



- 198 Removals + 195 Additions

6 2 c++

```

1 /**
2 Name: Phoenix Bishea
3 Eid: pnb338
4
5 Q1: Why are the arrays passed to both print life functions c
  onst?
6
7 The arrays passed to print life are not being directly m
  odified.
8 Since the value never changes it can be declared const.
9
10 Q2: Why are the arrays passed to both play life functions no
  t const?
11
12 The arrays passed to play life are not const because the
  y are modified.
13 Play life changes the cells in the array every iteration
  depending
14 on the rules of the game of life.
15
16 Q3: If we did change the arrays passed to play life functio
  ns to make them const,
17 what else would have to happen to make play life work?
18
19 Under these circumstances an additional array would have
  to be created and returned
20 from play life to main to make play life work. The array
  passed to play life could not
21 be modified.
22 */
23
24 #include <iostream>
25
26 using namespace std;
27
28 /// global constant for fixed rows and cols of the 2D array
29 const int NUM_ROWS = 10;
30
31 const int NUM_COLS = 10;
32 const int MIN = 0;
33 const int MAX = 9;
34
35 /** function declarations
36 you will need to write the definitions of these function
37 s below.
38 DO NOT MODIFY the declarations.
39 You may create your own functions, but you must use thes
40 e.
41 */
42
43 /** 2d array functions, notice that for 2d arrays the second
44 size

```

```

1 /**
2 Name: Phoenix Bishea
3 Eid: pnb338
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

41     needs to be fixed, or more precisely it needs to be a co
nst.
    if you typed a literal, like 5, it would be a literal co
42 nstant,
43     which also works.
44 */
45 void printLife2DArray(const bool[][NUM_COLS]);
46 void playLife2DArray(bool[][NUM_COLS]);
47
48 /** Initializes the bool array created in main to the input
array.
49     Note: '*' is true and '.' is false.
50 */
51 void initializeLife2DArray(bool[][NUM_COLS]);
52 int getNeighbors(const bool[][NUM_COLS], int row, int col);
53
54
55 int main() {
56
57     /// read in the number of iterations to run
58
59     /** make a 2d bool array with the number of rows and col
s
60
61     Some input examples and explanation of game of life
    Look at http://en.wikipedia.org/wiki/Conway%27s\_Game\_of\_
62 Life#Examples\_of\_patterns
63 */
64
65     /** print out what game we are playing */
    // cout << "Game Of Life rows=" << /* n rows */ << " cols
66 =" << ncols <<
    //      " iterations=" << /* number of iterations */ <<
67 endl;
68
69     /**
70     start your code
71     */
72
73     // reads in the number of iterations to run
74     int iterations;
75     cin >> iterations;
76     cin.ignore(); // consumes the newline character at the e
nd of the first line
77
78     // initializes a 2d bool array with the number of rows a
nd cols
79     bool arr[NUM_ROWS][NUM_COLS];
80     initializeLife2DArray(arr);
81
82     // prints the game we are playing
83     cout << "Game Of Life rows=" << NUM_ROWS
<< " cols=" << NUM_COLS <<
84         " iterations=" << iterations << endl;
85

```

```

15
16 void showGrid(const bool[][columnnumber]);
17 void startGrid(bool[][columnnumber]);
18 void initializeGrid(bool[][columnnumber]);
19 void getGridCopy(bool grid[][columnnumber]); //hard copies c
reated grid for game use
20 int returnNeighbor(const bool[][columnnumber], int r, int
c);
21
22
23 int main()
24 {
25
26     // reads in the number of numIterate to run
27     int numIterate;
28     cin >> numIterate;
29
30     cin.ignore(); // consumes the newline character at the e
nd of the first line
31
32     // initializes a 2d bool array with the number of rows a
nd cols
33     bool arr[rownumber][columnnumber];
34     initializeGrid(arr);
35
36     // prints the game we are playing
37     cout << "Game Of Life rows=" << rownumber
<< " cols=" << columnnumber <<
    " numIterate=" << numIterate << endl;

```

```

86 // prints the initial board
87 printLife2DArray(arr);
88
89 // completes x iterations
90 for (int i = 0; i < iterations; i++) {
91     cout << endl;
92     playLife2DArray(arr);
93     printLife2DArray(arr);
94 }
95
96 return 0; /// return a ok
97 }
98
99 void initializeLife2DArray(bool arr[][NUM_COLS]) {

