知识点1【动态内存申请的API】(重要)

1、malloc申请 堆区空间

```
#include <stdlib.h>
 void *malloc(size_t size);
 size表示申请的空间字节数
 函数的返回值：
 成功：返回值空间起始地址
 失败：NULL
```

特点：malloc申请的堆区空间 不自动清0

案例1：申请int空间

```c
#include <stdio.h>
#include <stdlib.h>
void test01()
{
    //空間申請
    int *p = NULL;
    p = (int *)malloc(sizeof(int));
    if (p == NULL)
    {
        return;
    }

    *p = 100;

    printf("*p = %d\n", *p);

    //空间释放
    free(p);
}
```

2、free释放堆区空间

```c
#include <stdlib.h>
void free(void *ptr);
ptr需要释放的堆区空间的起始地址
```

3、memset内存设置函数

```c
#include <string.h>
void *memset(void *s, int c, size_t n);
s就是空间的起始地址
n就是空间的字节宽度
c空间中每个字节 填充的值
```

案例2：动态数组

```c
#include <string.h>
void test02()
{
    int n = 0;
    printf("请输入int元素的个数:");
    scanf("%d", &n);

    //根据元素的个数申请空间
    int *p = NULL;
     p = (int *)malloc(n * sizeof(int));
```

```c
    if (p == NULL)
    {
        return;
    }
    //将堆区空间清0
    memset(p, 0, n * sizeof(int));

    //获取键盘输入
    int i = 0;
    for (i = 0; i < n; i++)
    {
        scanf("%d", p + i);
    }

    //遍历数组元素
    for (i = 0; i < n; i++)
    {
        printf("%d ", p[i]); //*(p+i)
    }
    printf("\n");

    //释放空间
    free(p);

    return;
}
```

4、calloc 函数

```c
#include <stdlib.h>
```

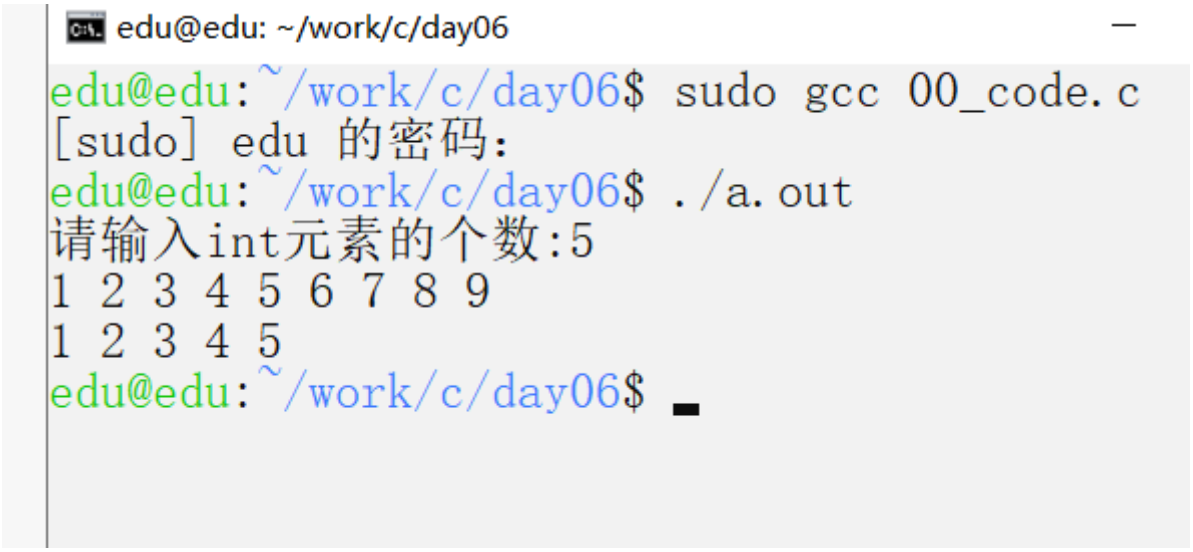
```
void *calloc(size_t nmemb, size_t size);
nmemb：内存的块数
size：每一块的字节数
返回值：成功为堆区空间起始地址 失败为NULL
calloc会对申请的空间 自动清0
```

```
malloc(n*sizeof(int));
calloc(n, sizeof(int));
```

