1、指针

1.1 指针的概念

```
1   内存访问的最小单位为字节，1字节是8bit位。
2   内存的访问本质是通过地址进行访问的，每个字节都有一个唯一的地址。
3   内存的地址是连续的。
4
5   地址可以理解为指针。
6
7   专门用来存储指针(地址)的变量，称为指针变量，
8   要想使用一个变量存储内存的地址，因此需要定义指针类型的变量。
```

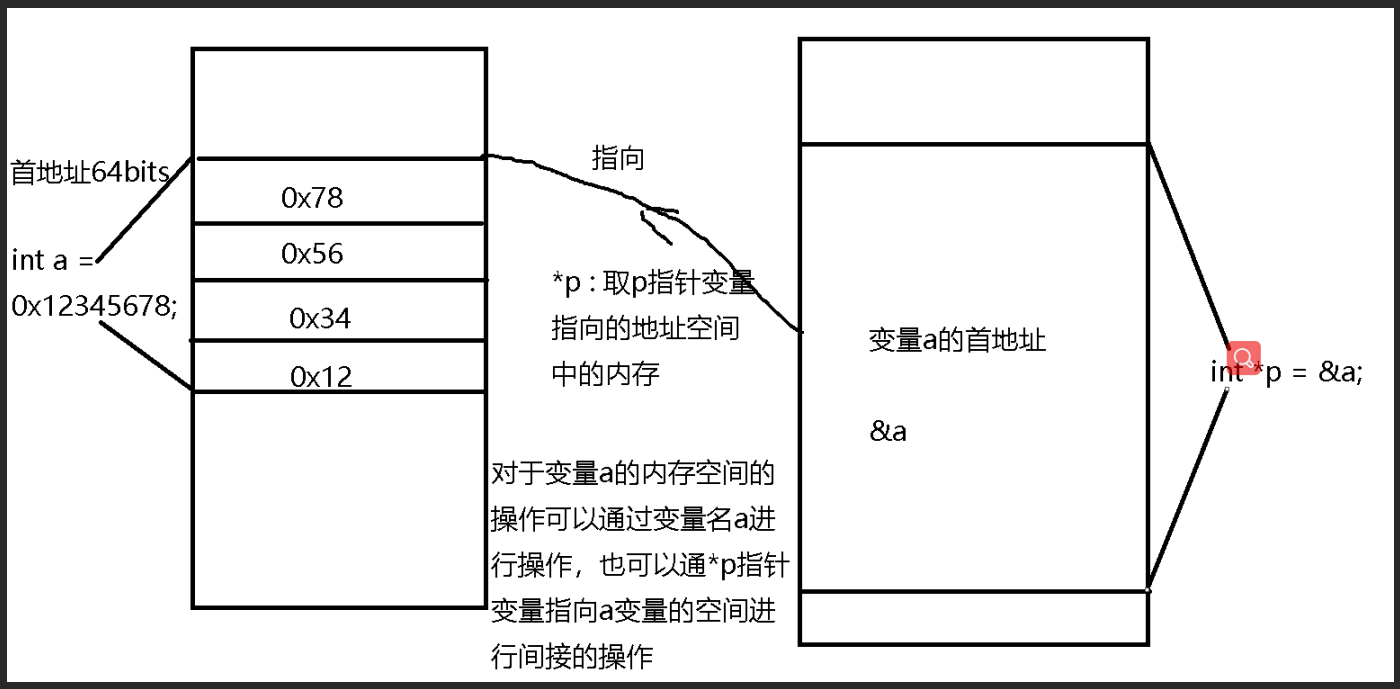
```
9
10   定义指针变量和定义普通变量的区别：
11   普通变量就是存储的普通的数据，比如整型数据，字符数据。
12   指针变量就是用来存储内存地址的变量，因此要想使用一个变量存储内存的地址，
13         需要定义指针类型的变量。
14
15         地址：   访问内存的一个编号
16         指针：   用来存储地址的指针变量
17
18   定义指针：就是再说定义指针类型的变量。
19
20   指针：地址
21   指针：指针变量
22   指针：数据类型(指针类型)
```

