

c语言基础

前言：为什么学习C语言？

C语言是其它众多高级语言的鼻祖语言，所以说学习C语言是进入编程世界的必修课。C语言执行效率高而且可移植性好，可以用来开发应用软件、驱动、操作系统等。

好的大学，应该是教大家计算机思维。教How to Think Like a Computer Scientist，而不是教你如何用好某一种语言去搬砖。C语言能轻松对应上汇编代码，这也就意味着学生学会C语言能很好代表学生理解了计算机行为。

而更高级一些的语言，为了更加符合特定的思维方式，都在往着更抽象的概念方向进行设计。因此它们不同程度上脱离了汇编层面，所以难起到教学目的。

一、Hello World

先来看看一个简单的C语言程序

```
#include <stdio.h>
int main()
{
    printf("Hello world\n");
    return 0;
}
```

1、C语言结构

简单来说，一个C程序就是由若干 头文件 和 函数 组成。

```
#include <stdio.h>           //包含头文件
```

```
int main()                   //主函数
{
    printf("Hello world\n");
    return 0;
}
```

- `include <stdio.h>` 就是一条预处理命令，作用为通知C语言编译系统在对C程序进行正式编译前需要做一些预处理工作。
- 函数 就是实现代码逻辑的一个小的**单元**。许多程序设计语言中，可以将一段代码封装起来，在需要使用时可以直接调用，所以，函数也可以说是许多代码的集合，这就是程序中的函数。

2、主函数

主函数是必不可少的!!!

- 一个C程序有且只有一个主函数，即 `main` 函数，**主函数**就是C语言中的**唯一入口**。
- `main`前面的`int`就是主函数的类型。（`int/void`代表一个函数是否有返回值，这里先暂时不讲）

- `printf()` 是**格式输出**函数，这里就记住它的功能就是在**屏幕上输出指定的信息**
- **return**是函数的返回值，根据函数类型的不同，返回的值也是不同的。这里**return 0**代表正常退出。

注：代码规范

1. **一个说明或一个语句占一行**。例如：包含头文件、一个可执行语句结束都需要**换行**。
2. 函数体内的语句要有明显**缩进**，通常以**按一下Tab键为一个缩进**。
3. 括号要**成对写**。（无论是大括号中括号还是小括号）
4. 当一句可执行语句结束的时候末尾需要有**分号**。
5. 代码中所有符号均为**英文半角符号**。
6. **习惯写注释!!!**

程序解释——注释

（注释是写给程序员看的，不是写给电脑看的。）

C语言注释方法有两种：

- 多行注释： `/*注释内容*/`
- 单行注释： `//注释一行`

标识符

C语言规定，标识符可以是字母（A~Z，a~z）、数字（0~9）、下划线_组成的字符串，并且第一个字符必须是**字母或下划线**。

注意以下几点：

1. 严格区分大小写的。例如 `data` 和 `Data` 是两个不同的标识符。
2. 标识符不能是C语言的关键字。

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

转义字符

转义字符是一种特殊的字符常量。转义字符以反斜线"\"开头，后跟一个或几个字符。

转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符。例如，在`printf`函数的格式串中用到的“\n”就是一个转义字符，其意义是“回车换行”。

转义字符	含义	ASCII码（16/10进制）
\0	空字符(NULL)	00H/0
\n	换行符(LF)	0AH/10
\r	回车符(CR)	0DH/13
\t	水平制表符(HT)	09H/9
\v	垂直制表(VT)	0B/11
\a	响铃(BEL)	07/7
\b	退格符(BS)	08H/8
\f	换页符(FF)	0CH/12
\'	单引号	27H/39
\"	双引号	22H/34
\\	反斜杠	5CH/92
\?	问号字符	3F/63
\ddd	任意字符	三位八进制
\xhh	任意字符	二位十六进制

