期中复习串讲

flags
width
precision
length

写在最前面：

把编译选项里的显示所有警告（-Wall）打开！！
本地新建代码文件保存的时候改文件类型为.c！！
代码编写完成后一定要保存好，改完代码运行前先编译一下！！
代码中的逗号、分号、方括号等一定要使用半角符号！！
多看看之前的答疑要点文档！！

变量的定义、类型和范围

标识符	名称	范围	常量后缀	所占空间	输入输出标识符
`int`	有符号整型	$-2^{31} \sim 2^{31}-1$，上下限数量级为10^9		32位（4字节）	`%d`
`long long int`（或写为`long long`）	有符号长整型	$-2^{63} \sim 2^{63}-1$，上下限数量级为10^{18}	`LL`	64位（8字节）	`%lld`
`unsigned int`	无符号整型	$0 \sim 2^{32}-1$	`U`	32位（4字节）	`%u`
`unsigned long long int`	无符号长整型	$0 \sim 2^{64}-1$	`ULL`	64位（8字节）	`%llu`
`char`	字符型	有效 $ASCII$ 字符对应的值范围为 $0 \sim 127$		1字节	`%c`
`double`	双精度浮点型	$-1.7 \times 10^{-308} \sim 1.7 \times 10^{308}$		8字节	输入是`%lf`，输出是`%f`

其中`char`类型比较特殊，标准并没有规定`char`是否有符号，有无符号取决于编译器的具体实现。当有符号时范围为 $-128 \sim 127$，无符号时范围为 $0 \sim 255$

表格中，“常量后缀”的意思是，如果你写了一个常量，而且希望这个常量是对应的数据类型，那么需要添加相应的后缀。如：

```
long long int a=1LL;//声明常量1的类型是long long int
```

特别的，如果想要声明为浮点数，一般在后面加 `.0` 即可，例如：

```
double a=1.0/3.0;//1.0、3.0是浮点数
```

一切变量，除非定义完了以后立刻进行输入，否则必须赋予初始值，严禁变量不被赋初值而被使用。例如：

```
int sum;//sum未赋初值
for(int i=1;i<=n;++i)
    sum+=a[i];
```

```
int j(int a, int b, int c) {
    int p;//p未赋初值
    if ((a == b) && (a == c)) {
        p = 1;
    } else if ((a == b) || (a == c) || (c == b)) {
        p = 2;
    }
    return p;
}
```

关于浮点误差

浮点误差是困扰很多同学的问题。关于浮点误差，有如下的忠告：

1. **不要**直接用 `==` 判断两个浮点数是否相等

   这一条也包括：不要用 `>=` 判断大于等于，不要用 `<=` 判断小于等于，不要用 `!=` 判断不等于。

   正确的判断浮点数相等的做法是：判断两数之差的**绝对值**（即两数之间的距离）是否小于一个预先定义的非常小的浮点数值 `eps`，`eps` 具体值是多少取决于题目要求，常用的为 `1e-10`（求浮点数绝对值：使用 `math.h` 中的 `fabs` 函数）

   示例代码：

   ```
   double a,b;
   double eps=1e-10;
   scanf("%lf%lf",&a,&b);  //将浮点值读入a,b，或计算a,b值的操作，再次提醒不要未赋值就使用变量
   if(a==b){//错误
       ;
   }
   if(fabs(a-b)<eps){//正确
      ;
   }
   ```

2. 优化计算步骤

   相信大家都做了《E5-F 哪吒的水题》，其中的 `s` 函数有如下两种写法：

```
double s(double mid) {
    double ans, one, tw;
    one = s1 * sqrt(x1 * x1 - 2 * mid * x1 + mid * mid + Y1 * Y1);
    tw = s2 * sqrt(x2 * x2 - 2 * mid * x2 + mid * mid + y2 * y2);
    ans = one + tw;
    return ans;
}
```

