1、break,continue,return关键字的使用

1.1 break关键字

```
1. 和switch...case配合使用，退出switch....case分支。
2. 和循环语句配合使用，退出循环体，后边所有的循环都将不会再被执行。
```

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    for (int i = 0; i < 10; i++)
    {
        putchar('#');
        if (i == 5)
        {
            break;  // 退出循环体，后边的所有的循环都将不会被执行
        }
    }

    putchar('\n');
    return 0;
```

```
}

输出结果：######
```

```
#include <stdio.h>
int main(int argc, const char *argv[])
{
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            putchar('#');
            if (j == 5)
            {
                break;  // 退出循环体，后边的所有的循环都将不会被执行
            }
        }
        putchar('*');
    }

    putchar('\n');
    return 0;
}
输出结果：
######*######*######*######*######*
```

1.2 continue关键字

```
1. 和循环语句配合使用，退出本次循环，continue之后的代码不会被执行，
    而是执行下一次的循环
```

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    for (int i = 0 ; i < 10; i++)
    {
        putchar('#');
        if (i >= 5)
        {
            continue; // 退出本次循环，本次循环后边的代码不会被执行，执行下一次的循环
        }
        putchar('*');
    }
    putchar('\n');
    return 0;
}
输出结果：#*#*#*#*#*#####
```

1.3 return关键字

```
1. 常用于函数的退出，并返回一个结果。
```

```
案例：
#include <stdio.h>

int main(int argc, const char *argv[])
```

```
 5 {
 6     int i,j;
 7
 8     for (i = 1; i <= 9; i++)
 9     {
10         for(j = 1; j <= 9; j++)
11         {
12             printf("%d * %d = %d\t", i,
   j, i*j);
13             if ( i == j)
14             {
15                 break;
16             }
17         }
18         putchar('\n');
19     }
20     return 0;
21 }
22
```

2、函数

2.1 概念

```
1       将具有特定功能的一段代码，封装成一个代码块，当使用此代码时，
2       可以通过调用的方式进行使用。
3       封装函数之后，需要重复被使用的代码不需要重复书写，直接通过
4       函数的调用实现即可。
5
6       比如：printf,scanf,putchar,getchar,puts,gets
7               算法库，
8
9       调用别人实现的函数，无需了解函数的内部的实现，
10      只需要掌握被调用函数的(三要素)：功能，参数，返回值
```

2.2 定义函数的语法格式

```
1
2  返回类型   函数名 (形参列表)
3  {
4       函数体;
5       return 返回值;
6  }
7
8  返回类型   函数名 (数据类型 形参变量名1, 数据类型 形参变量名2,...)
9  {
10      函数体;
11      return 返回值;
```

```
}

注:
    1. 返回类型 :  数据类型
    2. 函数名: 遵循标识符的命名的规则
    3. 形参变量名 : 变量名, 遵循标识符的命名的规则,
        此变量属于局部变量, 只能再函数内被调用。

    4. 如果函数有返回值通过 return 返回值;进行返回
        如果函数没有返回值值, return;可以省略不写,
        或者写成return;
    5. 函数的形参也是可有可无, 如果没有形参写成()或者(void)
```

2.3 函数的调用

```
函数没有形参, 没有返回值:
    函数名();        ---> ()中不可以写任何的东西

函数有形参, 没有返回值:
    函数名(实参列表);  ---> 比如: 函数名(1,2);

函数没有形参, 有返回值:
    变量名 = 函数名();
```

```
函数有形参，有返回值：
    变量名 = 函数名(实参列表);  ---> 比如:sum = 函数名(a, b);

如果函数的返回类型为int类型，一般使用int类型的变量接收函数的返回值，
如果使用float类型的变量接收返回类型为int类型的函数的返回值，
会发生隐式类型转换。
```

```
案例1：封装函数，打印以下图形
**************************************
*** 1. 增 2. 删 3. 改 4. 查 5. 退出 ***
**************************************

案例2：封装一个加法运算的函数

#include <stdio.h>

void print_menu(void)
{

    puts("**************************************");
    puts("*** 1.增 2.删 3.改 4.查 5.退出 ***");

    puts("**************************************");

}
```

```
16      return; // return;可以省略
17  }
18
19  /*
20   * 封装函数时先书写函数的名字，见名知意,确定函数的功能,
21   * 在考虑函数的形参，如果需要通过外部传递那就添加对应的参数,
22   * 如果函数的结果需要被外部的使用，那就通过返回值进行返回。
23   *
24   *
25   * */
26
27  int add_func(int a, int b)
28  {
29  #if 0
30      int sum = 0;
31      sum = a + b;
32      return sum;
33  #else
34      return (a + b);
35  #endif
36  }
37
38
39
40  int main(int argc, const char *argv[])
41  {
42      // 函数的调用
43      print_menu();
```

```c

    int sum = 0;
    int a = 1000, b = 2000;
    sum = add_func(a, b);
    printf("sum = %d\n", sum);

    printf("sum = %d\n", add_func(123, 456));


    return 0;
}

