Contents

1 Basic Test Results

```
Running...
Opening tar file
Draw2DCharArray.c
Makefile
PitPlugger.c
RadiusReader.c
OK
Tar extracted O.K.
Checking files...
OK
Making sure files are not empty...
OK
Importing files
OK
Compilation check...
Compiling...
gcc -Wall -Wvla -c -std=c99 Draw2DCharArray.c
gcc -Wall -Wvla -c DrawArrayDriver.c
gcc -Wall -Wvla Draw2DCharArray.o DrawArrayDriver.o -o DrawArrayDriver
OK
Compiling...
gcc -Wall -Wvla -c -std=c99 PitPlugger.c
gcc -Wall -Wvla -c RadiusReader.c
gcc -Wall -Wvla -lm PitPlugger.o Draw2DCharArray.o RadiusReader.o -o PitPlugger
OK
Compilation went without errors, BUT you must check to see if you got warnings!!!
Check some inputs:
Running test...
OK
OK

========================
= Checking coding style =
========================
 ** Total Violated Rules      : 0
 ** Total Errors Occurs       : 0
 ** Total Violated Files Count: 0
```

2 Draw2DCharArray.c

```c
//-----------------------------------------------------------------------------------------------
//                                 Ex2 - Draw2DCharArray
// General    :    Initialize the given char array to be filled with space and print it to the screen
// Input      :    arr - 2D char array
// Process    :    run over the array and fill/print
// Output     :    print of the array
//
//-----------------------------------------------------------------------------------------------
#include "Draw2DCharArray.h"
#include <stdio.h>

#define WHITE_SPACE ' '

/**
 * Initialize the given char array to be filled with space (' ') characters.
 */
void initializeArray(char arr[ROWS][COLS])
{
    for (int i = 0; i < ROWS; i++)
    {
        for (int j = 0; j < COLS; j++)
        {
            arr[i][j] = WHITE_SPACE;
        }
    }
}

/**
 * Draw the char array on the screen.
 * After each row go down a line ('\n').
 * Stop drawing when a nul character ('\0') is met.
 */
void drawArray(char arr[ROWS][COLS])
{
    for (int i = 0; i < ROWS; i++)
    {
        for (int j = 0; j < COLS; j++)
        {
            if (arr[i][j] == '\0')
            {
                return;
            }
            printf("%c", arr[i][j]);
        }
        printf("\n");
    }
}
```

3 Makefile

```
FULL_PATH = /cs/course/2014/slabc/public/ex2/inputOutput

all: PitPlugger DrawArrayDriver

tar: Draw2DCharArray.c RadiusReader.c PitPlugger.c Makefile
    tar cfv ex2.tar Draw2DCharArray.c RadiusReader.c PitPlugger.c Makefile

PitPlugger: PitPlugger.o Draw2DCharArray.o RadiusReader.o
    gcc -Wall -Wvla -lm PitPlugger.o Draw2DCharArray.o RadiusReader.o -o PitPlugger

DrawArrayDriver: Draw2DCharArray.o DrawArrayDriver.o
    gcc -Wall -Wvla Draw2DCharArray.o DrawArrayDriver.o -o DrawArrayDriver

PitPlugger.o: PitPlugger.c Draw2DCharArray.h RadiusReader.h
    gcc -Wall -Wvla -c -std=c99 PitPlugger.c

Draw2DCharArray.o: Draw2DCharArray.c Draw2DCharArray.h
    gcc -Wall -Wvla -c -std=c99 Draw2DCharArray.c

RadiusReader.o: RadiusReader.c RadiusReader.h
    gcc -Wall -Wvla -c RadiusReader.c

DrawArrayDriver.o: DrawArrayDriver.c Draw2DCharArray.h
    gcc -Wall -Wvla -c DrawArrayDriver.c

test1: PitPlugger
    PitPlugger $(FULL_PATH)/pit_radius_1.in $(FULL_PATH)/stone_radius_1.in \
    | diff $(FULL_PATH)/map_pit_1_stone_1.out -

.PHONY: all tar test1
```

