File - D:\cpl\2024-cpl-coding\13-recursion\min.c

```c
// Created by hfwei on 2024/12/25.

// Visualization of function call: https://tinyurl.com/min-func-call
// (Using https://tinyurl.com/)

#include <stdio.h>

int Min(int a, int b);

int main() {
  int a = 25;
  int b = 37;

  int min = Min(a, b);
  printf("%d", min);

  return 0;
}

int Min(int a, int b) { return a > b ? b : a; }
```

```c
// Created by hfwei on 2024/12/25.

#include <stdio.h>

#define LEN_L 5
#define LEN_R 6

int L[LEN_L] = {1, 3, 5, 7, 9};
int R[LEN_R] = {0, 2, 4, 6, 8, 10};

int main(void) {
  // TODO: merge L and R into a sorted array
  int l = 0;
  int r = 0;

  while (l < LEN_L && r < LEN_R) {
    if (L[l] <= R[r]) {
      printf("%d ", L[l]);
      l++;
    } else {
      printf("%d ", R[r]);
      r++;
    }
  }

  // l >= LEN_L || r >= LEN_R
  while (r < LEN_R) {
    printf("%d ", R[r]);
    r++;
  }

  while (l < LEN_L) {
    printf("%d ", L[l]);
    l++;
  }

  return 0;
}
```

File - D:\cpl\2024-cpl-coding\13-recursion\fib-re.c

```c
// Created by hfwei on 2024/12/25.

// Visualization (for n = 4): https://tinyurl.com/fib-re-visual
// (Using https://tinyurl.com/)

#include <stdio.h>

long long Fib(int n);

int main() {
  int n;
  scanf("%d", &n);

  printf("Fib(%d) = %lld\n", n, Fib(n));

  return 0;
}

long long Fib(int n) {
  if (n <= 1) {
    return n;
  }

  return Fib(n - 1) + Fib(n - 2);
}
```

```c
// Created by hfwei on 2024/12/25.

//  Euclidean algorithm:
//  gcd(a, b) = gcd(b, a % b)
//
//  Visualization (gcd(64, 48) for illustration):
// https://tinyurl.com/gcd-re-visual
// (Using https://tinyurl.com/)

#include <stdio.h>

int GCD(int a, int b);

int main() {
  int a = 0;
  int b = 0;
  scanf("%d %d", &a, &b);

  printf("GCD(%d, %d) = %d\n", a, b, GCD(a, b));

  return 0;
}

// gcd(130, 124) = 2
// gcd(414, 662) = 2
int GCD(int a, int b) {
  if (b == 0) {
    return a;
  }

  return GCD(b, a % b);
}
```

```c
// Created by hfwei on 2024/12/25.

// Visualization: https://tinyurl.com/min-re
// (Using https://tinyurl.com/)

#include <stdio.h>

#define NUM 3
const int numbers[NUM] = {65, 28, 37};

int Min(const int nums[], int len);

int main() {
  int min = Min(numbers, NUM);

  printf("min = %d\n", min);

  return 0;
}

int Min(const int nums[], int len) {
  if (len == 1) {
    return nums[0];
  }

  int partial_min = Min(nums, len - 1);

  return partial_min < nums[len - 1] ? partial_min : nums[len - 1];
}
```

```c
// Created by hfwei on 2024/10/25.

// Visualization: https://tinyurl.com/sum-re
// (Using https://tinyurl.com/)

#include <stdio.h>

int Sum(const int nums[], int len);

int main() {
  const int numbers[] = {1, 2, 3, 4, 5};

  int sum = Sum(numbers, sizeof numbers / sizeof numbers[0]);

  printf("sum = %d\n", sum);

  return 0;
}

int Sum(const int nums[], int len) {
  if (len == 0) {
    return 0;
  }

  int partial_sum = Sum(nums, len - 1);

  return partial_sum + nums[len - 1];
}
```

```c
// Created by hfwei on 2024/12/25.
//
// WARNING: You can even call the "main" function in itself.
// But, do NOT write code like this.
// Never call the "main" function in your own code.

