレコードが格納されるバッファポインタ。
(ただし、データレコードには、キーは含まれません)
データレコードの最終には、NULL文字が付加されます。
もし、入力されたキーと一致するデータレコードが見つからない場合はNULLポインタを返します。

実行時に必要なメモリ（単位＝バイト）:

DT-900	PC（16ビット）	PC（32ビット）
512＋2×キー長	482＋2×キー長	462＋2×キー長

リターン値:

内容	リターンコード
正常終了	0 (HASH_OPERATIONOK)
データファイルリードエラー	102 (DATA_NOFILEREAD)
データファイル未オープン	105 (DATA_NOFILEOPEN)
データファイルシークエラー	111 (DATA_NOFILESEEK)
インデックスファイルリードエラー	202 (IDX_NOFILEREAD)
インデックスファイル未オープン	205 (IDX_NOFILEOPEN)
インデックスファイルシークエラー	211 (IDX_NOFILESEEK)
実行メモリ不足	320 (HASH_NOMEMORY)
該当インデックスなし	330 (HASH_KEYNOTFOUND)
不正キー入力（キー長不一致）	331 (HASH_INVALIDKEY)
キーポインタ不正 (NULL)	332 (HASH_KEYNULL)

C. *iHashWrite*

1. 概要

本ファンクションは、指定キーのデータレコードを書き換えます。
データファイルのデータレコード以外は、不変です。

2. コーリングシーケンス

```
int  iHashWrite (char  *pszKey,
                 FILE *DataFilePointer,
                 FILE *IndexFilePointer,
                 char  *pszBuff)
```

pszKey:

変更するデータのキーデータポインタ。
キーデータの末尾には、NULL文字を入れて下さい。
キーデータ長が、データファイル内の実際のキーデータ長と一致しない場合は、エラーとなります。

DataFilePointer :

fopen関数によりオープンされたデータファイルポインタ。
本関数を使用する前に、データファイルは以下のようにオープンされていなければなりません。

```
#include <stdio.h>
FILE *DataFilePointer;
DataFilePointer = fopen (char* file,char* mode);
  file     : データファイル名
  mode   : ファイルアクセスモード ("r+")
```

IndexFilePointer:

fopen関数によりオープンされたインデックスファイルポインタ。
本関数を使用する前に、データファイルは以下のようにオープンされていなければなりません。

```
#include <stdio.h>
FILE *IndexFilePointer;
IndexFilePointer = fopen (char* file,char* mode);
  file     : インデックスファイル名
  mode   : ファイルアクセスモード ("r")
```

pszBuff:

書き換えるデータのポインタ。(キーは、含みません)
先頭から、NULL文字までをデータとみなします。
データ長は、インデックスファイル作成時に指定した値(IL－II)と一致していなければなりません。
一致しない場合は、エラーとなります。.

実行時に必要なメモリ（単位＝バイト）:

DT-900	PC（16ビット）	PC（32ビット）
500 ＋2×キー長 ＋データ長	470 ＋2×キー長 ＋データ長	452 ＋2×キー長 ＋データ長

リターン値:

内容	リターンコード
正常終了	0 (HASH_OPERATIONOK)
データファイルリードエラー	102 (DATA_NOFILEREAD)
データファイルライトエラー	103 (DATA_NOFILEWRITE)
データファイル未オープン	105 (DATA_NOFILEOPEN)
データファイルシークエラー	111 (DATA_NOFILESEEK)
インデックスファイルリードエラー	202 (IDX_NOFILEREAD)
インデックスファイル未オープン	205 (IDX_NOFILEOPEN)
インデックスファイルシークエラー	211 (IDX_NOFILESEEK)
実行メモリ不足	320 (HASH_NOMEMORY)
該当インデックスなし	330 (HASH_KEYNOTFOUND)
不正キー入力（キー長不一致）	331 (HASH_INVALIDKEY)
キーポインタ不正 (NULL)	332 (HASH_KEYNULL)
不正データ入力（データ長不一致）	333 (HASH_INVALIDDATA)
データポインタ不正 (NULL)	334 (HASH_DATANULL)

D. iHashAdd

1. 概要

本ファンクションは、データファイルに新しいレコード(キー＋データ)を追加します。 一度に追加できるレコードは、1レコードで、既存レコードの末尾に追加されます。
(既存レコードの間に追加することはできません)
データ・ファイルがオーバーフローしていないこと、データレコードの数が、インデックスヘッダーに指定されている数より少ないことが条件です。