3、变量及赋值

变量就是可变化的量，每一个变量都有一个名字（标识符）。变量占据内存中一定的存储单元。**使用变量之前必须先定义变量。**

变量定义的一般形式为：数据类型 变量名；

多个类型相同的变量：数据类型 变量名, 变量名, 变量名...;

```
int num;           //定义了一个整型变量，名字叫num
num = 100;         //给num这个整型变量赋值为100

int a,b,c;         //同时定义3个类型相同的变量，都是整型变量
a=1;               //分别赋值
b=2;
c=3;
```

注：在定义中不允许连续赋值，如 `int a=b=c=5;` 是不合C语言语法的。

另，变量的赋值分为两种方式：

1. 先声明再赋值

```
int a;
a=1;
```

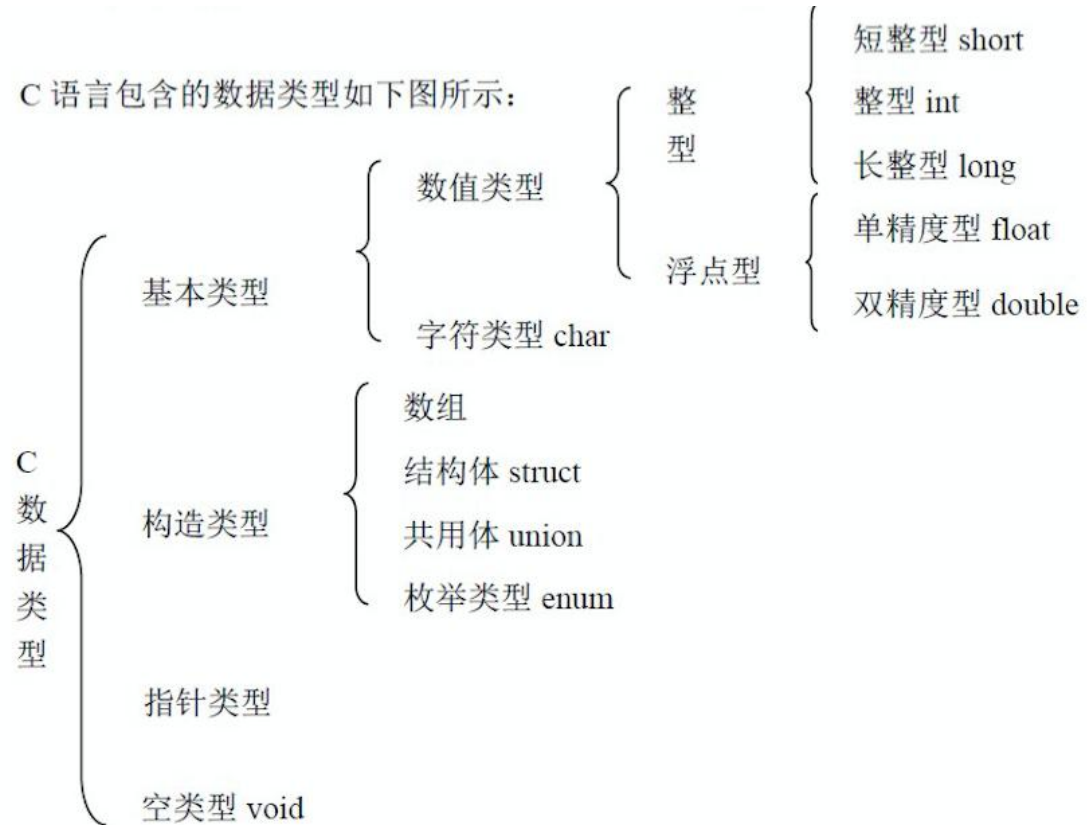
2. 声明的同时赋值

```
int a=1;
```

4、基本数据类型

C语言中，数据类型可分为：

1. 基本数据类型
2. 构造数据类型
3. 指针类型
4. 空类型四大类



今天主要讲基本类型。整型、浮点型、字符型(int,float,double,char)最为常见。

整型（不带小数的数字）

数据类型	说明	字节	取值范围
int	整型	2	$(-32768 \sim 32767) -2^{15} \sim 2^{15}-1$
short int	短整型（int可以省略）	2	$(-32768 \sim 32767) -2^{15} \sim 2^{15}-1$
long int	长整型（int可以省略）	4	$(-2147483648 \sim 2147483647) -2^{31} \sim 2^{31}-1$
unsigned int	无符号整型	2	$(0 \sim 65535) 0 \sim 2^{16}-1$
unsigned short int	无符号短整型（int可以省略）	2	$(0 \sim 65535) 0 \sim 2^{16}-1$
unsigned long int	无符号长整型（int可以省略）	4	$(0 \sim 4294967295) 0 \sim 2^{32}-1$

