```c
// Created by hfwei on 2024/10/6.

#include <stdio.h>

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

  int leap = 0;

  // TODO: leap year or not (logical expressions)
  if ((year % 4 == 0 && year % 100 != 0) ||
        year % 400 == 0) {
    leap = 1;
  }

  if (leap == 0) {
    printf("%d is a common year\n", year);
  } else {
    printf("%d is a leap year\n", year);
  }

  return 0;
}
```

```c
// Created by hfwei on 2024/10/16.

#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++;
    }
  }

  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 2024/10/10.

#include <stdio.h>

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

  // TODO: print stars pyramid
  for (int i = 0; i < lines; i++) {
    // print n - 1 - i spaces
    for (int j = 0; j < lines - 1 - i; ++j) {
      printf(" ");
    }

    // print 2 * i + stars
    for (int j = 0; j < 2 * i + 1; ++j) {
      printf("*");
    }

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

  return 0;
}
```

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

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

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

  int count = 0;

  // TODO: print primes between 1 and max
  for (int i = 2; i <= max; i++) {
    bool is_prime = true;

    for (int j = 2; j * j <= i; j++) {
      // if j is a factor of i
      if (i % j == 0) {
        is_prime = false;
        break;
      }
    }

    if (is_prime) {
      count++;
    }
  }

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

  return 0;
}
```

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

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

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

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

  int len = strlen(string);
  printf("The length of \"%s\" is %d.\n", string, len);

  // TODO: test palindrome
  bool is_palindrome = true;

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

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

  return 0;
}
```

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

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

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)
```

```c
// Created by hfwei on 2024/10/16.

#include <stdio.h>
#include <unistd.h>
#include <stdlib.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 board1[SIZE][SIZE] = {
//    [0] = { 0 },
//    [1] = { [1] = 1, [2] = 1 },
//    [2] = { [1] = 1, [2] = 1 },
//    [3] = { [3] = 1, [4] = 1 },
//    [4] = { [3] = 1, [4] = 1 },
//};

//const int board2[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
//};

//const int board3[SIZE][SIZE] = {
//    0, 0, 0, 0, 0, 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,
//};

int main(void) {
  // TODO: play game-of-life

  // expand this board
```

```c
  int old_board[SIZE + 2][SIZE + 2] = {0};
  for (int i = 1; i <= SIZE; i++) {
    for (int j = 1; j <= SIZE; j++) {
      old_board[i][j] = board[i - 1][j - 1];
    }
  }

  // print the old_board
  for (int i = 1; i <= SIZE; i++) {
    for (int j = 1; j <= SIZE; j++) {
      printf("%c ", old_board[i][j] ? '*' : ' ');
    }
    printf("\n");
  }
  system("clear");

  // count live neighbors for each cell
  int new_board[SIZE + 2][SIZE + 2] = {0};

  for (int i = 0; i < 10; ++i) {
    for (int i = 1; i <= SIZE; i++) {
      for (int j = 1; j <= SIZE; j++) {
        int num_of_live_neighbors =
            old_board[i - 1][j - 1] +
            old_board[i - 1][j] +
            old_board[i - 1][j + 1] +
            old_board[i][j - 1] +
            old_board[i][j + 1] +
            old_board[i + 1][j - 1] +
            old_board[i + 1][j] +
            old_board[i + 1][j + 1];

        if (old_board[i][j] == 1) {
          new_board[i][j] = (num_of_live_neighbors == 2
   || num_of_live_neighbors == 3);
        } else {
          new_board[i][j] = num_of_live_neighbors == 3;
        }
      }
    }

    // print the new board
    for (int i = 1; i <= SIZE; i++) {
      for (int j = 1; j <= SIZE; j++) {
```

```c
        printf("%c ", new_board[i][j] ? '*' : ' ');
      }
      printf("\n");
    }

    // #include <unistd.h> (Linux, macOS)
    // <windows.h> Sleep(1000)
    sleep(1);

    // <stdlib.h> (Linux)
    // Windows: stdlib.h (system("cls"))
    system("clear");

    // copy new board onto old board
    for (int i = 1; i <= SIZE; i++) {
      for (int j = 1; j <= SIZE; j++) {
        old_board[i][j] = new_board[i][j];
      }
    }
  }

  return 0;
}
```

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

#include <stdio.h>

#define LEN 10
int dictionary[LEN] = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 };

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

  int low = 0;
  int high = LEN - 1;

  int index = -1;
  // TODO: binary search: search for key in dictionary[]
  while (low <= high) {
    int mid = (low + high) / 2;
    if (dictionary[mid] == key) {
      index = mid;
      break;
    } else if (dictionary[mid] > key) {
      high = mid - 1;
    } else {
      low = mid + 1;
    }
  }

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

  return 0;
}
```

```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 2024/10/10.

#include <stdio.h>

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

void SelectionSort(int arr[], int len);
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++) {
    // find the minimum value of numbers[i .. n-1]
    int min = arr[i];
    int min_index = i;

    for (int j = i + 1; j <= len - 1; ++j) {
      if (arr[j] < min) {
        min = arr[j];
        min_index = j;
      }
    }

    // swap arr[i] and arr[min_index]
    int temp = arr[i];
    arr[i] = arr[min_index];
    arr[min_index] = temp;
  }
}

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

```
45      printf("%d ", arr[i]);
46    }
47    printf("\n");
48  }
```

```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 hengxin on 10/19/22.
// 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);
  }

```

```c
  return 0;
}

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 ||
```

```c
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];
    }
  }
}

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