和

```
double s(double x) {
    double ans, sum1, sum2;
    sum1 = s1 * sqrt(Y1 * Y1 + (x1 - x) * (x1 - x));
    sum2 = s2 * sqrt(y2 * y2 + (x2 - x) * (x2 - x));
    ans = sum1 + sum2;
    return ans;
}
```

从数学上来说自然是完全一样的，但是事实上第一个就是`Wrong Answer`，第二个才是`Accepted`。因为如果 x 和 x_1 都较大的话，$x^2+x_1^2$ 与$2\cdot x\cdot x_1$就会很大，你把这两个很大的东西相减，误差就会很大。如果不信的话，试试`(1e23)-(1e23+1)`，你会信的。

3. 尽量别用浮点数

   世间花草千千万，不止浮点这一朵。能用整数的，就用整数吧。例如，如果题目让你判断 a、b 为正整数时的$\frac{a}{b}\leq\frac{3}{5}$，那么为何不直接写`if(a*5<=b*3)`呢？当然，这么写需要考虑两边表达式计算发生整型溢出的可能性，应当选择合适的整型类型。

关于常用常数值

对于某些数学常数，除非题目有明确说明"取圆周率 π 值为`3.14`"、"取自然对数的底数 e 值为`2.71828`"、"取 $\sqrt{2}$ 值为`1.414`"等要求，否则不要在题目中先入为主指定这些值，手输这些数一来容易记错，二来位数十分有限，容易对计算结果精度带来较大影响

当题目没有给出其他计算公式时，计算和存储圆周率 π 值可以使用如下代码：

```
double pi = acos(-1);   //使用反余弦函数arccos计算并存储圆周率pi值，比手输3.14精度高
```

要计算自然对数的底数 e 的 x 次幂，可以直接使用`math.h`库中的`exp`函数：`exp(x)`

当题目没有给出其他计算公式时，计算和存储自然对数的底数 e 可以使用如下代码：

```
double e = exp(1);  //e的1次方就是e的值
```

`2e8`、`3e-6`这样的写法是科学计数法，是浮点数字面值，不是整型，不能直接赋给整型变量，也不要在声明数组时使用科学计数法的字面值作为数组大小

输入输出

基本输入

当程序中只涉及数字（整数、实数）的输入时，格式控制字符串中无需加入任何空格、换行等空白字符，只需按顺序排列类型占位符即可。

```
scanf("%d%d%d",&a,&b,&c);//推荐
scanf("%d %d %d",&a,&b,&c);//不推荐
```

数字与字符混合读入

使用`scanf`读入整型和浮点型能够忽略空白字符，而使用`scanf("%c",&n)`或`n=getchar()`这两种基本的字符型读入方法不能忽略空白字符，因此当需要**混合使用整型/浮点型与字符型读入**时，一定要确保每行输入行末的换行符`'\n'`，以及夹杂在输入数字和字符之间的空格被妥善地处理，不会被误读到变量中产生错误。

处理方法可以选择多使用一个`getchar()`，将输入内容中在要读入的有用字符前残留的换行符读入并丢弃，也可以选择使用字符型的特殊读入控制，即在`%c`前面加一个空格，写作：

```
scanf(" %c",&c);
```

这个空格可以“吃掉”要输入的字符之前所有的空白字符。

出处：C2-B 这里是BUAA

长度控制

有时，我们需要控制输入的数字的长度。比如，我们知道这个数字只有`1`位。这时，可以这样写：

```
scanf("%1d",&a);
```

这种操作常见于读入`01`串时。详情请看：E3-B 补码解译

常见输出

- 基本输出：`printf("%d",a)`、`printf("%c",c)`等
- 输出实数保留 n 位小数：`printf("%.nf",a);` 这里格式控制字符串里面的n需要你自己填上
- 输出整数，占 n 位右对齐，前面补充零（常见于输出月份、日期）：`printf("%0nd",a);`
- 输出特殊符号时，需要转义

