C语言详解 - 输入输出

1. 输入输出一览

以下输入输出库函数的操作主要在标准输入输出设备（键盘和屏幕）与数据缓冲区之间进行。

1.1 printf()与scanf ()

printf()：将指定的文字/字符串输出到标准输出设备(屏幕)。
注意宽度输出和精度输出控制

scanf()：从标准输入设备(键盘)读取数据，并将值存放在变量中。

1、格式说明符个数与输入数据个数不相等的情况：

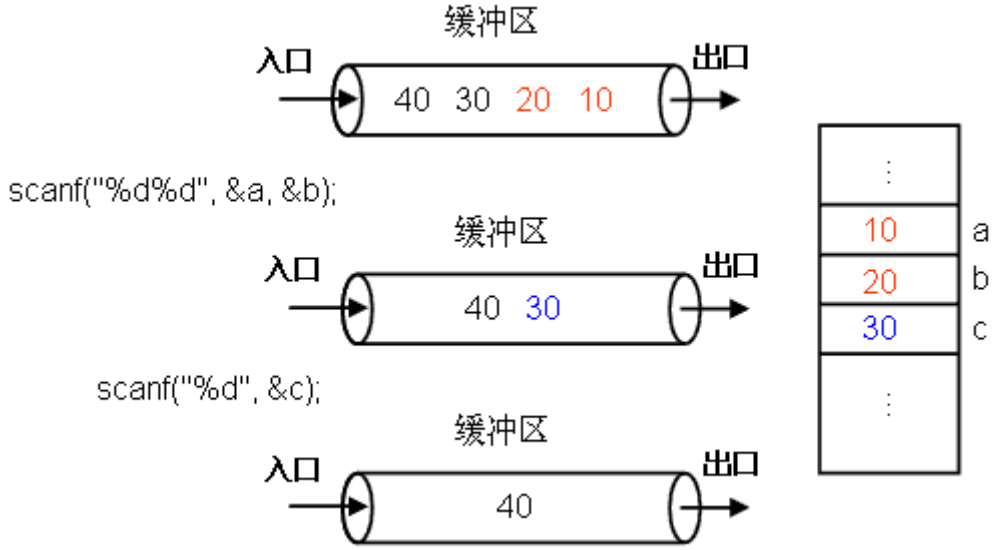
当我们从键盘输入数据时，数据之间必须用分隔符分开(空格、Tab键、回车)。并且要求scanf函数中格式说明符的个数与数据列表中数据的个数相等。如：

scanf("%d%d", &a, &b);
scanf("%d%d%d", &a, &b, &c);

如果出现不相等的情况，我们应该怎么分析呢?

其实，当我们输入数据时，只要没有按回车键，所输入的数据都还存放在缓冲区，并没有存入变量中。按【Enter】键后，scanf()函数才会从缓冲区中取走数据。缓冲区是一个先进先出的队列，即取走数据的时候，遵循先输入的数据先取走的原则。scanf函数的格式说明符有几个就要取几次数据，只要碰到格式说明符就必须把数据取走，至于是不是要把取走的数据存放起来，就得看数据列表中的数据个数。没取完的数据继续留在缓冲区中。

以下是缓冲区示意图，它像一根管道，有一个入口和一个出口，数据只能从入口中存入，只能从出口中取出。



以下是针对该图的C程序，数据输入为：10<空格>20<空格>30<空格>40<回车>

```
/*
 * 输入数据格式：10<空格>20<空格>30<空格>40<回车>
 * 输出：a=10 b=20
 *       c=30
 */
#include <stdio.h>
```

```c
int main()
{
    int a, b, c;

    scanf("%d%d", &a, &b); /* scanf中有2个格式说明符，表明要取两次数据，取走10和20
                                缓冲区中还剩下30和40
                            */
    printf("a=%d b=%d \n", a, b);

    scanf("%d", &c); /* scanf中只有1个格式说明符，表明要取1次数据，取走30
                        缓冲区中还剩下40，因此，读者还可以取一次。。。。。
                     */
    printf("c=%d \n", c);

    return 0;
}
```

2、宽度输出的情况：

(1) 在%和格式字符之间加入一个整数来控制输出数据所占宽度

```c
printf("%d\n", 365);
printf("%2d\n", 365);
printf("%5d\n", 365);
printf("%f\n", 3.14);
printf("%6f\n", 3.14);
printf("%12f\n", 3.14);
```