```

```

38 // prints the initial board
39 showGrid(arr);
40
41 // completes x numIterate
42 for (int i = 0; i < numIterate; i++)
43 {
44     cout << endl;
45     startGrid(arr);
46     showGrid(arr);
47 }
48
49 return 0; /// return a ok
50 }
51
52 void showGrid(const bool arr[][columnnumber])
53 {
54     // converts the bool array into the board of life
55     // '*' is true and '.' is false
56     for (const i = 0; i < rownumber; i++)
57     {
58         for (const j = 0; j < columnnumber; j++)
59         {
60             if (arr[i][j])
61             {
62                 cout << '*';
63             } else
64             {
65                 cout << '.';
66             }
67         }
68         cout << endl;
69     }
70 }
71
72 void getgrid(char line[], bool grid[][columnnumber])
73 {
74     //Local Declarations
75     char c;
76     int iterationsC = 0;
77     while (iterationsC < columnnumber)
78     {
79         c = line[iterationsC];
80         cout << c;
81         if ((isNotNullChar(c)) && (c == '.') || (c
82             != 32))
83         {
84             grid[iterationsR][iterationsC] = fa
85 lse;
86         }
87         else if ((c == '*') && ((isNotNullChar(c)) |
88             (c != 32)))
89         {
90             grid[iterationsR][iterationsC] = tru
91 e;
92         }
93         iterationsC++;
94     }
95     iterationsR++;
96     cout << endl;
97     if (iterationsR >= 10)
98     {
99

```

```

100
101 char line[NUM_COLS + 1]; // line to take in row input
102
103 // gets the first line
104 cin.getline(line, NUM_COLS + 1);
105
106 // sets values to true and false in the bool array
107 // '*' is true and '.' is false
108 for (size_t i = 0; i < NUM_ROWS; i++) {
109     for (size_t j = 0; j < NUM_COLS; j++) {
110         if (line[j] == '*') {
111             arr[i][j] = 1;
112         } else {
113             arr[i][j] = 0;
114         }
115     }
116     cin.getline(line, NUM_COLS + 1);
117 }
118 }
119
120 void printLife2DArray(const bool arr[][NUM_COLS]) {
121
122     // converts the bool array into the board of life
123     // '*' is true and '.' is false
124     for (size_t i = 0; i < NUM_ROWS; i++) {
125         for (size_t j = 0; j < NUM_COLS; j++) {
126             if (arr[i][j]) {
127                 cout << '*';
128             } else {
129                 cout << '.';
130             }
131         }
132         cout << endl;
133     }
134 }
135
136 void playLife2DArray(bool board[][NUM_COLS]) {

```

137

```

        getGridCopy(grid);
96     }
97 }
98
99
100 void initializeGrid(bool arr[][columnnumber])
101 {
102     char line[columnnumber + 1]; // line to take in r input
103     // gets the first line
104     cin.getline(line, columnnumber + 1);
105
106     // sets values to true and false in the bool array
107     // '*' is true and '.' is false
108     for (const i = 0; i < rownumber; i++)
109     {
110         for (const j = 0; j < columnnumber; j++)
111         {
112             if (line[j] == '*')
113             {
114                 arr[i][j] = 1;
115             } else
116             {
117                 arr[i][j] = 0;
118             }
119             cin.getline(line, columnnumber + 1);
120         }
121     }
122
123 bool isNotNullChar(char c)
124 {
125     if (c == '\0')
126     {
127         return 0
128     }
129     else
130     {
131         return 1
132     }
133 }
134
135 void getGridCopy(bool grid[][NUM_COLS])
136 {
137     //Declarations
138     iterationsR = 0, iterationsC = 0;
139
140     while (iterationsR < NUM_ROWS) {
141         while (iterationsC < NUM_COLS)
142         {
143             gridCopy[iterationsR][iterationsC] =
144             grid[iterationsR][iterationsC];
145             iterationsC++;
146         }
147         iterationsR++; iterationsC = 0;
148     }
149