表达式和运算

表达式，是由变量、常量和运算符组合而成的式子。其中运算符有算术运算、关系运算、逻辑运算和位运算等类型。

算术运算

基本操作：加减乘除

变量和数字之间进行算术运算时，结果为参与运算的变量中类型容纳范围更大的那种类型，例如：

1. `int`和`int`加减乘除，结果类型为`int`
2. `int`和`long long`加减乘除，结果类型为`long long`
3. `int`和`long long`和`double`加减乘除，结果类型为`double`

注意：

1. 由第 1 条可知，两个`int`类型的变量作加、减、乘法，结果仍然是`int`类型，如果结果超出了`int`能表示的范围，就会立即发生溢出错误。因此，当判断出运算结果可能会超出`int`所能表示的范围时，一定要先将其中至少一个运算数强制类型转换为`long long`，然后再执行运算，以防溢出，此时运算的结果如果需要保存，也要存入`long long`类型的变量中
2. 上面两种整型范围内的运算，结果类型都为整型，对于除法而言就是**整除**，而不是实数除法。因此，如果想要从整型值出发实现实数除法，必须先将其中至少一个运算数强制类型转换为`double`（也可以通过给运算数乘`1.0`的方式实现，原因为上面运算规则第 3 条），然后再执行除法运算
3. 浮点数存储和运算都存在误差，因此，当需要使用浮点数运算时，最好先考虑一下能否写成等价的整型运算。

> **小技巧：不同字符 $ASCII$ 值之差的应用**
>
> 我们知道：在 $ASCII$ 码表中，数字字符`'0'~'9'`的 $ASCII$ 值是连续递增的，小写英文字母`'a'~'z'`的 $ASCII$ 值也是连续递增的，大写字母`'A'~'Z'`也一样。因此，在每类字符中，可以通过计算一个字符的 $ASCII$ 值与该类中第一个字符的 $ASCII$ 值之差，得知这个字符是这类字符中的第几个。反过来也类似，可以将某类字符中第一个的 $ASCII$ 值加上一个偏移值 x，得到这类字符中后面的字符
>
> 常见的应用场景是**进制转换**与**英文字符大小写转换**，灵活使用可以避免写出过多繁杂的分支判断语句，方便清晰，简洁高效
>
> 出处：E2-B 这里是BUAA 2

关系运算

大于（`>`）、小于（`<`）、大于等于（`>=`）、小于等于（`<=`）、判断相等（`==`）、判断不相等（`!=`）

满足关系时，关系运算表达式结果为`1`，不满足关系时，结果为`0`，类型均为`int`。当然了，关系运算的表达式也可以参与`int`的加减乘除运算。

需要注意的易错点

1. 判断相等使用两个等号`==`，一个等号`=`起到的是赋值作用，将右面的值赋给左面的变量
2. 所有的关系运算符都是双目运算符，只能连接两个表达式，不能连接三个或以上。例如，不能写

```
if(a<=b<=c)//错误写法
if(a==b==c)
```

应该写

```
if(a<=b && b<=c)//正确写法
if(a==b && b==c)
```

逻辑运算

与（`a && b`）、或（`a || b`）、非（`!a`），这里`a`、`b`通常为关系运算语句，且可以嵌套多层。

逻辑运算的运算结果是`int`类型，当满足条件时，其值为`1`，不满足条件时，其值为`0`。

逻辑运算性质：短路求值（参考课件《P2-C语言编程基础框架framework》67 页）

运算符优先级

不必硬记，加括号更方便直观

参考课件《P2-C语言编程基础框架framework》79 页

分支结构

`if`语句

`if`语句的基本结构如下：

```
if(条件1){
    //代码1
}else if(条件2){
  //代码2
}else if(条件3){
    //代码3
}else{
    //代码else
}
```

