第8章 善于利用指针

8.8 动态内存分配与指向它的指针变量

8.8.1 什么是内存的动态分配

- 非静态的局部变量是分配在内存中的动态存储区的，这个存储区是一个称为栈的区域
- **C**语言还允许建立内存动态分配区域，以存放一些临时用的数据，这些数据需要时随时开辟，不需要时随时释放。这些数据是临时存放在一个特别的自由存储区，称为堆区
- 对内存的动态分配是通过系统提供的库函数来实现的，主要有**malloc**，**calloc**，**realloc**，**free**这**4**个函数。

8.8.2 怎样建立内存的动态分配

1．**malloc**函数(**memory allocation**)

- 其函数原型为

  void *malloc(unsigned int size);

  - 其作用是在内存的动态存储区中分配一个长度为**size**的连续空间，但不对存储空间初始化
  - 函数的值是所分配区域的第一个字节的地址，或者说，此函数是一个指针型函数，返回的指针指向该分配域的开头位置
  - **malloc(100);**开辟**100**字节的临时分配域，函数值为其第**1**个字节的地址
- 注意指针的基类型为**void**，即不指向任何类型的数据，只提供一个地址
- 如果此函数未能成功地执行(例如内存空间不足)，则返回空指针(**NULL**)
  - 用法：**T *p = (T*)malloc(n*sizeof(T));**

8.8.2 怎样建立内存的动态分配

2．calloc函数(clear allocation)

- 其函数原型为
  void *calloc(unsigned n,unsigned size);
- 其作用是在内存的动态存储区中分配**n**个长度为**size**的连续空间，并自动初始化内存空间为零。这个空间一般比较大，足以保存一个数组。
- 用**calloc**函数可以为一维数组开辟动态存储空间，**n**为数组元素个数，每个元素长度为**size**。这就是动态数组。函数返回指向所分配区域起始位置的指针；如果分配不成功，返回**NULL**。如：**p = calloc(50,4);**
  - ◆开辟**50×4**个字节的临时分配域，把起始地址赋给指针变量**p**
- 经典用法：
  - ◆**T *p = (T*)calloc(n, sizeof(T));**

8.8.2 怎样建立内存的动态分配

3. realloc函数

- 其函数原型为

void *realloc(void *p, unsigned int size);

- 如果已经通过**malloc**函数或**calloc**函数获得了动态空间，想改变其大小，可以用**recalloc**函数重新分配。
- 用**realloc**函数将**p**所指向的动态空间的大小改变为**size**。**p**的值不变。如果重分配不成功，返回**NULL**。

  如：　　**realloc(p,50);**

  ◆将**p**所指向的已分配的动态空间改为**50**字节
- 经典用法：

  ◆**p = (T*)realloc(p, n*sizeof(T));**

8.8.2 怎样建立内存的动态分配

4．free函数

- 其函数原型为

  void free(void *p);

- 其作用是释放指针变量 p 所指向的动态空间，使这部分空间能重新被其他变量使用。**p**应是最近一次调用**calloc**或**malloc**函数时得到的函数返回值。
- 经典用法：
  - **free(p);**
    - 释放指针变量 p 所指向的已分配的动态空间
    - **free**函数无返回值
- 以上**4**个函数的声明在**stdlib.h**头文件中，在用到这些函数时应当用“**#include <stdlib.h>**”指令把**stdlib.h**头文件包含到程序文件中。

8.8.3 void指针类型

例**8.21** 建立动态数组，输入**5**个学生的成绩**(**整数**)**，另外用一个函数检查有无低于**60**分的，并输出不合格的成绩。

- 分析：用**malloc/calloc**函数开辟一个动态存储区域，用来存放**5**个学生的成绩
- 然后得到这个动态区域的起始地址，因为内存本身是无类型之分的，故它的基类型是**void**型。用一个基类型为**int**的指针变量**p**来指向该动态数组，并输出它们的值。但必须先把**malloc**函数返回的**void**指针转换为整型指针，然后赋给**p**

```
#include <stdio.h>
#include <stdlib.h>//#include <malloc.h>
void check(int *, int);
int main()
{
   int *p, i;
   p = (int *)calloc(5, sizeof(int));
   for(i=0;i<5;i++) scanf("%d",p+i);
   check(p, 5);
   return 0;
}
void check(int *p, int n) {
   int i;
   printf("They are fail:");
   for(i=0;i<n;i++)
      if (p[i]<60) printf("%d ",p[i]);
   printf("\n");
}
```

```
67 98 59 78 57
They are fail:59 57
```

8.9有关指针的小结

1.首先要准确地弄清楚指针的含义。指针就是地址，凡是出现“指针”的地方，都可以用“地址”代替，例如，变量的指针就是变量的地址，指针变量就是地址变量

- 要区别指针和指针变量。指针就是地址本身，而指针变量是用来存放地址的变量。

2. 什么叫“指向”？地址就意味着指向，因为通过地址能找到具有该地址的对象。对于指针变量来说，把谁的地址存放在指针变量中，就说此指针变量指向谁。但应注意：只有与指针变量的基类型相同的数据的地址才能存放在相应的指针变量中。

8.9有关指针的小结

void *指针是一种特殊的指针，不指向任何类型的数据，如果需要用此地址指向某类型的数据，应先对地址进行类型转换。可以在程序中进行显式的类型转换，也可以由编译系统自动进行隐式转换。无论用哪种转换，读者必须了解要进行类型转换

3. 要深入掌握在对数组的操作中怎样正确地使用指针，搞清楚指针的指向。一维数组名代表数组首元素的地址

int *p, a[10]; p=a;

- **p**是指向**int**类型的指针变量，**p**只能指向数组中的元素，而不是指向整个数组。在进行赋值时一定要先确定赋值号两侧的类型是否相同，是否允许赋值。
- 对“**p=a;**”，准确地说应该是：**p**指向**a**数组的首元素

8.9有关指针的小结

4.各种指针变量的定义形式

T a, *p = &a; //T类型的指针变量**p**，指向**T**类型变量**a**

T a[N], *p = a;//T类型指针变量**p**，指向**T**类型数组**a**

T a[M][N], (*p)[N];//T指向一行有**N**个元素的二维数组

T *p[N];//T类型指针数组，包含**N**个元素，每个元素均为**T**类型的指针变量

T **p;//T类型二重指针变量**p**，指向**T**类型的指针变量

T *func([参数列表**]);//**返回指针值的函数

T (*p)(T1, T2, …, Tn);//指向函数的指针

8.9有关指针的小结

5.指针运算

(1) 指针变量加（减）一个整数

例如：**p++,p--,p+i,p-i,p+=i,** p **-=i**等均是指针变量加（减）一个整数。

- 将该指针变量的原值**(**是一个地址**)**和它指向的变量所占用的存储单元的字节数相加（减）。

(2) 指针变量赋值

- 将一个变量地址赋给一个指针变量
- 不应把一个整数赋给指针变量

(3) 两个指针变量可以相减

- 如果两个指针变量都指向同一个数组中的元素，则两个指针变量值之差是两个指针之间的元素个数

8.9有关指针的小结

5.指针运算

(4) 两个指针变量比较

- 若两个指针指向同一个数组的元素，则可以进行比较
- 指向前面的元素的指针变量“小于”指向后面元素的指针变量
- 如果**p1**和**p2**不指向同一数组则比较无意义

(5) 指针变量可以有空值，即该指针变量不指向任何变量，可以这样表示：

p=NULL;