**CSC 250 – Program 1**

**Database Checker**

**Due – February 27, 2013**

**Overview:**

The local HAM radio club is moving their database from a binary file to an up to date application that uses a proprietary database. They are worried that not all of the data will migrate because the old database did no checking for errors. They have requested that we analyze all the data in the binary file looking for errors that would prevent the data from migrating.

**Starting your program:**

Your program will be run from the command prompt using command line arguments. I have my own files that have different names then what you will test with. You must check that 3 arguments were given after the filename. If the arguments are not found, you will output a useful message to the screen and exit the program. The first argument will be the name of the binary file that has all of the old records to be analyzed. The second argument is the name of a binary file where all the error free records will be written. The third argument is the name of a text file where the records will be written out in human readable form with all the errors that were found under the record.

An example of starting your program.

C:\>\csc250\prog1.exe dbham.bin good.bin bad.txt

**The Record Structure:**

struct Record

{

    char name[30];

    char address[30];

    char city[28];

    char state[2];

    unsigned int zipCode;

    unsigned int birthDate;

    unsigned int licensedDate;

    unsigned int expirationDate;

    char radioClass;

    char callSign[5];

}; // size of struct should be 112 must use #pragma pack(1)

This structure was used to create the binary file that contains the old records. You must use this structure as is with no modifications. In order to read the structure correctly, you will need to put a #pragma pack(1) at the top of every file before any includes. The file has one record written to it followed by another record followed by another ….. **You will need to read to the end of the file processing each record as you go. Do not allocate an array to hold all the records.** You will need to use the read function that is part of the ifstream object to successfully read the structure in and you will use the write function to output the good binary file. Remember, all files passed to any function will be passed by reference.

**No credit will be given to anyone who does not use this structure for reading and writing to and from a binary file.**

**The Fields within the Structure:**

**Name**

The name field contains 30 characters and you guaranteed that it will be a null terminated string. The only characters that the new software will allow to be stored in the new application database are the characters A-Z, a-z, spaces and a period. This field contains the full name of a licensed amateur radio operator.

**Address**

The address field contains 30 characters and you are guaranteed that it is a null terminated string. The only characters that will be allowed are the characters A-Z, a-z, 0-9, spaces, the pound # and the period. This field is the last known address of the operator above.



**City**



The City field contains 28 characters and you are guaranteed that it is a null terminated string. The only characters that will be allowed are the characters A-Z, a-z, spaces and a period.



**State**

The State field contains 2 characters and there is no null terminator within the array. This field needs to match one of the 50 states abbreviations that the post office uses. I.e. South Dakota => SD. The first character is required to be capitalized and the second character can be either case.

**Zip Code**

The zip code is 4 bytes stored in an unsigned integer. Zip codes are in 2 parts. The 5 digit zip code and the 4 digit plus code. I.e. 57702-1745. The range of the 5 digit zip code is 10000 to 99999. The range of the 4 digit plus code that will be permitted is from 1000 to 9999, a zero will also be permitted if the 4 digit plus code is unknown.

Since the Zip code is stored in an integer, you will need to extract the 2 number using bitwise operators. The upper 18 bits are the 5 digit zip code and the lower 14 bits are the 4 digit plus code.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3  0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2  0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |

Note: The post office range for the 4 digit plus code is really 0 to 9999 but we are checking for 1000 – 9999.

**Birth Date**

The birth date is stored in a 4 byte integer. 3 values are stored within the integer. You will need to extract the 3 values using bitwise operators. 0 - 5 bits are the day, bits 6-9 are the month, and the bits 12 – 23 are the year. The bits that are not colored are reserved for future use.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3  0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2  0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |



For a date to be valid, the year needs to be >= 1900 and <= 2015. The month needs a value between 1 and 12. And the number of days depends on the month. If the month is Feb, it has 1-28 days (do not worry about leap years). If the month is April, June, September, or November, it has 1-30 days. The other months have 1 – 31 days. If the month is an invalid one, make sure that the day is between 1 and 31.

**License Date**

The license date is stored in a 4 byte integer and represents the day the person became a licensed ham radio operator. The 3 values are stored within the integer. You will need to extract the 3 values using bitwise operators. 0 - 5 bits are the day, bits 6-9 are the month, and the bits 12 – 23 are the year. The bits that are not colored are reserved for future use.



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3  0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2  0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |

For a date to be valid, the year needs to be >= 1900 and <= 2015. The month needs a value between 1 and 12. And the number of days depends on the month. If the month is Feb, it has 1-28 days (do not worry about leap years). If the month is April, June, September, or November, it has 1-30 days. The other months have 1 – 31 days. If the month is an invalid one, make sure that the day is between 1 and 31.



**Expiration Date**

The expiration date is stored in a 4 byte integer and represents the day the person license becomes invalid. The 3 values are stored within the integer. You will need to extract the 3 values using bitwise operators. 0 - 5 bits are the day, bits 6-9 are the month, and the bits 12 – 23 are the year. The bits that are not colored are reserved for future use.



