Εργασία 5: Πίνακες - Δείκτες - Αρχεία

Χρήστος Μαργιώλης - Εργαστηριακό τμήμα 9

Ιανουάριος 2020

Περιεχόμενα

1 Δομή προγραμμάτων και οδηγίες εκτέλεσης

1.1 Εκτέλεση από Linux

```
$ cd path-to-program
$ make
$ make run
$ make run ARGS=txt/data.txt #fcombinations ONLY
$ make clean #optional
```

2 combinations - συνδυασμοί

2.1 main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "combinations.h"
#include "arrhandler.h"


int main(int argc, char **argv)
{
    int N, K;
    int *arr;
    int x1, x2, y1, y2;

    N = get_n();
    arr = fill_array(N);
    quicksort(arr, 0, N-1);
    //printarray(arr, N);

    x_pair(&x1, &x2);
    y_pair(&y1, &y2);

    combinations(arr, x1, x2, y1, y2);

    free(arr);

    return 0;
}
```

2.2 combinations.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include "combinations.h"

#define COMBSN 6


int get_n()
{
    int N;

    do
    {
        printf("N (6 < N <= 49): ");
        scanf("%d", &N);
    } while (N <= 6 || N > 49);

    return N;
}
```

```c


void x_pair(int *x1, int *x2)
{
    do
    {
        printf("x1: ");
        scanf("%d", x1);
        printf("x2: ");
        scanf("%d", x2);
    } while (*x1 < 0 || *x1 > *x2 || *x2 > 6);
}


void y_pair(int *y1, int *y2)
{
    do
    {
        printf("y1: ");
        scanf("%d", y1);
        printf("y2: ");
        scanf("%d", y2);
    } while (*y1 < 21 || *y1 > *y2 || *y2 > 279);
}


void combinations(int *arr, int x1, int x2, int y1, int y2)
{
    int i, j, k, l, m, n;

    for (i = 0; i < COMBSN-5; i++)
        for (j = i+1; j < COMBSN-4; j++)
            for (k = j+1; k < COMBSN-3; k++)
                for (l = k+1; l < COMBSN-2; l++)
                    for (m = l+1; m < COMBSN-1; m++)
                        for (n = m+1; n < COMBSN; n++)
                        {
                            printf("%d %d %d %d %d %d", *(
    arr + i), *(arr + j), *(arr + k), *(arr + l), *(arr + m),
     *(arr + n));
                            printf("\n");
                        }
}


int combinations_count(int N)
{
    return factorial(N) / (factorial(COMBSN) * factorial(N -
     COMBSN));
}
```

```
69
70
71 int factorial(int num)
72 {
73     int i, fac;
74     for (i = 1, fac = 1; i <= num; i++) fac *= i;
75     return fac;
76 }
```