浮点型（带小数的数字）

因为精度的不同又分为3种

数据类型	说明	字节	取值范围
float	单精度型	4	$-3.4 \times 10^{-38} \sim 3.4 \times 10^{38}$
double	双精度型	8	$-1.7 \times 10^{-308} \sim 1.7 \times 10^{308}$
long double	长双精度型	16	$-1.2 \times 10^{-4932} \sim 1.7 \times 10^{4932}$

字符型

数据类型	说明	字节	应用
char	字符型	1	用于储存单个字符

示例:

```
int age = 18;
char sex = 'M';
float height = 170.1;
double pi = 3.1415926;
```

5、格式化输出语句

格式化输出语句，是将各种类型的数据按照**格式化后的类型及指定的位置**从计算机上显示。

其格式为： `printf("输出格式符", 输出项);`

格式符	说明	举例
%d	带符号十进制整数	int a =10;printf("%d", a);输出结果为10
%c	单个字符	char x = 'a';printf("%c", x);输出结果为a
%s	字符串	printf("%s", "红岩网校");输出结果为红岩网校
%f	6位小数	float a = 1.23;printf("%f", a);输出结果为1.230000

另：若想控制输出小数位数,使用以下格式：

```
#include <stdio.h>
int main()
{
    printf("%.2f", 1.2345678);    //则输出为1.23
    return 0;
}
```

注：格式符的个数要与变量、常量或者表达式的个数一一对应

```
int a = 10;
float b = 7.56;
char x = 'c';
printf("整数: %d,小数: %f,字符: %c", a, b, c);
```

6、输入与输出

输出: `printf("%d",a);`

输入: `scanf("%d",&a);`

```
#include <stdio.h>
int main()
{
    char a[10];
    scanf("%s",&a);
}
```

```
#include <stdio.h>
int main()
{
    char a[10];
    scanf("%s",a);
}
```

对于初学者而言，肯定有过这样一个困惑：为什么在使用scanf函数输入数据时有时需要在参数前加一个&。

在回答"&"是什么之前我们先来看看scanf函数的函数原型是这样的：

```
int scanf(const char * restrict format,...)
```

该函数为int类型，函数接受指针类型的常量，当然对于初入江湖的少侠而言“指针”这个概念肯定十分陌生，不过没关系，其实指针就是数据储存的地址，但对于没有计算机基础的少侠而言，地址肯定有是一个陌生的概念，打个比方，我们将数据比作我们自己，计算机中的数据像我们一样都需要一个安身之处，我们的住所通常都有一个门牌号，而数据的安身之处就是地址。

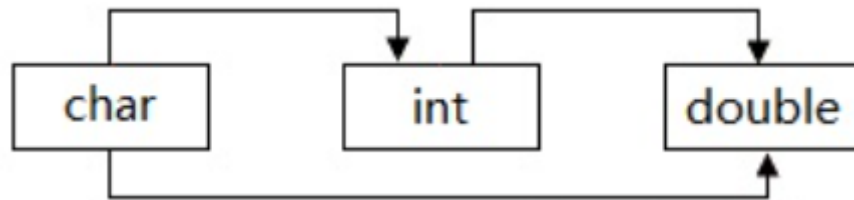
当我们声明一个变量名时(int a),计算机就会为它分配一个相应的储存单元，这时a就有了安身之处，也就有了一个地址，当我们使用scanf()函数输入数据时，我们就需要找到a的地址，并将数据存储到那个位置上。