```

3、数组

3.1 数组的概念

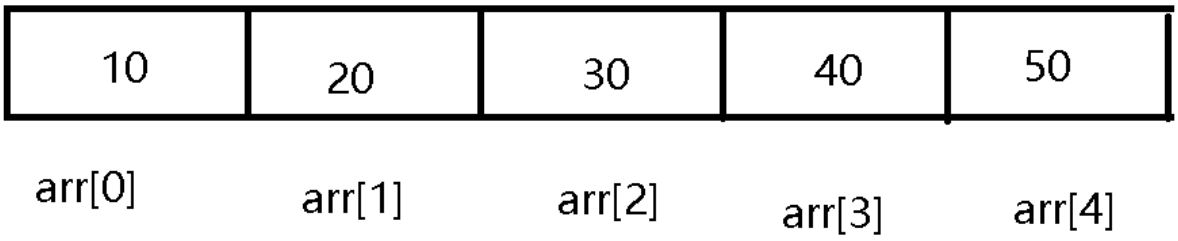
```
    数组属于构造类型。
    特点：
        1> 数组中的每个成员的类型都是一样的；
        2> 数组中的所有的成员在内存中地址是连续的；

    数组分类：一维数组，二维数组，多维数组
```

3.2 一维数组

3.2.1 一维数组的概念

```
1        一维数组的下标只有一个，每个元素的下标(列)。
2        一维数组中的每个成员在内存中都是连续的。
```

int arr[5] = {10,20,30,40,50};



arr[0] arr[1] arr[2] arr[3] arr[4]

3.3.2 定义一维数组

```
1  存储类型    数据类型      数组名[数组中成员个数];
2
3  数组中成员个数 ： 定义数组时必须指定数组的长度(成员个数)，
4          数组一旦定义好之后，数组的长度不可以被修改。
5
6  数组中的每个成员进行访问时，数组的下标从0开始；
7          数组的第0个元素，数组的第1个元素，数组的第2个元素.....
8          数组的第0个成员，数组的第1个成员，数组的第2个成员.....
```

```
1 数组的使用：使用数组时一般都是使用数组的某个元素。
2 访问数组的某个元素：  数组名[下标];
```

3.3.3 定义数组并进行初始化的方式

```c
#include <stdio.h>
/*
 * 功能：打印数组中每个元素的值
 * 参数：
 *  @ arr ：传递一个int类型的数组的数组名，数组名表示数组的首地址
 *  @ len ：传递的数组的长度
 * 返回值：无
 * */
void print_arr(int arr[], int len)
{
    printf("----------------------\n");
    for(int i = 0; i < len; i++)
    {
        printf("arr[%d] = %d\n", i, arr[i]);
    }
}

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

```c
    // 1. 定义数组的同时进行初始化
    int arr[5] = {10,20,30,40,50};
    // 打印数组中每个成员的值
    for (int i = 0; i < 5; i++)
    {
        printf("arr[%d] = %d\n", i, arr[i]);
    }

    // 2. 定义数组的同时，对部分成员进行初始化
    // 依次对数组中的每个元素进行初始化，后边的没有初始化的
    // 默认会初始化为0.
    int arr2[5] = {100, 200};
    // 如果函数的形参是一个数组类型的，传递数组的名字
    print_arr(arr2, 5);

    // 3. 定义数组时，将数组中的所有成员初始化为0
    int arr3[10] = {0};
    print_arr(arr3, 10);

    // 4. 先定义数组，然后在对每个成员分别进行初始化
    int arr4[5];            // 不进行初始化，默认初始化为随机值
    print_arr(arr4, 5);
    // 数组一旦被定义了，就不可以对其整体进行初始化，
```

