顺序结构

赋值表达式

在C 语言中可以使用等号“=”来实现赋值操作：

```
int n = 5;
n = 6;
```

上面的第一个语句在定义变量的时候将 5 赋值给 `int` 型变量 `n`，然后在第二个语句中又把 6 赋值给了 `n`。而如果要给多个变量赋同一个值，可以使用连续等号的方法：

```
int n, m;
n = m = 5;
```

另外，等号右边也可以是一个表达式，例如：

```
#include <stdio.h>
int main(){
    int n = 3 * 2 + 1;
    int m = (n > 6) && (n < 8);
    n = n + 2;
    printf("%d %d\n", n, m);
    return 0;
}
```

输出结果：

```
9 1
```

上面的第一个语句将一个四则运算表达式的结果 7 赋值给了 `n`，而第二个语句判断 `n > 6` 和 `n < 8` 同时成立，因此将返回值 1 赋值给了 `m`。接着 `n = n + 2` 将 `n + 2` 赋值给 `n`，使得 `n` 变成 9。

最后，赋值运算符可以通过将其他运算符放在前面，来实现赋值操作的简化。例如，`n += 2` 的意思即为 `n = n + 2`，而 `n *= 3` 的意思即为 `n = n * 3`。下面再举个例子：

```
#include <cstdio>
int main() {
    int n = 12, m =3;
    n /= m + 1;
    m %= 2;
    printf("%d %d\n", n, m);
    return 0;
}
```

输出结果：

```
3 1
```

上面的代码中，`n /= m + 1` 等价于 `n = n / (m + 1)`，因此结果是 3；而 `m %= 2` 等价于 `m = m % 2`，因此结果是 1。当然，初学者应当尽量写成 `n /= (m + 1)` 的形式，这样可以避免因为基础不好而产生一些错误。

这种复合赋值运算符在程序中会被经常使用，并且可以加快编译速度，提高代码可读性，因此初学者即便没办法马上接受，也尽量去学着写。

使用 scanf 和 printf 输入输出

C 语言的 `stdio.h` 库函数中给我们提供了 `scanf` 函数和 `printf` 函数，分别对应输入和输出，学会这两个函数是 C 语言学习中必不可少的。

scanf 函数的使用

`scanf` 是输入函数，格式如下：

```
scanf("格式控制", 变量地址);
```

这看起来似乎有些抽象，不过其实很好理解。举个例子：

```
scanf("%d", &n);
```

代码中，双引号里面是一个`%d`，表示通过这个 `scanf` 我们需要输入一个 `int` 型的变量。那这个变量输入进来我们存在哪里呢？就是后面给出的 `n`。也就是说，通过这个 `scanf`，我们把输入的一个整数存放在 `int` 型变量 `n` 中。

接下来解释 `&n` 前面的`&`。在 C 语言中，变量在定义之后，就会在计算机内存中分配一块空间给这个变量，该空间在内存中的地址称为变量的地址。为了得到变量的地址，我们只需要在变量前加一个`&`（称为取地址运算符），也就是`&`变量名的写法。

既然`%d` 是 `int` 型变量的格式符，那么其他类型变量自然也有对

应的格式符，如下表：

	格式符	举例
int	%d	scanf("%d", &n);
long long	%lld	scanf("%lld", &n);
float	%f	scanf("%f", &fl);
double	%lf	scanf("%lf", &db);
char	%c	scanf("%c", &c);
字符串（char 数组）	%s	scanf("%s", str);

应该会有同学注意到，表 1-5 对字符数组的举例中，数组名 `str` 前面并没有&取地址运算符。这是由于数组比较特殊，数组名称本身就代表了这个数组第一个元素的地址，所以不需要再加取地址运算符。也许学弟妹现在对数组还不是太有概念，我们会在后面介绍到数组的时候再次提到这一点，现在只需要记住，在 `scanf` 中，除了 `char` 数组整个输入的情况不加`&`之外，其他变量类型都需要加`&`。

那么，如果有类似 13:45:20 这种 `hh:mm:ss` 的时间需要输入，应该怎么做？事实上我们可以使用下面代码的方法：

```
int hh, mm, ss;
scanf("%d:%d:%d", &hh, &mm, &ss);
```