输出效果：

```
365
365
  365
3.140000
3.140000
    3.140000
```

(2) 在%和格式字符f之间加入一个“整数1. 整数2”来控制输出数据的格式

整数1：整个输出数据占的总宽度

整数2：输出实数的小数部分的个数

记住：先用整数2处理小数部分，再用整数1处理整个数据，包括已处理好的小数部分

```c
printf("%3.3f\n", 3.1415);
printf("%3.5f\n", 3.1415);
printf("%9.5f\n", 3.1415);
printf("%9.0f\n", 3.1415);
```

输出效果：

```
3.142
3.14150
  3.14150
        3
```

3、宽度输入的情况：

宽度输入指的是在%和格式说明符d之间加入一个整数。如：scanf("%2d", &x);

规则：

(1) 注意：%d与%1d是不同的

(2) 当宽度小于数据的实际宽度时，截取指定宽度的部分作为一个数进入缓冲区，再将剩余部分作为另一个数放入缓冲区。

基于以下程序进行测试，测试结果如下：

```
#include <stdio.h>

main()
{
    int a;
    float b,c;
    scanf("%3d%f%f",&a,&b,&c);
    printf("%d %f %f \n", a, b, c);
}
```

test case #1：

输入：1<空格>2<空格>3<回车>

输出：1 2.000000 3.000000

test case #2：

输入：123<空格>4<空格>5.1234567<回车>

输出：123 4.000000 5.123456

test case #3：

输入：1234<空格>5<空格>6<回车>

输出：123 4.000000 5.000000

test case #4：

输入：1.23<空格>4<空格>5<回车>

输出：1 0.230000 4.000000

test case #5：

输入：123.45<空格>6<空格>7<回车>

输出：123 0.450000 6.000000

test case #6：
输入：1234.5<空格>6<空格>7<回车>
输出：123 4.500000 6.000000

1.2 getchar()与putchar()

getchar()：将用户输入的字符输出到标准输出设备(屏幕)。按【Enter】键后，getchar()函数才会读入第一个字符，并返回该字符常量。
注：由于缓冲区的读取特性，当用户由键盘键入字符时，计算机并不会马上处理，而会暂存到系统的缓冲区(Buffter)内。到按【Enter】键后，getchar()函数才会读入缓冲区的第一个字符。而其它字符继续保留在缓冲区，等待下一个读取字符/字符串的函数来读入。
putchar()：用来输出指定的单一字符。
例1. 运行以下程序，输入：Hello!<回车>

```
#include <stdio.h>

int main()
{
    char c, s[20];
    printf("Enter a string: ");
    c = getchar();
    printf("Read the remaining from the buffer
…
\n");
    scanf("%s", s);

    putchar(c);
    putchar('\n');
    printf("%s \n", s);
}
```

例2. 输入一个汉字，并将它显示在屏幕上。
先输入：B超<回车>，观察输出。再运行程序，输入：超<回车>，比较输出结果：

```
#include <stdio.h>

int main()
{
    char c1, c2;
    printf("Enter an Chinese character: ");

    c1 = getchar();
    c2 = getchar();

    printf("The Chinese character entered is: ");
```

```
    putchar(c1);
    putchar(c2);

    putchar('\n');
}
```

注：由对比可知，当输入第一个数据时，只输出了字母“B”，汉字“超”没有输出。原因是一个汉字需要两个字节（字符）才能表示，所以对于第一个输入而言，所输入的字符“超”的第二个字节内容仍然保留在缓冲区。第二次运行，只输入“超”字时，通过两个getchar()函数将缓冲区中的“超”字全部读出，并用两次putchar()显示了一个完整的汉字。

1.3 getche()与getch()

getche()：该函数会由键盘输入一个字符，返回给调用者，并在屏幕上显示读入的字符。由于它并不读取缓冲区的字符，只要用户输入字符，getche()函数会立刻读取，而不需等待按【Enter】键。通常用于程序中只需用户输入一个字符，即可往下继续执行的情形。

getch()：它与getche()的区别是，getch()不需将所输入的字符显示到屏幕上。

例3. 测试getche()和getch()

```
#include <stdio.h>
#include <conio.h> /* getche(), getch() */

int main()
{
    char c1, c2;

    printf("Press any key to exit
...
");
    c1 = getche();
    putchar('\n');

    printf("Press any key once more to exit
...
");
    c2 = getch();
    putchar('\n');

    printf("The character getche() read: %c \n", c1);
    printf("The character getch() read: %c \n", c2);
}
```

