

```
1 # 10-double-pointers
2
3 ## `echo.c`
4
5 - Linux `echo`
6 - C standard
7
8 ## `echo-escaped.c`
9
10 ## `scores.c`
11
12 - `student_score_table`: as a 2D array
13 - `Print`
14 - `int table[][COLS]` vs. `int (*table)[COLS]`
15 - `malloc`
16 - `int *`
17 - `int (*)[COLS]`
18
19 ## `game-of-life-pointer.c`
20
21 - `int (*)[]`
22
23 ## `sort.c`
24
25 - function pointers
26 - `f(a)`: `f` is actually a function pointer
27 - `&f`: `f` is a function
28
29 ## `decl.c`
30
31 - for more fun
```

```
1 /**
2  * file: echo-escaped.c
3  *
4  * Created by hengxin on 12/01/22.
5  */
6
7 #include <stdio.h>
8 #include <stdbool.h>
9 #include <string.h>
10
11 // -e: escaped; -E: not escaped; must be in argv[1]
12 int main(int argc, char *argv[]) {
13     if (argc < 2) {
14         printf("Usage: -e for escaped; -E for unescaped\n");
15         return 0;
16     }
17
18     bool escaped = false;
19     char *flag = argv[1];
20     if (strcmp(flag, "-e") == 0) {
21         escaped = true;
22     } else if (strcmp(flag, "-E") != 0) {
23         printf("Illegal flag '%s'\n"
24             "Use -e or -E.\n", flag);
25         return 0;
26     }
27
28     char **args = argv + 1;
29     char *arg = NULL;
30     while ((arg = *++args) != NULL) {
31         if (strcmp(arg, "\\t") == 0 && escaped) {
32             printf("\t");
33         } else if (strcmp(arg, "\\n") == 0 && escaped) {
34             printf("\n");
35         } else {
36             printf("%s ", arg);
37         }
38     }
39 }
```

```

1  /**
2   * file: echo.c
3   *
4   * Echo program (command-line) arguments.
5   *
6   * Created by hengxin on 12/01/22.
7   */
8
9  #include <stdio.h>
10
11 /**
12  * @param argc argument count.
13  * @param argv argument vector.
14  * By convention (not the C standard), argv[0] is the name by which
15  * the program was invoked.
16  * By the C standard, argv[argc] is a null pointer.
17  * argv[1] .. argv[argc - 1] are the command-line parameters.
18  */
19 // int argv[] : int *arg
20 // char *argv[] : char **argv
21 int main(int argc, char **argv) {
22     printf("argc = %d, program = %s\n", argc, argv[0]);
23
24     // (1) char *argv[]
25     // for (int i = 1; i < argc; i++) {
26     //     printf("%s\n", argv[i]);
27     // }
28
29     // (2) char **argv (for version)
30     // for (char **ptr = argv + 1; *ptr != NULL; ptr++) {
31     //     printf("%s ", *ptr);
32     // }
33
34     // (3) char **argv (while version)
35     // char **ptr = argv + 1;
36     // while (*ptr != NULL) {
37     //     printf("%s ", *ptr++);
38     // }
39
40     // (4) char **argv (while version with ***)
41     // wrong version!!!
42     // char **ptr = argv + 1;
43     // while (*ptr++ != NULL) {
44     //     printf("%s ", *ptr);
45     // }
46
47     // correct version ^^
48     char **ptr = argv;
49     char *arg = NULL;
50     while ((arg = **ptr) != NULL) {
51         printf("%s ", arg);
52     }

```

```
53  
54     return 0;  
55 }
```

```
1  /**
2   * file: scores.c
3   *
4   * Created by hengxin on 12/01/22.
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9
10 #define COLS 3
11
12 void Print(int table[][COLS], int rows);
13
14 int main() {
15     /**
16      * C, Java, Python
17      *
18      * musician_score_table is a 2D (two-dimensional) array.
19      * musician_score_table is an array of 5 arrays,
20      * each of which contains 3 elements.
