```c
//
// Created by hfwei on 2023/10/6.
//

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

bool IsLeapYear(int year);

int main(void) {
  // year: in the main function: local variable
  // life time:
  // scope: from this point on until the main function exits
  int year = 0;
  scanf("%d", &year);

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

  return 0;
}

// year: parameter
// life time
// scope as a local variable
bool IsLeapYear(int year) {
  return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}
```

```c
//
// Created by hfwei on 2023/10/11.
//

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

/**
 * @brief decide whether number is a prime
 * @param number the number to decide
 * @return true if number is prime; false otherwise
 */
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 2023/10/11.
//

#include <stdio.h>

void Print(char ch, int count);

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");
    }
  }

  return 0;
}

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

```c
//
// Created by hengxin on 9/14/23.
//

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

```

```c
    /*
     * (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 2023/10/11.
//

#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] = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 };

/**
 * @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] = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 };

  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;
```

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

```c
//
// Created by hfwei on 2023/10/12.
//

#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;
}
```

```c
//
// Created by hfwei on 2023/10/19.
//

#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) {
  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++;
  }

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

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

  return 0;
}
```

```c
//
// Created by hfwei on 2023/10/12.
//

#include <stdio.h>

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

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

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

  // sizeof numbers / sizeof(numbers[0])
  Print(numbers, len);
  SelectionSort(numbers, len);
  Print(numbers, len);

  return 0;
}

// arr: the (copy of the) address of the first element of the `numbers` array
void SelectionSort(int arr[], int len) {
  for (int i = 0; i < len; i++) {
    int min_index = GetMinIndex(arr, i, len);

    // ERROR: 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 i = begin + 1; i < end; ++i) {
    if (arr[i] < min) {
      min = arr[i];
      min_index = i;
    }
  }

  return min_index;
}

```

```c
void Swap(int left, int right) {
  int temp = left;
  left = right;
  right = temp;
}

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

```c
//
// Created by hfwei on 2023/10/19.
//

#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 };

  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] +
```

```c
                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");
    // 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 hengxin on 10/19/22.
// Run it with "Terminal"
//

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

#define SIZE 6

// extended_board as a parameter
// 1D array: (int, the address of the first element), the length
// 2D array: array of arrays [1][3]:
// address + 1 * (sizeof int) * size of col + (sizeof int) * 3
// 0: ------
// 1: ------
// 2: ------
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 = 0; row < SIZE + 2; row++) {
    for (int col = 0; col < SIZE + 2; col++) {
      if (row == 0 || row == SIZE + 1 || col == 0 || col == SIZE + 1
) {
        extended_board[row][col] = 0;
      } else {
        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
        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];
```

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