

```
1 //
2 // Created by hfwei on 2023/10/6.
3 //
4
5 #include <stdio.h>
6 #include <stdbool.h>
7
8 bool IsLeapYear(int year);
9
10 int main(void) {
11     // year: in the main function: local variable
12     // life time:
13     // scope: from this point on until the main function exits
14     int year = 0;
15     scanf("%d", &year);
16
17     if (IsLeapYear(year)) {
18         printf("%d is a leap year\n", year);
19     } else {
20         printf("%d is a common year\n", year);
21     }
22
23     return 0;
24 }
25
26 // year: parameter
27 // life time
28 // scope as a local variable
29 bool IsLeapYear(int year) {
30     return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
31 }
```

```
1 //
2 // Created by hfwei on 2023/10/11.
3 //
4
5 #include <stdio.h>
6 #include <stdbool.h>
7
8 /**
9  * @brief decide whether number is a prime
10  * @param number the number to decide
11  * @return true if number is prime; false otherwise
12  */
13 bool IsPrime(int number);
14
15 int main(void) {
16     int max = 0;
17     scanf("%d", &max);
18
19     int count = 0;
20
21     for (int number = 2; number <= max; number++) {
22         if (IsPrime(number)) {
23             count++;
24             printf("%d ", number);
25         }
26     }
27
28     printf("\ncount = %d\n", count);
29
30     return 0;
31 }
32
33 bool IsPrime(int number) {
34     for (int factor = 2; factor * factor <= number; factor++) {
35         if (number % factor == 0) {
36             return false;
37         }
38     }
39
40     return true;
41 }
```

```
1 //
2 // Created by hfwei on 2023/10/11.
3 //
4
5 #include <stdio.h>
6
7 void Print(char ch, int count);
8
9 int main(void) {
10     int lines = 0;
11     scanf("%d", &lines);
12
13     for (int i = 0; i < lines; ++i) {
14         Print(' ', lines - 1 - i);
15         Print('*', 2 * i + 1);
16
17         if (i < lines - 1) {
18             printf("\n");
19         }
20     }
21
22     return 0;
23 }
24
25 void Print(char ch, int count) {
26     for (int i = 0; i < count; ++i) {
27         printf("%c", ch);
28     }
29 }
```

```
1  //
2  // Created by hengxin on 9/14/23.
3  //
4
5  #include <stdio.h>
6  #include <time.h>
7  #include <stdlib.h>
8
9  int main(void) {
10     int high = 100;
11     int number_of_tries = 7;
12
13     /*
14      * (1) print the rules of the game to players
15      */
16     printf("Let us play the Guess the Number game.\n"
17           "The computer will generate a random number between 1 and %d.
18           \n"
19           "You have %d tries.\n",
20           high, number_of_tries);
21
22     /*
23      * (2) generate a random number between 1 and high
24      */
25     srand(time(NULL));
26     int secret = rand() % high + 1;
27     printf("Test: secret = %d\n", secret);
28
29     while (number_of_tries > 0) {
30         /*
31          * (3) ask the player to enter his/her guess
32          */
33         printf("Please enter your guess number.\n"
34               "You still have %d tries.\n", number_of_tries);
35
36         /*
37          * (4) obtain the guessed number, compare it with the secret
38          *      number,
39          *      and inform the player of the result.
40          */
41         int guess = 0;
42         scanf("%d", &guess);
43         printf("Test: guess = %d\n", guess);
44
45         if (guess == secret) {
46             printf("You Win!\n");
47             break;
48         } else if (guess > secret) {
49             printf("guess > secret\n");
50         } else {
51             printf("guess < secret\n");
52         }
53     }
54 }
```

```
52     /*
53     * (5) repeat (3) and (4) until the player wins or loses.
54     */
55     number_of_tries--;
56
57     if (number_of_tries == 0) {
58         printf("You Lose!\n");
59     }
60 }
61
62 return 0;
63 }
```

```
1 //
2 // Created by hfwei on 2023/10/11.
3 //
4
5 #include <stdio.h>
6
7 #define LEN 10
8
9 // dictionary: out of any functions; global variables
10 // life time: program start to end
11 // scope: from this point on until the end of the file (file scope)
12 // int dictionary[LEN] = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 };
13
14 /**
15  * @brief Search for the key in the dict using the binary search
16  * algorithm.