可以看到，双引号内使用`%d:%d:%d`的写法跟输入格式`hh:mm:ss`是一样的，只是把`hh`、`mm`、`ss`的部分换成了`%d`以告诉计算机此处输入的是`int`型。这给我们一个启示：`scanf`的双引号内的内容其实就是整个输入，只不过把数据换成它们对应的格式符、并把变量的地址按次序写在后面而已。因此，如果要输入 12,18.23,t 这种格式的数据，那么就把 12 替换成`%d`、18.23 替换成`%lf`、t 替换成`%c`即可：

```
int a;
double b;
char c;
scanf("%d,%lf,%c", &a, &b, &c);
```

另外，如果要输入 3 4 这种用空格隔开的两个数字的话，两个`%d`之间可以不加空格：

```
int a, b;
scanf("%d%d", &a, &b);
```

可以不加空格的原因是，除了`%c`外，`scanf`对其他格式符（如`%d`）的输入是以空白符（即空格、`Tab`）为结束判断标志的，因此除非使用`%c`把空格按字符读入，其他情况都会自动跳过空格。另外，字符数组使用`%s`读入的时候以空格跟换行为读入结束的标志，如下面的代码所示。

```
#include <cstdio>
int main() {
    char str[10];
    scanf("%s", str);
    printf("%s",str);
    return 0;
}
```

输入数据：

```
abcd efg
```

输出结果：

```
abcd
```

再次强调，`scanf`的`%c`格式是可以读入空格跟换行的，因此下面的例子中字符`c`是一个空格，请学弟妹认真研究清楚这个例子。

```
#include <cstdio>
int main() {
    char c, str[10];
    scanf("%d%c%s", &a, &c, str);
    printf("a=%d,c=%c,str=%s", a, c, str);
    return 0;
}
```

输入数据：

```
1 a bcd
```

输出结果：

```
a=1,c= ,str=a
```

特别提醒：初学者特别容易在写 `scanf` 的时候漏写 `&`，因此如果在输入数据后程序异常退出，要马上考虑是否是在 `scanf` 中漏写了 `&`。

printf函数的使用

在 C 语言中，`printf` 函数用来输出。与 `scanf` 类似，`printf` 的格式如下：

```
printf("格式控制", 变量名称)
```

我们发现，`printf` 的双引号中的部分和 `scanf` 的用法是相同的，但是后面并不像 `scanf` 那样需要给出变量地址，而是直接跟上变量名称就行了，例如下面的例子：

```
int n = 5;
printf("%d", n);
```

如果上面的代码中使用 `scanf` 来输入 `n` 的话，就需要在 `scanf` 中使用 `&n`，但是 `printf` 只需要填写 `n` 就可以了。和 `scanf` 一样，我们给出各种常见变量类型对应的格式符，其中只有一个和 `scanf` 不同。

	格式符	举例
int	%d	printf("%d", n);
long long	%lld	printf("%lld", n);
float	%f	printf("%f", fl);
double	%f	printf("%f", db);
char	%c	printf("%c", c);
字符串(char 数组)	%s	printf("%s", str);

我们发现，`double` 类型的变量，其输出格式变成了 `%f`，而在 `scanf` 中却是 `%lf`。不过在有些系统中如果把输出格式写成 `%lf` 倒也不会出错，不过尽量还是按标准来。另外，不要因为 `float` 的 `scanf` 和 `printf` 的格式符都是 `%f` 比较好记而偷懒用 `float`，因为 `float` 的精度确实不能直视，如下面的代码所示：

```
#include <cstdio>
int main() {
    float f1 = 8765.4, f2 = 8765.4;
    double d1 = 8765.4, d2 = 8765.4;
    printf("%f\n%f\n", f1 * f2, d1 * d2);
    return 0;
}
```

输出结果：

```
76832244.007969
76832237.160000
```

我们可以发现，两个 `float` 类型的浮点数相乘，精度在整数部分就已经断线了，完全不能满足我们的要求。所以，`double` 才是真爱。

在 `printf` 中也可以使用转义字符（其实 `scanf` 里也可以，只是一般用不到），因此如果想在必要的地方换行，可以加上 `\n`：

```
#include <cstdio>
int main() {
    printf("abcd\nefg\n\nhijklmn");
    return 0;
}
```

输出结果：

```
abcd
efg

hijklmn
```

另外，如果想要输出 `'%'`、`'\'`，则需要在前面再加一个 `%` 或 `\`，例如下面的代码：

```
printf("%%");
printf("\\");
```