#include <stdio.h>

int main(int argc, char *argv[]) {
  if (argc == 1) {
    return 0;
  }

  printf("%s\n", argv[argc - 1]);

  main(argc - 1, argv);

  return 0;
}
```

```markdown
# `6-recursion`

# 6-recursion

## Recursion

- `main-re.c`

- `min.c`
  - stack/heap
  - automatic variable
- `min-re.c`

- `fib-re.c`
- `fib-iter.c`

- `gcd-re.c`
- `gcd-iter.c`

- `bsearch-iter.c`
- `bsearch-re.c`

- `mergesort-re.c`

## Backup

- `hanoi.c`
- `hilbert-curve-text.c`
- `n-queens.c`
- `quicksort.c`
```

```c
// Created by hfwei on 2024/12/25.

#include <stdio.h>

// Fib(92) =  7 540 113 804 746 346 429
// long long: 9 223 372 036 854 775 807
// Fib(93) = 12 200 160 415 121 876 738
int main() {
  int n;
  scanf("%d", &n);

  long long fib0 = 0L;
  long long fib1 = 1L;

  long long fib2 = 0L;
  for (int i = 2; i <= n; i++) {
    fib2 = fib0 + fib1;
    fib0 = fib1;
    fib1 = fib2;
  }

  printf("Fib(%d) = %lld ", n, fib2);

  return 0;
}
```

```c
// Created by hfwei on 2024/12/25.

// Euclidean algorithm:
// gcd(a, b) = gcd(b, a % b)

#include <stdio.h>

int GCD(int a, int b);

int main() {
  int a = 130;
  int b = 124;

  printf("gcd(%d, %d) = %d\n", a, b, GCD(a, b));

  return 0;
}

int GCD(int a, int b) {
  while (b != 0) {
    int temp = a;
    a = b;
    b = temp % b;
  }

  return a;
}
```

```c
// Created by hfwei on 2024/10/25.
// Code generated by ChatGPT.

#include <stdbool.h>
#include <stdio.h>

#define N 8  // Change N as needed

// Function to print the board configuration
void printBoard(int board[][N]) {
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      printf("%s ", board[i][j] ? "Q" : ".");
    }
    printf("\n");
  }
  printf("\n");
}

// Function to check if placing a queen at board[row][col] is safe
bool isSafe(int board[][N], int row, int col) {
  // Check column for conflicts
  for (int i = 0; i < row; i++) {
    if (board[i][col]) return false;
  }

  // Check upper left diagonal
  for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
    if (board[i][j]) return false;
  }

  // Check upper right diagonal
  for (int i = row, j = col; i >= 0 && j < N; i--, j++) {
    if (board[i][j]) return false;
  }

  return true;
}

// Recursive function to solve N-Queens
bool solveNQueens(int board[][N], int row) {
  if (row >= N) {
    printBoard(board);
    return true;
  }

  bool res = false;
  for (int col = 0; col < N; col++) {
    if (isSafe(board, row, col)) {
      board[row][col] = 1;
      res = solveNQueens(board, row + 1) || res;
      board[row][col] = 0;
    }
  }

  return res;
}

int main() {
  int board[N][N] = {0};

  if (!solveNQueens(board, 0)) {
    printf("Solution does not exist for %d-Queens.\n", N);
  }

  return 0;
}
```

```c
// Created by hfwei on 2024/12/25.

#include <stdio.h>

#define LEN 93

int main() {
  long long fibs[LEN] = {0LL, 1LL};

  int n;
  scanf("%d", &n);

  for (int i = 2; i <= n; ++i) {
    fibs[i] = fibs[i - 1] + fibs[i - 2];
  }

  printf("Fib(%d) = %lld\n", n, fibs[n]);

  return 0;
}
```

```c
// Created by hfwei on 2024/12/25.

// Visualization (search for 2 as an example):
// https://tinyurl.com/bsearch-re
// (Using https://tinyurl.com/)

#include <stdio.h>

#define LEN 10

int BinarySearch(int key, const int dict[], int low, int high);

int main() {
  const int dictionary[LEN] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34};

  int key;
  scanf("%d", &key);

  printf("The index of %d is %d.\n", key,
         BinarySearch(key, dictionary, 0, LEN - 1));

  return 0;
}

int BinarySearch(int key, const int dict[], int low, int high) {
  // if (low == high) {
  //   if (dict[low] == key) {
  //     return low;
  //   }
  //   return - 1;
  // }

  if (low > high) {
    return -1;
  }

  int mid = (low + high) / 2;
  if (dict[mid] == key) {
    return mid;
  }

  if (dict[mid] > key) {
    return BinarySearch(key, dict, low, mid - 1);
  }

  return BinarySearch(key, dict, mid + 1, high);
}
```

```c
// Created by hfwei on 2024/12/25.

