

```
1 // Created by hfwei on 2024/12/25.
2
3 // Visualization of function call: https://tinyurl.com/min-func-call
4 // (Using https://tinyurl.com/)
5
6 #include <stdio.h>
7
8 int Min(int a, int b);
9
10 int main() {
11     int a = 25;
12     int b = 37;
13
14     int min = Min(a, b);
15     printf("%d", min);
16
17     return 0;
18 }
19
20 int Min(int a, int b) { return a > b ? b : a; }
```

```
1 // Created by hfwei on 2024/12/25.
2
3 #include <stdio.h>
4
5 #define LEN_L 5
6 #define LEN_R 6
7
8 int L[LEN_L] = {1, 3, 5, 7, 9};
9 int R[LEN_R] = {0, 2, 4, 6, 8, 10};
10
11 int main(void) {
12     // TODO: merge L and R into a sorted array
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 {
21             printf("%d ", R[r]);
22             r++;
23         }
24     }
25
26     // l >= LEN_L || r >= LEN_R
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     return 0;
38 }
```

```
1 // Created by hfwei on 2024/12/25.
2
3 // Visualization (for n = 4): https://tinyurl.com/fib-re-visual
4 // (Using https://tinyurl.com/)
5
6 #include <stdio.h>
7
8 long long Fib(int n);
9
10 int main() {
11     int n;
12     scanf("%d", &n);
13
14     printf("Fib(%d) = %lld\n", n, Fib(n));
15
16     return 0;
17 }
18
19 long long Fib(int n) {
20     if (n <= 1) {
21         return n;
22     }
23
24     return Fib(n - 1) + Fib(n - 2);
25 }
```

```
1 // Created by hfwei on 2024/12/25.
2
3 // Euclidean algorithm:
4 // gcd(a, b) = gcd(b, a % b)
5 //
6 // Visualization (gcd(64, 48) for illustration):
7 // https://tinyurl.com/gcd-re-visual
8 // (Using https://tinyurl.com/)
9
10 #include <stdio.h>
11
12 int GCD(int a, int b);
13
14 int main() {
15     int a = 0;
16     int b = 0;
17     scanf("%d %d", &a, &b);
18
19     printf("GCD(%d, %d) = %d\n", a, b, GCD(a, b));
20
21     return 0;
22 }
23
24 // gcd(130, 124) = 2
25 // gcd(414, 662) = 2
26 int GCD(int a, int b) {
27     if (b == 0) {
28         return a;
29     }
30
31     return GCD(b, a % b);
32 }
```

```
1 // Created by hfwei on 2024/12/25.
2
3 // Visualization: https://tinyurl.com/min-re
4 // (Using https://tinyurl.com/)
5
6 #include <stdio.h>
7
8 #define NUM 3
9 const int numbers[NUM] = {65, 28, 37};
10
11 int Min(const int nums[], int len);
12
13 int main() {
14     int min = Min(numbers, NUM);
15
16     printf("min = %d\n", min);
17
18     return 0;
19 }
20
21 int Min(const int nums[], int len) {
22     if (len == 1) {
23         return nums[0];
24     }
25
26     int partial_min = Min(nums, len - 1);
27
28     return partial_min < nums[len - 1] ? partial_min : nums[
    len - 1];
29 }
```

```
1 // Created by hfwei on 2024/10/25.
2
3 // Visualization: https://tinyurl.com/sum-re
4 // (Using https://tinyurl.com/)
5
6 #include <stdio.h>
7
8 int Sum(const int nums[], int len);
9
10 int main() {
11     const int numbers[] = {1, 2, 3, 4, 5};
12
13     int sum = Sum(numbers, sizeof numbers / sizeof numbers[0
14     ]);
15     printf("sum = %d\n", sum);
16
17     return 0;
18 }
19
20 int Sum(const int nums[], int len) {
21     if (len == 0) {
22         return 0;
23     }
24
25     int partial_sum = Sum(nums, len - 1);
26
27     return partial_sum + nums[len - 1];
28 }
```

```
1 // Created by hfwei on 2024/12/25.
