数据结构与算法部分

关于**数据结构**和**算法**的通俗解释：数据结构要解决的问题就是把现实中大量复杂的问题存储到内存中，把单个数据的类型确定，再把数据之间关系确定，这样就可以存储到内存中去了，**算法就是对数据结构的操作。**

比如数组是一个数据结构，单个数据类型就确定，数据之间关系就是连续存储，操作数组就是一个算法。数据结构解决存储问题，算法解决数据间的关系。

数据结构 = 个体 + 个体的关系，算法 = 对存储数据的操作。 **侠义的算法定义：解题的方法和步骤。**

数据结构前提基础知识章

本章站在901考研的角度上介绍考试中到底怎么书写一些算法描述代码，另外掌握这些辅助知识可以方便入门。

关键点1：考研综合应用中算法设计题中的代码部分只需写出一个或多个可以解决问题的有着 *清楚接口* 描述的函数。

> Ps：所谓接口，是用户和函数打交道的地方，通过接口，用户输入了自己的数据，得到了自己想要的结果，也可以简单看成为函数的参数表。

对于基本的数据类型，例如：整型 `int`,`long`,.....（考研中涉及到处理整数的题目如果没有特别要求用 `int` 足够了），字符型 `char`，浮点型 `float`，`double`.....（**对于处理小数的问题，题目没有特殊要求的请况下用 float 就足够了**）。

指针型是**比较难理解**的部分。对于其他类型的变量，变量里所装的东西是数据元素的内容，而指针型变量内部装的是变量的地址，通过它可以找出这个变量在内存中的位置，就像一个指示方向的指针，指出了某个变量的位置，因此叫做指针型。

指针型的定义方法对每种数据类型有特定的写法。有专门指向 int 型变量的指针，有专门指向 char 型变量的指针，等等。对于每种变量，指针的定义方法有相似的规则，如以下语句：

```
int *a;     //对比一下定义 int 型变量的语句：int a;
char *b;    //对比一下定义 char 型变量的语句：char b;
float *c;   //对比一下定义float 型变量的语句：float c;
TypeA *d;   //对比一下定义 TypeA 型变量的语句：TypeA d;
```

上边四句分别定义了指向整型变量的指针 a，指向字符型变量的指针 b，指向浮点型变量的指针 c，和指向 TypeA 型变量的指针 d 。我们看到比起之前所讲其他变量的定义，指针型变量的定义只是在变量名之前多出一个 `*` 而已。

如果 a 是个指针型变量，且它已经指向一个变量 b，则 a 中存放了变量 b 所在的地址。`*a` 就是取变量 b 的内容 `x=*a;` 等价于 `x=b;`， `&b` 就是取变量 b 的地址，语句 `a=&b;` 就是将变量 b 的地址存于 a 中，即大家常说的指针 a 指向 b。

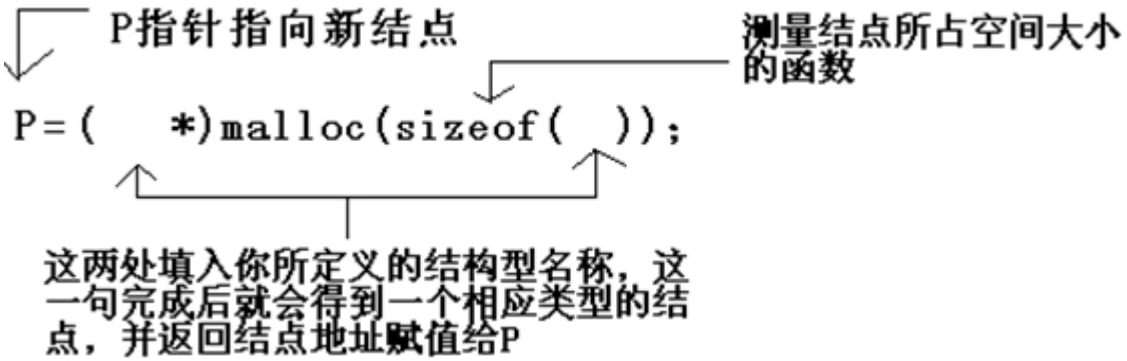
指针型在考研中用的最多的就是和结构型结合起来构造结点（如链表的结点，二叉树的结点.....）。下边我们就来具体讲讲常用结点的构造，这里的 构造 我们就把它理解成先定义一个结点的结构类型，然后用这个定义好的结构型制作一个结点。

以二叉树为例，有以下两种写法：

```
BTNode BT;
```

```
BTNode *BT;
BT=(BTNode*) malloc (sizeof(BTNode)); // 此句要熟练掌握
```

①中只有一句就制作了一个结点，而②中需要两句，比①要繁琐，但是考研中用的最多的是②。②的执行过程是这样：先定义一个结点的指针 BT，然后用 `malloc` 函数来申请一个结点的内存空间，最后让指针 BT 指向这片内存空间，就完成了一个结点的制作。②中的第二句就是用系统已有的 `malloc()` 函数申请新结点所需内存空间的方法，考研数据结构中所有类型结点的内存分配都可用 `malloc()` 来完成，模式固定，容易记忆。