17  * @param key the key to search for
18  * @param dict the dictionary to search
19  * @param len the length of the dictionary
20  * @return the index of the key in the dictionary; -1 if not found
21  */
22 int BinarySearch(int key, const int dict[100], int len);
23
24 int main(void) {
25     const int dictionary[LEN] = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 };
26
27     int key = 0;
28     scanf("%d", &key);
29
30     int index = BinarySearch(key, dictionary, LEN);
31
32     if (index == -1) {
33         printf("Not found!\n");
34     } else {
35         printf("The index of %d is %d.\n", key, index);
36     }
37
38     return 0;
39 }
40
41 int BinarySearch(int key, const int dict[], int len) {
42     int low = 0;
43     int high = len - 1;
44
45     while (low <= high) {
46         int mid = (low + high) / 2;
47
48         if (key > dict[mid]) {
49             low = mid + 1;
50         } else if (key < dict[mid]) {
51             high = mid - 1;
52         } else { // key == dict[mid]
53             return mid;
54         }
55     }
56 }
```

File - D:\cpl\2023-cpl-coding-0\5-function\binary-search.c

```
53     }  
54 }  
55  
56 return -1;  
57 }
```

```
1 //
2 // Created by hfwei on 2023/10/12.
3 //
4
5 #include <stdio.h>
6 #include <string.h>
7 #include <stdbool.h>
8
9 #define LEN 21
10 char string[LEN] = "";
11
12 bool IsPalindrome(const char str[]);
13
14 int main() {
15     printf("Input a string containing at most 20 characters.\n");
16     scanf("%20s", string);
17
18     printf("\'%s\' is %s a palindrome.\n", string,
19           IsPalindrome(string) ? "" : "not");
20
21     return 0;
22 }
23
24 bool IsPalindrome(const char str[]) {
25     int len = strlen(str);
26
27     for (int i = 0, j = len - 1; i < j; i++, j--) {
28         if (str[i] != str[j]) {
29             return false;
30         }
31     }
32
33     return true;
34 }
```



```
1  //
2  // Created by hfwei on 2023/10/19.
3  //
4
5  #include <stdio.h>
6
7  #define LEN_L 5
8  #define LEN_R 6
9
10 int L[LEN_L] = { 1, 3, 5, 7, 9 };
11 int R[LEN_R] = { 0, 2, 4, 6, 8, 10 };
12
13 int main(void) {
14     int l = 0;
15     int r = 0;
16
17     while (l < LEN_L && r < LEN_R) {
18         if (L[l] <= R[r]) {
19             printf("%d ", L[l]);
20             l++;
21         } else {
22             printf("%d ", R[r]);
23             r++;
24         }
25     }
26
27     while (r < LEN_R) {
28         printf("%d ", R[r]);
29         r++;
30     }
31
32     while (l < LEN_L) {
33         printf("%d ", L[l]);
34         l++;
35     }
36
37     // l >= LEN_L || r >= LEN_R
38     // if (l >= LEN_L) {
39     //     while (r < LEN_R) {
40     //         printf("%d ", R[r]);
41     //         r++;
42     //     }
43     // }
44
45     // if (r >= LEN_R) {
46     //     while (l < LEN_L) {
47     //         printf("%d ", L[l]);
48     //         l++;
49     //     }
50     // }
51
52     return 0;
53 }
```

```

1  //
2  // Created by hfwei on 2023/10/12.
3  //
4
5  #include <stdio.h>
6
7  #define LEN 20
8  int numbers[LEN] = { 0 };
9
10 void SelectionSort(int arr[], int len);
11 void Swap(int left, int right);
12 int GetMinIndex(const int arr[], int begin, int end);
13 void Print(const int arr[], int len);
14
15 int main(void) {
16     int len = -1;
17     while (scanf("%d", &numbers[++len]) != EOF);
18
19     // sizeof numbers / sizeof(numbers[0])
20     Print(numbers, len);
21     SelectionSort(numbers, len);
22     Print(numbers, len);
23
24     return 0;
25 }
26
27 // arr: the (copy of the) address of the first element of the `numbers
   ` array
28 void SelectionSort(int arr[], int len) {
29     for (int i = 0; i < len; i++) {
30         int min_index = GetMinIndex(arr, i, len);
31
32         // ERROR: Swap(arr[i], arr[min_index]);
33         int temp = arr[i];
34         arr[i] = arr[min_index];
35         arr[min_index] = temp;
36     }
37 }
38
39 int GetMinIndex(const int arr[], int begin, int end) {
40     int min = arr[begin];
41     int min_index = begin;
42
43     for (int i = begin + 1; i < end; ++i) {
44         if (arr[i] < min) {
45             min = arr[i];
46             min_index = i;
47         }
48     }
49
50     return min_index;
51 }
52

```

```
53 void Swap(int left, int right) {
54     int temp = left;
55     left = right;
56     right = temp;
57 }
58
59 void Print(const int arr[], int len) {
60     printf("\n");
61     for (int i = 0; i < len; i++) {
62         printf("%d ", arr[i]);
63     }
64     printf("\n");
65 }
```

```
1 //
2 // Created by hfwei on 2023/10/19.
