



Version 1.0 Manual  
By Matthew Mikolay

`matt.mik@hotmail.com`  
`mattmik.com`



The chasm Version 1.0 Manual is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Note: The program presented in this manual (chasm) is licensed under the GNU General Public License Version 3, *not* the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

**Table of Contents**

About the Author	4
Introduction	5
Features	6
Supported Instructions	7
Example Output	8
Proper Usage	9
Design Description	10
Flowchart	12
Source Code/Compiling	13
License	42

## About the Author

*December 24<sup>th</sup>, 2010*

Matthew James Mikolay is currently a student living in New Jersey. His interests include computer science, and specifically, the design and interpretation of programming languages.

Matthew has a strong passion for vintage computers and their history. He is currently a member of the MidAtlantic Retro Computing Hobbyists (MARCH).

Matthew Mikolay is the founder and maintainer for the RCA COSMAC VIP Yahoo! Group (<http://groups.yahoo.com/group/rcacosmac>), which is dedicated to the preservation of the COSMAC VIP and related computers.



Matthew James Mikolay

<http://www.mattmik.com/>  
[matt.mik@hotmail.com](mailto:matt.mik@hotmail.com)

## Introduction

CHIP-8 is an interpreted programming language initially developed by Joseph Weisbecker for the COSMAC VIP computer in 1977. Created to simplify the programming of video games, CHIP-8 was popular and portable for the computers of its generation. As a result, a variety of games have been developed in the CHIP-8 language.



An example of a game programmed in CHIP-8\*

CHIP-8 allows graphical output to a sixty-four by thirty-two monochrome pixel display. One sound timer triggers the playing of a monotone frequency, and one delay timer can be used for scheduling. A sixteen-key hexadecimal keypad is used for input. Sixteen eight-bit data registers can be used to store data, and the sixteen-bit address register can be used to store a memory address.

The portability of the CHIP-8 language is due to the fact that it is an interpreted hexadecimal language. All CHIP-8 instructions are hexadecimal numbers and can be easily stored and read in memory. However, programming in CHIP-8 has often been perceived as a difficult task due to this hexadecimal format, as the purpose of each instruction in a program is not immediately evident. Because of this, the need for a CHIP-8 pseudo-assembler arises.

The CHIP-8 language is an interpreted programming language, as its instructions are read by an interpreting program, which then executes corresponding code on the host computer. chasm presents the CHIP-8 language to a programmer using a system of easy-to-read and remember mnemonics, which are then translated into the traditional interpreted CHIP-8 opcodes. Because the chasm mnemonics are translated into an interpreted language rather than a machine language, chasm is known as a pseudo-assembler instead of a regular assembler.

Although this difference may be important, it over-complicates the matters that this manual concerns. Therefore, the mnemonic language used by chasm will be identified as assembly language from this point on, and the interpreted programming language that chasm outputs will be identified as machine language or machine code.

---

\* Astro Dodge programming and graphics by Martijn Wenting, Revival Studios, [www.revival-studios.com](http://www.revival-studios.com)

## Features

chasm is a pseudo-assembler for the CHIP-8 programming language. It was designed to accept a text file containing 'assembly language' mnemonics as input, and output the resulting CHIP-8 'machine code' to a separate file.

Version 1.0 of chasm supports all thirty-five original CHIP-8 commands defined by Joseph Weisbecker for the COSMAC VIP computer. These commands and their corresponding assembly language mnemonics are found in the table on the next page.

chasm also supports an additional command called the `.START` command, used to specify the memory address at which the resulting CHIP-8 program should be loaded on the host machine. This designated value is used by chasm to determine the values of label addresses during the label linking process. Although the `.START` command is completely optional, it should be the first command found in the input file when present. This additional command is highlighted in red in the table on the next page.

chasm supports two other commands, the `DB` and `DW` commands, which accept an 8-bit value and a 16-bit value respectively as arguments. These commands insert the given argument into the generated output code at the corresponding address, and can be used to insert graphics data into CHIP-8 assembly source code. These additional commands are highlighted in red in the table on the next page.

## Supported Instructions

- Code in { } brackets designate optional parameters for an instruction.
- $V_x$  and  $V_y$  are register names,  $kk$  is a byte,  $nnn$  is an address,  $n$  is a nibble.
- Mnemonics in red represent commands specific to chasm, and not implemented by the original CHIP-8 specification.

Opcode	Mnemonic
00E0	CLS
00EE	RET
0nnn	SYS <addr>
1nnn	JP <addr>
2nnn	CALL <addr>
3xkk	SE <Vx>, <byte>
4xkk	SNE <Vx>, <byte>
5xy0	SE <Vx>, <Vy>
6xkk	LD <Vx>, <byte>
7xkk	ADD <Vx>, <byte>
8xy0	LD <Vx>, <Vy>
8xy1	OR <Vx>, <Vy>
8xy2	AND <Vx>, <Vy>
8xy3	XOR <Vx>, <Vy>
8xy4	ADD <Vx>, <Vy>
8xy5	SUB <Vx>, <Vy>
8xy6	SHR <Vx> {, <Vy>}
8xy7	SUBN <Vx>, <Vy>
8xyE	SHL <Vx> {, <Vy>}

Opcode	Mnemonic
9xy0	SNE <Vx>, <Vy>
Annn	LD I, <addr>
Bnnn	JP V0, <addr>
Cxkk	RND <Vx>, <byte>
Dxyn	DRW <Vx>, <Vy>, <nibble>
Ex9E	SKP <Vx>
ExA1	SKNP <Vx>
Fx07	LD <Vx>, DT
Fx0A	LD <Vx>, K
Fx15	LD DT, <Vx>
Fx18	LD ST, <Vx>
Fx1E	ADD I, <Vx>
Fx29	LD F, <Vx>
Fx33	LD B, <Vx>
Fx55	LD [I], <Vx>
Fx65	LD <Vx>, [I]
----	<b>.START &lt;addr&gt;</b>
----	<b>DB &lt;byte&gt;</b>
----	<b>DW &lt;word&gt;</b>

### Example Output

The following table contains a side-by-side comparison of two files: `input.asm`, a text file containing assembly language mnemonics, and `output.c8`, a data file containing corresponding CHIP-8 machine language opcodes. Each line (or lines) of `input.asm` has its corresponding CHIP-8 opcode printed directly adjacent under the `output.c8` column. `input.asm` is printed as if viewed in a standard ASCII text editor, and `output.c8` is printed as if viewed in a hexadecimal editor with a byte-span value of 2. It should be noted that CHIP-8 commands are stored using big-endian mode, with the most-significant byte first and the least-significant byte last.

input.asm		output.c8
START:	CLS	00E0
	RND V0, #FF	C0FF
	LD I, #224	A224
	LD B, V0	F033
	LD V2, [I]	F265
	LD F, V0	F029
	LD V0, #00	6000
	LD V3, #00	6300
	DRW V0, V3, 5	D035
	LD F, V1	F129
	LD V0, #05	6005
	DRW V0, V3, 5	D035
	LD F, V2	F229
	LD V0, 10	600A
	DRW V0, V3, 5	D035
	LD V0, K	F00A
	JP START	1200
	DB #FF	FFEA
	DB #EA	
	DW #21AC	21AC



### Proper Usage

chasm accepts two files: an input file and an output file. The lines stored in the input file are read and converted into corresponding CHIP-8 code, which is then stored in the output file. If an error occurs while opening either the input or output file, an error message is displayed to the user.

The proper syntax for chasm is:

```
chasm <input filename> <output filename>
```

If this syntax is not followed, an error message, along with the guidelines for the proper syntax, are displayed to the user.

## Design Description

A simplified summary of the overall design of chasm is presented in a flow chart on page 10.

### Initialization

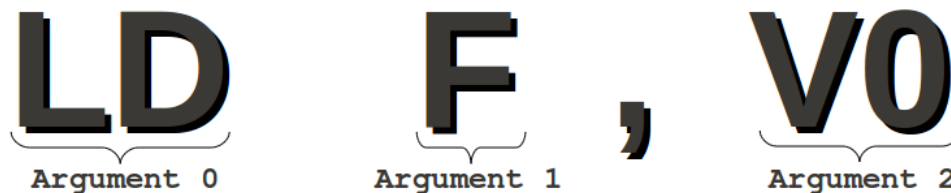
Upon startup, chasm checks to make sure the correct amount of command line arguments were entered and processes them accordingly. chasm is called with the following syntax:

```
chasm <input filename> <output filename>
```

chasm attempts to open the file passed by the user as input.