2.3 arrhandler.c

```
1  #include <stdlib.h>
2  #include "arrhandler.h"
3  #include "ccolors.h"
4
5  #define COMBSN 6
6
7
8  int *fill_array(int N)
9  {
10     int *arr, num, i = 0;
11
12     arr = (int *)malloc(N * sizeof(int));
13
14     if (arr == NULL)
15     {
16         set_color(BOLD_RED);
17         printf("Error! Not enough memory, exiting...\n");
18         set_color(STANDARD);
19         exit(EXIT_FAILURE);
20     }
21     else
22     {
23         do
24         {
25             printf("arr[%d]: ", i);
26             scanf("%d", &num);
27
28             if (num >= 1 && num <= 49)
29             {
30                 if (i == 0) { *(arr + i) = num; i++; }
31                 else
32                 {
33                     if (!exists_in_array(arr, N, num)) { *(
    arr + i) = num; i++; }
34                     else printf("Give a different number.\n"
    );
35                 }
36             }
37             else printf("Give a number in [1, 49].\n");
```

```
38          } while (i < N);
39      }
40
41      return arr;
42  }
43
44
45  bool exists_in_array(int *arr, int N, int num)
46  {
47      int *arrEnd = arr + (N - 1);
48      while (arr <= arrEnd && *arr != num) arr++;
49      return (arr <= arrEnd) ? true : false;
50  }
51
52
53  void quicksort(int *arr, int low, int high)
54  {
55      if (low < high)
56      {
57          int partIndex = partition(arr, low, high);
58          quicksort(arr, low, partIndex - 1);
59          quicksort(arr, partIndex + 1, high);
60      }
61  }
62
63
64  int partition(int *arr, int low, int high)
65  {
66      int pivot = *(arr + high);
67      int i = (low - 1), j;
68
69      for (j = low; j <= high - 1; j++)
70          if (*(arr + j) < pivot)
71              swap(arr + ++i, arr + j);
72
73      swap(arr + (i + 1), arr + high);
74      return (i + 1);
75  }
76
77
78  void swap(int *a, int *b)
79  {
80      int temp = *a;
81      *a = *b;
82      *b = temp;
83  }
84
85
86  void printarray(int *arr, int N)
87  {
```

```
88      for (int i = 0; i < N; i++)
89          printf("arr[%d] = %d\n", i, *(arr + i));
90  }
91
92  int even_calc(int *arr)
93  {
94
95  }
96
97
98  bool belongs_x(int numEven, int x1, int x2)
99  {
100
101 }
102
103
104 int sum_calc(int *arr)
105 {
106
107 }
108
109
110 bool belongs_y(int sumNums, int y1, int y2)
111 {
112
113 }
114
115
116 void print_combs(int *arr)
117 {
118     int i;
119
120     for (i = 0; i < COMBSN; i++)
121         printf("%d ", *(arr + i));
122     printf("\n");
123 }
124
125
126 void print(int N)
127 {
128
129 }
130
131
132
133 int sum_comb_calc()
134 {
135
136 }
137
```

```
138
139 int not_printed()
140 {
141
142 }
143
144
145 int not_first_condition()
146 {
147
148 }
149
150
151 int not_second_condition_only()
152 {
153
154 }
155
156
157 int frequency()
158 {
159
160 }
```

2.4 combinations.h

```
1  #ifndef COMBINATIONS_H
2  #define COMBINATIONS_H
3
4  int get_n();
5  int get_k();
6
7  void x_pair(int *, int *);
8  void y_pair(int *, int *);
9
10 void print_combs(int *);
11 int combinations_count(int);
12 int factorial(int);
13
14 #endif
```

2.5 arrhandler.h

```
1  #ifndef ARRHANDLER_H
2  #define ARRHANDLER_H
3
4  #include <stdbool.h>
5
6  int *fill_array(int);
7  bool exists_in_array(int *, int, int);
```

```
8
9  void quicksort(int *, int, int);
10 int partition(int *, int, int);
11 void swap(int *, int *);
12 void printarray(int *, int);
13
14 void combinations(int *, int, int, int, int);
15 int even_calc(int *);
16 bool belongs_x(int, int, int);
17 int sum_calc(int *);
18 bool belongs_y(int, int, int);
19 int sum_comb_calc();
20
21 void print();
22 int not_printed();
23 int not_first_condition();
24 int not_second_condition_only();
25 int frequency();
26
27 #endif
```

