航C加油站——第二期

多组数据的输入

多组数据的输入一般采用 EOF 作为输入结束的判断，EOF是文件结束符（End of File）。C语言中把标准输入流stdin、标准输出流stdout、标准错误流stderr都当做文件处理，有关文件的内容会在后面的课程中讲。

```
while(scanf("%d",&n)!=EOF)//多组数据读入整数，直到读入所有内容
while((ch=getchar())!=EOF)//连续读入字符
while(get(a)!=NULL)//连续读入字符串，NULL是指向地址为0的指针，有关指针的内容后续课程中会讲，现在看个眼熟就行
```

有的同学会在处理一行字符串时使用 `while((ch=getchar())!='\n')` 判断结束，这在本地可行是因为本地按下回车时产生了换行符，但测评机测试时使用重定向输入输出到测试点文件的方法测试代码，当助教出题说明“一行”字符串时往往不会在行末添加换行符，这可能会导致使用换行符判断结束的代码因为一直遇不上换行符而不停读入，直到出现数组越界、超时等问题，因此轻易不要使用换行符判断结束。

for循环的嵌套

```
for(i=0;i<n;i++)
{
    for(j=i+1;j<n;j++)
    {
        //Your code
    }
}
```

这就是一个简单的 `for` 循环嵌套代码，`for` 循环理论上可以无限嵌套，但是需要考虑代码执行的时间，测评机可不会让你无限制的跑循环。使用循环需要注意使用的变量，在循环内改变循环变量（例子中的 `i`，`j`）的值之前建议多次确认，错误的使用变量会让你的代码产生错误的结果而你一时半会还找不到错哪儿了。

不同数据类型的精度范围

C语言标准文档并未明确规定不同的数据类型占用的字节数，只规定了 short 不短于 char，long 不短于 int，故下表描述的数据类型占用的字节数和最值仅适用于 32/64 位的 GCC 编译器：

数据类型	占用字节数 byte	最小值	最大值
char	1	-128	127
short	2	-32,768	32,767
int	4	-2,147,483,648	2,147,483,647
long	4	-2,147,483,648	2,147,483,647
long long	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
float	4		
double	8		
*	4/8		

讨论浮点数和指针地址的最大值和最小值是没有意义的，故上表没有写明这些类型的最值。

一般情况下，一个函数可使用的内存空间是1M，也就是 $1024 \times 1024 = 1,048,576\ Byte$。所以函数内最多可使用的 int 类型变量数量为 $1,048,576/4 = 262,144$，如果变量数量超过这个数，例如 `int a[300000];`，就需要放到函数外定义。

数组的定义和应用

```
//最常用的定义方式
int a[100];

//动态数组
int N;
scanf("%d",&N);
int a[N];

//数组的初始化
int a[100]={0};//全部初始化为0，动态数组不支持此方法
int b[100];
for(int i=0;i<100;i++)
{
    b[i]=i;//初始化为一个序列
}
```

以上是常见的几种数组的定义方式，其中动态数组的定义方式C99起开始支持，如果使用了这种定义方式被编译器报错，需要加入编译参数 `-std=c99` 。动态数组仅供了解，因为动态数组的空间数量依靠输入决定，输入数值过大容易爆栈，编译器无法对其空间数量合法性进行检验等问题，**不推荐使用。**