### Input Processing

chasm processes the assembly language commands found in the input file by looping through the file and processing each individual line, separating it into sections called “arguments.” An argument is any part of an assembly command. The following diagram provides an example:



While separating each line into arguments, chasm checks if a label has been included. If one is found, a label is created, and the corresponding argument is removed from the argument array. This allows processing of the line to continue regardless of the created label.

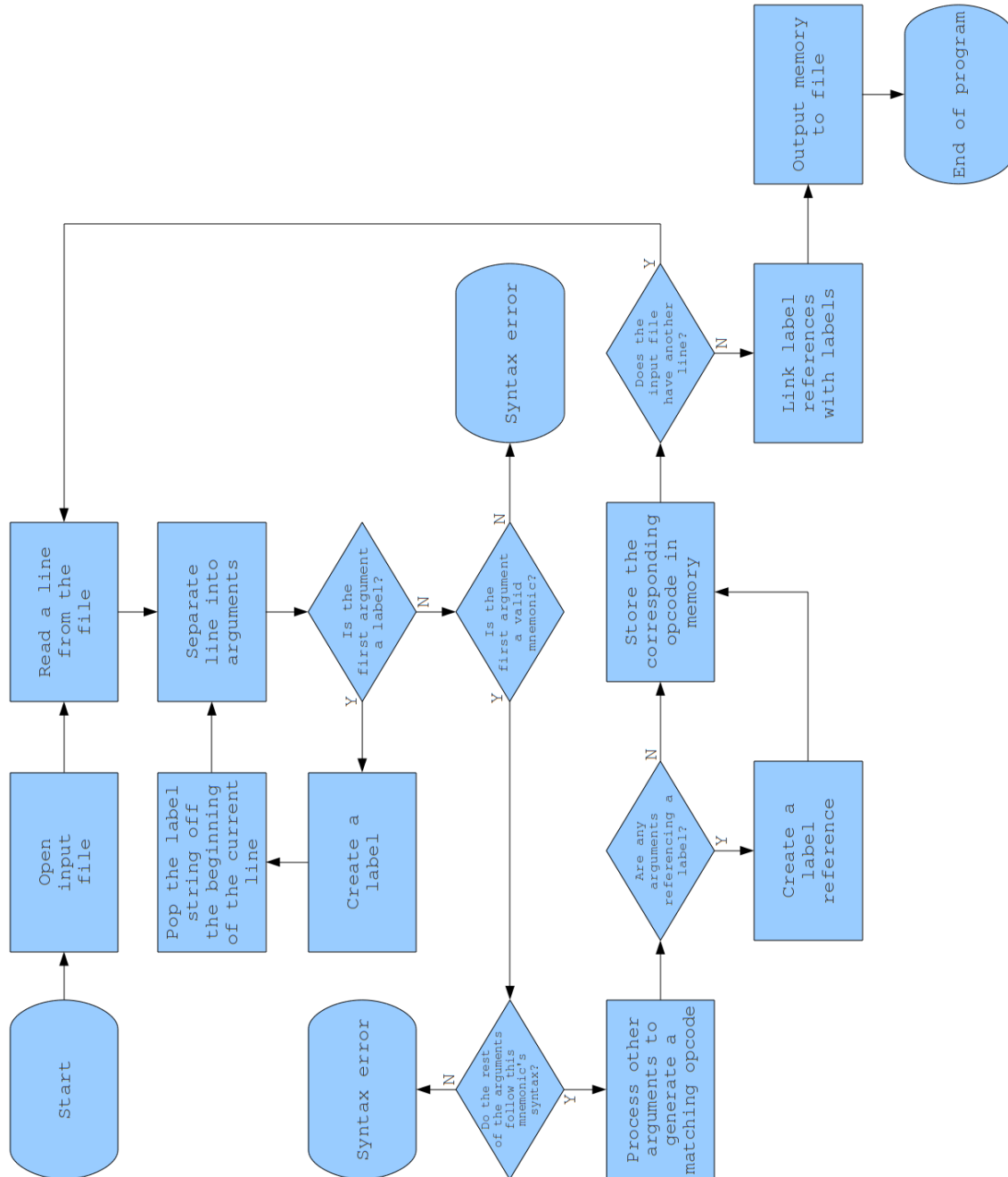
After the line has been separated into arguments and any present labels have been removed, chasm begins processing the arguments and generating the resulting machine code. If an error is found in the syntax of the arguments, or the assembler undergoes an error, relevant information is printed to the screen.

**Label Linking**

After the entire input file has been processed, any references to labels made by assembly language commands will have been stored in a label reference array. chasm loops through this array, checking to make sure that each label referenced actually exists, and linking the identified labels with their corresponding memory addresses.

**File Output**

chasm has now finished generating machine code corresponding to the assembly instructions found in the input file. This machine code is sent to the designated output file for storage, and the program reaches completion.



## Source Code/Compiling

The following pages contain the complete and unabridged source code for version 1.0 of chasm. Much of the code is documented through the use of in-code comments. The following table describes the functions of the various source files.

chasm.h	<ul style="list-style-type: none"> <li>■ The primary header file for chasm</li> <li>■ Contains functions to perform the following:               <ul style="list-style-type: none"> <li>○ Check for whitespace</li> <li>○ Convert a lowercase character to uppercase</li> <li>○ Output an error</li> <li>○ Add a command argument to an array</li> <li>○ Check if a string is numeric</li> <li>○ Convert strings to numeric data and register data</li> </ul> </li> </ul>	Page 14
label.h	<ul style="list-style-type: none"> <li>■ The header file defining the label class</li> </ul>	Page 19
lref.h	<ul style="list-style-type: none"> <li>■ The header file defining the label reference class</li> </ul>	Page 21
chasm.cpp	<ul style="list-style-type: none"> <li>■ The primary source code file for chasm</li> </ul>	Page 22

chasm should be compiled using a standard C++ compiler. The GNU C++ Compiler (G++) of the GNU Compiler Collection (GCC) is strongly recommended. `chasm.cpp` should be in the same directory as the header files when compiling.

**chasm.h**


---

```

1 #include <string>           // For the string class
2 #include <sstream>         // For conversions from string to number
3 using namespace std;
4
5 /*
6     Copyright 2010 Matthew Mikolay
7
8     This file is part of chasm.
9
10    chasm is free software: you can redistribute it and/or modify
11    it under the terms of the GNU General Public License as published by
12    the Free Software Foundation, either version 3 of the License, or
13    (at your option) any later version.
14
15    chasm is distributed in the hope that it will be useful,
16    but WITHOUT ANY WARRANTY; without even the implied warranty of
17    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
18    GNU General Public License for more details.
19
20    You should have received a copy of the GNU General Public License
21    along with chasm. If not, see <http://www.gnu.org/licenses/>.
22 */
23
24 // Returns true if a given character is whitespace
25 bool isWhitespace(char letter)
26 {
27     return (letter == ' ' || letter == '\t');
28 }
29
30 // Converts a given character to uppercase if the input is in lowercase
31 char toUpper(char letter)
32 {
33     if(letter >= 0x61 && letter <= 0x7A)
34         return (letter - 0x20);
35     return letter;
36 }
37
38 // Writes to the screen that an error occurred
39 bool error(string filename, unsigned int lineNumber, string message)
40 {
41     cout << filename << ":" << lineNumber << ":" << message << endl;
42     return true;
43 }
44
45 // Try to add an argument to the array of arguments, making sure no overflow occurs
46 bool addArg(string &input, string (&args)[4], unsigned char &number)
47 {
48     // Make sure the maximum number of arguments is not already in the array
49     if(number >= 4)
50         return false;
51
52     // Store the word, increment the counter, and clear the contents of word
53     args[number] = input;
54     number++;
55     input = "";
56     return true;
57 }
58
59 // Returns true if the given string is a number
60 bool isNumeric(string input)
61 {
62     if(input.at(0) == '#' || input.at(0) == '$')
63         return true;
64 }

