|  |
| --- |
| עזרא דשט |
| דו"ח לפרוייקט בארגון וניהול קבצים |
| סמסטר ב התשע"א |

|  |
| --- |
| פנחס פדידה 7665692 אבירם מיכאלי 302210224 |

Level 3: Deleting and updating records

## delrec, update, updateoff

For this level, we will still stick to the same classes (*PhysicalFile*, *HashFIle* and the different *Block* structures) and add 3 functions to the *HashFile* class that will together allow us to "physically" delete and to update any record that has already been written into the file, or at least into the buffer: *delrec* deletes a record by overwriting the bits with last record's, *update* updates a record by replacing the bits with a given. Both functions use the *updateflag* parameter in the *HashFile* class in order to lock the record while reading the record using the proper parameter in read (that was written in Level 2). This ensures that following this reading, no parameters will change until the record is unlocked, using the last of the three functions: *updateoff*.

### API Methods

void delrec(void);

Deletes the record that is present in the buffer "physically" (bit-per-bit overwriting) instead of "logically" (marking the bits as invalid): the function read has to be called for update previously (using the third parameter, cf. Level 2), thereby locking the buffer on the record that has just been read. It then proceeds to read the record's key for future use ([see further](#Key)) and overwrites the record's bits with the last record in the block using *memcpy*. The block's and the file header's variables are then updated (*NrOfRecsInBlock* and *LBuffChanged*, *NrOfRecsInFile* and *LHBuffChanged*, respectively). If the record should have originally been in another block (using the key that we previously read), this block is read in the buffer (using *HashFunction* to get its position), and its *NrOfOverflowedRecs* and *LHBuffChanged* variables are then updated accordingly.

void update(char\* record);

This function updates a record by overwriting the old record's bytes with the new ones: as in delrec, the function read has to be called for update previously (using the third parameter, cf. Level 2), thereby locking the buffer on the record that has just been read. Using the pointer to the updated record received as a parameter, it then checks if its key is identical to the new record and continues (attempting to update a record with another record that has a different key is an illegal action). As everything is ready, the old record's bits are overwritten by the updated one's using *memcpy*. The block's LHBuffChanged is then updated to true, as is usual when the buffer is modified and not immediately written.

Parameters:

* record: a pointer to the record that contains the new data in byte shape.

void updateoff(void);

This function unlocks the class that was previously locked for update by turning the updateflag class inner variables from true to false.