# 第 7 章　函数与结构体

程序中有时会多次使用相同的语句，而且无法通过循环来减少重复编程。对于这样的功能，如果能像使用 sqrt( )、max( )这样变成一个函数，那该多好啊！其实每个程序都用到了主函数——main( )。除此之外，还可以自己定义其他函数，并将参数“喂给”这些函数，使其能够根据这些参数完成要求的任务。不过，这方面还有更复杂的一些知识点，比如参数传递与变量的作用域，接下来也需要学习。函数内还能调用自己，也就是递归函数，这是程序设计新手入门公认的第一道坎，但却是非常重要的一部分。

本章最后介绍了结构体，利用它可以建立并操作对象，并使存储一些和对象有关的信息变得相当便利。例如，可以设计结构体来存储一位同学的各项信息，如姓名、年龄、性别、考试成绩等，而一个确定的同学就是一个对象。利用结构体可以很方便地操作一个对象，也可以用数组批量存储对象。

本章是第 1 部分语言入门的最后一章，很快就能介绍完 C++ 的基础知识。图 7-1 所示为本章思维导图。

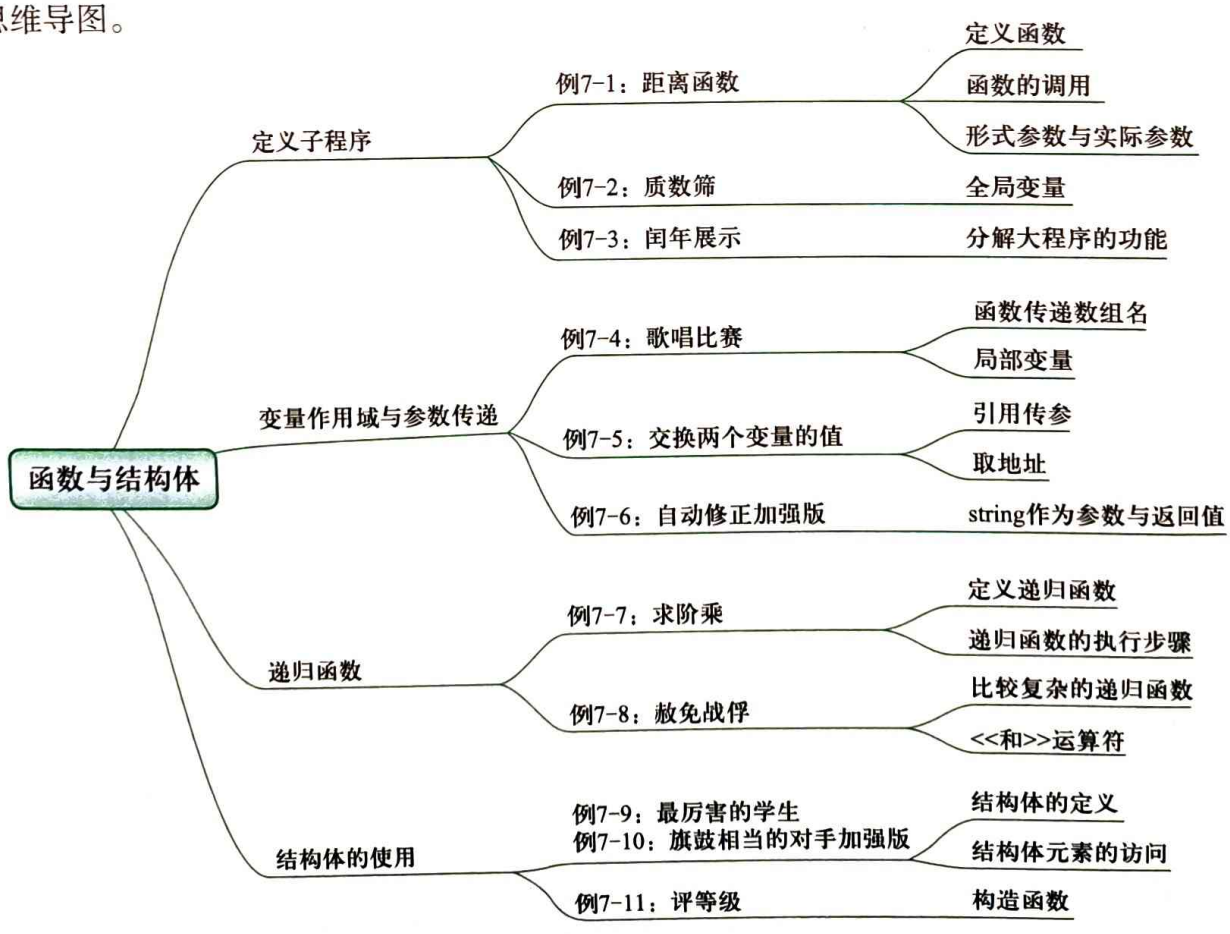


图 7-1　本章思维导图

## 7.1 定义子程序

**例 7-1**　距离函数(洛谷 P5735)。给出平面坐标上不在一条直线上 3 个点坐标$(x_1,y_1)$、$(x_2,y_2)$、$(x_3,y_3)$,坐标值是实数,且绝对值不超过 100.00,求其围成的三角形周长。

**分析:**3 个点,两两组成一条线段。平面上两个点的距离是$\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$。分别计算这三条线段的长度,累加到一起,就可以得到三角形的周长。可以得到下面的程序:

```
#include <cstdio>
#include <cmath>
using namespace std;
int main() {
    double x1, y1, x2, y2, x3, y3, ans;
    scanf("%lf%lf%lf%lf%lf%lf", &x1, &y1, &x2, &y2, &x3, &y3);
    ans = sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
    ans += sqrt((x3 - x2) * (x3 - x2) + (y3 - y2) * (y3 - y2));
    ans += sqrt((x3 - x1) * (x3 - x1) + (y3 - y1) * (y3 - y1));
    printf("%.2f", ans);
}
```

