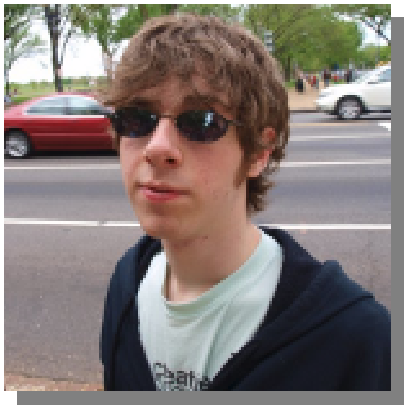
# CHASM

Version 1.0 Manual
By Matthew Mikolay

matt.mik@hotmail.com
mattmik.com

## Table of Contents

## About the Author



*December 24th, 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.

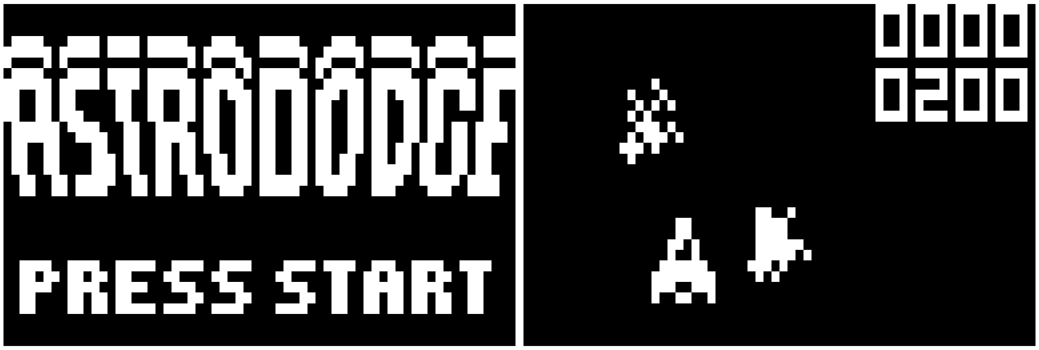
Matthew has a strong passion for vintage computers and their history. He is currently a member of the MidAtlantic Retro Computing Hobbyists (MARCH).

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

## 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

## 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.*
- *Vx and Vy 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>} |
| 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] |
| ---- | **.START <addr>** |
| ---- | **DB <byte>** |
| ---- | **DW <word>** |

## 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

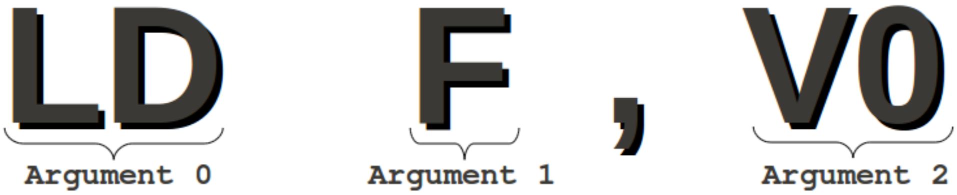
Upon startup, chasm checks to make sure the correct amount of command line arguments were entered and processes them accordingly. chasm is called with the following syntax:

```
    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.

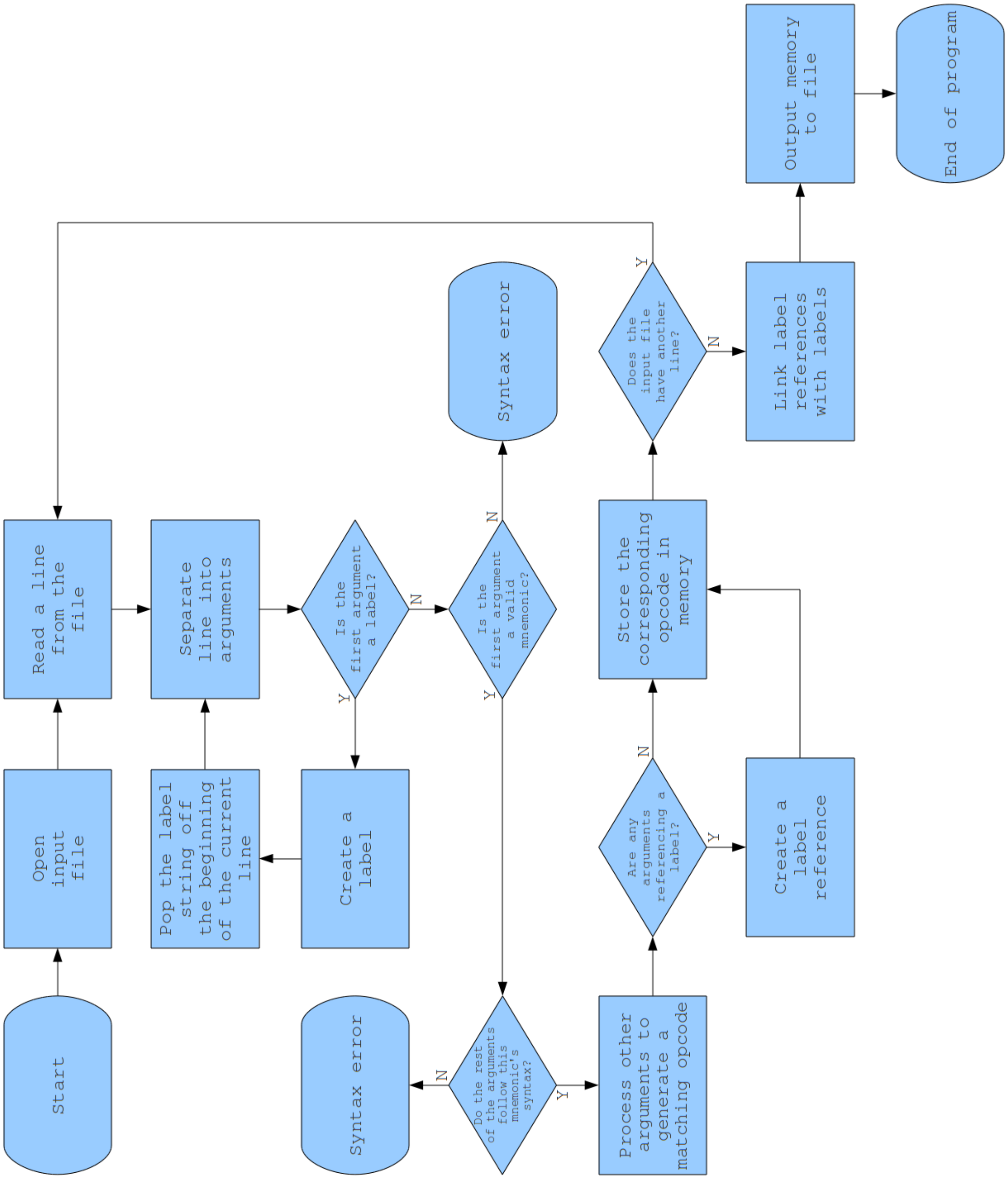
After the line has been separated into arguments and any present labels have been removed, chasm begins processing the arguments and generating the resulting machine code. If an error is found in the syntax of the arguments, or the assembler undergoes an error, relevant information is printed to the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


Start
Open input file
Read a line from the file
Separate line into arguments
Is the first argument a label?
Y
Create a label
Pop the label string off the beginning of the current line
N
Is the first argument a valid mnemonic?
N
Syntax error
Y
Do the rest of the arguments follow this mnemonic's syntax?
N
Syntax error
Y
Process other arguments to generate a matching opcode
Are any arguments referencing a label?
Y
Create a label reference
N
Store the corresponding opcode in memory
Does the input file have another line?
Y
N
Link label references with labels
Output memory to file
End of program

## 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` | ■ The primary header file for chasm<br>■ Contains functions to perform the following:<br>○ Check for whitespace<br>○ Convert a lowercase character to uppercase<br>○ Output an error<br>○ Add a command argument to an array<br>○ Check if a string is numeric<br>○ Convert strings to numeric data and register data | Page 14 |
| `label.h` | ■ The header file defining the label class | Page 19 |
| `lref.h` | ■ The header file defining the label reference class | Page 21 |
| `chasm.cpp` | ■ The primary source code file for chasm | 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

