```c
/*
 * Brandon Bell
 * CSCI: 4576
 * Midterm 1: Conway's Game of Life.
 * 10-5-16
 *
 * Popt was causeing my some frustrations so i grabbed a working example from
 * https://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch15s02s04.html
 * and modified it to fit my needs.
 *
 * Additionaly, I barrowed the globals.h, pgm, and pprintf file provided on
 * moddle.
 *
 * TO-DO
 * -> Count Bugs
 *
 * -> Serial np=1 rules.
 * -> Block implementation
 * -> Rule implementation
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
#include <popt.h>

// Include global variables.
#define __MAIN
#include "globals.h"
#undef __MAIN

// User includes
#include "pprintf.h"
#include "pgm.h"

////////////////////////////////////////////////////////////////////////////////
// Global Variables.
////////////////////////////////////////////////////////////////////////////////

// MPI Variables
int rank;
int np;
int my_name_len;
char my_name[255];

// global verbose tag.
int verbose;

////////////////////////////////////////////////////////////////////////////////
// Count Bugs:
// Find the number of bugs alive in the processes peice of the world.
//
// Inputs:
// - Ptr to the array for wich the bugs are to be counted.
// - int representing the iteration the array represents.
//
// Results:
// - Summs the bugs into a local variable and prints the result to stdout.
////////////////////////////////////////////////////////////////////////////////
void countBugs( int *world, int iteration )
{
    int count = 0;
    // Loop variables.
    int i;
    // hight and width of world minus any ghost rows/cols.
    int width;
    int height;

    // determine if the world is parallelized. If the program is running
    // serialy then there are no ghost rows and if it's running parallel then
    // ghost rows need to be alloted for and not count to prevent double
    // counting bugs. Set the loop variable ranges based on this information.

    // The serial Case.
    if ( np == 1 )
    {
        width   = field_width;
        height  = field_height;
    }
    // The block distribution case.
    else if ( ncols == 1 && np > 1 )
    {
        width   = field_width;
        height  = field_height;
        //printf("The count array address is %d, col %d, row %d\n",world, ncols, nrows);
    }

    // Count the bugs for a serial world. This gets me the known 26,301 bugs.
    for ( i=(width+1); i<( width*height - width -1 ); i++ )
    {
        if ( *(world + i) == 1 )
            count++;
    }
    /* for ( i=0; i<( width*height*sizeof(int) ); i++ ) */
    /* { */
        /* if ( *(world + i) == 1 ) */
            /* count++; */
    /* } */


    printf("[ %d, %d , %d ]\n", rank, iteration, count);
}


int main( int argc, char* argv[] )
{
    MPI_Status  status;
    int tag = 0;
    // Popt cmd line argument variables.
    int iter_number  = 0;
    int count_alive  = 0;
    int block_type   = 0;
    int verbose      = 0;
    int async_comm   = 0;
    int checker_type = 0;
    char* filename   = "";

    // Loop variables.
    int i;
    int j;

    // Conways variables.
```

```
    int neighbors;

    ////////////////////////////////////////////////////////////////////////
    // Parse the comand line arguments with Popt.
    ////////////////////////////////////////////////////////////////////////
    struct poptOption optionsTable[] =
    {
    {"interations",   'i',  POPT_ARG_INT,    &iter_number,  0, "Set the number of world iter
ations.", "2, 3, ... n" },
    {"count-alive",   'c',  POPT_ARG_INT,    &count_alive,  0, "Specifiy the iteration after
 which to count bugs." , NULL },
    {"verbose",       'v',  POPT_ARG_NONE,   &verbose,      0, "set verbose level to 1." , N
ULL },
    {"block",         'b',  POPT_ARG_NONE,   &block_type,   0, "Set the process distribution
 to block type.", NULL },
    {"async-comm",    NULL, POPT_ARG_NONE,   &async_comm,   0, "Set the communication type t
o asyncronous.", NULL },
    {"checker-board", NULL, POPT_ARG_NONE,   &checker_type, 0, "Set the process distribution
 to checker board type.", NULL },
    {"filename",      'f',  POPT_ARG_STRING, &filename,     0, "Set the name of the world fi
le to read.", "*.pgm" },
    POPT_AUTOALIAS
    POPT_AUTOHELP
    POPT_TABLEEND
    };
    poptContext context = poptGetContext( "popt1", argc, argv, &optionsTable, 0);
    int option          = poptGetNextOpt(context);

    // Handle verbose output of command line arguments if v switch set.