```

```

65
66 // Loop through the contents of the string, looking for non-numeric characters
67 for(unsigned int i = 0; i < input.length(); i++)
68 {
69     if(input.at(i) < '0' || input.at(i) > '9')
70         return false;
71 }
72
73 return true;
74 }
75
76 // Convert a given string to a register number
77 bool strToRegister(string input, unsigned char &output)
78 {
79     if(input.length() == 2 && input.at(0) == 'V')
80     {
81         // Is the register a number register?
82         if(input.at(1) >= 'A' && input.at(1) <= 'F')
83         {
84             output = input.at(1) - 55;
85             return true;
86         }
87         // Is the register a letter register?
88         if(input.at(1) >= '0' && input.at(1) <= '9')
89         {
90             output = input.at(1) - '0';
91             return true;
92         }
93     }
94     return false;
95 }
96
97 // Convert a given string to a 4-bit number (nibble)
98 bool strToNibble(string input, unsigned char &output)
99 {
100     if(input.length() > 0)
101     {
102         // Process the input as hexadecimal
103         if(input.at(0) == '#')
104         {
105             unsigned int result = 0;
106             // Remove the hash
107             input = input.substr(1, input.length());
108             // Attempt to convert the string to a number
109             istringstream iss(input);
110
111             // Make sure the conversion did not fail and the result is a 4-bit number
112             if((iss >> hex >> result).fail() || result >= 16)
113                 return false;
114
115             output = result;
116             return true;
117         }
118
119         // Process the input as binary
120         if(input.at(0) == '$')
121         {
122             unsigned int result = 0;
123             // Loop through the string computing the binary number
124             for(unsigned char i = input.length() - 1; i > 0; i--)
125             {
126                 if(input.at(i) == '1')
127                     result += 1 << (input.length() - 1 - i);
128                 else if(input.at(i) != '0' && input.at(i) != '.')
129                     return false;
130             }
131
132             // Make sure the result is a 4-bit number
133             if(result >= 16)
134                 return false;
135

```

```

136         output = result;
137         return true;
138     }
139
140     // Try to process the input as decimal
141     else
142     {
143         unsigned int result = 0;
144         // Attempt to convert the string to a number
145         istream iss(input);
146
147         // Make sure the conversion did not fail and the result is a 4-bit number
148         if((iss >> dec >> result).fail() || result >= 16)
149             return false;
150
151         output = result;
152         return true;
153     }
154 }
155 return false;
156 }
157
158 // Convert a given string to an 8-bit number (byte)
159 bool strToByte(string input, unsigned char &output)
160 {
161     if(input.length() > 0)
162     {
163         // Process the input as hexadecimal
164         if(input.at(0) == '#')
165         {
166             unsigned int result = 0;
167             // Remove the hash
168             input = input.substr(1, input.length());
169             // Attempt to convert the string to a number
170             istream iss(input);
171
172             // Make sure the conversion did not fail and the result is an 8-bit number
173             if((iss >> hex >> result).fail() || result >= 256)
174                 return false;
175
176             output = result;
177             return true;
178         }
179
180         // Process the input as binary
181         if(input.at(0) == '$')
182         {
183             unsigned int result = 0;
184             // Loop through the string computing the binary number
185             for(unsigned char i = input.length() - 1; i > 0; i--)
186             {
187                 if(input.at(i) == '1')
188                     result += 1 << (input.length() - 1 - i);
189                 else if(input.at(i) != '0' && input.at(i) != '.')
190                     return false;
191             }
192
193             // Make sure the result is an 8-bit number
194             if(result >= 256)
195                 return false;
196
197             output = result;
198             return true;
199         }
200
201         // Try to process the input as decimal
202     else
203     {
204         unsigned int result = 0;
205         // Attempt to convert the string to a number
206         istream iss(input);

```



```

207
208 // Make sure the conversion did not fail and the result is an 8-bit number
209 if((iss >> dec >> result).fail() || result >= 256)
210     return false;
211
212 output = result;
213 return true;
214 }
215 }
216 return false;
217 }
218
219 // Convert a given string to a 12-bit number
220 bool strTo12Bit(string input, unsigned short &output)
221 {
222     if(input.length() > 0)
223     {
224         // Process the input as hexadecimal
225         if(input.at(0) == '#')
226         {
227             unsigned int result = 0;
228             // Remove the hash
229             input = input.substr(1, input.length());
230             // Attempt to convert the string to a number
231             istringstream iss(input);
232
233             // Make sure the conversion did not fail and the result is a 12-bit number
234             if((iss >> hex >> result).fail() || result >= 4096)
235                 return false;
236
237             output = result;
238             return true;
239         }
240
241         // Process the input as binary
242         if(input.at(0) == '$')
243         {
244             unsigned int result = 0;
245             // Loop through the string computing the binary number
246             for(unsigned char i = input.length() - 1; i > 0; i--)
247             {
248                 if(input.at(i) == '1')
249                     result += 1 << (input.length() - 1 - i);
250                 else if(input.at(i) != '0' && input.at(i) != '.')
251                     return false;
252             }
253
254             // Make sure the result is a 12-bit number
255             if(result >= 4096)
256                 return false;
257
258             output = result;
259             return true;
260         }
261
262         // Try to process the input as decimal
263         else
264         {
265             unsigned int result = 0;
266             // Attempt to convert the string to a number
267             istringstream iss(input);
268
269             // Make sure the conversion did not fail and the result is a 12-bit number
270             if((iss >> dec >> result).fail() || result >= 4096)
271                 return false;
272
273             output = result;
274             return true;
275         }
276     }
277     return false;

```

```

278 }
279
280 // Convert a given string to a 16-bit number
281 bool strToWorld(string input, unsigned short &output)
282 {
283     if(input.length() > 0)
284     {
285         // Process the input as hexadecimal
286         if(input.at(0) == '#')
287         {
288             unsigned int result = 0;
289             // Remove the hash
290             input = input.substr(1, input.length());
291             // Attempt to convert the string to a number
292             istringstream iss(input);
293
294             // Make sure the conversion did not fail and the result is a 16-bit number
295             if((iss >> hex >> result).fail() || result >= 65536)
296                 return false;
297
298             output = result;
299             return true;
300         }
301
302         // Process the input as binary
303         if(input.at(0) == '$')
304         {
305             unsigned int result = 0;
306             // Loop through the string computing the binary number
307             for(unsigned char i = input.length() - 1; i > 0; i--)
308             {
309                 if(input.at(i) == '1')
310                     result += 1 << (input.length() - 1 - i);
311                 else if(input.at(i) != '0' && input.at(i) != '.')
312                     return false;
313             }
314
315             // Make sure the result is a 16-bit number
316             if(result >= 65536)
317                 return false;
318
319             output = result;
320             return true;
321         }
322
323         // Try to process the input as decimal
324         else
325         {
326             unsigned int result = 0;
327             // Attempt to convert the string to a number
328             istringstream iss(input);
329
330             // Make sure the conversion did not fail and the result is a 16-bit number
331             if((iss >> dec >> result).fail() || result >= 65536)
332                 return false;
333
334             output = result;
335             return true;
336         }
337     }
338     return false;
339 }

```

**label.h**


---

```

1  /*
2     Copyright 2010 Matthew Mikolay
3
4     This file is part of chasm.
5
6     chasm is free software: you can redistribute it and/or modify
7     it under the terms of the GNU General Public License as published by
8     the Free Software Foundation, either version 3 of the License, or
9     (at your option) any later version.
10
11    chasm is distributed in the hope that it will be useful,
12    but WITHOUT ANY WARRANTY; without even the implied warranty of
13    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
14    GNU General Public License for more details.
15
16    You should have received a copy of the GNU General Public License
17    along with chasm. If not, see <http://www.gnu.org/licenses/>.
18
19 */
20
21 // A class to store label data
22 class label
23 {
24     private:
25         string name;                // A name to identify the label
26         unsigned short address;     // The memory address associated
27                                     // with the label name
28     public:
29         label(string paramName, unsigned short paramAddress);
30         label();
31         bool isValid();
32         string getName();
33         unsigned short getAddress();
34         ~label();
35 };
36
37 // Label constructor
38 label::label(string paramName, unsigned short paramAddress)
39 {
40     name = paramName;
41     address = paramAddress;
42 }
43
44 // Label constructor
45 label::label()
46 {
47 }
48
49 // Returns true if the label has a valid identifying name
50 // Labels cannot be numbers of any form, including binary, hexadecimal, and decimal.
51 // This would interfere with the jump command ("JMP") and prevent
52 // jumping to specific addresses.
53 bool label::isValid()
54 {
55     return !isNumeric(name);
56 }
57
58 // Return the label name
59 string label::getName()
60 {
61     return name;
62 }
63
64 // Return the label address

```

```
65 unsigned short label::getAddress()  
66 {  
67     return address;  
68 }  
69  
70 // Label destructor  
71 label::~~label() { }
```