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3  0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2  0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |



For a date to be valid, the year needs to be >= 1900 and <= 2013. The month needs a value between 1 and 12. And the number of days depends on the month. If the month is Feb, it has 1-28 days (do not worry about leap years). If the month is April, June, September, or November, it has 1-30 days. The other months have 1 – 31 days. If the month is an invalid one, make sure that the day is between 1 and 31.



You must verify that the expiration date comes after the licensed date.

**Radio Class**

The Radio class is 1 character used to identify how advanced the operator is. This character can be upper or lower case and must be one of the following 6 characters, {N,T,P,G,A,X}

(N) Novice

(T) Technician

(P) Technician Plus

(G) General

(A) Advanced

(X) Advanced Extra

**Call Sign**

The call sign is a character array of 5. There is **NO NULL TERMINATOR** in the array. For it to be a valid call sign, the first character needs to be a K,W, or N. the second character needs to be one of the characters A-Z. The middle character must be a digit 0-9, the last 2 characters must be two of the characters A-Z. All 4 characters must be upper case. An example call sign is KV1TP.

**Error Statements:**

If a record is found to contain errors, it is to be output to the text file that was specified on the command line. This record should be outputted in text form (not binary). After outputting the record, all errors should be outputted underneath the record indented at least 5 spaces. The record should be outputted only once and all error messages should be under it. You must use the following error message for the specified which errors you found.

Name Field

* Invalid character in the name field

Address Field

* Invalid character in the address field

City Field

* Invalid character in the city field

State Field

* Invalid State code
* First character of State Code not capitalized

Zip Code

* Invalid 5 digit zip code
* Invalid 4 digit plus code

Birth Date

* Invalid month in Birthday
* Invalid day in Birthday
* Invalid year in Birthday

License Date

* Invalid month in License
* Invalid day in License
* Invalid year in License

Expiration Date

* Invalid month in Expiration
* Invalid day in Expiration
* Invalid year Expiration
* Expiration Date is not after the License Date

Radio Class

* Radio Class code is invalid

Call Sign

* Invalid first character in call sign
* Invalid second, fourth or fifth character in call sign
* Invalid digit in call sign
* Call Sign not upper case

If after checking all fields for errors and none were found, you will output the record to the binary file specified on the command line.

**Bin Tool**

I have provided a tool for you to use. It is called bintool.exe It can be run in 2 modes from the command prompt.

C:\> bintool.exe –v dbham.bin

This will output all records within the binary file using the structure required for this program. This will enable you to check your values with what is actually in the file.

C:\> bintool.exe –a dbham.bin

This will allow you to input data to form records that you require to test the logic of your program. It will append the data to the end of the file.

Note: bintool.exe will create the file if it does not exist even if it is only viewing the records. It also does no error checking enabling you to put errors in records. If the field has 30 characters and is guaranteed to be null terminated, you must make sure that you do not provide more than 30 characters.

**Algorithm**

1. Verify command line arguments
2. Open all files and check for success
3. Read in a record from the file
4. Check for errors
5. If no errors output to the good file
6. If errors were found, output to the bad file
7. Repeat 3.
8. Close files
9. Exit program.

**Hints**

1. Work on program for short periods of time.
2. Consider writing small functions for each error.
3. Possibly use a Boolean array to keep track of errors.
4. When stuck, seek help. Do not look at the code for hours trying to find your error.
5. Do not procrastinate. This makes it miserable trying to get the program done at the last minute.
6. Do your own work.

**Suggestion**

I would consider reading a record from the db file and outputting it to the good file. Then use the bintool.exe to make sure you are reading the data in correctly.

**Project Setup**

You must use 3 files for this project. This will get you use to multiple file project. The files should be named prog1.cpp, functions.cpp, functions.h. functions.h will contain the structure, function prototypes, and any constants / enums / typedefs that you use. Functions.h will be included in both prog1.cpp and functions.cpp.

Prog1.cpp will contain only the main function and should be less than 50 lines long. The remaining functions should be in written in functions.cpp. I will demonstrate how to create a multiple file project when I cover the project.

In functions.h, just after the #pragma pack(1) statement, you will need to put #ifndef preprocessor command. This prevents multiple definitions of the same function.

#pragma pack(1)

#ifndef \_\_FUNCTIONS\_\_H\_\_

#define \_\_FUNCTIONS\_\_H\_\_

// structure here

// typedefs

// enums

// constants

// function prototypes

#endif

**Turning in your program**

The program is due Friday, February 27 at midnight. You need to have all files with doxygen information. Make sure you put a @file tag at the top of every file. You must submit a zip file that contains the three files (prog1.cpp, functions.cpp and functions.h) I will not accept any files named differently or not zipped. After zipping the project, submit the program using the mcs web site.