

Πληροφορική & Τηλεπικοινωνίες

Υλοποίηση Συστημάτων Βάσεων Δεδομένων Χειμερινό Εξάμηνο 2022 – 2023

Διδάσκων Θ. Μαΐλης

Άσκηση 1 - Προθεσμία: 20/12/2022

Άσκηση 2 - Προθεσμία: 09/01/2023

Σκοπός της Εργασίας	2
Συναρτήσεις BF (Block File)	3
Μεταβλητές & Enumerations	3
Συναρτήσεις	3
Άσκηση 1	6
Συναρτήσεις HP (σωρού (heap file))	6
Συναρτήσεις HT (Hash Table)	7
Άσκηση 2	9
Συναρτήσεις SHT (Secondary Hash Table)	9
Στατιστικά Κατακερματισμού	11
Συνάρτηση main Ελέγχου Λειτουργικότητας	11
Σημειώσεις:	11
Παράδοση εργασίας	11
Παραδοτέα	12

Σκοπός της Εργασίας

Σκοπός της εργασίας αυτής είναι η κατανόηση της εσωτερικής λειτουργίας των Συστημάτων Βάσεων Δεδομένων όσον αφορά τη διαχείριση σε **επίπεδο μπλοκ (block)** αλλά και ως προς τη διαχείριση σε **επίπεδο εγγραφών**. Επίσης, μέσω της εργασίας θα γίνει αντιληπτό το κατά πόσο μπορεί να βελτιώσει την απόδοση ενός Συστήματος Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) η ύπαρξη ευρετηρίων πάνω στις εγγραφές. Πιο συγκεκριμένα, στα πλαίσια της εργασίας θα υλοποιήσετε ένα σύνολο συναρτήσεων που διαχειρίζονται αρχεία που δημιουργήθηκαν βάσει (1) **οργάνωσης αρχείου σωρού (heap files)** και (2) **στατικού κατακερματισμού (Hash Table)**.

Οι συναρτήσεις που καλείστε να υλοποιήσετε αφορούν τη διαχείριση εγγραφών και τη διαχείριση ευρετηρίων. Η υλοποίησή τους θα γίνει πάνω από το επίπεδο διαχείρισης μπλοκ υποχρεωτικά, το οποίο δίνεται έτοιμο ως βιβλιοθήκη. Τα πρωτότυπα (definitions) των συναρτήσεων που καλείστε να υλοποιήσετε όσο και των συναρτήσεων της βιβλιοθήκης επιπέδου μπλοκ δίνονται στη συνέχεια, μαζί με επεξήγηση για τη λειτουργικότητα που θα επιτελεί η κάθε μία.

- Η διαχείριση των **αρχείων σωρού (heap files)** γίνεται μέσω των συναρτήσεων με το πρόθεμα HP_.
- Η διαχείριση των **αρχείων στατικού κατακερματισμού** γίνεται μέσω των συναρτήσεων με το πρόθεμα HT_.

Τα αρχεία σωρού και στατικού κατακερματισμού έχουν μέσα εγγραφές τις οποίες διαχειρίζεστε μέσω των κατάλληλων συναρτήσεων. Οι εγγραφές έχουν τη μορφή που δίνεται στη συνέχεια. Το id είναι το κλειδί.

```
typedef struct{
    int id,
    char name[15],
    char surname[25],
    char address[50];
} Record;
```

Το **πρώτο μπλοκ (block)** κάθε αρχείου, περιλαμβάνει “ειδική” πληροφορία σχετικά με το ίδιο το αρχείο. Η πληροφορία αυτή χρησιμεύει στο να *αναγνωρίζει κανείς αν πρόκειται για αρχείο σωρού ή αρχείο στατικού κατακερματισμού*, σε πληροφορία που *αφορά το πεδίο κατακερματισμού για τα αντίστοιχα αρχεία* κ.λπ. Στη συνέχεια δίνεται ένα παράδειγμα των εγγραφών που θα προστίθενται στα αρχεία που

θα δημιουργείται με την υλοποίησή σας. Εγγραφές με τις οποίες μπορείτε να ελέγξετε το πρόγραμμά σας θα δοθούν υπό μορφή αρχείων κειμένου μαζί με κατάλληλες συναρτήσεις `main` στο `eclass` του μαθήματος.

```
{15, "Giorgos", "Dimopoulos", "Ioannina"}  
{4, "Antonia", "Papadopoulou", "Athina"}  
{300, "Yannis", "Yannakis", "Thessaloniki"}
```

