本节主题：

算法及其描述

什么是算法

```
ADT <抽象数据类型名>
{
    数据对象：<数据对象的定义>
    数据关系：<数据关系的定义>
    基本操作：<基本操作的定义>
}
```

设计 ? 实现

数据表示
功能实现

```
34  double dMax(double *dArray, int iLenOfArray)
35  {
36      double max = dArray[0];
37      int iCount;
38
39      for (iCount = 0; iCount < iLenOfArray; ++iCount)
40      {
41          if (dArray[iCount] > max)
42          {
43              max = dArray[iCount];
44          }
45      }
46
47      return max;
48  }
```

- 数据元素之间的逻辑关系 ===> 逻辑结构上的操作功能
- 数据元素之间的物理关系 ===> 具体存储结构上的操作实现
- 算法的定义1：算法是在具体存储结构上，实现某个抽象运算
- 算法的定义2：算法是对特定问题求解方法(步骤)的一种描述，是指令的有限序列，其中每一条指令表示一个或多个操作。

算法的重要的特性

（1） 有穷性：在有穷步之后结束

（2） 确定性：无二义性

（3） 可行性：可通过基本运算有限次执行来实现

（4） 有输入

（5） 有输出

算法和程序的关系？

一个计算机程序是对一个算法使用某种程序设计语言的具体实现。

算法必须可终止，意味着不是所有的计算机程序都是算法。

```
void exam1()
{
   int  n = 2;
   while (n%2 == 0)
      n = n+2;
   printf("%d\n",n);
}
```

死循环，违反了算法的有穷性特征

不是算法

```
void exam2()
{
   int x,y;
   y=0;
   x=5/y;
   printf("%d,%d\n",x,y);
}
```

包含除零错误，违反了算法的可行性特征

好算法还应该

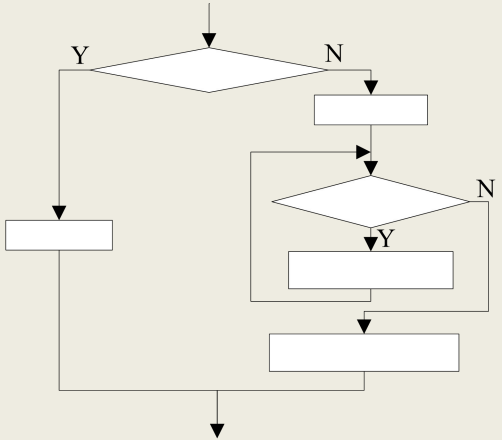
- **正确性(Correctness)**： 算法应满足具体问题的需求。
- **可读性(Readability)**： 算法应容易供人阅读和交流。可读性好的算法有助于对算法的理解和修改。
- **健壮性(Robustness)**： 算法应具有容错处理。当输入非法或错误数据时，算法应能适当地作出反应或进行处理，而不会产生莫名其妙的输出结果。
- **通用性(Generality)**： 算法应具有一般性 ，即算法的处理结果对于一般的数据集合都成立。
- **效率与存储量需求**： 效率指的是算法执行的时间；存储量需求指算法执行过程中所需要的最大存储空间。一般地，这两者与问题的规模有关。

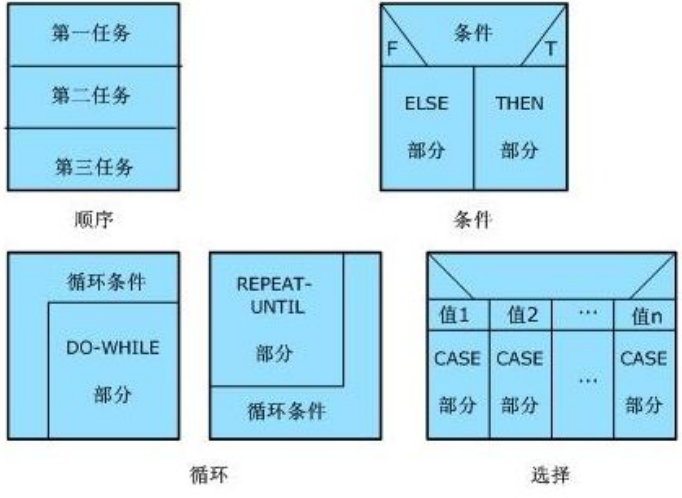
算法描述

- 自然语言
- 伪代码
- 传统流程图
- N-S结构化流程图
- 程序设计语言

算法 *Binomial* (*n*,*k*)

```
//用动态规划算法计算 C(n,k)
//输入：一对非负整数 n ≥ k ≥ 0
//输出：C(n,k)的值
for i ← 0 to n do
    for j ← 0 to min(i, k) do
        if j = 0 or j=k
            C[i, j] ← 1
        ekse C[i, j] ← C[i−1, j−1]+C[i−1, j]
return C[n,k]
```





用程序设计语言表示算法

- 例 读入两个整数x、y，要求交换它们的值

```
void swap1(int x,int y)
{
   int tmp;
   tmp=x;
   x=y;
   y=tmp;
}
```

```
void swap2(int *x,int *y)
{
   int tmp;
   tmp=*x;
   *x=*y;
   *y=tmp；
}
```

```
void swap3(int &x,int &y)
{
   int tmp;
   tmp=x;
   x=y;
   y=tmp;
}
```