21      */
22     int rows = 0;
23     printf("Please input the number of students.\n");
24     scanf("%d", &rows);
25
26     int (*musician_score_table)[COLS] = malloc(rows * COLS * sizeof *
musician_score_table);
27     if (musician_score_table == NULL) {
28         printf("Failed to allocate memory.\n");
29         return 0;
30     }
31
32     printf("Please input the scores of these students.\n");
33     // for (int i = 0; i < rows * COLS; i++) {
34     //     scanf("%d", &musician_score_table[i]);
35     // }
36
37     for (int row = 0; row < rows; row++) {
38         for (int col = 0; col < COLS; col++) {
39             scanf("%d", &musician_score_table[row][col]);
40         }
41     }
42
43     Print(musician_score_table, rows);
44
45     printf("musician_score_table[3][2] = %d\n",
46           musician_score_table[3][2]);
47     printf("musician_score_table[3][2] = %d\n",
48           *(musician_score_table[3] + 2));
49     printf("musician_score_table[3][2] = %d\n",
50           (*(musician_score_table + 3))[2]);
51     printf("musician_score_table[3][2] = %d\n",
52           (*(musician_score_table + 3) + 2));
```

```
53
54  /**
55   * musician_score_table is a pointer to (an array of 3 ints)
56   */
57  int (*ptr_scores)[COLS] = musician_score_table;
58  printf("ptr_scores[3][2] = %d\n",
59        (*(ptr_scores + 3))[2]);
60
61  free(musician_score_table);
62
63  return 0;
64 }
65
66 // int table[] : int *table
67 // int table[][cols] : int (*table)[cols]
68 void Print(int (*table)[COLS], int rows) {
69     printf("\n");
70     for (int i = 0; i < rows; i++) {
71         for (int j = 0; j < COLS; j++) {
72             printf("%d ", table[i][j]);
73         }
74         printf("\n");
75     }
76 }
77
78 // int student_score_table[ROWS][COLS] = {
79 //     {0, 10, 20},
80 //     {10, 20, 30},
81 //     {20, 30, 40},
82 //     {30, 40, 50},
83 //     {40, 50, 60}
84 // };
```

```

1 //
2 // Created by hengxin on 12/01/22.
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
12 // extended_board as a parameter
13 void ExtendBoard(const int origin_board[][SIZE],
14                 int extended_board[][SIZE + 2]);
15 void PrintExtendedBoard(const int extended_board[][SIZE + 2]);
16 void GenerateNewBoard(const int old_board[][SIZE + 2],
17                      int new_board[][SIZE + 2]);
18 void CopyExtendedBoard(const int src_board[][SIZE + 2],
19                       int dest_board[][SIZE + 2]);
20 void SleepAndClear(int sec);
21
22 int main() {
23     const int board[SIZE][SIZE] = {
24         {0},
25         {0, 1, 1, 0, 0, 0},
26         {0, 1, 1, 0, 0, 0},
27         {0, 0, 0, 1, 1, 0},
28         {0, 0, 0, 1, 1, 0},
29         {0}
30     };
31
32     int old_board[SIZE + 2][SIZE + 2] = {0};
33     ExtendBoard(board, old_board);
34     PrintExtendedBoard(old_board);
35     // SleepAndClear(1);
36
37     int new_board[SIZE + 2][SIZE + 2] = {0};
38     for (int round = 0; round < 10; round++) {
39         GenerateNewBoard(old_board, new_board);
40         SleepAndClear(1);
41         PrintExtendedBoard(new_board);
42         CopyExtendedBoard(new_board, old_board);
43     }
44
45     return 0;
46 }
47
48 void ExtendBoard(const int origin_board[][SIZE],
49                 int extended_board[][SIZE + 2]) {
50     for (int row = 0; row < SIZE + 2; row++) {
