

```
1 # 4-loops
2
3 - `Alt + 6`: Problems on the status bar
4 - `SonarLint` on the status bar
5
6 ## `game-of-life.c`
7
8 - play with it
9   - [wiki](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)
10  - [Demo](https://playgameoflife.com/)
11  - [Gosper_glider_gun](https://playgameoflife.com/lexicon/
    Gosper_glider_gun)
12  - [Life Lexicon Home Page](https://conwaylife.com/ref/lexicon/
    lex_home.htm)
13 - 2D-array
14   - initialization (Section 8.2.1)
15     - row-major
16     - row by row
17     - indicator
18 - extension of board
19 - how many boards?
20 - one round
21 - multiple rounds
22 - pause
23 - screen clear
24 - [ ] try a new board?
25   - [Life Lexicon Home Page](https://conwaylife.com/ref/lexicon/
    lex_home.htm)
26
27 # `merge.c`
28
29 - examples
30 - for `merge-sort.c` later
31
32 # `bubble-sort.c`
33
34 - generating test cases
35 - timing
36 - `swapped`
37   - TAOCP Vol. 3 (P109)
```

```
1 //
2 // Created by hengxin on 10/19/22.
3 //
4
5 #include <stdio.h>
6 #define LEN_L 5
7 #define LEN_R 6
8
9 int L[LEN_L] = {1, 3, 5, 7, 9};
10 int R[LEN_R] = {0, 2, 4, 6, 8, 10};
11
12 int main() {
13     int l = 0;
14     int r = 0;
15
16     while (l < LEN_L && r < LEN_R) {
17         if (L[l] <= R[r]) {
18             printf("%d ", L[l]);
19             l++;
20         } else { // L[l] > R[r]
21             printf("%d ", R[r]);
22             r++;
23         }
24     }
25
26     while (l < LEN_L) {
27         printf("%d ", L[l]);
28         l++;
29     }
30
31     while (r < LEN_R) {
32         printf("%d ", R[r]);
33         r++;
34     }
35
36     return 0;
37 }
```

```

1 //
2 // Created by hengxin on 10/19/22.
3 // Run it with "Terminal"
4 //
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <unistd.h>
9
10 #define SIZE 6
11 //const int board[SIZE][SIZE] = {
12 //    {0},
13 //    {0, 1, 1, 0, 0, 0},
14 //    {0, 1, 1, 0, 0, 0},
15 //    {0, 0, 0, 1, 1, 0},
16 //    {0, 0, 0, 1, 1, 0},
17 //    {0}
18 //};
19
20 const int board[SIZE][SIZE] = {
21     [1][1] = 1, [1][2] = 1,
22     [2][1] = 1, [2][2] = 1,
23     [3][3] = 1, [3][4] = 1,
24     [4][3] = 1, [4][4] = 1
25 };
26
27 int main() {
28     int old_board[SIZE + 2][SIZE + 2];
29     for (int row = 0; row < SIZE + 2; row++) {
30         for (int col = 0; col < SIZE + 2; col++) {
31             if (row == 0 || row == SIZE + 1 || col == 0 || col == SIZE + 1
32 ) {
33                 old_board[row][col] = 0;
34             } else {
35                 old_board[row][col] = board[row - 1][col - 1];
36             }
37         }
38     }
39     // print the original board
40     for (int row = 1; row <= SIZE + 1; row++) {
41         for (int col = 1; col <= SIZE + 1; col++) {
42             printf("%c ", old_board[row][col] ? '*' : ' ');
43         }
44         printf("\n");
45     }
46
47     // clear the screen
48     system("clear");
49
50     int new_board[SIZE + 2][SIZE + 2] = {0};
51
52     for (int round = 1; round < 10; round++) {

```

```

53     for (int row = 1; row <= SIZE; row++) {
54         for (int col = 1; col <= SIZE; col++) {
55             // count the number of neighbours of old_board[row][col]
56             int neighbours =
57                 old_board[row - 1][col - 1] +
58                 old_board[row - 1][col] +
59                 old_board[row - 1][col + 1] +
60                 old_board[row][col - 1] +
61                 old_board[row][col + 1] +
62                 old_board[row + 1][col - 1] +
63                 old_board[row + 1][col] +
64                 old_board[row + 1][col + 1];
65
66             // evaluate the new board
67             if (old_board[row][col]) { // old_board[row][col] is alive
68                 new_board[row][col] = (neighbours == 2 || neighbours == 3);
69             } else { // old_board[row][col] is dead
70                 new_board[row][col] = (neighbours == 3);
71             }
72         }
73     }
74
75     // print the new board
76     for (int row = 1; row <= SIZE + 1; row++) {
77         for (int col = 1; col <= SIZE + 1; col++) {
78             printf("%c ", new_board[row][col] ? '*' : ' ');
79         }
80         printf("\n");
81     }
82
83     // sleep for a while
84     // Linux: #include <unistd.h>
85     // Windows: #include <windows.h>: Sleep(ms)
86     sleep(1);
87
88     // clear the screen
89     // Windows: #include <conio.h> clrscr();
90     system("clear");
91
92     // start the next round
93     for (int row = 0; row < SIZE + 2; row++) {
94         for (int col = 0; col < SIZE + 2; col++) {
95             old_board[row][col] = new_board[row][col];
96         }
97     }
98 }
99
100 return 0;
101 }

```

```
1 //
2 // Created by hengxin on 10/19/22.
3 //
4 // For the usage of clock(),
5 // please refer to https://legacy.cplusplus.com/reference/ctime/clock/.
6 //
7
8 #include <stdlib.h>
9 #include <time.h>
10 #include <stdio.h>
11 #include <stdbool.h>
12
13 #define LEN 1000000
14 int numbers[LEN] = {0};
15
16 int main() {
17     srand(time(NULL));
18     for (int i = 0; i < LEN; i++) {
19         numbers[i] = rand() % LEN;
20     }
21
22     // set a clock
23     clock_t start = clock();
24
25     bool swapped = true;
26     for (int i = 0; i < LEN && swapped; i++) {
27         swapped = false;
28         for (int j = 0; j < LEN - 1 - i; j++) {
29             if (numbers[j] > numbers[j + 1]) {
30                 int temp = numbers[j];
31                 numbers[j] = numbers[j + 1];
32                 numbers[j + 1] = temp;
33                 swapped = true;
34             }
35         }
36     }
37
38     // record the end clock
39     clock_t end = clock();
40
41     for (int i = 0; i < LEN; i++) {
42         printf("%d ", numbers[i]);
43     }
44
45     long sec = (end - start) / CLOCKS_PER_SEC;
46     printf("Sorting %d numbers in %ld seconds.\n", LEN, sec);
47
48     return 0;
49 }
```