**lref.h**


---

```

1  /*
2     Copyright 2010 Matthew Mikolay
3
4     This file is part of chasm.
5
6     chasm is free software: you can redistribute it and/or modify
7     it under the terms of the GNU General Public License as published by
8     the Free Software Foundation, either version 3 of the License, or
9     (at your option) any later version.
10
11    chasm is distributed in the hope that it will be useful,
12    but WITHOUT ANY WARRANTY; without even the implied warranty of
13    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
14    GNU General Public License for more details.
15
16    You should have received a copy of the GNU General Public License
17    along with chasm. If not, see <http://www.gnu.org/licenses/>.
18
19 */
20
21 // A class to store label reference data
22 class lref
23 {
24     private:
25         string name;                // The name of the referenced label
26         unsigned short address;     // The memory address of
27                                     // this reference
28     public:
29         lref(string paramName, unsigned short paramAddress);
30         lref();
31         string getName();
32         unsigned short getAddress();
33         ~lref();
34 };
35
36 // Label reference constructor
37 lref::lref(string paramName, unsigned short paramAddress)
38 {
39     name = paramName;
40     address = paramAddress;
41 }
42
43 // Label reference constructor
44 lref::lref()
45 {
46 }
47
48 // Return the label reference name
49 string lref::getName()
50 {
51     return name;
52 }
53
54 // Return the label reference address
55 unsigned short lref::getAddress()
56 {
57     return address;
58 }
59
60 // Label reference destructor
61 lref::~~lref(){ }
```

**chasm.cpp**


---

```

1 // Standard C++ libraries
2 #include <fstream>
3 #include <iostream>
4
5 // chasm specific libraries
6 #include "chasm.h"
7 #include "label.h"
8 #include "lref.h"
9
10 /*
11
12 XXXXXXXXXXXX XXXX XXXX XXXXXXXXXXXX XXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXX
13 XXXXXXXXXXXX XXXX XXXX XXXXXXXXXXXX XXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXX
14 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
15 XXXX XXXXXXXXXXXX XXXXXXXXXXXX XXXXXXXXXXXX XXXX XXXX XXXX
16 XXXX XXXXXXXXXXXX XXXXXXXXXXXX XXXXXXXXXXXX XXXX XXXX XXXX
17 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
18 XXXXXXXXXXXX XXXX XXXX XXXX XXXX XXXXXXXXXXXX XXXX XXXX XXXX
19 XXXXXXXXXXXX XXXX XXXX XXXX XXXX XXXXXXXXXXXX XXXX XXXX XXXX
20
21                                     VERSION 1.0
22
23             Dedicated to the amazingly sweet Melanie Ridgway. <3
24
25                                     Love,
26                                     Matt
27
28             Copyright 2010 Matthew Mikolay
29
30             This file is part of chasm.
31
32             chasm is free software: you can redistribute it and/or modify
33             it under the terms of the GNU General Public License as published by
34             the Free Software Foundation, either version 3 of the License, or
35             (at your option) any later version.
36
37             chasm is distributed in the hope that it will be useful,
38             but WITHOUT ANY WARRANTY; without even the implied warranty of
39             MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
40             GNU General Public License for more details.
41
42             You should have received a copy of the GNU General Public License
43             along with chasm. If not, see <http://www.gnu.org/licenses/>.
44
45 */
46
47 int main(int argc, char *argv[])
48 {
49     // Create needed variables
50     unsigned short start          = 0x200;           // Starting value of the program counter
51     unsigned short offset        = 0x000;           // Program counter offset from the starting
address, initially zero
52     unsigned char memory[4096];           // Memory to store CHIP-8 machine code
53     string i_file = "input.asm";          // The filename of the input file
54     string o_file = "output.c8";          // The filename of the output file
55     bool eflag = false;                   // Signal if an error has occurred
56
57     // *****
58     // Begin processing command line arguments
59     // *****
60
61     // Syntax: chasm <input filename> <output filename>
62
63     // Make sure the correct number of command line arguments is present

```

```

64  if(argc != 3)
65  {
66      // Return an error message
67      eflag = error("chasm 1.0", 0, "Incorrect command line argument
usage.\n\nSyntax:\n\tchasm <input filename> <output filename>");
68      // Exit the program
69      return false;
70  }
71
72  // Set the input and output filenames
73  i_file = argv[1];
74  o_file = argv[2];
75
76  // *****
77  // Begin reading data from the input file
78  // *****
79
80  // Open the source file for reading
81  ifstream source;
82  source.open(i_file.c_str(), ifstream::in);
83
84  // Check if the source file has not been opened correctly
85  if(!source.is_open())
86  {
87      // Return an error message
88      eflag = error(i_file, 0, "File error. Could not open file \"" + i_file + "\" for
input.");
89      // Exit the program
90      return false;
91  }
92
93  // Create arrays and variables to store label data and references
94  const unsigned int list_labels_size = 256; // The maximum number of labels capable of
being processed by chasm
95  const unsigned int list_lrefs_size = 256; // The maximum number of label references
capable of being processed by chasm
96
97  label list_labels[list_labels_size]; // An array to store labels
98  lref list_lrefs [list_lrefs_size]; // An array to store label references
99
100 unsigned char num_labels = 0; // The current number of labels stored in
list_labels
101 unsigned char num_lrefs = 0; // The current number of label references
stored in list_lrefs
102
103 // Create a string to store line contents
104 string line = "";
105
106 // Create a string to store the current word of a line
107 string word;
108
109 // Create a string array to store the arguments
110 // argument[0] is the command name (I.E. "SE", "DRW", "LD", etc.)
111 // argument[1 ... 3] are command parameters (I.E. "V1", "#4A", etc.)
112 // Note: The current version of CHIP-8 assembly processed by chasm
113 // supports a maximum of four arguments, as no mnemonics
114 // process more than this number.
115 string arguments[4];
116
117 // Create an integer to store the number of arguments currently stored in the array
118 unsigned char numArgs;
119
120 // Create an integer to store the line number of the line being currently read
121 unsigned int lineNumber = 0;
122
123 // *****
124 // Begin processing the current line into arguments
125 // *****
126
127 // Loop through the lines of the source file
128 while(!source.eof())

```

```

129 {
130     // Get the next line from the source file
131     getline(source, line);
132
133     // Reset the value of the current word string and the number of arguments
134     word = "";
135     numArgs = 0;
136
137     // Increase the line number
138     lineNumber++;
139
140     // Loop through the line, storing arguments and looking for labels
141     for(unsigned char i = 0; i < line.length(); i++)
142     {
143         // If the current character is a semicolon, process the rest of the line as
a comment
144         if(line.at(i) == ';')
145         {
146             // If word has data in it, store it
147             if(word.length() != 0)
148             {
149                 if(!addArg(word, arguments, numArgs))
150                 {
151                     // Return an error message
152                     eflag = error(i_file, lineNumber, "Syntax error.
Argument array overflow. Check line?");
153                     // Exit the program
154                     return false;
155                 }
156             }
157
158             // Stop processing the rest of the line
159             break;
160         }
161
162         // If the current character is a colon, process the stored word as a label
163         if(line.at(i) == ':')
164         {
165             // Make sure the colon was placed correctly
166             if(word == "")
167             {
168                 // Incorrect colon usage
169                 eflag = error(i_file, lineNumber, "Syntax error. Check colon
usage.");
170                 // Read the next line
171                 break;
172             }
173
174             // Try to store the word
175             if(!addArg(word, arguments, numArgs))
176             {
177                 // Return an error message
178                 eflag = error(i_file, lineNumber, "Syntax error. Argument
array overflow.");
179                 // Exit the program
180                 return false;
181             }
182
183             // Make sure only one word has been stored
184             if(numArgs != 1)
185             {
186                 // Return an error message
187                 eflag = error(i_file, lineNumber, "Syntax error. Check label
identifier.");
188                 // Exit the program
189                 return false;
190             }
191
192             // Create a new label
193             label myLabel = label(arguments[0], offset);
194