1.2 定义指针类型的变量及初始化

```
1  1. 定义指针变量的格式：
2        存储类型  数据类型   *指针变量名;
3                          |     |       |----> 指针变量名，需要使用地址进行初始化
4                          |     |----> 定义指针类型的变量
5                          |---> 数据类型(基本类型/构造类型)
6        存储类型  数据类型*   指针变量名;
```

```
                    |    |      |----> 指针变量名，需要使用地址进行初始化
                    |    |----> 定义指针类型的变量
                    |---> 数据类型(基本类型/构造类型)
    存储类型 数据类型   *   指针变量名;
                    |    |      |----> 指针变量名，需要使用地址进行初始化
                    |    |----> 定义指针类型的变量
                    |---> 数据类型(基本类型/构造类型)
    "数据类型 *"看成一个整体,指针变量名为"数据类型 *",指针类型
2. 指针变量的初始化:
    1> 定义指针变量的同时进行初始化
        int a = 100;
        int *p = &a;
    2> 先定义指针变量，后进行初始化
        int a = 200;
        int *p = NULL;  // NULL : 0地址
                    // 防止野指针的出现，如果定义指针变量时，没有初始化，
                    // 指针变量中存储的就是一个随机的地址，访问此地址，
                    // 就会访问非法的内存空间，导致段错误的出现，
                    // 野指针:  指针变量指向的地址空间值不确定。
        p = &a;
```

```
27
28 *和&的作用：都属于单目运算符
29      &  ：  放到变量名的前边表示对变量进行取地址运算。
30      *  ：
31          1> 定义变量时，放到数据类型和指针变量名之间，表示定义指针类型的变量。
32              int *p = &a;
33          2> 放到指针变量名前，表示取指针变量指向的空间的内容。
34              int b = *p; // 将指针变量指向的内存空间的内容赋值给变量b。
35              int c = 200;
36              *p = c;      // 将变量c的值赋值给直播变量p指向的内存空间中，
37                           // 间接修改变量a对应内存空的值。
```



1.3 指针类型变量的定义及初始化的案例

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{
    // 定义一个普通的变量，普通变量的访问通过变量名进行访问变量对应的内存空间
    int a = 100;

    // 定义int类型的指针变量，指向int类型的变量
    // 定义指针变量的同时进行初始化
    int *p = &a;


    printf("修改之前: a = %d\n", a);
    printf("修改之前: *p = %d\n", *p);
    // 使用指针变量修改指针变量指向的对应的变量的内存空间
    *p = 200;    // 通过指针变量间接修改变量a中的值
    printf("修改之后: a = %d\n", a);
    printf("修改之后: *p = %d\n", *p);

    // 指针变量p中存放的是变量a对应内存的首地址
    printf("变量a的地址 = %p\n", &a);
```

```c
    printf("指针变量p中存放的地址 = %p\n", p);

    int b = *p;       // 将指针变量p指向的内存中的数据读到到b变量中
    printf("b = %d\n", b);

    /* ------------------------------------*/
    // 先定义指针变量后进行初始化
    int *p1;          // 如果定义时没有初始化，也没有指向NULL，
                      // p1指针变量中存放的就是随机地址，就是一个野指针
                      // 对野指针进行操作，可能会出现段错误。
/*
    *p1 = 200;        // 操作了非法的内存空间，可能会导致段错误
    printf("*p1 = %d\n", *p1);  //
*/

    // 避免野指针的出现，定义指针变量时，初始化为NULL.
    int *p2 = NULL;
    // *p2 = 200;          // 一定会报段错误

    p2 = &b;          // 对指针变量进行赋值
    *p2 = 300;

    return 0;
```

```
}

```

练习题：
    定义unsigned int类型指针的变量，
    并将指针类型的变量指向一个unsigned int类型 普通的变量，
    对其进行初始化为0x12345678。

    定义一个unsigned char类型的指针变量，让其指向unsigned int类型的变量。