```
43      // 只能一个一个元素的进行初始化
44      // arr4 =
    {1000,2000,3000,4000,5000};  // 错误
45      arr4[0] = 1000;
46      arr4[1] = 2000;
47      arr4[2] = 3000;
48      arr4[3] = 4000;
49      arr4[4] = 5000;
50      print_arr(arr4, 5);
51
52      // 5. 定义数组时不指定数组的长度，通过初
    始化成员的个数，
53      // 指定数组的长度
54      // 如果定义数组时不指定数组的长度，必须对
    数组定义的同时，
55      // 进行初始化
56      int arr5[] = {111,222,333,444,555};
    // 数组长度为5
57      print_arr(arr5, 5);
58
59      return 0;
60  }
61
```

3.3.4 数组的特性及使用时注意事项

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int arr[10] = {0};
    int arr1[10] = {0};

    // 1. 计算数组占用内存空间的大小 sizeof(数组名)
    printf("数组的占用内存空间大小 = %ld\n", sizeof(arr));

    // 2. 计算数组中的成员个数(数组长度)
    printf("数组成员的个数 = %ld\n", sizeof(arr)/sizeof(int));

    // 3. 数组的名字就是数组的在内存分配内存空间的首地址,
    // 和数组的第0个元素的地址是一样的。
    printf("数组的首地址 = %p\n", arr);
    // & : 单目运算符, 对变量进行取地址运算
    // %p : 按照十六进制形式打印地址。
    printf("数组的第0个元素的地址 = %p\n", &arr[0]);

    // 4. 数组名是一个常量, 不可以进行赋值操作,
```

```
21      // 数组名也不可以进行自增自减运算。
22      // arr = arr1;                  // 错误
23      // arr++;                       // 错误
24
25      // 5. 数组中的所有的成员在内存中地址是连续的
26      for(int i = 0; i < 10;i++)
27      {
28          // 由于数组的每个元素都是int类型的,
29          // 因此每个成员的地址相差4字节。
30          printf("&arr[%d] = %p\n", i, &arr[i]);
31      }
32
33
34      // 6. 对数组进行访问时，编译器不会对数组的越界进行检查
35      // 因此在对数组进行操作时一定要注意越界的问题。
36      // 如果访问数组越界，相当于操作了非法的内存空间,
37      // 有可能会导致程序的运行结果不可预知，或者发生段错误。
38      //  arr[2000] = 1000; // 数组越界访问,出现段错误
39      arr[10] = 1000;  // 数组越界访问，没有出现段错误
40
41      return 0;
42  }
43
```

3.3.5 练习题

```
1. 定义一个整型数组，数组长度为10，
    从终端输入的方式对数组的每个元素进行初始化。
    将数组中的成员进行倒叙。
    比如：int arr[10] = {1,2,3,4,5,6,7,8,9,10};
        倒叙之后的结果如下所示：
        arr[10] = {10,9,8,7,6,5,4,3,2,1};

    交换两个变量的值：
    int tmp;
    tmp = a;
    a = b;
    b = tmp;

    tmp = a + b;
    b = tmp - b;
    a = tmp - b;

#include <stdio.h>
void init_array(int arr[], int len)
{
    for (int i = 0; i < len; i++)
    {
        /*
        int tmp = 0;
        scanf("%d", &tmp);
```

```
26          arr[i] = tmp;
27          */
28          scanf("%d", &arr[i]);
29      }
30  }
31
32
33  void array_flashback(int arr[], int
    len)
34  {
35  #if 0
36      for (int i = 0; i < len / 2; i++)
37      {
38          int tmp;
39          tmp = arr[i];                   //
    0 1 2 3 4
40          arr[i] = arr[len - i - 1];  //
    9 8 7 6 5
41          arr[len - i - 1] = tmp;
42      }
43  #endif
44      for (int i = 0, j = len - 1; i < j;
    i++, j--)
45      {
46          int tmp;
47          tmp = arr[i];                   //
    0 1 2 3 4
48          arr[i] = arr[j];    // 9 8 7 6
    5
49          arr[j] = tmp;
50
```

```
51      }
52  }
53
54  void print_array(int arr[], int len)
55  {
56      for (int i = 0; i < len;i++)
57      {
58          printf("%d ", arr[i]);
59      }
60      putchar('\n');
61  }
62
63  int main(int argc, const char *argv[])
64  {
65      int arr[10] = {0};
66      init_array(arr,
    sizeof(arr)/sizeof(int));
67      array_flashback(arr, sizeof(arr)/
    sizeof(int));
68
    print_array(arr,sizeof(arr)/sizeof(int)
    );
69
70      return 0;
71  }
72
```

```
1  2. 定义一个整型数组，数组长度为10，
2      从终端输入的方式对数组的每个元素进行初始
   化，
3      求获取数组中最大成员的值；
```

```
4       求数组中最大成员的下标。
5
6    #include <stdio.h>
7   void init_array(int arr[], int len)
8   {
9       for (int i = 0; i < len; i++)
10      {
11          /*
12          int tmp = 0;
13          scanf("%d", &tmp);
14          arr[i] = tmp;
15          */
16          scanf("%d", &arr[i]);
17      }
18  }
19
20  int get_array_max_value(int arr[], int
    len)
21  {
22      int max_value = arr[0];  // 假设第0
    个元素的值最大
23      for (int i = 1; i < len; i++)
24      {
25          if (max_value < arr[i])
26          {
27              max_value = arr[i]; // 把最
    大元素的值赋值给max_value
28          }
29      }
30      return max_value;
31  }
```

```c
int get_array_max_index(int arr[], int len)
{
    int max_value = arr[0];  // 假设第0个元素的值最大
    int index = 0; // 假设第0个元素最大
    for (int i = 1; i < len; i++)
    {
        if (max_value < arr[i])
        {
            max_value = arr[i]; // 把最大元素的值赋值给max_value
            index = i;
        }
    }
    return index;
}



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

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