```



```

195                                     // If the label has an invalid name, return an error
196 if(!myLabel.isValid())
197 {
198     // Return an error message
199     eflag = error(i_file, lineNumber, "Label error. Invalid
label identifier.");
200     // Exit the program
201     return false;
202 }
203
204                                     // Make sure a label with the same name has not already been
inserted into the array
205 for(unsigned int curLblNum = 0; curLblNum < num_labels; curLblNum+
+)
206 {
207     if(myLabel.getName() == list_labels[curLblNum].getName())
208     {
209         // Return an error message
210         eflag = error(i_file, lineNumber, "Syntax error.
Label \"\" + myLabel.getName() + \"\" already exists.");
211         // Exit the program
212         return false;
213     }
214 }
215
216                                     // Make sure no array overflow will occur
217 if(num_labels >= list_labels_size)
218 {
219     // Return an error message
220     eflag = error(i_file, lineNumber, "Assembler error. Label
array overflow. Maximum number of labels reached.");
221     // Exit the program
222     return false;
223 }
224
225                                     // Add the current label to the list for storage
226 list_labels[num_labels] = myLabel;
227 num_labels++;
228
229                                     // ++++++
230                                     // Enable the following line to signal the creation of a new label
231                                     // ++++++
232                                     // cout << "Label \"\" << arguments[0] << \"\" created at address "
<< offset << "!" << endl;
233
234                                     // Reset the value of word
235 word = "";
236
237                                     // Process the rest of the line as if the label has been removed
238 numArgs = 0;
239
240 continue;
241 }
242
243                                     // If the current character is a comma, and parameters are currently being
read, store the word
244 if(line.at(i) == ',' && word != "")
245 {
246     // Detect if the comma was unnecessary, as no other arguments have
been stored yet
247     if(numArgs == 0)
248     {
249         // Return an error message
250         eflag = error(i_file, lineNumber, "Syntax error. Unnecessary
comma?");
251         // Exit the program
252         return false;
253     }
254
255     // Store the current word
256     if(!addArg(word, arguments, numArgs))

```

```

257         {
258             // Return an error message
259             eflag = error(i_file, lineNumber, "Syntax error. Argument
array overflow.");
260             // Exit the program
261             return false;
262         }
263     }
264
265     // If the current character is not whitespace, add the uppercase version of
the current character to the word
266     else if(!isWhitespace(line.at(i)))
267         word += toUpper(line.at(i));
268
269     // If whitespace is encountered, word is not empty, and no arguments
270     // Are currently stored, then store the first word as a command
271     else if(word.length() != 0 && numArgs == 0 && isWhitespace(line.at(i)))
272     {
273         if(!addArg(word, arguments, numArgs))
274         {
275             // Return an error message
276             eflag = error(i_file, lineNumber, "Syntax error. Argument
array overflow. Check line?");
277             // Exit the program
278             return false;
279         }
280         continue;
281     }
282
283     // If this is the end of the line, and word contains data, store it
depending upon if we are storing commands or parameters
284     if(i == line.length() - 1)
285     {
286         // If the final character is a comma, return an error
287         if(line.at(i) == ',')
288         {
289             // Return an error message
290             eflag = error(i_file, lineNumber, "Syntax error. Unnecessary
comma?");
291             // Exit the program
292             return false;
293         }
294
295         if(word != "")
296         {
297             // Store the current word
298             if(!addArg(word, arguments, numArgs))
299             {
300                 // Return an error message
301                 eflag = error(i_file, lineNumber, "Syntax error.
Argument array overflow. Check line?");
302                 // Exit the program
303                 return false;
304             }
305         }
306     }
307 }
308
309 // ++++++
310 // Enable the following lines of code to output all arguments to the screen
311 // ++++++
312
313 // cout << endl;
314 // for(unsigned char a = 0; a < numArgs; a++)
315 //     cout << arguments[a] << "\t";
316
317
318 // *****
319 // Begin processing the arguments and generating machine code
320 // *****
321

```

```

322 // Check if at least one argument was entered
323 if(numArgs > 0)
324 {
325     // Process all single argument commands
326     if(numArgs == 1)
327     {
328         // CLS - 00E0
329         if(arguments[0] == "CLS")
330         {
331             // Add corresponding machine code to memory
332             memory[offset] = 0x00; offset++;
333             memory[offset] = 0xE0; offset++;
334             // Read the next line
335             continue;
336         }
337
338         // RET - 00EE
339         if(arguments[0] == "RET")
340         {
341             // Add corresponding machine code to memory
342             memory[offset] = 0x00; offset++;
343             memory[offset] = 0xEE; offset++;
344             // Read the next line
345             continue;
346         }
347
348         // Unknown command
349         eflag = error(i_file, lineNumber, "Syntax error. Unknown command
\"\" + arguments[0] + "\".");
350         // Read the next line
351         continue;
352     }
353
354     // Process all double argument commands
355     if(numArgs == 2)
356     {
357         // Macro: .START ADDR
358         // Signifies start address
359         if(arguments[0] == ".START")
360         {
361             // Make sure no other commands have been entered yet
362             if(offset != 0)
363             {
364                 // Return an error message
365                 eflag = error(i_file, lineNumber, "Assembler
error. Input code attempts to modify start address after other commands have been processed.");
366                 // Exit the program
367                 return false;
368             }
369
370             unsigned short result = 0x0000;
371             // Try to convert the second argument into an address
372             if(!strTo12Bit(arguments[1], result))
373             {
374                 // Return an error message
375                 eflag = error(i_file, lineNumber, "Syntax
error. Invalid number \"\" + arguments[1] + "\" designated as start address.");
376                 // Exit the program
377                 return false;
378             }
379
380             // Set the start address
381             start = result;
382
383             // Read the next line
384             continue;
385         }
386
387         // DB BYTE
388         // Store a byte
389         if(arguments[0] == "DB")

```

```

390     {
391         unsigned char result = 0x00;
392         // Try to convert the second argument into a byte
393         if(!strToByte(arguments[1], result))
394         {
395             // Invalid argument
396             eflag = error(i_file, lineNumber, "Syntax error.
Invalid argument \"" + arguments[2] + "\". Expected an 8-bit number.");
397             // Read the next line
398             continue;
399         }
400
401         memory[offset] = result;
402         offset++;
403
404         // Read the next line
405         continue;
406     }
407
408     // DW WORD
409     // Store a word
410     if(arguments[0] == "DW")
411     {
412         unsigned short result = 0x0000;
413         // Try to convert the second argument into a byte
414         if(!strToWord(arguments[1], result))
415         {
416             // Invalid argument
417             eflag = error(i_file, lineNumber, "Syntax error.
Invalid argument \"" + arguments[2] + "\". Expected a 16-bit number.");
418             // Read the next line
419             continue;
420         }
421
422         memory[offset] = (unsigned char)(result >> 8);
423         offset++;
424         memory[offset] = (unsigned char)(result & 0x00FF); offset++;
425
426         // Read the next line
427         continue;
428     }
429
430     // SYS ADDR - ONNN
431     if(arguments[0] == "SYS")
432     {
433         unsigned short result = 0x0000;
434         // Try to convert the second argument into an address
435         // If that fails, process the second argument as a label
436         if(!strTo2Bit(arguments[1], result))
437         {
438             // Create a new label reference
439             lref myLref = lref(arguments[1], offset);
440
441             // Make sure no array overflow will occur
442             if(num_lrefs >= list_lrefs_size)
443             {
444                 // Return an error message
445                 eflag = error(i_file, lineNumber, "Assembler
error. Label reference array overflow. Maximum number of label references reached.");
446                 // Exit the program
447                 return false;
448             }
449
450             // Add the current label to the list for storage
451             list_lrefs[num_lrefs] = myLref;
452             num_lrefs++;
453         }
454
455         // Add corresponding machine code to memory

```

```

455         memory[offset] = 0x00 + ((result & 0x0F00) >> 8);    offset+
+;
456         memory[offset] = 0x00 + (result & 0x00FF);            offset+
+;
457         // Read the next line
458         continue;
459     }
460
461     // JP ADDR - 1NNN
462     if(arguments[0] == "JP")
463     {
464         unsigned short result = 0x0000;
465         // Try to convert the second argument into an address
466         // If that fails, process the second argument as a label
reference
467         if(!strTo12Bit(arguments[1], result))
468         {
469             // Create a new label reference
470             lref myLref = lref(arguments[1], offset);
471
472             // Make sure no array overflow will occur
473             if(num_lrefs >= list_lrefs_size)
474             {
475                 // Return an error message
476                 eflag = error(i_file, lineNumber, "Assembler
error. Label reference array overflow. Maximum number of label references reached.");
477                 // Exit the program
478                 return false;
479             }
480
481             // Add the current label to the list for storage
482             list_lrefs[num_lrefs] = myLref;
483             num_lrefs++;
484         }
485
486         // Add corresponding machine code to memory
487         memory[offset] = 0x10 + ((result & 0x0F00) >> 8);    offset+
+;
488         memory[offset] = 0x00 + (result & 0x00FF);            offset+
+;
489         // Read the next line
490         continue;
491     }
492
493     // CALL ADDR - 2NNN
494     if(arguments[0] == "CALL")
495     {
496         unsigned short result = 0x0000;
497         // Try to convert the second argument into an address
498         // If that fails, process the second argument as a label
reference
499         if(!strTo12Bit(arguments[1], result))
500         {
501             // Create a new label reference
502             lref myLref = lref(arguments[1], offset);
503
504             // Make sure no array overflow will occur
505             if(num_lrefs >= list_lrefs_size)
506             {
507                 // Return an error message
508                 eflag = error(i_file, lineNumber, "Assembler
error. Label reference array overflow. Maximum number of label references reached.");
509                 // Exit the program
510                 return false;
511             }
512
513             // Add the current label to the list for storage
514             list_lrefs[num_lrefs] = myLref;
515             num_lrefs++;
516         }
517