2. コーリングシーケンス

```
int  iHashAdd (char *pszKey,
               FILE *DataFilePointer,
               FILE *IndexFilePointer,
               char  *pszAppendData)
```

pszKey:

追加するキーデータのポインタ。
キーデータの末尾には、NULL文字を入れて下さい。
キーデータ長が、データファイル内の実際のキーデータ長と一致しない場合は、エラーとなります。

DataFilePointer :

fopen関数によりオープンされたデータファイルポインタ。
本関数を使用する前に、データファイルは以下のようにオープンされていなければなりません。

```
#include <stdio.h>
FILE *DataFilePointer;
DataFilePointer = fopen (char* file,char* mode);
  file    : データファイル名
  mode    : ファイルアクセスモード ("r+")
```

IndexFilePointer:

fopen関数によりオープンされたインデックスファイルポインタ。
本関数を使用する前に、データファイルは以下のようにオープンされていなければなりません。

```
#include <stdio.h>
FILE *IndexFilePointer;
IndexFilePointer = fopen (char* file,char* mode);
  file    : インデックスファイル名
  mode    : ファイルアクセスモード ("r+")
```

pszAppendData:

追加するレコードのポインタ。
レコードの末尾には、NULLを入れて下さい。
レコード長がデータファイルの実際のレコード長と一致しない場合は、エラーとなります。

実行時に必要なメモリ（単位＝バイト）:

DT-900	PC（16ビット）	PC（32ビット）
514 ＋キー長 ＋2×データ長	484 ＋キー長 ＋2×データ長	466 ＋キー長 ＋2×データ長

リターン値:

内容	リターンコード
正常終了	0 (HASH_OPERATIONOK)
データファイルライトエラー	103 (DATA_NOFILEWRITE)
データファイルフル	104 (DATA_FILEOVERFLOW)
データファイル未オープン	105 (DATA_NOFILEOPEN)
データファイルシークエラー	111 (DATA_NOFILESEEK)
インデックスファイルリードエラー	202 (IDX_NOFILEREAD)
インデックスファイルライトエラー	203 (IDX_NOFILEWRITE)
インデックスファイル未オープン	205 (IDX_NOFILEOPEN)
インデックスファイルシークエラー	211 (IDX_NOFILESEEK)
実行メモリ不足	320 (HASH_NOMEMORY)
該当インデックスなし	330 (HASH_KEYNOTFOUND)
不正キー入力（キー長不一致）	331 (HASH_INVALIDKEY)
キーポインタ不正 (NULL)	332 (HASH_KEYNULL)
不正データ入力（データ長不一致）	333 (HASH_INVALIDDATA)
データポインタ不正 (NULL)	334 (HASH_DATANULL)

E. 開発環境

ライブラリ開発環境は以下の通りです。

ﾗｲﾌﾞﾗﾘ	使用ｺﾝﾊﾟｲﾗ	ﾊﾞｰｼﾞｮﾝ
PC 32 Bit	Microsoft 32-Bit C/C++ Optimizing Compiler Microsoft Incremental Linker (Supplied with VC++ version 4.2)	10.20.6166 for 80x86 4.20.6164
PC 16 Bit	Microsoft (R) C/C++ Optimizing Compiler Microsoft segmented executable Linker (Supplied with VC++ version 1.52)	8.00c 5.60.339
DT 900	KJ_CNVRT SH SERIES C Compiler (SHC) SH SERIES LINKER (LNK) SUBFILE'S B SECTION MAKE TOOL (SUBMK) OBJECT CONVERTER (CNVS) AP CONVERT PROGRAM (APCNVY)	--- 2.0D 5.1 1.01 1.3C 1.02