其中，条件1、条件2、条件3...都是一个表达式。

程序首先检查条件1，如果满足，则执行代码1，否则继续检查条件2，如果满足，则执行代码2，否则继续检查条件3.....如果所有条件都不满足，执行代码else。

大家经常接触“真”、“假”的概念，而有些同学不知道其实质。事实上，“假”意味着该表达式的值为零；而“真”则意味着该表达式的值不为零。请注意，这里唯一的区别只是表达式是否为零：只要表达式的值不为零，比如`1, -1, 0.01, 'a', EOF, NAN, '\n', 空格`等等千奇百怪的东西，它们都是“真”，放到`if`后面的括号里，都会触发`if`分支。

`switch-case`语句

基本结构如下：

```
switch (表达式) {
    case 值1:
        //代码1
        break;
    case 值2:
        //代码2
        break;
    ...
    default:
        //代码default
        break;
}
```

程序会检查表达式的值是否是值1、值2...，如果是，执行对应的代码；如果不是，执行代码default。

关于`switch-case`结构的其它规范，请参考《中华人民共和国国家军用标准 GJB 8114-2013 C/C+ +语言编程安全子集》：

42	R−1−4−3	禁止使用空 switch 语句。
43	R−1−4−4	禁止对 bool 量使用 switch 语句。
44	R−1−4−5	禁止 switch 语句中只包含 default 语句。
45	R−1−4−6	除枚举类型列举完全外，switch 必须要有 default。
46	R−1−4−7	switch 中的 case 和 default 必须以 break 或 return 终止，共用 case 必须加以明确注释。
47	R−1−4−8	switch 语句的所有分支必须具有相同的层次范围。

循环结构

`for`语句的三个表达式

课件上说：

```
for(表达式1;表达式2;表达式3){
    statement
}
```

和

```
表达式1;
while ( 表达式2 )
{
    statement
    表达式3;
}
```

等价。因为表达式3是单独出现的，并没有用到它的值，所以这里面写`++i`和`i++`是一样的，写`--i`和`i--`是一样的。

其它注意事项，请参考《中华人民共和国国家军用标准 GJB 8114-2013 C/C+ +语言编程安全子集》：

90	R−1−9−1	for 循环控制变量必须使用局部变量。
91	R−1−9−2	for 循环控制变量必须使用整数型变量。
92	R−1−9−3	禁止在 for 循环体内部修改循环控制变量。

多组数据读入问题

经常出现多组数据读入的问题。接下来，我选取三个典型例子，看看多组数据读入具体怎么处理。

在题面中，经常出现像这样的描述：

```
本题为多组输入输出，第一行输入一个正整数 n ，表示总共有 n 组数据。
```

这时，你需要采取这样的代码进行编写：

```
int main(){
    int n;
    scanf("%d",&n);
    while(n--){
        //清零或定义[所有]在运算中用到的变量

        //输入数据
        //进行运算
        //输出数据，记得最后加换行符'\n'
    }
    return 0;
}
```

如果题目只说了多组输入输出，而没有给出具体的组数，例如：

```
输入若干个整数，对每个输入的整数，计算.....
```

这时，你需要采取这样的代码进行编写：

```
int main(){
    int x;
    while(scanf("%d",&x)!=EOF){
        //清零或定义[所有]在运算中用到的变量

        //输入数据
        //进行运算
        //输出数据，记得最后加换行符'\n'
    }
    return 0;
}
```

如果题目是输入若干个数据，并以一个特定的字符结尾，例如：

> 输入若干个整数 k_i，以空格分隔，表示第 i 个学生的成绩。最后输入 `-1`，表示输入结束。

这时，你应该这样写：

```
int main() {//请注意本例中的数组处理方式
    int a[MAXN];
    int n = 1;
    while (scanf("%d", &a[n]) != EOF) {
        if (a[n] == -1) {
            n--;//删除数组最后一个-1
            break;
        }
        n++;
    }
    //此时，n是数组a[]的有效数据的数量，要访问这个数组，可以这样写：
    for (int i = 1; i <= n; ++i) {
        //访问a[i]
    }
    return 0;
}
```