Συναρτήσεις BF (Block File)

Το επίπεδο **block** (BF) είναι ένας **διαχειριστής μνήμης** (**memory manager**) που λειτουργεί σαν κρυφή μνήμη (cache) ανάμεσα στο επίπεδο του δίσκου και της μνήμης. Το επίπεδο block κρατάει block δίσκου στην μνήμη. Κάθε φορά που ζητάμε ένα block δίσκου, το επίπεδο BF πρώτα εξετάζει την περίπτωση να το έχει φέρει ήδη στην μνήμη. Αν το block υπάρχει στην μνήμη τότε δεν το διαβάζει από τον δίσκο, σε αντίθετη περίπτωση το διαβάζει από τον δίσκο και το τοποθετεί στην μνήμη. Επειδή το επίπεδο BF δεν έχει άπειρη μνήμη κάποια στιγμή θα χρειαστεί να “πετάξουμε” κάποιο block από την μνήμη και να φέρουμε κάποιο άλλο στην θέση του. Οι πολιτικές που μπορούμε να πετάξουμε ένα block από την μνήμη στο επίπεδο block που σας δίνεται είναι οι **LRU (Least Recently Used)** και **MRU (Most Recently Used)**. Στην LRU “θυσιάζουμε” το λιγότερο πρόσφατα χρησιμοποιημένο block ενώ στην MRU το block που χρησιμοποιήσαμε πιο πρόσφατα.

Στη συνέχεια, περιγράφονται οι συναρτήσεις που αφορούν το επίπεδο από block, πάνω στο οποίο θα βασιστείτε για την υλοποίηση των συναρτήσεων που ζητούνται. Η υλοποίηση των συναρτήσεων αυτών θα δοθεί έτοιμη με τη μορφή βιβλιοθήκης.

Μεταβλητές & Enumerations

Στο αρχείο κεφαλίδας `bf.h` που σας δίνεται ορίζονται οι πιο κάτω μεταβλητές:

```
BF_BLOCK_SIZE 512 /*Το μέγεθος ενός block σε bytes*/  
BF_BUFFER_SIZE 100 /*Ο μέγιστος αριθμός block που κρατάμε στην μνήμη*/  
BF_MAX_OPEN_FILES 100 /*Ο μέγιστος αριθμός ανοικτών αρχείων*/
```

και enumerations:

```
enum BF_ErrorCode { ... }
```

Το `BF_ErrorCode` είναι ένα enumeration που ορίζει κάποιους κωδικούς λάθους που μπορεί να προκύψουν κατά την διάρκεια της εκτέλεσης των συναρτήσεων του επιπέδου BF.

```
enum ReplacementAlgorithm { LRU, MRU }
```

Το ReplacementAlgorithm είναι ένα enumeration που ορίζει τους κωδικούς για τους αλγορίθμους αντικατάστασης (LRU ή MRU).

Συναρτήσεις

Πιο κάτω υπάρχουν τα πρωτότυπα των συναρτήσεων που σχετίζονται με την δομή BF_Block.

```
typedef struct BF_Block BF_Block;
```

Το struct BF_Block είναι η βασική δομή που δίνει οντότητα στην έννοια του Block. Το BF_Block έχει τις πιο κάτω λειτουργίες.

```
void BF_Block_Init(BF_Block **block  
/*δομή που προσδιορίζει το Block*/)
```

Η συνάρτηση BF_Block_Init αρχικοποιεί και δεσμεύει την κατάλληλη μνήμη για την δομή BF_BLOCK.

```
void BF_Block_Destroy(BF_Block **block  
/*δομή που προσδιορίζει το Block*/)
```

Η συνάρτηση BF_Block_Destroy αποδεσμεύει την μνήμη που καταλαμβάνει η δομή BF_BLOCK.

```
void BF_Block_SetDirty(BF_Block *block  
/*δομή που προσδιορίζει το Block*/)
```

