西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

获取动态内存

1、动态内存的概念

2、分配动态内存

3、释放动态内存

7.6 动态内存

- 有时，需要在程序运行中获得存储空间大小随时变化（而非固定大小）的内存块，这就需要用到动态内存。

7.6.1 动态内存的概念

- 在使用数组的时候，总有一个问题困扰着我们：数组应该有多大？例如编写程序求N阶行列式的值，用数组表示行列式，需要如下定义：

```
double A[N][N]; //NxN二维数组表示N阶行列式
```

7.6.1 动态内存的概念

- 前面讲过数组定义时，方括号内必须是常量，因此确切的定义应如下：

```
#define N 10
double A[N][N]; //NxN二维数组表示N阶行列式
```

- 而意图

```
int n;
scanf("%d",&n); //输入n值
double A[n][n]; //意图由输入n值确定数组大小
```

- 是错误的。

7.6.1 动态内存的概念

- 在很多情况下，并不知道实际运行时数组到底有多大。那么就要把数组定义得足够大，并且运行时要对下标值做限制。否则，当定义的数组不够大时，可能引起数组越界，导致严重错误。

- 如果因为某种特殊原因对数组的大小有增加或者减少，则必须重新去修改和编译程序。之所以出现这样的问题，是因为静态内存分配。

7.6.1 动态内存的概念

- C语言内存分配有两种方式：静态分配和动态分配。
- 静态分配指在编译时为程序中的数据对象分配相应的存储空间，例如程序中所有全局变量和静态变量，函数中的非静态局部变量等，本课程前面所有例子中的变量、数组、指针定义等均是静态分配方式。
- 由于是在编译时为数据对象分配存储空间，因此就要求在编译时空间大小必须是明确的，所以数组的长度必须是常量。而一旦编译完成，运行期间这个数组的长度就是固定不变的。

7.6.1 动态内存的概念

- 动态分配是程序运行期间根据实际需要动态地申请或释放内存的方式，它不象数组等静态内存分配方式那样需要预先分配存储空间，而是根据程序的需要适时分配，且分配的大小就是程序要求的大小。因此，动态分配方式有如下特点：
- ①不需要预先分配存储空间；
- ②分配的空间可以根据程序的需要扩大或缩小；

7.6.1 动态内存的概念

- 静态分配的内存在程序内存布局的数据区和栈区中，动态分配的内存在程序内存布局的堆区中。堆区的存储空间上限是物理内存的上限，甚至有的操作系统在物理内存不够时，用硬盘来虚拟内存，因此动态分配能得到比静态分配更大的内存。
- 动态分配的缺点是运行效率不如静态分配，因为它的分配和释放会产生额外的调用开销。实际编程中，在运行时分配或内存大小需要随时调整等情况下才使用动态分配方式。

7.6.2 动态内存的分配和释放

- C语言动态内存管理是通过标准库函数来实现的，其头文件为stdlib.h。

7.6.2 动态内存的分配和释放

- 1．动态内存分配函数
- （1）malloc函数
- malloc用于分配一个指定大小的内存空间，函数原型为：

```
void *malloc(size_t size);
```

- 若分配成功，函数返回一个指向该内存空间起始地址的void类型指针；分配失败，函数返回0值指针NULL。参数size表示申请分配的字节数，类型size_t一般为unsigned int。

7.6.2 动态内存的分配和释放

- 实际编程中，malloc函数返回的void类型指针可以显式转换为其他指针类型。调用函数时，一般使用sizeof来计算内存空间的大小，因为不同系统中数据类型的空间大小可能不一样。需要注意，分配得到的内存空间是未初始化的，即内存中的数据是不确定的。

7.6.2 动态内存的分配和释放

- 例如：分配一个int型内存空间

```
int *p;
p=(int *) malloc( sizeof(int) );
```

- 若分配成功，p指向分配得到的内存单元，*p表示该内存单元。显然，动态分配得到的内存空间要按指针方式访问。一般情况下，p值不能改变，否则该内存单元的起始地址就“永远是个谜”。换言之，程序再无可能使用该内存单元（因为不知指向哪里）。由程序申请的一块动态内存，且没有任何一个指针指向它，那么这块内存就泄露了。

7.6.2 动态内存的分配和释放

- malloc函数分配失败的主要原因是没有足够的内存空间可以分配，所以在内存分配后，要对它的返回值进行检查，确保指针是否有效，如下代码形式：

```
if (p!=NULL) { //分配失败时p为NULL
   ...//引用*p
}
```

7.6.2 动态内存的分配和释放

- （2）calloc函数
- calloc用于分配n个连续的指定大小的内存空间，函数原型为：

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

- 每个内存空间的大小为size个字节，总字节为n*size，并且将分配得到的内存空间的所有数据初始化为0。若分配成功，函数返回一个指向该内存空间起始地址的void类型指针；分配失败，函数返回0值指针NULL。

7.6.2 动态内存的分配和释放

- 例如：分配50个int型（相当于int A[50]数组）内存空间

```
int *p;
p=(int *) calloc(50, sizeof(int) );
```

- 等价于

```
int *p;
p=(int *) malloc( 50*sizeof(int) );
```

7.6.2 动态内存的分配和释放

- 2. 动态内存调整函数
- realloc函数用于调整已分配内存空间的大小，函数原型为：

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

- realloc将指针ptr所指向的动态内存空间扩大或缩小为size大小，无论扩大或缩小，原有内存中的内容将保持不变，缩小空间会丢失缩小的那部分内容。如果调整成功，函数返回一个指向调整后的内存空间起始地址的void类型指针。

7.6.2 动态内存的分配和释放

- 例如：

```
int *p;
p=(int*)malloc( 50*sizeof(int) );
//分配一个有50个int整型的内存空间，相当于int A[50]
p=(int*)realloc(p,10*sizeof(int) );
//调整为有10个int整型的内存空间，相当于int A[10]
p=(int*)realloc(p,100*sizeof(int) );
//再次调整为有100个整型的内存空间，相当于int A[100]
```

7.6.2 动态内存的分配和释放

- 3. 动态内存释放函数
- free函数用来释放动态分配的内存空间，函数原型为：

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

- 参数ptr指向已有的动态内存空间。如果ptr为NULL，则free函数什么也不做。

7.6.2 动态内存的分配和释放

- 实际编程中，若某个动态分配的内存空间不再使用时，应该及时将其释放。在动态分配的内存空间释放后，就不能再通过指针去访问，否则会导致程序出现崩溃性错误。
- 通常，ptr释放之后，需要设置ptr等于NULL，避免产生“迷途指针”。例如：

```
p=(int*) malloc(sizeof(int) ); //分配一个整型空间，指针p是有效指针
.......
free(p); //释放p所指向的内存空间，指针p变成迷途指针
p=NULL; //设置p为0值指针NULL，指针p不是迷途指针
```


程序设计