#include <stdio.h>

#define LEN 10

// dictionary: out of any functions; global variables
// life time: program start to end
// scope: from this point on until the end of the file (file scope)
// int dictionary[LEN] = { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 };

/**
 * @brief Search for the key in the dict using the binary search algorithm.
 * @param key the key to search for
 * @param dict the dictionary to search
 * @param len the length of the dictionary
 * @return the index of the key in the dictionary; -1 if not found
 */
int BinarySearch(int key, const int dict[100], int len);

int main(void) {
  const int dictionary[LEN] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34};

  int key = 0;
  scanf("%d", &key);

  int index = BinarySearch(key, dictionary, LEN);

  if (index == -1) {
    printf("Not found!\n");
  } else {
    printf("The index of %d is %d.\n", key, index);
  }

  return 0;
}

int BinarySearch(int key, const int dict[], int len) {
  int low = 0;
  int high = len - 1;

  while (low <= high) {
    int mid = (low + high) / 2;

    if (key > dict[mid]) {
      low = mid + 1;
    } else if (key < dict[mid]) {
      high = mid - 1;
    } else {  // key == dict[mid]
      return mid;
    }
  }

  return -1;
}
```

```
add_executable(main-re main-re.c)

add_executable(min min.c)

add_executable(min-re min-re.c)
add_executable(sum-re sum-re.c)

add_executable(fib-re fib-re.c)
add_executable(fib-iter fib-iter.c)
add_executable(fib-array fib-array.c)

add_executable(gcd-re gcd-re.c)
add_executable(gcd-iter gcd-iter.c)

add_executable(bsearch-iter bsearch-iter.c)
add_executable(bsearch-re bsearch-re.c)

add_executable(merge-array merge.c)
add_executable(mergesort-re mergesort-re.c)

add_executable(n-queens n-queens.c)

add_executable(hilbert-curve-text hilbert-curve-text.c)
#add_executable(hilbert-curve-ui hilbert-curve-ui.c)
```

```c
// Created by hfwei on 2024/12/25.

// Visualizatin: https://tinyurl.com/mergesort-re (for LEN = 4)
// (Using https://tinyurl.com/)

#include <stdio.h>
#include <stdlib.h>

#define LEN 7

/**
 * @brief sort nums[left .. right] using merge sort
 * @param nums
 * @param left
 * @param right
 */
void MergeSort(int nums[], int left, int right);

/**
 * @brief merge nums[left .. mid] and nums[mid + 1 .. right]
 * @param nums
 * @param left
 * @param mid
 * @param right
 */
void Merge(int nums[], int left, int mid, int right);

void Print(const int nums[], int len);

int main() {
  int numbers[LEN] = {38, 27, 43, 3, 9, 82, 10};
  Print(numbers, LEN);

  // TODO: merge sort
  MergeSort(numbers, 0, LEN - 1);
  Print(numbers, LEN);

  return 0;
}

void MergeSort(int nums[], int left, int right) {
  if (left == right) {
    return;
  }

  int mid = (left + right) / 2;
  MergeSort(nums, left, mid);        // ask the Mirror
  MergeSort(nums, mid + 1, right);  // ask the Mirror

  Merge(nums, left, mid, right);
}

void Merge(int nums[], int left, int mid, int right) {
  int size = right - left + 1;
  int *copy = malloc(size * sizeof *copy);

  int left_index = left;
  int right_index = mid + 1;

  int copy_index = 0;
  while (left_index <= mid && right_index <= right) {
    if (nums[left_index] <= nums[right_index]) {
      copy[copy_index] = nums[left_index];
      left_index++;
    } else {
      copy[copy_index] = nums[right_index];
      right_index++;
    }

    copy_index++;
  }

  while (left_index <= mid) {
    copy[copy_index] = nums[left_index];
    left_index++;
    copy_index++;
  }

  while (right_index <= right) {
    copy[copy_index] = nums[right_index];
    right_index++;
    copy_index++;
  }

  for (int i = 0; i < size; ++i) {
    nums[i + left] = copy[i];
  }

  free(copy);
}

void Print(const int nums[], int len) {
  for (int i = 0; i < len; i++) {
    printf("%d ", nums[i]);
  }
  printf("\n");
}
```

```c
// Created by hfwei on 2024/12/25.

