知识点1【结构体概述】（了解）

1、结构体 类型的定义方式

知识点2【结构体变量的初始化】（了解）

清空结构体变量

键盘给结构体变量中成员赋值

可以单独操作结构体中的成员

相同类型的结构体变量 之前赋值

知识点3【结构体嵌套结构体】（了解）

知识点4【结构体数组】（了解）

1、结构体数组：本质是数组 每个元素是结构体

2、键盘给结构体赋值

知识点5【结构体指针变量】（了解）

1、结构体指针变量 本质是指针变量 保存的是结构体变量的地址

2、结构体数组元素的指针变量

知识点6【结构体的指针成员】（了解）

1、指针成员：指针变量 作为结构体中的成员

2、结构体的指针成员 指向堆区

3、相同类型的结构体变量可以整体赋值

如果结构体中有指针成员 尽量使用深拷贝。

4、结构体在堆区 结构体的指针成员指向堆区。（了解）

5、结构体指针数组在堆区、结构体在堆区、结构指针成员在堆区（了解）

知识点7【结构体的对齐规则】（了解）

知识点1【结构体概述】（了解）

将多种数据结构封装在一起 形成新的结构胶结构体

每种数据结构 都有自己独立的空间。

结构体的关键字struct.

1、结构体 类型的定义方式

推荐 1、先定义结构体类型，再定义结构体变量

```
struct stu
{
    int num;
    char name[32];
};

struct stu lucy;
```

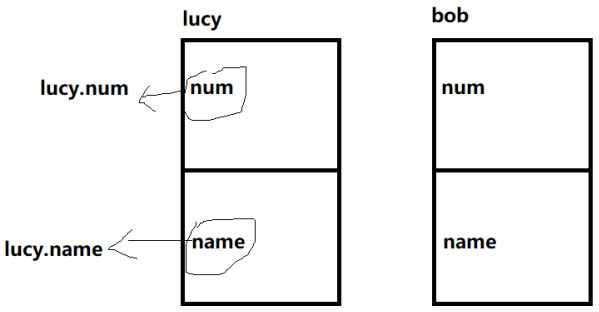
2、定义类型的同时 定义结构体变量

```
struct stu
{
    int num;
    char name[32];
} lucy ;

struct stu bob;
```

3、定义一次性结构体类型

```
struct
{
    int num;
    char name[32];
} lucy ;
```

```
//定义结构体类型 系统是不分配空间
struct stu
{
    //定义类型是 不要给成员赋值
    //int num = 10; //error
    int num; //结构体的成员
    char name[32];
}; //必须有分号
void test01()
{
    struct stu lucy;
}
```



struct stu bob;

```
//定义结构体类型 系统是不分配空间
struct stu
{
    //定义类型是 不要给成员赋值
    //int num = 10; //error
    int num; //结构体的成员
    char name[32];
}; //必须有分号
void test01()
{
    struct stu lucy;

    //访问结构体成员  必须遵循成员自身的类型
    printf("%d %s\n", lucy.num, lucy.name);
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 00_code.c
edu@edu:~/work/c/day07$ ./a.out
1835627636 e
edu@edu:~/work/c/day07$
```

知识点2【结构体变量的初始化】（了解）

```
void test02()
{
    //结构体变量的初始化  必须遵循成员的顺序以及自身的类型
    struct stu lucy = {100, "lucy"};

    printf("%d %s\n", lucy.num, lucy.name);
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 00_code.c
edu@edu:~/work/c/day07$ ./a.out
100 lucy
edu@edu:~/work/c/day07$
```

清空结构体变量

```c
#include <string.h>
void test03()
{
    struct stu lucy;
    //使用memset给结构体变量清空
    memset(&lucy, 0, sizeof(lucy));

    printf("%d %s\n", lucy.num, lucy.name);
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 00_code.c
edu@edu:~/work/c/day07$ ./a.out
0
edu@edu:~/work/c/day07$
```

键盘给结构体变量中成员赋值

```c
#include <string.h>
void test03()
{
    struct stu lucy;
    //使用memset给结构体变量清空
    memset(&lucy, 0, sizeof(lucy));

    printf("请输入num name:");
    //&lucy.num 取的是lucy中num成员的地址
    scanf("%d %s", &lucy.num, lucy.name);

    printf("%d %s\n", lucy.num, lucy.name);
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 00_code.c
edu@edu:~/work/c/day07$ ./a.out
请输入num name:100 lucy
100 lucy
edu@edu:~/work/c/day07$
```

可以单独操作结构体中的成员

```c
void test04()
{
    struct stu lucy = {100, "lucy"};

    lucy.num += 100;
    //字符串数组 必须使用 字符串操作函数 进行操作
    strcpy(lucy.name, "bob");

    printf("%d %s\n", lucy.num, lucy.name);
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 00_code.c
edu@edu:~/work/c/day07$ ./a.out
200 bob
edu@edu:~/work/c/day07$
```

相同类型的结构体变量 之前赋值

```c
void test04()
{
    struct stu lucy = {100, "lucy"};
    struct stu bob;

#if 0
    //方式一：逐个成员赋值
    bob.num = lucy.num;
    strcpy(bob.name, lucy.name);
#elif 0
    //方式二：相同类型的结构体变量 可以直接赋值 （推荐）
    bob = lucy;
#else
    //方式三：相同类型的结构体变量 内存赋值
    memcpy(&bob, &lucy, sizeof(lucy));
#endif

    printf("%d %s\n", bob.num, bob.name);
}
```

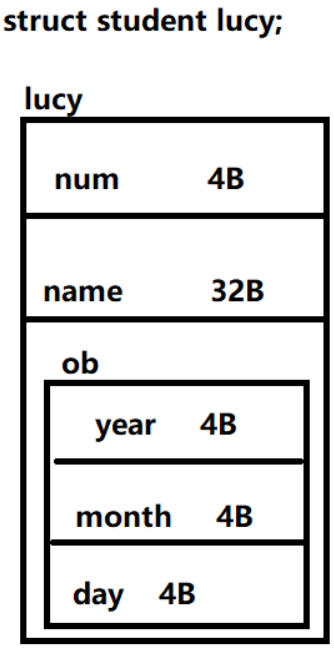
```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 00_code.c
edu@edu:~/work/c/day07$ ./a.out
100 lucy
edu@edu:~/work/c/day07$
```