2 //
3 // WARNING: You can even call the "main" function in
  itself.
4 // But, do NOT write code like this.
5 // Never call the "main" function in your own code.
6
7 #include <stdio.h>
8
9 int main(int argc, char *argv[]) {
10     if (argc == 1) {
11         return 0;
12     }
13
14     printf("%s\n", argv[argc - 1]);
15
16     main(argc - 1, argv);
17
18     return 0;
19 }
```

```
1 # `6-recursion`
2
3 # 6-recursion
4
5 ## Recursion
6
7 - `main-re.c`
8
9 - `min.c`
10 - stack/heap
11 - automatic variable
12 - `min-re.c`
13
14 - `fib-re.c`
15 - `fib-iter.c`
16
17 - `gcd-re.c`
18 - `gcd-iter.c`
19
20 - `bsearch-iter.c`
21 - `bsearch-re.c`
22
23 - `mergesort-re.c`
24
25 ## Backup
26
27 - `hanoi.c`
28 - `hilbert-curve-text.c`
29 - `n-queens.c`
30 - `quicksort.c`
```



```
1 // Created by hfwei on 2024/12/25.
2
3 #include <stdio.h>
4
5 // Fib(92) = 7 540 113 804 746 346 429
6 // long long: 9 223 372 036 854 775 807
7 // Fib(93) = 12 200 160 415 121 876 738
8 int main() {
9     int n;
10    scanf("%d", &n);
11
12    long long fib0 = 0L;
13    long long fib1 = 1L;
14
15    long long fib2 = 0L;
16    for (int i = 2; i <= n; i++) {
17        fib2 = fib0 + fib1;
18        fib0 = fib1;
19        fib1 = fib2;
20    }
21
22    printf("Fib(%d) = %lld ", n, fib2);
23
24    return 0;
25 }
```

```
1 // Created by hfwei on 2024/12/25.
2
3 // Euclidean algorithm:
4 // gcd(a, b) = gcd(b, a % b)
5
6 #include <stdio.h>
7
8 int GCD(int a, int b);
9
10 int main() {
11     int a = 130;
12     int b = 124;
13
14     printf("gcd(%d, %d) = %d\n", a, b, GCD(a, b));
15
16     return 0;
17 }
18
19 int GCD(int a, int b) {
20     while (b != 0) {
21         int temp = a;
22         a = b;
23         b = temp % b;
24     }
25
26     return a;
27 }
```

```
1 // Created by hfwei on 2024/10/25.
2 // Code generated by ChatGPT.
3
4 #include <stdbool.h>
5 #include <stdio.h>
6
7 #define N 8 // Change N as needed
8
9 // Function to print the board configuration
10 void printBoard(int board[][N]) {
11     for (int i = 0; i < N; i++) {
12         for (int j = 0; j < N; j++) {
13             printf("%s ", board[i][j] ? "Q" : ".");
14         }
15         printf("\n");
16     }
17     printf("\n");
18 }
19
20 // Function to check if placing a queen at board[row][col]
   is safe
21 bool isSafe(int board[][N], int row, int col) {
22     // Check column for conflicts
23     for (int i = 0; i < row; i++) {
24         if (board[i][col]) return false;
25     }
26
27     // Check upper left diagonal
28     for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
29         if (board[i][j]) return false;
30     }
31
32     // Check upper right diagonal
33     for (int i = row, j = col; i >= 0 && j < N; i--, j++) {
34         if (board[i][j]) return false;
35     }
36
37     return true;
38 }
39
40 // Recursive function to solve N-Queens
41 bool solveNQueens(int board[][N], int row) {
42     if (row >= N) {
43         printBoard(board);
```

```
44     return true;
45 }
46
47 bool res = false;
48 for (int col = 0; col < N; col++) {
49     if (isSafe(board, row, col)) {
50         board[row][col] = 1;
51         res = solveNQueens(board, row + 1) || res;
52         board[row][col] = 0;
53     }
54 }
55
56 return res;
57 }
58
59 int main() {
60     int board[N][N] = {0};
61
62     if (!solveNQueens(board, 0)) {
63         printf("Solution does not exist for %d-Queens.\n", N);
64     }
65
66     return 0;
67 }
```

```
1 // Created by hfwei on 2024/12/25.