7、类型转化

- 自动类型转换
- 强制类型转换

自动类型转换

数据类型存在自动转换的情况。自动转换发生在**不同数据类型**运算时，在编译的时候**自动完成**。



`char` 类型数据转换为 `int` 类型数据遵循 ASCII 码中的对应值。

注：

字节小的可以向字节大的自动转换，但字节大的不能向字节小的自动转换。

`char` 可以转换为 `int`，`int` 可以转换为 `double`，`char` 可以转换为 `double`。但是不可以反向。

强制类型转换

强制类型转换是通过**定义类型转换运算**来实现的。其一般形式为：`(数据类型) (表达式)`

其作用是把表达式的运算结果强制转换成**类型说明符所表示的类型**

eg:

```
a = 5 / 2;  
(int)(a);
```

在使用强制转换时应注意以下问题：

1. 数据类型和表达式都必须加括号，如把 `(int)(x/2+y)` 写成 `(int)x/2+y` 则成了把 `x` 转换成 `int` 型之后再除 2 再与 `y` 相加了。
2. 转换后不会改变原数据的类型及变量值，只在本次运算中临时性转换。
3. 强制转换后的运算结果**不遵循四舍五入原则**。

二、运算

1、运算符

C语言中的运算符：

- ※ 算术运算符
- ※ 赋值运算符
- ※ 关系运算符
- ※ 逻辑运算符
- ※ 三目运算符

2、算术运算符

C语言基本运算符：

名称	运算符号	举例
加法运算符	+	2+10=12
减法运算符	-	10-3=7
乘法运算符	*	2*10=20
除法运算符	/	30/10=3
求余运算符（模运算符）	%	23%7=2
自增运算符	++	int a = 1; a++;
自减运算符	--	int a = 1; a--;

除法运算中要注意：

如果相除的两个数都是整数的话，则结果也为整数，**小数部分省略**，如 $8/3 = 2$ ；
而两数中有一个为小数，结果则为小数，如： $9.0/2 = 4.500000$ 。

取余运算中要注意：

该运算只适合用**两个整数**进行取余运算，如： $10\%3 = 1$ ；而 $10.0\%3$ 则是错误的；
运算后的符号取决于被模数的符号，如 $(-10)\%3 = -1$ ；而 $10\%(-3) = 1$ ；

注：

C语言中没有乘方这个运算符，也不能用 \times ， \div 等算术符号。

C语言中什么时候四舍五入什么时候直接截取舍呢？

- 会进行四舍五入：

```
#include <stdio.h>
int main()
{
    float a=5.666;
    printf("%.2f",a);
}
```

- 其他情况均不会进行四舍五入，直接截取舍
①：强制类型转换；自动类型转换
②：用%d输出浮点型数

自增与自减运算符

- 自增运算符为 `++`，其功能是使变量的值自增1。

- 自减运算符为 `--`，其功能是使变量的值自减1。

它们经常使用在循环中。自增自减运算符有以下几种形式：

运算表达式	说明	运算规则
<code>++a</code>	a自增1后，再取值	先运算，再取值
<code>--a</code>	a自减1后，再取值	先运算，再取值
<code>a++</code>	a取值后，a的值再自增1	先取值，再运算
<code>a--</code>	a取值后，a的值再自减1	先取值，再运算

3、赋值运算符

- 简单赋值运算符
- 复合赋值运算符

简单赋值运算符：就是=号；例如`a = 1`，就是把1赋值给a。

复合赋值运算符：复合赋值运算符就是在简单赋值符 `=` 之前加上其它运算符构成。

例如 `+=`、`-=`、`*=`、`/=`、`%=`

eg:

```
int a = 3;
a += 5;
```

定义整型变量a并赋值为3，`a += 5`; 这个算式就等价于 `a = a+5`; 将变量a和5相加之后再赋值给a。

注：

复合运算符中运算符和等号之间是不存在空格的。

4、关系运算符

符号	意义	举例	结果
>	大于	10>5	1
>=	大于等于	10>=10	1
<	小于	10<5	0
<=	小于等于	10<=10	1
==	等于	10==5	0
!=	不等于	10!=5	1