知识点3【结构体嵌套结构体】（了解）

```c
struct student
{
    int num;
    char name[32];
    struct Date ob;
};

void test05()
{
    struct student lucy = {100,"lucy", {2021, 7, 30}};
    //结构体嵌套结构体 访问到最底层
    printf("num = %d, name=%s\n", lucy.num, lucy.name);
    printf("%d %d %d\n", lucy.ob.year, lucy.ob.month, lucy.ob.day);
}
int main(int argc, char const *argv[])
{
    test05();
    return 0;
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 00_code.c
[sudo] edu 的密码：
edu@edu:~/work/c/day07$ ./a.out
num = 100, name=lucy
2021 7 30
edu@edu:~/work/c/day07$
```

```c
struct Date
{
    int year;
    int month;
    int day;
};

struct student
{
    int num;
    char name[32];
    struct Date ob;
};
```



知识点4【结构体数组】（了解）

1、结构体数组：本质是数组 每个元素是结构体

```c
struct stu2
{
    int num; //结构体的成员
    char name[32];
};
void test06()
{
    struct stu2 edu[5] = {{100, "lucy"}, {101, "bob"}, {102, "tom"},\
    {103, "德玛西亚"}, {104, "小炮"}};
    //memset(edu, 0,sizeof(edu));//将整个结构体数组清0
    int n = sizeof(edu) / sizeof(edu[0]);

    int i = 0;
    for (i = 0; i < n; i++)
    {
        printf("%d %s\n", edu[i].num, edu[i].name);
    }
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 00_code.c
edu@edu:~/work/c/day07$ ./a.out
100 lucy
101 bob
102 tom
103 德玛西亚
104 小炮
edu@edu:~/work/c/day07$
```

2、键盘给结构体赋值

```c
void test07()
{
    struct stu2 edu[5];
    memset(edu, 0, sizeof(edu)); //将整个结构体数组清0
    int n = sizeof(edu) / sizeof(edu[0]);

    printf("请输入%d个学生信息num name\n", n);
    //获取键盘输入
    int i = 0;
    for (i = 0; i < n; i++)
    {
        scanf("%d %s", &edu[i].num, edu[i].name);
    }

    for (i = 0; i < n; i++)
    {
        printf("%d %s\n", edu[i].num, edu[i].name);
    }
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 00_code.c
edu@edu:~/work/c/day07$ ./a.out
请输入5个学生信息num name
101 lucy
102 bob
103 tom
104 德玛西亚
105 小法
101 lucy
102 bob
103 tom
104 德玛西亚
105 小法
edu@edu:~/work/c/day07$
```

知识点5【结构体指针变量】（了解）

1、结构体指针变量 本质是指针变量 保存的是结构体变量的地址

```c
#include <stdio.h>
#include <string.h>
struct stu
{
    int num; //结构体的成员
    char name[32];
};
void test01()
{
    struct stu lucy = {100, "lucy"};
    //定义一个指针变量p保存lucy的地址
    struct stu *p = &lucy;

    //*p == lucy
    printf("%d %s\n", lucy.num, lucy.name);
    printf("%d %s\n", (*p).num, (*p).name);
    //通过 结构体指针变量 直接访问 结构体空间中的成员
    printf("%d %s\n", p->num, p->name);
    //不管.还是->  就看符号 左边如果是变量 使用.  如果左边是结构体地址 使用->
    printf("%d %s\n", (&lucy)->num, (&lucy)->name);
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 01_code.c
edu@edu:~/work/c/day07$ ./a.out
100 lucy
100 lucy
100 lucy
100 lucy
edu@edu:~/work/c/day07$
```

2、结构体数组元素的指针变量

```c
void input_stu_array(struct stu *arr, int n)
{
    printf("请输入%d个学员的信息num name\n", n);
    int i = 0;
    for (i = 0; i < n; i++)
    {
        printf("请输入第%d个学生的信息:", i + 1);
```

```
        //scanf("%d %s", &arr[i].num, arr[i].name);
        scanf("%d %s", &(arr + i)->num, (arr + i)->name);
    }
    return;
}
void sort_stu_array(struct stu *arr, int n)
{
    int i = 0;
    for (i = 0; i < n - 1; i++)
    {
        int min = i;
        int j = min + 1;
        for (; j < n; j++)
        {
            //if(strcmp(arr[min].name, arr[j].name) > 0)
            if (arr[min].num > arr[j].num)
                min = j;
        }

        if (i != min)
        {
            struct stu tmp = arr[min];
            arr[min] = arr[i];
            arr[i] = tmp;
        }
    }
    return;
}
void print_stu_array(struct stu *arr, int n)
{
    int i = 0;
    for (i = 0; i < n; i++)
    {
        // printf("%d %s\n", arr[i].num, arr[i].name);
        printf("%d %s\n", (arr + i)->num, (arr + i)->name);
    }
    return;
}
void test02()
{
```

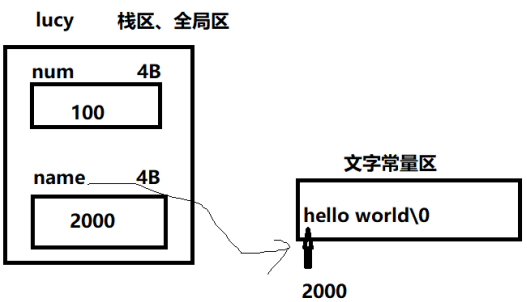
```c
    struct stu edu[5];
    memset(edu, 0, sizeof(edu));
    int n = sizeof(edu) / sizeof(edu[0]);

    //封装函数获取键盘输入
    input_stu_array(edu, n);

    //对结构体数组排序
    sort_stu_array(edu, n);

    //遍历结构体数组的内容
    print_stu_array(edu, n);
}
```

```
edu@edu:~/work/c/day07$ sudo gcc 01_code.c
edu@edu:~/work/c/day07$ ./a.out
请输入5个学员的信息num name
请输入第1个学生的信息:103 lucy
请输入第2个学生的信息:101 bob
请输入第3个学生的信息:105 tom
请输入第4个学生的信息:104 德玛
请输入第5个学生的信息:102 小法
101 bob
102 小法
103 lucy
104 德玛
105 tom
edu@edu:~/work/c/day07$
```

知识点6【结构体的指针成员】（了解）

1、指针成员：指针变量 作为结构体中的成员

```c
struct stu
{
    int num;
    char *name; //指针成员
};
void test01()
{
    struct stu lucy = {100, "hello world"};
```