51         for (int col = 0; col < SIZE + 2; col++) {
52             if (row == 0 || row == SIZE + 1 || col == 0 || col == SIZE + 1
53 ) {

```

```

53     extended_board[row][col] = 0;
54 } else {
55     extended_board[row][col] = origin_board[row - 1][col - 1];
56 }
57 }
58 }
59 }
60
61 void PrintExtendedBoard(const int extended_board[][SIZE + 2]) {
62     for (int row = 1; row <= SIZE; row++) {
63         for (int col = 1; col <= SIZE; col++) {
64             printf("%c ", extended_board[row][col] ? '*' : ' ');
65         }
66         printf("\n");
67     }
68 }
69
70 void GenerateNewBoard(const int old_board[][SIZE + 2],
71                      int new_board[][SIZE + 2]) {
72     for (int row = 1; row <= SIZE; row++) {
73         for (int col = 1; col <= SIZE; col++) {
74             // count the number of neighbours of old_board[row][col]
75             int neighbours =
76                 old_board[row - 1][col - 1] +
77                 old_board[row - 1][col] +
78                 old_board[row - 1][col + 1] +
79                 old_board[row][col - 1] +
80                 old_board[row][col + 1] +
81                 old_board[row + 1][col - 1] +
82                 old_board[row + 1][col] +
83                 old_board[row + 1][col + 1];
84
85             // evaluate the new board
86             if (old_board[row][col]) { // old_board[row][col] is alive
87                 new_board[row][col] = (neighbours == 2 || neighbours == 3);
88             } else { // old_board[row][col] is dead
89                 new_board[row][col] = (neighbours == 3);
90             }
91         }
92     }
93 }
94
95 void CopyExtendedBoard(const int src_board[][SIZE + 2],
96                      int dest_board[][SIZE + 2]) {
97     for (int row = 1; row <= SIZE; row++) {
98         for (int col = 1; col <= SIZE; col++) {
99             dest_board[row][col] = src_board[row][col];
100         }
101     }
102 }
103
104 void SleepAndClear(int sec) {
105     sleep(sec);

```



```
106     system("clear");  
107 }
```

```
1 /**
2  * file: decl.c
3  *
4  * Created by hengxin on 12/01/22.
5  */
6
7 int main() {
8     // argv is a pointer to pointer to char
9     char **argv;
10
11     // names is an array consisting of 10 pointers to int
12     int *names[10];
13
14     // musician_score_table is a pointer to an array
15     // consisting of 10 integers
16     int (*musician_score_table)[10];
17
18     // StrCpyStd is a function
19     int *StrCpyStd(char *dest, const char *src);
20
21
22     // comp is a pointer to function returning int
23     int (*comp)(const void *left, const void *right);
24
25     // <stdlib.h>
26     // Registers the function pointed to by func to be called on normal
    program termination.
27     int atexit(void (*func)(void));
28
29     // Sets the error handler for signal sig.
30     // Previous signal handler on success or SIG_ERR on failure
31     void (*signal(int sig, void (*handler)(int)))(int);
32
33     // func is a function returning pointer to array[] of
34     // pointer to function returning char
35     char ((*func(int num, char *str))[])(int);
36
37     // array[3] of pointer to function
38     // returning pointer to array[5] of char
39     char ((*arr[3])())[5];
40 }
```

```
1 // Created by hfwei on 2022/11/25.
