```c
// Created by hfwei on 2024/10/23.

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

bool IsLeapYear(int year);

int main(void) {
  // local variable
  // scope: block scope
  int year = 0;
  scanf("%d", &year);

  // caller
  // callee
  bool leap = IsLeapYear(year);

  if (leap) {
    printf("%d is a leap year\n", year);
  } else {
    printf("%d is a common year\n", year);
  }

  return 0;
}

// year: formal parameter
// block scope
bool IsLeapYear(int year) {
  // local variable
  // scope: block scope
  bool leap = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
  return leap;
}
```

```c
// Created by hfwei on 2024/10/23.

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

int main(void) {
  int high = 100;
  int number_of_tries = 7;

  /*
   * (1) print the rules of the game to players
   */
  printf("Let us play the Guess the Number game.\n"
         "The computer will generate a random number between 1 and %d.\n"
         "You have %d tries.\n",
         high, number_of_tries);

  /*
   * (2) generate a random number between 1 and high
   */
  srand(time(NULL));
  int secret = rand() % high + 1;
  printf("Test: secret = %d\n", secret);

  while (number_of_tries > 0) {
    /*
     * (3) ask the player to enter his/her guess
     */
    printf("Please enter your guess number.\n"
           "You still have %d tries.\n", number_of_tries);

    /*
     * (4) obtain the guessed number, compare it with the secret number,
     *     and inform the player of the result.
     */
    int guess = 0;
    scanf("%d", &guess);
    printf("Test: guess = %d\n", guess);

    if (guess == secret) {
      printf("You Win!\n");
```

```c
      break;
    } else if (guess > secret) {
      printf("guess > secret\n");
    } else {
      printf("guess < secret\n");
    }

    /*
     * (5) repeat (3) and (4) until the player wins or loses.
     */
    number_of_tries--;

    if (number_of_tries == 0) {
      printf("You Lose!\n");
    }
  }

  return 0;
}
```

```c
// Created by hfwei on 2024/10/23.

#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 };

void Merge(const int left[], int left_len,
           const int right[], int right_len);

int main(void) {
  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++;
    }
  }

  while (r < LEN_R) {
    printf("%d ", R[r]);
    r++;
  }

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

  return 0;
}
```

```c
// Created by hfwei on 2024/10/23.

#include <stdio.h>

void Print(char ch, int count);
void NewLine(void);

int main(void) {
  int lines = 0;
  scanf("%d", &lines);

  for (int i = 0; i < lines; ++i) {
    Print(' ', lines - 1 - i);
    Print('*', 2 * i + 1);

    if (i < lines - 1) {
      printf("\n");
      NewLine();
    }
  }

  return 0;
}

void Print(char ch, int count) {
  for (int i = 0; i < count; ++i) {
    printf("%c", ch);
  }
}

void NewLine(void) {
  printf("\n");
}
```

```c
// Created by hfwei on 2024/10/23.

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

bool IsPrime(int number);

int main(void) {
  int max = 0;
  scanf("%d", &max);

  int count = 0;

  for (int number = 2; number <= max; number++) {
    if (IsPrime(number)) {
      count++;
      printf("%d ", number);
    }
  }

  printf("\ncount = %d\n", count);

  return 0;
}

bool IsPrime(int number) {
  for (int factor = 2; factor * factor <= number; factor
++) {
    if (number % factor == 0) {
      return false;
    }
  }

  return true;
}
```

```c
// Created by hfwei on 2024/10/23.

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

#define LEN 21
char string[LEN] = "";

bool IsPalindrome(const char str[]);

int main() {
  printf("Input a string containing at most 20 characters.\n");
  scanf("%20s", string);

  printf("\"%s\" is %s a palindrome.\n", string,
         IsPalindrome(string) ? "" : "not");

  return 0;
}

bool IsPalindrome(const char str[]) {
  int len = strlen(str);

  for (int i = 0, j = len - 1; i < j; i++, j--) {
    if (str[i] != str[j]) {
      return false;
    }
  }

  return true;
}
```

```
# 0-intro
add_executable(guess-func guess.c)

# 2-if-for-array
add_executable(leap-func leap.c)

# 3-for-a-while
add_executable(stars-func stars.c)
add_executable(primes-func primes.c)
add_executable(binary-search-func binary-search.c)
add_executable(palindrome-func palindrome.c)
add_executable(selection-sort-func selection-sort.c)

# 4-loops
add_executable(merge-func merge.c)
add_executable(game-of-life-func game-of-life.c)
add_executable(game-of-life-transformed game-of-life-
transformed.c)

add_executable(insertion-sort-func insertion-sort.c)
add_executable(binary-insertion-sort-func binary-insertion
-sort.c)
```

```c
// Created by hfwei on 2024/10/23.