2.6 Περιγραφή υλοποιήσης

3 fcombinations - συνδυασμοί από αρχείο

3.1 main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "fcombinations.h"
4  #include "arrhandler.h"
5
6
7  int main(int argc, char **argv)
8  {
9      int N, K;
10     int *arr;
11     int x1, x2, y1, y2;
12
13     read_file(argv);
14
15     return 0;
16 }
```

3.2 fcombinations.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include <string.h>
```

```c
#include "fcombinations.h"
#include "ccolors.h"

#define COMBSN 6


void read_file(char **argv)
{
    FILE *dataFile = fopen(argv[1], "r");

    if (dataFile == NULL)
    {
        set_color(BOLD_RED);
        printf("Error opening the file, exiting...\n");
        set_color(STANDARD);
        exit(EXIT_FAILURE);
    }
    else
    {
        printf("Cool\n");
        // fscanf();
    }

    fclose(dataFile);
}


void x_pair(int *x1, int *x2)
{
    do
    {
        printf("x1: ");
        scanf("%d", x1);
        printf("x2: ");
        scanf("%d", x2);
    } while (*x1 < 0 || *x1 > *x2 || *x2 > 6);
}


void y_pair(int *y1, int *y2)
{
    do
    {
        printf("y1: ");
        scanf("%d", y1);
        printf("y2: ");
        scanf("%d", y2);
    } while (*y1 < 21 || *y1 > *y2 || *y2 > 279);
}

```

```
55
56 void combinations(int *arr, int x1, int x2, int y1, int y2)
57 {
58     int i, j, temp;
59
60     for (i = 1; i <= COMBSN; i++)
61     {
62         for (j = 0; j < COMBSN-1; j++)
63         {
64             temp = *(arr + j);
65             *(arr + j) = *(arr + j + 1);
66             *(arr + j + 1) = temp;
67         }
68     }
69 }
70
71
72 int combinations_count(int N)
73 {
74     int numCombinations;
75
76     numCombinations = factorial(N) / (factorial(COMBSN) *
    factorial(N - COMBSN));
77
78     return numCombinations;
79 }
80
81
82 int factorial(int num)
83 {
84     int i, fac;
85
86     for (i = 1, fac = 1; i <= num; i++)
87         fac *= i;
88
89     return fac;
90 }
```

3.3 arrhandler.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "arrhandler.h"
4 //#include "ccolors.h"
5
6 #define COMBSN 6
7
8
9 int *fill_array(int N)
10 {
```

```
11      int *arr, num, i = 0;
12
13      arr = (int *)malloc(N * sizeof(int));
14
15      if (arr == NULL)
16      {
17          //set_color(BOLD_RED);
18          printf("Error! Not enough memory, exiting...\n");
19          //set_color(STANDARD);
20          exit(EXIT_FAILURE);
21      }
22      else
23      {
24          do
25          {
26              printf("arr[%d]: ", i);
27              scanf("%d", &num);
28
29              if (num >= 1 && num <= 49)
30              {
31                  if (i == 0) { *(arr + i) = num; i++; }
32                  else
33                  {
34                      if (!exists_in_array(arr, N, num)) { *(
    arr + i) = num; i++; }
35                      else printf("Give a different number.\n"
    );
36                  }
37              }
38              else printf("Give a number in [1, 49].\n");
39          } while (i < N);
40      }
41
42      return arr;
43  }
44
45
46  bool exists_in_array(int *arr, int N, int num)
47  {
48      int *arrEnd = arr + (N - 1);
49      while (arr <= arrEnd && *arr != num) arr++;
50      return (arr <= arrEnd) ? true : false;
51  }
52
53
54  void quicksort(int *arr, int low, int high)
55  {
56      if (low < high)
57      {
58          int partIndex = partition(arr, low, high);
```

```
59          quicksort(arr, low, partIndex - 1);
60          quicksort(arr, partIndex + 1, high);
61      }
62  }
63
64
65  int partition(int *arr, int low, int high)
66  {
67      int pivot = *(arr + high);
68      int i = (low - 1), j;
69
70      for (j = low; j <= high - 1; j++)
71          if (*(arr + j) < pivot)
72              swap(arr + ++i, arr + j);
73
74      swap(arr + (i + 1), arr + high);
75      return (i + 1);
76  }
77
78
79  void swap(int *a, int *b)
80  {
81      int temp = *a;
82      *a = *b;
83      *b = temp;
84  }
85
86
87  void printarray(int *arr, int N)
88  {
89      for (int i = 0; i < N; i++)
90          printf("arr[%d] = %d\n", i, *(arr + i));
91  }
92
93  int even_calc(int *arr)
94  {
95
96  }
97
98
99  bool belongs_x(int numEven, int x1, int x2)
100 {
101
102 }
103
104
105 int sum_calc(int *arr)
106 {
107
108 }
```