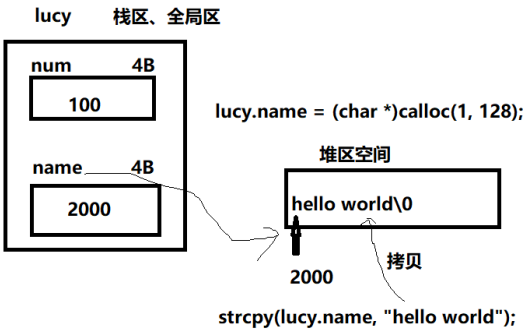
```c
#include <stdio.h>
#include <string.h>
struct stu
{
    int num;
    char *name; //指针成员
};
void test01()
{
    struct stu lucy = {100, "hello world"};

    printf("%d %s\n", lucy.num, lucy.name);
    printf("%c\n", lucy.name[1]);
    lucy.name[1] = 'E'; //Lucy.name[1]文字常量区的第1个字符 被赋值 出现段错误
    printf("%s\n", lucy.name);
}
int main(int argc, char const *argv[])
{
    test01();
    return 0;
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 02_code.c
edu@edu:~/work/c/day07$ ./a.out
100 hello world
e
段错误 (核心已转储)
edu@edu:~/work/c/day07$
```

2、结构体的指针成员 指向堆区

```c
struct stu
{
    int num;
    char *name; //指针成员
};
```

```c
if(lucy.name != NULL)
{
    free(lucy.name);
    lucy.name= NULL;
}
```




```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct stu
{
    int num;
    char *name; //指针成员
};
void test01()
{
    struct stu lucy;
```

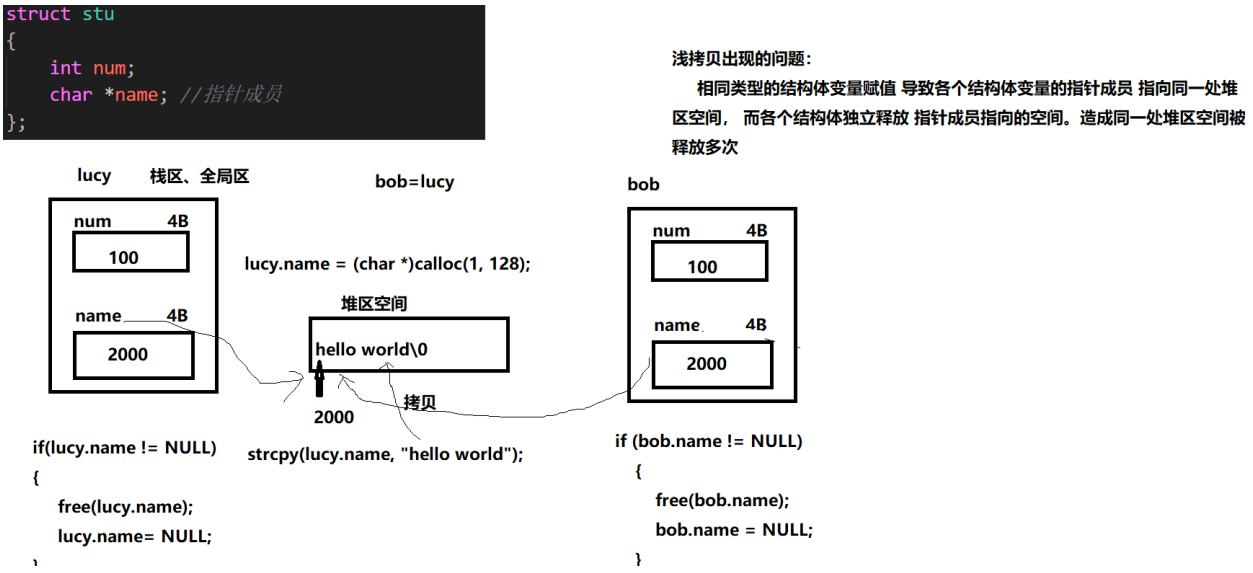
```c
    lucy.num = 100;
    //让name指向堆区
    lucy.name = (char *)calloc(1, 128);
    //此处造成内存泄漏 name先指向堆区 然后name又指向文字常量区 丢失堆区空间地址
    //lucy.name = "hello world";
    strcpy(lucy.name, "hello world");
    printf("%d %s\n", lucy.num, lucy.name);

    printf("%c\n", lucy.name[1]);
    lucy.name[1] = 'E';
    printf("%d %s\n", lucy.num, lucy.name);

    //释放lucy.name指向的堆区空间
    if (lucy.name != NULL)
    {
        free(lucy.name);
        lucy.name = NULL;
    }
}
int main(int argc, char const *argv[])
{
    test01();
    return 0;
}
```

```
edu@edu:~/work/c/day07$ sudo gcc 02_code.c
edu@edu:~/work/c/day07$ ./a.out
100 hello world
e
100 hEllo world
edu@edu:~/work/c/day07$
```

3、相同类型的结构体变量可以整体赋值

如果结构体中没有指针成员 赋值 不会出现浅拷贝。

如果结构体中有指针成员 赋值 容易造成浅拷贝




```c
#include <string.h>
#include <stdlib.h>
struct stu
{
    int num;
    char *name; //指针成员
};
void test01()
{
    struct stu lucy;
    lucy.num = 100;
    //让name指向堆区
    lucy.name = (char *)calloc(1, 128);
    strcpy(lucy.name, "hello world");
    printf("%d %s\n", lucy.num, lucy.name);

    struct stu bob;
    bob = lucy;//浅拷贝

    //释放lucy.name指向的堆区空间
    if (lucy.name != NULL)
    {
        free(lucy.name);
        lucy.name = NULL;
    }

    if (bob.name != NULL)
    {
```

```
29          free(bob.name);
30          bob.name = NULL;
31      }
32  }
```

```
edu@edu:~/work/c/day07$ sudo gcc 02_code.c
edu@edu:~/work/c/day07$ ./a.out
100 hello world
*** Error in `./a.out': double free or corruption (!prev): 0x000000
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x77725)[0x7fe91a776725]
/lib/x86_64-linux-gnu/libc.so.6(+0x7ff4a)[0x7fe91a77ef4a]
/lib/x86_64-linux-gnu/libc.so.6(cfree+0x4c)[0x7fe91a782abc]
./a.out[0x40064a]
./a.out[0x40066e]
/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xf0)[0x7fe91a71f
```

