知识点 1【堆区空间概述】

需要使用 malloc calloc realloc 从堆区申请空间，必须使用指针变量 取保存得到的地址编号。使用完后 记得调用 free 释放堆区空间。

堆区空间 是在程序运行过程中 根据需要 申请。所以又叫动态空间。

申请空间、使用空间、释放空间

1、malloc 函数 申请堆区空间

```
#include <stdlib.h>

void *malloc(size_t size);

参数 size：表示申请空间的大小（单位是字节）

返回值：

    成功：返回堆区空间的起始地址

    失败：返回 NULL

注意：

    1、记得对 malloc 的返回值 强转成你需要的地址类型

    2、在 ubuntu16.04 中使用 gcc 编译器 下 malloc 会对申请到的堆区空间自动清零
```

2、free 释放堆区空间函数

```
void free(void *ptr);
```

2.1 案例申请一个 int 空间

```c
#include <stdio.h>
#include <stdlib.h>
void test01()
{
    // 1、申请一个int空间
    int *p1 = (int *)malloc(sizeof(int));
    if (NULL == p1)
    {
        printf("malloc error");
        return;
    }

    // 2、使用空间
    *p1 = 1000;
    printf("%d\n", *p1);

    // 3、释放空间
    free(p1);
}
```

p1 在栈区 占 8 字节 保存的是堆区空间地址编号 申请的 4 字节在堆区。

2.2 使用 malloc 申请一个数组空间

```c
void test02()

{

    int n = 0;

    printf("请输入 int 数据的个数:");

    scanf("%d", &n);



    // 申请 int 数组空间

    int *arr = (int *)malloc(n * sizeof(int));

    if (NULL == arr)
```

```
    {
        printf("malloc error");
        return;
    }

    // 使用数组
    int i = 0;
    for (i = 0; i < n; i++)
    {
        // printf("%d ", *(arr+i));
        printf("%d ", arr[i]);
    }
    printf("\n");

    // 释放空间
    free(arr);
}
```

3、calloc 函数申请堆区空间

```
void *calloc(size_t nmemb, size_t size);

参数 nmemb：内存的块数

参数 size：每一块的大小

申请的总大小=nmemb*size
```

返回值：

成功：返回堆区空间的起始地址

失败：返回 NULL

在任何平台下 calloc 都会对申请的堆区空间清零

```
void test03()
{
    int n=0;
    printf("请输入 int 数据的个数:");
    scanf("%d", &n);

    //申请 int 数组空间
    //int *arr = (int *)malloc(n*sizeof(int));
    int *arr=(int *)calloc(n,sizeof(int));
    if(NULL == arr)
    {
        printf("calloc error");
        return;
    }

    //使用数组
    int i=0;
    for ( i = 0; i < n; i++)
```

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

    //释放空间
    free(arr);
}
```

4、realloc 函数追加空间

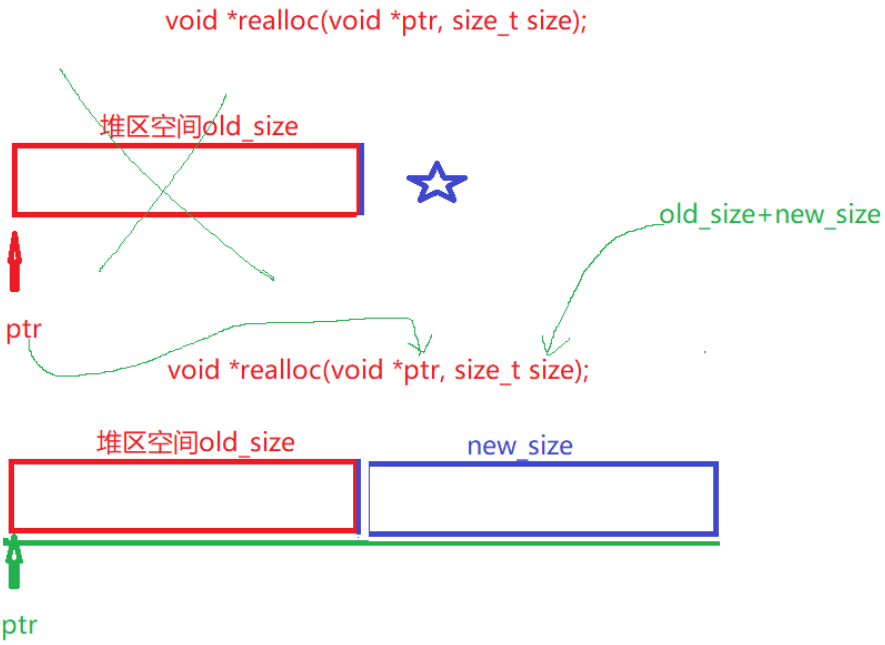
realloc 只能对 malloc 或 calloc 申请的堆区空间 进行追加。

