河海大学

第七章 指针

7.2 数组与指针

1. 指向数组元素的指针

每个数组元素都在内存中占用存储单元，它们都有相应的地址。指针变量也可以指向数组元素（把某一元素的地址放到一个指针变量中）。所谓数组元素的指针就是数组元素的地址。

```
int a[10];          //定义一个整型数组a，它有10个元素
int *p;             //定义一个基类型为整型的指针变量p
p = &a[0];    //将元素a[0]的地址赋给指针变量p，使p指向a[0]
printf("%d %d",*p, a[0]);
```

在定义指针变量时可以给它赋初值：

```
int *p = &a[0];          //p的初值为a[0]的地址
```

指针运算

```
#include <stdio.h>
int main(){
        int i,j;
        int *p1=&i, *p2=&j;
        printf("%d", p1+p2);
}
```

(1) 两个指针之间不能加，因为没有意义。

[Error] invalid operands of types 'int*' and 'int*' to binary 'operator+'

指针运算

```
#include <stdio.h>
int main(){
    int i;
    int *p;
    p = &i;
    printf("%d", sizeof(int) );
   printf("%d %d", p, (p+1));
}
```

```
4
0x28fed8   0x28fedc
```

（2） 指针p加n，即p+n，向前跳过n个p所指向的基类型变量。

T *p;

p = p+n;

则p实际增加的值=n*sizeof(T)

指针运算

```
#include <stdio.h>
int main(){
    int a[] = {100,200,301,40};
    int *p = &a[0];

    for(int i=0; i<4; i++){
        printf("%d ",*(p+i));
    }
}
```

（2） 指针p加n，即p+n，向前跳过n个p所指向的基类型变量。

T *p;

p = p+n;

则p实际增加的值=n*sizeof(T)

指针运算

```
#include <stdio.h>
int main(){
        int a[] = {100,200,301,40};
        int *p = &a[0];

        for(int i=0; i<4; i++){
            printf("%d "",*);
            p++;
        }
}
```

（2） 指针p加n，即p+n，向前跳过n个p所指向的基类型变量。

T *p;

p = p+n;

则p实际增加的值=n*sizeof(T)

指针运算

```
#include <stdio.h>
int main(){
    float f;
    float*pf;
    pf = &f;
    printf("%d", sizeof(float));
    printf("%d %d",pf , (pf-1);
}
```

```
4
0x28fed8   0x28fed4
```

（3） 指针p减n，即p-n，向后跳过n个p所指向的基类型变量。

```
T *p;
p = p-n;
```

则p实际减少的值=n*sizeof(T)

指针运算

```
#include <stdio.h>
int main(){
   float a[] = {100,200,301,40};
   float *p = &a[3];

      for(int i=0; i<4; i++){
          printf("%d ", *p);
          p--;
      }
}
```

（3） 指针p减n，即p-n，向后跳过n个p所指向的基类型变量。

```
T *p;
p = p-n;
```

则p实际减少的值=n*sizeof(T)

指针运算

```
#include <stdio.h>
int main(){
   float a[] = {100,200,301,40};
   float *p1, *p2;
   p1 = &a[0]; p2 = &a[3];

   printf("%d", p2-p1); //3
   printf("%d",(char*)p2-(char*)p1;//12
    return 0;
}
```

（4）指针相减，得到指针之间的元素个数。

指针运算

```
#include <stdio.h>
int main(){
   float a[] = {100,200,301,40};
   float *p1, *p2;
   p1 = &a[0];    p2 = &a[3];
   for(;p1<=p2;){
              printf("%d ", *p1);
              p1++;
   }
}
```

（5）指针之间比大小。

两个相同类型指针指向连续区域时，才有可能进行比较。

指针运算

```
#include <stdio.h>

int main(){
   float f1, f2;
   float *p1, *p2;
   p1 = &f1;
   p2 = &f2;
   for(;p1<p2;)
       printf("%d ", *p1);
   return 0;
}
```

（5）指针之间比大小。

系统不保证f1, f2之间内存的位置关系，因此p1<p2没有固定的关系，有可能相差很远。

指针运算总结

```
T *p;
p = p+n; //则p实际增加的值= n*sizeof(T)
p = p-n; //则p实际减少的值= n*sizeof(T)
```