```
60      int arr[10] = {0};
61      init_array(arr,
    sizeof(arr)/sizeof(int));
62
    print_array(arr,sizeof(arr)/sizeof(int)
    );
63
64      int max, index;
65      max = get_array_max_value(arr,
    sizeof(arr)/sizeof(int));
66      index = get_array_max_index(arr,
    sizeof(arr)/sizeof(int));
67      printf("arr[%d] = %d\n", index,
    max);
68      return 0;
69  }
70
```

4、二维数组

4.1 二维数组的概念

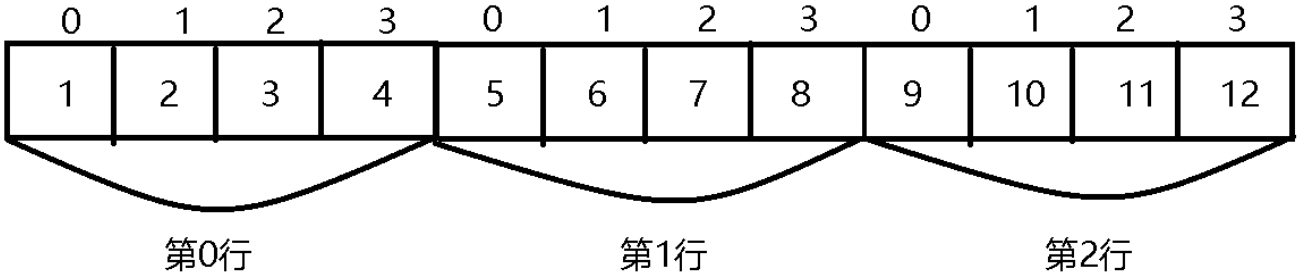
```
1  二维数组有行和列下标。
```

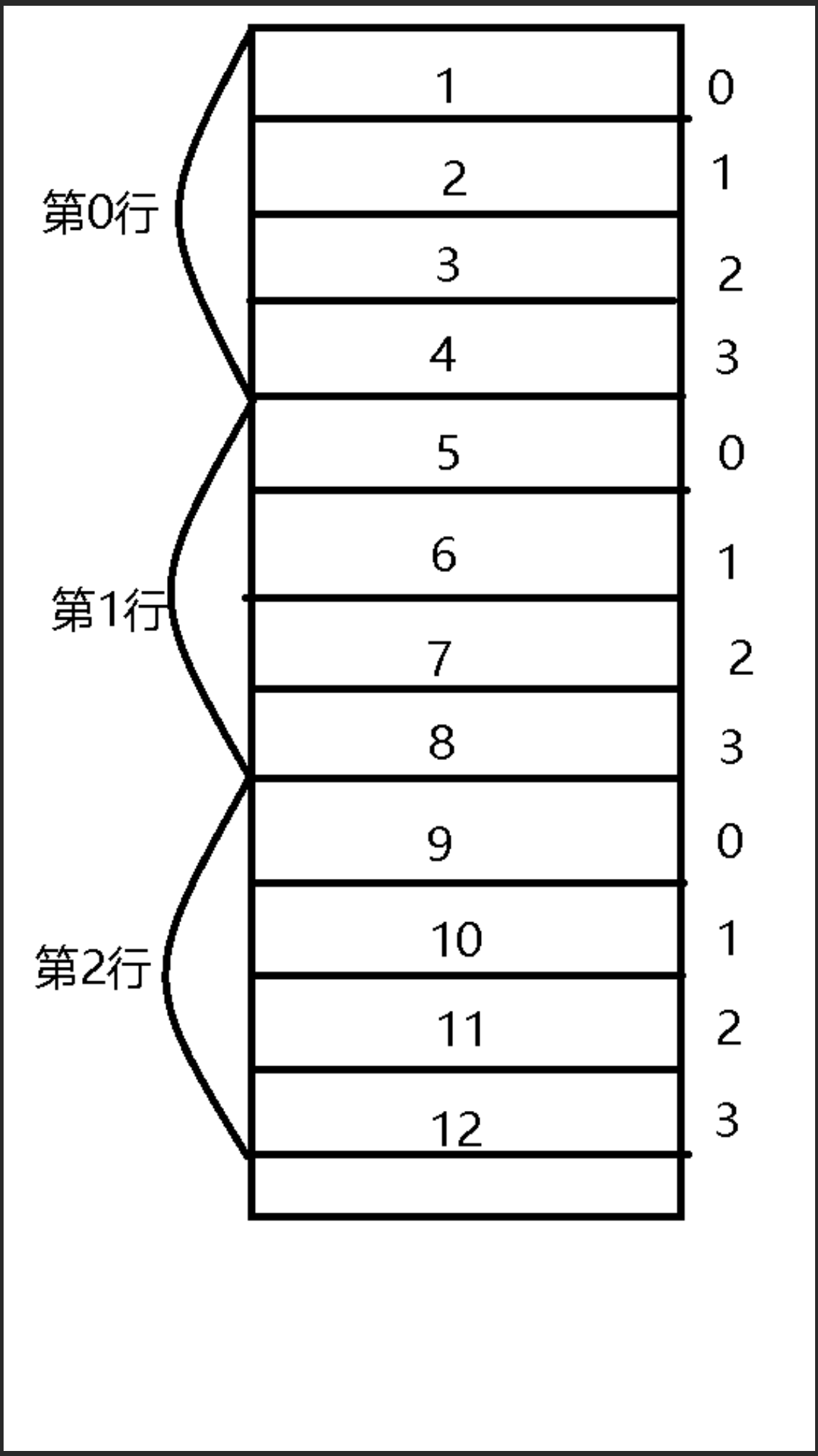
4.2 定义二维数组的格式