```
109
110
111 bool belongs_y(int sumNums, int y1, int y2)
112 {
113
114 }
115
116
117 void print_combs(int *arr)
118 {
119     int i;
120
121     for (i = 0; i < COMBSN; i++)
122         printf("%d ", *(arr + i));
123     printf("\n");
124 }
125
126
127 void print(int N)
128 {
129
130 }
131
132
133
134 int sum_comb_calc()
135 {
136
137 }
138
139
140 int not_printed()
141 {
142
143 }
144
145
146 int not_first_condition()
147 {
148
149 }
150
151
152 int not_second_condition_only()
153 {
154
155 }
156
157
158 int frequency()
```

```
159 {
160
161 }
```

3.4 fcombinations.h

```
1  #ifndef COMBINATIONS_H
2  #define COMBINATIONS_H
3
4  #include <stdbool.h>
5
6  void read_file();
7
8  int get_n();
9  int get_k();
10 int *fill_array(int);
11 bool exists_in_array(int *, int, int);
12 int *sort(int *);
13
14 void x_pair(int *, int *);
15 void y_pair(int *, int *);
16
17 void combinations(int *, int, int, int, int);
18 int even_calc(int *);
19 bool belongs_x(int, int, int);
20 int sum_calc(int *);
21 bool belongs_y(int, int, int);
22 void print_combs(int *);
23 int combinations_count(int);
24 int factorial(int);
25 int sum_comb_calc();
26
27 void print();
28 int not_printed();
29 int not_first_condition();
30 int not_second_condition_only();
31 int frequency();
32
33 #endif
```

3.5 arrhandler.h

```
1  #ifndef ARRHANDLER_H
2  #define ARRHANDLER_H
3
4  #include <stdbool.h>
5
6  int *fill_array(int);
7  bool exists_in_array(int *, int, int);
8
```

```
9  void quicksort(int *, int, int);
10 int partition(int *, int, int);
11 void swap(int *, int *);
12 void printarray(int *, int);
13
14 void combinations(int *, int, int, int, int);
15 int even_calc(int *);
16 bool belongs_x(int, int, int);
17 int sum_calc(int *);
18 bool belongs_y(int, int, int);
19 int sum_comb_calc();
20
21 void print();
22 int not_printed();
23 int not_first_condition();
24 int not_second_condition_only();
25 int frequency();
26
27 #endif
```

3.6 Περιγραφή υλοποιήσης

4 minesweeper - ναρκαλιευτής

4.1 main.c

