*Project Report*

**Command Line Utility to perform**

**Tasks on Intel Hex Files**

Name : Sagar Kulkarni

Duration: Dec 2016 – Jan 2017

**ABIOMED Inc.**

**22 Cherry Hill Drive,**

**Danvers, MA 01923 USA**

**Contents:**

*1. Introduction*

*2. User guide*

*3. Design Specification*

*4. Samples Test Outputs*

*5. Report Summary*

**Introduction:**

*Intel Hex format [1]-*

It is a file format that represents binary information in the form of hexadecimal characters. It is generally used in EPROMS and other programmable devices.

Each line of the file encodes multiple binary numbers in Hexadecimal characters. Each line of text is called a record.

*Record:*

A record consists of the following fields:

1. Start code: ‘:’ character which determines the beginning of an Intel Hex record.
2. Byte count: 2 hexadecimal digits represent the number of bytes in the data field.
3. Address: 4 hexadecimal digits represent the offset address of the record. The physical address is calculated by adding the offset address and the base address (predetermined).
4. Record Type: 2 hexadecimal digits represent the record type of the Intel hex record.

a. 00 – Data

b. 01 – End Of File

c. 02 – Extended Segment Address

d. 03 – Start Segment Address

e. 04 – Extended Linear Address

f. 05 – Start Linear Address

1. Data: 2\*(Byte count) number of hexadecimal digits represent the data of the record.
2. Checksum: 2 hexadecimal digits represent the checksum of the record which is used to verify whether the record has errors.

***References:***

*[1] - https://en.wikipedia.org/wiki/Intel\_HEX*

**User Guide:**

*Information about the tool:*

* The tool is written using C++ and compiled using the G++ compiler (egcs- v2.91.51).

*Steps to download and start using the program:*

* Download the executable file named ouput.exe from <https://github.com/sagark1911/upgraded-train> (download *Utility\_Version8.cpp* if required to edit the code/ add more functionality).
* Open command prompt window.
* Set command prompt to the path where the executable is stored.

*Steps to compile the program (Utility\_Version8.cpp):*

- Open command prompt

- Execute ‘g++ Utility\_Version8.cpp –o output’ in the command (‘*output*’ is the name of the executable file generated after compilation)

This tool can be used to perform certain tasks on Intel Hex files:

Usage**: [Name\_of\_exe] [-c file1 file2] [-y file1] [-r file1] [-b file1] [-h file1 byte\_count] [-v file1 max\_byte\_count] [-s file1 string]**

(Name of exe in our case is o*utput.exe*)

Options:

**[-c]** - Compare 2 Intel Hex Binary files based on only the contents in each file.

Input: [Name\_of\_exe -c file1 file2]

Scenario 1: The files are Identical.

Scenario 2: The files are not identical as data at byte 'x' is different.

**[-y]** - Calculate the Cyclic Redundancy Check of one Intel Hex Binary file.

Input: [Name\_of\_exe -y file1]

Output: The CRC of the file (In both Binary and Hexadecimal).

**[-r]** - Verify whether a file is in Intel Hex Binary File format by going through each record and verifying whether it is in the Intel Hex Binary File format.

Input: [Name\_of\_exe -r file1]

Output:

Scenario 1: All the records are in Intel Hex Binary File format.

Scenario 2: Record number 'x' is not in Intel Hex Binary File format.

**[-b]** - Convert the input Intel Hex file into binary format.

Input: [Name\_of\_exe -b file1]

Output: The contents of the file converted to binary format. \*\*

**[-h]** - Convert the input Binary file into Intel Hex records.

Input: [Name\_of\_exe -h file1 No\_of\_bytes]

Output: Intel Hex records with size given by Number of bytes (16 or 32 bytes) and having incremental addresses. \*\*

**[-v]** - Convert the given Intel Hex file to another Intel Hex file with given max byte count limit.

Input: [Name\_of\_exe -v file1 max\_byte\_count]

Output: Intel Hex records with maximum byte count provided in input. \*\*

**[-s]** - Search for the given string in the file.

Input: [Name\_of\_exe -s file1 string]

Output: whether string is present in file1 or not.