Η συνάρτηση BF_Block_SetDirty αλλάζει την κατάσταση του block σε dirty. Αυτό πρακτικά σημαίνει ότι τα δεδομένα του block έχουν αλλαχτεί και το επίπεδο BF, όταν χρειαστεί θα γράψει το block ξανά στον δίσκο. Σε περίπτωση που απλώς διαβάζουμε τα δεδομένα χωρίς να τα αλλάζουμε τότε δεν χρειάζεται να καλέσουμε την συνάρτηση.

```
char* BF_Block_GetData(const BF_Block *block  
/*δομή που προσδιορίζει το Block*/)
```

Η συνάρτηση BF_Block_GetData επιστρέφει ένα δείκτη στα δεδομένα του Block. Αμα αλλάξουμε τα δεδομένα θα πρέπει να κάνουμε το block dirty με την κλήση της συνάρτησης BF_Block_SetDirty. Σε καμία περίπτωση δεν πρέπει να αποδεσμεύσετε την θέση μνήμης που δείχνει ο δείκτης.

Πιο κάτω υπάρχουν τα πρωτότυπα των συναρτήσεων που σχετίζονται με το επίπεδο Block.

```
BF_ErrorCode BF_Init(const ReplacementAlgorithm repl_alg  
/*πολιτική αντικατάστασης*/)
```

Με τη συνάρτηση BF_Init πραγματοποιείται η αρχικοποίηση του επιπέδου BF. Μπορούμε να επιλέξουμε ανάμεσα σε δύο πολιτικές αντικατάστασης Block εκείνης της LRU και εκείνης της MRU.

```
BF_ErrorCode BF_CreateFile(const char* filename
```

```
/*όνομα αρχείου*/)
```

Η συνάρτηση BF_CreateFile δημιουργεί ένα αρχείο με όνομα filename το οποίο αποτελείται από blocks. Αν το αρχείο υπάρχει ήδη τότε επιστρέφεται κωδικός λάθους. Σε περίπτωση επιτυχούς εκτέλεσης της συνάρτησης επιστρέφεται BF_OK, ενώ σε περίπτωση αποτυχίας επιστρέφεται κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

```
BF_ErrorCode BF_OpenFile(  
const char* filename, /*όνομα αρχείου*/  
*file_desc /*αναγνωριστικό αρχείου block*/);
```

Η συνάρτηση BF_OpenFile ανοίγει ένα υπάρχον αρχείο από blocks με όνομα filename και επιστρέφει το αναγνωριστικό του αρχείου στην μεταβλητή file_desc. Σε περίπτωση επιτυχίας επιστρέφεται BF_OK ενώ σε περίπτωση αποτυχίας, επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

```
BF_ErrorCode BF_CloseFile( int file_desc  
/*αναγνωριστικό αρχείου block*/)
```

Η συνάρτηση BF_CloseFile κλείνει το ανοιχτό αρχείο με αναγνωριστικό αριθμό file_desc. Σε περίπτωση επιτυχίας επιστρέφεται BF_OK ενώ σε περίπτωση αποτυχίας, επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

```
BF_ErrorCode BF_GetBlockCounter(  
const int file_desc, /*αναγνωριστικό αρχείου block*/  
int *blocks_num /*τιμή που επιστρέφεται*/)
```

Η συνάρτηση Get_BlockCounter δέχεται ως όρισμα τον αναγνωριστικό αριθμό file_desc ενός ανοιχτού αρχείου από block και βρίσκει τον αριθμό των διαθέσιμων blocks του, τον οποίο και επιστρέφει στην μεταβλητή blocks_num. Σε περίπτωση επιτυχίας επιστρέφεται BF_OK ενώ σε περίπτωση αποτυχίας, επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

```
BF_ErrorCode BF_AllocateBlock(  
const int file_desc, /*αναγνωριστικό αρχείου block*/  
BF_Block *block /*το block που επιστρέφεται*/)
```

Με τη συνάρτηση BF_AllocateBlock δεσμεύεται ένα καινούριο block για το αρχείο με αναγνωριστικό αριθμό blockFile. Το νέο block δεσμεύεται πάντα στο τέλος του αρχείου, οπότε ο αριθμός του block είναι BF_getBlockCounter(...) - 1. Το block που δεσμεύεται καρφιτσώνεται στην μνήμη (pin) και επιστρέφεται στην μεταβλητή block. Όταν δεν χρειαζόμαστε άλλο αυτό το block τότε πρέπει να ενημερώσουμε το επίπεδο block καλώντας την συνάρτηση BF_UnpinBlock. Σε περίπτωση επιτυχίας