4 PitPlugger.c

```c
//------------------------------------------------------------------------------------------------
//                                  Ex2 - PitPlugger
// General     :    close the pit by fill it with stones
// Input      :    list of the pit radiuses and list of the stones radiuses
// Process     :    Throw the stone into the pit until it closed (or fail to close since no stone left)
// Output     :    Announcement about the success/fail of the mission and detail about the pit and the
//                 stones and draw a side-look of the pit.
//
//------------------------------------------------------------------------------------------------

#include "RadiusReader.h"
#include "Draw2DCharArray.h"
#include <math.h>

#define POINTS 3
#define SUCC 0
#define FAIL -1
#define NUM_OF_ARGS 3
#define ROW(x) (x + topStone)
#define FIXED_ROW(x) (x + 1)
#define FIXED_DEPTH(x) (x + 1)
#define TOP_STONE 1
#define NO_TOP_STONE 0
#define TOP_LEVEL -1
#define FAIL_MESSEGE "Unable to open file %s.\n"
#define STAR '*'
#define HYPHEN '-'
#define DOT '.'
#define PLUS '+'
#define STOP '\0'

/**
 *-----------------------------------------------------------------------------------------------
 *
 * General         :    throw the stone into the pit
 *                      The function goes over the stones, and every times go over the pit and find the
 *                      deepest the stone can fall before it reach the current bottom or stuck in the
 *                      middle.
 *                      The algorithm stop when no more stone left or when the pit is closed.
 *                      The algorithm use an extra array of size N for the location of the thrown
 *                      stones and another 3 int's(for the number of current thrown stones, the current
 *                      depth ant the current bottom.
 *                      In the worse case the algorithm will run for all the stones(M) and each time
 *                      all over the (remaining) pit (N). each time use constant number of operations
 *                      So the complexity is O(NM).
 * Parameters     :    pitRadius[MAX_DEPTH] - an array with the pit radiuses
 *                     stonesRadius[MAX_DEPTH] - an array with the stones radiuses
 *                     stone[MAX_DEPTH] - an empty array for the thrown stones
 * Return value     :
 *
 *-----------------------------------------------------------------------------------------------
 */

void throwStone(unsigned int const pitRadius[MAX_DEPTH],
                unsigned int const stonesRadius[MAX_DEPTH],
                unsigned int stone[MAX_DEPTH],
                unsigned int const maxDepth,
                unsigned int const stoneAmount)
{
    unsigned int stoneNum = 0;
    int depth;
    int curMaxDepth = maxDepth;

    if (pitRadius[0] == 0)
    {
        curMaxDepth = 0;
    }

    while ((stoneNum < stoneAmount)  && (curMaxDepth > 0))
    {
        depth = TOP_LEVEL;
        while ((stonesRadius[stoneNum] <= pitRadius[FIXED_DEPTH(depth)]) && (depth < curMaxDepth-1))
        {
            depth++;
        }
        stone[FIXED_DEPTH(depth)] = stonesRadius[stoneNum];
        curMaxDepth = depth;
        stoneNum++;
    }

    if (curMaxDepth <= 0)
    {
        printf("Hurrah!! You have successfully plugged that pit ;)\n");
    }
    else
    {
        printf("Oy Vey!! The pit is still open, what will we do now? :(\n");
    }

    printf("This pit is %d levels deep, of which %d levels remain open.\n",
            maxDepth, (curMaxDepth >= 0 ? curMaxDepth : 0));
    printf("We had %d stones and threw %d of them into the pit.\n\n", stoneAmount, stoneNum);


}

/**
 *-----------------------------------------------------------------------------------------------
 *
 * General        :    change part of the line from 'ptrStart' to 'stop' to the given char(chr)
 * Parameters     :    ptrStart - pointer to a cell in the array
 *                      stop - the number of cells
 *                      chr - a char to draw
 * Return value    :
 *
 *-----------------------------------------------------------------------------------------------
 */
void drawLine (unsigned int const length, char const chr, char* const ptrStart)
{
    for (int i = 0; i < length; i++)
    {
        *(ptrStart + i) = chr;
    }
}

/**
 *-----------------------------------------------------------------------------
 *
 * General        :    draw the pit array (up to ROWS rows).
 * Parameters     :    pit[ROWS][COLS] - the array that represent the pit
 *                     pitRadius[MAX_DEPTH] - an array with the pit radiuses
 *                     stonesRadius[MAX_DEPTH] - an array with the stones radiuses
 * Return value    :
 *
 *-----------------------------------------------------------------------------
 */
void drawPit(char pit[ROWS][COLS], unsigned int const pitRadius[MAX_DEPTH],
            unsigned int const stone[MAX_DEPTH], unsigned int const maxDepth)
{
    unsigned int row = 0;