这个程序是完全正确的,但是显得比较啰嗦——手动算了 3 次两点距离和 6 次平方,不仅看起来复杂,而且还容易笔误。因此希望有一种办法减少重复的代码,简化程序的主干。因此,使用到了**函数**。改进后的代码如下:

```
#include <cstdio>
#include <cmath>
using namespace std;
double sq(double x) { // 计算平方
    return x * x;
}
double dist(double x1, double y1, double x2, double y2) { // 两点距离
    return sqrt(sq(x1 - x2) + sq(y1 - y2));
}
int main() {
    double x1, y1, x2, y2, x3, y3, ans;
    scanf("%lf%lf%lf%lf%lf%lf", &x1, &y1, &x2, &y2, &x3, &y3);
    ans = dist(x1, y1, x2, y2);
    ans += dist(x1, y1, x3, y3);
    ans += dist(x2, y2, x3, y3);
    printf("%.2f", ans);
}
```

函数定义的一般形式如下:

```
返回类型 函数名（参数类型1 参数名1, ..., 参数类型n 参数名n）{
    函数体
    return 结果；
}
```

函数又称为子程序，可以认为是厨房中的自造的机器。本例中定义了两个函数：一个是 sq( ) 函数，需要调用一个 double 类型的变量 x，经过计算后“吐出”一个 double 类型的结果；另一个是 dist( )函数，调用4个 double 类型的变量 x1、y1、x2、y2，经过计算后返回一个 double 类型的结果。由于在 main( )函数中使用了 dist( ) 函数，dist( ) 函数又使用了 sq( ) 函数，所以 sq( ) 函数应在 dist( ) 函数前面定义，dist( ) 函数应在 main( ) 函数前面定义。

当输入数据为“0 0 3 0 0 4”，计算 dist(x2,y2,x3,y3); 时，步骤如图 7-2 所示。

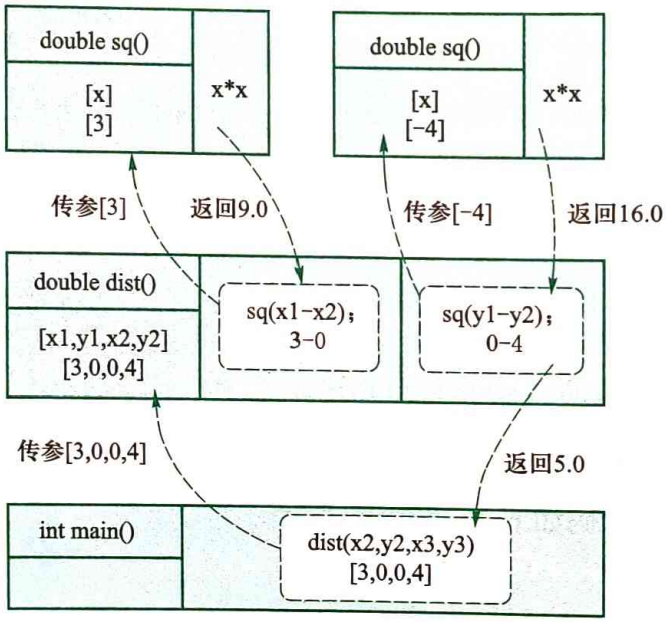


图 7-2　函数的运行步骤

运行步骤如下：

1) 进入 main( )函数(主程序)。由于主程序中的三个 dist( ) 函数是一样的形式，因此仅举 dist(x2,y2,x3,y3)作为例子。程序运行到这边时收集了4个**参数**[3,0,0,4]，然后传递给 dist( ) 函数，也就是调用 dist(3,0,0,4)。

2) 进入 dist( ) 函数，需要接收4个 double 类型的变量 x1、y1、x2、y2，这些是**形式参数**(因为传递参数之前，并不知道具体的值是什么)。传递过来的4个参数的值[3,0,0,4]称为**实际参数**，然后按照顺序分配给参数列表中的4个变量，因此 x1=3,y1=0,x2=0,y2=4。需要注意的是，这里的 x1 和 主程序的 x1 没有直接关系，是两个不同的变量。然后要求 sq(x1-x2)的值，程序接收到参数[3-0]，传递给 sq( ) 函数调用 sq(3)。

3) 进入 sq( ) 函数，接收到了传递归来的1个参数，按照顺序分配给参数列表中唯一的变量 x，因此 x=3。经过计算后，返回结果 9.0。

4) 回到 dist( ) 函数，得到了 sq(x1-x2) 的值是 9。同理，sq(y1-y2) 的值是 16。加起来取平方根，得到结果 5.0，返回这个结果。