对于①和②中的 BT 取分量的操作也是不同的。对于①如果我想取其 data 域的值付给x，则应该写成 `x=BT.data;` 而对于②则应该写成 `x=BT->data;` 一般来说，用结构体变量直接取分量，其操作用 `.` 指向结构体变量的指针来取分量，其操作用 `->`。这里再扩展一点，前边我们提到，如果 p 是指针（假设已经指向 x），`*p` 就是取这个变量的值，`a=*p;` 等价于 `a=x;` 那么对于②中的 BT 指针，怎么用 `.` 来取其 data 值呢？类比p，`*BT` 就是 `BT` 指向的变量，因此可以写成 `(*BT).data;`（`(*BT).data;` 与 `BT->data` 是等价的）。注意 `*BT` 外边要用括号括起来，不要写成 `*BT.data`。

在 C 或 C++语言中这是一种好的习惯，在你不知道系统默认的运算符优先级的情况下，你最好依照自己所期望的运算顺序加上括号。有可能这个括号加上是多余的，但是为了减少错误，这种做法是必要的。对于与刚才那句，我所期望的运算顺序是先算 `*BT`，即用 `*` 先将 `BT` 变成它所指的变量，然后再用 `.` 取分量值。因此写成 `(*BT).data`。比如这样一个式子 `a*b/c`，假设你不知道系统会默认先算乘再算除，而你所期望的运算优先顺序是先算乘再算除，为了减少错误，你最好是把它写成 `(a*b)/c`，即便这里的括号是多余的。

`#define` 对于考研数据结构可以说没有什么贡献，我们只要认得它就行，写程序时一般用不到。比如 `#define MAX 50` 这句，即定义了常量 `MAX`（此时 `x=50;` 等价于 `x=MAX;`）。在写程序大题的时候如果你要定义一个数组，如 `int A[MAX];` 加上一句注释：`/*MAX` 为已经定义的常量，其值为 `50*/` 即可。

被传入函数的参数是否会改变：

```
int a;
void f(int x) {   // 将 a 副本传入
    x++;
}
```

上边声明的函数，它需要一个整型变量作为参数，且在自己的函数体中将参数做自增 1 的运算。执行完以下程序段之后 a 的值是多少呢？

```
a = 0;
f(a);
```

有些同学可能以为 a 等于 1。这个答案是错误的，可以这样理解，对于函数 `f()`，在调用他的时候，括号里的变量 a 和句①中的变量 a 并不是同一个变量。在执行句②的时候，变量 a 只是把自己的值赋给了一个在 `f()` 的声明过程中已经定义好的整形变量，可以把这个变量想象为上述声明过程中的 `x`，即句②的执行过程拆开看来是这样两句：`x=a;x++;` 因此 a 的值在执行完①,②两句之后不变。

如果我想让 a 依照 `f()` 函数体中的操作来改变应该怎么写呢，这里就要用到函数的引用型（这种语法是 C++中的，C 中没有，C 中是靠传入变量的地址的方法来实现的，写起来比较麻烦且容易出错，因此这里采用 C++的语法），其函数声明方法如下：

```
void f(int &x) {
    x++;
}
```

这样就相当于 a 取代了 x 的位置，函数 f()就是在对 a 本身进行操作，执行完①②两句后，a 的值由 0 变为 1。

上边讲到的是对针对普通变量的“引用型”，如果传入的变量是指针型变量，且在函数体内要对传入的指针进行改变，则需写成如下形式：

```
void f(int *&x) {   // 指针型变量在函数体中需要改变的写法。
    x++;
}
```

执行完上述函数后，指针 x 的值自增 1。

说明：这种写法很多同学不太熟悉，但是它在树与图的算法中应用广泛，在之后的章节中考生要注意观察其与一般引用型变量的书写差别。

上边是单个变量作为函数参数的情况。如果一个数组作为函数的参数，该怎么写呢？传入的数组是不是也有“引用型”一说呢？对于数组作为函数的参数，这里讲两种情况，一维和二维数组。

一维数组作为参数的函数声明方法：

```
void f(int x[], int n) {
    .......
}
```

对于第一个参数位置上的数组的定义只需写出两个中括号即可，不需要限定数组长度（即不需要写成 `f(int x[5],int n)`，即便是你传入的数组真的是长度为 5），对于第二个参数 n，是写数组作参数的函数的习惯，用来说明将来要传进函数加工的数组元素的个数，并不是指数组的总长度。

二维数组作为参数的函数声明方法：

```
void f(int x[][MAX], int n) {   //一定要传入一个常量
    ......
}
```

如果函数的参数是二维数组，数组的第一个中括号内不需要写上数组长度，而第二个中括号内必须写上数组长度（假设 MAX 是已经定义的常量）。这里需要注意，你所传入的数组第二维长度也得是 MAX，否则出错，比如：

```
void f(int x[][5]) {
    ......
}
int a[10][5];
int b[10][3];
f(a);               //参数正确 f(b)
```

要注意的是，将数组作为参数传入函数，函数就是对传入的数组本身进行操作，即如果函数体内涉及到改变数组数据的操作，传入的数组中的数据就会依照函数的操作来改变。因此，对于数组来说，没有“引用型”和“非引用型”之分，可以理解为只要数组作为参数，都是引用型的。

对于一些基础稍差的同学来说，这个函数麻烦了，`STATUS`，`ELEMTYPE`，`ERROR`，`OK` 这都什么东西，其实严奶奶在离这个函数很远的地方写过这些语句：

```
#define ERROR 1
#define OK 0
typedef STATUS bool {} // 这句中的 bool 是布尔型，只取两个值 0 和 1，其实用 bool 用 int 型代替就可以
typedef ELEMTYPE int {}
```