***\*\*NOTE:***

***When executing output.exe with [-b] or [-h] or [-v] options, the output in simply printed in the command window. These options require the user to manually redirect the standard stream (STDOUT) to a file as shown below:***

* ***Execute the command “output -b test.hex > binary.bin”***

***\*‘>’ directs the STDOUT stream into the specified file ‘binary.bin’.\****

***- Execute the command “output -h binary.bin 16 > new.hex”***

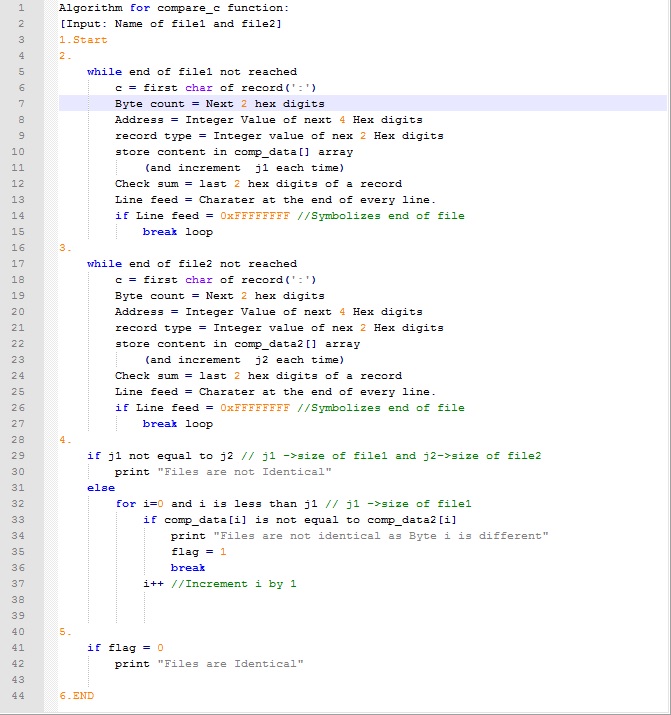
***\*‘>’ directs the STDOUT stream into the specified file ‘new.hex’.\****

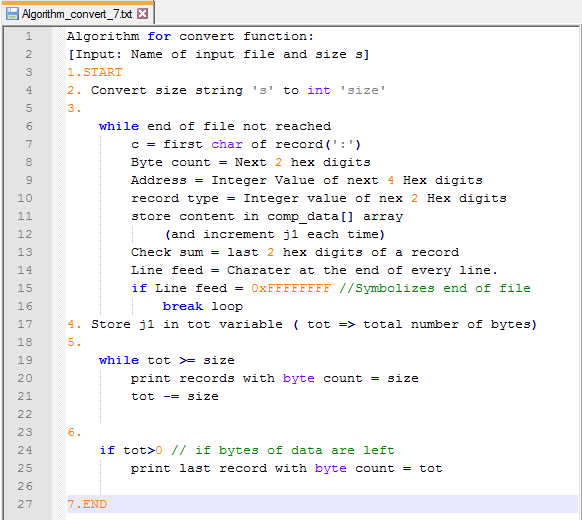
***- Execute the command “output -v test.hex 10 > new1.hex”***

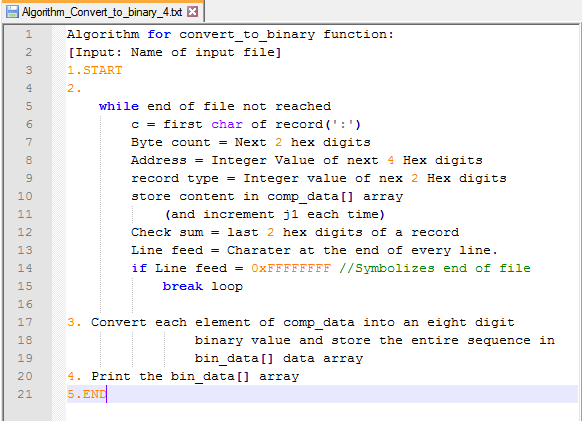
***\*‘>’ directs the STDOUT stream into the specified file ‘new.hex’.\****

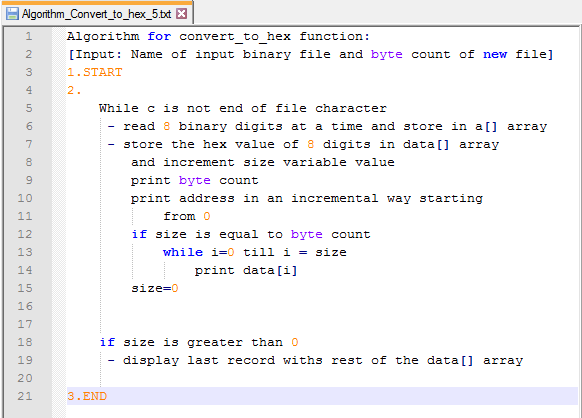
**Design Specification:**

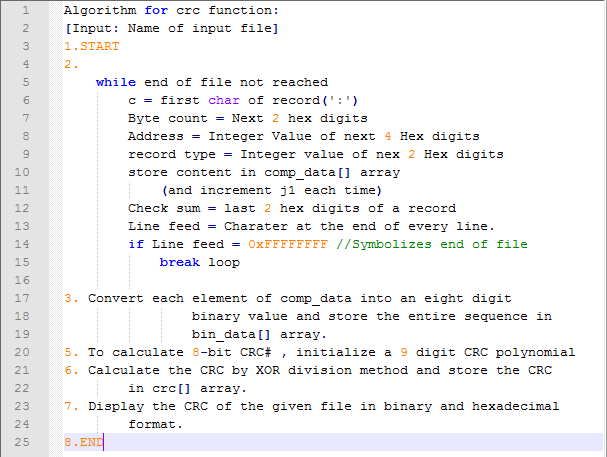
Includes a Step form algorithm explaining each function (option) used in the program.





