ISAM Class – Version .1

Designer: Matthew B. Gately

Table of Contents

Introduction

This class is designed to allow a user to use an ISAM storage system without knowing how it works or requiring them to write file input/output functions. The class is written in a general manner so that it can accept any class or structure as the object to be stored on disk. The class is responsible for the indexing the information, maintain ISAM data file statistics, and providing functions to interact with the data.

Limitations

The current implementation has several inherent limits resulting from the data types used to represent the primary keys and record numbers. We will consider for a moment that a primary key can represent any integer other than -1 allowing for a total of 4,294,967,295 possible unique items. However, while negative record numbers are possible, we will only use positive ones. The maximum number of record numbers is then 2,147,483,647 and thus the maximum number of possible primary keys is also limited to 2,147,483,647 to allow for a 1 to 1 relationship between primary keys and their respective record number.

Memory Considerations

Using the integer data type then for the record number and the primary key each record in the index file takes 8 bytes. This means that 131,072 records can be indexed for every megabyte of memory used. The size per record of the data file is solely dependent on the size of the data being stored. The statistics file takes only 16 bytes as it contains 4 integers.

Classes and Structures

Structure ISAM_stat

This structure is intended to hold key information regarding ISAM file statistics for later review. It contains 4 integers to hold the number new records that have been added, the number of existing records that have been updated, the number of records that have been deleted, and the number of records which have been retrieved.

Structure index_info

This structure holds the primary key and record number for every entry stored in the ISAM data file.

Structure person

This structure is used only as an example object to be stored in the ISAM file in the example program.

Class ISAM

This is the actual ISAM class. It contains all of the functions which interact with the ISAM files.

Source Code

ISAM Header File

```
#include <string>
#include <stdio.h>

using namespace std;


struct index_info {
	int primary_key;
	int record_number;
};

struct ISAM_stat {
	int recordsAdded;
	int recordsUpdated;
	int recordsDeleted;
	int recordsRetrieved;
};

class ISAM {
private:
	// Private Variables
	size_t size;
	string file_name_base;
	string dataFileName;
	string indexFileName;
	string statFileName;
	ISAM_stat stats;

	// Private Functions
	int findRecord(int primary_key);
	int sortIndex(void);
	int addIndexInfo(index_info info);
	double fileSize(FILE *myFile);
public:
	// Public Functions
	ISAM(size_t object_size, string file_name);
	~ISAM();
	int getRecord(void *data, int primary_key);
	int setRecord(void *data, int primary_key);
	int deleteRecord(int primary_key);
	int getNumRecordsAdded();
	int getNumRecordsUpdated();
	int getNumRecordsDeleted();
	int getNumRecordsRetrieved();
	int numIndexRecords();
	int numDataRecords();
	int dataFileWaste();
	void printIndexFile();

};
```