关系表达式的值是 `真` 和 `假`，在C程序用整数 `1` 和 `0` 表示。

注：

`>=`，`<=`，`==`，`!=` 这种符号之间**不能存在空格**。

5、逻辑运算符

符号	意义	举例	结果
&&	逻辑与	0&&1	0
	逻辑或	0 1	1
!	逻辑非	! 0	1

逻辑运算的值也是有两种分别为 `真` 和 `假`，C语言中用整型的1和0来表示。其求值规则如下：

- 与运算 `&&`

参与运算的两个变量都为真时，结果才为真，否则为假。例如：5>=5 && 7>5，运算结果为真；

- 或运算 `||`

参与运算的两个变量只要有一个为真，结果就为真。两个量都为假时，结果为假。例如：
5>=5 || 5>8，运算结果为真；

- 非运算 `!`

参与运算的变量为真时，结果为假；参与运算量为假时，结果为真。例如：!(5>8)，运算结果为真。

6、三目运算符

C语言中的三目运算符：`?:`，其格式为：

表达式1 ? 表达式2 : 表达式3;

执行过程是：

先判断表达式1的值是否为真，如果是真的话执行表达式2；如果是假的话执行表达式3。

```
#include <stdio.h>
int main()
{
    float money =12.0; //定义钱包里的钱
    float cost =11.5;  //定义所需费用
    printf("钱够吗? ");
    printf("%c\n",money>=cost?'y':'n'); //输出y就够，输出n就不够
    return 0;
}
```

7、运算符优先级比较

优先级	运算符
1	()
2	！（非） +（正号） -（负号） ++（自增） --（自减）
3	* / %
4	+（加） -（减）
5	< <= >= >
6	== !=
7	&&
8	
9	?:（三目运算符）
10	= += -= *= /= %=

三、分支结构

1、简单if语句

C语言中的分支结构语句中的if条件语句。

简单if语句的基本结构如下：

```
if(表达式)
{
    执行代码块;
}
```

其语义是：如果表达式的值为真，则执行其后的语句，否则不执行该语句。

注：if(表达式)后面没有分号。

2、简单if-else语句

简单的 if-else 语句的基本结构：

```
if(表达式)
{
    执行代码块1;
}
else
{
    执行代码块2;
}
```

语义：如果表达式的值为真，则执行代码块1，否则执行代码块2。

注：if(表达式)后面没有分号，else后面也没有分号。

3、多重if-else语句

C语言中多重 if-else 语句，其结构如下：

```
if(表达式1)
{
    执行代码块1;
}
else if(表达式2)
{
    执行代码块2;
}
...
else
{
    执行代码块n;
}
```

语义：依次判断表达式的值，当出现某个值为真时，则执行对应代码块，否则执行代码块n。

注：当某一条件为真的时候，则不会向下执行该分支结构的其他语句。

4、嵌套if-else语句

C语言中嵌套 if-else 语句。嵌套 if-else 语句的意思，就是在 if-else 语句中，再写 if-else 语句。其一般形式为：

```
if(表达式)
{
    if(表达式)
    {
        执行代码块1;
    }
    else
    {
        执行代码块2;
    }
}
else
{
    执行代码块3;
}
```

5、switch语句

switch语句需要用到break结束语句，可以看完后面break语句后再来理解。

switch语句结构如下：

```
switch(表达式)
{
    case 常量表达式1: 执行代码块1; break;
    case 常量表达式2: 执行代码块2; break;
    ...
    case 常量表达式n: 执行代码块n; break;
    default: 执行代码块n+1;
}
```

直接来看看示例：

```
#include <stdio.h>
int main()
{
    char a;          //a为输入的第一个字母，b为第二个
    printf("请输入一个字母: ");
    scanf("%c",&a);    //注意输入的类型为字符型，开始自己惯例写成整型
    switch(a)
    {
        case 'm':
            printf("monday\n");
            break;
        case 'w':
            printf("wendesday\n");
            break;
        case 'f':
```

```

        printf("friday\n");
        break;
    default :
        printf("error\n"); break;
    }
}