```

```

518 // Add corresponding machine code to memory
519 memory[offset] = 0x20 + ((result & 0x0F00) >> 8); offset+
+;
520 memory[offset] = 0x00 + (result & 0x00FF); offset+
+;
521 // Read the next line
522 continue;
523 }
524
525 // SKP VX - EX9E
526 if(arguments[0] == "SKP")
527 {
528     unsigned char byte = 0x00;
529     // Try to convert the second argument into a register
530     if(!strToRegister(arguments[1], byte))
531     {
532         // Unknown register
533         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
534         // Read the next line
535         continue;
536     }
537
538     // Add corresponding machine code to memory
539     memory[offset] = 0xE0 + byte; offset++;
540     memory[offset] = 0x9E; offset++;
541     // Read the next line
542     continue;
543 }
544
545 // SKNP VX - EXA1
546 if(arguments[0] == "SKNP")
547 {
548     unsigned char byte = 0x00;
549     // Try to convert the second argument into a register
550     if(!strToRegister(arguments[1], byte))
551     {
552         // Unknown register
553         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
554         // Read the next line
555         continue;
556     }
557
558     // Add corresponding machine code to memory
559     memory[offset] = 0xE0 + byte; offset++;
560     memory[offset] = 0xA1; offset++;
561     // Read the next line
562     continue;
563 }
564
565 // SHR VX - 8XY6
566 if(arguments[0] == "SHR")
567 {
568     unsigned char byte = 0x00;
569     // Try to convert the second argument into a register
570     if(!strToRegister(arguments[1], byte))
571     {
572         // Unknown register
573         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
574         // Read the next line
575         continue;
576     }
577
578     // Add corresponding machine code to memory
579     memory[offset] = 0x80 + byte; offset++;
580     memory[offset] = 0x06 + (byte << 4); offset++;
581     // Read the next line
582     continue;
583 }

```

```

584
585 // SHL VX - 8XYE
586 if(arguments[0] == "SHL")
587 {
588     unsigned char byte = 0x00;
589     // Try to convert the second argument into a register
590     if(!strToRegister(arguments[1], byte))
591     {
592         // Unknown register
593         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
594         // Read the next line
595         continue;
596     }
597
598     // Add corresponding machine code to memory
599     memory[offset] = 0x80 + byte;    offset++;
600     memory[offset] = 0x0E + (byte << 4); offset++;
601     // Read the next line
602     continue;
603 }
604
605 // Unknown command
606 eflag = error(i_file, lineNumber, "Syntax error. Unknown command
\"" + arguments[0] + " " + arguments[1] + "\".");
607 // Read the next line
608 continue;
609 }
610
611 // Process all triple argument commands
612 if(numArgs == 3)
613 {
614     // SE VX, BYTE - 3XKK
615     // SE VX, VY - 5XY0
616     if(arguments[0] == "SE")
617     {
618         unsigned char byte1 = 0x00;
619         // Try to convert the second argument into a register
620         if(!strToRegister(arguments[1], byte1))
621         {
622             // Unknown register
623             eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
624             // Read the next line
625             continue;
626         }
627
628         unsigned char byte2 = 0x00;
629         // Try to convert the third argument into a byte
630         if(strToByte(arguments[2], byte2))
631         {
632             byte1 += 0x30;
633         }
634         // Try to convert the third argument into a register
635         else if(strToRegister(arguments[2], byte2))
636         {
637             byte1 += 0x50;
638             byte2 = byte2 << 4;
639         }
640         else
641         {
642             // Invalid argument
643             eflag = error(i_file, lineNumber, "Syntax error.
Invalid argument \"" + arguments[2] + "\".");
644             // Read the next line
645             continue;
646         }
647
648         // Add corresponding machine code to memory
649         memory[offset] = byte1;    offset++;
650         memory[offset] = byte2;    offset++;

```

```

651                                     // Read the next line
652                                     continue;
653                                 }
654
655                                 // SNE VX, BYTE          - 4XKK
656                                 // SNE VX, VY - 9XY0
657                                 if(arguments[0] == "SNE")
658                                 {
659                                     unsigned char byte1 = 0x00;
660                                     // Try to convert the second argument into a register
661                                     if(!strToRegister(arguments[1], byte1))
662                                     {
663                                         // Unknown register
664                                         eflag = error(i_file, lineNumber, "Syntax error.");
Unknown register \" + arguments[1] + "\".";
665                                         // Read the next line
666                                         continue;
667                                     }
668
669                                     unsigned char byte2 = 0x00;
670                                     // Try to convert the third argument into a byte
671                                     if(strToByte(arguments[2], byte2))
672                                     {
673                                         byte1 += 0x40;
674                                     }
675                                     // Try to convert the third argument into a register
676                                     else if(strToRegister(arguments[2], byte2))
677                                     {
678                                         byte1 += 0x90;
679                                         byte2 = byte2 << 4;
680                                     }
681                                     else
682                                     {
683                                         // Invalid argument
684                                         eflag = error(i_file, lineNumber, "Syntax error.");
Invalid argument \" + arguments[2] + "\".";
685                                         // Read the next line
686                                         continue;
687                                     }
688
689                                     // Add corresponding machine code to memory
690                                     memory[offset] = byte1;      offset++;
691                                     memory[offset] = byte2;      offset++;
692                                     // Read the next line
693                                     continue;
694                                 }
695
696                                 // ADD VX, BYTE          - 7XKK
697                                 // ADD VX, VY - 8XY4
698                                 if(arguments[0] == "ADD" && arguments[1] != "I")
699                                 {
700                                     unsigned char byte1 = 0x00;
701                                     // Try to convert the second argument into a register
702                                     if(!strToRegister(arguments[1], byte1))
703                                     {
704                                         // Unknown register
705                                         eflag = error(i_file, lineNumber, "Syntax error.");
Unknown register \" + arguments[1] + "\".";
706                                         // Read the next line
707                                         continue;
708                                     }
709
710                                     unsigned char byte2 = 0x00;
711                                     // Try to convert the third argument into a byte
712                                     if(strToByte(arguments[2], byte2))
713                                     {
714                                         byte1 += 0x70;
715                                     }
716                                     // Try to convert the third argument into a register
717                                     else if(strToRegister(arguments[2], byte2))
718                                     {

```



```

719         byte1 += 0x80;
720         byte2 = (byte2 << 4) + 0x04;
721     }
722     else
723     {
724         // Invalid argument
725         eflag = error(i_file, lineNumber, "Syntax error.
Invalid argument \"" + arguments[2] + "\".");
726         // Read the next line
727         continue;
728     }
729
730     // Add corresponding machine code to memory
731     memory[offset] = byte1;      offset++;
732     memory[offset] = byte2;      offset++;
733     // Read the next line
734     continue;
735 }
736
737 // ADD I, VX - FX1E
738 if(arguments[0] == "ADD" && arguments[1] == "I")
739 {
740     unsigned char byte = 0x00;
741     // Try to convert the third argument into a register
742     if(!strToRegister(arguments[2], byte))
743     {
744         // Unknown register
745         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[2] + "\".");
746         // Read the next line
747         continue;
748     }
749
750     // Add corresponding machine code to memory
751     memory[offset] = 0xF0 + byte; offset++;
752     memory[offset] = 0x1E;      offset++;
753     // Read the next line
754     continue;
755 }
756
757 // RND VX, BYTE - CXKK
758 if(arguments[0] == "RND")
759 {
760     unsigned char byte1 = 0x00;
761     // Try to convert the second argument into a register
762     if(!strToRegister(arguments[1], byte1))
763     {
764         // Unknown register
765         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
766         // Read the next line
767         continue;
768     }
769
770     unsigned char byte2 = 0x00;
771     // Try to convert the third argument into a byte
772     if(!strToByte(arguments[2], byte2))
773     {
774         // Invalid argument
775         eflag = error(i_file, lineNumber, "Syntax error.
Invalid argument \"" + arguments[2] + "\". Expected an 8-bit number.");
776         // Read the next line
777         continue;
778     }
779
780     // Add corresponding machine code to memory
781     memory[offset] = 0xC0 + byte1;      offset++;
782     memory[offset] = byte2;            offset++;
783     // Read the next line
784     continue;
785 }