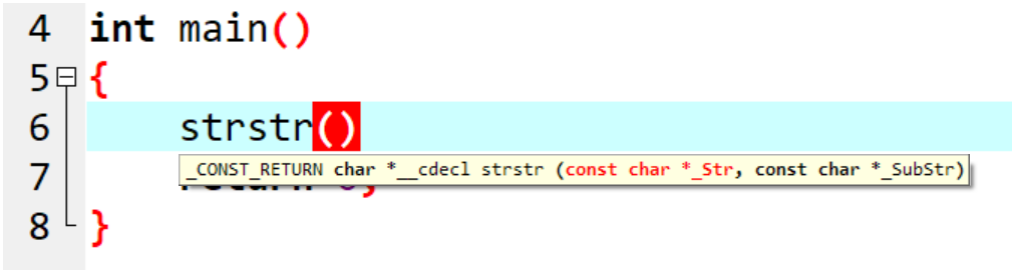
对于一些常用函数还不会使用，也没有储备。

常用函数：参考课本P243，使用方法百度即可（也可以选择看IDE的提示）。

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{

    return 0;
}
```

以Dev为例，写好 `#include<头文件名>` 之后保存文件，之后使用库函数可以看到提示：



这表示函数 `strstr()` 需要两个参数，分别是 `_Str（串）` 和 `_SubStr（子串）`，他们都是 `const char *` （字符指针常量）类型（这里可以简单理解为字符串变量名，具体含义会在指针那一讲说），这个函数的作用是在 `_Str` 中查找首个 `_SubStr`，找到返回第一次出现位置的指针，找不到返回 `NULL`。用法示例：

```
char a[100];
strstr(a,"What you want to find");//在字符串a中查找第一个"What you want to find"
```

标准库的函数名和函数参数一般都是英文单词或单词缩写，所以记得学好英语。

比如 `ctype.h` 库中的 `tolower()` 变成小写，和 `isalnum()` 是不是字母或数字 （is alpha number），`stdlib.h` 库中的 `memcpy()` 内存复制（memory copy）。

上课没听清比如stdio.h int main（）这类具体是什么意思、起什么作用

`stdio.h` 是C语言的标准库之一，本课程常用的几个标准库：

```
#include<stdio.h>//标准输入输出库，主要提供输入输出函数。standard input output
#include<stdlib.h>//标准库，主要提供一些不知道该放哪个库的函数，比如abs，system，abort等。standard libaray
#include<math.h>//数学库，主要提供数学函数。
#include<string.h>//字符串库，主要提供操作字符串的函数。
#include<ctype.h>//类型库，主要提供字符类型判断与转换函数。
//以上库的常用函数在教材P243
```

`int main()` ： `main` 表示这是程序的主入口，`int` 表示该函数的返回值类型是 `int`，通常主入口返回 0 表示程序正常结束，返回其他值则代表出现异常。

