```c
#include <stdio.h>
/**
 * file: array-pointer.c
 *
 * Created by hengxin on 12/5/21.
 */

#define ROWS 5
#define COLS 3

void Print(int table[][3], int rows);

int main() {
  /**
   * C, Java, Python
   *
   * student_score_table is a 2D (two-dimensional) array.
   * student_score_table is an array of 5 arrays,
   * each of which contains 3 elements.
   */
  int student_score_table[ROWS][COLS] = {
      {0, 10, 20},
      {10, 20, 30},
      {20, 30, 40},
      {30, 40, 50},
      {40, 50, 60}
  };

  printf("student_score_table[3][2] = %d\n",
         student_score_table[3][2]);
  printf("student_score_table[3][2] = %d\n",
         *(student_score_table[3] + 2));
  printf("student_score_table[3][2] = %d\n",
         (* (student_score_table + 3))[2]);
  printf("student_score_table[3][2] = %d\n",
         *(* (student_score_table + 3) + 2));

  /**
   * student_score_table is a pointer to (an array of 3 ints)
   */
  int (*ptr_scores)[3] = student_score_table;
  printf("student_score_table[3][2] = %d\n",
         (*(ptr_scores + 3))[2]);

  Print(student_score_table, ROWS);

  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++) {
```

```
54        printf("%d ", table[i][j]);
55      }
56      printf("\n");
57    }
58 }
59
```

```c
/**
 * file: decl.c
 *
 * Created by hengxin on 12/12/21.
 */

int main() {
  /**
   * argv is a pointer to pointer to char
   */
  char **argv;

  /**
   * names is an array consisting of 10 pointers to int
   */
  int *names[10];

  /**
   * student_score_table is a pointer to an array
   * consisting of 10 integers
   */
  int (*student_score_table)[10];

  /**
   * StrCpyStd is a function
   */
  int *StrCpyStd(char *dest, const char *src);

  /**
   * comp is a pointer to function returning int
   */
  int (*comp)(const void *left, const void *right);

  /**
   * func is a function returning pointer to array[] of
   * pointer to function returning char
   */
  char (*(*func(int num, char *str))[])();

  /**
   * array[3] of pointer to function
   * returning pointer to array[5] of char
   */
  char (*(*arr[3])())[5];
}
```

```c
/**
 * file: toupper.c
 *
 * Echo program (command-line) arguments.
 *
 * Created by hengxin on 12/12/21.
 */

#include <stdio.h>

/**
 * @param argc argument count.
 * @param argv argument vector.
 *   By convention, argv[0] is the name by which the program was invoked.
 *   By the C standard, argv[argc] is a null pointer.
 */
int main(int argc, char *argv[]) {
  printf("%d\n", argc);

  /**
   * 1st version: treats argv as an array of character pointers
   */
  for (int i = 1; i < argc; i++) {
    printf("%s%s", argv[i], (i < argc - 1) ? " " : "");
  }
  printf("\n");

  /**
   * 2st version: treat argv as a pointer to pointer to char
   */
  while (--argc > 0) {
    printf("%s%s", *++argv, (argc > 1) ? " " : "");
  }
  printf("\n");

  /**
   * 3rd version: use an expression as the format argument of printf
   */
  while (--argc > 0) {
    printf((argc > 1) ? "%s " : "%s", *++argv);
  }
  printf("\n");


  return 0;
}