```
#include <string>              // For the string class
#include <sstream>             // For conversions from string to number
using namespace std;

/*
    Copyright 2010 Matthew Mikolay

    This file is part of chasm.

    chasm 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.

    chasm 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 chasm.  If not, see <http://www.gnu.org/licenses/>.

*/

// Returns true if a given character is whitespace
bool isWhitespace(char letter)
{
 return (letter == ' ' || letter == '\t');
}

// Converts a given character to uppercase if the input is in lowercase
char toUpper(char letter)
{
 if(letter >= 0x61 && letter <= 0x7A)
        return (letter - 0x20);
 return letter;
}

// Writes to the screen that an error occured
bool error(string filename, unsigned int lineNumber, string message)
{
 cout << filename << ":" << lineNumber << ":" << message << endl;
 return true;
}

// Try to add an argument to the array of arguments, making sure no overflow occurs
bool addArg(string &input, string (&args)[4], unsigned char &number)
{
 // Make sure the maximum number of arguments is not already in the array
 if(number >= 4)
        return false;

 // Store the word, increment the counter, and clear the contents of word
 args[number] = input;
 number++;
 input = "";
 return true;
}

// Returns true if the given string is a number
bool isNumeric(string input)
{
 if(input.at(0) == '#' || input.at(0) == '$')
        return true;

 // Loop through the contents of the string, looking for non-numeric characters
 for(unsigned int i = 0; i < input.length(); i++)
 {
      if(input.at(i) < '0' || input.at(i) > '9')
           return false;
 }

 return true;
}

// Convert a given string to a register number
bool strToRegister(string input, unsigned char &output)
{
 if(input.length() == 2 && input.at(0) == 'V')
 {
      // Is the register a number register?
      if(input.at(1) >= 'A' && input.at(1) <= 'F')
      {
           output = input.at(1) - 55;
           return true;
      }
      // Is the register a letter register?
      if(input.at(1) >= '0' && input.at(1) <= '9')
      {
           output = input.at(1) - '0';
           return true;
      }
 }
 return false;
}

// Convert a given string to a 4-bit number (nibble)
bool strToNibble(string input, unsigned char &output)
{
 if(input.length() > 0)
 {
      // Process the input as hexadecimal
      if(input.at(0) == '#')
      {
           unsigned int result = 0;
           // Remove the hash
           input = input.substr(1, input.length());
           // Attempt to convert the string to a number
           istringstream iss(input);

           // Make sure the conversion did not fail and the result is a 4-bit number
           if((iss >> hex >> result).fail() || result >= 16)
                return false;

           output = result;
           return true;
      }

      // Process the input as binary
      if(input.at(0) == '$')
      {
           unsigned int result = 0;
           // Loop through the string computing the binary number
           for(unsigned char i = input.length() - 1; i > 0; i--)
           {
                if(input.at(i) == '1')
                     result += 1 << (input.length() - 1 - i);
                else if(input.at(i) != '0' && input.at(i) != '.')
                     return false;
           }

           // Make sure the result is a 4-bit number
           if(result >= 16)
                return false;

           output = result;
           return true;
      }

      // Try to process the input as decimal
      else
      {
           unsigned int result = 0;
           // Attempt to convert the string to a number
           istringstream iss(input);

           // Make sure the conversion did not fail and the result is a 4-bit number
           if((iss >> dec >> result).fail() || result >= 16)
                return false;

           output = result;
           return true;
      }
 }
 return false;
}

// Convert a given string to an 8-bit number (byte)
bool strToByte(string input, unsigned char &output)
{
 if(input.length() > 0)
 {
      // Process the input as hexadecimal
      if(input.at(0) == '#')
      {
           unsigned int result = 0;
           // Remove the hash
           input = input.substr(1, input.length());
           // Attempt to convert the string to a number
           istringstream iss(input);

           // Make sure the conversion did not fail and the result is an 8-bit number
           if((iss >> hex >> result).fail() || result >= 256)
                return false;

           output = result;
           return true;
      }

      // Process the input as binary
      if(input.at(0) == '$')
      {
           unsigned int result = 0;
           // Loop through the string computing the binary number
           for(unsigned char i = input.length() - 1; i > 0; i--)
           {
                if(input.at(i) == '1')
                     result += 1 << (input.length() - 1 - i);
                else if(input.at(i) != '0' && input.at(i) != '.')
                     return false;
           }

           // Make sure the result is an 8-bit number
           if(result >= 256)
                return false;

           output = result;
           return true;
      }

      // Try to process the input as decimal
      else
      {
           unsigned int result = 0;
           // Attempt to convert the string to a number
           istringstream iss(input);

                // Make sure the conversion did not fail and the result is an 8-bit number
                if((iss >> dec >> result).fail() || result >= 256)
                        return false;

                output = result;
                return true;
        }
 }
 return false;
}

// Convert a given string to a 12-bit number
bool strTo12Bit(string input, unsigned short &output)
{
 if(input.length() > 0)
 {
        // Process the input as hexadecimal
        if(input.at(0) == '#')
        {
                unsigned int result = 0;
                // Remove the hash
                input = input.substr(1, input.length());
                // Attempt to convert the string to a number
                istringstream iss(input);