16ビットライブラリのメモリモデル

本ライブラリは、Small／ Medium／Largeの各メモリモデルをサポートしています。

￥**PC16BIT￥LIB**ディレクトリには、次のようなファイルがあります。

SHASH.LIB　　スモールモデルライブラリ
MHASH.LIB　　ミディアムモデルライブラリ
LHASH.LIB　　ラージモデルライブラリ

￥**PC16BIT￥SAMPLE**ディレクトリには、次のようなバッチファイルがあります。

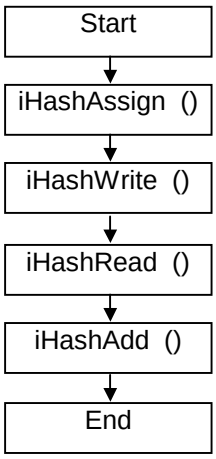
SSAMPLE.BAT
MSAMPLE.BAT
LSAMPLE.BAT

インクルードファイル

本ライブラリを使用する際は、HASHLIB.Hをインクルードしてください。

F. サンプルアプリケーション

本章では、SAMPLEディレクトリにあるサンプルアプリケーションを説明します。
サンプルアプリケーションは、次のような構成になっています。




```
/*****************************************************************************
  Copyright information    : Casio Computer Co. Ltd., Japan
  File Name                : Sample.c
  Project Name             : HASH LIBRARY
  Author Name              : Jei, Srinivasa Murthy, Vijay Mantri
  Date of Creation         : 18/8/1998
  Version Number           : 1.00
  Brief Description        : Sample program to test Hash Library Functions
                             On DT-900
 *****************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "HashLib.h"
#include "itron.h"
#include "cmndef.h"
#include "bios1mac.h"

/* ---------------------------------------------------------------------------
  Calls             : iHashAssign (),
                      iHashRead (),
                      iHashWrite (),
                      iHashAdd ()
 ---------------------------------------------------------------------------*/

int ap_start(void)
{

FILE *fpIndex,              /* File pointer for Index file.   */
     *fpData;               /* File pointer for Data file.    */

long ll = 8,               /* To hold Length of Key.         */
     lV = 40,              /* To hold Volume of Data file.   */ /* 28-NOV-1997*/
     lL = 10,              /* To hold Length of Data record. */
     lW1 = (lV*120)/100,   /* To hold volume of Index file.  */
     lCount = 0;           /* To hold a execution counter.   */
```

変数 W, V, L, Iの初期値は以下のとおりです。
I　= キー長 = 8
L　= レコード長 = 10
V　= 総レコード数 = 40
W = 総インデックス数 = Vの1.2倍 = 48

```
int iI;                  /* To hold result of HashFunctins.*/
char *pszBuff,           /* Key input to Hash Functions.   */
    *pszReadBuf;         /* To hold read/write data.       */

/* ap_init(); */         /* Removed on info from CASIO 28-NOV-1997 */

/*---------------------- HASH ASIGN START ----------------------------------*/

fpData  = fopen("a:¥¥sample.dat","r");
fpIndex = fopen("b:¥¥sample.idx","w+");

pszBuff = malloc((size_t)(40));

iDisplayMessage ("HASH asign started : ");
iI = iHashAssign (fpData, fpIndex, II, IV, IL, IW1);

if (iI != 0)
{
iDisplayMessage ("** ERROR: HashAssign **");
sprintf (pszBuff, "%d",iI);
iDisplayMessage (pszBuff);
}
```

SAMPLEディレクトリにある SAMPLE.DATのデータは、以下のとおりです。

```
00000000ab
00000001ab
00000002ab
00000003ab
00000004ab
00000005ab
00000006ab
00000007ab
00000008ab
00000009ab
```

iHashAssign() 関数は上記すべてのレコード(10ﾚｺｰﾄﾞ)をアサインした後にリターンします。
iHashAssign() 関数終了時にインデックスファイル(SAMPLE.IDX)が生成されます。
インデックスファイルの内容は以下のとおりです。

```
00000030 indicates W = 48     }
00000028 indicates V = 40     }  ﾍｯﾀﾞ
0000000A indicates L = 10     }
00000008 indicates I =  8     }
0000000A indicates N = 10     }
00000001 ここからインデックスのエントリが開始されます。
00000000
00000005
00000000
```

上記例では、**V=40**となっています。これは、インデックスファイルがトータル**40ﾚｺｰﾄﾞ**確保されていることを意味します。 **N=10**は、インデックスファイル容量40ﾚｺｰﾄﾞ中10ﾚｺｰﾄﾞが使用されていることを意味します。

```
iDisplayMessage ("HASH asign over¥n");

fclose (fpData);
fclose (fpIndex);

/*---------------------- HASH ASIGN END ----------------------------------*/
```

```
/*---------------------- HASH WRITE START ----------------------------------*/
fpData  = fopen("a:¥¥sample.dat","r+");
fpIndex = fopen("b:¥¥sample.idx","r");

iDisplayMessage ("HASH write started : ");
pszBuff = malloc((size_t)(40));
if (pszBuff == NULL) return -1;

for (lCount = 0; lCount < 10; lCount++ ) {
    memset (pszBuff, 0, (size_t)40);
    sprintf (pszBuff, "%08lX", lCount);
    iI = iHashWrite (pszBuff, fpData, fpIndex, "xy");
    if (iI != 0)
                        break;
}

if (iI != 0) iDisplayMessage ("** ERROR: HashWrite **");
free (pszBuff);
iDisplayMessage ("HASH write over¥n");
fclose (fpData);
fclose (fpIndex);
/*---------------------- HASH WRITE END ------------------------------------*/
```

iHashWrite() 関数は、SAMPLE.DAT中のデータ “**ab**” を “**xy**” に書き換えます。
SAMPLE.DATは、以下のように書き換えられます。

```
00000000xy
00000001xy
00000002xy
00000003xy
00000004xy
00000005xy
00000006xy
00000007xy
00000008xy
00000009xy
```