```
void *realloc(void *ptr, size_t size);

参数 ptr:是旧空间的其实地址

参数 size:旧空间大小+新空间大小

必须使用指针变量 保存 realloc 的返回值
```




```
void test04()

{

    // 申请旧空间大小

    int *arr = (int *)calloc(5, sizeof(int));

    if (NULL == arr)

    {

        printf("calloc error");

        return;

    }



    // 追加新空间

    arr = (int *)realloc(arr, (5 + 5) * sizeof(int));

    if (NULL == arr)
```

```
    {
        printf("calloc error");
        return;
    }

    int i = 0;
    for (i = 0; i < 10; i++)
    {
        // printf("%d ", *(arr+i));
        printf("%d ", arr[i]);
    }
    printf("\n");

    // 释放空间
    free(arr);
}
```

5、内存设置函数 memset

```
#include <string.h>

void *memset(void *s, int c, size_t n);

功能：将指定的内存的每一个字节 设置成特定的值

参数 s:指定内存的起始地址

参数 c:每个字节特定值
```

参数 n:从 s 开始 操作 n 个字节

注意：主要用于将指定的内存清 0

6、内存拷贝函数 memcpy

void *memcpy(void *dest, const void *src, size_t n);

功能：将 src 指向的空间 n 个字节 拷贝到 dest 指向的空间中

参数 dest:目的空间的起始地址

参数 src:源空间的起始地址

参数 n：操作空间的额字节数

```c
#include <string.h>
void test05()
{
    int arr1[5] = {10, 20, 30, 40, 50};
    int arr2[5];

    // 内存拷贝
    memcpy(arr2, arr1, sizeof(arr1));

    int i = 0;
    for (i = 0; i < 5; i++)
    {
        printf("%d ", arr2[i]);
    }
    printf("\n");
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
edu@edu:~/work/c/day06$ gcc 00_code.c
edu@edu:~/work/c/day06$ ./a.out
10 20 30 40 50
edu@edu:~/work/c/day06$
```

知识点 2【静态数组和动态数组】

1、静态数组和动态数组的区别（背）

静态数组：类似 int arr[5];在编译阶段就已经确定了数组的大小。数据过多溢出 数据过少浪费空间。

动态数组：从堆区动态申请空间，在运行的时候 根据用户的输入决定空间的大小。

2、创建一个动态数组案例

```
#include <stdio.h>

#include <stdlib.h>

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

{

    int old_n = 0;

    printf("请输入旧空间的元素个数:");

    scanf("%d", &old_n);



    // 根据 old_n 申请堆区空间

    int *arr = (int *)calloc(old_n, sizeof(int));

    if (NULL == arr)

    {

        printf("calloc error\n");

        return 0;

    }
```

```
// 获取键盘输入
printf("请输入%d 个 int 数值:", old_n);
int i = 0;
for (i = 0; i < old_n; i++)
{
    scanf("%d", arr + i);
}

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

// 为新增的元素个数追加空间
arr = (int *)realloc(arr, (old_n + new_n) * sizeof(int));
if (NULL == arr)
{
    printf("realloc error\n");
    return 0;
}

// 为新增的空间输入元素
printf("请输入%d 个新增的 int 数据:", new_n);
```

```
    for (i = old_n; i < (old_n + new_n); i++)
    {
        scanf("%d", arr + i);
    }

    // 遍历整个数组
    for (i = 0; i < old_n + new_n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // 释放空间
    free(arr);

    return 0;
}
```

知识点 3【free 释放的问题】

1、不要操作 已经释放的空间

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char const *argv[])
{
    int *p = (int *)malloc(sizeof(int));
    *p = 1000;
    free(p);//回收空间权限 内容是否清零不确定
    printf("%d\n", *p); // 反问非法空间
    return 0;
}

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

edu@edu:~/work/c/day06$ gcc 02_code.c
edu@edu:~/work/c/day06$ ./a.out
0
edu@edu:~/work/c/day06$
```

2、防止多重释放

```c
#include <stdlib.h>
int main(int argc, char const *argv[])
{
    int *p = (int *)malloc(sizeof(int));
    *p = 1000;
    if (p != NULL)
    {
        free(p);//free释放的是p保存的地址编号对应的空间 不是p变量本身的空间
        p = NULL;
    }

    if (p != NULL)
    {
        free(p);
        p = NULL;
    }

    return 0;
}
```