```
存储类型    数据类型    二维数组名[行数][列数];

特点:
        1> 二维数组的所有的成员在内存中依然是连续的
        2> 二维数组的行和列的下标都是从0开始
        3> 数组的每个成员都和定义二维数组时，数据类型一致。

        int arr[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};

如何访问数组中的每一个元素:
        二维数组名[行下标][列下标];
```

int arr[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};

行 \ 列	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12




1
2
3
4
5
6
7
8
9
10
11
12
0
1
2
3
0
1
2
3
0
1
2
3
第0行
第1行
第2行

4.3 定义二维数组并进行初始化

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{
    // 1. 定义二维数组的同时进行初始化
    short two_arr[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};

    for (int i = 0; i < 3; i++) // 遍历二维数组的行
    {
        for (int j = 0; j < 4; j++) // 遍历二维数组的列
        {
            printf("two_arr[%d][%d] = %d ", i, j, two_arr[i][j]);
        }
        putchar('\n');
    }

    // 2. 定义数组，将二维数组的所有元素初始化为0
    short two_arr2[3][4] = {0};  // 将二维数组的所有的元素初始化为0

```

```
20      // 3. 定义数组的同时进行初始化，
21      // 逐行进行初始化，如果没有指定则初始化为
    0
22      short two_arr3[3][4] =
    {1,2,3,4,5,6,7,8};
23      for (int i = 0; i < 3; i++) // 遍历
    二维数组的行
24      {
25          for (int j = 0; j < 4; j++) //
    遍历二维数组的列
26          {
27              printf("two_arr3[%d][%d] =
    %d ", i, j, two_arr3[i][j]);
28          }
29          putchar('\n');
30      }
31
32      // 4. 定义数组的同时进行初始化，指定每行
    的元素的值
33      // 每行中没有进行初始化的则初始化为0
34      short two_arr4[3][4] = {{1,2,3},
    {4,5},{6}};
35      for (int i = 0; i < 3; i++) // 遍历
    二维数组的行
36      {
37          for (int j = 0; j < 4; j++) //
    遍历二维数组的列
38          {
39              printf("two_arr4[%d][%d] =
    %d ", i, j, two_arr4[i][j]);
40          }
```

```
41          putchar('\n');
42      }
43
44      // 5. 定义二维数组时，不指定数组的行数，通过初始化时指定数组的行数
45      // 注：定义数组时不能省略列数
46      short two_arr5[][4] = {1,2,3,4,5,6,7,8,9};
47      for (int i = 0; i < 3; i++) // 遍历二维数组的行
48      {
49          for (int j = 0; j < 4; j++) // 遍历二维数组的列
50          {
51              printf("two_arr5[%d][%d] = %d ", i, j, two_arr5[i][j]);
