动态存储分配申请

在数组一章中，曾介绍过数组的长度是预先定义好的，在整个程序中固定不变。C语言中不允许动态数组类型。例如：

```
1. int n;
2. scanf("%d",&n);
3. int a[n];
```

用变量表示长度，想对数组的大小作动态说明，这是错误的。但是在实际的编程中，往往会发生这种情况，即所需的内存空间取决于实际输入的数据，而无法预先确定。对于这种问题，用数组的办法很难解决。为了解决上述问题，C语言提供了一些内存管理函数，这些内存管理函数可以按需要动态地分配内存空间，也可把不再使用的空间回收待用，为有效地利用内存资源提供了手段。

常用的内存管理函数有以下三个。

1) 分配内存空间函数malloc

调用形式：
(类型说明符*)malloc(size)

功能：在内存的动态存储区中分配一块长度为"size"字节的连续区域。函数的返回值为该区域的首地址。

- “类型说明符”表示把该区域用于何种数据类型。
- (类型说明符*)表示把返回值强制转换为该类型指针。
- “size”是一个无符号数。

例如：
pc=(char *)malloc(100);
表示分配100个字节的内存空间，并强制转换为字符数组类型，函数的返回值为指向该字符数组的指针，把该指针赋予指针变量pc。

2) 分配内存空间函数 calloc

calloc 也用于分配内存空间。调用形式：
(类型说明符*)calloc(n,size)
功能：在内存动态存储区中分配n块长度为“size”字节的连续区域。函数的返回值为该区域的首地址。

- (类型说明符*)用于强制类型转换。
- calloc函数与malloc 函数的区别仅在于一次可以分配n块区域。

例如：
ps=(struet stu*)calloc(2,sizeof(struct stu));
其中的sizeof(struct stu)是求stu的结构长度。因此该语句的意思是：按stu的长度分配2块连续区域，强制转换为stu类型，并把其首地址赋予指针变量ps。

3) 释放内存空间函数free

调用形式：
free(void*ptr);
功能：释放ptr所指向的一块内存空间，ptr是一个任意类型的指针变量，它指向被释放区域的首地址。被释放区应是由malloc或calloc函数所分配的区域。

【例11-8】分配一块区域，输入一个学生数据。

```
1. main(){
2. struct stu{
3. int num;
4. char*name;
```

```
5. char sex;
6. float score;
7. }*ps;
8.    ps=(struct stu*)malloc(sizeof(struct stu));
9.    ps->num=102;
10.    ps->name="Zhang ping";
11.    ps->sex='M';
12.    ps->score=62.5;
13. printf("Number=%d\nName=%s\n",ps->num,ps->name);
14. printf("Sex=%c\nScore=%f\n",ps->sex,ps->score);
15. free(ps);
16. }
```

本例中，定义了结构stu，定义了stu类型指针变量ps。然后分配一块stu大内存区，并把首地址赋予ps，使ps指向该区域。再以ps为指向结构的指针变量对各成员赋值，并用printf输出各成员值。最后用free函数释放ps指向的内存空间。整个程序包含了申请内存空间、使用内存空间、释放内存空间三个步骤，实现存储空间的动态分配。

补充 realloc

语法

指针名=（数据类型*）realloc（要改变内存大小的指针名，新的大小）。

新的大小可大可小（但是要注意，如果新的大小小于原内存大小，可能会导致数据丢失，慎用！）

头文件

#include <stdlib.h> 有些编译器需要#include <malloc.h>，在TC2.0中可以使用alloc.h头文件

功能

先判断当前的指针是否有足够的连续空间，如果有，扩大mem_address指向的地址，并且将mem_address返回，如果空间不够，先按照newsize指定的大小分配空间，将原有数据从头到尾拷贝到新分配的内存区域，而后释放原来mem_address所指内存区域（注意：原来指针是自动释放，不需要使用free），同时返回新分配的内存区域的首地址。即重新分配存储器块的地址。

返回值

如果重新分配成功则返回指向被分配内存的指针，否则返回空指针NULL。

注意

当内存不再使用时，应使用free()函数将内存块释放。

使用总结

1. realloc失败的时候，返回NULL
2. realloc失败的时候，原来的内存不改变，不会释放也不会移动
3. 假如原来的内存后面还有足够多剩余内存的话，realloc的内存=原来的内存+剩余内存,realloc还是返回原来内存的地址 假如原来的内存后面没有足够多剩余内存的话，realloc将申请新的内存，然后把原来的内存数据拷贝到新内存里，原来的内存将被free掉,realloc返回新内存的地址
4. 如果size为0，效果等同于free()。这里需要注意的是只对指针本身进行释放，例如对二维指针**a，对a调用realloc时只会释放一维，使用时谨防内存泄露。
5. 传递给realloc的指针必须是先前通过malloc(), calloc(), 或realloc()分配的
6. 传递给realloc的指针可以为空，等同于malloc。

realloc,malloc,calloc的区别

三个函数的申明分别是：

void* realloc(void* ptr, unsigned newsize);

void* malloc(unsigned size);
void* calloc(size_t numElements, size_t sizeOfElement);
都在stdlib.h函数库内
它们的返回值都是请求系统分配的地址,如果请求失败就返回NULL

malloc用于申请一段新的地址,参数size为需要内存空间的长度,如:
char* p;
p=(char*)malloc(20);
calloc与malloc相似,参数sizeOfElement为申请地址的单位元素长度,numElements为元素个数,如:
char* p;
p=(char*)calloc(20,sizeof(char));
这个例子与上一个效果相同
realloc是给一个已经分配了地址的指针重新分配空间,参数ptr为原有的空间地址,newsize是重新申请的地址长度
如:
char* p;
p=(char*)malloc(sizeof(char)*20);
p=(char*)realloc(p,sizeof(char)*40);
注意，这里的空间长度都是以字节为单位。

C语言的标准内存分配函数：malloc，calloc，realloc，free等。
malloc与calloc的区别为1块与n块的区别：
malloc调用形式为(类型*)malloc(size)：在内存的动态存储区中分配一块长度为“size”字节的连续区域，返回该区域的首地址。
calloc调用形式为(类型*)calloc(n，size)：在内存的动态存储区中分配n块长度为“size”字节的连续区域，返回首地址。
realloc调用形式为(类型*)realloc(*ptr，size)：将ptr内存大小增大到size。

free的调用形式为free(void*ptr)：释放ptr所指向的一块内存空间。
C++中为new/delete函数。