    if ( verbose == 1 )
    {
        printf("option = %d\n", option);
        printf("After processing, options have values:\n");
        printf("\t iterations holds %d\n", iter_number);
        printf("\t count_alive holds %d\n", count_alive);
        printf("\t block flag holds %d\n", block_type);
        printf("\t verbose flag holds %d\n", verbose);
        printf("\t checker flag holds %d\n", checker_type);
        printf("\t async_comm flag holds %d\n", async_comm);
        printf("\t filename holds [%s]\n", filename);
    }

    ////////////////////////////////////////////////////////////////////////
    // Initialize MPI and retreive world size, p's rank, and p's node name.
    ////////////////////////////////////////////////////////////////////////
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &np);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(my_name,&my_name_len);
    // Verbose output of basic MPI process information.
    if ( verbose == 1 )
        printf("[ %s, %d ] World size = %d\n", my_name, rank, np);


    ////////////////////////////////////////////////////////////////////////
    // Set-up the MPI and conways variables based on partition scheme being used
    // and read the starting world file.
    ////////////////////////////////////////////////////////////////////////

    // Make sure that only partition type is specified.
    if ( checker_type == 1 && block_type == 1 )
    {
        if ( rank == 0 )
        {
            printf(" => [ERROR] more than one partition scheme specified.\n");
        }
        // Call finalize before quiting, it's just good manners.
        MPI_Finalize();
        return 1;
    }
    // set ncol and nrows for the serial case.
    if ( np == 1 )
    {
        ncols = 1;
        nrows = 1;
    }
    // set ncol and n rows for the block distribution case.
    else if ( block_type == 1  )
    {
        my_row = rank;
        //my_col = 1;
        ncols = 1;
        nrows = np;

        ////////////////////////////////////////////////////////////////////////
        // Syncroness exchange of ghost rows.
        // The top row sends first and all the other processes recv first.
        ////////////////////////////////////////////////////////////////////////
        // If top p.
        if (rank == 0 )
        {
            // send the bottom gohst row to next p.
            MPI_Send( (field_a + field_width*field_height - 2*field_width -1 ),
                      field_width, MPI_INT, rank+1, tag, MPI_COMM_WORLD );
            MPI_Recv( (field_a + field_width*field_height - 2*field_width -1),
                      field_width, MPI_INT, rank+1, tag, MPI_COMM_WORLD, &status );
        }

        // If bottom p.
        else if (rank == np-1 )
        {
            // Send the top row to gohst row of above block and recv into top gohst // row.
            MPI_Send( (field_a + 2*field_width -1 ),
                      field_width, MPI_INT, rank-1, tag, MPI_COMM_WORLD );
            MPI_Recv( (field_a + 2*field_width -1),
                      field_width, MPI_INT, rank-1, tag, MPI_COMM_WORLD, &status );
        }

        // for the non edge processes.
        else
        {
            // Send the top row to gohst row of above block and recv into top gohst // row.
            MPI_Recv( (field_a + field_width*field_height - 2*field_width -1),
                      field_width, MPI_INT, rank-1, tag, MPI_COMM_WORLD, &status );
            MPI_Send( (field_a + 2*field_width -1 ),
                      field_width, MPI_INT, rank-1, tag, MPI_COMM_WORLD );
            // send the bottom gohst row to next p.
            MPI_Send( (field_a + field_width*field_height - 2*field_width -1 ),
                      field_width, MPI_INT, rank+1, tag, MPI_COMM_WORLD );
            MPI_Recv( (field_a + field_width*field_height - 2*field_width -1),
                      field_width, MPI_INT, rank+1, tag, MPI_COMM_WORLD, &status );
        }
    }
    else if ( checker_type == 1 )
    {
        printf("you choose a not yet implemented checkerbord distribution.\n");
```

```
        // Call finalize before quiting, it's just good manners.