最后介绍三个**实用**的输出格式：

（1）`%md`

`%md` 可以使不足 `m` 位的 `int` 型变量以 `m` 位进行右对齐输出，其中高位用空格补齐；而如果变量本身超过 `m` 位，则保持原样。

来看一个实例：

```
#include <stdio.h>
int main(){
    int a = 123, b = 1234567;
    printf("%5d\n", a);
    printf("%5d\n", b);
    return 0;
}
```

输出结果：

```
  123
1234567
```

可以看见，123 有 3 位数字，不足 5 位，因此前面自动用两个空格填充，使整个输出凑足 5 位；而 1234567 已经大于 5 位，因此仍然直接输出。

（2）`%0md`

`%0md` 只是在`%md`中间多加了 0。和`%md` 的唯一不同点在于，当变量不足 `m` 位时，将在前面补足够数量的 0 而不是空格。

下面是一个例子：

```
#include <stdio.h>
int main(){
    int a = 123, b = 1234567;
    printf("%05d\n", a);
    printf("%05d\n", b);
    return 0;
}
```

输出结果：

```
00123
1234567
```

这里 123 的前面并不是用空格补齐了，而是使用 0 补齐。这**个格式在上机考试的某些题中非常适用。**

（3）`%.mf`

`%.mf` 可以让浮点数保留 `m`位小数输出，这个“保留”使用的是精度的“四舍六入五成双”规则（具体细节不必掌握）。很多题目都会要求浮点数的输出保留 XX 位小数（或是精确到小数点后 XX 位），就是用这个格式来进行输出（如果规定四舍五入，那么需要用到后面会介绍的 round 函数）。实例如下：

```
#include <cstdio>
int main() {
    double d1 = 12.3456;
    printf("%.0f\n", d1);
    printf("%.1f\n", d1);
    printf("%.2f\n", d1);
    printf("%.3f\n", d1);
    printf("%.4f\n", d1);
    return 0;
}
```

输出结果：

```
12
12.3
12.35
12.346
12.3456
```

使用 getchar 和 putchar 输入输出字符

`getchar` 用来输入单个字符，`putchar` 用来输出单个字符，在某些 `scanf` 使用不便的场合可以使用 `getchar` 来代替输入字符。

来看下面的例子：

```
#include <stdio.h>
int main(){
    char c1, c2, c3;
    c1 = getchar();
    getchar();
    c2 = getchar();
    c3 = getchar();
    putchar(c1); putchar(c2); putchar(c3);
    return 0;
}
```

输入数据：

```
abcd
```

输出结果：

```
Acd
```

此处第一个字符 `a` 被 `c1` 接收；第二个字符 `b` 虽然被接收，但是没有将它存储在某个变量中；第三个字符 `c` 被 `c2` 接收；第四个字符 `d` 被 `c3` 接收。之后，连续三次 `putchar` 将从 `c1`、`c2`、`c3` 连续输出。而如果如果我们输入 `ab`，然后敲回车，再输入 `c`，再敲回车，输出结果会是这样：

```
a
c
```

这是由于 `getchar` 可以识别换行符，所以 `c2` 实际上储存的是换行符 `\n`，因此在 `ac` 之间会有一个换行出现。

注释

注释是 `C/C++` 中常用到的，用来在需要作注解的语句旁边对语句进行解释。在程序编译的时候会自动跳过该部分，不执行这些被注释的内容。`C/C++` 的注释有两种：

（1）使用/**/注释

`/**/` 对 `/*` 跟 `*/` 之间的内容进行注释，且可以注释若干连续行的内容，实例如下：

```
#include <cstdio>
int main() {
    int a, b;
    scanf("%d%d", &a, &b); /*a++;
    b++;
    a = a * 2;*/
    printf("%d %d\n", a, b);
    return 0;
}
```

这样在 `/*` 跟 `*/` 之间的内容就都不会被执行了。

（2）使用//注释

`//`可以注释一行中在该符号之后的所有内容，效果仅限于该行，实例如下：

```
#include <cstdio>
int main() {
    int a, b;
    scanf("%d%d", &a, &b);
    a++;    //将 a 自增
    b++; //将 b 自增 //a = a * 2;
    printf("%d %d\n", a, b);
    return 0;
}
```