ISAM CPP File

```
#include <string>
#include <stdio.h>
#include <iostream>
#include "ISAM.h"

using namespace std;

// Default Constructor
ISAM::ISAM(size_t object_size, string file_name) {
	size = object_size;
	file_name_base = file_name;
	// Specify File Name For Index
	indexFileName = file_name_base + "_index.bin";
	// Specify File Name For Data
	dataFileName = file_name_base + "_data.bin";
	// Speeify File Name For Statistics
	statFileName = file_name_base + "_stat.bin";

	// Load Stats
	FILE *myFile;
	myFile = fopen(statFileName.c_str(), "r+b");
	if (myFile == NULL) {
		myFile = fopen(statFileName.c_str(), "w+b");
		stats.recordsAdded = 0;
		stats.recordsUpdated = 0;
		stats.recordsRetrieved = 0;
		stats.recordsDeleted = 0;
	} else {
		fread(&stats, sizeof(struct ISAM_stat), 1, myFile);
	}
	fclose(myFile);
}

// Default Destructor
ISAM::~ISAM() {
	// Write Stats Back Out
	FILE *myFile;
	myFile = fopen(statFileName.c_str(), "r+b");
	fwrite(&stats, sizeof(struct ISAM_stat), 1, myFile);
	fclose(myFile);
}

// Return File Size
double ISAM::fileSize(FILE *myFile) {
	double start_location = ftell(myFile);
	if (myFile == NULL)
		return -1;
	fseek(myFile, 0, SEEK_END);
	double file_size = ftell(myFile);
	fseek(myFile, start_location, SEEK_SET);
	return file_size;
}

// Return The Number Of Records In The Data File
int ISAM::numDataRecords() {
	FILE *myFile; // Create a file instance
	myFile = fopen(dataFileName.c_str(), "r+b");
	if (myFile == NULL) {
		return -1;
	}

	double file_size = fileSize(myFile);
	if (file_size == -1)
		return -1;

	return file_size / size;
}

// Return The Number Of Records In The Index File
int ISAM::numIndexRecords() {
	FILE *myFile;
	myFile = fopen(indexFileName.c_str(), "r+b");
	if (myFile == NULL)
		return -1;

	double file_size = fileSize(myFile);
	if (file_size == -1)
```

```
                return -1;

        return file_size / sizeof(struct index_info);
}

// Returns The Ammount of Wasted Space In The Data File
int ISAM::dataFileWaste() {
        return (numDataRecords() - numIndexRecords()) * size;
}

// Find Record Function
int ISAM::findRecord(int primary_key) {
        double file_size = -1;
        double num_records = -1;
        index_info *indexArray;
        FILE *myFile; // Create a file instance
        myFile = fopen(indexFileName.c_str(), "r+b"); // Convert string to a c string and open file
        if (myFile == NULL) {
                return -1;
        }

        // Get File Size
        file_size = fileSize(myFile);
        if (file_size == -1)
                return -1;
        num_records = file_size / sizeof(struct index_info);
        indexArray = new index_info[num_records];

        // Load Array
        fread(indexArray, sizeof(struct index_info), num_records, myFile);

        // Search Array
        for (int counter = 0; counter < num_records; counter++) {
                if (indexArray[counter].primary_key == primary_key)
                        return counter+1;
        }

        fclose(myFile);
        return -1;
}

// Add Index Record Function
int ISAM::addIndexInfo(index_info info) {
        // Load Index File Into Memory
        double file_size = -1;
        int num_records = -1;
        index_info *indexArray;
        FILE *myFile;
        myFile = fopen(indexFileName.c_str(), "r+b");
        if (myFile == NULL)
                return -1;

        file_size = fileSize(myFile);
        if (file_size == -1)
                return -1;
        num_records = file_size / sizeof(struct index_info) + 1;
        indexArray = new index_info[num_records];

        fread(indexArray, sizeof(struct index_info), num_records, myFile);

        // Add New Item To Array
        indexArray[num_records-1].primary_key = info.primary_key;
        indexArray[num_records-1].record_number = info.record_number;

        // Sort Array - Using Bubble Sort For Simplicity
        index_info temp;
        for (int counter1 = num_records; counter1 >0; counter1--) {
                for (int counter2 = 0; counter2 < counter1-1; counter2++) {
                        if (indexArray[counter2].primary_key > indexArray[counter2+1].primary_key) {
                                temp = indexArray[counter2];
                                indexArray[counter2] = indexArray[(counter2+1)];
                                indexArray[(counter2+1)] = temp;
                        }
                }
        }

        // Write Array Back To Binary File
        myFile = fopen(indexFileName.c_str(), "r+b");
        if (myFile == NULL) {
```

```
                myFile = fopen(indexFileName.c_str(), "w+b");
        }
        fwrite(indexArray, sizeof(struct index_info), num_records, myFile); // Write struct or class to