```
    unsigned int num ;
    unsigned char *c_p = (unsigned char *)&num;
        // &num : 表示一个unsigned int*类型的地址
        // (unsigned char *)&num : 表示一个unsigned char *类型的地址。

    *(c_p+0)
    *(c_p+1)
    *(c_p+2)
    *(c_p+3)        读取对应地址中存储的数据。
```

```
    // 在进行赋值操作时，考虑等会左边表达式的类型，
    // 相同类型直接的变量才可以进行赋值的操作。

    int a = int类型的常量/int类型的变量;
```

```
          |----> int类型的变量，使用int类型的数据赋值
    int *p = int类型的地址/int类型的变量取地址/int*类型的指针变量;
           |----> int *类型的指针变量，使用一个int类型的地址进行赋值
    short *p1 = short类型的地址/short类型的变量取地址/short*类型的指针变量;
             |----> short *类型的指针变量，使用一个short类型的地址进行赋值

    p = int类型的地址/int类型的变量取地址/int*类型的指针变量;
    |-----> int*类型的指针变量

    (*p) = int类型的常量/int类型的变量;
     |----> p指针变量指向的int类型的地址空间，看成int类型的普通变量

    int b = (*p);
          |     |----> p指针变量指向的int类型的地址空间，看成int类型的普通变量
          |-----> int类型的普通变量

```

```
0x40008000  --> 整型常量
unsigned int *p = 0x40008000;        // 错误，类型不同不可以进行赋值
                |          |----> 整型常量
                |----> 指向变量

unsigned int *p = (unsigned int *)0x40008000;   // OK
                |             |----> unsigned int *类型的地址
                |----> 指针变量

```

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{
    unsigned int *p = NULL;
    unsigned int a = 0;
    p = &a;
    *p = 0x12345678;

    unsigned char *q =(unsigned char *)&a;
    printf("*(q + 0) = %#x\n", *(q + 0));
    printf("*(q + 1) = %#x\n", *(q + 1));
    printf("*(q + 2) = %#x\n", *(q + 2));
```

```
14      printf("*(q + 3) = %#x\n", *(q +
    3));
15
16      return 0;
17  }
18
```

1.4 指针变量占用内存空间的大小

```
1  32位操作系统的寻址空间为：0x0000_0000 ~
   0xFFFF_FFFF(0-4G)
2      32位操作系统的地址占32bits,及地址占4字节
   空间。
3
4  64位操作系统的寻址空间为：
   0x0000_0000_0000_0000 ~
   0xFFFF_FFFF_FFFF_FFFF(0-很大)
5      64位操作系统的理论的寻址空间很大，但是实际
   在设计硬件时就
6      已经规定了内存的最大寻址空间。
7
8      64位操作系统的地址占64bits,及地址占8字节
   空间。
9
10 32位操作系统的指针变量占用多大的内存空间：4字
   节，跟指针的类型无关
11 64位操作系统的指针变量占用多大的内存空间：8字
   节，跟指针的类型无关
```

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    printf("char * type size = %ld\n", sizeof(char *));
    printf("short * type size = %ld\n", sizeof(short *));
    printf("int * type size = %ld\n", sizeof(int *));
    printf("long * type size = %ld\n", sizeof(long *));
    printf("long long * type size = %ld\n", sizeof(long long *));
    printf("float * type size = %ld\n", sizeof(float *));
    printf("double * type size = %ld\n", sizeof(double *));
    printf("long double * type size = %ld\n", sizeof(long double *));

    return 0;
}

```

1.5 指针变量进行算数运算

```
指针变量中存放的是一个地址，地址也是一个无符号整型的数据，
地址也可以进行算数运算。

    算数运算：+ -   （不是两个地址相加，而是加减一个偏移量）
    单目运算符：  ++ --
    赋值：= += -=
    条件运算符 ：  !=    == （操作的是否同一块地址空间）
```

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{
    int *p = NULL;
    int arr[10] = {111,222,333,444,555,666,777,888,999,123};
    // 让指针变量p指向arr数组的第0个元素
    p = arr;    // 等价 p = &arr[0];
    printf("arr[0] = %d\n", arr[0]);
    printf("*p = %d\n", *p);

    int *q = NULL;
    // 让指针变量指向arr数组的第5个元素
    q = arr + 5;      // 等价于 q = &arr[5];
    // 指针类型的变量进行加减运算时，偏移的是“指针变量指向类型大小 * 偏移量”的大小
    printf("arr[5] = %d\n", arr[5]);
```

```c
    printf("*q = %d\n", *q);

    return 0;
}