用于对连续的空间的访问。

一维数组名是一个地址常量

T ar[100];

ar在内部表示一个地址，该地址为数组第一个元素的地址。类型为 T*。

一维数组名代表整个数组，但是从数据类型的角度来看，数据名是一个指针，类型为指向元素的指针。

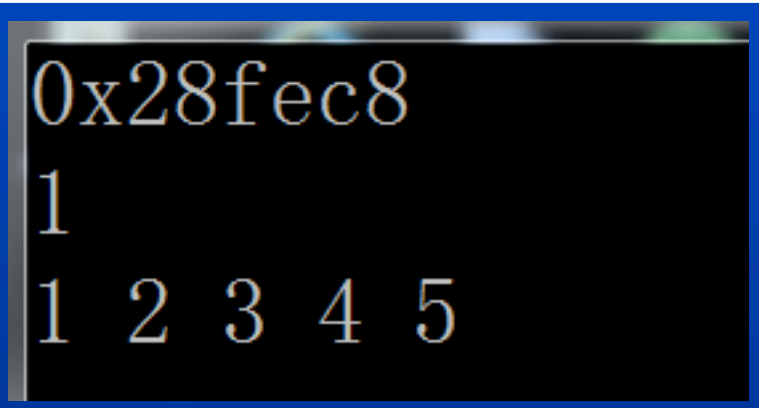
```
int a[10];
int *p;
p = a; //将数组名代表的地址赋值给指针变量
//a是int * const类型
```

2. 一维数组名是一个地址常量

```
#include <stdio.h>
int main(){
    int ar[] = {1,2,3,4,5};

    printf("%d\n", ar);
    printf("%d\n", *ar);

    for(int i=0; i<5; i++){
        printf("%d", *(ar+i));
    }
}
```



当我们访问ar[1]时，在内部是转换成*(ar+1).

当我们访问ar[n]时，在内部是转换成*(ar+n).

3. 指针的下标访问方式

```
#include <stdio.h>
int main(){
    int ar[] = {1,2,3,4,5};

   int *p = ar;

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

    return 0;
}
```