2
3 #include <stdio.h>
4
5 #define LEN 93
6
7 int main() {
8     long long fibs[LEN] = {0LL, 1LL};
9
10    int n;
11    scanf("%d", &n);
12
13    for (int i = 2; i <= n; ++i) {
14        fibs[i] = fibs[i - 1] + fibs[i - 2];
15    }
16
17    printf("Fib(%d) = %lld\n", n, fibs[n]);
18
19    return 0;
20 }
```

```
1 // Created by hfwei on 2024/12/25.
2
3 // Visualization (search for 2 as an example):
4 // https://tinyurl.com/bsearch-re
5 // (Using https://tinyurl.com/)
6
7 #include <stdio.h>
8
9 #define LEN 10
10
11 int BinarySearch(int key, const int dict[], int low, int
    high);
12
13 int main() {
14     const int dictionary[LEN] = {0, 1, 1, 2, 3, 5, 8, 13, 21
        , 34};
15
16     int key;
17     scanf("%d", &key);
18
19     printf("The index of %d is %d.\n", key,
        BinarySearch(key, dictionary, 0, LEN - 1));
20
21
22     return 0;
23 }
24
25 int BinarySearch(int key, const int dict[], int low, int
    high) {
26     // if (low == high) {
27     //     if (dict[low] == key) {
28     //         return low;
29     //     }
30     //     return - 1;
31     // }
32
33     if (low > high) {
34         return -1;
35     }
36
37     int mid = (low + high) / 2;
38     if (dict[mid] == key) {
39         return mid;
40     }
41 }
```

```
42  if (dict[mid] > key) {  
43      return BinarySearch(key, dict, low, mid - 1);  
44  }  
45  
46  return BinarySearch(key, dict, mid + 1, high);  
47 }
```

```
1 // Created by hfwei on 2024/12/25.
2
3 #include <stdio.h>
4
5 #define LEN 10
6
7 // dictionary: out of any functions; global variables
8 // life time: program start to end
9 // scope: from this point on until the end of the file (
  file scope)
10 // int dictionary[LEN] = { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
    };
11
12 /**
13  * @brief Search for the key in the dict using the binary
    search algorithm.
14  * @param key the key to search for
15  * @param dict the dictionary to search
16  * @param len the length of the dictionary
17  * @return the index of the key in the dictionary; -1 if
    not found
18  */
19 int BinarySearch(int key, const int dict[100], int len);
20
21 int main(void) {
22     const int dictionary[LEN] = {0, 1, 1, 2, 3, 5, 8, 13, 21
    , 34};
23
24     int key = 0;
25     scanf("%d", &key);
26
27     int index = BinarySearch(key, dictionary, LEN);
28
29     if (index == -1) {
30         printf("Not found!\n");
31     } else {
32         printf("The index of %d is %d.\n", key, index);
33     }
34
35     return 0;
36 }
37
38 int BinarySearch(int key, const int dict[], int len) {
39     int low = 0;
```



```
40  int high = len - 1;
41
42  while (low <= high) {
43      int mid = (low + high) / 2;
44
45      if (key > dict[mid]) {
46          low = mid + 1;
47      } else if (key < dict[mid]) {
48          high = mid - 1;
49      } else { // key == dict[mid]
50          return mid;
51      }
52  }
53
54  return -1;
55 }
```

```
1 add_executable(main-re main-re.c)
2
3 add_executable(min min.c)
4
5 add_executable(min-re min-re.c)
6 add_executable(sum-re sum-re.c)
7
8 add_executable(fib-re fib-re.c)
9 add_executable(fib-iter fib-iter.c)
10 add_executable(fib-array fib-array.c)
11
12 add_executable(gcd-re gcd-re.c)
13 add_executable(gcd-iter gcd-iter.c)
14
15 add_executable(bsearch-iter bsearch-iter.c)
16 add_executable(bsearch-re bsearch-re.c)
17
18 add_executable(merge-array merge.c)
19 add_executable(mergesort-re mergesort-re.c)
20
21 add_executable(n-queens n-queens.c)
22
23 add_executable(hilbert-curve-text hilbert-curve-text.c)
24 #add_executable(hilbert-curve-ui hilbert-curve-ui.c)
```

```
1 // Created by hfwei on 2024/12/25.