// #include <SDL2/SDL.h>
// #include <math.h>
// #include <stdio.h>
//
// void drawHilbert(SDL_Renderer *renderer, int x, int y, int size, int level,
// int angle) {
//   if (level == 0) {
//     return;
//   }
//
//   // Draw the curve recursively
//   drawHilbert(renderer, x, y, size / 2, level - 1, -angle);
//   SDL_RenderDrawLine(renderer, x, y, x + size * cos(angle * M_PI / 180.0), y
//   + size * sin(angle * M_PI / 180.0)); x += size * cos(angle * M_PI / 180.0);
//   y += size * sin(angle * M_PI / 180.0);
//   drawHilbert(renderer, x, y, size / 2, level - 1, angle);
//   SDL_RenderDrawLine(renderer, x, y, x + size * cos(angle * M_PI / 180.0), y
//   + size * sin(angle * M_PI / 180.0)); x += size * cos(angle * M_PI / 180.0);
//   y += size * sin(angle * M_PI / 180.0);
//   drawHilbert(renderer, x, y, size / 2, level - 1, angle);
//   SDL_RenderDrawLine(renderer, x, y, x + size * cos(angle * M_PI / 180.0), y
//   + size * sin(angle * M_PI / 180.0)); x += size * cos(angle * M_PI / 180.0);
//   y += size * sin(angle * M_PI / 180.0);
//   drawHilbert(renderer, x, y, size / 2, level - 1, -angle);
// }
//
// int main(int argc, char *argv[]) {
//   SDL_Window *window;
//   SDL_Renderer *renderer;
//   int width = 800;
//   int height = 800;
//
//   SDL_Init(SDL_INIT_VIDEO);
//   window = SDL_CreateWindow("Hilbert Curve", SDL_WINDOWPOS_CENTERED,
//   SDL_WINDOWPOS_CENTERED, width, height, 0); renderer =
//   SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
//   SDL_SetRenderDrawColor(renderer, 255, 255, 255); // Set the background
//   color to white SDL_RenderClear(renderer);
//
//   int level = 5; // Change the level for more or fewer iterations
//   drawHilbert(renderer, 100, 100, 400, level, 90); // Start drawing the
//   Hilbert curve
//
//   SDL_RenderPresent(renderer);
//   SDL_Delay(5000); // Wait for 5 seconds before closing
//
//   SDL_DestroyRenderer(renderer);
//   SDL_DestroyWindow(window);
//   SDL_Quit();
//   return 0;
// }
```

```c
// Created by hfwei on 2024/12/25.
// Code generated by ChatGPT.

#include <math.h>
#include <stdio.h>

void drawHilbert(int x, int y, int size, int level, int angle) {
  if (level == 0) {
    return;
  }

  // Store the initial coordinates
  int x_start = x;
  int y_start = y;

  // Draw the curve recursively
  drawHilbert(x, y, size / 2, level - 1, -angle);

  // Calculate new coordinates after the first segment
  x += size * cos(angle * M_PI / 180.0);
  y += size * sin(angle * M_PI / 180.0);
  printf("LINE %d %d %d %d\n", x_start, y_start, x, y);

  drawHilbert(x, y, size / 2, level - 1, angle);

  // Calculate new coordinates after the second segment
  x_start = x;
  x += size * cos(angle * M_PI / 180.0);
  y += size * sin(angle * M_PI / 180.0);
  printf("LINE %d %d %d %d\n", x_start, y_start, x, y);

  drawHilbert(x, y, size / 2, level - 1, angle);

  // Calculate new coordinates after the third segment
  x_start = x;
  x += size * cos(angle * M_PI / 180.0);
  y += size * sin(angle * M_PI / 180.0);
  printf("LINE %d %d %d %d\n", x_start, y_start, x, y);

  drawHilbert(x, y, size / 2, level - 1, -angle);
}

int main(int argc, char *argv[]) {
  int level = 5;     // Change the level for more or fewer iterations
  int size = 400;    // Size of the curve segment
  int startX = 100;  // Starting X coordinate
  int startY = 100;  // Starting Y coordinate

  printf("HILBERT CURVE OF LEVEL %d:\n", level);
  drawHilbert(startX, startY, size, level,
              90);  // Start drawing the Hilbert curve

  return 0;
}
```