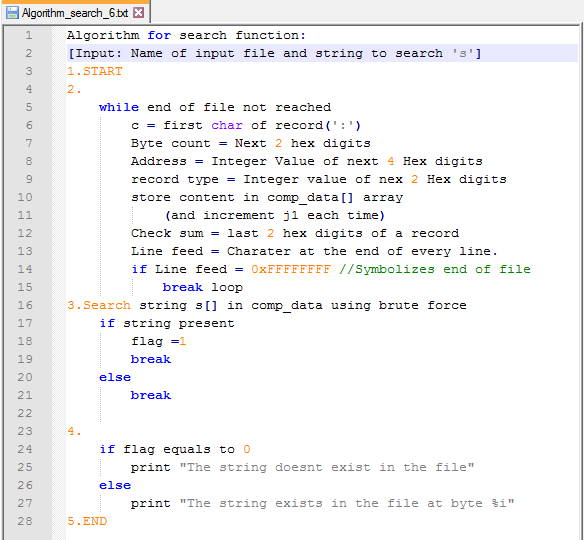


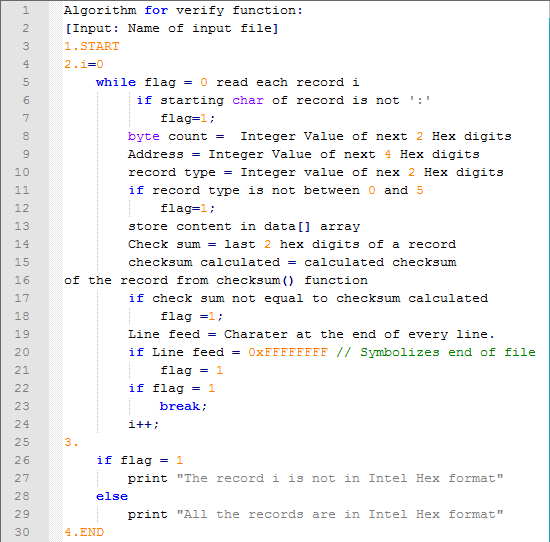




***References:***

*# - https://en.wikipedia.org/wiki/Cyclic\_redundancy\_check*

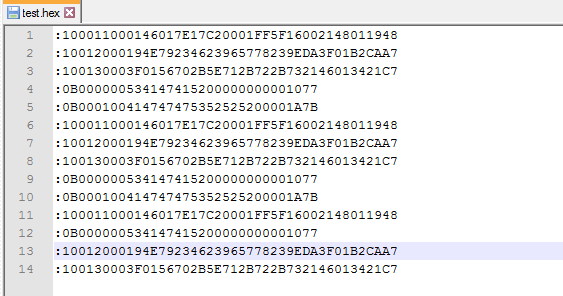
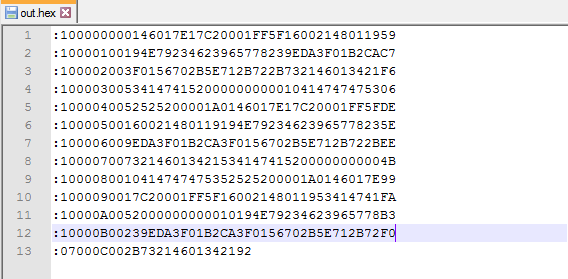
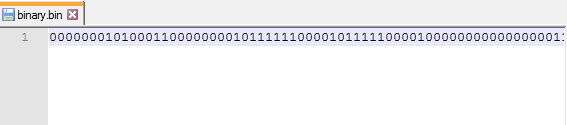




These step-form algorithms explain the options of the command line utility and are useful in understanding the code as well.

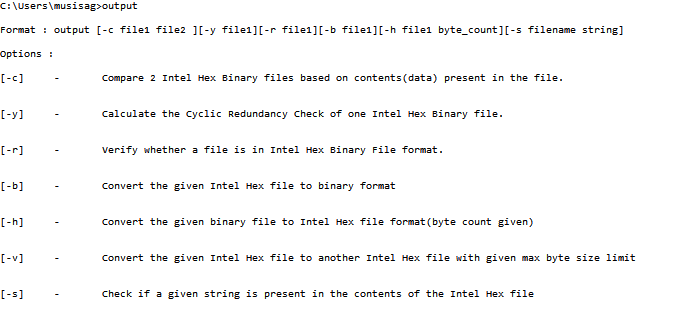
**Sample Test Outputs**

* Contents of binary.bin, out.hex, test.hex



As the program is written in C++ we need to compile it before running the exe file.