```

练习题：

定义一个unsigned short类型的指针变量，

定义一个unsigned short类型的数组，有10个成员，

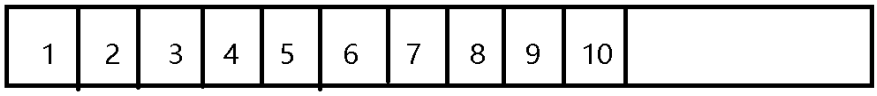
让指针变量指向数组的首地址。使用地址偏移的方式，打印数组中所有成员的值。

在定义一个指针变量指向数组的尾，

使用地址偏移的方式，打印数组中的所有成员的值。

unsigned short a[10] = {1,2,3,4,5,6,7,8,9,10};

a数组的首地址



每个成员占2字节的空间

a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] --》通过数组名[下标]访问数组的每个成员

unsinged short *p = a; //unsigned short *p = &a[0];

*p *(p+1) *(p+2) *(p+3) ..... 指针变量p中的地址没有被修改

*(p++) <==> *p++ ---> 先取*p指向空间的内容，p在进行加加运算，p中的值被修改

*++p

```c
#include <stdio.h>

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

```

```
5       unsigned short a[10] =
    {1,2,3,4,5,6,7,8,9,10};
6       unsigned short *p = a; // 让p指针数组
    a的首地址
7
8       for (int i = 0; i < 10; i++)
9       {
10          // 通过数组名和下标的方式访问数组中的
    每个元素的值。
11          printf("a[%d] = %u\n", i,
    a[i]);
12      }
13
14      for (int i = 0; i < 10; i++)
15      {
16          // 通过指针变量指向数组之后，使用指针
    变量通过地址偏移的方式，
17          // 访问数组中的所有的成员，注意不要越
    界访问
18          printf("a[%d] = %u\n", i,
    *p++);
19          // *p++ : 改变p指针变量的指向
20      }
21
22      // 修正指针变量p的指向
23      p = &a[0];  // p -= 10;
24      for (int i = 0; i < 10; i++)
25      {
26          // 通过指针变量指向数组之后，使用指针
    变量通过地址偏移的方式，
```

```
27          // 访问数组中的所有的成员，注意不要越界访问
28          printf("a[%d] = %u\n", i, *(p + i));
29          // *(p+i) : 没有改变p的指向，指向数组的首地址
30      }
31
32      // 定义指针变量指向数组的尾元素
33      unsigned short *q = p + sizeof(a)/sizeof(unsigned short) - 1;
34
35      for (int i = 0 ; i < 10; i++)
36      {
37          printf("a[%d] = %u\n", 9 - i, *(q--));
38      }
39
40      return 0;
41  }
42
```