της BF_AllocateBlock επιστρέφεται BF_OK, ενώ σε περίπτωση αποτυχίας επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

```
BF_ErrorCode BF_GetBlock(  
    const int file_desc, /*αναγνωριστικό αρχείου block*/  
    const int block_num, /*αναγνωριστικός αριθμός block*/  
    BF_Block *block /*το block που επιστρέφεται*/)

```

Η συνάρτηση BF_GetBlock βρίσκει το block με αριθμό block_num του ανοιχτού αρχείου file_desc και το επιστρέφει στην μεταβλητή block. Το block που δεσμεύεται καρφιτσώνεται στην μνήμη (pin). Όταν δεν χρειαζόμαστε άλλο αυτό το block τότε πρέπει να ενημερώσουμε τον επίπεδο block καλώντας την συνάρτηση BF_UnpinBlock. Σε περίπτωση επιτυχίας της BF_GetBlock επιστρέφεται BF_OK, ενώ σε περίπτωση αποτυχίας επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

```
BF_ErrorCode BF_UnpinBlock(BF_Block *block  
    /*δομή block που γίνεται unpin*/)

```

Η συνάρτηση BF_UnpinBlock ξεκαρφιτσώνει το block από το επίπεδο BF το οποίο κάποια στιγμή θα το γράψει στο δίσκο. Σε περίπτωση επιτυχίας επιστρέφεται BF_OK, ενώ σε περίπτωση αποτυχίας επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

```
void BF_PrintError( BF_ErrorCode err /*κωδικός λάθους*/)

```

Η συνάρτηση BF_PrintError βοηθά στην εκτύπωση των σφαλμάτων που δύναται να υπάρξουν με την κλήση συναρτήσεων του επιπέδου αρχείου block. Εκτυπώνεται στο stderr μια περιγραφή του σφάλματος. void BF_Close(). Η συνάρτηση BF_Close κλείνει το επίπεδο Block γράφοντας στον δίσκο όποια block είχε στην μνήμη.

Ασκηση 1

Συναρτήσεις HP (σωρού (heap file))

Στη συνέχεια περιγράφονται οι συναρτήσεις που καλείστε να υλοποιήσετε στα πλαίσια της εργασίας αυτής και που αφορούν το αρχείο σωρού (heap file).

```
int HP_CreateFile(  
    char *fileName, /*όνομα αρχείου*/  
    Record_Attribute keyAttribute; /*Προσδιορίζει το κλειδί του  
    Αρχείου*/)

```

Η συνάρτηση `HP_CreateFile` χρησιμοποιείται για τη δημιουργία και κατάλληλη αρχικοποίηση ενός άδειου αρχείου σωρού με όνομα `fileName`. Σαν είσοδο έχει το όνομα του αρχείου που θα αποθηκευτεί η σχέση και το αντίστοιχο κλειδί. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1.

```
HP_info* HP_OpenFile(char *fileName /*όνομα αρχείου*/)
```

Η συνάρτηση `HP_OpenFile` ανοίγει το αρχείο με όνομα `filename` και διαβάζει από το πρώτο μπλοκ την πληροφορία που αφορά το αρχείο σωρού. Κατόπιν, ενημερώνεται μια δομή που κρατάτε όσες πληροφορίες κρίνονται αναγκαίες για το αρχείο αυτό προκειμένου να μπορείτε να επεξεργαστείτε στη συνέχεια τις εγγραφές του. Μια ενδεικτική δομή με τις πληροφορίες που πρέπει να κρατάτε δίνεται στη συνέχεια:

```
typedef struct {  
    int fileDesc; /*αναγνωριστικός αριθμός ανοίγματος αρχείου από το  
    επίπεδο block*/  
    Record_Attribute keyAttribute; /*Προσδιορίζει το κλειδί του  
    Αρχείου Σωρού*/  
} HP_info;
```

Όπου `keyAttribute` αφορά το πεδίο κλειδί, και `fileDesc` είναι ο αναγνωριστικός αριθμός του ανοίγματος του αρχείου, όπως επιστράφηκε από το επίπεδο διαχείρισης μπλοκ. Αφού ενημερωθεί κατάλληλα η δομή πληροφοριών του αρχείου, την επιστρέφετε. Σε περίπτωση που συμβεί οποιοδήποτε σφάλμα, επιστρέφεται τιμή `NULL`. Αν το αρχείο που δόθηκε για άνοιγμα δεν αφορά αρχείο κατακερματισμού, τότε αυτό επίσης θεωρείται σφάλμα.

```
int HP_CloseFile(HP_info header_info )
```