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

#define SIZE 6
const int board[SIZE][SIZE] = {
    { 0 },
    { 0, 1, 1, 0, 0, 0 },
    { 0, 1, 1, 0, 0, 0 },
    { 0, 0, 0, 1, 1, 0 },
    { 0, 0, 0, 1, 1, 0 },
    { 0 }
};

//const int board[SIZE][SIZE] = {
//    [1][1] = 1, [1][2] = 1,
//    [2][1] = 1, [2][2] = 1,
//    [3][3] = 1, [3][4] = 1,
//    [4][3] = 1, [4][4] = 1
//};

int main() {
  // extended board
  int old_board[SIZE + 2][SIZE + 2] = { 0 };

  for (int row = 1; row <= SIZE; row++) {
    for (int col = 1; col <= SIZE; col++) {
      old_board[row][col] = board[row - 1][col - 1];
    }
  }

  // print the original board
  for (int row = 1; row <= SIZE; row++) {
    for (int col = 1; col <= SIZE; col++) {
      printf("%c ", old_board[row][col] ? '*' : ' ');
    }
    printf("\n");
  }
  system("clear");  // clear the screen/terminal

  int new_board[SIZE + 2][SIZE + 2] = { 0 };
```

```c

  for (int round = 1; round < 10; round++) {
    for (int row = 1; row <= SIZE; row++) {
      for (int col = 1; col <= SIZE; col++) {
        // count the number of neighbours of old_board[row][col]
        int neighbours =
            old_board[row - 1][col - 1] +
                old_board[row - 1][col] +
                old_board[row - 1][col + 1] +
                old_board[row][col - 1] +
                old_board[row][col + 1] +
                old_board[row + 1][col - 1] +
                old_board[row + 1][col] +
                old_board[row + 1][col + 1];

        // evaluate the new board
        if (old_board[row][col]) {  // old_board[row][col] is alive
          new_board[row][col] = (neighbours == 2 || neighbours == 3);
        } else {  // old_board[row][col] is dead
          new_board[row][col] = (neighbours == 3);
        }
      }
    }

    // print the new board
    for (int row = 1; row <= SIZE; row++) {
      for (int col = 1; col <= SIZE; col++) {
        printf("%c ", new_board[row][col] ? '*' : ' ');
      }
      printf("\n");
    }

    // sleep for a while
    // Linux: #include <unistd.h>
    sleep(1);
    // Windows: #include <windows.h>: Sleep(ms)
    // Sleep(1000);

    // clear the screen
    // Linux: #include <stdlib.h>
    system("clear");
```

```c
    // Windows: #include <stdlib.h> system("clr");
    // system("clr");

    // start the next round
    for (int row = 1; row <= SIZE; row++) {
      for (int col = 1; col <= SIZE; col++) {
        old_board[row][col] = new_board[row][col];
      }
    }
  }

  return 0;
}
```

```c
// Created by hfwei on 2024/10/23.

#include <stdio.h>

#define LEN 10

// global variable
// file scope
// const int dictionary[LEN] = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 };

// int dict[]: the address of the first element of the array
int BinarySearch(int key, const int dict[], int len);

int main(void) {
  const int dictionary[LEN] = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 };

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

  // dictionary (actual argument): const int[]
  // dict (formal parameter): int[]
  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]) {
```

```
42        high = mid - 1;
43      } else {  // key == dict[mid]
44        return mid;
45      }
46    }
47
48    return -1;
49  }
```

```c
// Created by hfwei on 2024/10/16.
// Code generated by ChatGPT.

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

#define MAX_LEN 10000
#define RANGE 10

int main(void) {
  int numbers[MAX_LEN] = { 0 };

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

  // generate an array of random integers between 0 and RANGE - 1
  srand(time(NULL));
  for (int i = 0; i < size; i++) {
    numbers[i] = rand() % RANGE;
  }

  // print the original array
  for (int i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  // TODO: insertion sort
  for (int i = 1; i < size; i++) {
    // numbers[0 .. i - 1] is already sorted
    int key = numbers[i];

    int j = i - 1;
    while (j >= 0 && numbers[j] > key) {
      numbers[j + 1] = numbers[j];
      j--;
    }
    numbers[j + 1] = key;

    // numbers[0 .. i] is already sorted
    for (int i = 0; i < size; i++) {
      printf("%d ", numbers[i]);
```

```c
    }
    printf("\n");
  }
  // i = size
  // numbers[0 .. size - 1] is already sorted

  // print the sorted array
  for (int i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  return 0;
}
```

```c
// Created by hfwei on 2024/10/23.