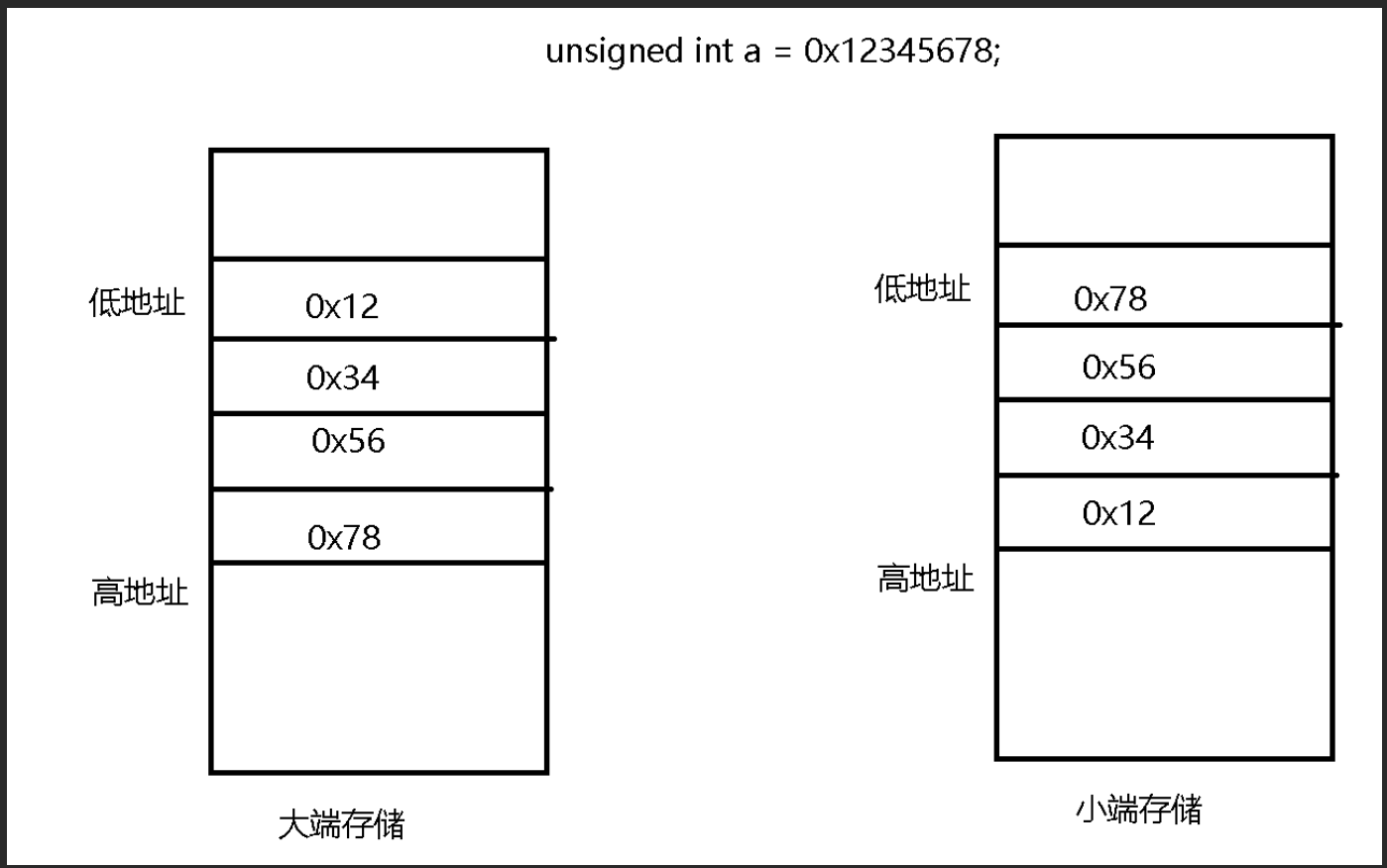
1.6 大小端存储的问题

内存的访问的最小单位为字节，部分数据类型定义变量时分配的空间大于1字节。
对于大于1字节的数据的存储，最终也要拆分成1个字节1个字节的存储，
这样就会涉及到数据大小端存储的问题。

大端存储：低地址存放数据的高字节(高有效位)；高地址存放数据的低字节(低有效位)；
小端存储：低地址存放数据的低字节(低有效位)；高地址存放数据的高字节(高有效位)。