                // Make sure the conversion did not fail and the result is a 12-bit number
                if((iss >> hex >> result).fail() || result >= 4096)
                        return false;

                output = result;
                return true;
        }

        // Process the input as binary
        if(input.at(0) == '$')
        {
                unsigned int result = 0;
                // Loop through the string computing the binary number
                for(unsigned char i = input.length() - 1; i > 0; i--)
                {
                        if(input.at(i) == '1')
                                result += 1 << (input.length() - 1 - i);
                        else if(input.at(i) != '0' && input.at(i) != '.')
                                return false;
                }

                // Make sure the result is a 12-bit number
                if(result >= 4096)
                        return false;

                output = result;
                return true;
        }

        // Try to process the input as decimal
        else
        {
                unsigned int result = 0;
                // Attempt to convert the string to a number
                istringstream iss(input);

                // Make sure the conversion did not fail and the result is a 12-bit number
                if((iss >> dec >> result).fail() || result >= 4096)
                        return false;

                output = result;
                return true;
        }
 }
 return false;
```

```
}

// Convert a given string to a 16-bit number
bool strToWord(string input, unsigned short &output)
{
 if(input.length() > 0)
 {
         // Process the input as hexadecimal
         if(input.at(0) == '#')
         {
                 unsigned int result = 0;
                 // Remove the hash
                 input = input.substr(1, input.length());
                 // Attempt to convert the string to a number
                 istringstream iss(input);

                 // Make sure the conversion did not fail and the result is a 16-bit number
                 if((iss >> hex >> result).fail() || result >= 65536)
                         return false;

                 output = result;
                 return true;
         }

         // Process the input as binary
         if(input.at(0) == '$')
         {
                 unsigned int result = 0;
                 // Loop through the string computing the binary number
                 for(unsigned char i = input.length() - 1; i > 0; i--)
                 {
                         if(input.at(i) == '1')
                                 result += 1 << (input.length() - 1 - i);
                         else if(input.at(i) != '0' && input.at(i) != '.')
                                 return false;
                 }

                 // Make sure the result is a 16-bit number
                 if(result >= 65536)
                         return false;

                 output = result;
                 return true;
         }

         // Try to process the input as decimal
         else
         {
                 unsigned int result = 0;
                 // Attempt to convert the string to a number
                 istringstream iss(input);

                 // Make sure the conversion did not fail and the result is a 16-bit number
                 if((iss >> dec >> result).fail() || result >= 65536)
                         return false;

                 output = result;
                 return true;
         }
 }
 return false;
}
```

## label.h

```
/*
    Copyright 2010 Matthew Mikolay

    This file is part of chasm.

    chasm 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.

    chasm 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 chasm.  If not, see <http://www.gnu.org/licenses/>.

*/

// A class to store label data
class label
{
 private:
         string name;                  // A name to identify the label
         unsigned short address;       // The memory address associated
                                       // with the label name
 public:
         label(string paramName, unsigned short paramAddress);
         label();
         bool isValid();
         string getName();
         unsigned short getAddress();
         ~label();
};

// Label constructor
label::label(string paramName, unsigned short paramAddress)
{
 name    = paramName;
 address = paramAddress;
}

// Label constructor
label::label()
{
}

// Returns true if the label has a valid identifying name
// Labels cannot be numbers of any form, including binary, hexadecimal, and decimal.
// This would interfere with the jump command ("JMP") and prevent
// jumping to specific addresses.
bool label::isValid()
{
 return !isNumeric(name);
}

// Return the label name
string label::getName()
{
 return name;
}

// Return the label address
```

```
unsigned short label::getAddress()
{
 return address;
}

// Label destructor
label::~label(){ }
```

## lref.h

```
/*
    Copyright 2010 Matthew Mikolay

    This file is part of chasm.

    chasm 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.

    chasm 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 chasm.  If not, see <http://www.gnu.org/licenses/>.

*/

// A class to store label reference data
class lref
{
 private:
        string name;                  // The name of the referenced label
        unsigned short address;       // The memory address of
                                      // this reference
 public:
        lref(string paramName, unsigned short paramAddress);
        lref();
        string getName();
        unsigned short getAddress();
        ~lref();
};

// Label reference constructor
lref::lref(string paramName, unsigned short paramAddress)
{
 name    = paramName;
 address= paramAddress;
}

// Label reference constructor
lref::lref()
{
}

// Return the label reference name
string lref::getName()
{
 return name;
}

// Return the label reference address
unsigned short lref::getAddress()
{
 return address;
}

// Label reference destructor
lref::~lref(){ }
```

## chasm.cpp

```cpp
// Standard C++ libraries
#include <fstream>
#include <iostream>

// chasm specific libraries
#include "chasm.h"
#include "label.h"
#include "lref.h"

/*

 XXXXXXXXXXXX  XXXX    XXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX
 XXXXXXXXXXXX  XXXX    XXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX
 XXXX    XXXX  XXXX    XXXX  XXXX    XXXX  XXXX          XXXX    XXXX    XXXX
 XXXX          XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXX    XXXX    XXXX
 XXXX          XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXX    XXXX    XXXX
 XXXX    XXXX  XXXX    XXXX  XXXX    XXXX          XXXX  XXXX    XXXX    XXXX
 XXXXXXXXXXXX  XXXX    XXXX  XXXX    XXXX  XXXXXXXXXXXX  XXXX    XXXX    XXXX
 XXXXXXXXXXXX  XXXX    XXXX  XXXX    XXXX  XXXXXXXXXXXX  XXXX    XXXX    XXXX

                                VERSION 1.0

             Dedicated to the amazingly sweet Melanie Ridgway. <3

                                   Love,
                                     Matt

    Copyright 2010 Matthew Mikolay

    This file is part of chasm.

    chasm 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.

    chasm 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 chasm.  If not, see <http://www.gnu.org/licenses/>.

*/

int main(int argc, char *argv[])
{
 // Create needed variables
 unsigned short start          = 0x200;       // Starting value of the program counter
 unsigned short offset         = 0x000;       // Program counter offset from the starting address, initially zero
 unsigned char  memory[4096];                 // Memory to store CHIP-8 machine code
 string         i_file  = "input.asm";        // The filename of the input file
 string         o_file  = "output.c8";        // The filename of the output file
 bool           eflag   = false;              // Signal if an error has occured

 // ***************************************
 // Begin processing command line arguments
 // ***************************************

 // Syntax: chasm <input filename> <output filename>

 // Make sure the correct number of command line arguments is present
```

```
if(argc != 3)
{
        // Return an error message
        eflag = error("chasm 1.0", 0, "Incorrect command line argument usage.\n\nSyntax:\n\tchasm <input filename> <output filename>");
        // Exit the program
        return false;
}

// Set the input and output filenames
i_file = argv[1];
o_file = argv[2];

// **************************************
// Begin reading data from the input file
// **************************************