出现多重释放

如果结构体中有指针成员 尽量使用深拷贝。

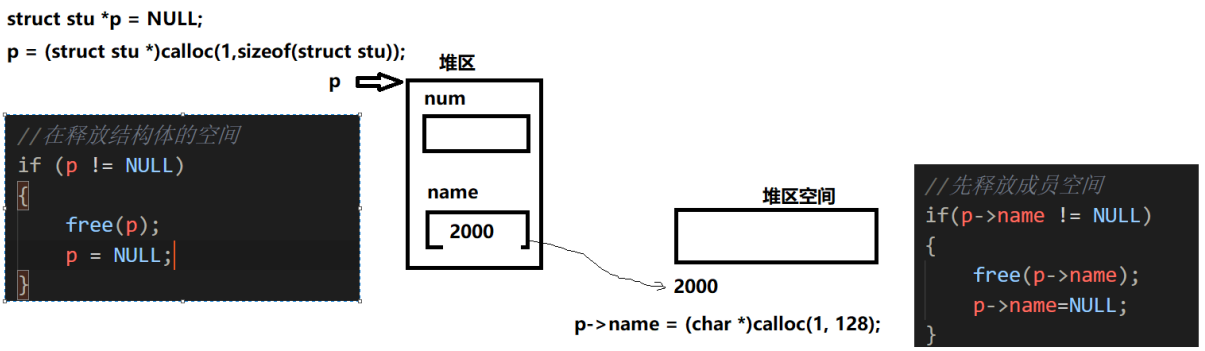
```
1  struct stu bob;
2  bob.name = (char *)calloc(1, 128);
3  bob.num = lucy.num;
4  strcpy(bob.name, lucy.name);
```

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  struct stu
5  {
6      int num;
7      char *name; //指针成员
8  };
9  void test01()
10 {
11     struct stu lucy;
12     lucy.num = 100;
13     //让name指向堆区
14     lucy.name = (char *)calloc(1, 128);
15     strcpy(lucy.name, "hello world");
16     printf("%d %s\n", lucy.num, lucy.name);
17
18     struct stu bob;
19     bob.name = (char *)calloc(1, 128);
20     bob.num = lucy.num;
```

```c
    strcpy(bob.name, lucy.name);

    printf("%d %s\n", bob.num, bob.name);
    //释放lucy.name指向的堆区空间
    if (lucy.name != NULL)
    {
        free(lucy.name);
        lucy.name = NULL;
    }

    if (bob.name != NULL)
    {
        free(bob.name);
        bob.name = NULL;
    }
}
int main(int argc, char const *argv[])
{
    test01();
    return 0;
}
```

```
edu@edu:~/work/c/day07$ sudo gcc 02_code.c
edu@edu:~/work/c/day07$ ./a.out
100 hello world
100 hello world
edu@edu:~/work/c/day07$
```

4、结构体在堆区 结构体的指针成员指向堆区。（了解）

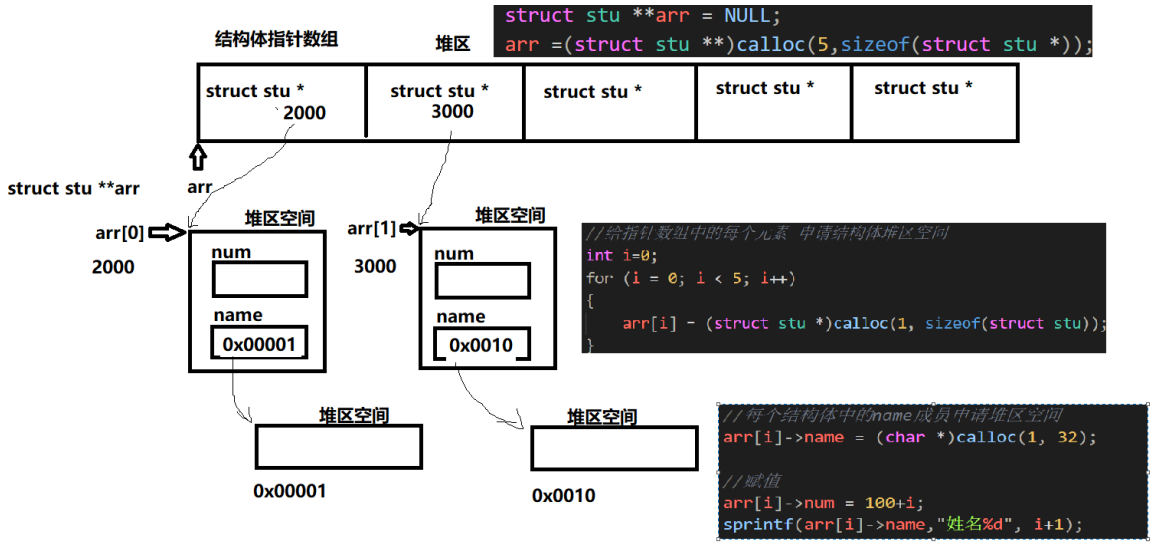



```c
void test02()
{
```

```c
    //结构体本身在堆区
    struct stu *p = NULL;
    p = (struct stu *)calloc(1, sizeof(struct stu));

    //为结构体中的指针成员 申请堆区空间
    p->name = (char *)calloc(1, 128);

     //给结构体中的成员赋值
     p->num = 100;
     strcpy(p->name, "hello world");

     printf("%d %s\n", p->num, p->name);

     //先释放成员空间
     if (p->name != NULL)
     {
         free(p->name);
         p->name = NULL;
     }

     //在释放结构体的空间
     if (p != NULL)
     {
         free(p);
         p = NULL;
     }
}
```

```
edu@edu:~/work/c/day07$ sudo gcc 02_code.c
[sudo] edu 的密码：
edu@edu:~/work/c/day07$ ./a.out
100 hello world
edu@edu:~/work/c/day07$
```

5、结构体指针数组在堆区、结构体在堆区、结构指针成员在堆区（了解）




```c
void test03()
{
    //给结构体指针数组申请堆区空间
    struct stu **arr = NULL;
    arr = (struct stu **)calloc(5, sizeof(struct stu *));

    //给指针数组中的每个元素 申请结构体堆区空间
    int i = 0;
    for (i = 0; i < 5; i++)
    {
        arr[i] = (struct stu *)calloc(1, sizeof(struct stu));
        //每个结构体中的name成员申请堆区空间
        arr[i]->name = (char *)calloc(1, 32);