```

```

786
787 // OR VX, VY - 8XY1
788 // AND VX, VY - 8XY2
789 // XOR VX, VY - 8XY3
790 if(arguments[0] == "OR" || arguments[0] == "AND" || arguments[0] ==
"XOR")
791 {
792     unsigned char byte1 = 0x00;
793     // Try to convert the second argument into a register
794     if(!strToRegister(arguments[1], byte1))
795     {
796         // Unknown register
797         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
798         // Read the next line
799         continue;
800     }
801
802     unsigned char byte2 = 0x00;
803     // Try to convert the third argument into a register
804     if(!strToRegister(arguments[2], byte2))
805     {
806         // Invalid argument
807         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[2] + "\".");
808         // Read the next line
809         continue;
810     }
811
812     // Shift byte2 left by four bits
813     byte2 = byte2 << 4;
814
815     // Determine what number to end the command with
816     if(arguments[0] == "OR")
817         byte2 += 0x01;
818     if(arguments[0] == "AND")
819         byte2 += 0x02;
820     if(arguments[0] == "XOR")
821         byte2 += 0x03;
822
823     // Add corresponding machine code to memory
824     memory[offset] = 0x80 + byte1;    offset++;
825     memory[offset] = byte2;          offset++;
826     // Read the next line
827     continue;
828 }
829
830 // SUB VX, VY - 8XY5
831 if(arguments[0] == "SUB")
832 {
833     unsigned char byte1 = 0x00;
834     // Try to convert the second argument into a register
835     if(!strToRegister(arguments[1], byte1))
836     {
837         // Unknown register
838         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
839         // Read the next line
840         continue;
841     }
842
843     unsigned char byte2 = 0x00;
844     // Try to convert the third argument into a register
845     if(!strToRegister(arguments[2], byte2))
846     {
847         // Invalid argument
848         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[2] + "\".");
849         // Read the next line
850         continue;
851     }

```

```

852
853 // Shift byte2 left by four bits
854 byte2 = byte2 << 4;
855
856 // Add corresponding machine code to memory
857 memory[offset] = 0x80 + byte1;      offset++;
858 memory[offset] = byte2 + 0x05;      offset++;
859 // Read the next line
860 continue;
861 }
862
863 // SUBN VX, VY - 8XY7
864 if(arguments[0] == "SUBN")
865 {
866     unsigned char byte1 = 0x00;
867     // Try to convert the second argument into a register
868     if(!strToRegister(arguments[1], byte1))
869     {
870         // Unknown register
871         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
872         // Read the next line
873         continue;
874     }
875
876     unsigned char byte2 = 0x00;
877     // Try to convert the third argument into a register
878     if(!strToRegister(arguments[2], byte2))
879     {
880         // Invalid argument
881         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[2] + "\".");
882         // Read the next line
883         continue;
884     }
885
886     // Shift byte2 left by four bits
887     byte2 = byte2 << 4;
888
889     // Add corresponding machine code to memory
890     memory[offset] = 0x80 + byte1;      offset++;
891     memory[offset] = byte2 + 0x07;      offset++;
892     // Read the next line
893     continue;
894 }
895
896 // SHR VX, VY - 8XY6
897 if(arguments[0] == "SHR")
898 {
899     unsigned char byte1 = 0x00;
900     // Try to convert the second argument into a register
901     if(!strToRegister(arguments[1], byte1))
902     {
903         // Unknown register
904         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
905         // Read the next line
906         continue;
907     }
908
909     unsigned char byte2 = 0x00;
910     // Try to convert the third argument into a register
911     if(!strToRegister(arguments[2], byte2))
912     {
913         // Invalid argument
914         eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[2] + "\".");
915         // Read the next line
916         continue;
917     }
918

```

```

919          // Shift byte2 left by four bits
920          byte2 = byte2 << 4;
921
922          // Add corresponding machine code to memory
923          memory[offset] = 0x80 + byte1;      offset++;
924          memory[offset] = byte2 + 0x06;      offset++;
925          // Read the next line
926          continue;
927      }
928
929      // SHL VX, VY - 8XYE
930      if(arguments[0] == "SHL")
931      {
932          unsigned char byte1 = 0x00;
933          // Try to convert the second argument into a register
934          if(!strToRegister(arguments[1], byte1))
935          {
936              // Unknown register
937              eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
938              // Read the next line
939              continue;
940          }
941
942          unsigned char byte2 = 0x00;
943          // Try to convert the third argument into a register
944          if(!strToRegister(arguments[2], byte2))
945          {
946              // Invalid argument
947              eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[2] + "\".");
948              // Read the next line
949              continue;
950          }
951
952          // Shift byte2 left by four bits
953          byte2 = byte2 << 4;
954
955          // Add corresponding machine code to memory
956          memory[offset] = 0x80 + byte1;      offset++;
957          memory[offset] = byte2 + 0x0E;      offset++;
958          // Read the next line
959          continue;
960      }
961
962      // JP V0, ADDR - BNNN
963      if(arguments[0] == "JP")
964      {
965          // Make sure the second argument is the correct register
966          if(arguments[1] != "V0")
967          {
968              // Unknown register
969              eflag = error(i_file, lineNumber, "Syntax error.
Invalid register \"" + arguments[1] + "\". Expected V0.");
970              // Read the next line
971              continue;
972          }
973
974          unsigned short result = 0x0000;
975          // Try to convert the third argument into an address
976          // If that fails, process the third argument as a label
reference
977          if(!strTo12Bit(arguments[2], result))
978          {
979              // Create a new label reference
980              lref myLref = lref(arguments[2], offset);
981
982              // Make sure no array overflow will occur
983              if(num_lrefs >= list_lrefs_size)
984              {
985                  // Return an error message

```

```

986         eflag = error(i_file, lineNumber, "Assembler
error. Label reference array overflow. Maximum number of label references reached.");
987         // Exit the program
988         return false;
989     }
990
991     // Add the current label to the list for storage
992     list_lrefs[num_lrefs] = myLref;
993     num_lrefs++;
994 }
995
996 // Add corresponding machine code to memory
997 memory[offset] = 0xB0 + ((result & 0x0F00) >> 8);    offset+
+;
998 memory[offset] = 0x00 + (result & 0x00FF);            offset+
+;
999 // Read the next line
1000 continue;
1001 }
1002
1003 // LD VX, BYTE      - 6XKK
1004 // LD I,  ADDR      - ANNN
1005 // LD VX, DT  - FX07
1006 // LD VX, K   - FX0A
1007 // LD VX, [I] - FX65
1008 // LD VX, VY  - 8XY0
1009 // LD DT, VX  - FX15
1010 // LD ST, VX  - FX18
1011 // LD F,  VX  - FX29
1012 // LD B,  VX  - FX33
1013 // LD [I], VX - FX55
1014 if(arguments[0] == "LD")
1015 {
1016     unsigned char byte1 = 0x00;
1017     unsigned char byte2 = 0x00;
1018
1019     // Try to convert the third argument into a register
1020     if(strToRegister(arguments[2], byte2))
1021     {
1022         // Try to convert the second argument into a register
1023         if(strToRegister(arguments[1], byte1))
1024         {
1025             byte1 += 0x80;
1026
1027             // Shift byte 2 left by four bits
1028             byte2 = byte2 << 4;
1029         }
1030         else if(arguments[1] == "DT")
1031         {
1032             byte1 += 0xF0 + byte2;
1033             byte2 = 0x15;
1034         }
1035         else if(arguments[1] == "ST")
1036         {
1037             byte1 += 0xF0 + byte2;
1038             byte2 = 0x18;
1039         }
1040         else if(arguments[1] == "F")
1041         {
1042             byte1 += 0xF0 + byte2;
1043             byte2 = 0x29;
1044         }
1045         else if(arguments[1] == "B")
1046         {
1047             byte1 += 0xF0 + byte2;
1048             byte2 = 0x33;
1049         }
1050         else if(arguments[1] == "[I]")
1051         {
1052             byte1 += 0xF0 + byte2;
1053             byte2 = 0x55;