大小端存储的问题，只跟单个的数据有关，更多个数据在内存中的存储先后无关。

网络设备一般使用的是大端存储；其他的设备一般使用是的小端存储。




```
练习题(笔试题)：验证操作系统的大小端?
思路：定义一个unsigned int类型的变量。
    在定义一个unsigned char类型的指针变量，指向unsigned int类型的变量。
    通过unsigned char类型的指针变量将数据按照字节进行读取。
#include <stdio.h>
int main(int argc, const char *argv[])
{
    unsigned int little = 0x12345678;
    unsigned char *lit_p = (unsigned char *)&little;

    for (int i = 0; i < sizeof(unsigned int) / sizeof(unsigned char); i++)
    {
```

```
13          printf("%p -- %#x\n", lit_p +
    i, *(lit_p + i));
14      }
15      // 0x78U : 无符号的整型
16      if (*lit_p == 0x78U)
17      {
18          puts("小端存储");
19      }
20      else
21      {
22          puts("大端存储");
23      }
24
25      return 0;
26  }
27
```

1.7 一级指针和一维数组的关系

```
int a = 100;
int *p = &a;            // p --> int *类型的指针变量
                        // p + 1 : 表示偏移int类型大小的空间，偏移4字节
int arr[5] = {0};
                        // arr:表示数组的首地址， 数组每个成员都是int类型
                        // 数组名可以看成int *类型指针
                        // arr + 1 : 表示偏移一个数组元素大小的空间，
                        //                  数组元素是int类型，因此偏移4字节.

指针变量和数组名的区别:
    指针变量是一个变量，可以修改指针变量的指向;
    数组名是一个常量，不可以被修改。
```

```c
#include <stdio.h>

void print_arr(int s[], int len)
{
    /*
    for(int i = 0; i < len; i++)
    {
        printf("%d ", s[i]);
    }
    printf("\n");
    */
```

```
12      for(int i = 0; i < len; i++)
13      {
14          printf("%d ", *(s + i));
15      }
16      printf("\n");
17  }
18
19  // 函数的参数为指针类型，通过指针和数组进行关联
20  void printf_arr_p(int *p, int len)
21  {
22  #if 0
23      for (int i = 0; i < 5; i++)
24      {
25          printf("%d ", p[i]);
26      }
27      putchar('\n');
28  #endif
29
30  #if 0
31      // 3.2 使用指针加地址偏移的方式访问数组的所有的成员
32      for (int i = 0; i < 5; i++)
33      {
34          printf("%d ", *(p+i)); // 不会改变p的指向
35      }
36      putchar('\n');
37  #endif
38
39      for (int i = 0; i < 5; i++)
```

```c
    {
        printf("%d ", *p++);  // 会改变p的指向
    }
    putchar('\n');

}


int main(int argc, const char *argv[])
{
    // 定义一个数组
    int s[5] = {111,222,333,444,555};

    // 研究数组的名字:
    // 1. 数组名表示数组的首地址，数组名是一个常量不可以被修改
    //  数组名可以看成一个int *类型的地址，跟数组中成员的类型有关
    // 2. 将数组名当成一个int *类型的地址使用
    printf("s = %p\n", s);
    printf("s + 1 = %p\n", s + 1);  // 加1，加指针类型大小，加int类型的大小

    printf("*s = %d\n", *s);
    printf("*s+1 = %d\n", *s+1);  // 取*s指向的内容加1
    printf("*(s+1) = %d\n", *(s+1));  // 取下一个元素的内容。

```

```
64      // 总结：通过数组名访问数组中的元素的值的方式
65      // 数组名[下标] <==> *(s+下标)
66      // &数组名[下标]   <==>  (s + 下标)
67
68      // 3. 定义一个int类型的指针，指向数组的首地址，
69      // 通过指针的方式访问数组中的所有的成员
70      int *p = s;
71
72      // 3.1 将指针变量当成数组名进行使用
73      for (int i = 0; i < 5; i++)
74      {
75          printf("%d ", p[i]);
76      }
77      putchar('\n');
78
79      // 3.2 使用指针加地址偏移的方式访问数组的所有的成员
80      for (int i = 0; i < 5; i++)
81      {
82          printf("%d ", *(p+i)); // 不会改变p的指向
83      }
84      putchar('\n');
85
86      for (int i = 0; i < 5; i++)
87      {
88          printf("%d ", *p++);  // 会改变p的指向
```

```
            // 如果后续还想使用指针变量p，访问s数组，需要修改指针变量p指向
        }
        putchar(10);


        // 总结：一级指针和一维数组的关系
        // 数组名[下标] <=> *(数组名 + 下标) <=> *(指针变量名+下标)
        // <=> 指针变量名[下标]

        // &数组名[下标] <=> 数组名+下标 <=> 指针变量名+下标
        // <=> &指针变量名[下标]

        printf_arr_p(s, 5);


        return 0;
}