        //赋值
        arr[i]->num = 100 + i;
        sprintf(arr[i]->name, "姓名%d", i + 1);
    }

    //遍历
    for (i = 0; i < 5; i++)
    {
        printf("%d %s\n", arr[i]->num, arr[i]->name);
    }

    //释放arr中的每个元素 以及每个元素中的name
    for (i = 0; i < 5; i++)
    {
        //先释放结构体中的name的指向
```

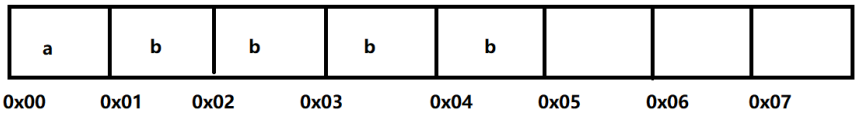
```
30          if (arr[i]->name != NULL)
31          {
32              free(arr[i]->name);
33              arr[i]->name = NULL;
34          }
35
36          //释放结构体
37          if (arr[i] != NULL)
38          {
39              free(arr[i]);
40              arr[i] = NULL;
41          }
42      }
43      //释放整个arr指针数组
44      if (arr != NULL)
45      {
46          free(arr);
47          arr = NULL;
48      }
49
50      return;
51  }
```

```
edu@edu:~/work/c/day07$ sudo gcc 02_code.c
[sudo] edu 的密码:
edu@edu:~/work/c/day07$ ./a.out
100 姓名1
101 姓名2
102 姓名3
103 姓名4
104 姓名5
edu@edu:~/work/c/day07$
```

知识点7【结构体的对齐规则】（了解）

1、自动对齐规则

不对齐



访问a:只需要一个周期 0x00~0x03 只要0x00

方向b:需要两个周期

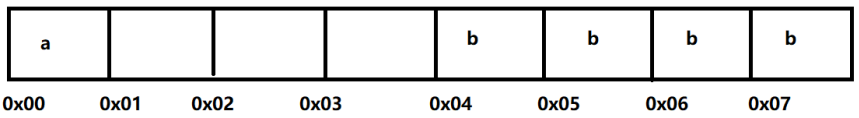
第一个周期:0x00~0x03 只要0x01~0x03

第二个周期:0x04~0x07 只要0x04 最后将0x01~0x03 和0x04拼接

```
struct Data
{
    char a;
    int b;
};
```

CPU一次提取4字节

对齐 用空间换时间



访问a:只需要一个周期 0x00~0x03 只要0x00

访问b:只需要一个周期0x04~0x07

1、确定分配单位（一行分配多少字节）

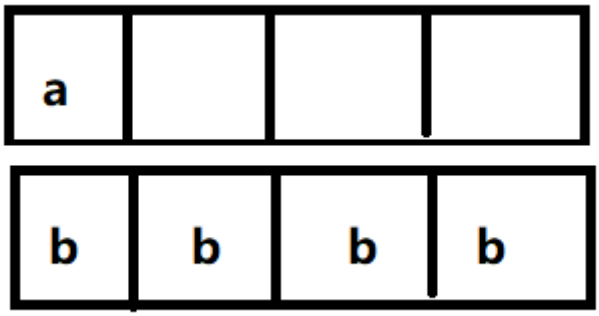
结构体中最大的基本类型 长度决定。

2、确定成员的偏移量

成员偏移量 = 成员自身类型的整数倍。

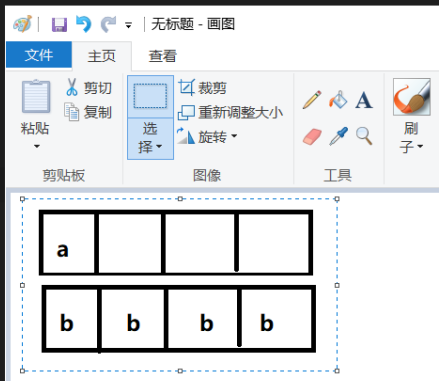
3、收尾工作

结构体的总大小 = 分配单位整数倍



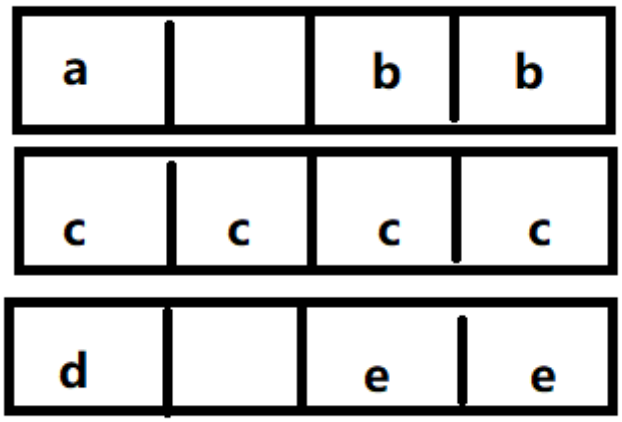

```
struct Data
{
    char a;
    int b;
};

void test01()
{
    printf("sizeof(Data)=%lu\n", sizeof(struct Data));
    struct Data ob;
    printf("%lu\n", &ob.a);
    printf("%lu\n", &ob.b);
}
int
{
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ ./a.out
sizeof(Data)=8
140726437122768
140726437122772
edu@edu:~/work/c/day07$
```



案例1：画出以下结构体的对齐