```

注意以下几点：

1. 在case后的各**常量表达式**的值不能相同，否则会出现错误。
2. 在case子句后如果没有**break**；会一直往后执行**一直到遇到break**；才会跳出switch语句。
3. switch后面的表达式语句只能是**整型**或者**字符类型**。
4. 在case后，允许有多个语句，**可以不用{}括起来**。
5. 各case和default子句的先后顺序可以变动，而不会影响程序执行结果。
6. `default` 子句可以省略不用。

四、循环结构

1、while循环

C语言中有**三种**循环结构,先看一下C语言while循环的结构。

```

while(表达式)      //循环条件
{
    执行代码块;
}                  //循环体

```

其中表达式表示**循环条件**，执行代码块为**循环体**。

while语句的语义是：计算表达式的值，当值为 **真(非0)** 时， 执行循环体代码块。

1. while语句中的表达式一般是关系表达或逻辑表达式，当表达式的值为假时不执行循环体，反之则循环体一直执行。
2. 一定要记着在循环体中改变循环变量的值，否则会出现死循环（无休止的执行）。
3. 循环体如果包括有一个以上的语句，则必须用 `{}` 括起来，组成复合语句。

从1加到100， eg：

```

#include <stdio.h>
int main()
{
    int i = 1,sum = 0;
    while(i <= 100)
    {
        sum += i;
        i++;
    }
    printf("%d", sum);
    return 0;
}

```

2、do-while循环

C语言中的 `do-while` 循环，一般形式如下：

```
do
{
    执行代码块;
}while(表达式);    //注意：这里有分号!!!
```

do-while循环语句的语义是：

它先执行循环中的执行代码块，然后再判断while中表达式是否为真，如果为真则继续循环；如果为假，则终止循环。因此，**do-while循环至少要执行一次循环语句。**

注：while括号后必须有分号。

从1加到100，eg：

```
#include <stdio.h>
int main()
{
    int i = 1, sum = 0;
    do
    {
        sum += i;
        i++;
    }while(i <= 100);
    printf("%d", sum);
    return 0;
}
```

3、for循环

c语言中**for循环**一般形式：

```
for(表达式1;表达式2;表达式3)
{
    执行代码块;
}
```

它的执行过程如下：

1. 执行表达式1，对循环变量做初始化；
2. 判断表达式2，若其值为**真(非0)**，则执行for循环体中执行代码块，然后向下执行；若其值为**假(0)**，则结束循环；
3. 执行表达式3，(i++)等对于循环变量进行操作的语句；
4. 循环结束，程序继续向下执行。

注：for循环中的两个分号一定要写。

for循环中三个表达式：

- 表达式1是一个或多个**赋值语句**，它用来控制变量的**初始值**
- 表达式2是一个**关系表达式**，它决定什么时候退出循环
- 表达式3是**循环变量的步进值**，定义控制循环变量每循环一次后按什么方式变化
- 这三部分之间用分号 ; 分开

从1加到100, eg:

```
#include <stdio.h>
int main()
{
    int i = 1, sum = 0;
    for(; i <= 100; i++)
    {
        sum+=i;
    }
    printf("%d", sum);
    return 0;
}
```

注：

- 1、for循环中的“表达式1、2、3”均可不写为空，但两个分号 (;) 不能缺省。
- 2、省略“表达式1 (循环变量赋初值) ”，表示不对循环变量赋初始值。
- 3、省略“表达式2(循环条件)”，不做其它处理，循环一直执行（死循环）。
- 4、省略“表达式3(循环变量增减量)”，不做其他处理，循环一直执行（死循环）。
- 5、表达式1可以是设置循环变量的初值的赋值表达式，也可以是其他表达式。
- 6、表达式1和表达式3可以是一个简单表达式也可以是多个表达式以逗号分割。

