

Contents

1	Basic Test Results	2
2	Draw2DCharArray.c	3
3	Makefile	4
4	PitPlugger.c	5
5	RadiusReader.c	9

1 Basic Test Results

```
1  Running...
2  Opening tar file
3  Draw2DCharArray.c
4  Makefile
5  PitPlugger.c
6  RadiusReader.c
7  OK
8  Tar extracted O.K.
9  Checking files...
10 OK
11 Making sure files are not empty...
12 OK
13 Importing files
14 OK
15 Compilation check...
16 Compiling...
17 gcc -Wall -Wvla -c -std=c99 Draw2DCharArray.c
18 gcc -Wall -Wvla -c DrawArrayDriver.c
19 gcc -Wall -Wvla Draw2DCharArray.o DrawArrayDriver.o -o DrawArrayDriver
20 OK
21 Compiling...
22 gcc -Wall -Wvla -c -std=c99 PitPlugger.c
23 gcc -Wall -Wvla -c RadiusReader.c
24 gcc -Wall -Wvla -lm PitPlugger.o Draw2DCharArray.o RadiusReader.o -o PitPlugger
25 OK
26 Compilation went without errors, BUT you must check to see if you got warnings!!!
27 Check some inputs:
28 Running test...
29 OK
30 OK
31
32 =====
33 = Checking coding style =
34 =====
35 ** Total Violated Rules      : 0
36 ** Total Errors Occurs      : 0
37 ** Total Violated Files Count: 0
```

2 Draw2DCharArray.c

```
1  //-----
2  //                                     Ex2 - Draw2DCharArray
3  // General      :   Initialize the given char array to be filled with space and print it to the screen
4  // Input       :   arr - 2D char array
5  // Process      :   run over the array and fill/print
6  // Output      :   print of the array
7  //
8  //-----
9  #include "Draw2DCharArray.h"
10 #include <stdio.h>
11
12 #define WHITE_SPACE ' '
13
14 /**
15  * Initialize the given char array to be filled with space ( ' ') characters.
16  */
17 void initializeArray(char arr[ROWS][COLS])
18 {
19     for (int i = 0; i < ROWS; i++)
20     {
21         for (int j = 0; j < COLS; j++)
22         {
23             arr[i][j] = WHITE_SPACE;
24         }
25     }
26 }
27
28 /**
29  * Draw the char array on the screen.
30  * After each row go down a line ('\n').
31  * Stop drawing when a nul character ('\0') is met.
32  */
33 void drawArray(char arr[ROWS][COLS])
34 {
35     for (int i = 0; i < ROWS; i++)
36     {
37         for (int j = 0; j < COLS; j++)
38         {
39             if (arr[i][j] == '\0')
40             {
41                 return;
42             }
43             printf("%c", arr[i][j]);
44         }
45         printf("\n");
46     }
47 }
```

3 Makefile

```
1  FULL_PATH = /cs/course/2014/slabc/public/ex2/inputOutput
2
3  all: PitPlugger DrawArrayDriver
4
5  tar: Draw2DCharArray.c RadiusReader.c PitPlugger.c Makefile
6      tar cvf ex2.tar Draw2DCharArray.c RadiusReader.c PitPlugger.c Makefile
7
8  PitPlugger: PitPlugger.o Draw2DCharArray.o RadiusReader.o
9      gcc -Wall -Wvla -lm PitPlugger.o Draw2DCharArray.o RadiusReader.o -o PitPlugger
10
11 DrawArrayDriver: Draw2DCharArray.o DrawArrayDriver.o
12     gcc -Wall -Wvla Draw2DCharArray.o DrawArrayDriver.o -o DrawArrayDriver
13
14 PitPlugger.o: PitPlugger.c Draw2DCharArray.h RadiusReader.h
15     gcc -Wall -Wvla -c -std=c99 PitPlugger.c
16
17 Draw2DCharArray.o: Draw2DCharArray.c Draw2DCharArray.h
18     gcc -Wall -Wvla -c -std=c99 Draw2DCharArray.c
19
20 RadiusReader.o: RadiusReader.c RadiusReader.h
21     gcc -Wall -Wvla -c RadiusReader.c
22
23 DrawArrayDriver.o: DrawArrayDriver.c Draw2DCharArray.h
24     gcc -Wall -Wvla -c DrawArrayDriver.c
25
26 test1: PitPlugger
27     PitPlugger $(FULL_PATH)/pit_radius_1.in $(FULL_PATH)/stone_radius_1.in \
28     | diff $(FULL_PATH)/map_pit_1_stone_1.out -
29
30 .PHONY: all tar test1
```