常见的数组操作

在以下的代码中，默认数组的数据类型为`int`，下标从1开始，长度为n。如果你更喜欢写下标从0开始的，或者要用到其它数据类型，请自行修改代码。

数组的定义和初始化

```
数据类型 数组名[数组长度];
```

此时数组的合法下标的范围是0~数组长度-1。如果你和我一样喜欢下标从1开始，那么定义时需要给数组长度加一。

例如：

```
int a[101];
```

请注意：此处的数组长度必须在程序开始运行之前确定，非常非常建议直接写一个常数。例如，如果题目里面说了"数组的长度$n \leq 10^5$"，那你就直接把这个上界写进去，也就是：

```
int a[100001];
```

当然，如果你看了一些高手的代码，里面会这样写：

```
#define maxN 100001
```

这就是用宏定义的方法定义了数组的长度，如果一个题目里需要定义很多数组，那么就可以这样写了：

```
int a[maxN],b[maxN],c[maxN];
```

在定义数组时，往往需要初始化，即给它赋予初始值，例如：

```
int a[100001]={};//初始化为零
int a[100001]={1,2,3,4};//下标为0~3的元素的值为1，2，3，4，其余为零
int a[]={1,2,3,4,5};//定义了一个长度为5（下标范围是0~4）的数组，此时可以不写长度
```

在定义数组时，如果数组的长度超过了十万，请使用全局方式定义。

有些人说，全局数组一定初始化为零，这是不严谨的。具体而言，这和你程序运行的操作系统有关。强烈建议大家即使定义的是全局数组，也要进行初始化。

在定义较大的数组时，请注意内存开销。例如，一个`int`变量是`32`位，占`4B`。如果你开了一个长度为一亿的`int`数组，那么它就需要近100GB的内存存储。

输入一个数组

```
for(int i=1;i<=n;++i){
    scanf("%d",&a[i]);
}
```

输出一个数组

```
for(int i=1;i<=n;++i){
    printf("%d ",a[i]);
}
```

求数组最大值

```
int Max=a[1];    //待存找到的最大值
//要给Max一个必定不大于数组所有元素中最大值的初始值，保证在遇到数组中真正的最大值时，可以更新Max的值
//可以如上设置Max的初始值为a[1]，这时理解为“假设数组的第一个元素是最大值”
//然后通过遍历数组的所有元素，用后面可能出现的更大的值更新Max的值，如果遍历数组后没有执行更新，说明a[1]即为最大
//如果数组中的所有元素都是自然数，那么可以给Max赋值为-1
//如果数组中所有元素都是int范围内的，可以给Max赋值为-2147483648(即INT_MIN)
int MaxCur=1;    //待存最大值在数组中的下标
for(int i=2;i<=n;++i){  //遍历数组后面的位置
    if(Max<a[i]){
        //怎么记？很简单：“当出现异常情况时，说明最大值需要更新”
        //在这里，“有元素的值比最大值还大”就是所谓的“异常情况”
        Max=a[i];
        Maxcur=i;
    }
}
//此时，Max是数组a中最大值的数值，Maxcur是数组a中第一个最大值的下标。
//如果要获取最后一个最大值的下标，需要将if(Max<a[i])改为if(Max<=a[i])
```

求数组最小值

```
int Min=a[1];    //待存找到的最小值
//要给Min一个必定不小于数组所有元素中最小值的初始值，保证在遇到数组中真正的最小值时，可以更新Min的值
//可以如上设置Min的初始值为a[1]，这时理解为“假设数组的第一个元素是最小值”
//然后通过遍历数组的所有元素，用后面可能出现的更小的值更新Min的值，如果遍历数组后没有执行更新，说明a[1]即为最小
int MinCur=1;    //待存最小值在数组中的下标
for(int i=2;i<=n;++i){  //遍历数组后面的位置
    if(Min>a[i]){
        Min=a[i];
        Mincur=i;
    }
}
//此时，Min是数组a中最小值的数值，Mincur是数组a中第一个最小值的下标。
//如果要获取最后一个最小值的下标，需要将if(Min>a[i])改为if(Min>=a[i])
```

