```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: 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 {
```

```
 53         extended_board[row][col] = origin_board[row - 1][col - 1];
 54       }
 55     }
 56   }
 57 }
 58
 59 void PrintExtendedBoard(const int (*extended_board)[SIZE + 2]) {
 60   for (int row = 1; row < SIZE + 1; row++) {
 61     for (int col = 1; col < SIZE + 1; col++) {
 62       printf("%c ", extended_board[row][col] ? '*' : ' ');
 63     }
 64     printf("\n");
 65   }
 66 }
 67
 68 void GenerateNewBoard(const int (*old_extended_board)[SIZE + 2],
 69                       int (*new_extended_board)[SIZE + 2]) {
 70   for (int row = 1; row < SIZE + 1; row++) {
 71     for (int col = 1; col < SIZE + 1; col++) {
 72       int neighours = old_extended_board[row - 1][col - 1]
 73           + old_extended_board[row - 1][col]
 74           + old_extended_board[row - 1][col + 1]
 75           + old_extended_board[row][col - 1]
 76           + old_extended_board[row][col + 1]
 77           + old_extended_board[row + 1][col - 1]
 78           + old_extended_board[row + 1][col]
 79           + old_extended_board[row + 1][col + 1];
 80
 81       new_extended_board[row][col] =
 82           (old_extended_board[row][col] && (neighours == 2 ||
    neighours == 3))
 83               || (!old_extended_board[row][col] && neighours == 3);
 84     }
 85   }
 86 }
 87
 88 void CopyExtendedBoard(const int (*src_board)[SIZE + 2],
 89                        int (*dest_board)[SIZE + 2]) {
 90   for (int row = 0; row < SIZE + 2; row++) {
 91     for (int col = 0; col < SIZE + 2; col++) {
 92       dest_board[row][col] = src_board[row][col];
 93     }
 94   }
 95 }
 96
 97 void ClearTerminal(int sec) {
 98   sleep(sec);
 99   printf("\033c");
100 }
```

```c
/**
 * file: selection-sort-strings.c
 *
 * Created by hengxin on 11/28/21.
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#define LEN 10

void SelectionSort(char *str[], int len);
void Swap(char **left, char **right);
void SwapWrong(char *left, char *right);
void SwapWrong2(char *left, char *right);
void PrintStrs(const char *str[], int len);

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

  PrintStrs(names, LEN);

  SelectionSort(names, LEN);

  PrintStrs(names, LEN);

  return 0;
}

void SelectionSort(char *str[], int len) {
  for (int i = 0; i < len; ++i) {
    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;
      }
    }

    Swap(&str[i], &str[min_index]);
```

```
54 //    SwapWrong(str[i], str[min_index]);
55   }
56 }
57
58 void Swap(char **left, char **right) {
59   char *tmp = *left;
60   *left = *right;
61   *right = tmp;
62 }
63
64 /**
65  * segment fault
66  * SwapWrong(str[i], str[min_index]);
67  */
68 void SwapWrong(char *left, char *right) {
69   char tmp = *left;
70   *left = *right;
71   *right = tmp;
72 }
73
74 /**
75  * does not work
76  * (it is performed on copies of str[i] and str[min_index])
77  * SwapWrong2(str[i], str[min_index]);
78  */
79 void SwapWrong2(char *left, char *right) {
80   char tmp = *left;
81   *left = *right;
82   *right = tmp;
83 }
84
85 void PrintStrs(const char *str[], int len) {
86   printf("\n");
87   for (int i = 0; i < len; i++) {
88     printf("%s\n", str[i]);
89   }
90   printf("\n");
91 }
```

```c
/**
 * file: strcmp.c
 *
 * Created by hengxin on 11/28/21.
 */

#include <stdio.h>

/**
 * Compare two strings.
 *
 * @param s1 The first string to compare with.
 * @param s2 The second string to compare with.