```

```c
/**
 * file: toupper-flag.c
 *
 * Created by hengxin on 12/12/21.
 */

#include <stdio.h>

int main(int argc, char *argv[]) {
  int escaped = 0;

  char **args = argv;
  while (*++args != NULL) {
    if ((*args)[0] == '-') {
      char flag = (*args)[1];
      switch (flag) {
        case 'e':
          escaped = 1;
          break;
        case 'E':
          escaped = 0;
          break;
        default:
          printf("toupper: illegal flag '%c'\n"
                 "Use -e or -E.\n", flag);
          return 0;
      }
    }
  }

  char *arg = NULL;
  while (--argc > 0) {
    arg = *++argv;
    if (arg[0] == '\\' && escaped) {
      if (arg[1] == 't') {
        printf("\t");
      }
      if (arg[1] == 'n') {
        printf("\n");
      }
    }
    else if (arg[0] != '-') {
        printf((argc > 1) ? "%s " : "%s", arg);
    }
  }
}
```

```c
/**
 * file: game-of-life-pointer.c
 *
 * Created by hengxin on 12/5/21.
 */

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

#define ROUND 10
#define SIZE 6
int board[SIZE][SIZE] = {
    {0},
    {0},
    {0, 0, 1, 1, 1, 0},
    {0, 1, 1, 1, 0, 0},
    {0},
    {0}};

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_extended_board)[SIZE + 2],
                      int (*new_extended_board)[SIZE + 2]);
void CopyExtendedBoard(const int (*src_board)[SIZE + 2],
                       int (*dest_board)[SIZE + 2]);
void ClearTerminal(int sec);

int main() {
  int old_board[SIZE + 2][SIZE + 2];
  ExtendBoard(board, old_board);
  PrintExtendedBoard(old_board);
  ClearTerminal(1);

  int new_board[SIZE + 2][SIZE + 2];
  for (int round = 0; round < ROUND; round++) {
    GenerateNewBoard(old_board, new_board);
    PrintExtendedBoard(new_board);
    ClearTerminal(1);
    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
  ) {
        extended_board[row][col] = 0;
      } else {
```

```c
        extended_board[row][col] = origin_board[row - 1][col - 1];
      }
    }
  }
}

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

void GenerateNewBoard(const int (*old_extended_board)[SIZE + 2],
                      int (*new_extended_board)[SIZE + 2]) {
  for (int row = 1; row < SIZE + 1; row++) {
    for (int col = 1; col < SIZE + 1; col++) {
      int neighours = old_extended_board[row - 1][col - 1]
          + old_extended_board[row - 1][col]
          + old_extended_board[row - 1][col + 1]
          + old_extended_board[row][col - 1]
          + old_extended_board[row][col + 1]
          + old_extended_board[row + 1][col - 1]
          + old_extended_board[row + 1][col]
          + old_extended_board[row + 1][col + 1];

      new_extended_board[row][col] =
          (old_extended_board[row][col] && (neighours == 2 ||
  neighours == 3))
              || (!old_extended_board[row][col] && neighours == 3);
    }
  }
}

void CopyExtendedBoard(const int (*src_board)[SIZE + 2],
                       int (*dest_board)[SIZE + 2]) {
  for (int row = 0; row < SIZE + 2; row++) {
    for (int col = 0; col < SIZE + 2; col++) {
      dest_board[row][col] = src_board[row][col];
    }
  }
}

void ClearTerminal(int sec) {
  sleep(sec);
  printf("\033c");
}
```

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

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

#define LEN 10

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

int main() {
  const char *musicians[LEN] = {
      "Luo Dayou",
      "Cui Jian",
      "Dou Wei",
      "Zhang Chu",
      "Li Zhi",
      "Wan Qing",
      "WuTiaoRen",
      "ZuoXiao",
      "He Mage",
      "He Yong",
  };

  PrintStrs(musicians, LEN);
  SelectionSort(musicians, LEN);
  PrintStrs(musicians, LEN);
}

void PrintStrs(const char *str[], int len) {
  printf("\n");
  for (int i = 0; i < len; i++) {
    printf("%s\n", str[i]);
  }
  printf("\n");
}

// arr: the (copy of the) address of the first element of the `numbers ` array
void SelectionSort(char *str[], int len) {
  for (int i = 0; i < len; i++) {
    // find the minimum of musicians[i .. len - 1]
    const char *min = str[i];
    int min_index = i;

    for (int j = i + 1; j < len; j++) {
      if (strcmp(min, str[j]) > 0) {
        min = str[j];
        min_index = j;
      }
    }

```

```
53      // swap str[i] and str[min_index]
54      Swap(str + i, str + min_index);
55    }
56  }
57
58  void Swap(char **left, char **right) {
59    char *temp = *left;
60    *left = *right;
61    *right = temp;
62  }
```

```c
/**
 * file: sort.c
 *
 * Created by hengxin on 12/5/21.
 *
 * A good 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() {
  /**
   * Sorting 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;
  printf("%p\n", comp);
  printf("%p\n", *comp);
  printf("%p\n", CompareInts);
  printf("%p\n", &CompareInts);

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

  /**
   * You can 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",
```

```
 54       "Dou Wei",
 55       "Zhang Chu",
 56       "He Yong",
 57       "Wan Qing",
 58       "WuTiaoRen",
 59       "ZuoXiao",
 60       "Hu Mage",
 61       "Li Zhi"
 62   };
 63   int size_of_names = sizeof names / sizeof *names;
 64
 65   comp = CompareStrs;
 66   qsort(names, size_of_names, sizeof *names, comp);
 67   PrintStrs(names, size_of_names);
 68 }
 69
 70 int CompareInts(const void *left, const void *right) {
 71   int int_left = * (const int*) left;
 72   int int_right = * (const int*) right;
 73
 74   if (int_left < int_right) {
 75     return -1;
 76   }
 77
 78   if (int_left > int_right) {
 79     return 1;
 80   }
 81
 82   return 0;
 83
 84 //   return int_left - int_right; // erroneous shortcut (fails if
    INT_MIN is present)
 85 }
 86
 87 int CompareStrs(const void *left, const void *right) {
 88   char * const *pp1 = left;
 89   char * const *pp2 = right;
 90   return strcmp(*pp1, *pp2);
 91 }
 92
 93 void PrintInts(const int integers[], int len) {
 94   printf("\n");
 95   for (int i = 0; i < len; ++i) {
 96     printf("%d ", integers[i]);
 97   }
 98   printf("\n");
 99 }
100
101 void PrintStrs(const char *str[], int len) {
102   printf("\n");
103   for (int i = 0; i < len; i++) {
104     printf("%s\n", str[i]);
105   }
```

```
106   printf("\n");
107 }
```