        MPI_Finalize();
        return 1;
    }
    else
    {
        if ( rank == 0 )
        {
            printf(" => [ERROR] More than 1 process started but no partion scheme specified.
\n");
            printf("   -> Pleas specify one of --checker-board or -b, --block\n");
        }
        // Call finalize before quiting, it's just good manners.
        MPI_Finalize();
        return 1;
    }

    // Read the world file using the wonderful provided function.
    // populates field_a and _b with the data from that region of the file with
    // size accomidations for the ghost rows/cols. I'll use them as field_a is
    // the i array and field_b is the i+1 array initialy.
    readpgm(filename);

    // Get an initial Bug count.
    countBugs( field_a, 0 );

    ////////////////////////////////////////////////////////////////////////////
    // Play the game: serial version.
    ////////////////////////////////////////////////////////////////////////////
    int *swap;
    int l = field_width;
    int *cell;
    int *newcell;
    for (j=0; j<iter_number; j++)
    {
        for ( i=(field_width +1 ); i<( field_width*field_height - field_width -1 ); i++ )
        {
            cell = field_a + i;
            newcell = field_b + i;
            // Calculate a cells neighbors by summing the stencil.
            neighbors = 0;
            neighbors = *(cell-1)+ *(cell+1)+ *(cell-l)+ *(cell-l-1)+ *(cell-l+1)+ *(cell+l)
+ *(cell+l+1)+ *(cell+l-1);

            // Apply Conway's Rules.
            if ( *cell == 1 && neighbors <= 1 )
                *newcell = 0;
            else if ( *cell == 1 && neighbors >= 4 )
                *newcell = 0;
            else if ( (*cell == 1 && neighbors == 2) || neighbors == 3 )
                *newcell = 1;
            else if ( *cell == 0 && neighbors == 3 )
                *newcell = 1;
            swap = field_b;
            field_b = field_a;
            field_a = swap;
        }

        //printf("iter j %d\n", j);
        // Bug count output.
        if ( count_alive !=0 )
        {
            if ( (j+1) % count_alive == 0 )
            {
                // swap holds what ever the last i+1 ptr was, so the last
                // iteration.
                countBugs( swap, j );
            }
        }
    }

    ////////////////////////////////////////////////////////////////////////////
    // Final clean-up and shutdown.
    ////////////////////////////////////////////////////////////////////////////

    // Free the popt context memory.
    poptFreeContext(context);
    // Shutdown MPI.
    MPI_Finalize();

    return 0;
}
```

```c
// System includes
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mpi.h"

// User includes
#include "globals.h"
#include "pprintf.h"

typedef enum { false, true } bool; // Provide C++ style 'bool' type in C

bool readpgm( char *filename )
{
  // Read a PGM file into the local task
  //
  // Input: char *filename, name of file to read
  // Returns: True if file read successfully, False otherwise
  //
  // Preconditions:
  //  * global variables nrows, ncols, my_row, my_col must be set
  //
  // Side effects:
  //  * sets global variables local_width, local_height to local game size
  //  * sets global variables field_width, field_height to local field size
  //  * allocates global variables field_a and field_b

  pp_set_banner( "pgm:readpgm" );

  // Open the file
  if( rank==0 )
    pprintf( "Opening file %s\n", filename );
  FILE *fp = fopen( filename, "r" );
  if( !fp )
  {
    pprintf( "Error: The file '%s' could not be opened.\n", filename );
    return false;
  }

  // Read the PGM header, which looks like this:
  //  |P5        magic version number
  //  |900 900       width height
  //  |255         depth
  char header[10];
  int width, height, depth;
  int rv = fscanf( fp, "%6s\n%i %i\n%i\n", header, &width, &height, &depth );
  if( rv != 4 )
  {
    if(rank==0)
      pprintf( "Error: The file '%s' did not have a valid PGM header\n",
        filename );
    return false;
  }
  if( rank==0 )
    pprintf( "%s: %s %i %i %i\n", filename, header, width, height, depth );

  // Make sure the header is valid
  if( strcmp( header, "P5") )
  {
    if(rank==0)
      pprintf( "Error: PGM file is not a valid P5 pixmap.\n" );
    return false;
  }
  if( depth != 255 )
  {
    if(rank==0)
      pprintf( "Error: PGM file has depth=%i, require depth=255 \n",
        depth );
    return false;
  }

  // Make sure that the width and height are divisible by the number of
  // processors in x and y directions

  if( width % ncols )
  {
    if( rank==0 )
      pprintf( "Error: %i pixel width cannot be divided into %i cols\n",
          width, ncols );
    return false;
  }
  if( height % nrows )
  {
    if( rank==0 )
      pprintf( "Error: %i pixel height cannot be divided into %i rows\n",
          height, nrows );
    return false;
  }

  // Divide the total image among the local processors
  local_width = width / ncols;
  local_height = height / nrows;

  // Find out where my starting range it
  int start_x = local_width * my_col;
  int start_y = local_height * my_row;

  pprintf( "Hosting data for x:%03i-%03i y:%03i-%03i\n",
      start_x, start_x + local_width,
      start_y, start_y + local_height );

  // Create the array!