// Open the source file for reading
ifstream source;
source.open(i_file.c_str(), ifstream::in);

// Check if the source file has not been opened correctly
if(!source.is_open())
{
        // Return an error message
        eflag = error(i_file, 0, "File error. Could not open file \"" + i_file + "\" for input.");
        // Exit the program
        return false;
}

// Create arrays and variables to store label data and references
const unsigned int list_labels_size = 256;    // The maximum number of labels capable of being processed by chasm
const unsigned int list_lrefs_size  = 256;    // The maximum number of label references capable of being processed by chasm

label list_labels[list_labels_size];          // An array to store labels
lref  list_lrefs [list_lrefs_size];           // An array to store label references

unsigned char  num_labels     = 0;            // The current number of labels stored in list_labels
unsigned char  num_lrefs      = 0;            // The current number of label references stored in list_lrefs

// Create a string to store line contents
string line = "";

// Create a string to store the current word of a line
string word;

// Create a string array to store the arguments
// argument[0] is the command name (I.E. "SE", "DRW", "LD", etc.)
// argument[1 ... 3] are command parameters (I.E. "V1", "#4A", etc.)
// Note:       The current version of CHIP-8 assembly processed by chasm
//             supports a maximum of four arguments, as no mnemonics
//             process more than this number.
string arguments[4];

// Create an integer to store the number of arguments currently stored in the array
unsigned char numArgs;

// Create an integer to store the line number of the line being currently read
unsigned int lineNumber = 0;

// ************************************************
// Begin processing the current line into arguments
// ************************************************

// Loop through the lines of the source file
while(!source.eof())
```

```
{
        // Get the next line from the source file
        getline(source, line);

        // Reset the value of the current word string and the number of arguments
        word    = "";
        numArgs = 0;

        // Increase the line number
        lineNumber++;

        // Loop through the line, storing arguments and looking for labels
        for(unsigned char i = 0; i < line.length(); i++)
        {
                // If the current character is a semicolon, process the rest of the line as a comment
                if(line.at(i) == ';')
                {
                        // If word has data in it, store it
                        if(word.length() != 0)
                        {
                                if(!addArg(word, arguments, numArgs))
                                {
                                        // Return an error message
                                        eflag = error(i_file, lineNumber, "Syntax error. Argument array overflow. Check line?");
                                        // Exit the program
                                        return false;
                                }
                        }

                        // Stop processing the rest of the line
                        break;
                }

                // If the current character is a colon, process the stored word as a label
                if(line.at(i) == ':')
                {
                        // Make sure the colon was placed correctly
                        if(word == "")
                        {
                                // Incorrect colon usage
                                eflag = error(i_file, lineNumber, "Syntax error. Check colon usage.");
                                // Read the next line
                                break;
                        }

                        // Try to store the word
                        if(!addArg(word, arguments, numArgs))
                        {
                                // Return an error message
                                eflag = error(i_file, lineNumber, "Syntax error. Argument array overflow.");
                                // Exit the program
                                return false;
                        }

                        // Make sure only one word has been stored
                        if(numArgs != 1)
                        {
                                // Return an error message
                                eflag = error(i_file, lineNumber, "Syntax error. Check label identifier.");
                                // Exit the program
                                return false;
                        }

                        // Create a new label
                        label myLabel = label(arguments[0], offset);

```

```
            // If the label has an invalid name, return an error
            if(!myLabel.isValid())
            {
                // Return an error message
                eflag = error(i_file, lineNumber, "Label error. Invalid label identifier.");
                // Exit the program
                return false;
            }

            // Make sure a label with the same name has not already been inserted into the array
            for(unsigned int curLblNum = 0; curLblNum < num_labels; curLblNum++)
            {
                if(myLabel.getName() == list_labels[curLblNum].getName())
                {
                    // Return an error message
                    eflag = error(i_file, lineNumber, "Syntax error. Label \"" + myLabel.getName() + "\" already exists.");
                    // Exit the program
                    return false;
                }
            }

            // Make sure no array overflow will occur
            if(num_labels >= list_labels_size)
            {
                // Return an error message
                eflag = error(i_file, lineNumber, "Assembler error. Label array overflow. Maximum number of labels reached.");
                // Exit the program
                return false;
            }

            // Add the current label to the list for storage
            list_labels[num_labels] = myLabel;
            num_labels++;

            // ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            // Enable the following line to signal the creation of a new label
            // ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            // cout << "Label \"" << arguments[0] << "\" created at address " << offset << "!" << endl;

            // Reset the value of word
            word = "";

            // Process the rest of the line as if the label has been removed
            numArgs = 0;

            continue;
        }

        // If the current character is a comma, and parameters are currently being read, store the word
        if(line.at(i) == ',' && word != "")
        {
            // Detect if the comma was unnecessary, as no other arguments have been stored yet
            if(numArgs == 0)
            {
                // Return an error message
                eflag = error(i_file, lineNumber, "Syntax error. Unnecessary comma?");
                // Exit the program
                return false;
            }

            // Store the current word
            if(!addArg(word, arguments, numArgs))
            {
                // Return an error message
                eflag = error(i_file, lineNumber, "Syntax error. Argument array overflow.");
                // Exit the program
                return false;
            }
        }

        // If the current character is not whitespace, add the uppercase version of the current character to the word
        else if(!isWhitespace(line.at(i)))
            word += toUpper(line.at(i));

        // If whitespace is encountered, word is not empty, and no arguments
        // Are currently stored, then store the first word as a command
        else if(word.length() != 0 && numArgs == 0 && isWhitespace(line.at(i)))
        {
            if(!addArg(word, arguments, numArgs))
            {
                // Return an error message
                eflag = error(i_file, lineNumber, "Syntax error. Argument array overflow. Check line?");
                // Exit the program
                return false;
            }
            continue;
        }

        // If this is the end of the line, and word contains data, store it depending upon if we are storing commands or parameters
        if(i == line.length() - 1)
        {
            // If the final character is a comma, return an error
            if(line.at(i) == ',')
            {
                // Return an error message
                eflag = error(i_file, lineNumber, "Syntax error. Unnecessary comma?");
                // Exit the program
                return false;
            }

            if(word != "")
            {
                // Store the current word
                if(!addArg(word, arguments, numArgs))
                {
                    // Return an error message
                    eflag = error(i_file, lineNumber, "Syntax error. Argument array overflow. Check line?");
                    // Exit the program
                    return false;
                }
            }
        }
    }

    // +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    // Enable the following lines of code to output all arguments to the screen
    // +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    // cout << endl;
    // for(unsigned char a = 0; a < numArgs; a++)
    //      cout << arguments[a] << "\t";


    // ***********************************************************
    // Begin processing the arguments and generating machine code
    // ***********************************************************

```