5) 回到 main( ) 函数，得到了 dist(x2,y2,x3,y3) 的结果是 5.0。用同样的方法，得到其他两

个 dist 的结果。得到这些值之后累加，就得到了最终的答案并输出。

需要注意的是，在一个函数里定义的变量，在其他函数里是不能直接使用的。例如，在 dist( ) 函数里不能访问 main( ) 里面定义的 ans 变量进行累加操作。

**例 7-2**　质数筛（洛谷 P5736）。输入 $n$（$n \leqslant 100$）个不大于 100000 的整数，要求全部存储在数组中，去除不是质数的数字，依次输出剩余的质数。

**分析：**之前已介绍了如何判断质数，但是完全可以把判断质数这一部分给独立出来成为一个函数，这样主程序就会更加清楚明了。代码如下：

```
#include <iostream>
using namespace std;
int a[100], n;
bool is_prime(int x) {
    if (x == 0 || x == 1) return 0; //0 和 1 都不是质数，需要特判
    for (int i = 2; i * i <= x; i++) // 枚举 1 到 sqrtx，来判断 x 是否为质数
        if (x % i == 0)
            return 0;
    return 1;
    // 如果是质数，则返回 1，否则返回 0
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> a[i];
    for (int i = 0; i < n; i++)
        if (is_prime(a[i]))
            cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

程序中定义了一个 is_prime( ) 的函数，接收一个 int 类型的变量。在主程序中读入数组后，is_prime( ) 函数依次调用 a [ i ] 对应的数字。进入 is_prime( ) 函数后，调用的数字就成为了函数内的 x 变量的值。根据质数判断条件，返回 1 或者 0 代表是质数或者不是质数。回到主程序，得到了函数返回的结果，如果返回的是 1，就输出这个数字。

这里的 a 数组和 n 变量定义在了主程序外面，这样就是**全局变量**。全局变量的特点是所有的函数都可以访问这个变量（除非函数中定义了同名变量，或者函数的参数表中有这个变量名）。全局变量会自动进行初始化操作，全部都会成为 0。

**例 7-3**　闰年展示（洛谷 P5737）。输入 $x$ 和 $y$（$1582 \leqslant x < y \leqslant 3000$），输出 [ $x$, $y$ ] 区间中闰年个数，并在下一行输出所有闰年年份数字，使用空格隔开。

**分析：**虽然本例题比较简单，但是对于一个比较复杂的程序，可以分割成几个相对独立的部分，组成一个个模块，便分解问题，便于编写与调试。因此，本例也尝试这种方法，写出如下程序：

```
#include <iostream>
using namespace std;
void init(); // 定义读入部分
void doit(); // 定义处理部分
void output(); // 定义输出部分

int x, y, ans[500], cnt;

int main() {
    init(); // 读入部分
    doit(); // 处理部分
    output(); // 输出部分
    return 0;
}
void init() {
    cin >> x >> y;
}
void doit() {
    for (int i = x; i <= y; i++)
        if (!(i % 400) || !(i % 4) && i % 100) // 之前章节介绍过的闰年判断
            ans[cnt++] = i; // 等同于 ans[cnt]=i, cnt++;
}
void output() {
    cout << cnt << endl;
    for (int i = 0; i < cnt; i++)
        cout << ans[i] << " ";
    cout << endl;
}
```

本例中,将整个算法分解成读入部分、处理部分、输出部分。之前说过,函数必须先定义,后面才能使用。但是在函数使用前,只需要像例题程序一样,进行一行定义即可。程序定义需要写清楚函数返回值、函数名和参数列表,别忘了分号。void 的意思是这个函数没有返回值,只会执行一些操作。而函数的具体完整实现可以放在后面。

和例 7-2 一样,这里也使用了全局变量。全局变量是会默认初始化为 0 的。

## 7.2 变量作用域与参数传递

**例 7-4** 歌唱比赛(洛谷 P5738)。$n(n\leqslant 100)$ 名同学参加歌唱比赛,并接受 $m(m\leqslant 20)$ 名评委的评分,评分范围是 0~10 分。每名同学的得分就是这些评委给分中去掉一个最高分,去掉一个最低分,剩下 $m-2$ 个评分的平均数。请问:得分最高的同学分数是多少?评分保留 2 位小数。

**分析:**相信读者已经能够熟练地读入数组,抽出每名同学得分中的最大、最小评分,去掉后

求平均值，然后再取最大值。但是这里使用了一个 stat( )函数处理一个给定的数组评分信息，然后更新最高分的同学分数。