案例1、简易 的动态数组

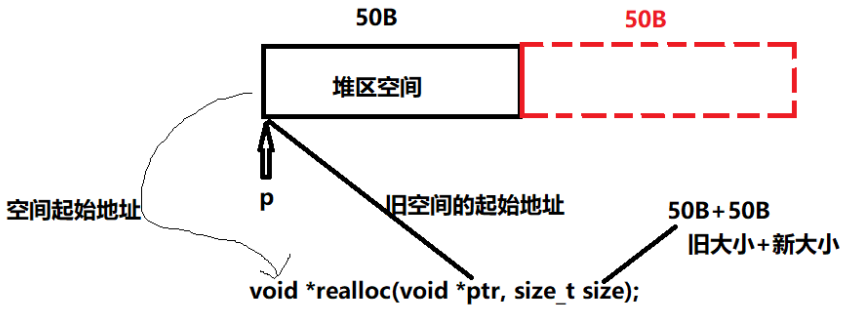
```
void *get_addr(int n, int elem_size)
{
    return calloc(n, elem_size);
}
void input_int_array(int *p, int n)
{
    int i = 0;
    for (i = 0; i < n; i++)
    {
        scanf("%d", p + i);
    }
}
void print_int_array(int *p, int n)
{
    int i = 0;
    for (i = 0; i < n; i++)
    {
        printf("%d ", p[i]);
    }
    printf("\n");
}
void test03()
{
    int n = 0;
    printf("请输入int元素的个数:");
    scanf("%d", &n);

    //根据元素个数 申请空间
    int *p = NULL;
    p = (int *)get_addr(n, sizeof(int));
    if (p == NULL)
```

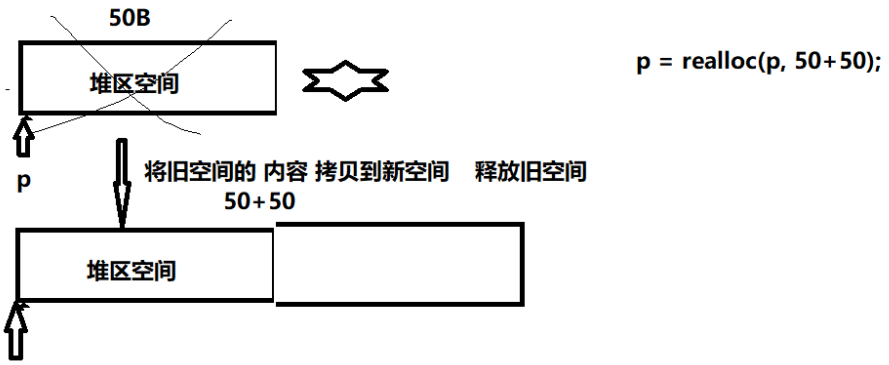
```
32      {
33          return;
34      }
35
36      //获取键盘输入
37      input_int_array(p, n);
38
39      //遍历
40      print_int_array(p, n);
41
42      //释放空间
43      free(p);
44  }
```



5、realloc函数 追加空间

```
1  #include <stdlib.h>
2  void *realloc(void *ptr, size_t size);
```