2
3 // Visualizatin: https://tinyurl.com/mergesort-re (for LEN
  = 4)
4 // (Using https://tinyurl.com/)
5
6 #include <stdio.h>
7 #include <stdlib.h>
8
9 #define LEN 7
10
11 /**
12  * @brief sort nums[left .. right] using merge sort
13  * @param nums
14  * @param left
15  * @param right
16  */
17 void MergeSort(int nums[], int left, int right);
18
19 /**
20  * @brief merge nums[left .. mid] and nums[mid + 1 ..
  right]
21  * @param nums
22  * @param left
23  * @param mid
24  * @param right
25  */
26 void Merge(int nums[], int left, int mid, int right);
27
28 void Print(const int nums[], int len);
29
30 int main() {
31     int numbers[LEN] = {38, 27, 43, 3, 9, 82, 10};
32     Print(numbers, LEN);
33
34     // TODO: merge sort
35     MergeSort(numbers, 0, LEN - 1);
36     Print(numbers, LEN);
37
38     return 0;
39 }
40
41 void MergeSort(int nums[], int left, int right) {
42     if (left == right) {
```

```
43     return;
44 }
45
46 int mid = (left + right) / 2;
47 MergeSort(nums, left, mid);    // ask the Mirror
48 MergeSort(nums, mid + 1, right); // ask the Mirror
49
50 Merge(nums, left, mid, right);
51 }
52
53 void Merge(int nums[], int left, int mid, int right) {
54     int size = right - left + 1;
55     int *copy = malloc(size * sizeof *copy);
56
57     int left_index = left;
58     int right_index = mid + 1;
59
60     int copy_index = 0;
61     while (left_index <= mid && right_index <= right) {
62         if (nums[left_index] <= nums[right_index]) {
63             copy[copy_index] = nums[left_index];
64             left_index++;
65         } else {
66             copy[copy_index] = nums[right_index];
67             right_index++;
68         }
69
70         copy_index++;
71     }
72
73     while (left_index <= mid) {
74         copy[copy_index] = nums[left_index];
75         left_index++;
76         copy_index++;
77     }
78
79     while (right_index <= right) {
80         copy[copy_index] = nums[right_index];
81         right_index++;
82         copy_index++;
83     }
84
85     for (int i = 0; i < size; ++i) {
86         nums[i + left] = copy[i];
```

```
87     }
88
89     free(copy);
90 }
91
92 void Print(const int nums[], int len) {
93     for (int i = 0; i < len; i++) {
94         printf("%d ", nums[i]);
95     }
96     printf("\n");
97 }
```

```
1 // Created by hfwei on 2024/12/25.