```
1  #include "minesweeper.h"
2
3
4  int main(int argc, char **argv)
5  {
6      main_win();
7      start();
8      endwin();
9
10     return 0;
11 }
```

4.2 minesweeper.c

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <time.h>
4  #include "minesweeper.h"
5
6
7  void main_win()
8  {
9      initscr();
10     noecho();
```

```
11      cbreak();
12
13      WINDOW *mainWin = newwin(0, 0, 0, 0);
14      box(mainWin, 0, 0);
15      refresh();
16      wrefresh(mainWin);
17      keypad(mainWin, true);
18  }
19
20
21  void start()
22  {
23      int yMax, xMax;
24      int numSettings = 3;
25      getmaxyx(stdscr, yMax, xMax);
26
27      WINDOW *menuWin = newwin(numSettings+2, xMax-10, yMax-7,
        5);
28      box(menuWin, 0, 0);
29      refresh();
30      wrefresh(menuWin);
31      keypad(menuWin, true);
32
33      //set_mode(menuWin);
34
35      int WIDTH = set_width(menuWin, xMax);
36      int HEIGHT = set_height(menuWin, yMax);
37      int NMINES = set_nmines(menuWin, WIDTH * HEIGHT);
38
39      game_win(WIDTH, HEIGHT, NMINES);
40      getchar();
41  }
42
43
44  void set_mode(WINDOW *menuWin) // later
45  {
46      char mode;
47      mvwprintw(menuWin, 1, 1, "Keyboard or text mode (k/t): "
        );
48      wrefresh(menuWin);
49      scanw("%c", &mode);
50      mvwprintw(menuWin, 1, strlen("Keyboard or text mode (k/t
        ): ") + 1, "%c", mode);
51      wrefresh(menuWin);
52
53      switch (mode) // clear contents first
54      {
55          case 'k':
56          case 'K':
57              mvwprintw(menuWin, 2, 1, "Keyboard mode");
```

```
58              wrefresh(menuWin);
59              break;
60          case 't':
61          case 'T':
62              mvwprintw(menuWin, 2, 1, "Text mode");
63              wrefresh(menuWin);
64              break;
65          default:
66              break;
67      }
68  }
69
70
71  int set_width(WINDOW *menuWin, int xMax)
72  {
73      int WIDTH;
74
75      do
76      {
77          mvwprintw(menuWin, 1, 1, "Width (Max = %d): ", xMax-
        12);
78          wrefresh(menuWin);
79          scanw("%d", &WIDTH);
80          mvwprintw(menuWin, 1, strlen("Width (Max = XXX): ")
        + 1, "%d", WIDTH);
81          wrefresh(menuWin);
82      } while (WIDTH < 5 || WIDTH > xMax - 12);
83
84      return WIDTH;
85  }
86
87
88  int set_height(WINDOW *menuWin, int yMax)
89  {
90      int HEIGHT;
91
92      do
93      {
94          mvwprintw(menuWin, 2, 1, "Height (Max = %d): ", yMax
        -12);
95          wrefresh(menuWin);
96          scanw("%d", &HEIGHT);
97          mvwprintw(menuWin, 2, strlen("Height (Max = YYY): ")
         + 1, "%d", HEIGHT);
98          wrefresh(menuWin);
99      } while (HEIGHT < 5 || HEIGHT > yMax - 12);
100
101     return HEIGHT;
102 }
103
```

```
104
105 int set_nmines(WINDOW *menuWin, int DIMENSIONS)
106 {
107     int NMINES;
108
109     do
110     {
111         mvwprintw(menuWin, 3, 1, "Mines (Max = %d): ",
    DIMENSIONS-600); // -500 so the player has a chance to
    win
112         wrefresh(menuWin);
113         scanw("%d", &NMINES);
114         mvwprintw(menuWin, 3, strlen("Mines (Max = MMMM): ")
     + 1, "%d", NMINES);
115         wrefresh(menuWin);
116     } while (NMINES < 1 || NMINES > DIMENSIONS-500);
117
118     return NMINES;
119 }
120
121
122 void game_win(int WIDTH, int HEIGHT, int NMINES)
123 {
124     int yMax, xMax;
125     getmaxyx(stdscr, yMax, xMax);
126
127     WINDOW *gameWin = newwin(43, xMax-10, (yMax/2) - 24, 5);
128     box(gameWin, 0, 0);
129     refresh();
130     wrefresh(gameWin);
131     keypad(gameWin, true);
132
133     init_dispboard(gameWin, WIDTH, HEIGHT);
134     init_mineboard(gameWin, WIDTH, HEIGHT, NMINES);
135 }
136
137
138 void init_dispboard(WINDOW *gameWin, int WIDTH, int HEIGHT)
139 {
140     int i;
141     char **dispboard = (char **)malloc(WIDTH * sizeof(char *
    ));
142     for (i = 0; i < WIDTH; i++)
143         dispboard[i] = (char *)malloc(HEIGHT);
144
145     if (dispboard == NULL)
146     {
147         mvprintw(1, 1, "Error, not enough memory, exiting...
    ");
148         exit(EXIT_FAILURE);
```

```
149     }
150     else
151     {
152         fill_dispboard(dispboard, WIDTH, HEIGHT);
153         print(gameWin, dispboard, WIDTH, HEIGHT);
154         getchar();
155     }
156 
157     free(dispboard);
158 }
159 
160 void fill_dispboard(char **dispboard, int WIDTH, int HEIGHT)
161 {
162     int i, j;
163 
164     for (i = 0; i < WIDTH; i++)