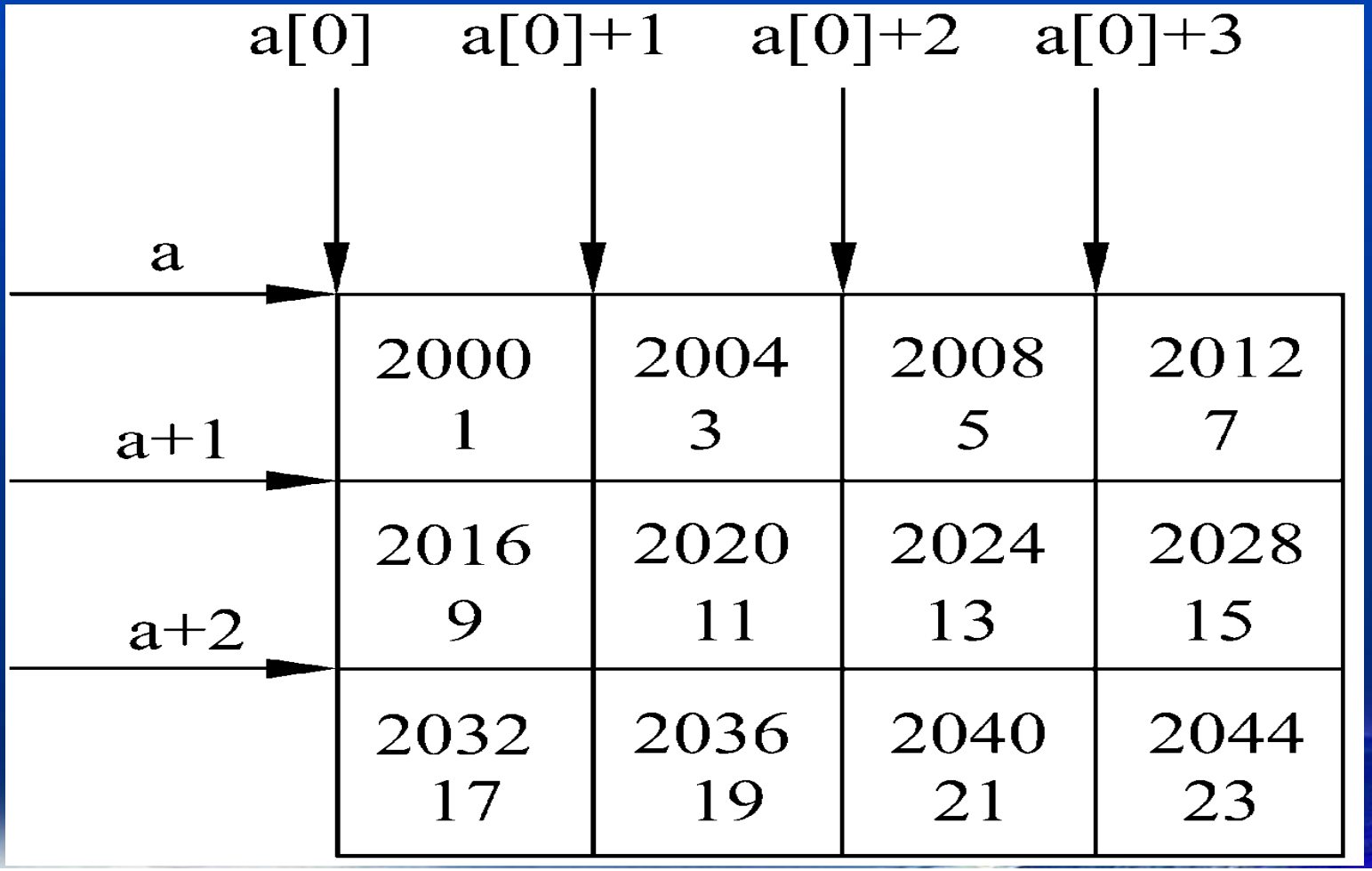
对指针也可以通过下标方式访问。

p[i]　与 *(p+i)等价

4. 二维数组名是一个地址常量

二维数组名指向的是整个一行

int a[3][4]



二维数组名是一个地址常量，指向的是整个一行

（1）二维数组可以看成是一个一维数组，每行（一行的所有元素作为一个整体）是该一维数组的一个元素；

（2）二维数组名a作为一个指针时，它指向它的元素（一整行）；

（3）*a表示整个第一行的一维数组，因此，与a[0]是相同的；

（4）a+1指向a所代表的一维数组的第二个元素（二维数组的第二行）；

（4）*(a+1)表示整个第二行一维数组，因此与a[1]是相同的。

二维数组名指向的是整个一行

a 指向的是a[0]代表的整个第一行。
a+1指向的是a[1]代表的整个第二行。

因此，
*a与a[0]值和类型相同；
*(a+1)与a[1]值和类型相同；
(*a)[0]代表a[0][0]；
(*(a+1))[0]代表a[1][0]。

二维数组名是一个地址常量，指向的是整个一行

```
#include <stdio.h>
int main(){
int a[3][3] ={{1,2,3},{4,5,6},{7,8,9}};
printf("%d %d\n", a ,a[0]);
printf("%d %d\n", a+1, a[1]);
printf("%d %d\n", a+2, a[2]);
printf("%d %d\n", *a, a[0]);
printf("%d %d\n", *(a+1), a[1]);
printf("%d %d\n", (*a)[0], a[0][0]);
}
```

```
0x28febc 0x28febc
0x28fec8 0x28fec8
0x28fed4 0x28fed4
0x28febc  0x28febc
0x28fec8  0x28fec8
1  1
```

a 与 a[0] 虽然值相同，但是类型不同。

```
C:\Users\Administrator\Desktop\1.exe
0x28fec8  0x28fec8
0x28fed4  0x28fed4
1  1
```