```

```

1054         }
1055         else
1056         {
1057             // Invalid argument
1058             eflag = error(i_file, lineNumber, "Syntax
error. Invalid argument \"" + arguments[1] + "\".");
1059             // Read the next line
1060             continue;
1061         }
1062     }
1063
1064     // Try to convert the second argument into a register
1065     else if(strToRegister(arguments[1], byte1))
1066     {
1067         unsigned short result = 0x0000;
1068
1069         // Try to convert the third argument into a byte
1070         if(strToByte(arguments[2], byte2))
1071         {
1072             byte1 += 0x60;
1073         }
1074         else if(arguments[2] == "DT")
1075         {
1076             byte1 += 0xF0;
1077             byte2 = 0x07;
1078         }
1079         else if(arguments[2] == "K")
1080         {
1081             byte1 += 0xF0;
1082             byte2 = 0x0A;
1083         }
1084         else if(arguments[2] == "[I]")
1085         {
1086             byte1 += 0xF0;
1087             byte2 = 0x65;
1088         }
1089         else
1090         {
1091             // Invalid argument
1092             eflag = error(i_file, lineNumber, "Syntax
error. Invalid argument \"" + arguments[2] + "\".");
1093             // Read the next line
1094             continue;
1095         }
1096     }
1097
1098     else if(arguments[1] == "I")
1099     {
1100         unsigned short result = 0x0000;
1101         // Try to convert the third argument into an address
1102         // If that fails, process the third argument as a
label reference
1103
1104         if(!strTo2Bit(arguments[2], result))
1105         {
1106             // Create a new label reference
1107             lref myLref = lref(arguments[2], offset);
1108
1109             // Make sure no array overflow will occur
1110             if(num_lrefs >= list_lrefs_size)
1111             {
1112                 // Return an error message
1113                 eflag = error(i_file, lineNumber,
"Assembler error. Label reference array overflow. Maximum number of label references reached.");
1114                 // Exit the program
1115                 return false;
1116             }
1117             // Add the current label to the list for
storage
1118             list_lrefs[num_lrefs] = myLref;
1119             num_lrefs++;

```

```

1120         }
1121
1122         byte1 = 0xA0 + ((result & 0x0F00) >> 8);
1123         byte2 = 0x00 + (result & 0x00FF);
1124     }
1125
1126     // Unknown command
1127     else
1128     {
1129         eflag = error(i_file, lineNumber, "Syntax error.
Unknown command \"" + arguments[0] + " " + arguments[1] + ", " + arguments[2] + "\".");
1130     }
1131
1132     // Add corresponding machine code to memory
1133     memory[offset] = byte1;    offset++;
1134     memory[offset] = byte2;    offset++;
1135     // Read the next line
1136     continue;
1137 }
1138
1139 // Unknown command
1140 eflag = error(i_file, lineNumber, "Syntax error. Unknown command
\"" + arguments[0] + " " + arguments[1] + ", " + arguments[2] + "\".");
1141 // Read the next line
1142 continue;
1143 }
1144
1145 // Process all quadruple argument commands
1146 if(numArgs == 4)
1147 {
1148     // DRW VX, VY, NIBBLE - DXYN
1149     if(arguments[0] == "DRW")
1150     {
1151         unsigned char byte1 = 0x00;
1152         // Try to convert the second argument into a register
1153         if(!strToRegister(arguments[1], byte1))
1154         {
1155             // Unknown register
1156             eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[1] + "\".");
1157             // Read the next line
1158             continue;
1159         }
1160
1161         unsigned char byte2 = 0x00;
1162         // Try to convert the third argument into a register
1163         if(!strToRegister(arguments[2], byte2))
1164         {
1165             // Unknown register
1166             eflag = error(i_file, lineNumber, "Syntax error.
Unknown register \"" + arguments[2] + "\".");
1167             // Read the next line
1168             continue;
1169         }
1170
1171         unsigned char nibble = 0x00;
1172         // Try to convert the third argument into a nibble
1173         if(!strToNibble(arguments[3], nibble))
1174         {
1175             // Invalid argument
1176             eflag = error(i_file, lineNumber, "Syntax error.
Invalid argument \"" + arguments[3] + "\". Expected a 4-bit number.");
1177             // Read the next line
1178             continue;
1179         }
1180
1181         // Add corresponding machine code to memory
1182         memory[offset] = 0xD0 + byte1;    offset++;
1183         memory[offset] = (byte2 << 4) + nibble;    offset++;
1184         // Read the next line
1185         continue;

```

```

1186         }
1187
1188         // Unknown command
1189         eflag = error(i_file, lineNumber, "Syntax error. Unknown command
\" + arguments[0] + " " + arguments[1] + ", " + arguments[2] + ", " + arguments[3] + "\".");
1190         // Read the next line
1191         continue;
1192     }
1193 }
1194 }
1195
1196 // *****
1197 // Begin linking label references with labels
1198 // *****
1199
1200 // Loop through the array of label references
1201 for(unsigned char i_lref; i_lref < num_lrefs; i_lref++)
1202 {
1203     // The current label reference
1204     lref current_lref = list_lrefs[i_lref];
1205
1206     // Locate the label being referenced by this label reference
1207     bool found = false;
1208     unsigned char i_label;
1209     for(i_label = 0; i_label < num_labels; i_label++)
1210     {
1211         if(current_lref.getName() == list_labels[i_label].getName())
1212         {
1213             found = true;
1214             break;
1215         }
1216     }
1217
1218     // If it is not found, output an error
1219     if(!found)
1220     {
1221         // Output an error
1222         eflag = error(i_file, 0, "Label error. Label \"" + current_lref.getName() +
"\" not found.");
1223         // Exit the program
1224         return false;
1225     }
1226
1227     // The current label
1228     label current_label = list_labels[i_label];
1229
1230     // Jump to the offset address given by the label reference and insert the address
designated by the corresponding label
1231     memory[current_lref.getAddress()] += ((current_label.getAddress() + start) &
0x0F00) >> 8;
1232     memory[current_lref.getAddress() + 1] += ((current_label.getAddress() +
start) & 0x00FF);
1233 }
1234
1235 // Close the source file
1236 source.close();
1237
1238 // *****
1239 // Output all CHIP-8 code to the output file
1240 // *****
1241
1242 // Has an error occurred?
1243 if(!eflag)
1244 {
1245     // Open the output file for writing
1246     ofstream output;
1247     output.open(o_file.c_str(), ofstream::out);
1248
1249     // Check if the output file has not been opened correctly
1250     if(!output.is_open())
1251     {

```



```

1252         // Return an error message
1253         error(o_file, 0, "File error. Could not open file \"" + o_file + "\" for
output.");
1254         // Exit the program
1255         return false;
1256     }
1257     // Output all the generated machine code to the file
1258     for(unsigned int current_byte = 0; current_byte < offset; current_byte++)
1259     {
1260         output << memory[current_byte];
1261     }
1262     // Close the output file
1263     output.close();
1264 }
1265
1266 // *****
1267 // Enable the following lines of code to output
1268 // all generated machine code to the screen****
1269 // *****
1270 /*
1271 cout << "#####" << endl;
1272 cout << "Machine Code Output" << endl;
1273 cout << "#####" << endl;
1274
1275 if(offset > 0) {
1276     for(unsigned int z = 0; z < offset - 1; z += 2)
1277         printf ("%03X \t %02X %02X \n", z + start, memory[z], memory[z+1]);
1278     }
1279 */
1280
1281 return 0;
1282 }

```

## License

Version 1.0 of chasm is licensed under the GNU General Public License Version 3. A copy of this license has been reproduced on the following pages.

### GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

#### Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps:

(1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

#### TERMS AND CONDITIONS

##### 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

#### 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

#### 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is

covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

### 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

### 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

### 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7.
- 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

## 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation

of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

#### 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

#### 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular

copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

#### 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

#### 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

#### 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding

Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

## 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

## 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

## 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional



obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

#### 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

#### 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

### END OF TERMS AND CONDITIONS

#### How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it under certain conditions; type
`show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a

"copyright disclaimer" for the program, if necessary.  
For more information on this, and how to apply and follow the GNU GPL, see  
<<http://www.gnu.org/licenses/>>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read  
<<http://www.gnu.org/philosophy/why-not-lgpl.html>>.