#include <stdio.h>

#define LEN 20
int numbers[LEN] = { 0 };

void SelectionSort(int arr[], int len);
int GetMinIndex(const int arr[], int begin, int end);
void Swap(int left, int right);
void Print(const int arr[], int len);

int main(void) {
  int len = -1;
  while (scanf("%d", &numbers[++len]) != EOF);

  Print(numbers, len);
  SelectionSort(numbers, len);
  Print(numbers, len);

  return 0;
}

void SelectionSort(int arr[], int len) {
  for (int i = 0; i < len; i++) {
    int min_index = GetMinIndex(arr, i, len);

    // swap arr[i] and arr[min_index]
    // Swap(left, right)
    // arr[i] = 3   arr[min_index] = 5
    // left = 3   right = 5
    // Swap(arr[i], arr[min_index]);

    int temp = arr[i];
    arr[i] = arr[min_index];
    arr[min_index] = temp;
  }
}

int GetMinIndex(const int arr[], int begin, int end) {
  int min = arr[begin];
  int min_index = begin;

  for (int j = begin + 1; j <= end - 1; ++j) {
```

```c
    if (arr[j] < min) {
      min = arr[j];
      min_index = j;
    }
  }

  return min_index;
}

// left = 3   right = 5
void Swap(int left, int right) {
  int temp = left;
  left = right;
  right = temp;
  // left = 5 right = 3
}

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_LEN 10000
#define RANGE 10

int main() {
  int numbers[MAX_LEN] = { 0 };

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

  srand(time(NULL));
  for (int i = 0; i < size; i++) {
    numbers[i] = rand() % RANGE;
  }

  // print the original array
  for (int i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  // TODO
  for (int i = 1; i < size; i++) {
    int key = numbers[i];
    int low = 0;
    int high = i - 1;

    while (low <= high) {
      int mid = (low + high) / 2;
      if (key >= numbers[mid]) {
        low = mid + 1;
      } else {
        high = mid - 1;
      }
    }

    for (int j = i - 1; j >= low; j--) {
      numbers[j + 1] = numbers[j];
    }
    numbers[low] = key;

```

```c
    for (int i = 0; i < size; i++) {
      printf("%d ", numbers[i]);
    }
    printf("\n");
  }

  // Print the sorted array
  for (int i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  return 0;
}
```

```c
// Created by hfwei on 2024/10/23.
// Run it with "Terminal"

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

#define SIZE 6

void ExtendBoard(const int origin_board[][SIZE],
                 int extended_board[][SIZE + 2]);
void PrintExtendedBoard(const int extended_board[][SIZE + 2]);
void GenerateNewBoard(const int old_board[][SIZE + 2],
                      int new_board[][SIZE + 2]);
void CopyExtendedBoard(const int src_board[][SIZE + 2],
                       int dest_board[][SIZE + 2]);
void SleepAndClear(int sec);

int main() {
  const int board[SIZE][SIZE] = {
      { 0 },
      { 0, 1, 1, 0, 0, 0 },
      { 0, 1, 1, 0, 0, 0 },
      { 0, 0, 0, 1, 1, 0 },
      { 0, 0, 0, 1, 1, 0 },
      { 0 }
  };

  int old_board[SIZE + 2][SIZE + 2] = { 0 };
  ExtendBoard(board, old_board);
  PrintExtendedBoard(old_board);
  SleepAndClear(1);

  int new_board[SIZE + 2][SIZE + 2] = { 0 };
  for (int round = 0; round < 10; round++) {
    GenerateNewBoard(old_board, new_board);
    SleepAndClear(1);
    PrintExtendedBoard(new_board);
    CopyExtendedBoard(new_board, old_board);
  }

  return 0;
}
```

```c

void ExtendBoard(const int origin_board[][SIZE],
                 int extended_board[][SIZE + 2]) {
  for (int row = 1; row <= SIZE; row++) {
    for (int col = 1; col <= SIZE; col++) {
      extended_board[row][col] = origin_board[row - 1][col - 1];
    }
  }
}

void PrintExtendedBoard(const int extended_board[][SIZE + 2]) {
  for (int row = 1; row <= SIZE; row++) {
    for (int col = 1; col <= SIZE; col++) {
      printf("%c ", extended_board[row][col] ? '*' : ' ');
    }
    printf("\n");
  }
}

void GenerateNewBoard(const int old_board[][SIZE + 2],
                      int new_board[][SIZE + 2]) {
  for (int row = 1; row <= SIZE; row++) {
    for (int col = 1; col <= SIZE; col++) {
      // count the number of neighbours of old_board[row][col]
      int neighbours =
          old_board[row - 1][col - 1] +
              old_board[row - 1][col] +
              old_board[row - 1][col + 1] +
              old_board[row][col - 1] +
              old_board[row][col + 1] +
              old_board[row + 1][col - 1] +
              old_board[row + 1][col] +
              old_board[row + 1][col + 1];

      // evaluate the new board
      if (old_board[row][col]) {  // old_board[row][col] is alive
        new_board[row][col] = (neighbours == 2 || neighbours == 3);
      } else {  // old_board[row][col] is dead
```

```c
        new_board[row][col] = (neighbours == 3);
      }
    }
  }
}

void CopyExtendedBoard(const int src_board[][SIZE + 2],
                       int dest_board[][SIZE + 2]) {
  for (int row = 1; row <= SIZE; row++) {
    for (int col = 1; col <= SIZE; col++) {
      dest_board[row][col] = src_board[row][col];
    }
  }
}

void SleepAndClear(int sec) {
  sleep(sec);
  system("clear");
}
```