165         for (j = 0; j < HEIGHT; j++)
166             dispboard[i][j] = HIDDEN;
167 }
168 
169 
170 void init_mineboard(WINDOW *gameWin, int WIDTH, int HEIGHT,
    int NMINES)
171 {
172     int i;
173     char **mineboard = (char **)malloc(WIDTH * sizeof(char *
    ));
174     for (i = 0; i < WIDTH; i++)
175         mineboard[i] = (char *)malloc(HEIGHT);
176 
177     if (mineboard == NULL)
178     {
179         mvprintw(1, 1, "Error, not enough memory, exiting...
    ");
180         exit(EXIT_FAILURE);
181     }
182     else
183     {
184         fill_spaces(mineboard, WIDTH, HEIGHT, NMINES);
185         place_mines(mineboard, WIDTH, HEIGHT, NMINES);
186         add_adj(mineboard, WIDTH, HEIGHT);
187 
188         print(gameWin, mineboard, WIDTH, HEIGHT);
189         filewrite(mineboard, WIDTH, HEIGHT);
190     }
191 
192     free(mineboard);
193 }
194 
195 
```

```
196 void place_mines(char **mineboard, int WIDTH, int HEIGHT,
    int NMINES)
197 {
198     int i, wRand, hRand;
199 
200     srand(time(NULL));
201 
202     for (i = 0; i < NMINES; i++)
203     {
204         wRand = rand() % WIDTH;
205         hRand = rand() % HEIGHT;
206         mineboard[wRand][hRand] = MINE;
207     }
208 }
209 
210 
211 void add_adj(char **mineboard, int WIDTH, int HEIGHT)
212 {
213     int i, j;
214 
215     for (i = 0; i < WIDTH; i++)
216         for (j = 0; j < HEIGHT; j++)
217             if (!is_mine(mineboard, i, j))
218                 mineboard[i][j] = adj_mines(mineboard, i, j,
     WIDTH, HEIGHT) + '0';
219 }
220 
221 
222 bool is_mine(char **mineboard, int row, int col)
223 {
224     return (mineboard[row][col] == MINE) ? true : false;
225 }
226 
227 bool outof_bounds(int row, int col, int WIDTH, int HEIGHT)
228 {
229     return (row < 0 || row > WIDTH || col < 0 || col >
    HEIGHT) ? true : false;
230 }
231 
232 
233 int8_t adj_mines(char **mineboard, int row, int col, int
    WIDTH, int HEIGHT)
234 {
235     int8_t numAdj = 0;
236 
237     if (!outof_bounds(row, col - 1, WIDTH, HEIGHT)      &&
    mineboard[row][col-1]    == MINE) numAdj++; // North
238     if (!outof_bounds(row, col + 1, WIDTH, HEIGHT)      &&
    mineboard[row][col+1]    == MINE) numAdj++; // South
239     if (!outof_bounds(row + 1, col, WIDTH, HEIGHT)      &&
```

```
    mineboard[row+1][col]    == MINE) numAdj++; // East
240     if (!outof_bounds(row - 1, col, WIDTH, HEIGHT)      &&
    mineboard[row-1][col]    == MINE) numAdj++; // West
241     if (!outof_bounds(row + 1, col - 1, WIDTH, HEIGHT)  &&
    mineboard[row+1][col-1]  == MINE) numAdj++; // North-East
242     if (!outof_bounds(row - 1, col - 1, WIDTH, HEIGHT)  &&
    mineboard[row-1][col-1]  == MINE) numAdj++; // North-West
243     if (!outof_bounds(row + 1, col + 1, WIDTH, HEIGHT)  &&
    mineboard[row+1][col+1]  == MINE) numAdj++; // South-East
244     if (!outof_bounds(row - 1, col + 1, WIDTH, HEIGHT)  &&
    mineboard[row-1][col+1]  == MINE) numAdj++; // South-West
245
246     return numAdj;
247 }
248
249
250 void fill_spaces(char **mineboard, int WIDTH, int HEIGHT,
    int NMINES)
251 {
252     int i, j;
253
254     for (i = 0; i < WIDTH; i++)
255         for (j = 0; j < HEIGHT; j++)
256             if (mineboard[i][j] != MINE)
257                 mineboard[i][j] = '-';
258 }
259
260
261 void print(WINDOW *gameWin, char **mineboard, int WIDTH, int
     HEIGHT)
262 {
263     int i, j;
264
265     for (i = 0; i < WIDTH; i++)
266     {
267         for (j = 0; j < HEIGHT; j++)
268         {
269             mvwaddch(gameWin, j + 1, i + 1, mineboard[i][j])
    ;
270             wrefresh(gameWin);
271         }
272     }
273 }
274
275
276 void filewrite(char **mineboard, int WIDTH, int HEIGHT)
277 {
278     FILE *mnsOut = fopen("mnsout.txt", "w");
279     int i, j;
280
```

```
281     if (mnsOut == NULL)
282     {
283         mvprintw(1, 1, "Error opening file, exiting...");
284         exit(EXIT_FAILURE);
285     }
286     else
287     {
288         fprintf(mnsOut, "Mine hit at position (%d, %d)\n\n",
        1, 2); // add actual position
289         fprintf(mnsOut, "Board overview\n\n");
290 
291         for (i = 0; i < WIDTH; i++)
292         {
293             for (j = 0; j < HEIGHT; j++)
294                 fprintf(mnsOut, "%c ", mineboard[i][j]);
295             fprintf(mnsOut, "\n");
296         }
297 
298         mvprintw(1, 1, "Session written to file");
299         refresh();
300     }
301 
302     fclose(mnsOut);
303 }
```