```
Process exited after xxx seconds with return value 0 表示程序正常结束，如果出现下面的值可以对照解决
3221225477 (0xC0000005)：访问越界，访问了系统不允许访问的内存地址（其他程序正在使用或地址位于系统保护区域），常见于数组越界或scanf没加取址符，测评机对此错误返回REG（Runtime Error SIGSEGV）
3221225725 (0xC00000FD)：堆栈溢出，递归层数过多或函数内定义的变量数量过多
3221225620 (0xC0000094)：除0错误，一般发生在整型数据除0时，测评机对此错误返回REP（Runtime Error SIGFPE）
```

程序主入口还有其他的定义方式，例如 `void main()` 、`int main(int argc, char *argv[])` 、`int main(void)` 等等。

单精度浮点数 float 和双精度浮点数 double 的区别、浮点数精度问题，为什么要加eps

E2-I 题目提供了浮点数在计算机中的存储方式：

在IEEE 754标准中，一个 m 位二进制浮点数通常由如下形式来表示：

$$se_{k-1}.....e_1e_0f_{n-1}.....f_1f_0 \quad (1+k+n=m; s, e_i, f_j \in \{0,1\})$$

其中：

- s 是符号位，决定这个数是负数 $(s=1)$ 还是正数 $(s=0)$
- $e_{k-1}.....e_1e_0$ 是阶码（指数）部分
- $f_{n-1}.....f_1f_0$ 是尾数部分

IEEE 754标准规定，通过如下方法计算二进制浮点数对应的十进制实数值 x：

首先计算 $bias = 2^{k-1}-1$，接下来分情况讨论：

- 若 $e_{k-1}.....e_1e_0$ 的每一位全为 1，且 $f_{n-1}.....f_1f_0$ 的每一位全为 0，则 x 为正无穷（$s=0$）或负无穷（$s=1$）
- 若 $e_{k-1}.....e_1e_0$ 的每一位全为 1，且 $f_{n-1}.....f_1f_0$ 的每一位不全为 0，则 x 不是一个数字
- 若 $e_{k-1}.....e_1e_0$ 的每一位全为 0，计算 $x=(-1)^s \times (0.f_{n-1}.....f_1f_0)_2 \times 2^{(1-bias)}$
- 若 $e_{k-1}.....e_1e_0$ 的每一位既不全为 1 也不全为 0，
  - 将二进制数 $e_{k-1}.....e_1e_0$ 转换为十进制数 E
  - 计算 $x=(-1)^s \times (1.f_{n-1}.....f_1f_0)_2 \times 2^{(E-bias)}$

单精度浮点数 float 如图所示，双精度浮点数 double 由 64 位二进制数据构成，单精度浮点数 float 由 32 位二进制数据构成，其区别是 double 中 k=16，n=47 而 float 中 k=8，n=23 。根据上述定义方式可知，当 n 和 k 变大时，其数据表示精度也相应变大，故双精度浮点数 double 的表示精度要高于单精度浮点数 float。

根据上图中的浮点数计算方法，可知 $小数 \times 2^x$ 这种计算方法不能表示所有的数字，尤其是在和整型数据进行计算时系统需要将整型转换为浮点数（隐式类型转换），计算和转换过程中系统可能无法完全准确的使用浮点数表示一个实数，从而使用可表示的最接近需要表示的实数的浮点数代替，这就是浮点数精度丢失问题。例如 `double a=6;` 在调试模式下可能会看到 `a=5.9999999999999` ，这就是系统使用了近似值表示。此时对 a 进行强制类型转换为 int 将变为 5，所以转换前需要对浮点数进行精度修正，最简单的办法是加上一个极小的实数，例如 `1e-6` 使 a 变为 `6.00000100000` 再转换为 int 。我们把这个极小的实数称为 eps 。

对于浮点数判断相等，一般差值小于 eps 即可认定相等，故 eps 也用于验证精度是否达到要求。

?:条件运算符

条件运算符是C语言当中唯一的三目运算符，它的基本构成是：`(a>b)?a:b` ，包含 3 个对象，分别是条件语句 `(a>b)` 、选项1 `(a)` 和选项2 `(b)` 。当条件语句为真时，条件运算符运算结果为选项1，否则运算结果为选项2。上面的用法是条件运算符用作获得最大值的一种常见用法。

位运算的算术右移

根据C99国际标准 ISO/IEC 9899:TC2 ： 左移或右移 `E1<<E2` 、`E1>>E2` ，运算结果的类型是 E1 的类型。如果右操作数 E2 是负值或大于等于左操作数 E1 的宽度，行为未定义。

