```
/*
* DBMS Implementation
*/

#ifndef RECORDOPS_H
#define RECORDOPS_H

#include <string.h>

#include "dbmsproj.h"
#include "recordPtr.h"

// given a buffer and a recordPtr, returns corresponding record

template <typename T> T getRecord(block_t<T> *buffer, recordPtr ptr) {
    return buffer[ptr.block].entries[ptr.record];
};

// given a buffer, a record and a recordPtr, places the record where recordPtr points

template <typename T> void setRecord(block_t<T> *buffer, T rec, recordPtr ptr) {
    buffer[ptr.block].entries[ptr.record] = rec;
};

// given a buffer and 2 recordPtrs, swaps the records where ptrs point

template <typename T> void swapRecords(block_t<T> *buffer, recordPtr ptr1, recordPtr ptr2) {
    T tmp = getRecord(buffer, ptr1);
    setRecord(buffer, getRecord(buffer, ptr2), ptr1);
    setRecord(buffer, tmp, ptr2);
};

// given 2 records and field, compares them
// -1 is returned if rec1 has lower field value than rec2
// 0 is returned if rec1 and rec2 have equal field values
// 1 is returned if rec1 has higher field value than rec2
template <typename T1, typename T2> int compareRecords(T1 &rec1, T2 &rec2, std::string const& field) {
    if (rec1.getCol(field) == rec2.getCol(field)) { /*printf("Two records matched! \n");*/ return 0; }
    else return -1;
}

// hash function for integers

inline unsigned int hashInt(unsigned int num, unsigned int mod, unsigned int seed) {
    num += seed;
    num = (num + 0x7ed55d16) + (num << 12);
    num = (num^0xc761c23c) ^ (num >> 19);
    num = (num + 0x165667b1) + (num << 5);
    num = (num + 0xd3a2646c) ^ (num << 9);
    num = (num + 0xfd7046c5) + (num << 3);
    num = (num^0xb55a4f09) ^ (num >> 16);
    return num % mod;
}

// hash function for strings

```

```
57  inline unsigned int hashString(std::string str_text, unsigned int mod, unsigned int seed)
      {
58
59      char *str = new char[str_text.length() + 1];
60
61      std::strcpy(str, str_text.c_str());
62      str[str_text.length()] = '\0';
63      unsigned long long hash = 5381;
64      int c;
65
66      while ((c = *str++)) {
67          hash = ((hash << 5) + hash) + c;
68      }
69      return hashInt(hash, 8701123, seed) % mod;
70  }
71
72  // given a record and the field of interest, hashes it and returns a value
73  template<typename T> int hashRecord(std::string seed, T rec, unsigned int mod, std::string
       const& field)
74  {
75      unsigned int s = hashString(seed, 8701123, 0);
76      std::string test = rec.getCol(field);
77      return hashString(rec.getCol(field), mod, s);
78  }
79
80  // frees memory allocated for a hash index
81  void destroyHashIndex(linkedRecordPtr **hashIndex, unsigned int size);
82
83  #endif
84
```