3 //
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <unistd.h>
8 #include <synchapi.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     // extended board
29     int old_board[SIZE + 2][SIZE + 2] = { 0 };
30
31     for (int row = 1; row <= SIZE; row++) {
32         for (int col = 1; col <= SIZE; col++) {
33             old_board[row][col] = board[row - 1][col - 1];
34         }
35     }
36
37     // print the original board
38     for (int row = 1; row <= SIZE; row++) {
39         for (int col = 1; col <= SIZE; col++) {
40             printf("%c ", old_board[row][col] ? '*' : ' ');
41         }
42         printf("\n");
43     }
44     system("clear"); // clear the screen/terminal
45
46     int new_board[SIZE + 2][SIZE + 2] = { 0 };
47
48     for (int round = 1; round < 10; round++) {
49         for (int row = 1; row <= SIZE; row++) {
50             for (int col = 1; col <= SIZE; col++) {
51                 // count the number of neighbours of old_board[row][col]
52                 int neighbours =
53                     old_board[row - 1][col - 1] +
```

```
54         old_board[row - 1][col] +
55         old_board[row - 1][col + 1] +
56         old_board[row][col - 1] +
57         old_board[row][col + 1] +
58         old_board[row + 1][col - 1] +
59         old_board[row + 1][col] +
60         old_board[row + 1][col + 1];
61
62     // evaluate the new board
63     if (old_board[row][col]) { // old_board[row][col] is alive
64         new_board[row][col] = (neighbours == 2 || neighbours == 3);
65     } else { // old_board[row][col] is dead
66         new_board[row][col] = (neighbours == 3);
67     }
68 }
69 }
70
71 // print the new board
72 for (int row = 1; row <= SIZE; row++) {
73     for (int col = 1; col <= SIZE; col++) {
74         printf("%c ", new_board[row][col] ? '*' : ' ');
75     }
76     printf("\n");
77 }
78
79 // sleep for a while
80 // Linux: #include <unistd.h>
81 sleep(1);
82 // Windows: #include <windows.h>: Sleep(ms)
83 // Sleep(1000);
84
85 // clear the screen
86 // Linux: #include <stdlib.h>
87 system("clear");
88 // Windows: #include <stdlib.h> system("clr");
89 // system("clr");
90
91 // start the next round
92 for (int row = 1; row <= SIZE; row++) {
93     for (int col = 1; col <= SIZE; col++) {
94         old_board[row][col] = new_board[row][col];
95     }
96 }
97 }
98
99 return 0;
100 }
```

```
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
12 // extended_board as a parameter
13 // 1D array: (int, the address of the first element), the length
14 // 2D array: array of arrays [1][3]:
15 // address + 1 * (sizeof int) * size of col + (sizeof int) * 3
16 // 0: -----
17 // 1: -----
18 // 2: -----
19 void ExtendBoard(const int origin_board[][SIZE],
20                 int extended_board[][SIZE + 2]);
21 void PrintExtendedBoard(const int extended_board[][SIZE + 2]);
22 void GenerateNewBoard(const int old_board[][SIZE + 2],
23                      int new_board[][SIZE + 2]);
24 void CopyExtendedBoard(const int src_board[][SIZE + 2],
25                       int dest_board[][SIZE + 2]);
26 void SleepAndClear(int sec);
27
28 int main() {
29     const int board[SIZE][SIZE] = {
30         { 0 },
31         { 0, 1, 1, 0, 0, 0 },
32         { 0, 1, 1, 0, 0, 0 },
33         { 0, 0, 0, 1, 1, 0 },
34         { 0, 0, 0, 1, 1, 0 },
35         { 0 }
36     };
37
38     int old_board[SIZE + 2][SIZE + 2] = { 0 };
39     ExtendBoard(board, old_board);
40     PrintExtendedBoard(old_board);
41     // SleepAndClear(1);
42
43     int new_board[SIZE + 2][SIZE + 2] = { 0 };
44     for (int round = 0; round < 10; round++) {
45         GenerateNewBoard(old_board, new_board);
46         SleepAndClear(1);
47         PrintExtendedBoard(new_board);
48         CopyExtendedBoard(new_board, old_board);
49     }
50
51     return 0;
52 }
53
```

```

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

```

File - D:\cpl\2023-cpl-coding-0\5-function\game-of-life-transformed.c

```
106     }  
107 }  
108 }  
109  
110 void SleepAndClear(int sec) {  
111     sleep(sec);  
112     system("clear");  
113 }
```