判断两个数组是否完全相同

```
int flag=1;
for(int i=1;i<=n;++i){
    if(a[i]!=b[i]){
        flag=0;
        break;
    }
}
if(flag==1){
    //数组a和b完全相同
} else {
    //数组a和b不完全相同
}
```

恢复一个数组

在多组输入输出的问题中，如果用到了数组，那么一定要记得每组数据开始时需要将其恢复为初始值。有两种写法，第一种是用循环：

```
for(int i=1;i<=n;++i){
    a[i]=0;
}
```

第二种是用定义在`string.h`里的`memset`函数：

```
memset(a,0,sizeof(a));
```

【特别注意】：在现阶段，`memset`只能将数组置为0，第二个参数只能写`0`。如果你要用到其它的初始值，例如`-1`，请用循环法。

访问时跳过数组中的某些元素

在E2-F Wings咖啡中，有这样的一句提示

> 本题可以参考思路：如何找到 n 个自然数里最大的 m 个呢？循环 m 次，每次找到最大值，将数组最大值的位置进行标记，在下次循环中跳过这个位置。

如何"将数组中某个位置进行标记，在下次循环中跳过这个位置"呢？有两种写法。在这道题中，可以这样写：

```
//假设需要标记位置x
a[x]=0;
```

因为在正常情况下，a中的元素的不可能为零的，所以在下次循环时，只要遇到0，那就跳过（continue），即：

```
for (int i = 1; i <= n; ++i) {
    if (a[i] == 0) {
        continue;
    } else {
        //继续运算
    }
}
```

但是这种方法会破坏数组原有的元素。如果不想破坏，可以另外定义一个标记数组：

```
int flag[100001]={0};//如果flag[i]==1,那么说明位置i需要跳过
```

标记：

```
//假设需要标记位置x
flag[x]=1;
```

下次循环：

```
for (int i = 1; i <= n; ++i) {
    if (flag[i] == 1) {
        continue;
    } else {
        //继续运算
    }
}
```

位运算

在本文档讲解中，记变量最右面的比特位为最低位第 `0` 位，左面 1 位为第 `1` 位，依此类推，最左面的比特位为最高位

运算特点

直接对参与运算的值的二进制比特位进行操作，有多个操作数时，比特位按位置对应参与运算

需要考虑优先级问题，但不必强记，**加括号**

基本操作

与运算（`a & b`）：需要两个运算数，参与运算的两个比特位都是 `1` 时结果才是 `1`，只要其中有一个为 `0` 结果就是 `0`，优先级低于比较运算

或运算（`a | b`）：需要两个运算数，参与运算的两个比特位都是 `0` 时结果才是 `0`，只要其中有一个为 `1` 结果就是 `1`，优先级低于比较运算

按位取反运算（`~a`）：需要一个运算数，将该数的各比特位取反，`1` 变 `0`，`0` 变 `1`

异或运算（`a ^ b`）：需要两个运算数，参与运算的两个比特位相同时结果是 `0`，不同时结果是 `1`，优先级低于比较运算

异或性质：交换律，结合律，相同两数异或结果为 `0`（进一步推广：偶数个相同的数异或结果为 `0`，奇数个相同的数异或结果为原数）

出处：E3-G 某咸鱼与中秋节

位移运算（`a << b`，`a >> b`）：需要两个运算数。对于左移，将 `a` 的二进制比特位左移 `b` 位，右面填 `0`；对于右移，将 `a` 的二进制比特位右移 `b` 位，当 `a` 是无符号数时，左面填的比特是 `0`（逻辑右移），当 `a` 是有符号数时，左面填的比特是原数的最高位比特值（算术右移）

对于一个 32 比特的数，左移超过 31 位是**未定义行为**！不要将一个数的所有比特都左移出这个数的变量空间外面

一些常用操作

将变量`a`的第`i`位置为`0`

使用与运算`&`，对`a`的某一比特位`bit`，`bit & 1`结果为`bit`原值，`bit & 0`结果为`0`（“与`1`不变，与`0`为`0`”）

```
a = a & (~(1 << i));
```