```
#include <cstdio>
#include <algorithm>
using namespace std;
int s[25], n, m, maxsum;
void stat(int a[], int m) {
    int maxscore = 0, minscore = 10, sum = 0;
    for (int i = 0; i < m; i++) {
        maxscore = max(a[i], maxscore);
        minscore = min(a[i], minscore);
        sum += a[i];
    }
    maxsum = max(maxsum, sum - maxscore - minscore); // 记录剩下的 n-2 评分总和
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++)
            scanf("%d", &s[j]);
        stat(s, m);
    }
    printf("%.2f", double(maxsum) / (m - 2)); // 最后再计算平均值
    return 0;
}
```

本例的 stat( )函数中，接收的参数列表中的 int a [ ]是指可以接收一个 int 类型的数组名。在函数中单个变量是可以传递实际参数的，但是传递数组名作为参数可不会把数组的所有信息当作实际参数传入函数，而只是传递数组在内存中的地址。就像在操作厨房自制机器时，可以把一个土豆、一个番茄放入机器中，但是如果有一排碗的食材，则只能告诉机器那一排碗的第一个碗在什么位置，让机器自己去指定位置找食材。因此在函数 stat( )中，数组 a [ ]其实就是全局变量 s [ ]的别名，如果在函数中改变 a [ ]中的一项值，s [ ]也会相应地改变。

和全局变量对应的，在 stat( )函数中的变量 m、maxscore、sum 等，都是在函数中定义的、在机器内部的变量，因此被称为**局部变量**。在函数中无论怎么改变这些变量，都不会影响到函数外面（函数里的 m 变量和外面的 m 没有关系了，也就是说函数外面的变量 m 被屏蔽了）。

在程序设计竞赛中，建议大数组（超过 1000）作为全局变量定义。一是方便程序所有地方都可以访问到这个数组，二是有些评测环境栈空间（就是每个函数被分配到的内存空间）有限，可能会造成栈溢出。不过现在大多数比赛的评测环境下，栈空间可以达到要求的内存上限了。

**例7-5** 交换两个变量的值。输入两个整数变量a和b,设计一个交换函数将其交换后再输出。注意:不能直接输出b和a。

**分析:** 可能会写出如下代码。

```
#include <iostream>
using namespace std;
void swap(int x, int y) {
    int t = x;
    x = y;
    y = t;
}
int main() {
    int a, b;
    cin >> a >> b;
    swap(a, b);
    cout << a << " " << b << endl;
    return 0;
}
```

经过测试,发现完全不起作用! 这是因为在swap( )函数中,x和y都是局部变量,所以怎么变都不会影响到外面了。经过改正的函数如下:

```
void swap(int &x, int &y) {
    int t = x;
    x = y;
    y = t;
}
```

在参数的变量名前面加上一个&符号,代表引用传参,相当于告诉swap( )函数a和b这两个碗放在了什么地方,而不是直接把食材丢入机器中。如果这样,x就是a的别名,y就是b的别名,修改x和y的值,就会影响到a和b。

回想一下scanf( )函数,比如scanf("%d",&n),这里加上&是一样的道理。将变量n的地址告诉给这个函数,这样才能让输入函数直到将读到的数据放入那个碗里。

**例7-6** 自动修正加强版。设计一个函数,将一个string类型的字符串中的小写字母转换为大写字母。输出原字符串和新字符串。

**分析:** 定义并实现了to_upper的函数,传入的参数是一个string字符串s,经过处理后返回它。之前介绍过了如何将字母转换成大写,这里不再赘述。代码如下:

```
#include <iostream>
#include <string>
using namespace std;
```

```
string to_upper(string s) {
    for (int i = 0; i < s.length(); i++)
        if ('a' <= s[i] && s[i] <= 'z')
            s[i] -= 'a' - 'A';
    return s;
}
int main() {
    string s1, s2;
    getline(cin, s1);
    s2 = to_upper(s1);
    cout << s1 << endl << s2 << endl;
    return 0;
}
```

有些读者会有疑惑:原来的字符串 s1 并没有变化。字符串不就是一个“加强版”的字符数组吗?为什么可以直接向函数传输实际参数而不是传递地址?这正是 string 类型的一个高明之处——将字符数组封装起来成为一个整体,传进函数就是从原来的字符串复制了一份新的字符串,所有的改动都在新的字符串上进行。而字符数组的本质还是数组,只能传递一个地址。

## 7.3 递归函数

**例 7-7**　计算阶乘(洛谷 P5739)。求 $n!$($n \leqslant 12$),也就是 $1 \times 2 \times 3 \times \cdots \times n$。不允许使用循环语句。

**分析**:计算阶乘很简单,但如果不允许使用循环语句,那么得想想别的办法。

假设 f(n) 代表 n 的阶乘,那么阶乘还有另外一种表示方法:f(1)=1;f(n)=n*f(n-1)。当想求 f(n) 时,就可以调用 f(n-1) 进而调用 f(n-2)…… 直到调用 f(1) 为止。像这样将规模大的问题转化成形式相同,但是规模更小的问题,就称为子问题。代码如下:

```
#include <iostream>
using namespace std;
int f(int x) {
    if (x == 1) return 1; // 如果 x 为 1, 则返回 1!=1
    return x * f(x - 1); /* 否则递归调用函数计算 (x-1)!, 并且将其乘上 x 返回, 从而得到 x! 的结果 */
}
int main() {
    int n;
    cin >> n;
    cout << f(n) << endl;
    return 0;
}
```

一个函数不仅可以调用其他的函数，甚至还能调用自己，这种自己调用自己被称为**递归**。因此根据题意和上述思路，可以写出上面的递归做法。假设输入是 3，那么函数执行的步骤如图 7-3 所示。

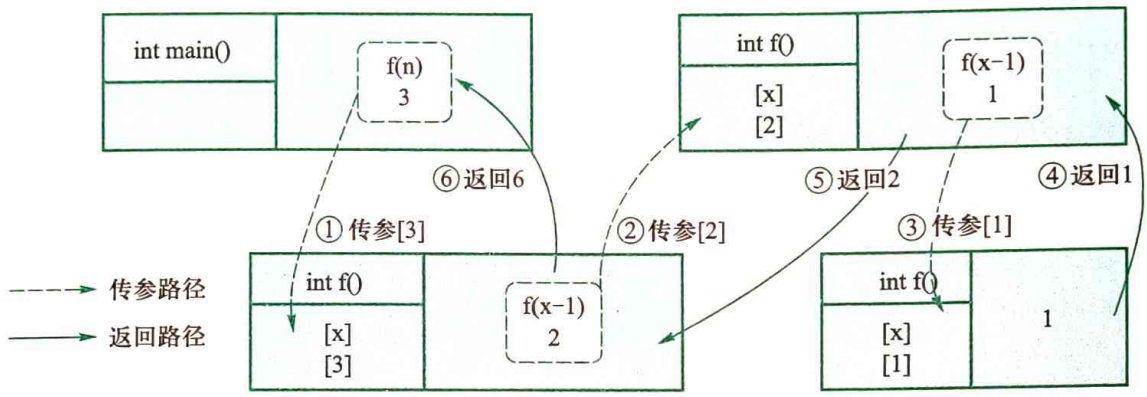


图 7-3　递归函数的步骤

程序计算步骤如下：

1) 进入 main( ) 函数，收集到了参数[ 3 ]，传递参数给 f( ) 函数，也就是调用 f(3)。

2) 进入第 1 层 f( ) 函数，需要接收一个 int 类型的变量 x。传递进来了 3，所以这里的 x=3。现在需要知道 f(3-1) 的值。因此传递参数[ 2 ]给 f( ) 函数，调用 f(2)。

3) 进入第 2 层 f( ) 函数(别忘了第一层 f( ) 函数还没结束，在等待答案呢)，这里的 x=2。现在又要知道 f(2-1) 的值，因此再次传递参数[ 1 ]给 f( ) 函数，调用 f(1)。

4) 进入第 3 层 f( ) 函数。发现当 x 等于 1 时就返回 1，返回到第 2 层 f( ) 函数，得到 f(2-1) 的值就是 1，继续进行运算。

5) 返回结果 2*1=2，返回到第 1 层 f( ) 函数，得到 f(3-2) 的值就是 2，继续进行运算。

6) 返回结果 3*2=6，返回到 main( ) 函数，得到 f(3) 的值是 6，输出这个结果。

显然，虽然递归函数可以自己调用自己，但是不能无限制调用下去，所以必须要设置递归终止条件。本例的递归终止条件是 f(1)=1。

递归函数有点类似于剥洋葱，一层套着一层，直到掰到最里层。请读者听听下面的故事，进一步了解递归。

> 从前有座山，山中有座庙，庙里有个老和尚，老和尚在给小和尚讲故事："从前有座山，山中有座庙，庙里有个老和尚，老和尚在给小和尚讲故事："从前有座山，山中有座庙，庙里有个老和尚，老和尚在给小和尚讲故事："从前有座山，山中有座庙，庙里有个老和尚，老和尚在给小和尚讲故事："太困了不讲了不讲了"，于是都回去睡觉了。"于是都回去睡觉了。"于是都回去睡觉了。"于是都回去睡觉了。

这样的故事可以抽象成：

```
void 讲故事 () {
    if (困了) return;
    讲故事 ();
    回去睡觉 ;
}
```

**例 7-8**　赦免战俘(洛谷 P5461)。现有 $2^n \times 2^n$ ($n \leqslant 10$) 名战俘站成一个正方形方阵等候国王的发落。国王决定赦免一些俘虏。他将正方形矩阵均分为 4 个更小的正方形矩阵,每个更小的矩阵的边长是原矩阵的一半。其中左上角那一个矩阵的所有战俘都将得到赦免,剩下 3 个小矩阵中,每一个矩阵继续分为 4 个更小的矩阵,然后通过同样的方式赦免战俘……直到矩阵无法再分下去为止。

给出 $n$,请输出每名战俘的命运,其中 0 代表被赦免,1 代表不被赦免。例如,当 $n$=3 时,输出如下:

```
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 1
0 0 0 0 0 1 0 1
0 0 0 0 1 1 1 1
0 0 0 1 0 0 0 1
0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1
1 1 1 1 1 1 1 1
```

**分析:**需要一个方形数组矩阵记录每名战俘是否被豁免,$2^n$ 不超过 1024,因为需要定义足够大的数组 $a$。这个数组可以定义在函数外面作为全局变量,自动初始化为 0。

每次处理一个大矩阵,将这个矩阵分成 4 部分——左上角不处理,右上角、左下角、右下角矩阵使用相同的方式处理。函数 cal($x$,$y$,$n$) 就是递归函数,$x$ 和 $y$ 表示正在处理的矩阵的左上角坐标是 ($x$,$y$);$n$ 表示这个矩阵的大小是 $2^n$,如图 7-4 所示。当 $n$=0 时说明矩阵无法再分割了,那么这个倒霉的战俘将不能被赦免,因此在 $a$ 数组的相应坐标加上标记;当 $n$ 不等于 0 时,说明矩形可以分割,就确定好剩下 3 个矩形的左上角坐标,递归继续执行下去。

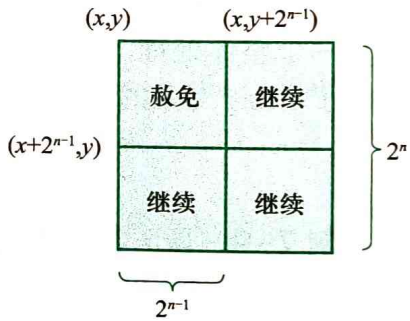

图 7-4　矩形分割

已知这个矩阵左上角坐标($x$,$y$),和矩阵的边长 $2^n$,就可以求出剩下 3 个矩阵的左上角坐标,并且进行下一层的处理。代码如下:

```
#include <cstdio>
using namespace std;
int a[1050][1050], n;
void cal(int x, int y, int n) {
    if (n == 0) a[x][y] = 1;
    else {
        cal(x + (1 << n - 1), y, n - 1); // 右上方矩阵
        cal(x, y + (1 << n - 1), n - 1); // 左下角矩阵
        cal(x + (1 << n - 1), y + (1 << n - 1), n - 1); // 右下角矩阵
    }
}
```

```
int main() {
    int n;
    scanf("%d", &n);
    cal(0, 0, n);
    for (int i = 0; i < 1 << n; i++)
        for (int j = 0; j < 1 << n; j++)
            printf("%d%c", a[i][j], j == (1 << n) - 1 ? '\n' : ' ');
    return 0;
}
```

这里有一个技巧，i<<n 的值等于 $i \times 2^n$。<< 被称为左移运算符。相应的，i>>n 的值等于 $i/2^n$（向下取整），这叫右移运算符。要注意的是 $i$ 应当是整数，而且 $n$ 不能太大以免使运算结果溢出。左移右移的优先级相当低，而且注意和输入输出流的符号 << 和 >> 区分。

## 7.4 结构体的使用

有时候要大量存储批量数据，比如说某位考生的信息，可以考虑使用数组。但是数组只能存储一组同样数据类型的信息，如果同时记录考生的姓名、成绩等不同的信息，就不能使用一个数组来存储了。

第6章介绍过 string 类型字符串，它可以存下很多信息（字符数组数据、长度等），将字符串作为一个整体处理。可以赋值，可以作为参数直接传递给函数。本节介绍的结构体，可以达到类似于 string 对象的效果。也就是说，可以将一些不同类型的信息聚合成整体，以便于处理这些信息。

**例 7-9** 最厉害的学生（洛谷 P5740）。现有 $N$（$N \leqslant 1000$）名同学参加了期末考试，并且获得了每名同学的信息：姓名（不超过 8 个字符的字符串，没有空格），语文、数学、英语成绩（均为不超过 150 的自然数）。总分最高的学生就是最厉害的，请输出最厉害学生的各项信息（姓名、各科成绩）。如果有多个总分相同的学生，输出靠前的那位。

**分析：**对每个学生的信息使用结构体存储，每次比较当前总分最大的答案和枚举到的学生的总分，如果后者更大，就把当前学生的结构体赋值给答案的结构体，代码如下：

```
#include <iostream>
#include <string>
using namespace std;
struct student {
    string name;
    int chinese, math, english;    // 定义一个结构体记录每个学生的信息
} a, ans;

int main() {
    int n;
    cin >> n;
```

```
    for (int i = 1; i <= n; i++) {
        cin >> a.name >> a.chinese >> a.math >> a.english;
        if (a.chinese + a.math + a.english > ans.chinese + ans.math + ans.
english)
            ans = a; // 比较两个结构体的大小，如果这个更大，就用这个来更新答案
    }
    cout << ans.name<<" "<<ans.chinese<<" "<<ans.math<<" "<<ans.english<<
endl;
    return 0;
}
```

C++ 的**结构体**是由一系列具有相同类型或不同类型的数据构成的数据集合。比如说，一名学生有姓名信息（字符串），有成绩信息（整数）。结构体定义的一般形式如下：

```
struct 类型名 {
    数据类型 1 成员变量 1;
    数据类型 2 成员变量 2;
} [结构体变量名];