```

```
练习题：重新my_strlen,my_strcpy,my_strcat,my_strcmp函数
当函数的形参需要传递一个一维数组时，可以使用一级指针的方式作为函数的形参。

#include <stdio.h>
int my_strlen(char *s)
{
    int len = 0;
```

```
 8      while(*s != '\0')
 9      {
10          len++;
11          s++;
12      }
13      return len;
14  }
15  char *my_strcpy(char *s1, char *s2)
16  {
17      char *tmp = s1;
18      while (*s2 != '\0')
19      {
20          *s1 = *s2;
21          s1++;
22          s2++;
23      }
24      *s1 = '\0';
25      return tmp;
26
27  }
28  char *my_strcat(char *s1, char *s2)
29  {
30      char *tmp = s1;
31      while (*s1 != '\0')
32      {
33          s1++;
34      }
35      while (*s2 != '\0')
36      {
37          *s1 = *s2;
38          s1++;
```

```
39          s2++;
40      }
41      *s1 = '\0';
42      return tmp;
43  }
44  int my_strcmp(char *s1, char *s2)
45  {
46      while (*s1 != '\0')
47      {
48          if (*s1 != *s2)
49          {
50              return *s1 - *s2;
51          }
52          s1++;
53          s2++;
54      }
55      return *s1 - *s2;
56  }
57
58  int main(int argc, const char *argv[])
59  {
60      char s1[20] = "hello";
61      char s2[20] = "world";
62      printf("len = %d\n",
    my_strlen(s1));
63      printf("cmp = %d\n", my_strcmp(s1,
    s2));
64      printf("cpy = %s\n", my_strcpy(s1,
    s2));
65      printf("cat = %s\n", my_strcat(s1,
    s2));
```