将变量`a`的第`i`位置为`1`

使用或运算`|`，对`a`的某一比特位`bit`，`bit | 1`结果为`1`，`bit | 0`结果为`bit`原值（“或`1`为`1`，或`0`不变”）

```
a = a | (1 << i);
```

将变量`a`的第`i`位取反

使用异或运算`^`，对`a`的某一比特位`bit`，`bit ^ 1`结果为`~bit`（`bit`取反），`bit ^ 0`结果为`bit`原值（异或`1`取反，异或`0`不变）

```
a = a ^ (1 << i);
```

以上操作如果需要对多个位置执行，可以与循环结合使用

取出变量`a`的指定位置的比特

取出`unsigned int`类型变量`a`中值的第`i`位，其他位置为`0`，将结果存入`unsigned int`类型变量`b`

```
unsigned int b;
b = a & (1 << i);
```

如果需要取出一个区间范围内的比特，可以对这个区间范围内的每一位比特分别执行上述与运算`&`，得到多个中间结果，然后使用或运算`|`将这些中间结果合并到一个最终结果中。也可以先设置一个掩码变量`mask`，让它在要取出的比特位置上的值为`1`，然后让`a`与`mask`执行与运算`&`

如：取出`unsigned int`类型变量`a`中值的最高`4`位，其他位置为`0`，将结果存入`unsigned int`类型变量`b`

```
unsigned int b,mask = 0xf0000000;   //0xf0000000为十六进制数，对应32位二进制中最高4位为1，其余为0的情况
b = a & mask;
```

在编写位运算代码时，如果实在很难想象各比特位发生的变化，可以在纸上手写模拟变量中的各比特位变化，数数确定要左移或者右移多少位，要置`0`还是置`1`

函数

基本写法

函数的定义

```
#include <stdio.h>

int function(int a,int b,int c);

int main(){

}

int function(int a,int b,int c){//上下两部分参数表必须完全相同
    int ans=0;
    //运算
    return ans;
}
```

函数的调用

```
#include <stdio.h>

int function(int a,int b,int c);

int main(){
    int x,a,b,c;
    //这样是正确的：
    x=function(a,b,c);
    //不要写成这样：
    x=int function(int a,int b,int c);
}

int function(int a,int b,int c){
    int ans=0;
    //运算
    return ans;
}
```

常见函数

判断一个数是否是素数

```
int isprime(int x) {
    /*
    @isprime:判断素数
    @输入：
      x：int类型，被判断的数
    @返回：
      int类型，当x是素数时返回1，否则返回0
    */
    if (x == 1)
        return 0;
    else if(x == 2)
```

```
        return 1;
    else if(x % 2 == 0)
        return 0;
    else
        for (int i = 3; i * i <= x; i+=2)
            if(x % i == 0)
                return 0;
    return 1;
}
```

计算阶乘

```
long long int fact(int x){
    long long int ans=1LL;
    for(int i=1;i<=x;++i){
        ans=ans*i;
    }
    return ans;
}
```

关于这个函数有两点说明：

- 因为阶乘一般来说都很大，所以返回了`long long int`类型
- 请不要用递归写法。或者说，当能用循环时，尽量不要用递归写法。

递归的写法

递归的思想，以组合数递归为例：C(n,m)=C(n-1,m-1)+C(n-1,m)

在写递归的时候，你不要去关注电脑到底是怎么把C(m-1,n-1)和C(m-1,n)算出来的，也不需要去关心这俩到底是多少，你只需要知道：这两个值是**可以被算出来**的，写好边界条件，这就够了。