file
        fclose(myFile);

        return 1;
}

// Return Data Record Function
int ISAM::getRecord(void *data, int primary_key) {
        int record = findRecord(primary_key);
        if (record == -1)
                return -1;
        FILE *myFile; // Create a file instance;
        myFile = fopen(dataFileName.c_str(), "r+b"); // Convert string to c string and open file
        fseek (myFile, size * (record-1), SEEK_SET );
        fread(data, size, 1, myFile); // Read data into struct or class from file
        fclose(myFile);
        // Update Stats
        stats.recordsRetrieved++;
        return 1;
}

// Set Data Record Function
int ISAM::setRecord(void *data, int primary_key) {
        FILE *myFile; // Create a file instance
        myFile = fopen(dataFileName.c_str(), "r+b"); // Convert string to c string and open file
        // Check To See If File Opened
        if (myFile == NULL) { // If Not Create New File
                myFile = fopen(indexFileName.c_str(), "w+b");
                fclose(myFile);
                myFile = fopen(dataFileName.c_str(), "w+b");
        }
        int record = findRecord(primary_key);
        if (record == -1) {
                // Update Index
                struct index_info info;
                info.primary_key = primary_key;
                info.record_number = fileSize(myFile) / size + 1;
                if (addIndexInfo(info) == -1)
                        return -1;
                fseek(myFile, 0, SEEK_END);
                // Update Stats
                stats.recordsAdded++;

        } else {
                fseek (myFile, size * (record-1), SEEK_SET );
                // Update Stats
                stats.recordsUpdated++;
        }
        fwrite(data, size, 1, myFile); // Write struct or class to file

        fclose(myFile); // Close File
        return 1;

}

int ISAM::getNumRecordsAdded() {
        return stats.recordsAdded;
}
int ISAM::getNumRecordsUpdated() {
        return stats.recordsUpdated;
}

int ISAM::getNumRecordsDeleted() {
        return stats.recordsDeleted;
}
int ISAM::getNumRecordsRetrieved() {
        return stats.recordsRetrieved;
}

int ISAM::deleteRecord(int primary_key) {
        int record = findRecord(primary_key);
        if (record == -1)
                return -1;

        // Load Index File Into Memory
        double file_size = -1;
```

```
	int num_records = -1;
	index_info *indexArray;
	index_info *finalArray;
	FILE *myFile;
	myFile = fopen(indexFileName.c_str(), "r+b");
	if (myFile == NULL)
		return -1;

	file_size = fileSize(myFile);
	if (file_size == -1)
		return -1;
	num_records = file_size / sizeof(struct index_info);
	indexArray = new index_info[num_records];
	finalArray = new index_info[num_records-1];

	fread(indexArray, sizeof(struct index_info), num_records, myFile);

	// Change Primary Key To Filter To The Front
	for (int counter = 0; counter < num_records; counter++) {
		if (indexArray[counter].primary_key == primary_key)
			indexArray[counter].primary_key = -99999;
	}

	// Sort Array - Using Bubble Sort For Simplicity
	index_info temp;
	for (int counter1 = num_records; counter1 >0; counter1--) {
		for (int counter2 = 0; counter2 < counter1-1; counter2++) {
			if (indexArray[counter2].primary_key > indexArray[counter2+1].primary_key) {
				temp = indexArray[counter2];
				indexArray[counter2] = indexArray[(counter2+1)];
				indexArray[(counter2+1)] = temp;
			}
		}
	}

	// Put Index File Into New Array
	for (int counter = 1; counter < num_records; counter++)
		finalArray[counter-1] = indexArray[counter];

	// Write Array Back To Binary File
	myFile = fopen(indexFileName.c_str(), "w+b");
	if (myFile == NULL) {
		return -1;
	}
	fwrite(finalArray, sizeof(struct index_info), (num_records-1), myFile); // Write struct or class
to file
	fclose(myFile);
	stats.recordsDeleted++;

	return 1;
}

void ISAM::printIndexFile() {
	// Load Index File Into Memory
	double file_size = -1;
	int num_records = -1;
	index_info *indexArray;
	FILE *myFile;
	myFile = fopen(indexFileName.c_str(), "r+b");
	if (myFile == NULL)
		return;

	file_size = fileSize(myFile);
	if (file_size == -1)
		return;
	num_records = file_size / sizeof(struct index_info);
	indexArray = new index_info[num_records];

	fread(indexArray, sizeof(struct index_info), num_records, myFile);


	cout << "Printing Index File" << endl;
	for (int counter = 0; counter < num_records; counter++) {
		cout << "\tRow " << counter << endl;
		cout << "\t\tPrimary Key: " << indexArray[counter].primary_key << endl;
		cout << "\t\tRecord Number: " << indexArray[counter].record_number << endl;
	}

	cout << "Finished Printing Index File" << endl;
```

```
}
```