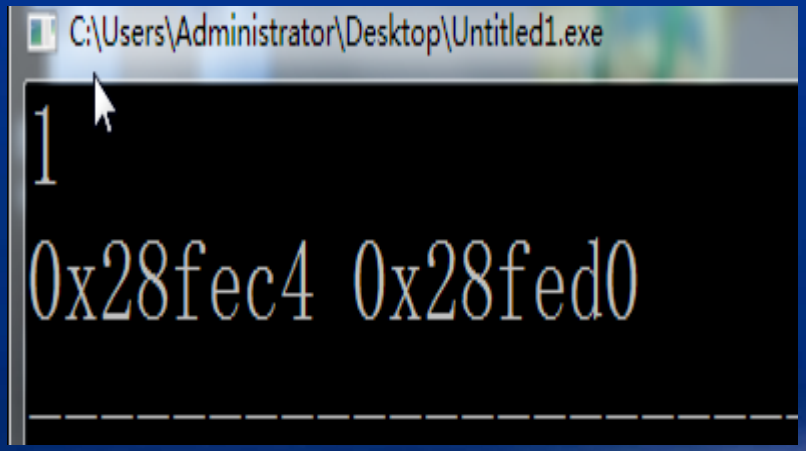
```
#include <stdio.h>
int main(){
        int a[3][3] ={{1,2,3},{4,5,6},{7,8,9}};

        printf("%d %d\n", *a, a[0]);
        //注意：a指向一维数组，*a指向整数
        printf("%d %d\n", *(a+1), a[1]);
        printf("%d %d\n", (*a)[0] , a[0][0]);
}
```

5. 指向数组的指针 与 指针数组

（1）指向数组的指针

T（*p）[n]；//n为常量，P指向一个包含n个元素的一维数组，即p+1将跳过整个数组.




```
#include <stdio.h>

int main(){
        int a[2][3] ={{1,2,3},{4,5,6}};
        int (*p)[3];
        p = a;

        printf("%d", p[0][0]); //1
        printf("%d %d", p, p+1);
}
```

5. 指向数组的指针 与 指针数组

（1）指向数组的指针

a是指向数组的指针，其指向的数组具有3个元素，p是指向数组的指针，其指向的数组具有4个元素，因此a, p类型不兼容。

```
#include <stdio.h>
int main(){
        int a[2][3] ={{1,2,3},{4,5,6}};
        int (*p)[4];
        p = a;
}
```

5. 指向数组的指针 与 指针数组

（2）指针数组

指针数组中的每个元素都是一个指针。

T* p[n]; //n为常量

p是一个一维数组，每个元素都是一个指向T类型的指针变量。

```
#include <stdio.h>
int main(){
        int a[2][3] ={{1,2,3},{4,5,6}};
        int* p[3];
        p[0] = a[0];    p[1] = a[1];    p[2] = a[2];
        printf("%d", p[0][0]);
}
```

5. 指向数组的指针 与 指针数组

（2）指针数组 --指针数组与二级指针的关系

T* p[n]; //n为常量
p是一个一维数组，每个元素都是一个指向T类型的指针变量,或者说每个元素都是 T* 类型 。P指向 T*类型的变量。p是T**类型。

int p1[5]; //p是什么类型?
int* p2[5]; //p是什么类型?
int** p3; //p是什么类型?

分析程序的输出

```
#include <stdio.h>

int main(){
        int* p[2];
        int** p1;
        int a[][3]      = {1,2,3,4,5,6};
        p[0] = a[0];
        p[1] = a[1];
        p1 = p;

        //通过p1访问a中元素
        printf("%d\n", (*p1)[0]); //1
        printf("%d", (*(p1+1))[1]); //5
        printf("%d", *(p1+1)[1]);  //????
}
```

作 业

1. 编程判断输入的一串字符是否为“回文”，回文就是一个对称的字符串。
2. 用指针数组编程实现：从键盘任意输入一个数字表示月份值n,程序输出该月份的英文表示。若n不在1-12之间，则输出“Illegal month” 。

作 业

3.员工管理系统v1，实现一个员工信息管理系统，员工信息包括：姓名、工号、工资。姓名为不超过30个字符的字符串，工号是长度为5的字符串，假设员工数量不超过100人。

功能包括：

（1）员工信息增加；

（2）员工信息删除；

（3）查询员工信息：通过工号或者姓名

（4）输出所有员工信息，按照工资排序，按照工号排序。

作 业

4.预习字符串及其操作。

5. 熟悉math.h和string.h中的函数。

河海大学

欢 迎 交 流