2
3 // #include <SDL2/SDL.h>
4 // #include <math.h>
5 // #include <stdio.h>
6 //
7 // void drawHilbert(SDL_Renderer *renderer, int x, int y,
8 // int size, int level,
9 // int angle) {
10 //     if (level == 0) {
11 //         return;
12 //     }
13 //     // Draw the curve recursively
14 //     drawHilbert(renderer, x, y, size / 2, level - 1, -
15 // angle);
16 //     SDL_RenderDrawLine(renderer, x, y, x + size * cos(
17 // angle * M_PI / 180.0), y
18 // + size * sin(angle * M_PI / 180.0)); x += size * cos(
19 // angle * M_PI / 180.0);
20 //     y += size * sin(angle * M_PI / 180.0);
21 //     drawHilbert(renderer, x, y, size / 2, level - 1,
22 // angle);
23 //     SDL_RenderDrawLine(renderer, x, y, x + size * cos(
24 // angle * M_PI / 180.0), y
25 // + size * sin(angle * M_PI / 180.0)); x += size * cos(
26 // angle * M_PI / 180.0);
27 //     y += size * sin(angle * M_PI / 180.0);
28 //     drawHilbert(renderer, x, y, size / 2, level - 1, -
29 // angle);
30 // }
31 //
32 // int main(int argc, char *argv[]) {
33 //     SDL_Window *window;
34 //     SDL_Renderer *renderer;
35 //     int width = 800;
36 //     int height = 800;
```

```
34 //
35 //     SDL_Init(SDL_INIT_VIDEO);
36 //     window = SDL_CreateWindow("Hilbert Curve",
37 //                               SDL_WINDOWPOS_CENTERED,
38 //                               SDL_WINDOWPOS_CENTERED, width, height, 0); renderer =
39 //                               SDL_CreateRenderer(window, -1,
40 //                               SDL_RENDERER_ACCELERATED);
41 //     SDL_SetRenderDrawColor(renderer, 255, 255, 255); //
42 //     Set the background
43 //     color to white SDL_RenderClear(renderer);
44 //
45 //     int level = 5; // Change the level for more or fewer
46 //     iterations
47 //     drawHilbert(renderer, 100, 100, 400, level, 90); //
48 //     Start drawing the
49 //     Hilbert curve
50 //
51 //     SDL_RenderPresent(renderer);
52 //     SDL_Delay(5000); // Wait for 5 seconds before closing
53 //
54 //     SDL_DestroyRenderer(renderer);
55 //     SDL_DestroyWindow(window);
56 //     SDL_Quit();
57 //     return 0;
58 // }
```

```
1 // Created by hfwei on 2024/12/25.
2 // Code generated by ChatGPT.
3
4 #include <math.h>
5 #include <stdio.h>
6
7 void drawHilbert(int x, int y, int size, int level, int
    angle) {
8     if (level == 0) {
9         return;
10    }
11
12    // Store the initial coordinates
13    int x_start = x;
14    int y_start = y;
15
16    // Draw the curve recursively
17    drawHilbert(x, y, size / 2, level - 1, -angle);
18
19    // Calculate new coordinates after the first segment
20    x += size * cos(angle * M_PI / 180.0);
21    y += size * sin(angle * M_PI / 180.0);
22    printf("LINE %d %d %d %d\n", x_start, y_start, x, y);
23
24    drawHilbert(x, y, size / 2, level - 1, angle);
25
26    // Calculate new coordinates after the second segment
27    x_start = x;
28    x += size * cos(angle * M_PI / 180.0);
29    y += size * sin(angle * M_PI / 180.0);
30    printf("LINE %d %d %d %d\n", x_start, y_start, x, y);
31
32    drawHilbert(x, y, size / 2, level - 1, angle);
33
34    // Calculate new coordinates after the third segment
35    x_start = x;
36    x += size * cos(angle * M_PI / 180.0);
37    y += size * sin(angle * M_PI / 180.0);
38    printf("LINE %d %d %d %d\n", x_start, y_start, x, y);
39
40    drawHilbert(x, y, size / 2, level - 1, -angle);
41 }
42
43 int main(int argc, char *argv[]) {
```



```
44  int level = 5;      // Change the level for more or fewer
    iterations
45  int size = 400;     // Size of the curve segment
46  int startX = 100;  // Starting X coordinate
47  int startY = 100;  // Starting Y coordinate
48
49  printf("HILBERT CURVE OF LEVEL %d:\n", level);
50  drawHilbert(startX, startY, size, level,
51              90);    // Start drawing the Hilbert curve
52
53  return 0;
54 }
```