ISAM Sample Use Program

```
#include <iostream>
#include <stdio.h>
#include <string>
#include "ISAM.h"

using namespace std;

struct person {
	double gpa;
	int idnum;
	char Name[100];
};

void main(void) {
	// Initilize an instanc of the ISAM class
	// Pass The Size of the object it will hold and the base name for the file
	ISAM storage(sizeof(struct person), "sample_ISAM");

	cout << "ISAM Class" << endl;

	person Smith;
	Smith.gpa = 3.34;
	Smith.idnum = 4;
	Smith.Name[0] = 'B';
	Smith.Name[1] = 'C';
	Smith.Name[2] = 'U';
	Smith.Name[3] = '!';
	Smith.Name[4] = '\0';
	// Add or Update a Record
	// Pass the object and the primary key
	storage.setRecord(&Smith, Smith.idnum);


	Smith.gpa = 3.59;
	Smith.idnum = 3;
	Smith.Name[0] = 'A';
	Smith.Name[1] = 'B';
	Smith.Name[2] = 'C';
	Smith.Name[3] = 'D';
	Smith.Name[4] = '\0';
	// Add or Update a Record
	// Pass the object and the primary key
	storage.setRecord(&Smith, Smith.idnum);

	Smith.gpa = 3.94;
	Smith.idnum = 6;
	Smith.Name[0] = 'G';
	Smith.Name[1] = 'H';
	Smith.Name[2] = 'U';
	Smith.Name[3] = '!';
	Smith.Name[4] = '\0';
	// Add or Update a Record
	// Pass the object and the primary key
	storage.setRecord(&Smith, Smith.idnum);

	Smith.gpa = 4.00;
	Smith.idnum = 1;
	Smith.Name[0] = 'W';
	Smith.Name[1] = 'o';
	Smith.Name[2] = 'w';
	Smith.Name[3] = '!';
	Smith.Name[4] = '\0';
	// Add or Update a Record
	// Pass the object and the primary key
	storage.setRecord(&Smith, Smith.idnum);

	Smith.gpa = 3.00;
	Smith.idnum = 11;
	Smith.Name[0] = 'D';
	Smith.Name[1] = 'e';
	Smith.Name[2] = 'l';
	Smith.Name[3] = '!';
	Smith.Name[4] = '\0';
	// Add or Update a Record
	// Pass the object and the primary key
	storage.setRecord(&Smith, Smith.idnum);
```

```
	// Delete a record
	// Pass the primary key
	if (storage.deleteRecord(11) == 1)
		cout << "Record Deleted" << endl;
	else
		cout << "Record NOT Deleted" << endl;


	person Sam;
	// Get a record
	// Pass an object to hold the data and the primary key of the record you want
	if (storage.getRecord(&Sam, 1) != -1)
		cout <<  Sam.gpa << Sam.Name << endl;
	else
		cout << "ERROR: Could Not Read Record" << endl;

	// Dispay the entire index file (this function is for debugging and verification)
	storage.printIndexFile();

	// Display Statistical Information About The ISAM Files
	cout << "Statistical Information:" << endl;
	cout << "\tNumber of records added: " << storage.getNumRecordsAdded() << endl;
	cout << "\tNumber of records updated: " << storage.getNumRecordsUpdated() << endl;
	cout << "\tNumber of records retrieved: " << storage.getNumRecordsRetrieved() << endl;
	cout << "\tNumber of records deleted: " << storage.getNumRecordsDeleted() << endl;

}
```