* We use the *‘g++’* compiler and use the *‘-o’* option to name the exe as *‘output’* (it is saved as *output.exe*).
* Running the exe (*output*) without any options displays the screen below, which contains the list of all the options and the respective formats required to use them.

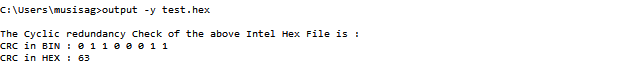


(1)*[-c filename1 filename2]* comparing 2 hex files



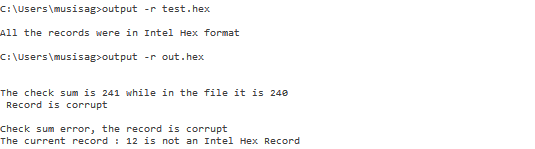
- compares the contents (data) of 2 Intel Hex Files

(2)*[-y filename]* calculating the CRC



* Calculates the CRC of the entire contents of the file.

(3)*[-r filename]* verify if it is in Intel hex format



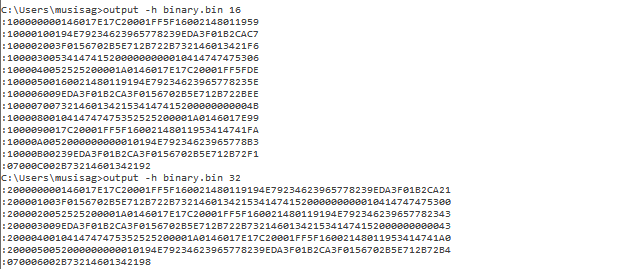
* To verify whether the current file is in Intel Hex format.

(4)*[-b filename]* convert Intel Hex file to binary file

4.png

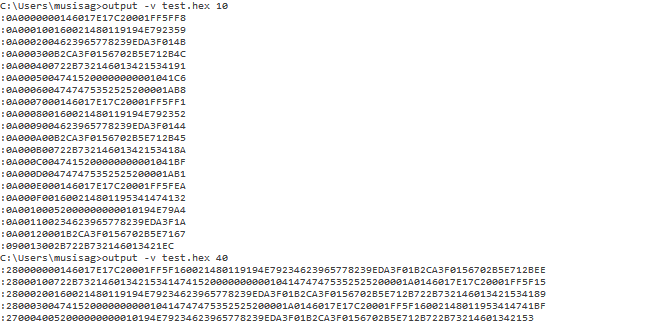
* Convert the contents(only data) of the Intel Hex file into binary format and store it into binary.bin

(5)*[-h filename 16/32]* convert the binary file into Intel Hex file of 16/32 byte format



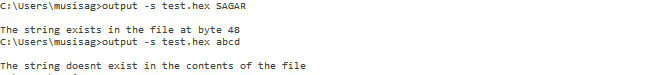
* Convert the file, binary.bin to Intel Hex format with records of size 16 and 32 bytes respectively (this output can be saved into another file).

(6)*[-v filename max\_byte\_count]* convert an Intel Hex file to another Intel hex file format with specified max\_byte\_count.



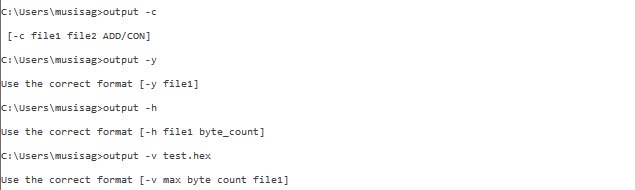
* Converts an Intel hex file ‘test.hex’ to another set of records with max\_byte\_count = 10 and max\_byte\_count = 40 and displays the records (This output can be saved into another file).

(7*)[-s filename string]* checks whether the string is present in the file.



*In case wrong option/ format has been entered:*

l.png



**Report Summary:**

In this project a Command Line Utility program has been created using C++ to perform tasks on Intel Hex files. Some of these tasks include comparing two Intel hex files, calculating the CRC (Cyclic Redundancy Check) of an Intel hex file, converting an Intel hex file into a binary file and vice versa, etc., after the completion of a new functionality an incremental version has been uploaded on to a github repository (<https://github.com/sagark1911/upgraded-train>) and the changes made are specified. ‘Utility\_Version8.cpp” is the 8th and the final version of the Utility program and contains 7 unique options for the command line utility.

This project report contains a user guide which tells the end user how to use the program and from where to download it. This user guide also includes the different formats required for each option (function).

Next, step-form algorithms have been provided for each option (function) to allow the end user to understand the function or code so that it’ll be easier for him/her to make changes and add additional functionalities to the program.

Finally, sample test cases have been provided for each of the options with the contents of the files the utility program is used on.