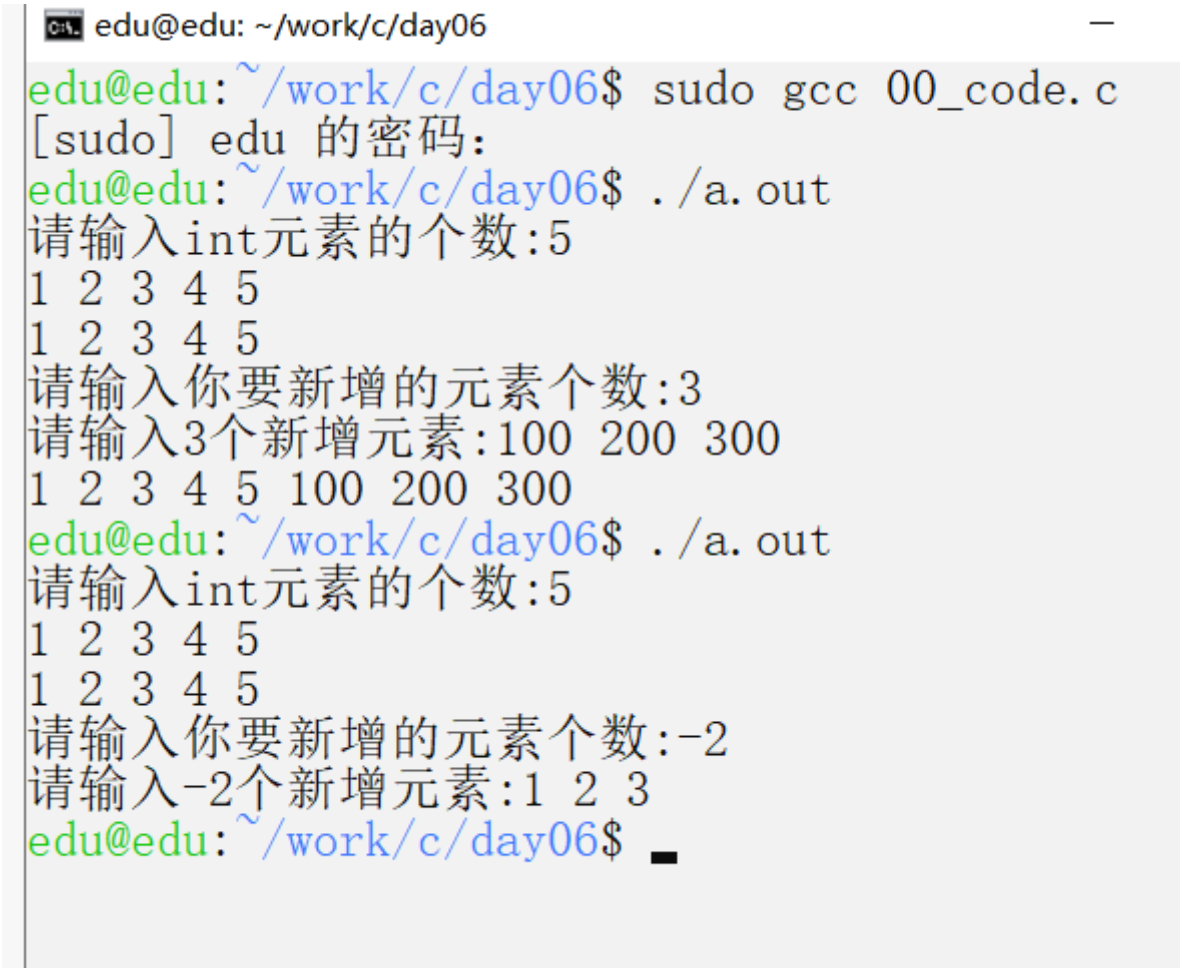




必须使用指针变量 获取realloc的返回值.

```c
void test04()
{
    int n = 0;
    printf("请输入int元素的个数:");
    scanf("%d", &n);

    //根据元素的个数申请空间
    int *p = NULL;
    p = (int *)calloc(n, sizeof(int));
     if (p == NULL)
     {
         return;
     }

     //获取键盘输入
     int i = 0;
     for (i = 0; i < n; i++)
     {
         scanf("%d", p + i);
     }

     //遍历数组元素
     for (i = 0; i < n; i++)
     {
```

```c
        printf("%d ", p[i]); //*(p+i)
    }
    printf("\n");

    printf("请输入你要新增的元素个数:");
    int new_n = 0;
    scanf("%d", &new_n);

    //追加空间
    p = (int *)realloc(p, (n + new_n) * sizeof(int));

    printf("请输入%d个新增元素:", new_n);
    //请输入新增的元素
    for (i = n; i < n + new_n; i++)
    {
        scanf("%d", p + i);
    }

    //遍历数组元素
    for (i = 0; i < n + new_n; i++)
    {
        printf("%d ", p[i]); //*(p+i)
    }
    printf("\n");

    free(p);
}
```



知识点2【内存泄漏】(了解)

申请的内存，首地址丢了，找不了，再也没法使用了，也没法释放了，这块内存就被泄露了。

案例1：

```
char *p;
p=(char *)malloc(100);
//接下来，可以用 p 指向的内存了
p="hello world";//p 指向别的地方了
//从此以后，再也找不到你申请的 100 个字节了。则动态申请的 100 个字节就被泄露了
```

案例2：

```
void fun()
{
    char *p;
    p=(char *)malloc(100);
    //接下来，可以用 p 指向的内存了
    ;
    ;
}

int main()
{
    fun();
    fun();
    return 0;
}
```

知识点3【内存的回顾】(了解)

```
1  //在32位平台
2  void test05()
3  {
4      //字符数组 在栈区 开辟12字节 存放"hello world"
5      char str1[] = "hello world";
6
7      //str2字符指针变量 在栈区 4B 保存文字常量区"hello world"的首元素地址
8      char *str2 = "hello world";
9
10      //str3字符指针变量 在栈区 4B 保存堆区128B空间的起始地址
```

```
    char *str3 = (char *)calloc(128);
    //将字符串"hello world"拷贝到str3指向的堆区空间
    strcpy(str3, "hello world");

}
```

```
//在32位平台
char a;
int b;
//void fun(char str4[128])
void fun(char *str4)
{
    printf("%lu\n", sizeof(str4)); //4
}
void test06()
{
    char str1[] = "hello";
    char *str2 = "hello";
    char *str3 = (char *)malloc(128);
    printf("%lu\n", sizeof(a));    //1
    printf("%lu\n", sizeof(b));    //4
    printf("%lu\n", sizeof(str1)); //6
    printf("%lu\n", sizeof(str2)); //4
    printf("%lu\n", sizeof(str3)); //4
}
```

知识点4【标准的空间释放】(了解)

```
void test07()
{
    int *p = NULL;

    p = (int *)calloc(1, sizeof(int));

    *p = 100;
    printf("*p = %d\n", *p);

    if (p != NULL)
    {
```

```
12          free(p);
13          p = NULL;
14      }
15
16      if (p != NULL)
17      {
18          free(p);
19          p = NULL;
20      }
21
22      return;
23  }
```

释放free本质功能：回收空间权限