```c
/**
 * file: decl.c
 *
 * Created by hengxin on 12/01/22.
 */

int main() {
  char **argv;

  int *names[10];

  int (*musician_score_table)[10];

  int *StrCpyStd(char *dest, const char *src);

  int (*comp)(const void *left, const void *right);

  int atexit(void (*func)(void));

  void (*signal(int sig, void (*handler)(int)))(int);

  char (*(*func(int num, char *str))[])();

  char (*(*arr[3])())[5];
}
```

```c
/**
 * file: echo.c
 *
 * Echo program (command-line) arguments.
 *
 * Created by hengxin on 12/01/22.
 */

#include <stdio.h>

int main(int argc, char *argv[]) {

  return 0;
}
```

```c
/**
 * file: echo-escaped.c
 *
 * Created by hengxin on 12/01/22.
 */

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

int main(int argc, char *argv[]) {

  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
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;
}

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
  ) {
```

```c
        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];
    }
  }
}

void SleepAndClear(int sec) {
  sleep(sec);
```

```
106   system("clear");
107 }
```

```c
//
// Created by hfwei on 2022/12/8.
//

int main() {
  double low = 0.0;
  double high = 1.0;
  double integration = 0.0;

  return 0;
}
```

```c
//
// Created by hengxin on 11/18/22.
//

#include <stdio.h>

#define LEN_L 5
#define LEN_R 6

void Merge(int L[], int llen, int R[], int rlen);

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

  Merge(L, LEN_L, R, LEN_R);

  return 0;
}

void Merge(int L[], int llen, int R[], int rlen) {
  int l = 0;
  int r = 0;

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

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

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

```
 1 # 10-double-pointers
 2
 3 ## `echo.c`
 4
 5 - Linux `echo`
 6 - C standard
 7
 8 ## `echo-escaped.c`
 9
10 ## `scores.c`
11
12 - `student_score_table`: as a 2D array
13 - `Print`
14   - `int table[][COLS]` vs. `int (*table)[COLS]`
15 - `malloc`
16   - `int *`
17   - `int (*)[COLS]`
18
19 ## `game-of-life-pointer.c`
20
21 - `int (*)[]`
22
23 ## `sort.c`
24
25 - function pointers
26   - `f(a)`: `f` is actually a function pointer
27   - `&f`: `f` is a function
28
29 ## `decl.c`
30
31 - for more fun
```

```c
/**
 * file: scores.c
 *
 * Created by hengxin on 12/01/22.
 */

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

#define COLS 3

void Print(int table[][COLS], int rows);

int main() {
  /**
   * C, Java, Python scores of several musicians
   */
  int rows = 0;
  printf("Please input the number of students.\n");
  scanf("%d", &rows);

  // malloc here

  printf("Please input the scores of these students.\n");

  // fill in data here

  // print it here

  // access musician_score_table[3][2]

  // ptr_scores here
  // int (*ptr_scores)[COLS] = musician_score_table;
  // printf("ptr_scores[3][2] = %d\n",
  //        (*(ptr_scores + 3))[2]);

  // do not forget to free it

  return 0;
}

void Print(int table[][COLS], int rows) {
  printf("\n");
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < COLS; j++) {
      printf("%d ", table[i][j]);
    }
    printf("\n");
  }
}
```

```
1 5
2 0 10 20
3 10 20 30
4 20 30 40
5 30 40 50
6 40 50 60
```

```c
// Created by hfwei on 2022/11/25.