```
        // Check if at least one argument was entered
        if(numArgs > 0)
        {
            // Process all single argument commands
            if(numArgs == 1)
            {
                // CLS - 00E0
                if(arguments[0] == "CLS")
                {
                    // Add corresponding machine code to memory
                    memory[offset] = 0x00; offset++;
                    memory[offset] = 0xE0; offset++;
                    // Read the next line
                    continue;
                }

                // RET - 00EE
                if(arguments[0] == "RET")
                {
                    // Add corresponding machine code to memory
                    memory[offset] = 0x00; offset++;
                    memory[offset] = 0xEE; offset++;
                    // Read the next line
                    continue;
                }

                // Unknown command
                eflag = error(i_file, lineNumber, "Syntax error. Unknown command \"" + arguments[0] + "\".");
                // Read the next line
                continue;
            }

            // Process all double argument commands
            if(numArgs == 2)
            {
                // Macro: .START ADDR
                // Signifies start address
                if(arguments[0] == ".START")
                {
                    // Make sure no other commands have been entered yet
                    if(offset != 0)
                    {
                            // Return an error message
                            eflag = error(i_file, lineNumber, "Assembler error. Input code attempts to modify start address after other commands have been processed.");
                            // Exit the program
                            return false;
                    }

                    unsigned short result = 0x0000;
                    // Try to convert the second argument into an address
                    if(!strTo12Bit(arguments[1], result))
                    {
                            // Return an error message
                            eflag = error(i_file, lineNumber, "Syntax error. Invalid number \"" + arguments[1] + "\" designated as start address.");
                            // Exit the program
                            return false;
                    }

                    // Set the start address
                    start = result;

                    // Read the next line
                    continue;
                }

                // DB BYTE
                // Store a byte
                if(arguments[0] == "DB")
                {
                    unsigned char result = 0x00;
                    // Try to convert the second argument into a byte
                    if(!strToByte(arguments[1], result))
                    {
                        // Invalid argument
                        eflag = error(i_file, lineNumber, "Syntax error. Invalid argument \"" + arguments[2] + "\". Expected an 8-bit number.");
                        // Read the next line
                        continue;
                    }

                    memory[offset] = result;
                    offset++;

                    // Read the next line
                    continue;
                }

                // DW WORD
                // Store a word
                if(arguments[0] == "DW")
                {
                    unsigned short result = 0x0000;
                    // Try to convert the second argument into a byte
                    if(!strToWord(arguments[1], result))
                    {
                        // Invalid argument
                        eflag = error(i_file, lineNumber, "Syntax error. Invalid argument \"" + arguments[2] + "\". Expected a 16-bit number.");
                        // Read the next line
                        continue;
                    }

                    memory[offset] = (unsigned char)(result >> 8);      offset++;
                    memory[offset] = (unsigned char)(result & 0x00FF);  offset++;

                    // Read the next line
                    continue;
                }

                // SYS ADDR - 0NNN
                if(arguments[0] == "SYS")
                {
                    unsigned short result = 0x0000;
                    // Try to convert the second argument into an address
                    // If that fails, process the second argument as a label reference
                    if(!strTo12Bit(arguments[1], result))
                    {
                        // Create a new label reference
                        lref myLref = lref(arguments[1], offset);

                        // Make sure no array overflow will occur
                        if(num_lrefs >= list_lrefs_size)
                        {
                            // Return an error message
                            eflag = error(i_file, lineNumber, "Assembler error. Label reference array overflow. Maximum number of label references reached.");
                            // Exit the program
                            return false;
                        }

                        // Add the current label to the list for storage
                        list_lrefs[num_lrefs] = myLref;
                        num_lrefs++;
                    }

                    // Add corresponding machine code to memory
                    memory[offset] = 0x00 + ((result & 0x0F00) >> 8);   offset++;
                    memory[offset] = 0x00 + (result & 0x00FF);          offset++;
                    // Read the next line
                    continue;
            }

            // JP ADDR - 1NNN
            if(arguments[0] == "JP")
            {
                    unsigned short result = 0x0000;
                    // Try to convert the second argument into an address
                    // If that fails, process the second argument as a label reference
                    if(!strTo12Bit(arguments[1], result))
                    {
                            // Create a new label reference
                            lref myLref = lref(arguments[1], offset);

                            // Make sure no array overflow will occur
                            if(num_lrefs >= list_lrefs_size)
                            {
                                    // Return an error message
                                    eflag = error(i_file, lineNumber, "Assembler error. Label reference array overflow. Maximum number of label references reached.");
                                    // Exit the program
                                    return false;
                            }

                            // Add the current label to the list for storage
                            list_lrefs[num_lrefs] = myLref;
                            num_lrefs++;
                    }

                    // Add corresponding machine code to memory
                    memory[offset] = 0x10 + ((result & 0x0F00) >> 8);   offset++;
                    memory[offset] = 0x00 + (result & 0x00FF);          offset++;
                    // Read the next line
                    continue;
            }

            // CALL ADDR - 2NNN
            if(arguments[0] == "CALL")
            {
                    unsigned short result = 0x0000;
                    // Try to convert the second argument into an address
                    // If that fails, process the second argument as a label reference
                    if(!strTo12Bit(arguments[1], result))
                    {
                            // Create a new label reference
                            lref myLref = lref(arguments[1], offset);

                            // Make sure no array overflow will occur
                            if(num_lrefs >= list_lrefs_size)
                            {
                                    // Return an error message
                                    eflag = error(i_file, lineNumber, "Assembler error. Label reference array overflow. Maximum number of label references reached.");
                                    // Exit the program
                                    return false;
                            }

                            // Add the current label to the list for storage
                            list_lrefs[num_lrefs] = myLref;
                            num_lrefs++;
                    }

```

```
                    // Add corresponding machine code to memory
                    memory[offset] = 0x20 + ((result & 0x0F00) >> 8);   offset++;
                    memory[offset] = 0x00 + (result & 0x00FF);          offset++;
                    // Read the next line
                    continue;
                }

                // SKP VX - EX9E
                if(arguments[0] == "SKP")
                {
                    unsigned char byte = 0x00;
                    // Try to convert the second argument into a register
                    if(!strToRegister(arguments[1], byte))
                    {
                        // Unknown register
                        eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                        // Read the next line
                        continue;
                    }

                    // Add corresponding machine code to memory
                    memory[offset] = 0xE0 + byte; offset++;
                    memory[offset] = 0x9E;        offset++;
                    // Read the next line
                    continue;
                }

                // SKNP VX - EXA1
                if(arguments[0] == "SKNP")
                {
                    unsigned char byte = 0x00;
                    // Try to convert the second argument into a register
                    if(!strToRegister(arguments[1], byte))
                    {
                        // Unknown register
                        eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                        // Read the next line
                        continue;
                    }