    unsigned int topStone = NO_TOP_STONE;
    char* ptr;
    unsigned int length;

    initializeArray(pit);

    //take care in case the stone at the top of the pit
    if (stone[0] != 0)
    {
        ptr = &pit[row][(stone[0] <= COLS ? COLS / 2 - stone[0] / 2 : 0)];
        length = (stone[0] <= COLS ? stone[0] : COLS);
        drawLine(length, HYPHEN, ptr);
        topStone = TOP_STONE; //use to fix the index of the row in the draw pit
    }

    for ( ; (row < maxDepth) && (row < ROWS); row++)
    {
        if (pitRadius[row] <= COLS)
        {
            //left wall
            ptr = &pit[ROW(row)][0];
            length = (COLS-pitRadius[row]) / 2;
            drawLine(length, STAR, ptr);
            //stone
            ptr = &pit[ROW(row)][(int)(COLS-round((COLS-pitRadius[row]) / 2.0))];
            length = (int)round((COLS - pitRadius[row]) / 2.0);
            drawLine(length, STAR, ptr);
            //right wall
            ptr = &pit[ROW(row)][COLS / 2 - stone[FIXED_ROW(row)] / 2];
            length = stone[FIXED_ROW(row)];
            drawLine(length, HYPHEN, ptr);
        }
        else
        {
            //left wall(too long)
            ptr = &pit[ROW(row)][0];
            length = POINTS;
            drawLine(length, DOT, ptr);
            //right wall(too long)
            ptr = &pit[ROW(row)][COLS - POINTS];
            drawLine(length, DOT, ptr);

            if (stone[FIXED_ROW(row)] > (COLS - 2 * POINTS))
            {
                //stone (too long)
                ptr = &pit[ROW(row)][POINTS];
                length = (COLS - 2 * POINTS);
                drawLine(length, HYPHEN, ptr);
            }
            else
            {
                //stone
                ptr = &pit[ROW(row)][COLS / 2 - stone[FIXED_ROW(row)] / 2];
                length = stone[FIXED_ROW(row)];
                drawLine(length, HYPHEN, ptr);
            }
        }
    }

    //change the pit bottom to '+' if 1 row left
    if (row < ROWS)
    {
        ptr = &pit[ROW(row)][0];
        length = COLS;
        drawLine(length, PLUS, ptr);
```

```c
    }

    //if 2 row left put '\0' to stop the drawing.
    if (row++ < ROWS)
    {
        pit[ROW(row)][0] = STOP;
    }

    drawArray(pit);
}

/**
 *------------------------------------------------------------------------------
 *
 * General        :    the main function
 *                     read the files, throw the stone to the pit and draw the pit.
 * Argument        :    pitRadius - the pit radiuses
 *                    stonesRadius - the stones radiuses
 * Return value    :    -1 iff there are problem with the file, else 0.
 *
 *------------------------------------------------------------------------------
 */
int main(int argc, char *argv[])
{
    unsigned int pit[MAX_DEPTH];
    unsigned int usedStone[MAX_DEPTH];
    unsigned int stone[MAX_DEPTH];
    char pitDraw[ROWS][COLS];
    unsigned int maxDepth;
    unsigned int stoneAmount;
    FILE *pitRadius, *stoneRadius;

    if (argc != NUM_OF_ARGS)
    {
        printf("Usage: PitPlugger <Pit Radius input file> <Stone Radius input file>\n");
        return FAIL;
    }

    if ((pitRadius = fopen(argv[1], "r")) == NULL)
    {
        printf(FAIL_MESSEGE, argv[1]);
        return FAIL;
    }

    if ((stoneRadius = fopen(argv[2], "r")) == NULL)
    {
        printf(FAIL_MESSEGE, argv[2]);
        fclose(pitRadius);
        return FAIL;
    }

    maxDepth = readVector(pitRadius, pit);
    stoneAmount = readVector(stoneRadius, stone);

    throwStone(pit, stone, usedStone, maxDepth, stoneAmount);

    drawPit(pitDraw, pit, usedStone, maxDepth);



    fclose(pitRadius);
    fclose(stoneRadius);

    return SUCC;
}
```

good_coding_style

5 RadiusReader.c

```c
//------------------------------------------------------------------------------------------------
//                                Ex2 - RadiusReader
// General    :    read the given file. each line contains a single unsigned integer
// Input      :    file - the given file
// Process    :    read the file in 2 ways. all the file into vector or one number at a time.
// Output     :    numbers from the file
//
//------------------------------------------------------------------------------------------------

#include "RadiusReader.h"
#include <stdio.h>

#define SUCC_READ 1

/**
 * Read a vector of unsigned ints from a given FILE
 * each line contains a single unsigned integer
 * @return number of lines read
 */
unsigned int readVector(FILE * file, unsigned int vec[MAX_DEPTH])
{
    unsigned int val;
    unsigned int floor = 0;

    while (readSingleUInt(file, &val) == SUCC_CODE)
    {
        vec[floor++] = val;
    }

    return floor;
}

/**
 * Read a single unsigned int from a given FILE
 * each line contains a single unsigned integer
 * @return SUCC_CODE iff successful, FAIL_CODE otherwise
 */
int readSingleUInt(FILE * file, unsigned int * val)
{
    if (fscanf(file, "%u", val) == SUCC_READ)
    {
        return SUCC_CODE;
    }
    return FAIL_CODE;
}
```