struct 已经定义过的类型名 结构体变量名;
//"struct" 也可以不加
```

本例中定义了一个名字是 student 的结构体类型，里面包括 name、chinese、math、english 这几个成员变量，其中姓名为 string 字符串，而成绩是整数。然后定义了 a 和 ans 这两个变量，类型是 student。

变量 a 作为一个学生，可以用 a.name 来表示 a 同学的姓名，a.chinese 代表他的语文成绩。这些成员变量可以像普通的变量那样读入、赋值或者参与表达式运算。而相同类型的结构体变量之间也可以直接进行赋值运算

由于需要找到总分最高的，设置 ans 作为擂主。不断进行打擂比较，当总分更高时，就将 a 赋值给 ans，最后输出 ans 的姓名和各项成绩。

**例 7-10**　旗鼓相当的对手加强版（洛谷 P5741）。现有 $N(N\leqslant 1000)$ 名同学参加了期末考试，并且获得了每名同学的信息：姓名（不超过 8 个字符的字符串，没有空格），语文、数学、英语成绩（均为不超过 150 的自然数）。如果某对学生 <i,j> 的每一科成绩的分差都不大于 5，且总分分差不大于 10，那么这对学生就是“旗鼓相当的对手”。现在想知道这些同学中，哪些是“旗鼓相当的对手”，请输出他们的姓名。

**分析：**之前使用多维数组实现过类似的题目，但这回由于还需要记录学生的名字，所以要使用结构体。结构体也能批量定义，就跟数组一样，用于存储大量学生的信息。

```
#include <iostream>
#include <string>
#define MAXN 1024
using namespace std;
```

```
int n, x, ans;
struct student {
    string name;
    int chinese, math, english;    //定义一个结构体记录每个学生的信息
};
struct student a[MAXN];
bool check(int x, int y, int z) { //检查两个数x, y的差是否不超过z
    return x <= y + z && y <= x + z;
}
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i].name >> a[i].chinese >> a[i].math >> a[i].english;
    for (int i = 1; i <= n; i++) //枚举第一个学生i
        for (int j = i + 1; j <= n; j++) //枚举第二个学生j
            if (check(a[i].chinese, a[j].chinese, 5)
                && check(a[i].math, a[j].math, 5)
                && check(a[i].english, a[j].english, 5)
                && check(a[i].chinese + a[i].math + a[i].english,
                   a[j].chinese + a[j].math + a[j].english, 10)
              ){
                cout << a[i].name << " " << a[j].name << endl;
            }
    return 0;
}
```

这里定义了一个函数来判断两个数的差是否小于一个值,简化程序。事实上,结构体也能作为函数的参数或者返回值,请读者自己尝试实验。

**例7-11** (选读)评等级(洛谷P5742)。现有$N$($N \leqslant 1000$)名同学,每名同学需要设计一个结构体记录以下信息:学号(不超过100000的正整数)、学业成绩和素质拓展成绩(分别是0~100的整数)、综合分数(实数)。每行读入同学的学号、学业成绩和素质拓展成绩,并且计算综合分数(分别按照70%和30%的权重累加),存入结构体中。还需要在结构体中定义一个成员函数,返回该结构体对象的学业成绩和素质拓展成绩的总分。

然后需要设计一个函数,其参数是一个学生结构体对象,判断该学生是否“优秀”。优秀的定义是学业和素质拓展成绩总分大于140分,且综合分数不小于80分。

**分析:**在建立一个新的结构体对象时,可以在结构体中定义和结构体名称相同的**构造函数**(本例中结构体名称为student,在结构体定义中实现名为student的函数,不必加上函数返回类型)。在定义一个新的结构体并进行初始化时,给结构体定义一些变量,结构体对象调用构造函数,对这些变量进行处理。本例中,除了定义了一个接收3个变量的初始化构造函数,还定义了一个不接收任何变量的空的构造函数student( ){ },否则直接定义结构体对象时(如student one_student; )就会编译失败。

还可以在结构体中定义成员函数(如sum( )函数),用于处理结构体对象的内部事务。构造

函数是一种特殊的成员函数。在结构体内访问自己对象结构体的成员变量或者成员函数时，在对应的变量名或函数名前加 this->，但是如果不产生歧义，不加也可以。在结构体外调用结构体成员函数时只需要“结构体变量名 . 成员变量名”即可，例如本例的 one_student.sum( )。

不仅如此，结构体对象还能像单个变量一样，作为参数直接传参进入函数，而不是和数组一样只能传入一个地址。这样的话在函数中修改直接传入的结构体对象中的成员变量，也不会影响到函数外面，除非加上 & 符号变成引用传参。代码如下：

```
#include <iostream>
#include <string>
#define MAXN 1024
using namespace std;
int n, x, ans;
struct student {
    int id;
    int academic, quality;
    double overall;
    student(int _id, int _ac, int _qu){ // 初始化构造函数
        this->id = _id;
        this->academic = _ac;
        this->quality = _qu;
        this->overall = 0.7 * _ac + 0.3 * _qu;
        // 这里的 this-> 也可以省略
    }
    student() {} // 没有传递参数的初始化构造函数
    int sum(){
        return academic + quality;
        // 返回值也可写为 this->academic+this->quality;
    }
};

int is_excellent(student s) {
    // 访问成员变量与调用成员函数
    return s.overall >= 80 && s.sum() > 140;
}

int main() {
    cin >> n;
    for (int i = 1; i <= n; i++){
        int tmp_id, tmp_ac, tmp_qu;
        cin >> tmp_id >> tmp_ac >> tmp_qu;
        student one_student(tmp_id, tmp_ac, tmp_qu); // 结构体初始化
        if(is_excellent(one_student)) // 结构体变量作为参数传递
            cout << "Excellent" << endl;
        else
```