4 PitPlugger.c

```
1  //-----
2  //                                     Ex2 - PitPlugger
3  // General      :   close the pit by fill it with stones
4  // Input       :   list of the pit radiuses and list of the stones radiuses
5  // Process      :   Throw the stone into the pit until it closed (or fail to close since no stone left)
6  // Output      :   Announcement about the success/fail of the mission and detail about the pit and the
7  //               :   stones and draw a side-look of the pit.
8  //
9  //-----
10
11 #include "RadiusReader.h"
12 #include "Draw2DCharArray.h"
13 #include <math.h>
14
15 #define POINTS 3
16 #define SUCC 0
17 #define FAIL -1
18 #define NUM_OF_ARGS 3
19 #define ROW(x) (x + topStone)
20 #define FIXED_ROW(x) (x + 1)
21 #define FIXED_DEPTH(x) (x + 1)
22 #define TOP_STONE 1
23 #define NO_TOP_STONE 0
24 #define TOP_LEVEL -1
25 #define FAIL_MESSEGE "Unable to open file %s.\n"
26 #define STAR '*'
27 #define HYPHEN '-'
28 #define DOT '.'
29 #define PLUS '+'
30 #define STOP '\0'
31
32 /**
33  *-----
34  *
35  * General      :   throw the stone into the pit
36  *               :   The function goes over the stones, and every times go over the pit and find the
37  *               :   deepest the stone can fall before it reach the current bottom or stuck in the
38  *               :   middle.
39  *               :   The algorithm stop when no more stone left or when the pit is closed.
40  *               :   The algorithm use an extra array of size N for the location of the thrown
41  *               :   stones and another 3 int's(for the number of current throw stones, the current
42  *               :   depth ant the current bottom.
43  *               :   In the worse case the algorithm will run for all the stones(M) and each time
44  *               :   all over the (remaining) pit (N). each time use constant number of operations
45  *               :   So the complexity is O(NM).
46  * Parameters    :   pitRadius[MAX_DEPTH] - an array with the pit radiuses
47  *               :   stonesRadius[MAX_DEPTH] - an array with the stones radiuses
48  *               :   stone[MAX_DEPTH] - an empty array for the thrown stones
49  * Return value   :
50  *
51  *-----
52  */
53
54 void throwStone(unsigned int const pitRadius[MAX_DEPTH],
55                unsigned int const stonesRadius[MAX_DEPTH],
56                unsigned int stone[MAX_DEPTH],
57                unsigned int const maxDepth,
58                unsigned int const stoneAmount)
59 {
```

```

60     unsigned int stoneNum = 0;
61     int depth;
62     int curMaxDepth = maxDepth;
63
64     if (pitRadius[0] == 0)
65     {
66         curMaxDepth = 0;
67     }
68
69     while ((stoneNum < stoneAmount) && (curMaxDepth > 0))
70     {
71         depth = TOP_LEVEL;
72         while ((stonesRadius[stoneNum] <= pitRadius[FIXED_DEPTH(depth)]) && (depth < curMaxDepth-1))
73         {
74             depth++;
75         }
76         stone[FIXED_DEPTH(depth)] = stonesRadius[stoneNum];
77         curMaxDepth = depth;
78         stoneNum++;
79     }
80
81     if (curMaxDepth <= 0)
82     {
83         printf("Hurrah!! You have successfully plugged that pit ;)\n");
84     }
85     else
86     {
87         printf("Oy Vey!! The pit is still open, what will we do now? :(\n");
88     }
89
90     printf("This pit is %d levels deep, of which %d levels remain open.\n",
91           maxDepth, (curMaxDepth >= 0 ? curMaxDepth : 0));
92     printf("We had %d stones and threw %d of them into the pit.\n\n", stoneAmount, stoneNum);
93
94 }
95
96 /**
97  *-----
98  *
99  * General      :   change part of the line from 'ptrStart' to 'stop' to the given char(chr)
100 * Parameters   :   ptrStart - pointer to a cell in the array
101 *               :   stop - the number of cells
102 *               :   chr - a char to draw
103 * Return value :
104 *
105 *-----
106 */
107 void drawLine (unsigned int const length, char const chr, char* const ptrStart)
108 {
109     for (int i = 0; i < length; i++)
110     {
111         *(ptrStart + i) = chr;
112     }
113 }
114
115 /**
116  *-----
117  *
118  * General      :   draw the pit array (up to ROWS rows).
119 * Parameters   :   pit[ROWS][COLS] - the array that represent the pit
120 *               :   pitRadius[MAX_DEPTH] - an array with the pit radiuses
121 *               :   stonesRadius[MAX_DEPTH] - an array with the stones radiuses
122 * Return value :
123 *
124 *-----
125 */
126 void drawPit(char pit[ROWS][COLS], unsigned int const pitRadius[MAX_DEPTH],