循环示例1：打印三角形星星堆

```
#include <stdio.h>
int main()
{
    int i, j, k;
    for(i=1; i<5; i++)
    {
        /* 观察每行的空格数量，补全循环条件 */
        for(j=i; j<5; j++)
        {
            printf(" ");    //输出空格
        }
        /* 观察每行*号的数量，补全循环条件 */
        for( k=0; k<2*i-1; k++)
        {
            printf("*");    //每行输出的*号
        }
    }
}
```



```

    }
    printf("\n");    //每次循环换行
}
return 0;
}

```

循环示例2：打印9x9乘法表

```

#include <stdio.h>
int main()
{
    // 定义相乘数字i,j以及结果result
    int i, j, result;
    for(i=9;i>=1;i--)
    {
        for(j=1;j<=i;j++)
        {
            printf("%d*%d=%d ",i,j,result=i*j);
        }
        printf("\n");
    }
    return 0;
}

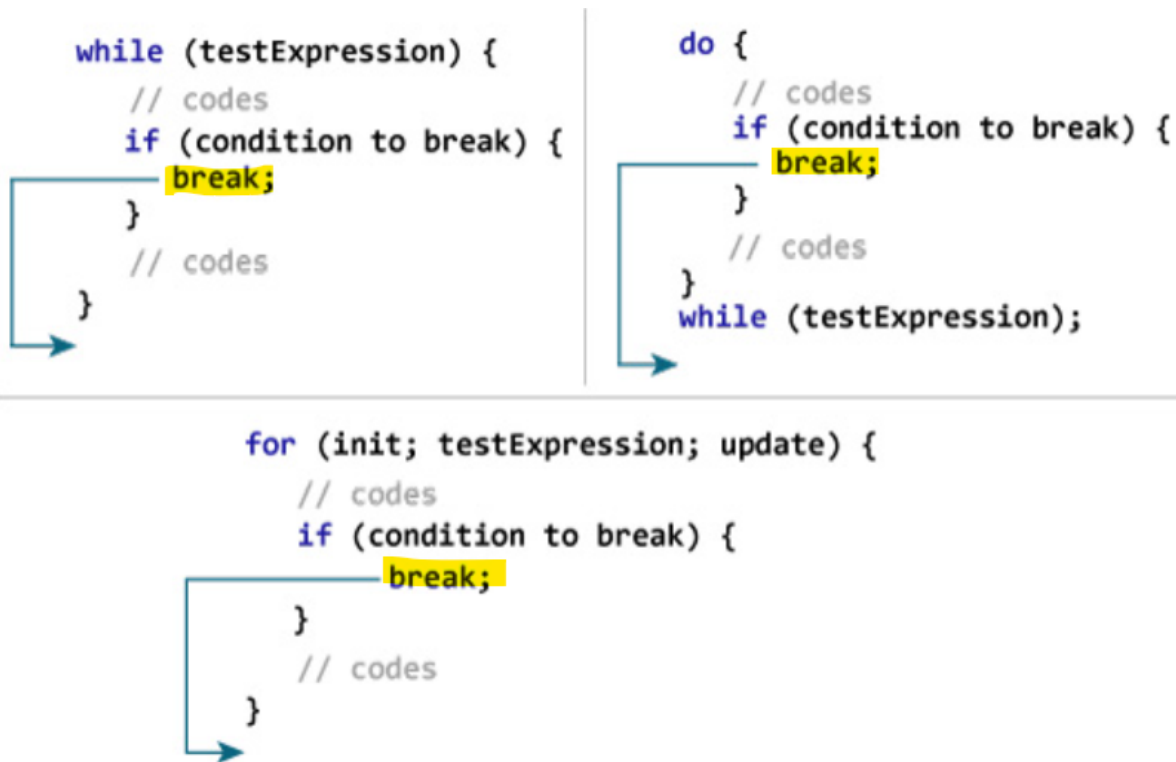
```

4、结束语句——break语句

在C语言中，可以使用 `break` 语句进行结束循环操作。

使用 `break` 语句时注意以下几点：

1. 在没有循环结构的情况下，`break`不能用在单独的if-else语句中。
2. 在多层循环中，一个`break`语句只跳出当前循环。



示例:

```
#include <stdio.h>
int main ()
{
    int a = 10;
    while( a < 20 )
    {
        printf("a 的值: %d\n", a);
        a++;
        if( a > 15)
        {
            /* 使用 break 语句终止循环 */
            break;
        }
    }
    return 0;
}
```