                    // Add corresponding machine code to memory
                    memory[offset] = 0xE0 + byte; offset++;
                    memory[offset] = 0xA1;        offset++;
                    // Read the next line
                    continue;
                }

                // SHR VX - 8XY6
                if(arguments[0] == "SHR")
                {
                    unsigned char byte = 0x00;
                    // Try to convert the second argument into a register
                    if(!strToRegister(arguments[1], byte))
                    {
                        // Unknown register
                        eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                        // Read the next line
                        continue;
                    }

                    // Add corresponding machine code to memory
                    memory[offset] = 0x80 + byte;        offset++;
                    memory[offset] = 0x06 + (byte << 4); offset++;
                    // Read the next line
                    continue;
                }
```

```

            // SHL VX - 8XYE
            if(arguments[0] == "SHL")
            {
                unsigned char byte = 0x00;
                // Try to convert the second argument into a register
                if(!strToRegister(arguments[1], byte))
                {
                    // Unknown register
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                    // Read the next line
                    continue;
                }

                // Add corresponding machine code to memory
                memory[offset] = 0x80 + byte;        offset++;
                memory[offset] = 0x0E + (byte << 4); offset++;
                // Read the next line
                continue;
            }

            // Unknown command
            eflag = error(i_file, lineNumber, "Syntax error. Unknown command \"" + arguments[0] + " " + arguments[1] + "\".");
            // Read the next line
            continue;
        }

        // Process all triple argument commands
        if(numArgs == 3)
        {
            // SE VX, BYTE - 3XKK
            // SE VX, VY   - 5XY0
            if(arguments[0] == "SE")
            {
                unsigned char byte1 = 0x00;
                // Try to convert the second argument into a register
                if(!strToRegister(arguments[1], byte1))
                {
                    // Unknown register
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                    // Read the next line
                    continue;
                }

                unsigned char byte2 = 0x00;
                // Try to convert the third argument into a byte
                if(strToByte(arguments[2], byte2))
                {
                    byte1 += 0x30;
                }
                // Try to convert the third argument into a register
                else if(strToRegister(arguments[2], byte2))
                {
                    byte1 += 0x50;
                    byte2 = byte2 << 4;
                }
                else
                {
                    // Invalid argument
                    eflag = error(i_file, lineNumber, "Syntax error. Invalid argument \"" + arguments[2] + "\".");
                    // Read the next line
                    continue;
                }

                // Add corresponding machine code to memory
                memory[offset] = byte1;       offset++;
                memory[offset] = byte2;       offset++;
                // Read the next line
                continue;
            }

            // SNE VX, BYTE        - 4XKK
            // SNE VX, VY  - 9XY0
            if(arguments[0] == "SNE")
            {
                unsigned char byte1 = 0x00;
                // Try to convert the second argument into a register
                if(!strToRegister(arguments[1], byte1))
                {
                    // Unknown register
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                    // Read the next line
                    continue;
                }

                unsigned char byte2 = 0x00;
                // Try to convert the third argument into a byte
                if(strToByte(arguments[2], byte2))
                {
                    byte1 += 0x40;
                }
                // Try to convert the third argument into a register
                else if(strToRegister(arguments[2], byte2))
                {
                    byte1 += 0x90;
                    byte2 = byte2 << 4;
                }
                else
                {
                    // Invalid argument
                    eflag = error(i_file, lineNumber, "Syntax error. Invalid argument \"" + arguments[2] + "\".");
                    // Read the next line
                    continue;
                }

                // Add corresponding machine code to memory
                memory[offset] = byte1;       offset++;
                memory[offset] = byte2;       offset++;
                // Read the next line
                continue;
            }

            // ADD VX, BYTE        - 7XKK
            // ADD VX, VY  - 8XY4
            if(arguments[0] == "ADD" && arguments[1] != "I")
            {
                unsigned char byte1 = 0x00;
                // Try to convert the second argument into a register
                if(!strToRegister(arguments[1], byte1))
                {
                    // Unknown register
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                    // Read the next line
                    continue;
                }

                unsigned char byte2 = 0x00;
                // Try to convert the third argument into a byte
                if(strToByte(arguments[2], byte2))
                {
                    byte1 += 0x70;
                }
                // Try to convert the third argument into a register
                else if(strToRegister(arguments[2], byte2))
                {
```

```
                                        byte1 += 0x80;
                                        byte2 = (byte2 << 4) + 0x04;
                                }
                                else
                                {
                                        // Invalid argument
                                        eflag = error(i_file, lineNumber, "Syntax error. Invalid argument \"" + arguments[2] + "\".");
                                        // Read the next line
                                        continue;
                                }

                                // Add corresponding machine code to memory
                                memory[offset] = byte1;       offset++;
                                memory[offset] = byte2;       offset++;
                                // Read the next line
                                continue;
                        }

                        // ADD I, VX - FX1E
                        if(arguments[0] == "ADD" && arguments[1] == "I")
                        {
                                unsigned char byte = 0x00;
                                // Try to convert the third argument into a register
                                if(!strToRegister(arguments[2], byte))
                                {
                                        // Unknown register
                                        eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[2] + "\".");
                                        // Read the next line
                                        continue;
                                }

                                // Add corresponding machine code to memory
                                memory[offset] = 0xF0 + byte; offset++;
                                memory[offset] = 0x1E;        offset++;
                                // Read the next line
                                continue;
                        }

                        // RND VX, BYTE - CXKK
                        if(arguments[0] == "RND")
                        {
                                unsigned char byte1 = 0x00;
                                // Try to convert the second argument into a register
                                if(!strToRegister(arguments[1], byte1))
                                {
                                        // Unknown register
                                        eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                                        // Read the next line
                                        continue;
                                }

                                unsigned char byte2 = 0x00;
                                // Try to convert the third argument into a byte
                                if(!strToByte(arguments[2], byte2))
                                {
                                        // Invalid argument
                                        eflag = error(i_file, lineNumber, "Syntax error. Invalid argument \"" + arguments[2] + "\". Expected an 8-bit number.");
                                        // Read the next line
                                        continue;
                                }

                                // Add corresponding machine code to memory
                                memory[offset] = 0xC0 + byte1;       offset++;
                                memory[offset] = byte2;              offset++;
                                // Read the next line
                                continue;
                        }
```

```

            // OR  VX, VY - 8XY1
            // AND VX, VY - 8XY2
            // XOR VX, VY - 8XY3
            if(arguments[0] == "OR" || arguments[0] == "AND" || arguments[0] == "XOR")
            {
                unsigned char byte1 = 0x00;
                // Try to convert the second argument into a register
                if(!strToRegister(arguments[1], byte1))
                {
                    // Unknown register
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                    // Read the next line
                    continue;
                }

                unsigned char byte2 = 0x00;
                // Try to convert the third argument into a register
                if(!strToRegister(arguments[2], byte2))
                {
                    // Invalid argument
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[2] + "\".");
                    // Read the next line
                    continue;
                }