2
3 #include <stdio.h>
4 #include <string.h>
5
6 #define LEN 10
7
8 void Swap(char **left, char **right);
9 void PrintStrs(const char *str[], int len);
10 void SelectionSort(char *str[], int len);
11
12 int main() {
13     const char *musicians[LEN] = {
14         "Luo Dayou",
15         "Cui Jian",
16         "Dou Wei",
17         "Zhang Chu",
18         "Li Zhi",
19         "Wan Qing",
20         "WuTiaoRen",
21         "ZuoXiao",
22         "He Mage",
23         "He Yong",
24     };
25
26     PrintStrs(musicians, LEN);
27     SelectionSort(musicians, LEN);
28     PrintStrs(musicians, LEN);
29 }
30
31 void PrintStrs(const char *str[], int len) {
32     printf("\n");
33     for (int i = 0; i < len; i++) {
34         printf("%s\n", str[i]);
35     }
36     printf("\n");
37 }
38
39 void SelectionSort(char *str[], int len) {
40     for (int i = 0; i < len; i++) {
41         // find the minimum of musicians[i .. len - 1]
42         const char *min = str[i];
43         int min_index = i;
44
45         for (int j = i + 1; j < len; j++) {
46             if (strcmp(min, str[j]) > 0) {
47                 min = str[j];
48                 min_index = j;
49             }
50         }
51
52         // swap str[i] and str[min_index]
53         Swap(str + i, str + min_index);
54     }
55 }
```

```
54     }  
55 }  
56  
57 void Swap(char **left, char **right) {  
58     char *temp = *left;  
59     *left = *right;  
60     *right = temp;  
61 }
```

```

1  /**
2   * file: sort.c
3   *
4   * Created by hengxin on 12/01/22.
5   *
6   * A nice function pointer example on Riemann integration:
7   * https://en.wikipedia.org/wiki/Function\_pointer
8   */
9
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <limits.h>
13 #include <string.h>
14
15 int CompareInts(const void *left, const void *right);
16 void PrintInts(const int *integers, int len);
17
18 int CompareStrs(const void *left, const void *right);
19 void PrintStrs(const char *str[], int len);
20
21 int main() {
22     // sort an array of integers
23     int integers[] = {-2, 99, 0, -743, 2, INT_MIN, 4};
24     int size_of_integers = sizeof integers / sizeof *integers;
25
26     /**
27      * void qsort( void *ptr, size_t count, size_t size,
28                  int (*comp)(const void *, const void *) );
29      */
30     int (*comp)(const void *, const void *) = CompareInts;
31     // you should not do this!!!
32     // printf("sizeof comp : %zu\n", sizeof comp);
33     printf("comp : %p\n", comp);
34     printf("*comp : %p\n", *comp);
35     printf("CompareInts : %p\n", CompareInts);
36     printf("&CompareInts : %p\n", &CompareInts);
37
38     qsort(integers, size_of_integers, sizeof *integers, comp);
39     PrintInts(integers, size_of_integers);
40
41     // Call functions indirectly via function pointers.
42     int a = 10;
43     int b = 20;
44     printf("%d %s %d\n", a, comp(&a, &b) > 0 ? ">" : "<=", b);
45
46     // Sorting an array of strings
47     const char *names[] = {
48         "Luo Dayou",
49         "Cui Jian",
50         "Dou Wei",
51         "Zhang Chu",
52         "He Yong",
53         "Wan Qing",

```

```

54     "WuTiaoRen",
55     "ZuoXiao",
56     "Hu Mage",
57     "Li Zhi"
58 };
59 int size_of_names = sizeof names / sizeof *names;
60
61 comp = CompareStrs;
62 qsort(names, size_of_names, sizeof *names, comp);
63 PrintStrs(names, size_of_names);
64 }
65
66 int CompareInts(const void *left, const void *right) {
67     int int_left = *(const int *) left;
68     int int_right = *(const int *) right;
69
70     if (int_left < int_right) {
71         return -1;
72     }
73
74     if (int_left > int_right) {
75         return 1;
76     }
77
78     return 0;
79
80 //    return int_left - int_right; // erroneous shortcut (fails if
81 //    INT_MIN is present)
82 }
83 // actual arguments: char *const *
84 int CompareStrs(const void *left, const void *right) {
85     char *const *pp1 = left;
86     char *const *pp2 = right;
87     return strcmp(*pp1, *pp2);
88 }
89
90 void PrintInts(const int *integers, int len) {
91     printf("\n");
92     for (int i = 0; i < len; ++i) {
93         printf("%d ", integers[i]);
94     }
95     printf("\n");
96 }
97
98 void PrintStrs(const char *str[], int len) {
99     printf("\n");
100     for (int i = 0; i < len; i++) {
101         printf("%s\n", str[i]);
102     }
103     printf("\n");
104 }

```