```
            cout << "Not excellent" << endl;
        }
        return 0;
    }
```

需要注意的是，本例题中的 one_student 这个结构体变量是在 for 循环里面定义的局部变量，循环结束后该变量即销毁。如果需要留存做后期处理，则需要像例 7-10 一样在循环体外面定义结构体数组以留存数据。

成员函数（除了构造函数以外）在算法竞赛中使用较少，因此本例仅作为了解。结构体还有很多复杂的特性，例如公开或私有访问权限，这里不进行介绍，感兴趣的读者可以自行查阅相关资料。

至此，已经介绍完了入门部分的全部内容。虽然只介绍了 C++ 的基本内容，但是已经足以应对大多数的算法竞赛所要求的语言知识了。C++ 是一种非常复杂的编程语言，可能在接下来的学习过程中会继续补充新的知识点。

编程绝不是靠读书或听课就能搞懂的，必须要亲自上机实践。章末的课后习题与实验是非常重要的部分，请尽力完成。如果有条件，可尽可能多地去完成洛谷中“入门难度”的题目。

## 7.5 课后习题与实验

**习题 7-1** 设计以下的函数：

(1) 判断某个整数是不是完全平方数。

(2) 给出三维空间的 2 对点，6 个变量，求出这两个点之间的距离。

(3) 给出圆柱的半径和高，求出这个圆柱的体积。

(4) 传入一个字符串，将这个字符串的所有空格去掉后返回处理好的字符串。

(5) 给出一位 int 类型整数，计算它的每位数字的和。

(6) 寻找指定数组中的平均数。

**习题 7-2** 观察以下程序，哪些变量是局部变量？哪些是全局变量？如果输入数据是“1 2”时，应该输出什么呢？

```
#include <iostream>
#include <string>
using namespace std;
const int MAXN = 10000;
int a, b, c, array[MAXN];
bool check(int x, int y, int z) {
    return x <= y + z && y <= x + z;
}
void add(int a, int b) {
    c = a + b;
}
int multi() {
    return a * b;
```

```
}
int main() {
    int c = 3;
    cin >> a >> b;
    add(c, a);
    b = multi();
    if (check(a, b, c))
        cout << a;
    else
        cout << c;
    return 0;
}
```

**习题 7-3**　质因数分解(洛谷 P1075,NOIP2012 普及组)。已知正整数 $n$($n \leqslant 2 \times 10^9$)是两个不同的质数的乘积,试求出两者中较大的那个质数。

**习题 7-4**　哥德巴赫猜想(洛谷 P1304)。输入一个偶数 $N$($N \leqslant 10000$),验证 4~$N$ 的所有偶数是否符合哥德巴赫猜想:任一大于 2 的偶数都可写成两个质数之和。如果一个数不止一种分法,则输出第一个加数相比其他分法最小的方案。例如输入 10,因为 10=3+7=5+5,因此 10=5+5 是错误答案。

**习题 7-5**　回文质数(洛谷 P1217,USACO Training)。因为 151 既是一个质数又是一个回文数(从左到右和从右到左看是一样的),所以 151 是回文质数。写一个程序来找出范围为[ $a$,$b$ ]($5 \leqslant a < b \leqslant 100000000$)(1 亿)的所有回文质数。

**习题 7-6**　集合求和(洛谷 P2415)。给定一个集合 $S$($|S| \leqslant 30$)(即集合元素数量不超过 30),求出此集合所有子集元素之和。例如,当集合是 {2,3} 时,子集包括[ ][ 2 ][ 3 ][ 2,3 ],子集元素的和是 2+3+(2+3)=10。

**习题 7-7**　利用递归函数求解:

(1) 不使用循环和数组,输入一串整数,然后将整数倒序输出。假设给出的整数串以 0 结尾。

(2) 使用辗转相除法,求出两个给定的整数的最大公约数。

(3) 求出斐波那契数列 fib($n$) 的值,$n \leqslant 10$。更进一步,如果 $n$=20,递归函数可能会很慢,因为进行了很多重复计算同一个 fib($n$) 的值,有什么办法改进效率吗?

**习题 7-8**　猴子吃桃(洛谷 P5743)。一只小猴买了若干桃子。第一天他刚好吃了这些桃子的一半,又贪嘴多吃了一个;接下来的每一天它都会吃剩余的桃子的一半外加一个。第 $n$($n \leqslant 20$)天早上起来一看,只剩下 1 个桃子了。请问:小猴买了几个桃子?

**提示:**虽然可以使用逆推求解,但也可以从第一天编写递归函数顺着求解,思维更加直接。

**习题 7-9**　培训(洛谷 P5744)。某培训机构的学员有如下信息:

1) 姓名(字符串);

2) 年龄(周岁,整数);

3) 去年 NOIP 成绩(整数,且保证是 5 的倍数)。

经过为期一年的培训,所有同学的成绩都有所提高,提升了 20%(当然 NOIP 满分是 600 分,不能超过这个得分)。

输入学员信息,请设计一个结构体存储这些学生信息,并设计一个函数模拟培训过程,其参

数是这样的结构体类型，返回同样的结构体类型，并输出学员信息。

例如输入：

```
3
kkksc03 24 0
chen_zhe 14 400
nzhtl1477 18 590
```

应当输出：

```
kkksc03 25 0
chen_zhe 15 480
nzhtl1477 19 600
```