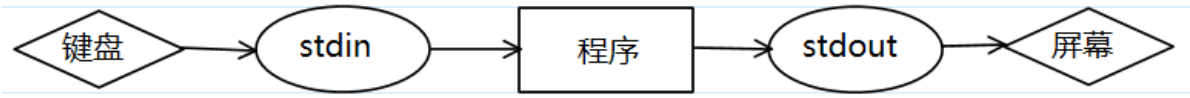
第二章 格式化输入/输出

`scanf` 和 `printf` 函数是C语言中使用最频繁的两个函数，它们用来进行格式化输入和输出。这两个函数功能强大，但要用好它们却不容易。

1. 输入/输出模型

这些年，我们的硬件设备：CPU，内存，I/O 设备都在不断迭代，不断朝着更快的方向努力。但是，在快速发展的过程中，有一个核心矛盾一直存在，那就是**这三者之间的速度差异**。我们可以形象地描述为：CPU 一天，内存一年；内存一天，IO 设备十年。(假设 CPU 执行一条普通指令需要一天，那么 CPU 读取内存就需要一年；假设内存之间传递单位数据需要一天，那么内存与 IO 设备之间传递单位数据就需要十年)。

为了平衡这三者之间的鸿沟，一个有效的手段是引入**缓冲**，如下图所示：



为了平衡内存和 IO 设备之间的速度差异，我们会在**内存**中设置一些缓冲区(buffer)，其中就有标准输入缓冲区 (stdin) 和标准输出缓冲区 (stdout). 一般情况下，stdin 关联到键盘，而 stdout 关联到屏幕。`scanf` 的作用是，从 stdin 读取数据到程序；而 `printf` 的作用是，将输出结果写入到 stdout。

2. printf 函数

`printf` 函数的作用：显示**格式串**中的内容，并用后面表达式的值替换格式串中的**转换说明**。

```
printf(格式字符串, 表达式1, 表达式2, ...);
```

格式串中包含普通字符和转换说明。其中，普通字符会原样显示，转换说明则会替换为后面表达式的值。如：

```
int i, j;
float x, y;

i = 10;
j = 20;
x = 43.2892f;
y = 5527.0f;

printf("i = %d, j = %d, x = %f, y = %f\n", i, j, x, y);
```

> Tips: `printf` 函数可以将其他类型的数据转换成字符数据，并输出到 stdout 缓冲区中。

2.1 转换说明

转换说明可以对输出格式进行精确的控制。

转换说明的格式为：`%m.pX` 或 `%-m.pX`，其中 m 和 p 都是整数常量，而 X 是字母。m 和 p 都是可选的；如果省略 p，则小数点也要省略。如：

```
%10.2f, %10f, %.2f, %f
```

最小字段宽度 (**m**inimum field width) m 指定了要显示的最少字符数量。如果显示值所需的字符数少于 m，那么会在值前面添加额外的空格(右对齐)。例如，转换说明 %4d 将以 •123 的形式显示 123 (•代表空格)。如果显示值所需的字符数多于m，那么会自动扩展为所需的大小，不会丢失数字。因此, %4d 将以 12345 的形式显示 12345。m前面的负号会导致值左对齐，%-4d 将以 123• 的形式显示123。

精度 (**p**recision) p 的含义依赖于**转换说明符** X 的选择。常用的转换说明符有以下几个：

- d(decimal) —— 表示十进制整数。p 表示待显示数字的最小个数 (必要时在数字前面补 0)；如果省略 p，则默认为1。
- f —— 表示"定点十进制"形式的浮点数。p 表示小数点后数字的个数 (默认为 6)。如果 p 为 0，则不显示小数点。

示例

```
int main(void) {
    int i = 40;
    float x = 839.21f;

    printf("|%d|%5d|%-5d|%5.3d|\n", i, i, i, i);
    printf("|%f|%10f|%10.2f|%-10.2f|\n", x, x, x, x);

    return 0;
}
```

3. scanf 函数

`scanf` 函数的作用：根据格式串读取 stdin 中的字符，并将字符转换成指定类型的数据后，写到后面表达式所指定的位置。

```
scanf(格式串, 表达式1, 表达式2, ...);
```

在绝大多数情况下，`scanf` 函数的格式串中只包含转换说明：

```
int i, j;
float x, y;

scanf("%d%d%f%f", &i, &j, &x, &y);
```

值得注意的是，通常我们会在后面变量的前面加 & 符号(取地址运算符)，但这并不是必须的。

3.1 scanf 函数的工作原理

`scanf`函数本质上是一个"模式匹配"函数，试图把 stdin 中的字符与格式串匹配。

`scanf`函数会从左到右依次匹配格式串中的每一项。如果匹配数据项成功，那么`scanf`函数会继续处理格式串的剩余部分；如果匹配不成功，那么`scanf`函数将不再处理格式串的剩余部分，而会立刻返回。

`scanf`函数的格式串中的每一项都表示一个匹配规则，其中：

%d: 忽略前置的**空白字符** (包括空格符、水平和垂直制表符、换页符和换行符)，然后匹配十进制的有符号整数。

%f: 忽略前置的空白字符，然后匹配浮点数。

注意：当 scanf 函数遇到一个不属于当前项的字符时，它不会读取该字符。在下一次读取输入时，才会读取该字符。

示例

```
scanf("%d%d%f%f", &i, &j, &x, &y);

输入：1-20.3-4.0e3\n
```

> Tips: `scanf` 可以从 stdin 缓冲区中读取字符数据，然后转换成其它类型的数据，并写到后面表达式所指定的位置。

3.2 格式串中的普通字符

`scanf`函数的格式串中也可以包含普通字符，普通字符也用来表示匹配规则。

- 空白字符：匹配 0 个或多个空白字符。
- 其他字符：精确匹配(是什么字符就匹配什么字符)。

示例

```
假设格式串是："%d/%d"
输入1：•5/•96
输入2：•5•/•96
```

3.3 不要混淆 printf 函数和 scanf 函数

虽然`scanf`函数调用和`printf`函数调用看起来很相似，但这两个函数之间有很大的差异!

一个常见的错误是：调用`printf`函数时，在变量的前面加 &。

```
printf("%d, %d\n", &i, &j); /*** WRONG ***/
```

`scanf`函数在寻找数据项时，通常会跳过前面的空白字符。所以除了转换说明，格式串通常不包含其他字符。

另一个常见的错误就是：认为`scanf`函数的格式串应该类似于`printf`函数的格式串。

```
scanf("%d, %d", &i, &j);
输入：10 20
```

课堂小练习

写一个程序，实现分数相加。用户以分子/分母的形式输入分数，程序打印相加后的结果。如：

```
Enter first fraction: 5/6
Enter second fraction: 3/4
The sum is 38/24
```

拓展：如何将结果化为最简分数？