Η συνάρτηση `HP_CloseFile` κλείνει το αρχείο που προσδιορίζεται μέσα στη δομή `header_info`. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1. Η συνάρτηση είναι υπεύθυνη και για την αποδέσμευση της μνήμης που καταλαμβάνει η δομή που περάστηκε ως παράμετρος, στην περίπτωση που το κλείσιμο πραγματοποιήθηκε επιτυχώς.

```
int HP_InsertEntry( HP_info header_info, /*επικεφαλίδα του αρχείου*/  
    Record record /*δομή που προσδιορίζει την εγγραφή*/)
```

Η συνάρτηση `HP_InsertEntry` χρησιμοποιείται για την εισαγωγή μιας εγγραφής στο αρχείο σωρού. Οι πληροφορίες που αφορούν το αρχείο βρίσκονται στη δομή `header_info`, ενώ η εγγραφή προς εισαγωγή προσδιορίζεται από τη δομή `record`. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφετε τον αριθμό του block στο οποίο έγινε η εισαγωγή (`blockId`), ενώ σε διαφορετική περίπτωση -1.

```
int HP_GetAllEntries( HP_info header_info, /*επικεφαλίδα του αρχείου*/
    void *value /*τιμή του πεδίου-κλειδιού προς αναζήτηση*/)

```

Η συνάρτηση αυτή χρησιμοποιείται για την εκτύπωση όλων των εγγραφών που υπάρχουν στο αρχείο κατακερματισμού οι οποίες έχουν τιμή στο πεδίο-κλειδί ίση με value. Η πρώτη δομή δίνει πληροφορία για το αρχείο κατακερματισμού, όπως αυτή είχε επιστραφεί από την HP_OpenFile. Για κάθε εγγραφή που υπάρχει στο αρχείο και έχει τιμή στο πεδίο-κλειδί (όπως αυτό ορίζεται στην HP_info) ίση με value, εκτυπώνονται τα περιεχόμενά της (συμπεριλαμβανομένου και του πεδίου-κλειδιού). Να επιστρέφεται επίσης το πλήθος των blocks που διαβάστηκαν μέχρι να βρεθούν όλες οι εγγραφές. Σε περίπτωση επιτυχίας επιστρέφει το πλήθος των blocks που διαβάστηκαν, ενώ σε περίπτωση λάθους επιστρέφει -1.

Συναρτήσεις HT (Hash Table)

Στη συνέχεια περιγράφονται οι συναρτήσεις που καλείστε να υλοποιήσετε στα πλαίσια της εργασίας αυτής και που αφορούν το αρχείο στατικού κατακερματισμού.

```
int HT_CreateIndex(char *fileName, /*όνομα αρχείου*/
    Record_Attribute keyAttribute; /*Προσδιορίζει το κλειδί του
Αρχείου*/
    int buckets /*αριθμός κάδων κατακερματισμού*/)

```

Η συνάρτηση HT_CreateIndex χρησιμοποιείται για τη δημιουργία και κατάλληλη αρχικοποίηση ενός άδειου αρχείου κατακερματισμού με όνομα fileName. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1.

```
HT_info* HT_OpenIndex( char *fileName /*όνομα αρχείου*/)

```

Η συνάρτηση HT_OpenIndex ανοίγει το αρχείο με όνομα filename και διαβάζει από το πρώτο μπλοκ την πληροφορία που αφορά το αρχείο κατακερματισμού. Κατόπιν, ενημερώνετε μια δομή όπου κρατάτε όσες πληροφορίες κρίνετε αναγκαίες για το αρχείο αυτό προκειμένου να μπορείτε να επεξεργαστείτε στη συνέχεια τις εγγραφές του. Μια ενδεικτική δομή με τις πληροφορίες που πρέπει να κρατάτε δίνεται στη συνέχεια:

```
typedef struct {
    int fileDesc; /*αναγνωριστικός αριθμός ανοίγματος αρχείου από το
επίπεδο block*/
    Record_Attribute keyAttribute; /*Προσδιορίζει το κλειδί του
Αρχείου*/
    long int numBuckets; /*το πλήθος των "κάδων" του αρχείου
κατακερματισμού*/} HT_info;