このとき、インデックスファイル(SAMPLE.IDX) の内容は、変化しません。

```
00000030 indicates W = 48     }
00000028 indicates V = 40     }  ﾍｯﾀﾞ
0000000A indicates L = 10     }
00000008 indicates I =  8     }
0000000A indicates N = 10     }
00000001 ここからインデックスのエントリが開始されます。
00000000
00000005
00000000
```

```
/*---------------------- HASH READ START -----------------------------------*/
fpData  = fopen("a:¥¥sample.dat","r");
fpIndex = fopen("b:¥¥sample.idx","r");

iDisplayMessage ("HASH read started  : ");
pszBuff = malloc((size_t)(40));
if (pszBuff == NULL) return -1;

pszReadBuf = malloc((size_t)(40));
if (pszReadBuf == NULL) return -1;
for (lCount=0; lCount < 10; lCount++) {
    memset (pszBuff, 0, (size_t)40);
    sprintf (pszBuff, "%08lX", lCount);
    iI = iHashRead( pszBuff, fpData, fpIndex, pszReadBuf);
    if (iI != 0)
        break;
}
if (iI != 0)
iDisplayMessage ("** ERROR: HashRead **");

free (pszBuff);
free (pszReadBuf);
iDisplayMessage ("HASH read over¥n");
fclose (fpData);
fclose (fpIndex);
/*---------------------- HASH READ END -------------------------------------*/
```

iHashRead() 関数は、インデックスファイルおよびデータファイルからデータを読み出します。従って、インデックスファイルおよびデータファイルの内容は変化しません。

```
/*---------------------- HASH ADD START -----------------------------------*/

fpData  = fopen("a:¥¥sample.dat","r+");
fpIndex = fopen("b:¥¥sample.idx","r+");
pszBuff = malloc((size_t)(40));
if (pszBuff == NULL) return -1;

iDisplayMessage ("HASH add started   : ");
for (lCount=10; lCount < 20; lCount++) {

    memset (pszBuff, 0, (size_t)40);
    sprintf (pszBuff, "%08lX", lCount);
    iI = iHashAdd( pszBuff, fpData, fpIndex, "nn");
    if (iI != 0)
        break;
}
```

レコード0～9は、既にデータファイルに存在します。ここでは、レコード10～20の10レコードをデータファイルに追加します。

```
if (iI != 0)
{
iDisplayMessage ("** ERROR: HashAdd **");
sprintf (pszBuff, "%d",iI);
iDisplayMessage (pszBuff);
}
iDisplayMessage ("HASH add over¥n");

free (pszBuff);
fclose (fpData);
fclose (fpIndex);
/*---------------------- HASH ADD END -------------------------------------*/
```

iHashAdd() 関数は、10個のレコードをデータファイルに追加します。
本関数が正常に終了すると、データファイルは以下のようになります。

```
        00000000xy  }
        00000001xy  }
        00000002xy  }
        00000003xy  }
        00000004xy  } 以前から存在するデータレコード
        00000005xy  }
        00000006xy  }
        00000007xy  }
        00000008xy  }
        00000009xy  }
        0000000Ann    }
        0000000Bnn    }
        0000000Cnn    }
        0000000Dnn    }
        0000000Enn    }
        0000000Fnn    } iHashAdd () 関数により追加されたレコード
        00000010nn    }
        00000011nn    }
        00000012nn    }
        00000013nn    }
```

このとき、インデックスファイルは、以下のようになります。

```
        00000030 indicates W = 48  }
        00000028 indicates V = 40   }
        0000000A indicates L = 10   }  ﾍｯﾀﾞ
        00000008 indicates I =  8    }
        00000014 indicates N = 20   }
        00000001 ここからインデックスのエントリが開始されます。
        00000000
        00000005
        00000000
```