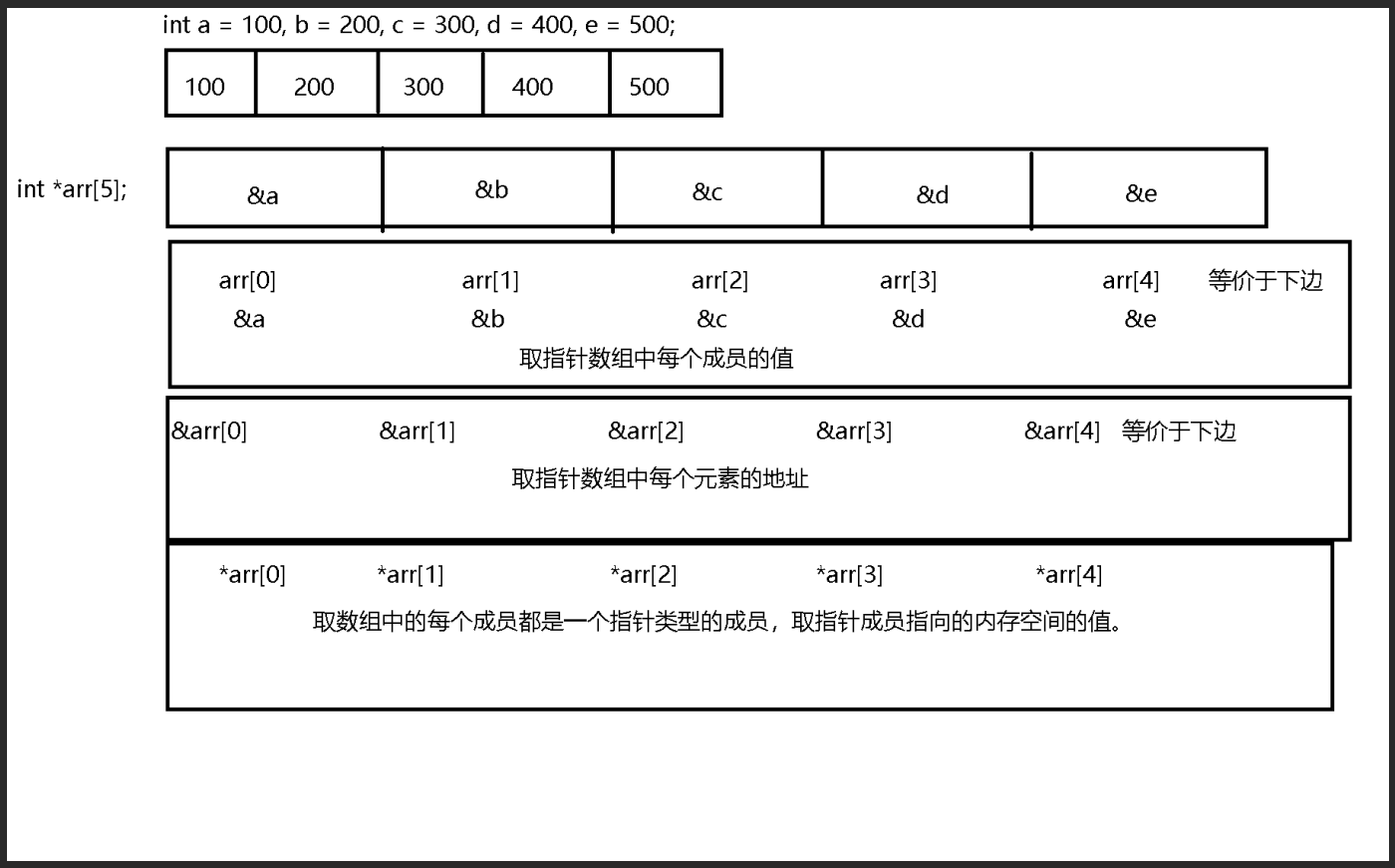
```
66      return 0;
67  }
68
```

1.8 指针数组

```
1  1. 定义指针数组的格式
2       数据类型 * 数组名[数组成员个数];
3
4  2. 特征
5       1> 指针数组本质是一个数组,
6       2> 数组的每个成员是一个"数据类型 *"指针类型,
7              及数组的每个成员存放的都是一个地址(指针)。
```

```
 1  #include <stdio.h>
 2  int main(int argc, const char *argv[])
 3  {
 4      // 1. 定义指针数组, 并进行初始化
 5      int a = 100, b = 200, c = 300, d = 400, e = 500;
 6      int *p_arr[5] = {&a, &b, &c, &d, &e};
 7
 8      printf("p_arr size = %ld\n", sizeof(p_arr));
 9      printf("p_arr 成员个数 = %ld\n", sizeof(p_arr)/sizeof(int *));
10
```

```
11      for (int i = 0; i < 5; i++)
12      {
13          // 访问指针数组每个成员的值，指针数组成员的值是一个地址
14          printf("p_arr[%d] = %p\n", i, p_arr[i]);
15      }
16      printf("----------------------------\n");
17      for (int i = 0; i < 5; i++)
18      {
19          // 打印指针数组每个元素的地址，数组的每个成员都是指针类型的变量
20          printf("&p_arr[%d] = %p\n", i, &p_arr[i]);
21
22      }
23      printf("----------------------------\n");
24      for (int i = 0; i < 5; i++)
25      {
26          // 访问指针数组每个成员指向的地址空间的内容
27          printf("*p_arr[%d] = %d\n", i, *p_arr[i]);
28      }
29
30      return 0;
31  }
32
```



明天授课内容：

1. 指针数组
2. 数组指针
3. 二级指针
4. 指针数组和二级指针关联
5. 数组指针和二维数组关联
6. 二级指针和二维数组：没有关系

7. 函数。