```

Όπου attrType, attrName, και attrLength αφορούν το πεδίο κλειδί, fileDesc είναι ο αναγνωριστικός αριθμός του ανοίγματος του αρχείου, όπως επιστράφηκε από το επίπεδο διαχείρισης μπλοκ, και

numBuckets είναι το πλήθος των κάδων που υπάρχουν στο αρχείο. Αφού ενημερωθεί κατάλληλα η δομή πληροφοριών του αρχείου, την επιστρέφετε.

Σε περίπτωση που συμβεί οποιοδήποτε σφάλμα, επιστρέφεται τιμή NULL. Αν το αρχείο που δόθηκε για άνοιγμα δεν αφορά αρχείο κατακερματισμού, τότε αυτό επίσης θεωρείται σφάλμα.

```
int HT_CloseIndex( HT_info* header_info )
```

Η συνάρτηση HT_CloseIndex κλείνει το αρχείο που προσδιορίζεται μέσα στη δομή header_info. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1. Η συνάρτηση είναι υπεύθυνη και για την αποδέσμευση της μνήμης που καταλαμβάνει η δομή που περάστηκε ως παράμετρος, στην περίπτωση που το κλείσιμο πραγματοποιήθηκε επιτυχώς.

```
int HT_InsertEntry( HT_info header_info, /*επικεφαλίδα του αρχείου*/  
                   Record record /*δομή που προσδιορίζει την εγγραφή*/)
```

Η συνάρτηση HT_InsertEntry χρησιμοποιείται για την εισαγωγή μίας εγγραφής στο αρχείο κατακερματισμού. Οι πληροφορίες που αφορούν το αρχείο βρίσκονται στη δομή header_info, ενώ η εγγραφή προς εισαγωγή προσδιορίζεται από τη δομή record. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφετε τον αριθμό του block στο οποίο έγινε η εισαγωγή (blockId) , ενώ σε διαφορετική περίπτωση -1.

```
int HT_GetAllEntries( HT_info header_info, /*επικεφαλίδα του αρχείου*/  
                     void *value /*τιμή του πεδίου-κλειδιού προς αναζήτηση*/)
```

Η συνάρτηση αυτή χρησιμοποιείται για την εκτύπωση όλων των εγγραφών που υπάρχουν στο αρχείο κατακερματισμού οι οποίες έχουν τιμή στο πεδίο-κλειδί ίση με value. Η πρώτη δομή δίνει πληροφορία για το αρχείο κατακερματισμού, όπως αυτή είχε επιστραφεί από την HT_OpenIndex. Για κάθε εγγραφή που υπάρχει στο αρχείο και έχει τιμή στο πεδίο-κλειδί (όπως αυτό ορίζεται στην HT_info) ίση με value, εκτυπώνονται τα περιεχόμενά της (συμπεριλαμβανομένου και του πεδίου-κλειδιού). Να επιστρέφεται επίσης το πλήθος των blocks που διαβάστηκαν μέχρι να βρεθούν όλες οι εγγραφές. Σε περίπτωση επιτυχίας επιστρέφει το πλήθος των blocks που διαβάστηκαν, ενώ σε περίπτωση λάθους επιστρέφει -1.

Άσκηση 2

Συναρτήσεις SHT (Secondary Hash Table)

Στη συνέχεια περιγράφονται οι συναρτήσεις που καλείστε να υλοποιήσετε στα πλαίσια της εργασίας αυτής και που αφορούν τη δημιουργία δευτερεύοντος ευρετηρίου στατικού κατακερματισμού, στο χαρακτηριστικό surname.