52          }
53          putchar('\n');
54      }
55
56      // 6. 先定义二维数组，后进行初始化
57      short two_arr6[2][3]; // 局部变量，默认初始化为随机值
58      for (int i = 0; i < 2; i++)
59      {
60          for (j = 0; j < 3; j++)
61          {
62              two_arr[i][j] = i+j;
63          }
64      }
65
```

```
66      // 注：一旦定义了二维数组之后，不可以整体进行访问，赋值或者读值
67      // 数组一旦被定义之后，只能使用单个元素的方式进行访问
68      // two_arr6 = {{1,2},{3,4},{5,6}}; // 错误
69      return 0;
70  }
71
```

4.4 二维数组的特性及使用注意事项

```
1  #include <stdio.h>
2  int main(int argc, const char *argv[])
3  {
4      short two_arr[3][4] = {0};
5
6      // 1. 获取二维数组占用内存空间的大小 sizeof(short) * 行数 * 列数
7      printf("二维数组占用内存空间的大小 = %ld\n", sizeof(two_arr));
8
9      // 2. 获取二维数组有多少个成员
10     printf("二维数组成员的个数 = %ld\n", sizeof(two_arr)/sizeof(short));
11
12
13     // 3. 获取二维数组的行数
```

```
14      // sizeof(two_arr[0]) : 计算二维数组一行占用的内存空间的大小
15      // two_arr[0] : 表示二维数组的第0行的首地址
16      printf("二维数组的行数 = %ld\n", sizeof(two_arr)/sizeof(two_arr[0]));
17
18      // 4. 获取二维数组的列数
19      // two_arr[1] : 表示二维数组的第1行的首地址
20      // sizeof(two_arr[1]) : 计算二维数组一行占用的内存空间的大小
21      printf("二维数组的列数 = %ld\n", sizeof(two_arr[1])/sizeof(short));
22
23
24      // 5. 数组名相关的使用
25      // 5.1 二维数组名是一个常量，不可以进行赋值运算，
26      // 也不可以进行自增自减运算
27
28      // 5.2 二维数组的名表示二维数组的首地址，此地址和第0行第0列元素的地址相等。
29      printf("two_arr = %p\n", two_arr);
30      printf("&two_arr[0][0] = %p\n", &two_arr[0][0]);
31
32      // 5.3 数组名[行下标] : 每一行的第0个元素的地址
33      printf("two_arr[0] = %p\n", two_arr[0]);
```

```
34      printf("&two_arr[0][0] = %p\n", &two_arr[0][0]);
35
36      printf("two_arr[1] = %p\n", two_arr[1]);
37      printf("&two_arr[1][0] = %p\n", &two_arr[1][0]);
38
39      printf("two_arr[2] = %p\n", two_arr[2]);
40      printf("&two_arr[2][0] = %p\n", &two_arr[2][0]);
41
42      // 6. 二维数组越界的问题，编译器不会检查数组越界的问题，
43      // 需要程序员在编写代码时，要思考代码越界的问题。
44
45
46      return 0;
47  }
48
```

```
1  作业：
2      将今天的内容消化吸收，逻辑思维能力。
```

```
明天授课内容:
    冒泡排序
    字符数组(strlen,strcpy,strcmp,strcat)
    指针
```