1.4 gets()与puts()

gets()：

scanf输入字符串可以配合%s格式，但缺点是当遇到字符串中有空白或tab字符时，会自动视为串输入结束。因此不适合输入包含空白/tab字符的字符串。这时gets()函数就可解决该问题。
gets()函数会将用户整段字符串响应到标准输出设备(屏幕)上，当用户按下【Enter】键时，会读取缓冲区的所有字符并存放到指定字符数组中。
比较适合应用在多字符，中文字或长字符串的读取。
puts(): 用来输出字符串，输出完成后光标自动移到下一行。当输出数据时，会以'\0'字符作为该字符串的结束。
例4. 测试gets()和puts()

```
#include <stdio.h>

int main()
{
    char s[50];

    printf("Enter a string: ");
    gets(s);

    printf("The string you entered: ");
    puts(s);
}
```

2. stdio.h文件

另外还有一组输入输出函数是getc()和putc()，它们的作用也是输入和输出一个字符，先看它们在头文件stdio.h中的定义。
以下是头文件stdio.h的部分内容，请注意注释部分：

```
/* size_t的定义 */
#ifndef _SIZE_T_DEFINED
typedef unsigned int size_t;
#define _SIZE_T_DEFINED
#endif

#define EOF   (-1) //EOF的定义


/* FILE结构的定义 */

#ifndef _FILE_DEFINED
struct _iobuf {
        char *_ptr;
        int   _cnt;
```

```
        char *_base;
        int   _flag;
        int   _file;
        int   _charbuf;
        int   _bufsiz;
        };
typedef struct _iobuf FILE;
#define _FILE_DEFINED
#endif

#define FILENAME_MAX    260 //最大文件名长度
#define FOPEN_MAX       20 //最多可打开的文件数


/* NULL指针值的定义 */

#ifndef NULL
#ifdef  __cplusplus
#define NULL    0
#else
#define NULL    ((void *)0)
#endif
#endif


/* 标准I/O设备数组_iob[]的定义 */
#ifndef _STDIO_DEFINED
_CRTIMP extern FILE _iob[];
#endif


/* 标准I/O设备的定义 */

#define stdin  (&_iob[0])
#define stdout (&_iob[1])
#define stderr (&_iob[2])


/* 函数原型 */

_CRTIMP int __cdecl scanf(const char *,
...
_CRTIMP int __cdecl getc(FILE *);
_CRTIMP int __cdecl getchar(void);
_CRTIMP char * __cdecl gets(char *);

_CRTIMP int __cdecl printf(const char *,
```

```
...
);
_CRTIMP int __cdecl putc(int, FILE *);
_CRTIMP int __cdecl putchar(int);
_CRTIMP int __cdecl puts(const char *);


/* 宏定义 */

#define feof(_stream)      ((_stream)->_flag & _IOEOF)
#define getc(_stream)      (--(_stream)->_cnt >= 0 \
                ? 0xff & *(_stream)->_ptr++ : _filbuf(_stream))
#define putc(_c,_stream)   (--(_stream)->_cnt >= 0 \
                ? 0xff & (*(_stream)->_ptr++ = (char)(_c)) :  _flsbuf((_c),
(_stream)))
#define getchar()          getc(stdin) //调用getchar()时，实际调用的是getc(stdin)
#define putchar(_c)        putc((_c),stdout) //调用putchar(_c)时，实际调用的是
putc((_c), stdin)
```

例5. getc()函数和putc()函数

```
#include <stdio.h>

void main()
{
    int ch;

    FILE *pfin = stdin; //定义一个文件指针，并指向标准输入设备(键盘)
    FILE *pfout = stdout; //定义一个文件指针，并指向标准输出设备(屏幕)

    printf("Enter a string: ");

    ch = getc(pfin); //使用getc()函数获取缓冲区中的第一个字符

    putc(ch, pfout); //使用putc()函数输出该字符
    putc('\n', pfout); //使用putc()函数输出换行字符
}
```

运行结果：

Enter a string: Testing!

T

Press any key to continue