```
struct Data
{
 char a;
 short b;
 int c;
 char d;
 short e;
};
```



2、结构体嵌套结构体 自动对齐规则

1、确定分配单位（一行分配多少字节）

所有结构体中最大的基本类型 长度决定。

2、确定成员的偏移量

普通成员偏移量 = 成员自身类型的整数倍。

被嵌套的结构体整体偏移量 = 该结构体中最大的基本类型整数倍

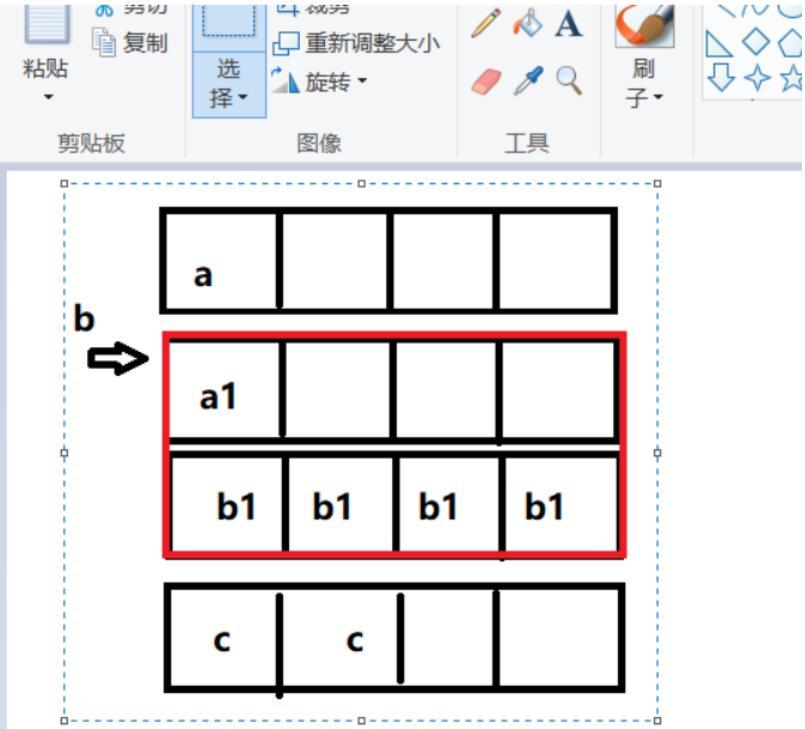
3、收尾工作

结构体成员大小 = 该结构体中最大的基本类型整数倍

结构体的总大小 = 分配单位整数倍

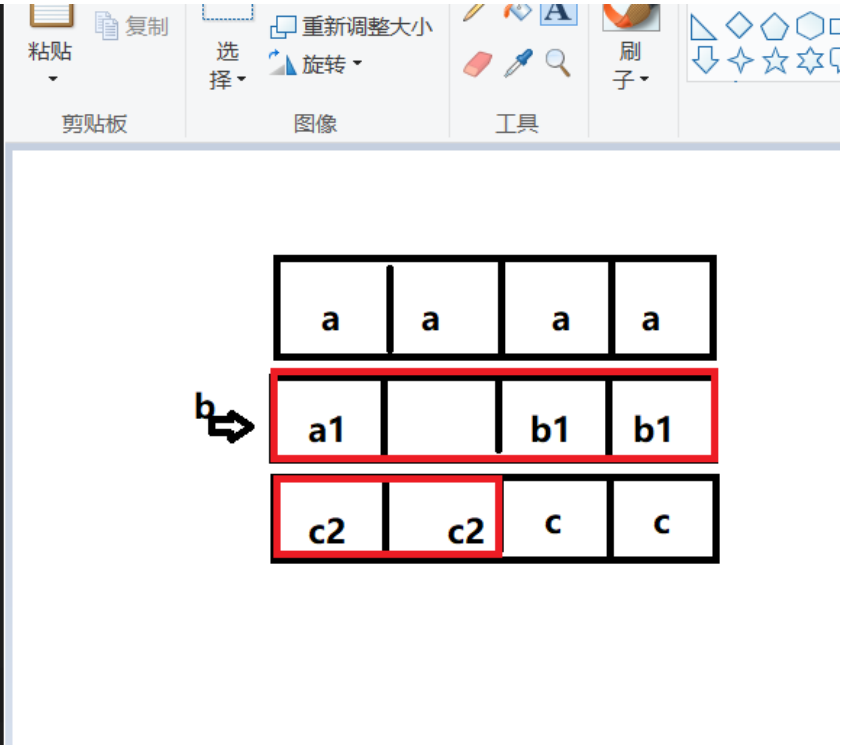
案例1：

```
struct A
{
    char a1;
    [redacted]
    int b1;
};

struct B
{
    char a;
    struct A b;
    short c;
};
```



案例2：

```
struct A
{
    char a1;
    short b1;
    short c2;
};

struct B
{
    int a;
    struct A b;
    short c;
};
```



3、强制对齐

#pragma pack (value)时的指定对齐值value。

注意value值为1 2 4 8 16

1、确定分配单位（一行分配多少字节）

分配单位 = min(结构体中最大的基本类型, value)

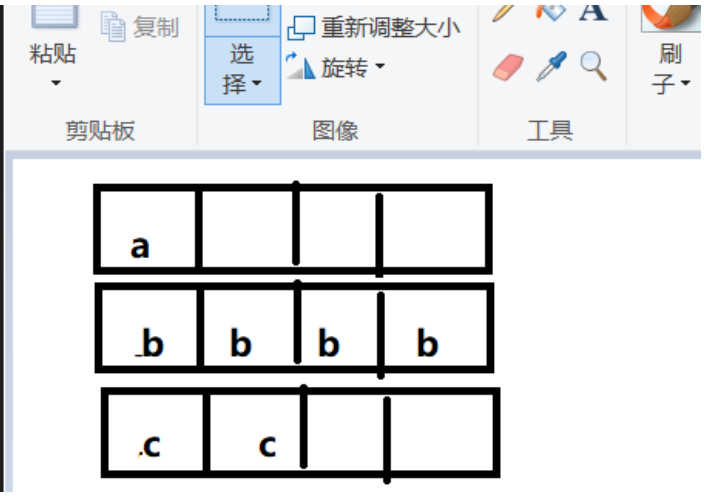
2、确定成员的偏移量

成员偏移量 = 成员自身类型的整数倍。

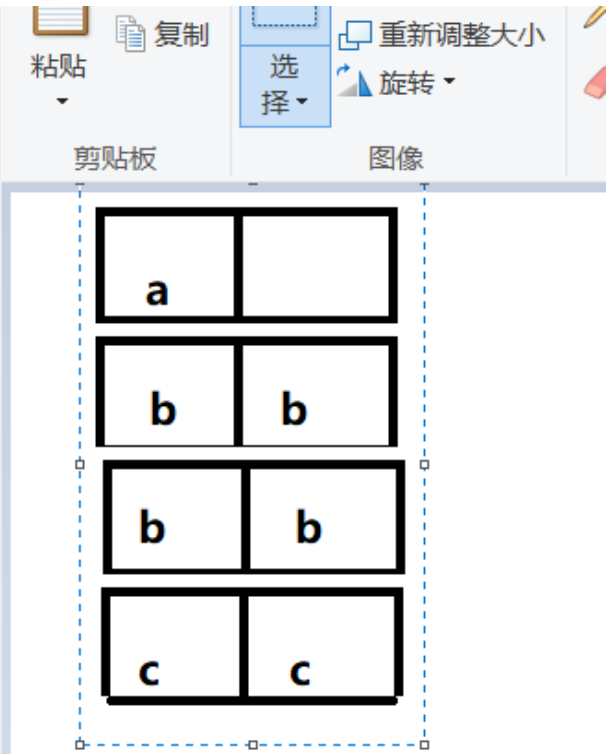
3、收尾工作

结构体的总大小 = 分配单位整数倍