在上面的代码中，“将 a 自增”、“将 b 自增”、“a = a * 2”都被注释了。

typedef

`typedef`是一个很有用的东西，它能给复杂的数据类型起一个别名，这样在使用中就可以用别名来代替原来的写法。例如当数据类型是`long long`时，我们就可以像下面的例子这样用`LL`来代替`long long`，以避免在程序中出现大量的`long long`，降低编码效率。

```
#include <cstdio>
typedef long long LL;   // 给 long long 起个别名 LL
int main() {
    LL a = 123456789012345, b = 234567890123456;    // 直接使用 LL
    printf("%lld\n", a + b);
    return 0;
}
```

输出结果：

```
358024679135801
```

常用 math 函数

C 语言提供了很多实用的数学函数，如果要使用的话，需要在程序开头加上`math.h`头文件。下面几个是比较常用的数学函数，需要学弟妹掌握一下。

`fabs(double x)`对 double 型变量取绝对值函数，实例如下：

```
#include <stdio.h>
#include <math.h>
int main(){
    double db = -12.56;
    printf("%.2f\n", fabs(db));
    return 0;
}
```

输出结果：

```
12.56
```

`floor(double x)`、`ceil(double x)`分别用于`double`型变量的向下取整和向上取整，返回类型为`double`型。实例如下：

```
#include <stdio.h>
#include <math.h>
int main(){
    double db1 = -5.2, db2 = 5.2;
    printf("%.0f %.0f\n", floor(db1), ceil(db1));
    printf("%.0f %.0f\n", floor(db2), ceil(db2));
    return 0;
}
```

输出结果：

```
-6 -5
5 6
```

`pow(double r, double p)`返回r^p，要求`r`和`p`都是`double`型，实例如下：

```
#include <stdio.h>
#include <math.h>
int main(){
    double db = pow(2.0, 3.0);
    printf("%f\n", db);
    return 0;
}
```

输出结果：

```
8.000000
```

`sqrt(double x)`返回`double`型变量的算术平方根，实例如下：

```
#include <stdio.h>
#include <math.h>
int main(){
    double db = sqrt(2.0);
    printf("%f\n", db);
    return 0;
}
```

输出结果：

```
1.414214
```

`log(double x)`返回`double`型变量的以自然对数为底的对数，实例如下：

```c
#include <stdio.h>
#include <math.h>
int main(){
    double db = log(1.0);
    printf("%f\n", db);
    return 0;
}
```

输出结果：

```
0.000000
```

顺带一提，C 语言中没有对任意底数求对数的函数，因此必须使用换底公式来将不是以 `e` 为底的对数转换为以自然对数为底。

`sin(double x)`、`cos(double x)`、`tan(double x)` 分别返回 `double` 型变量的正弦值、余弦值、正切值，参数要求是弧度制，实例如下：

```c
#include <stdio.h>
#include <math.h>
const double pi = acos(-1.0);
int main(){
    double db1 = sin(pi * 45 / 180);
    double db2 = cos(pi * 45 / 180);
    double db3 = tan(pi * 45 / 180);
    printf("%f, %f, %f\n", db1, db2, db3);
    return 0;
}
```

输出结果：

```
0.707107, 0.707107, 1.000000
```

此处我们把 `pi` 定义为了精确值 `acos(-1.0)`（因为 `cos(pi) = -1`）。

`asin(double x)`、`acos(double x)`、`atan(double x)` 分别返回 `double` 型变量的反正弦值、反余弦值、反正切值，实例如下：

```c
#include <stdio.h>
#include <math.h>
int main(){
    double db1 = asin(1);
    double db2 = acos(-1.0);
    double db3 = atan(0);
    printf("%f, %f, %f\n", db1, db2, db3);
    return 0;
}
```

输出结果：

```
1.570796, 3.141593, 0.000000
```

`round(double x)` 将 `double` 型变量 `x` 四舍五入，返回类型也是 `double` 型，需进行取整，实例如下：

```
#include <stdio.h>
#include <math.h>
int main(){
    double db1 = round(3.40);
    double db2 = round(3.45);
    double db3 = round(3.50);
    double db4 = round(3.55);
    double db5 = round(3.60);
    printf("%d, %d, %d, %d, %d\n", (int)db1, (int)db2, (int)db3, (int)db4, (int)db5);
    return 0;
}
```

输出结果：

```
3,3,4,4,4
```