计算概论（C语言）习题课讲义10

内容概要

- 习题讲解
- 指针
- 课堂练习

习题讲解

字典序输出所有排列

代码点评一:

```
#include <stdio.h>
#define Nmax 100

void print(int A[], int n)
{
    for(int i=0;i<n;i++)
    {
        printf("%d",A[i]);
    }
    printf("\n");
}

void swap(int A[], int m, int n) //交换两元素
{
    int tmp=A[m];
    A[m]=A[n];
    A[n]=tmp;
}

int next(int A[], int n)
{
    int k,i;
    for(k=n-2;A[k]>A[k+1]&&k>=0;k--); // Find first(right to left) A[k]<A[k+1]
    if(k<0) return 0; // This means A[j]>A[j+1] for all j
                      // Maximum in lexicographical order;
    for(i=n-1;A[k]>A[i];i--); // Find minimal A[i]>A[k]
    swap(A,k,i); // Exchange A[i] and A[k]
    //Sort A[k+1]--A[n-1]; Any sort is OK; But Notice it's in reverse order
    k=k+1,i=n-1;
    while(k<i)
    {
        swap(A,k,i);
        k++;
        i--;
```

```
    }
    return 1;
}

int main()
{
    int A[Nmax], n;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        A[i]=i+1;
    }
    do{
        print(A,n);
    }while(next(A,n));
    return 0;
}
```

代码点评二:

```
#include<stdio.h>
int a[50];
int b[50];
int num0;
void prt(int a[],int num){//递归
    if (num==1){
        int j=0;
        for(int i=0;i<num0;i++){
            if(a[i]!=0)j=i;
        }
        b[num0-1]=j+1;
        for(int i=0;i<num0;i++){
            printf("%d",b[i]);
        }
        printf("\n");
        return;
    }
    for(int i=0;i<num0;i++){
        if(a[i]==1){
            a[i]=0;
            b[num0-num]=i+1;
            prt(a,num-1);
            a[i]=1;
        }
    }
}
int main(){
    scanf("%d",&num0);
    for(int i=0;i<num0;i++)a[i]=1;
    prt(a,num0);
    return 0;
}
```

代码点评三:

```
// Depth First Search
#include <stdio.h>
int a[50],book[50],n;
void dfs(int tnt)
{
    int i;
    if(tnt==n+1)
    {
        for(i=1;i<=n;i++)
            printf("%d",a[i]);
        printf("\n");
        return;
    }
    for(i=1;i<=n;i++)
    {
        if(book[i]==0) // 找到还没有用过的最小数字
        {
            a[tnt]=i;
            book[i]=1;
            dfs(tnt+1);
            book[i]=0;
        }
    }
    return;
}
int main()
{
    int i;
    scanf("%d",&n);
    dfs(1);
    return 0;
}
```

指针

什么是指针

指针是一个变量，其值为另一个变量的地址.

```
int    *ip;    /* 一个整型的指针 */
double *dp;    /* 一个 double 型的指针 */
float  *fp;    /* 一个浮点型的指针 */
char   *cp;     /* 一个字符型的指针 */
```

所有指针的值的实际数据类型，其实都是一样的，都是一个代表内存地址的长整数。不同数据类型的指针之间唯一不同是，指针所指向的变量或常量的数据类型不同。

演示: 不同类型指针间的异同

指针的使用, 大体上和其他变量的使用相同. 需要特别注意的是, `&`取地址运算符以及`*`间接访问运算符. 这两个运算符,可以看作是互逆运算.

特殊的`NULL`指针

演示: NULL指针的地址

在大多数的操作系统上，内存地址0有特别重要的意义，所以按照惯例，如果指针包含空值（零值），则假定它不指向任何东西。

指针的算术运算

我们可以对指针进行这样四种算数运算: `++`、`--`、`+`、`-`; 这代表这地址的增加或减少.

问题**:** 指针加**1,**表示内存地址加**1?**

内存的基本单位是字节, 而指针加一不同于地址加一. 指针加一,地址增加多少需要具体根据指针的类型.

演示: 指针加一,地址加几?

指向指针的指针

一个指向指针的指针变量必须如下声明，即在变量名前放置两个星号。例如，下面声明了一个指向 int 类型指针的指针：

```
int **var;
```

当一个目标值被一个指针间接指向到另一个指针时，访问这个值需要使用两个星号运算符，如下面实例所示：

```
#include <stdio.h>
int main ()
{
   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   /* 获取 var 的地址 */
   ptr = &var;

   /* 使用运算符 & 获取 ptr 的地址 */
   pptr = &ptr;

   /* 使用 pptr 获取值 */
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);

   return 0;
}
```

指针做为函数参数

```
//指针可以做为函数参数,以及返回值
int *myFunction(int *a)
{

}
```

指针做为函数的参数时, 地址的"值传递",导致了所指向元素的实际改变. 典型的例子, 使用指针做为参数交换两个整数, 确实会交换两个整数.

指针做为函数的返回值时, *C语言不支持在调用函数时返回局部变量的地址，除非定义局部变量为static变量.* 这是为什么?? (考虑内存释放)

使用指针做为函数返回值的典例: 动态分配内存函数.

```
// 使用动态内存的一般做法
#include <stdlib.h>

int *ip=(int *)malloc(N*sizeof(int));

// ...

free(fp);
```

指针和数组的关系

数组名, 可以看做是一个指针, 它指向数组首个元素. 它和一般指针的区别在于: 它是一个常值, 即对它`++`或`--`是非法的.

更特殊的, `a[i]` 等价于 `*(a+i)`. 这也解释了, 为什么如下代码也是正确的了.

```
#include <stdio.h>

int main()
{
    int a[5]={0,1,2,3,4};
    printf("%d\n",0[a]); // 数组名和下标调换了位置
    printf("%d\n",1[a]);
    return 0;
}
```

对于*二维数组*呢? 二维数组的数组名相当于?

二维数组的数组名, 也是一个常指针, 指向数组的首个元素;

```c
#include <stdio.h>

int main()
{
    int B[2][3]={{1,2,3},{4,5,6}};
    printf("%d\n",**B); /为什么这里需要两个**
    return 0;
}
```

实际上, `B[i]`是一个指针数组, `B[i][j]` 相当于 `*(*(B+i)+j)`; 其中, `+j`内存地址加多少, 根据的是数组元素的类型; 但`+i`时内存地址的变化,实际上是`i*(j*sizeof(int))`. 这也是为什么二维数组的定义时,必须给定第二个维度大小的原因.

演示: 二维数组解析

课堂练习

根据上述二维数组和指针间的关系, 使用`malloc`函数动态生成一个二维数组.