```c
#include <stdio.h>
#pragma pack(8)
struct A
{
    char a;
    int b;
    short c;
};
```




```c
#include <stdio.h>
#pragma pack(2)
struct A
{
    char a;
    int b;
    short c;
};
void test01()
{
    printf("sizeof(Dat
}
```



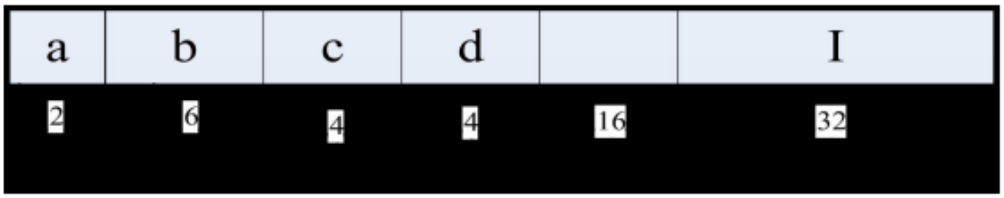
```c
#include <stdio.h>
struct A
{
    char a;
    int b;
    short c;
};
struct B
{
    char a;
    short c;
    int b;
};
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 03_code.c
edu@edu:~/work/c/day07$ ./a.out
sizeof(A)=12
sizeof(B)=8
edu@edu:~/work/c/day07$
```

知识点8【结构体的位域】（重要）

在结构体中，以位为单位的成员，咱们称之为位段(位域)

```c
struct packed_data{
    unsigned int a:2;
    unsigned int b:6;
    unsigned int c:4;
    unsigned int d:4;
    unsigned int i;
} data;
```



a的类型是unsigned int a的大小 只占2位二进制位。

没有非位域隔开的位域 叫相邻位域。

相邻位域可以压缩。但是压缩的位数 不能超过自身类型的大小。

```c
#include <stdio.h>
struct A
{
    unsigned char a : 2;
    unsigned char b : 2;
    unsigned char c : 4;
};
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 03_code.c
edu@edu:~/work/c/day07$ ./a.out
sizeof(A)=1
edu@edu:~/work/c/day07$
```

不要对位域取地址。

```c
struct A
{
    unsigned char a : 2;
    unsigned char b : 2;
    unsigned char c : 4;
};

void test01()
{
    //printf("sizeof(A)=%lu\n", sizeof(struct A));
    struct A ob;
    &ob.a;//error
}
```

对位域赋值 不要超过 位域本身位的宽度

```c
struct A
{
    unsigned char a : 2;
    unsigned char b : 2;
    unsigned char c : 4;
};

void test01()
{
    //printf("sizeof(A)=%lu\n", sizeof(struct A));
    struct A ob;
    ob.a = 5;//0101
    printf("ob.a = %d\n", ob.a); //1
}
```

另起一个存储单元

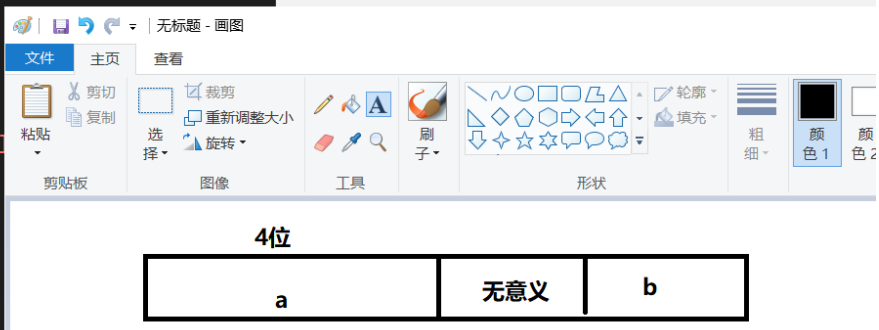
```c
#include <stdio.h>
struct A
{
    unsigned char a : 4;
    unsigned char : 0; //另起一个存储单元
    unsigned char b : 4;
};
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 03_code.c
edu@edu:~/work/c/day07$ ./a.out
sizeof(A)=2
edu@edu:~/work/c/day07$
```

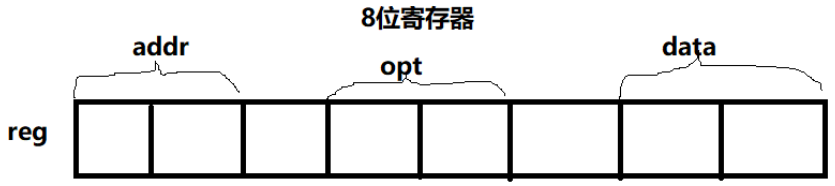
无意义位段：

```c
#include <stdio.h>
struct A
{
    unsigned char a : 4;
    unsigned char : 2; //2位无意义的位段
    unsigned char b : 2;
};

void test01()
{
    printf("sizeof(A)=%
}
int main(int argc, char
{
    test01();
    return 0;
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 03_code.c
edu@edu:~/work/c/day07$ ./a.out
sizeof(A)=1
edu@edu:~/work/c/day07$
```



案例1：



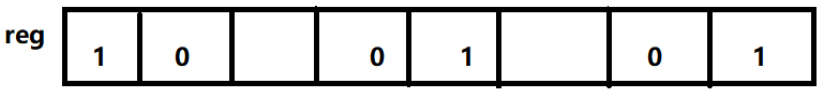
00:设备1
01:设备2
10:设备3
11:设备4

00:建立连接
01:发送
10:接收
11:关闭连接

00：关
01：开

```c
struct REG
{
    unsigned char data:2;
    unsigned char :1;
    unsigned char opt:2;
    unsigned char :1;
    unsigned char addr:2;
};
```

需求：请给设备3 发送 开灯指令




