Lec 10 Strings and Multi-dimensional Arrays

1. 数组作为函数参数的传递

传递方式

- C语言中，**数组作为函数参数时，传递的是引用**，即数组的首地址，而不是数组的副本。
- 函数内部对数组的修改会直接影响原始数组。
- 函数声明中，参数可以写成：

```
int func(int array[]);  // 或 int func(int *array);
```

- 这两种形式是等价的，因为数组名本质上是指针。

示例问题

通过一个掷骰子的例子，展示了数组传递的两种方式：

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main(){
  srand(time(0));
  int rnd_number[3];

  // update rnd_number as 3 integers
  roll_a_dice1(&rnd_number[0],&rnd_number[1], &rnd_number[2]);
  // update rnd_number as array
  roll_a_dice2(rnd_number);
}
```

1. **传递多个指针**：

```
roll_a_dice1(&rnd_number[0], &rnd_number[1], &rnd_number[2]);
```

- 传递每个元素的地址，函数分别修改这三个地址的值。

2. **传递数组名**：

```
roll_a_dice2(rnd_number);
```

- 传递数组名，函数通过下标或指针访问数组元素。

函数实现：

`roll_a_dice1`：

```
// generate 3 random numbers, 1-6
// rand(): range between 0 to INT_MAX
void roll_a_dice1(int* one, int* two, int* three){
  *one = (rand () % 6) + 1;
  *two = (rand () % 6) + 1;
  *three = (rand () % 6) + 1;
}
```

- 接收三个指针，分别赋值为 1-6 的随机数。

`roll_a_dice2`：

```
void roll_a_dice2(int array[]) {
  //void roll_a_dice2(int *array)
  //void roll_a_dice2(int array[3])
  array[0] = (rand () % 6) + 1;
  array[1] = (rand () % 6) + 1;
  array[2] = (rand () % 6) + 1;
}
```

```
void roll_a_dice2(int array[]) {
  //void roll_a_dice2(int *array)
  //void roll_a_dice2(int array[3])
  *(array) = (rand () % 6) + 1;
  *(array + 1) = (rand () % 6) + 1;
  *(array + 2) = (rand () % 6) + 1;
}
```

- 接收数组名，通过下标操作元素。也可以写成 `int *array` 或 `int array[3]`，效果相同。

练习：反转数组

任务：实现一个函数`array_reverse`，将整数数组反转，例如把`[1,2,3]`变为`[3,2,1]`

实现：

```
void array_reverse(int array[], int n) {
    int i, temp;
    for (i = 0; i < n / 2; i++) {
        temp = array[i];
        array[i] = array[n - 1 - i];
        array[n - 1 - i] = temp;
    }
}
```

- **逻辑**：从数组两端向中间交换元素，循环次数为 n / 2。
- **辅助函数** `print_array`：

```c
void print_array(int array[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        printf("%d ", *(array + i)); // 等价于 array[i]
    }
    printf("\n");
}
```

- 使用指针算术 `*(array + i)` 访问元素，展示了指针与数组的等价性。

字符串

定义

- 在C语言中，**字符串**是一个 `char` 类型的数组，以空字符 `'\0'` 结尾。
- 声明时需为 `'\0'` 预留空间，例如：

```c
char buf[200]; // 可存储最多199个字符 + '\0'
```

初始化

- 可以使用特殊语法：

```c
char buf[200] = "abc"; // buf[0]='a', buf[1]='b', buf[2]='c', buf[3]='\0'
```

- **注意**：不能直接赋值，例如 `buf = "abc";` 会报错，需使用 `strcpy` 函数。

练习：复制字符串

实现一个函数 `string_copy`，将源字符串复制到目标字符串。

示例代码：

```c
#include <stdio.h>
#include <string.h>
void string_copy(char des[], char src[]) {
    int i = 0;
    while (src[i] != '\0') {
        des[i] = src[i];
        i++;
    }
    des[i] = '\0'; // 添加终止符
}
int main() {
    char des[10];
    string_copy(des, "ABC");
    printf("%s\n", des); // 输出 "ABC"
    strcpy(des, "ABCDE"); // 使用标准库函数
    printf("%s\n", des); // 输出 "ABCDE"
```