N (データファイル中の使用レコード数) が10から20に変化します。

```
return 0;

}           /* End of Sample.C */

/*****************************************************************************
  Name              : int iDisplayMessage()
  Synopsis          : This function displays the given message on
                      the LCD screen of DT-900.
  Input parameters  : pszMessage: message to be displayed
  Output parameters : 0
  Return Type       : Integer.
  Calling Syntax    : iDisplayMessage ( char *pszMessage )
  Called by         : Sample program to test Hash Library Functions.

 *****************************************************************************/
int iDisplayMessage (char *pszMessage) {
        lcd_cls();
        lcd_csr_put(0,0);
        lcd_string(LCD_ANK_STANDARD,
                                LCD_ATTR_NORMAL,
        (UB*) pszMessage,
                                LCD_LF_ON);
return 0;
}
```

注意事項

インデックスファイルの総データレコード数(V)は、重要です。
サンプルプログラムでは、V=40でデータファイルを定義しています。
インデックスファイルの総インデックス数(W)は、総データレコードの1.2倍の48となっています。はじめに、10レコードを消費し、さらに10レコードを追加するとデータファイルの残容量は20になります。.もし、残容量(20)以上のデータを追加する必要が生じた場合は、Vの値を増やし、再度 iHashAssign() 関数を実行する必要があります。
以下に、プログラム例を示します。

1. Vをより大きな値 (80)に書き換えます.
2. iHashAssign() 関数を実行します。

```
long lI = 8,              /* To hold Length of Key.        */
    lV = 80,              /* To hold Volume of Data file.  */
    lL = 10,              /* To hold Length of Data record.*/
    lW1 = (lV*120)/100,   /* To hold volume of Index file. */
    lCount = 0;           /* To hold a execution counter.  */
```

W, V, L, Iを以下の値とします（Vのみ**40→80**に変更）
I　= キー長 = 8
L　= データレコード長 = 10
V　= 総データレコード数 = 80
W　= 96

```
int iI;                  /* To hold result of HashFunctins.*/
char *pszBuff,           /* Key input to Hash Functions.   */
    *pszReadBuf;         /* To hold read/write data.       */

/*---------------------- HASH ASIGN START ----------------------------------*/

fpData  = fopen("a:¥¥sample.dat","r");
fpIndex = fopen("b:¥¥sample.idx","w+");

pszBuff = malloc((size_t)(40));

iDisplayMessage ("HASH asign started : ");
iI = iHashAssign (fpData, fpIndex, lI, lV, lL, lW1);

if (iI != 0)
{
iDisplayMessage ("** ERROR: HashAssign **");
sprintf (pszBuff, "%d",iI);
iDisplayMessage (pszBuff);
}
```

iHashAssign()関数実行後、インデックスファイルのヘッダ部は以下のようになります。

```
00000060 indicates W = 96  }
00000050 indicates V = 80   }
0000000A indicates L = 10   }  ﾍｯﾀﾞ
00000008 indicates I = 8    }
00000028 indicates N = 40   }
00000001
0000000B
00000015
```

オペレーション	N	V	コメント
Startup	-	40	
iHashAssign() for 40.	10	40	10ﾚｺｰﾄﾞ追加
iHashWrite()	10	40	ﾚｺｰﾄﾞ書き換え
iHashRead()	10	40	ﾚｺｰﾄﾞ読み出し
iHashAdd() for 10.	20	40	20ﾚｺｰﾄﾞ追加
iHashAdd() for 10.	30	40	10ﾚｺｰﾄﾞ追加
iHashAdd() for 10.	40	40	ﾚｺｰﾄﾞﾌﾙ(追加不能)
iHashAssign() for 80.	40	80	40ﾚｺｰﾄﾞ追加可能
iHashAdd() for 5.	45	80	35ﾚｺｰﾄﾞ追加可能

G.　リリース媒体内容

リリースディスクの内容は、以下のとおりです。

DT900	**LIB**	HASH900.LIB
		HASHLIB.H
	SAMPLE	SAMPLE.C
		SAMPLE.DAT
		SAMPLE.MAK
PC16BIT	**LIB**	SHASH.LIB
		MHASH.LIB
		LHASH.LIB
		HASHLIB.H
	SAMPLE	SAMPLE.C
		SAMPLE.DAT
		LSAMPLE.BAT
		MSAMPLE.BAT
		SSAMPLE.BAT
PC32BIT	**LIB**	HASH32.LIB
		HASHLIB.H
	SAMPLE	SAMPLE.C
		SAMPLE.DAT
		SAMPLE.BAT