  field_width = local_width + 2;
  field_height = local_height + 2;
  field_a = (int *)malloc( field_width * field_height * sizeof(int));
  field_b = (int *)malloc( field_width * field_height * sizeof(int));

  // Read the data from the file. Save the local data to the local array.
  int b, ll, lx, ly;
  for( int y=0; y<height; y++ )
  {
    for( int x=0; x<width; x++ )
    {
      // Read the next character
      b = fgetc( fp );
      if( b == EOF )
      {
        pprintf( "Error: Encountered EOF at [%i,%i]\n", y,x );
        return false;
      }

      // From the PGM, black cells (b=0) are bugs, all other
      // cells are background
      if( b==0 )
      {
        b=1;
```

```
      }
      else
      {
        b=0;
      }


      // If the character is local, then save it!
      if( x >= start_x && x < start_x + local_width &&
        y >= start_y && y < start_y + local_height )
      {
        // Calculate the local pixels (+1 for ghost row,col)
        lx = x - start_x + 1;
        ly = y - start_y + 1;
        ll = (ly * field_width + lx );
        field_a[ ll ] = b;
        field_b[ ll ] = b;
      } // save local point

    } // for x
  } // for y

  fclose( fp );

  pp_reset_banner();
  return true;
}
```

```
/* $Id: pprintf.c,v 1.5 2006/02/09 20:42:25 mccreary Exp $ */

/*
 * Copyright (c) 2006 Sean McCreary <mccreary@mcwest.org>. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. The name of the author may not be used to endorse or promote products
 * derived from this software without specific prior written permission
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/* Pretty printf() wrapper for MPI processes */

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

#define PP_MAX_BANNER_LEN       14
#define PP_MAX_LINE_LEN         81
#define PP_PREFIX_LEN           27
#define PP_FORMAT               "[%3d:%03d] %-14s : "

static int pid = -1;
static int msgcount = 0;
static char banner[PP_MAX_BANNER_LEN] = "";
static char oldbanner[PP_MAX_BANNER_LEN] = "";

int init_pprintf(int);
int pp_set_banner(char *);
int pp_reset_banner();
int pprintf(char *, ...);

int init_pprintf( int my_rank )
{
    pp_set_banner("init_pprintf");
    pid = my_rank;
/*
    pprintf("PID is %d\n", pid);
*/
    return 0;
}

int pp_set_banner( char *newbanner )
{
    strncpy(oldbanner, banner, PP_MAX_BANNER_LEN);
    strncpy(banner, newbanner, PP_MAX_BANNER_LEN);
    return 0;
}

int pp_reset_banner()
{
    strncpy(banner, oldbanner, PP_MAX_BANNER_LEN);
    return 0;
}

int pprintf( char *format, ... )
{
    va_list ap;
    char output_line[PP_MAX_LINE_LEN];

    /* Construct prefix */
    snprintf(output_line, PP_PREFIX_LEN+1, PP_FORMAT, pid, msgcount, banner);

    va_start(ap, format);
    vsnprintf(output_line + PP_PREFIX_LEN,
              PP_MAX_LINE_LEN - PP_PREFIX_LEN, format, ap);
    va_end(ap);

    printf("%s", output_line);
    fflush(stdout);
    msgcount++;
    return 0;
}
```

```
// Conway's Game of Life
// Global variable include file
//
// CSCI 4576/5576 High Performance Scientific Computing
// Matthew Woitaszek

// <soapbox>
// This file contains global variables: variables that are defined throughout
// the entire program, even between multiple independent source files. Of
// course, global variables are generally bad, but they're useful here because
// it allows all of the source files to know their rank and the number of MPI
// tasks. But don't use it lightly.