```
    return 0;
}
```

对比：自定义 `string_copy` 需要手动复制字符，而 `strcpy` 是标准库提供的便捷函数。

字符串的I/O操作

输出：`printf`

- 使用`"%s"`格式化字符串，从首地址打印到`'\0'`

```
char buf[10] = "abc";
printf("%s\n", buf); // 输出 "abc"
```

输入：`scanf`

- 使用 `"%s"` 读取字符串，直到遇到空白字符，自动添加 `'\0'`

```
char input[10];
scanf("%s", input); // 直接传数组名
```

- **问题**：`scanf("%s", input[0]);` 错误，因为 `input[0]` 是 `char`，而 `scanf` 需要 `char*`。正确的是 `scanf("%s", &input[0]);` 或 `scanf("%s", input);`

字符串的处理

1. 设定字符串在打印第3个字符后终止

```
char temp[100] = "abcdef";
temp[3] = '\0';
printf("%s", temp); // 输出 "abc"
```

2. 设定字符串从第3个字符开始打印

```
char temp[100] = "abcdef";
printf("%s", &temp[2]); // 输出 "cdef"
```

结论：终止符 `'\0'` 决定了字符串的有效长度，指针可以用来访问子串。

读取字符串

1. `fgets`：读取一行，直到换行或指定长度。Read characters from stdin(keyboard) and store them into buf until SIZE_BUF-1 characters or a newline or the end-of-file.

```
char buf[SIZE_BUF];
fgets(buf, SIZE_BUF, stdin);
```

2. `scanf`：读取到空白字符为止。Read characters from stdin(keyboard) and store them into buf until whitespace characters.

```
char buf[SIZE_BUF]
scanf("%s", buf);
```

3. `sscanf`：从字符串中读取格式化数据。Read formatted data from string and return the number of items successfully read

```
char buf[100] = "abc1 34 21";
char str[20];
int num1, num2;
int rc = sscanf(buf, "%s%d%d", str, &num1, &num2);
// rc = 3, str = "abc1", num1 = 34, num2 = 21
```

多维数组

定义

- **二维数组**：如 `int a[2][3]`; 表示2行3列。
- **内存布局**：按行优先（Row-major order）连续存储，即`offset=(row index) * (size of column) + column index`

传递方式

- 一维数组：`int a[]` 或 `int *a`
- 二维数组：必须指定除第一维外的维度，如 `int a[][3]` 或 `int (*a)[3]`

初始化和打印

- 打印函数：

```
#define ROW 2
#define COL 3
void print_2D(int a[][COL]) {
    # compiler needs to know the size of column to calculate the memory offset
    int i, j;
    for (i = 0; i < ROW; i++) {
        for (j = 0; j < COL; j++) {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}
```

```
#define ROW 2
#define COL 3
void print_2D_ptr(int *a){
    # pass the array as a pointer, lose the 2D-array shape information
    int i, j;
    for(i=0; i<ROW; i++){
        for(j=0; j<COL; j++){
            printf("%d ", a[i*COL+ j]);
        }
        printf("\n");
    }
}
```

- 初始化函数：按行初始化，或者线性初始化

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
int a[2][3] = {1, 2, 3, 4, 5, 6};
```

练习：交换矩阵中的行

- `matrix_change`：交换矩阵的第`x`行和第`y`行
- `matrix_tr`：转置矩阵

```
void matrix_change(int x, int y, int src[N][M]) {
    int j, temp;
    for (j = 0; j < M; j++) {
        temp = src[x][j];
        src[x][j] = src[y][j];
        src[y][j] = temp;
    }
}

void matrix_tr(int src[N][M], int des[M][N]) {
    int i, j;
    for (i = 0; i < N; i++) {
        for (j = 0; j < M; j++) {
            des[j][i] = src[i][j];
        }
    }
}
```