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

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

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

  // return value: (void *)
  int *numbers = malloc(len * sizeof(*numbers));
  // NULL: null pointer ((void *) 0)
  if (numbers == NULL) {
    printf("Memory allocation failed!\n");
    return 0;
  }

  for (int i = 0; i < len; i++) {
    scanf("%d", &numbers[i]);
  }

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

  free(numbers);
}

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

void SelectionSort(int *arr, int len) {
  for (int i = 0; i < len; i++) {
    // find the minimum of numbers[i .. len - 1]
    int min = arr[i];
    int min_index = i;
    for (int j = i + 1; j < len; j++) {
      if (arr[j] < min) {
        min = arr[j];
        min_index = j;
      }
    }

    Swap(arr + i, arr + min_index);
```

```
54   }
55 }
56
57 void Swap(int *left, int *right) {
58   int temp = *left;
59   *left = *right;
60   *right = temp;
61 }
62
63 // "Luo Dayou",
64 // "Cui Jian",
65 // "Dou Wei",
66 // "Zhang Chu",
67 // "Yao"
68 // "Wan Qing",
69 // "ZuoXiao",
70 // "ErShou Rose"
71 // "Hu Mage",
72 // "Li Zhi",
```

```c
/**
 * file: sort.c
 *
 * Created by hengxin on 12/01/22.
 *
 * A nice function pointer example on Riemann integration:
 * https://en.wikipedia.org/wiki/Function_pointer
 */

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

int CompareInts(const void *left, const void *right);
void PrintInts(const int *integers, int len);

int CompareStrs(const void *left, const void *right);
void PrintStrs(const char *str[], int len);

int main() {
  // sort an array of integers
  int integers[] = {-2, 99, 0, -743, 2, INT_MIN, 4};
  int size_of_integers = sizeof integers / sizeof *integers;

  /**
   * void qsort( void *ptr, size_t count, size_t size,
             int (*comp)(const void *, const void *) );
   */
  int (*comp)(const void *, const void *) = CompareInts;
  // you should not do this!!!
  // printf("sizeof comp : %zu\n", sizeof comp);
  printf("comp : %p\n", comp);
  printf("*comp : %p\n", *comp);
  printf("CompareInts : %p\n", CompareInts);
  printf("&CompareInts : %p\n", &CompareInts);

  qsort(integers, size_of_integers, sizeof *integers, comp);
  PrintInts(integers, size_of_integers);

  // Call functions indirectly via function pointers.
  int a = 10;
  int b = 20;
  printf("%d %s %d\n", a, comp(&a, &b) > 0 ? ">" : "<=", b);

  // Sorting an array of strings
  const char *names[] = {
      "Luo Dayou",
      "Cui Jian",
      "Dou Wei",
      "Zhang Chu",
      "He Yong",
      "Wan Qing",
```

```
 54       "WuTiaoRen",
 55       "ZuoXiao",
 56       "Hu Mage",
 57       "Li Zhi"
 58   };
 59   int size_of_names = sizeof names / sizeof *names;
 60
 61   comp = CompareStrs;
 62   qsort(names, size_of_names, sizeof *names, comp);
 63   PrintStrs(names, size_of_names);
 64 }
 65
 66 int CompareInts(const void *left, const void *right) {
 67   int int_left = *(const int *) left;
 68   int int_right = *(const int *) right;
 69
 70   if (int_left < int_right) {
 71     return -1;
 72   }
 73
 74   if (int_left > int_right) {
 75     return 1;
 76   }
 77
 78   return 0;
 79
 80 //   return int_left - int_right; // erroneous shortcut (fails if
    INT_MIN is present)
 81 }
 82
 83 // actual arguments: char *const *
 84 int CompareStrs(const void *left, const void *right) {
 85   char *const *pp1 = left;
 86   char *const *pp2 = right;
 87   return strcmp(*pp1, *pp2);
 88 }
 89
 90 void PrintInts(const int *integers, int len) {
 91   printf("\n");
 92   for (int i = 0; i < len; ++i) {
 93     printf("%d ", integers[i]);
 94   }
 95   printf("\n");
 96 }
 97
 98 void PrintStrs(const char *str[], int len) {
 99   printf("\n");
100   for (int i = 0; i < len; i++) {
101     printf("%s\n", str[i]);
102   }
103   printf("\n");
104 }
```