Υποθέτουμε ότι υπάρχει ήδη το πρωτεύον ευρετήριο στατικού κατακερματισμού στο id.

```
int SHT_CreateSecondaryIndex(  
    char *sfileName, /*όνομα αρχείου*/  
    Record_Attribute keyAttribute; /*Προσδιορίζει το κλειδί του  
Ευρετηρίου (δευτερεύον κλειδί)*/  
    int buckets, /*αριθμός κάδων κατακερματισμού*/  
    char* fileName /*όνομα αρχείου πρωτεύοντος ευρετηρίου*/)
```

Η συνάρτηση SHT_CreateSecondaryIndex χρησιμοποιείται για τη δημιουργία και κατάλληλη αρχικοποίηση ενός αρχείου δευτερεύοντος κατακερματισμού με όνομα sfileName για το αρχείο πρωτεύοντος κατακερματισμού fileName. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1.

```
SHT_info* SHT_OpenSecondaryIndex( char *sfileName /*όνομα αρχείου*/)
```

Η συνάρτηση SHT_OpenSecondaryIndex ανοίγει το αρχείο με όνομα sfileName και διαβάζει από το πρώτο μπλοκ την πληροφορία που αφορά το δευτερεύον ευρετήριο κατακερματισμού. Κατόπιν, ενημερώνετε μια δομή όπου κρατάτε όσες πληροφορίες κρίνετε αναγκαίες για το αρχείο αυτό προκειμένου να μπορείτε να επεξεργαστείτε στη συνέχεια τις εγγραφές του. Μια ενδεικτική δομή με τις πληροφορίες που πρέπει να κρατάτε δίνεται στη συνέχεια:

```
typedef struct {  
    int fileDesc; /*αναγνωριστικός αριθμός ανοίγματος αρχείου από το  
επίπεδο block*/  
    Record_Attribute keyAttribute; /*Προσδιορίζει το κλειδί του  
Ευρετηρίου (δευτερεύον κλειδί)*/} SHT_info;
```

Όπου attrName, και attrLength αφορούν το πεδίο του δευτερεύοντος ευρετηρίου, fileDesc είναι ο αναγνωριστικός αριθμός του ανοίγματος του αρχείου, όπως επιστράφηκε από το επίπεδο διαχείρισης μπλοκ, numBuckets είναι το πλήθος των κάδων που υπάρχουν στο αρχείο, και fileName είναι το όνομα αρχείου με το πρωτεύον ευρετήριο στο id. Αφού ενημερωθεί κατάλληλα η δομή πληροφοριών του αρχείου, την επιστρέφετε.

Σε περίπτωση που συμβεί οποιοδήποτε σφάλμα, επιστρέφεται τιμή NULL. Αν το αρχείο που δόθηκε για άνοιγμα δεν αφορά αρχείο κατακερματισμού, τότε αυτό επίσης θεωρείται σφάλμα.

```
int SHT_CloseSecondaryIndex( SHT_info* header_info )
```

Η συνάρτηση `SHT_CloseSecondaryIndex` κλείνει το αρχείο που προσδιορίζεται μέσα στη δομή `header_info`. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1. Η συνάρτηση είναι υπεύθυνη και για την αποδέσμευση της μνήμης που καταλαμβάνει η δομή που περάστηκε ως παράμετρος, στην περίπτωση που το κλείσιμο πραγματοποιήθηκε επιτυχώς.

```
int SHT_SecondaryInsertEntry( SHT_info header_info,
    /*επικεφαλίδα του αρχείου*/
    SecondaryRecord record /*δομή που προσδιορίζει την εγγραφή*/)
```

Η συνάρτηση `HT_SecondaryInsertEntry` χρησιμοποιείται για την εισαγωγή μίας εγγραφής στο αρχείο κατακερματισμού. Οι πληροφορίες που αφορούν το αρχείο βρίσκονται στη δομή `header_info`, ενώ η εγγραφή προς εισαγωγή προσδιορίζεται από τη δομή `record`, η οποία πρέπει να περιλαμβάνει το block του πρωτεύοντος ευρετηρίου που υπάρχει η εγγραφή προς εισαγωγή. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1.

Η δομή `SecondaryRecord` η οποία χρησιμοποιείται από τη `SHT_InsertEntry` είναι η ακόλουθη:

```
typedef struct{
    Record record;
    int blockId; //Το block στο οποίο έγινε η εισαγωγή της εγγραφής
    στο πρωτεύον ευρετήριο.}SecondaryRecord;
```

```
int SHT_SecondaryGetAllEntries(SHT_info header_info_sht, /*επικεφαλίδα
του αρχείου δευτερεύοντος ευρετηρίου*/
    HT_info header_info_ht, /*επικεφαλίδα του αρχείου πρωτεύοντος
ευρετηρίου*/
    void *value /*τιμή του πεδίου-κλειδιού προς αναζήτηση*/)
```