//
// How it works:
//  * One .cpp file -- usually the one that contains main(), includes this file
//    within #define __MAIN, like this:
//      #define __MAIN
//      #include globals.h
//      #undef __MAIN
//  * The other files just "#include globals.h"

#ifdef __MAIN
// MPI procesess and node variables.
int rank;
int np;
int my_name_len;
char my_name[255];
// global verbose tag.
int verbose;
#else
extern int rank;
extern int np;
extern int my_name_len;
extern char *my_name;
// global verbose tag.
extern int verbose;
#endif

//
// Conway globals
//
#ifdef __MAIN
int nrows;          // Number of rows in our partitioning
int ncols;          // Number of columns in our partitioning
int my_row;         // My row number
int my_col;         // My column number

// Local logical game size
int local_width;    // Width and height of game on this processor
int local_height;

// Local physical field size
int field_width;        // Width and height of field on this processor
int field_height;       // (should be local_width+2, local_height+2)
int *field_a = NULL;      // The local data fields
int *field_b = NULL;

#else
extern int nrows;
extern int ncols;
extern int my_row;
extern int my_col;

extern int local_width;
extern int local_height;

extern int field_width;
extern int field_height;
extern int *field_a;
extern int *field_b;
#endif
```

```
typedef enum { false, true } bool; // Provide C++ style 'bool' type in C
bool readpgm( char *filename );
```

```
/* $Id: pprintf.h,v 1.3 2006/02/09 20:42:25 mccreary Exp $ */

/*
 * Copyright (c) 2006 Sean McCreary <mccreary@mcwest.org>. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. The name of the author may not be used to endorse or promote products
 * derived from this software without specific prior written permission
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

// Modified by Michael Oberg, 2015/10/01 to support both C or C++

#ifdef __cplusplus
extern "C" int init_pprintf(int);
extern "C" int pp_set_banner(char *);
extern "C" int pp_reset_banner();
extern "C" int pprintf(char *, ...);
#endif

extern int init_pprintf(int);
extern int pp_set_banner(char *);
extern int pp_reset_banner();
extern int pprintf(char *, ...);
```

```
CC=mpicc
flagGCC= -Wall -lm -lpopt
flagIntel= -Wall -lpopt

C_FILES = BellBrandon_Midterm1.c pgm.c pprintf.c
O_FILES = BellBrandon_Midterm1.o pgm.o pprintf.o
out=midterm

# all assumes gcc compiler wich needs -lm flag.
all:
        $(CC) $(flagGCC) -c $(C_FILES)
        $(CC) $(flagGCC) $(O_FILES) -o $(out)1

# Don't want the -lm flag on the intel complier for Stampede/Comet as they use
# thier own optomised math library.
intel:
        $(CC) $(flagIntel) -c $(C_FILES)
        $(CC) $(flagIntel) $(O_FILES) -o $(out)1

# Just compile the main program, not the library files I grabbed.
main:
        $(CC) $(flagGCC) -c BellBrandon_Midterm1.c
        $(CC) $(flagGCC) $(O_FILES) -o $(out)1

clean:
        rm $(out)*
        rm $(O_FILES)
```

.gitignore


```
conways_input.pgm
conways_c/
a.out
midterm1
*.o
```