```

```

128         unsigned int const stone[MAX_DEPTH], unsigned int const maxDepth)
129     {
130         unsigned int row = 0;
131         unsigned int topStone = NO_TOP_STONE;
132         char* ptr;
133         unsigned int length;
134
135         initializeArray(pit);
136
137         //take care in case the stone at the top of the pit
138         if (stone[0] != 0)
139         {
140             ptr = &pit[row][(stone[0] <= COLS ? COLS / 2 - stone[0] / 2 : 0)];
141             length = (stone[0] <= COLS ? stone[0] : COLS);
142             drawLine(length, HYPHEN, ptr);
143             topStone = TOP_STONE; //use to fix the index of the row in the draw pit
144         }
145
146         for ( ; (row < maxDepth) && (row < ROWS); row++)
147         {
148             if (pitRadius[row] <= COLS)
149             {
150                 //left wall
151                 ptr = &pit[ROW(row)][0];
152                 length = (COLS-pitRadius[row]) / 2;
153                 drawLine(length, STAR, ptr);
154                 //stone
155                 ptr = &pit[ROW(row)][(int)(COLS-round((COLS-pitRadius[row]) / 2.0))];
156                 length = (int)round((COLS - pitRadius[row]) / 2.0);
157                 drawLine(length, STAR, ptr);
158                 //right wall
159                 ptr = &pit[ROW(row)][COLS / 2 - stone[FIXED_ROW(row)] / 2];
160                 length = stone[FIXED_ROW(row)];
161                 drawLine(length, HYPHEN, ptr);
162             }
163             else
164             {
165                 //left wall(too long)
166                 ptr = &pit[ROW(row)][0];
167                 length = POINTS;
168                 drawLine(length, DOT, ptr);
169                 //right wall(too long)
170                 ptr = &pit[ROW(row)][COLS - POINTS];
171                 drawLine(length, DOT, ptr);
172
173                 if (stone[FIXED_ROW(row)] > (COLS - 2 * POINTS))
174                 {
175                     //stone (too long)
176                     ptr = &pit[ROW(row)][POINTS];
177                     length = (COLS - 2 * POINTS);
178                     drawLine(length, HYPHEN, ptr);
179                 }
180                 else
181                 {
182                     //stone
183                     ptr = &pit[ROW(row)][COLS / 2 - stone[FIXED_ROW(row)] / 2];
184                     length = stone[FIXED_ROW(row)];
185                     drawLine(length, HYPHEN, ptr);
186                 }
187             }
188         }
189
190         //change the pit bottom to '+' if 1 row left
191         if (row < ROWS)
192         {
193             ptr = &pit[ROW(row)][0];
194             length = COLS;
195             drawLine(length, PLUS, ptr);