左移：`E1<<E2`，E1 左移 E2 位，右侧用 0 填充，如果E1为无符号类型或有符号类型的非负值，则结果为 $E1 \times 2^{E2}$，否则行为未定义。

右移：`E1>>E2`，E1 右移 E2 位，如果E1为无符号类型或者E1为有符号类型的非负值，则结果为 $E1/2^{E2}$ 的整数部分。如果E1为有符号类型的复制，则结果由实现定义。

由上面的规范可知，左移都是逻辑左移，但右移则分为了逻辑右移和算术右移（左侧补1），出现算术右移的目的是将位移运算的结果统一为除2。如果二进制数的最高位是符号位，右移需要在最高位补1才能保证仍为负值。

通常情况下，编译器对负值的右移是算术右移，但不排除有一些编译器会讲右移统一为逻辑右移。

行为

编译器如何翻译或执行代码称作行为，除了规范明确定义的行为之外，还有以下几种情况：

行为未定义

在上文和下文中出现了 ”行为未定义“，这表明C语言规范不定义此类行为会导致的结果，同时编译器的开发者也没有定义此类行为的结果。行为未定义可能导致相同的代码在不同的编译器环境中编译的结果不一致，且代码多次运行的结果不确定。

未定义的行为是非法的、无意义的。一段正常的程序不应包含任何未定义的行为，程序员有义务保证其代码实现不依靠任何未定义行为。

例子：有符号变量计算过程中溢出的行为是未定义的。

例子：越界访问的行为是未定义的。

例子：无法确定运算顺序的运算行为是未定义的。（`i++ + ++i`）

行为由实现定义

在上文和下文中出现了 ”由实现定义“，这表明C语言规范没有规定此类行为会导致的结果，但编译器的开发者对此行为给出明确的结果定义。由实现定义可能导致相同的代码在不同的编译器环境中编译的结果不一致，但代码多次运行的结果是确定的。

例子：有符号且为负值的整数的右移的结果是由实现定义的。

例子：国际标准规定 int 的位宽不应小于 16 位，现在大多数环境定义为 32 位。

行为由当地环境定义

行为由当地环境定义表明编译器的具体行为依赖于所在的国家、文化或语言惯例。

由当地环境定义的行为是合法的。

例子：ctype.h库中的函数 islower 对26个拉丁小写字母以外的内容是否返回真值是由当地环境定义的。

未指明的行为

行为未指明表明使用了未指定的值，或标准提供两种或多种可能性的其他行为，并且在任何情况下都没有强加进一步的要求。

未指明的行为仍然是合法的行为。

例子：函数调用过程的实参计算顺序是未指明的。

例子：一次定义的变量在内存中分配的位置是未指明的。（可以连续也可以分散）

ctype.h 库内函数的变量形式

ctype.h 库内的函数分为判断字符类型函数和转换字符类型函数，所有函数接受的传入参数的类型都是 `int` 。对于判断字符类型函数，字符符合条件返回非零值，不符合条件返回 0 。对于转换字符类型函数（`tolower` `toupper`）如果传入参数是大写/小写字符，则返回对应的小写/大写字母，否则返回原值。

输出格式符

根据C99国际标准 ISO/IEC 9899:TC2 格式化输出使用的格式符的完整形式：`%[flag][width][.precision][length]type`，中括号表示此处内容不是必须的。

flag：可选，标志符号

标志符号	含义	无此符号的默认值	备注
-	左对齐	右对齐	
+	输出符号	仅负数输出符号	type需为 d 或 f 或 F
#	输出前缀	无前缀	type需为 o 或 x 或 X
#	无小数部分时输出小数点	无小数点	type需为 a, A, e, E, f, F
#	保留有效数字最后的0		type需为 g, G
空格	数字前若无字符则输出空格	无空格	不能与 + 搭配使用
0	填充内容修改为0	默认填充空格	不能与 - 搭配使用；type需为d, i, o, u, x, X, a, A, e, E, f, F, g, G；type为d, i, o, u, x, X时不能和 .precision 搭配使用

width：可选，最小输出宽度。如果需要输出的内容小于此处设置的值，则补充设置的填充内容（默认是空格）。该选项允许设置为 `*` 或正整数。

.precision：可选，输出精度设置。该选项允许设置为 `*` 或非负整数，点 `.` 是必须的，如果仅包含了点，则认为输入的是 `.0` 。如果该选项搭配的 type 不在下面的表格中，则行为未定义。

精度设置含义	适用的type
最小数字位宽	d, i, o, u, x, X
最大小数位数	a, A, e, E, f, F
最大有效数字位数	g, G
最长字符串长度	s

length：可选，长度修饰符。下表隐藏了不常用的长度修饰符。

长度修饰符	含义	适用的type
h	short（短整型）	d, i, o, u, x, X
l	long（长整型）	d, i, o, u, x, X
l	double（双精度浮点）	a, A, e, E, f, F, g, G
ll	long long（64位长整型）	d, i, o, u, x, X