```c
reg = reg | (0x01<<0|0x01<<3|0x01<<7)  &  ~(0x01<<1|0x01<<4|0x01<<6);
REG  reg;
reg.addr = 2;
reg.opt = 1;
reg.data = 1;
```

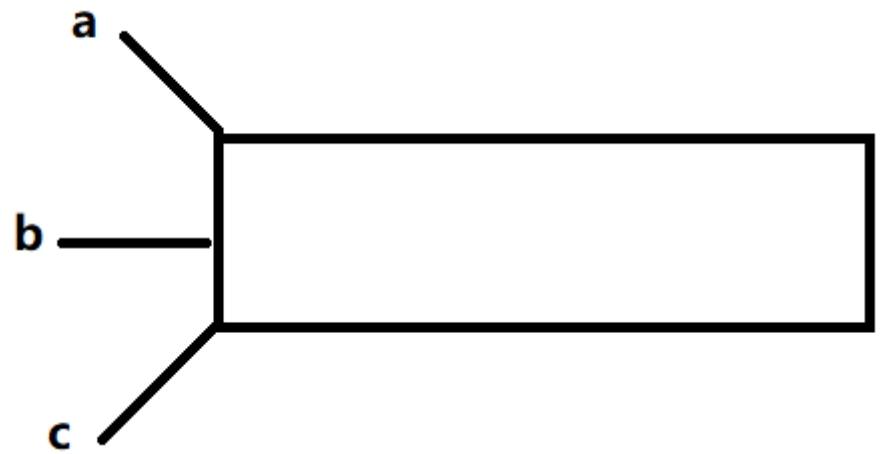
知识点9【共用体union】（了解）

结构体：所有成员拥有独立空间

共用体：所有成员共享同一块空间

```
union Data
{
    char a;
    short b;
    int c;
};
```

成员a b c共享同一块空间。空间大小 由最大的成员空间决定



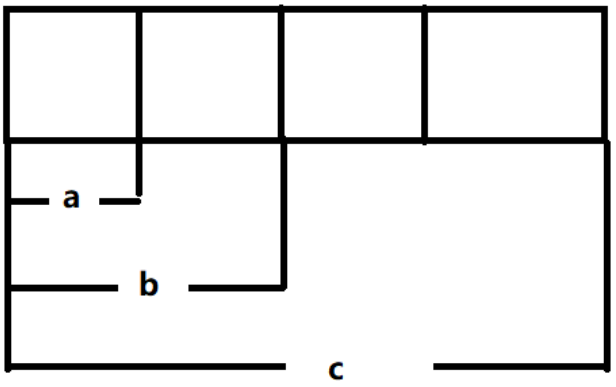
案例1：

```
#include <stdio.h>
union Data
{
    char a;
    short b;
    int c;
};

void test01()
{
    union Data ob;
    ob.a = 10;
    ob.b = 20;
    ob.c = 30;
    printf("%d\n", ob.a + ob.b + ob.c); //90
}
```

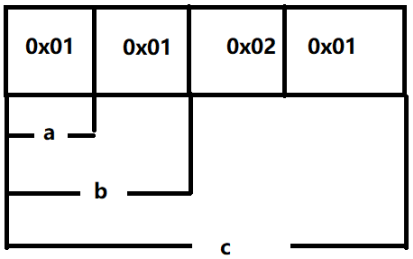
```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc 03_code.c
[sudo] edu 的密码：
edu@edu:~/work/c/day07$ ./a.out
90
edu@edu:~/work/c/day07$
```

成员a b c共享同一块空间，但是每个成员 能操作的空间的范围 是由成员自身类型长度决定

```
union Data
{
    char a;
    short b;
    int c;
};
```



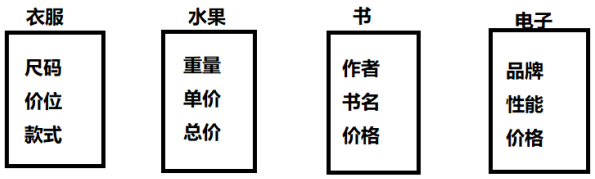

```
union Data
{
    char a;
    short b;
    int c;
};
```



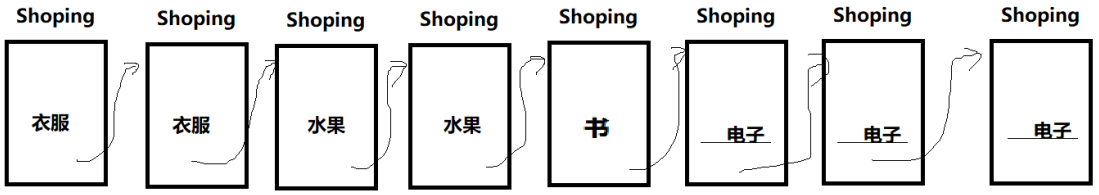

```c
void test01()
{
    union Data ob;
    ob.c = 0x01020304;
    ob.b = 0x0102;
    ob.a = 0x01;
    printf("%#x\n", ob.a + ob.b + ob.c);
}
```

```
0b.c: 0x01020101
ob.b:0x00000101
0b.a:0x00000001
---------------------------
0x01020203
```

应用：




```
union  Shoping
{
        衣服 衣;
        水果 水:
        书    book;
       电子   dian;
};
```



知识点10【枚举enum】（了解）

枚举:将枚举变量 要赋的值 一一列举出来

```
1  enum POKER_COLOR{HONGTAO,MEIHUA,FANGKUAI,HEITAO};
```

```c
enum POKER_COLOR{HONGTAO,MEIHUA,FANGKUAI,HEITAO};

void test01()
{
    enum POKER_COLOR p_c = MEIHUA;
    //HONGTAO,MEIHUA,FANGKUAI,HEITAO 枚举列表的值默认从0开始递增
    printf("%d %d %d %d\n", HONGTAO,MEIHUA,FANGKUAI,HEITAO);
}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gc
[sudo] edu 的密码：
edu@edu:~/work/c/day07$ ./a.out
0 1 2 3
edu@edu:~/work/c/day07$
```

如果修改某个枚举列表的值

```c
#include <stdio.h>
enum POKER_COLOR{HONGTAO,MEIHUA=10,FANGKUAI,HEITAO};

void test01()
{
    enum POKER_COLOR p_c = MEIHUA;
    //HONGTAO,MEIHUA,FANGKUAI,HEITAO 枚举列表的值默认从0开始递增
    printf("%d %d %d %d\n", HONGTAO,MEIHUA,FANGKUAI,HEITAO);

}
```

```
edu@edu: ~/work/c/day07
edu@edu:~/work/c/day07$ sudo gcc
edu@edu:~/work/c/day07$ ./a.out
0 10 11 12
edu@edu:~/work/c/day07$
```