```

```

196     }
197
198     //if 2 row left put '\0' to stop the drawing.
199     if (row++ < ROWS)
200     {
201         pit[ROW(row)][0] = STOP;
202     }
203
204     drawArray(pit);
205 }
206
207 /**
208  *-----
209  *
210  * General      :    the main function
211  *               :    read the files, throw the stone to the pit and draw the pit.
212  * Argument     :    pitRadius - the pit radiuses
213  *               :    stonesRadius - the stones radiuses
214  * Return value  :    -1 iff there are problem with the file, else 0.
215  *
216  *-----
217  */
218 int main(int argc, char *argv[])
219 {
220     unsigned int pit[MAX_DEPTH];
221     unsigned int usedStone[MAX_DEPTH];
222     unsigned int stone[MAX_DEPTH];
223     char pitDraw[ROWS][COLS];
224     unsigned int maxDepth;
225     unsigned int stoneAmount;
226     FILE *pitRadius, *stoneRadius;
227
228     if (argc != NUM_OF_ARGS)
229     {
230         printf("Usage: PitPluggger <Pit Radius input file> <Stone Radius input file>\n");
231         return FAIL;
232     }
233
234     if ((pitRadius = fopen(argv[1], "r")) == NULL)
235     {
236         printf(FAIL_MESSEGE, argv[1]);
237         return FAIL;
238     }
239
240     if ((stoneRadius = fopen(argv[2], "r")) == NULL)
241     {
242         printf(FAIL_MESSEGE, argv[2]);
243         fclose(pitRadius);
244         return FAIL;
245     }
246
247     maxDepth = readVector(pitRadius, pit);
248     stoneAmount = readVector(stoneRadius, stone);
249
250     throwStone(pit, stone, usedStone, maxDepth, stoneAmount);
251
252     drawPit(pitDraw, pit, usedStone, maxDepth);
253
254
255     fclose(pitRadius);
256     fclose(stoneRadius);
257
258     return SUCC;
259 }
260

```

good_coding_style

5 RadiusReader.c

```
1  //-----
2  //                                     Ex2 - RadiusReader
3  // General      :   read the given file. each line contains a single unsigned integer
4  // Input       :   file - the given file
5  // Process      :   read the file in 2 ways. all the file into vector or one number at a time.
6  // Output      :   numbers from the file
7  //
8  //-----
9
10 #include "RadiusReader.h"
11 #include <stdio.h>
12
13 #define SUCC_READ 1
14
15 /**
16  * Read a vector of unsigned ints from a given FILE
17  * each line contains a single unsigned integer
18  * @return number of lines read
19  */
20 unsigned int readVector(FILE * file, unsigned int vec[MAX_DEPTH])
21 {
22     unsigned int val;
23     unsigned int floor = 0;
24
25     while (readSingleUInt(file, &val) == SUCC_CODE)
26     {
27         vec[floor++] = val;
28     }
29
30     return floor;
31 }
32
33 /**
34  * Read a single unsigned int from a given FILE
35  * each line contains a single unsigned integer
36  * @return SUCC_CODE iff successful, FAIL_CODE otherwise
37  */
38 int readSingleUInt(FILE * file, unsigned int * val)
39 {
40     if (fscanf(file, "%u", val) == SUCC_READ)
41     {
42         return SUCC_CODE;
43     }
44     return FAIL_CODE;
45 }
```