Η συνάρτηση αυτή χρησιμοποιείται για την εκτύπωση όλων των εγγραφών που υπάρχουν στο αρχείο κατακερματισμού οι οποίες έχουν τιμή στο πεδίο-κλειδί του δευτερεύοντος ευρετηρίου ίση με `value`. Η πρώτη δομή δίνει πληροφορία για το αρχείο κατακερματισμού, όπως αυτή έχει επιστραφεί από την `SHT_OpenIndex`. Η δεύτερη δομή αντίστοιχα δίνει πληροφορία για το αρχείο κατακερματισμού, όπως αυτή είχε επιστραφεί από την `HT_OpenIndex`. Για κάθε εγγραφή που υπάρχει στο αρχείο και έχει τιμή στο πεδίο-κλειδί (όπως αυτό ορίζεται στην `SHT_info`) ίση με `value`, εκτυπώνονται τα περιεχόμενά της (συμπεριλαμβανομένου και του πεδίου-κλειδιού). Να επιστρέφεται επίσης το πλήθος των blocks που διαβάστηκαν μέχρι να βρεθούν όλες οι εγγραφές. Σε περίπτωση λάθους επιστρέφει -1.

Στατιστικά Κατακερματισμού

Αφού υλοποιήσετε τις συναρτήσεις κατακερματισμού, να τις αξιολογήσετε ως προς:

1. Το πόσα blocks έχει ένα αρχείο,

2. Το ελάχιστο, το μέσο και το μέγιστο πλήθος εγγραφών που έχει κάθε bucket ενός αρχείου,
3. Το μέσο αριθμό των blocks που έχει κάθε bucket, και
4. Το πλήθος των buckets που έχουν μπλοκ υπερχειλίστης, και πόσα μπλοκ είναι αυτά για κάθε bucket.

Για το σκοπό αυτό, να υλοποιήσετε τη συνάρτηση:

```
int HashStatistics( char* filename  
    /*όνομα του αρχείου που ενδιαφέρει*/)

```

Η συνάρτηση διαβάζει το αρχείο με όνομα filename και τυπώνει τα στατιστικά που αναφέρθηκαν προηγουμένως. Σε περίπτωση επιτυχίας επιστρέφει 0, ενώ σε περίπτωση λάθους επιστρέφει -1.

Συνάρτηση main Ελέγχου Λειτουργικότητας

Για να ελέγξετε την ορθότητα της υλοποίησης σας πρέπει να κατασκευάσετε και να παραδώσετε τουλάχιστον μια δική σας συνάρτηση main ελέγχου η οποία θα διεκπεραιώνει τις παρακάτω διαδικασίες:

1. Δημιουργία αρχείου στατικού κατακερματισμού (HT)
2. Δημιουργία δευτερεύοντος ευρετηρίου (SHT) στο πεδίο επίθετο (surname).
3. Εισαγωγή ενδεικτικών εγγραφών (Από τα αρχεία που έχουν δοθεί)
4. Αναζήτηση υποσυνόλου εγγραφών με κάποιο κριτήριο ως προς το πεδίο-κλειδί (id)
5. Αναζήτηση εγγραφών με το πεδίο στο οποίο έγινε η ευρετηρίαση στο δευτερεύον ευρετήριο (αναζήτηση με το πεδίο επίθετο (surname)).

Σημειώσεις:

- Θα πρέπει να ακολουθήσετε πιστά τα πρότυπα συναρτήσεων τα οποία σας δίνονται.
- Εισαγωγή στο δευτερεύον ευρετήριο (SHT) μπορεί να γίνει μόνο εφόσον έχει ολοκληρωθεί η αντίστοιχη διαδικασία στο πρωτεύον (HT).

Παράδοση εργασίας

Η εργασία είναι ομαδική **2 ατόμων**.

Γλώσσα υλοποίησης: **C / C++**

Παραδοτέα

Τα αρχεία πηγαίου κώδικα (sources) και τα αντίστοιχα αρχεία κεφαλίδας (headers), readme αρχείο με περιγραφή / σχόλια πάνω στην υλοποίησή σας. Καθώς και τις αντίστοιχες main συναρτήσεις με τις οποίες δοκιμάσατε την λειτουργικότητα του κώδικα σας.