10 动态内存分配

一. 传统数组缺点：

1. **数组长度必须事先指定，而且只能是长整数，不能是变量**

例：

int a[5] ; //ok

int len = 5; int a[len] //error

2. **传统形式定义的数组，该数组的内存无法手动释放**：数组一旦定义，系统为数组分配的存储空间就一直存在，除非该数组所在的函数执行结束，由系统释放

3. **数组的长度一旦定义，其长度就不能在程序运行过程中动态的更改**

4. **传统方式定义的数组不能跨函数使用：**A函数中定义的数组，在A函数运行期间可以被其他函数调用，但A函数运行完毕后，A函数中的数组将无法再被其他函数调用

二. 为什么需要动态分配内存

动态数组很好的解决了传统数组的4个缺点

传统数组也成为**静态数组**

三. 动态内存分配举例

▪ **malloc**函数：int\*p = (int \*)malloc(sizeof( int));  
▪ **calloc**函数：int\*p = (int \*)calloc(内存数,sizeof( int));（calloc分配完毕后该内存所有内容都被初始化为**0**）  
▪ **free**函数

例1：

node \*f(int n)

{

int i=0;

node \*p;

node \*h=NULL;//创建head指针

while(i<n)

{

p=(node\*)malloc(sizeof(node));//分配内存

scanf("%d",&(p->data));

p->next=h;

h=p;

i++;

}

return h;

}

例2：

char \*Ptr = NULL;  
        Ptr = (char \*)malloc(100 \* sizeof(char));  
        if (NULL == Ptr)  
    {   
        exit (1);   
    }

例3：

释放所有节点

node \*s(node \*l)

{

node \*p;

while(l!=NULL)

{

p=l;

l=l->next;

free(p);

}

return l;

}  
构造一维动态数组：

1. # include <stdio.h>
2. # include <malloc.h>
4. **int** main(**void**)
5. {
6. **int** len;
7. **int** \* pArr;
9. printf("请输入您要存放的元素的个数");
10. scanf("%d", &len);
12. /\*
13. 构造出了一个动态数组，该数组长度为len，数组名叫做pArr
14. 假设len的值是5
15. 那么该程序通过malloc函数开辟了20个字节的内存单位
16. pArr变量的值是这20个字节中第一个字节的地址！
17. pArr作为一个int型的指针，所指向的是前4个字节单位
18. \*/
19. pArr = (**int** \*)malloc(4 \* len);
21. // 此时pArr就是一个动态初始化出来的一个数组，一共可以存放5个int型变量
22. // 可以通过pArr[0]或者\*(pArr+1)来访问其中的值
24. free(pArr);
25. **return** 0;
26. }

变量n在编译时没有确定的值，而是在运行中输入，按运行时所需分配堆空间，这一点是动态分配的优点，可克服数组“大开小用”的弊端，在表、排序与查找中的[算法](http://lib.csdn.net/base/datastructure)，若用动态数组，通用性更佳。

四. 静态内存和动态内存比较

静态内存是由系统自动分配的，由系统自动释放

静态内存是在栈分配的

注：

**内存区间**

C/C++定义了4个内存区间：代码区，静态存储区，局部变量区即栈区（存放自动变量或者函数形参），动态存储区，即堆（heap）区或自由存储区。

**静态存储区**

外部变量、全局变量、静态变量存放在其中。

**栈区**

栈是线程独有的，保存其运行状态和局部自动变量、形式参数以及函数的返回地址等。通常定义变量（或对象），编译器在编译时都可以根据该变量（或对象）的类型知道所需内存空间的大小，从而系统在适当的时候为他们分配确定的存储空间。栈在线程开始的时候初始化，每个线程的栈互相独立。每个函数都有自己的栈，栈被用来在函数之间传递参数。[操作系统](http://lib.csdn.net/base/operatingsystem)在切换线程的时候会自动的切换栈，就是切换SS/ESP寄存器。

**堆区**

有些操作对象只在程序运行时才能确定，这样编译时就无法为他们预定存储空间，只能在程序运行时，系统根据运行时的要求进行内存分配。所有动态存储分配都在堆区中进行。

当程序运行到需要一个动态分配的变量或对象时，必须向系统申请取得堆中的一块所需大小的存储空间，用于存储该变量或对象。当不再使用该变量或对象时，也就是它的生命结束时，要显式释放它所占用的存贮空间，这样系统就能对该堆空间进行再次分配，做到重复使用有限的资源。