常用标准库函数

数学函数库`math.h`

使用时要在代码顶部加上头文件：`#include <math.h>`

`int abs(int num)`：对`int`类型的输入求绝对值

`long long llabs(long long num)`：对`long long`类型的输入求绝对值

`double fabs(double num)`：对`double`类型的输入求绝对值

`double pow(double a, double b)`：计算`a`的`b`次幂，返回值类型为`double`，不要用该函数来算整数的整数次幂

`double sqrt(double num)`：计算`num`的算术平方根，返回值类型为`double`，注意要求传入参数非负，否则会出错

`double sin(double x)`：计算三角函数正弦值，参数需要传入角的**弧度值**而不是角度值

`double cos(double x)`：计算三角函数余弦值，传入参数要求与`sin`相同

`double tan(double x)`：计算三角函数正切值，传入参数要求与`sin`相同

`double exp(double x)`：计算自然对数的底数 e 的 x 次幂

其他函数参考课件《P5-函数function》84 页

字符类型和映射函数库`ctype.h`

使用时要在代码顶部加上头文件：`#include <ctype.h>`

`int isdigit(int c)`：传入一个数字，判断数值是否是`'0'~'9'`这 10 个字符的 $ASCII$ 值，若是则返回一个非`0`值，否则返回`0`

`int islower(int c)`：传入一个数字，判断数值是否是`'a'~'z'`这 26 个小写字母字符的 $ASCII$ 值，若是则返回一个非`0`值，否则返回`0`

`int isupper(int c)`：传入一个数字，判断数值是否是`'A'~'Z'`这 26 个大写字母字符的 $ASCII$ 值，若是则返回一个非`0`值，否则返回`0`

其他函数参考课件《P5-函数function》81 页

编辑代码时的小细节

如果在输入内容的时候，发现光标后面的字符被新的输入字符覆盖了，而不是插入新的输入字符，原因是之前按下过键盘上的 $insert$ 键，再按一下就可以使输入方式恢复为在当前光标位置插入新输入的字符

扩展阅读：输出，以及格式控制字符串

在之前的学习中，大家见到了五花八门的格式控制字符串，例如：`%d、%1d、%02d、%lld、%.4f、%c`，那么它的规则到底是啥？有没有一个规则？实际上是有的。

格式控制字符串的标准格式如下：

```
%[flags][width][.precision][length]specifier
```

翻译成汉语：

```
%[标志][最小宽度][.精度][类型长度]类型。
```

specifier

有下面几种：

specifier	含义
d	十进制输出带符号整数
o	八进制输出无符号整数
x，X	十六进制输出无符号整数
u	十进制输出无符号整数
f	小数形式输出实数
e， E	指数形式输出实数
g， G	在f和e中选取较短的那一个形式输出实数
%	百分号
c	字符

以及大家还没学到，将要学的：

specifier	含义
p	指针地址
s	字符串

flags

flags	含义
-	在[width]给定的宽度里左对齐（默认右对齐）
+	强制在结果前显示符号
#	和o， x一起用时，输出前缀0或0x和e， f一起用，强制输出一个小数点
0	在[width]给定的右对齐时以0补位（默认是空格

width

width	含义
(数字)	输出所占据的长度。不足则右对齐，默认以空格补位，但超出不会被截断。
*	待定，作为附加整数值参数放置于要被格式化的参数之前

precision

.precision(精度)	specifier	含义
(数字)	整数	输出的最小位数，不足则以0补位，超过则不截断
(数字)	e，f	小数点后输出的小数位数
(数字)	g	最大有效位数
(数字)	s	长度
*		待定，作为附加整数值参数放置于要被格式化的参数之前。

length

length	含义
ll	long long
L	long double

这样一来，上面那些五花八门的格式控制字符串其实都是有规律可循的。例如

- `%.4f`，实际上是 flags ,width 和 length 被省略，precision 为 `4`，specifier 为 `f` 。
- `%02d`，实际上是 precision 和 length 被省略，flags 为 `0`，width 为 `2`，specifier 为 `d`。
- `%lld`，实际上是 flags,width,precision 被省略，length 为 `ll`，specifier 为 `d`。

$Author$: 梁秋月，逐月的游星（$also\ known\ as\ Asahi$）

审核：王君臣