4.3 minesweeper.h

```
1  #ifndef MINESWEEPER_H
2  #define MINESWEEPER_H
3  
4  #if defined linux || defined __unix__
5  #include <ncurses.h>
6  #elif defined _WIN32 || defined _WIN64
7  #include <pdcurses.h>
8  #include <stdint.h>
9  #endif
10 
11 #include <stdbool.h>
12 
13 #define HIDDEN '#'
14 #define MINE '*'
15 
16 void main_win();
17 void start();
18 void set_mode(struct _win_st*);
19 
20 int set_width(struct _win_st*, int);
21 int set_height(struct _win_st*, int);
22 int set_nmines(struct _win_st*, int);
23 
```

```
24 void game_win(int, int, int);
25 void init_dispboard(struct _win_st*, int, int);
26 void fill_dispboard(char **, int, int);
27 void init_mineboard(struct _win_st*, int, int, int);
28 void place_mines(char **, int, int, int);
29 void add_adj(char **, int, int);
30 bool is_mine(char **, int, int);
31 bool outof_bounds(int, int, int, int);
32 int8_t adj_mines(char **, int, int, int, int);
33
34 void fill_spaces(char **, int, int, int);
35
36 void selection();
37 void transfer(char **, char **, int, int);
38 void reveal(struct _win_st*, char **, char **, int, int);
39 void game_over(struct _win_st*);
40
41 void print(struct _win_st*, char **, int, int);
42 void filewrite(char **, int, int);
43
44 #endif
```

4.4 Περιγραφή υλοποιήσης

5 Διευκρινήσεις

6 Εργαλεία

- Editors: Visual Studio Code, Vim
- OS: Arch Linux
- Shell: zsh
- Συγγραφή: LaTeX