L	long double	a, A, e, E, f, F, g, G

type：必选，输出类型。下表隐藏了不常用的类型。

type	变量类型	输出形式	备注
d, i	int	dddd	
o	unsigned int	dddd	八进制
u	unsigned int	dddd	
x, X	unsigned int	dddd	十六进制，X表示字母大写，x表示字母小写
f, F	float	ddd.ddd	对于inf、nan，如果为F则输出INF、NaN
e, E	float	d.ddde±dd	
g, G	float		在十进制和指数形式中选择较短的输出
c	unsigned char	c	
s	char *	字符串	
p	void *	指针地址	
%		%	

输入格式符

根据C99国际标准 ISO/IEC 9899:TC2 格式化输入使用的格式符的完整形式：`%[flag][width][length]type`，中括号表示此处内容不是必须的。

flag：可选，标志符号

标志符号	含义	无此符号的默认值
*	屏蔽此输入	接受此输入

width：可选，最大输入宽度。该选项允许设置为正整数。

length：可选，长度修饰符。下表隐藏了不常用的长度修饰符。

长度修饰符	含义	适用的type
h	short（短整型）	d, i, o, u, x, X
l	long（长整型）	d, i, o, u, x, X
l	double（双精度浮点）	a, A, e, E, f, F, g, G
ll	long long（64位长整型）	d, i, o, u, x, X
L	long double	a, A, e, E, f, F, g, G

type：必选，输出类型。下表隐藏了不常用的类型。

type	变量类型	输入形式	备注
d	int	dddd	
o	unsigned int	dddd	八进制
u	unsigned int	dddd	
x	unsigned int	dddd	十六进制
f	float	ddd.ddd	
e	float	d.ddde±dd	科学计数法
c	unsigned char	c	
s	char *	字符串	
[]	char *	条件匹配	连续读入符合条件的字符（正则表达式）
	[abcd]		连续读入abcd直到遇到非abcd的字符
	[a-z]		连续读入小写字母
	[^\n]		连续读入任意字符直到遇到换行符；^表示不接受下列字符

运算符优先级

优先级	运算符	名称或含义	使用形式	结合方向	说明
1	[]	数组下标	数组名[常量表达式]	左到右	
	()	圆括号	（表达式）/函数名(形参表)		
	.	成员选择（对象）	对象.成员名		
	->	成员选择（指针）	对象指针->成员名		
2	-	负号运算符	-常量	右到左	单目运算符
	(type)	强制类型转换	(数据类型)表达式		
	++	自增运算符	++变量名		单目运算符
	--	自减运算符	--变量名		单目运算符
	*	取值运算符	*指针变量		单目运算符
	&	取地址运算符	&变量名		单目运算符
	!	逻辑非运算符	!表达式		单目运算符
	~	按位取反运算符	~表达式		单目运算符
	sizeof	长度运算符	sizeof(表达式)		
3	/	除	表达式/表达式	左到右	双目运算符
	*	乘	表达式*表达式		双目运算符
	%	余数（取模）	整型表达式%整型表达式		双目运算符
4	+	加	表达式+表达式	左到右	双目运算符
	-	减	表达式-表达式		双目运算符
5	<<	左移	变量<<表达式	左到右	双目运算符
	>>	右移	变量>>表达式		双目运算符
6	>	大于	表达式>表达式	左到右	双目运算符
	>=	大于等于	表达式>=表达式		双目运算符
	<	小于	表达式<表达式		双目运算符
	<=	小于等于	表达式<=表达式		双目运算符
7	==	等于	表达式==表达式	左到右	双目运算符
	!=	不等于	表达式!= 表达式		双目运算符
8	&	按位与	表达式&表达式	左到右	双目运算符

9	^	按位异或	表达式^表达式	左到右	双目运算符
10	|	按位或	表达式|表达式	左到右	双目运算符
11	&&	逻辑与	表达式&&表达式	左到右	双目运算符
12	||	逻辑或	表达式||表达式	左到右	双目运算符
13	?:	条件运算符	表达式1? 表达式2: 表达式3	右到左	三目运算符
14	=	赋值运算符	变量=表达式	右到左	
	/=	除后赋值	变量/=表达式		
	*=	乘后赋值	变量*=表达式		
	%=	取模后赋值	变量%=表达式		
	+=	加后赋值	变量+=表达式		
	-=	减后赋值	变量-=表达式		
	<<=	左移后赋值	变量<<=表达式		
	>>=	右移后赋值	变量>>=表达式		
	&=	按位与后赋值	变量&=表达式		
	^=	按位异或后赋值	变量^=表达式		
	|=	按位或后赋值	变量|=表达式		
15	,	逗号运算符	表达式,表达式,...	左到右	从左向右顺序运算