                // Shift byte2 left by four bits
                byte2 = byte2 << 4;

                // Determine what number to end the command with
                if(arguments[0] == "OR")
                    byte2 += 0x01;
                if(arguments[0] == "AND")
                    byte2 += 0x02;
                if(arguments[0] == "XOR")
                    byte2 += 0x03;

                // Add corresponding machine code to memory
                memory[offset] = 0x80 + byte1;      offset++;
                memory[offset] = byte2;             offset++;
                // Read the next line
                continue;
            }

            // SUB VX, VY - 8XY5
            if(arguments[0] == "SUB")
            {
                unsigned char byte1 = 0x00;
                // Try to convert the second argument into a register
                if(!strToRegister(arguments[1], byte1))
                {
                    // Unknown register
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                    // Read the next line
                    continue;
                }

                unsigned char byte2 = 0x00;
                // Try to convert the third argument into a register
                if(!strToRegister(arguments[2], byte2))
                {
                    // Invalid argument
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[2] + "\".");
                    // Read the next line
                    continue;
                }

                // Shift byte2 left by four bits
                byte2 = byte2 << 4;

                // Add corresponding machine code to memory
                memory[offset] = 0x80 + byte1;      offset++;
                memory[offset] = byte2 + 0x05;      offset++;
                // Read the next line
                continue;
            }

            // SUBN VX, VY - 8XY7
            if(arguments[0] == "SUBN")
            {
                unsigned char byte1 = 0x00;
                // Try to convert the second argument into a register
                if(!strToRegister(arguments[1], byte1))
                {
                    // Unknown register
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                    // Read the next line
                    continue;
                }

                unsigned char byte2 = 0x00;
                // Try to convert the third argument into a register
                if(!strToRegister(arguments[2], byte2))
                {
                    // Invalid argument
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[2] + "\".");
                    // Read the next line
                    continue;
                }

                // Shift byte2 left by four bits
                byte2 = byte2 << 4;

                // Add corresponding machine code to memory
                memory[offset] = 0x80 + byte1;      offset++;
                memory[offset] = byte2 + 0x07;      offset++;
                // Read the next line
                continue;
            }

            // SHR VX, VY - 8XY6
            if(arguments[0] == "SHR")
            {
                unsigned char byte1 = 0x00;
                // Try to convert the second argument into a register
                if(!strToRegister(arguments[1], byte1))
                {
                    // Unknown register
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                    // Read the next line
                    continue;
                }

                unsigned char byte2 = 0x00;
                // Try to convert the third argument into a register
                if(!strToRegister(arguments[2], byte2))
                {
                    // Invalid argument
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[2] + "\".");
                    // Read the next line
                    continue;
                }

```

```
                                // Shift byte2 left by four bits
                                byte2 = byte2 << 4;

                                // Add corresponding machine code to memory
                                memory[offset] = 0x80 + byte1;          offset++;
                                memory[offset] = byte2 + 0x06;          offset++;
                                // Read the next line
                                continue;
                        }

                        // SHL VX, VY - 8XYE
                        if(arguments[0] == "SHL")
                        {
                                unsigned char byte1 = 0x00;
                                // Try to convert the second argument into a register
                                if(!strToRegister(arguments[1], byte1))
                                {
                                        // Unknown register
                                        eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                                        // Read the next line
                                        continue;
                                }

                                unsigned char byte2 = 0x00;
                                // Try to convert the third argument into a register
                                if(!strToRegister(arguments[2], byte2))
                                {
                                        // Invalid argument
                                        eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[2] + "\".");
                                        // Read the next line
                                        continue;
                                }

                                // Shift byte2 left by four bits
                                byte2 = byte2 << 4;

                                // Add corresponding machine code to memory
                                memory[offset] = 0x80 + byte1;          offset++;
                                memory[offset] = byte2 + 0x0E;          offset++;
                                // Read the next line
                                continue;
                        }

                        // JP V0, ADDR - BNNN
                        if(arguments[0] == "JP")
                        {
                                // Make sure the second argument is the correct register
                                if(arguments[1] != "V0")
                                {
                                        // Unknown register
                                        eflag = error(i_file, lineNumber, "Syntax error. Invalid register \"" + arguments[1] + "\". Expected V0.");
                                        // Read the next line
                                        continue;
                                }

                                unsigned short result = 0x0000;
                                // Try to convert the third argument into an address
                                // If that fails, process the third argument as a label reference
                                if(!strTo12Bit(arguments[2], result))
                                {
                                        // Create a new label reference
                                        lref myLref = lref(arguments[2], offset);

                                        // Make sure no array overflow will occur
                                        if(num_lrefs >= list_lrefs_size)
                                        {
                                                // Return an error message
                        eflag = error(i_file, lineNumber, "Assembler error. Label reference array overflow. Maximum number of label references reached.");
                        // Exit the program
                        return false;
                    }

                    // Add the current label to the list for storage
                    list_lrefs[num_lrefs] = myLref;
                    num_lrefs++;
                }

                // Add corresponding machine code to memory
                memory[offset] = 0xB0 + ((result & 0x0F00) >> 8);   offset++;
                memory[offset] = 0x00 + (result & 0x00FF);          offset++;
                // Read the next line
                continue;
            }

            // LD VX,  BYTE        - 6XKK
            // LD I,   ADDR        - ANNN
            // LD VX,  DT  - FX07
            // LD VX,  K   - FX0A
            // LD VX,  [I] - FX65
            // LD VX,  VY  - 8XY0
            // LD DT,  VX  - FX15
            // LD ST,  VX  - FX18
            // LD F,   VX  - FX29
            // LD B,   VX  - FX33
            // LD [I], VX  - FX55
            if(arguments[0] == "LD")
            {
                unsigned char byte1 = 0x00;
                unsigned char byte2 = 0x00;

                // Try to convert the third argument into a register
                if(strToRegister(arguments[2], byte2))
                {
                    // Try to convert the second argument into a register
                    if(strToRegister(arguments[1], byte1))
                    {
                        byte1 += 0x80;

                        // Shift byte 2 left by four bits
                        byte2 = byte2 << 4;
                    }
                    else if(arguments[1] == "DT")
                    {
                        byte1 += 0xF0 + byte2;
                        byte2 = 0x15;
                    }
                    else if(arguments[1] == "ST")
                    {
                        byte1 += 0xF0 + byte2;
                        byte2 = 0x18;
                    }
                    else if(arguments[1] == "F")
                    {
                        byte1 += 0xF0 + byte2;
                        byte2 = 0x29;
                    }
                    else if(arguments[1] == "B")
                    {
                        byte1 += 0xF0 + byte2;
                        byte2 = 0x33;
                    }
                    else if(arguments[1] == "[I]")
                    {
                        byte1 += 0xF0 + byte2;
                        byte2 = 0x55;
                    }
                    else
                    {
                        // Invalid argument
                        eflag = error(i_file, lineNumber, "Syntax error. Invalid argument \"" + arguments[1] + "\".");
                        // Read the next line
                        continue;
                    }
                }