 * @return 0 if s1 equals to s2;
 *         positive if s1 is greater than s2;
 *         negative if s1 is less than s2
 */
int StrCmp(const char *s1, const char *s2);
int StrCmpStd(const char *s1, const char *s2);

int main() {
  const char *str1 = "hi, hengxin";
  const char *str2 = "hi, ant";

  printf("%s %c %s\n",
         str1, StrCmp(str1, str2) > 0 ? '>' : '<', str2);

  return 0;
}

int StrCmp(const char *s1, const char *s2) {
  while (*s1 == *s2 && (*s1 != '\0' && *s2 != '\0')) {
    s1++;
    s2++;
  }

  if (*s1 == '\0' && *s2 == '\0') {
    return 0;
  }

  return *s1 - *s2;
}

int StrCmpStd(const char *s1, const char *s2) {
  for ( ; *s1 == *s2; s1++, s2++) {
    if (*s1 == '\0') {
      return 0;
    }
  }

  return (* (const unsigned char *) s1)
    < (* (const unsigned char *) s2) ? -1 : 1;
}
```

```c
/**
 * file: strcpy.c
 *
 * Created by hengxin on 11/28/21.
 *
 * strcpy vs. strcpy_s (safe/secure; optional in C++)
 * strncpy vs. strncpy_s (optional in C11)
 * keyword "restrict"
 */

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

/**
 * Copy string at src to dest.
 * We assume that there is enough room in dest for storing src.
 *
 * @param dest
 * @param src
 */
void StrCpy(char *dest, const char *src);
void StrCpy1(char *dest, const char *src);
void StrCpy2(char *dest, const char *src);
void StrCpy3(char *dest, const char *src);
void StrCpy4(char *dest, const char *src);
char* StrCpyStd(char *dest, const char *src);

int main() {
  const char *src = "Hello World";
  char dest[strlen(src) + 1];

  StrCpy(dest, src);
  printf("dest = %s\n", dest);

  return 0;
}

void StrCpy(char *dest, const char *src) {
  int i = 0;
  while (src[i] != '\0') {
    dest[i] = src[i];
    i++;
  }

  dest[i] = '\0';
}

void StrCpy1(char *dest, const char *src) {
  int i = 0;
  while ((dest[i] = src[i]) != '\0') {
    i++;
  }

```

```
54   // the following code is unnecessary for this version
55   //  dest[i] = '\0';
56 }
57
58 void StrCpy2(char *dest, const char *src) {
59   int i = 0;
60   while ((* (dest + i) = * (src + i)) != '\0') {
61     i++;
62   }
63 }
64
65 void StrCpy3(char *dest, const char *src) {
66   while ((*dest = *src) != '\0') {
67     src++;
68     dest++;
69   }
70 }
71
72 void StrCpy4(char *dest, const char *src) {
73   while ((*dest++ = *src++) != '\0');
74 }
75
76 void StrCpy5(char *dest, const char *src) {
77   while ((*dest++ = *src++));
78 }
79
80 // for exam
81 void f(char *dest, const char *src) {
82   while ((*dest++ = *src++));
83 }
84
85 /**
86  * @return The dest pointer
87  */
88 char *StrCpyStd(char *dest, const char *src) {
89   for (char *s = dest; (*s++ = *src++) != '\0'; );
90
91   return dest;
92 }
```

```c
/**
 * file: strlen.c
 *
 * Created by hengxin on 11/28/21.
 */

#include <stdio.h>

int StrLen(const char *s);
size_t StrLenStd(const char *s);

int main() {
  /**
   * a copy of "Hello World!" in the data segment
   */
  char msg[20] = "Hello World!";
  msg[0] = 'N';

  /**
   * This string literal "Hello World!" is stored in the text segment
   */
  char *ptr_msg = "Hello World!";
//  *ptr_msg = 'N';

  printf("The length of the message \"%s\" = %d\n",
         msg, StrLen(msg));

  return 0;
}

int StrLen(const char *s) {
  int len = 0;
  while (s[len] != '\0') {
    len++;
  }

  return len;
}

size_t StrLenStd(const char *s) {
  const char *sc;
  for (sc = s; *sc != '\0'; sc++);

  return (sc - s);
}

// for exam
size_t f(const char *s) {
  const char *sc;
  for (sc = s; *sc != '\0'; sc++);

  return (sc - s);
}
```