```

```

138 // creates a temporary board and sets it equal to the ga
me board
139 bool tempBoard[NUM_ROWS][NUM_COLS];
140 for (size_t i = 0; i < NUM_ROWS; i++) {
141     for (size_t j = 0; j < NUM_COLS; j++) {
142         tempBoard[i][j] = board[i][j];
143     }
144 }
145 // updates cells using the game of life rules
146 for (size_t i = 0; i < NUM_ROWS; i++) {
147     for (size_t j = 0; j < NUM_COLS; j++) {
148         // gets a cell's neighbors and status
149         int neighbors = getNeighbors(tempBoard, i, j);
150         bool alive = tempBoard[i][j];
151         // changes the cell's condition depending on its neighbors
152         if (alive && (neighbors < 2 || neighbors > 3)) {
153             board[i][j] = 0;
154         } else if (!alive && (neighbors == 3)) {
155             board[i][j] = 1;
156         }
157     }
158 }
159 }
160 }
161 }
162 }
163
164 int getNeighbors(const bool board[][NUM_COLS], int row, int
col) {
165     int neighbors = 0; // initializes starting neighbors
166     // sets up the boundaries
167     bool colLessMax = (col + 1) <= MAX;
168     bool rowLessMax = (row + 1) <= MAX;
169     bool colGreaterMin = (col - 1) >= MIN;
170     bool rowGreaterMin = (row - 1) >= MIN;
171     // if the operation is within bounds, complete it and tally
    living neighbors
172     if (colLessMax
173         && board[row][col + 1]) neighbors++;
174     // increment if a living neighbor is found
175     if (colGreaterMin
176         && board[row][col - 1]) neighbors++;
177     if (rowLessMax
178         && board[row + 1][col]) neighbors++;
179     if (rowGreaterMin
180         && board[row - 1][col]) neighbors++;
181     if (colLessMax
182         && board[row][col + 1]) neighbors++;
183     if (rowGreaterMin
184         && board[row - 1][col]) neighbors++;

```

```

150 void startGrid(bool board[][columnnumber])
151 {
152     bool tempBoard[rownumber][columnnumber];
153     for (const i = 0; i < rownumber; i++)
154     {
155         for (const j = 0; j < columnnumber; j++)
156         {
157             tempBoard[i][j] = board[i][j];
158         }
159     }
160     for (const i = 0; i < rownumber; i++)
161     {
162         for (const j = 0; j < columnnumber; j++)
163         {
164             int neighbors = returnNeighbor(tempBoard, i, j);
165             bool alive = tempBoard[i][j];
166             if (alive && (neighbors < 2 || neighbors > 3))
167             {
168                 board[i][j] = 0;
169             } else if (!alive && (neighbors == 3))
170             {
171                 board[i][j] = 1;
172             }
173         }
174     }
175 }
176
177 int returnNeighbor(const bool board[][columnnumber], int r,
int c)
178 {
179     int neighbors = 0; // initializes starting neighbors
180     // sets up the boundaries
181     int a = c+1, b = r+1, d = c-1, e = r-1;
182     bool ifcolumnmax = (a) <= maximum;
183     bool ifrowmax = (b) <= maximum;
184     bool ifcolmin = (d) >= minimum;
185     bool ifrowmin = (e) >= minimum;
186     if (ifcolumnmax && board[r][c + 1]) neighbors++;
187     else if (ifcolmin && board[r][c - 1]) neighbors++;
188     else if (ifrowmax && board[r + 1][c]) neighbors++;
189     else if (ifrowmin && board[r - 1][c]) neighbors++;
190     else if (ifrowmax && ifcolumnmax && board[r + 1][c
+ 1]) neighbors++;
191     else if (ifrowmin && ifcolumnmax && board[r - 1][c
+ 1]) neighbors++;
192     else if (ifrowmax && ifcolmin && board[r + 1][c
- 1]) neighbors++;
193     else (ifrowmin&& ifcolmin && board[r - 1][c - 1])
neighbors++;

```

```
185     && board[row - 1][col]) neighbors++;
186
187     if (rowLessMax && colLessMax
188         && board[row + 1][col + 1]) neighbors++;
189
190     if (rowGreaterMin && colLessMax
191         && board[row - 1][col + 1]) neighbors++;
192
193     if (rowLessMax && colGreaterMin
194         && board[row + 1][col - 1]) neighbors++;
195
196     if (rowGreaterMin && colGreaterMin
197         && board[row - 1][col - 1]) neighbors++;
198
199     return neighbors;
200 }
```

```
194     return neighbors;
195 }
```