                // Try to convert the second argument into a register
                else if(strToRegister(arguments[1], byte1))
                {
                    unsigned short result = 0x0000;

                    // Try to convert the third argument into a byte
                    if(strToByte(arguments[2], byte2))
                    {
                        byte1 += 0x60;
                    }
                    else if(arguments[2] == "DT")
                    {
                        byte1 += 0xF0;
                        byte2 = 0x07;
                    }
                    else if(arguments[2] == "K")
                    {
                        byte1 += 0xF0;
                        byte2 = 0x0A;
                    }
                    else if(arguments[2] == "[I]")
                    {
                        byte1 += 0xF0;
                        byte2 = 0x65;
                    }
                    else
                    {
                        // Invalid argument
                        eflag = error(i_file, lineNumber, "Syntax error. Invalid argument \"" + arguments[2] + "\".");
                        // Read the next line
                        continue;
                    }
                }

                else if(arguments[1] == "I")
                {
                    unsigned short result = 0x0000;
                    // Try to convert the third argument into an address
                    // If that fails, process the third argument as a label reference
                    if(!strTo12Bit(arguments[2], result))
                    {
                        // Create a new label reference
                        lref myLref = lref(arguments[2], offset);

                        // Make sure no array overflow will occur
                        if(num_lrefs >= list_lrefs_size)
                        {
                            // Return an error message
                            eflag = error(i_file, lineNumber, "Assembler error. Label reference array overflow. Maximum number of label references reached.");
                            // Exit the program
                            return false;
                        }

                        // Add the current label to the list for storage
                        list_lrefs[num_lrefs] = myLref;
                        num_lrefs++;
                    }

                    byte1 = 0xA0 + ((result & 0x0F00) >> 8);
                    byte2 = 0x00 + (result & 0x00FF);
                }

                // Unknown command
                else
                {
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown command \"" + arguments[0] + " " + arguments[1] + ", " + arguments[2] + "\".");
                }

                // Add corresponding machine code to memory
                memory[offset] = byte1;       offset++;
                memory[offset] = byte2;       offset++;
                // Read the next line
                continue;
            }

            // Unknown command
            eflag = error(i_file, lineNumber, "Syntax error. Unknown command \"" + arguments[0] + " " + arguments[1] + ", " + arguments[2] + "\".");
            // Read the next line
            continue;
        }

        // Process all quadruple argument commands
        if(numArgs == 4)
        {
            // DRW VX, VY, NIBBLE - DXYN
            if(arguments[0] == "DRW")
            {
                unsigned char byte1 = 0x00;
                // Try to convert the second argument into a register
                if(!strToRegister(arguments[1], byte1))
                {
                    // Unknown register
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[1] + "\".");
                    // Read the next line
                    continue;
                }

                unsigned char byte2 = 0x00;
                // Try to convert the third argument into a register
                if(!strToRegister(arguments[2], byte2))
                {
                    // Unknown register
                    eflag = error(i_file, lineNumber, "Syntax error. Unknown register \"" + arguments[2] + "\".");
                    // Read the next line
                    continue;
                }

                unsigned char nibble = 0x00;
                // Try to convert the third argument into a nibble
                if(!strToNibble(arguments[3], nibble))
                {
                    // Invalid argument
                    eflag = error(i_file, lineNumber, "Syntax error. Invalid argument \"" + arguments[3] + "\". Expected a 4-bit number.");
                    // Read the next line
                    continue;
                }

                // Add corresponding machine code to memory
                memory[offset] = 0xD0 + byte1;              offset++;
                memory[offset] = (byte2 << 4) + nibble;     offset++;
                // Read the next line
                continue;
```

```
                                }

                                // Unknown command
                                eflag = error(i_file, lineNumber, "Syntax error. Unknown command \"" + arguments[0] + " " + arguments[1] + ", " + arguments[2] + ", " + arguments[3] + "\".");
                                // Read the next line
                                continue;
                        }
                }
        }

        // ******************************************
        // Begin linking label references with labels
        // ******************************************

        // Loop through the array of label references
        for(unsigned char i_lref; i_lref < num_lrefs; i_lref++)
        {
                // The current label reference
                lref current_lref = list_lrefs[i_lref];

                // Locate the label being referenced by this label reference
                bool found = false;
                unsigned char i_label;
                for(i_label = 0; i_label < num_labels; i_label++)
                {
                        if(current_lref.getName() == list_labels[i_label].getName())
                        {
                                found = true;
                                break;
                        }
                }

                // If it is not found, output an error
                if(!found)
                {
                        // Output an error
                        eflag = error(i_file, 0, "Label error. Label \"" + current_lref.getName() + "\" not found.");
                        // Exit the program
                        return false;
                }

                // The current label
                label current_label = list_labels[i_label];

                // Jump to the offset address given by the label reference and insert the address designated by the corresponding label
                memory[current_lref.getAddress()]       += ((current_label.getAddress() + start) & 0x0F00) >> 8;
                memory[current_lref.getAddress() + 1]   += ((current_label.getAddress() + start) & 0x00FF);
        }

        // Close the source file
        source.close();

        // *****************************************
        // Output all CHIP-8 code to the output file
        // *****************************************

        // Has an error occured?
        if(!eflag)
        {
                // Open the output file for writing
                ofstream output;
                output.open(o_file.c_str(), ofstream::out);

                // Check if the output file has not been opened correctly
                if(!output.is_open())
                {
                        // Return an error message
                        error(o_file, 0, "File error. Could not open file \"" + o_file + "\" for output.");
                        // Exit the program
                        return false;
                }
                // Output all the generated machine code to the file
                for(unsigned int current_byte = 0; current_byte < offset; current_byte++)
                {
                        output << memory[current_byte];
                }

                // Close the output file
                output.close();
        }

        // ++++++++++++++++++++++++++++++++++++++++++++
        // Enable the following lines of code to output
        // all generated machine code to the screen++++
        // ++++++++++++++++++++++++++++++++++++++++++++
        /*
        cout << "###################" << endl;
        cout << "Machine Code Output" << endl;
        cout << "###################" << endl;

        if(offset > 0) {
        for(unsigned int z = 0; z < offset - 1; z += 2)
                printf ("#%03X \t %02X %02X \n", z + start, memory[z], memory[z+1]);
        }
        */

        return 0;
      }
```

# 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

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. 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>.