最后结果:

```
C:\Users\12506\Desktop\Untitled1.exe
a 的值: 10
a 的值: 11
a 的值: 12
a 的值: 13
a 的值: 14
a 的值: 15

Process returned 0 (0x0)    execution time : 0.846 s
Press any key to continue.
```

5、结束语句——continue语句

continue语句的作用是结束本次循环开始执行下一次循环。

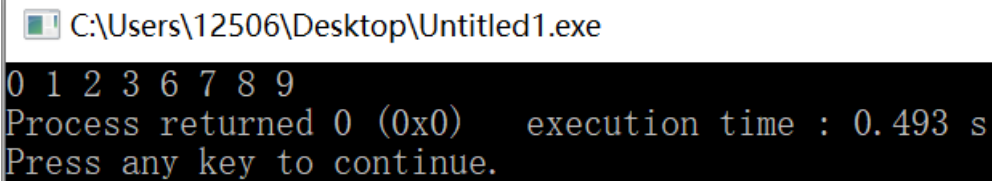
break语句与continue语句的区别是:

break是跳出当前整个循环，continue是结束本次循环开始下一次循环。

continue举例:

```
#include <stdio.h>
int main()
{
    int a;
    for(a = 0; a <= 9; a++)//a等于9结束循环
    {
        if(a == 4 || a == 5) continue; //如果a==4,则跳过当次循环,进入下次循环
        printf("%d ", a);
    }
    return 0;
}
```

结果为:



```
C:\Users\12506\Desktop\Untitled1.exe
0 1 2 3 6 7 8 9
Process returned 0 (0x0) execution time : 0.493 s
Press any key to continue.
```

五、函数

1、概念

函数是一组一起执行一个任务的语句。我们可以把代码划分到不同的函数中。函数**声明**告诉编译器函数的名称、返回类型和参数。函数**定义**提供了函数的实际主体。C语言提供了大量的库函数，例如，函数 **strcat()** 用来连接两个字符串，函数 **memcpy()** 用来复制内存到另一个位置。

2、自定义函数

一般形式：

```
返回类型 函数名称 (参数)           //函数头
{                                   //函数主体
    执行代码块;
    return (表达式);
}
```

在 C 语言中，函数由一个函数头和一个函数主体组成。下面列出一个函数的所有组成部分：

- **返回类型**：一个函数可以返回一个值，关键字默认为**int**。有些函数执行所需的操作而不返回值，在这种情况下，关键字 **void**。
- **函数名称**：这是函数的实际名称，建议取能代表函数作用的名称，方便他人理解。
- **参数**：当函数被调用时，您向参数传递一个值，这个值被称为实际参数。
- **函数主体**：函数主体包含一组定义函数执行任务的语句。

eg：来看看**max()** 函数的源代码。该函数有两个参数 num1 和 num2，会返回这两个数中较大的那个数：

```
/* 函数返回两个数中较大的那个数 */
int max(int num1, int num2)
{
    /* 局部变量声明 */
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

注：

1. 参数省略表示该函数是**无参函数**，参数不省略表示该函数是**有参函数**；
2. 函数名称遵循**标识符命名规范**；
3. 自定义函数尽量放在 **main** 函数之前，如果要**放在main函数后面的话**，需要在main函数之前**先声明**自定义函数，声明格式为：

```
[数据类型说明] 函数名称 ([参数]);
```

3、函数声明

函数**声明**会告诉编译器函数名称及如何调用函数。

针对上面定义的函数 max()，它的函数声明像这样：

```
int max(int num1, int num2);
```

4、调用函数

调用函数时，传递所需参数，如果函数返回一个值，则可以存储返回值。

```
#include <stdio.h>

/* 函数声明 */
int max(int num1, int num2);

int main ()
{
    /* 局部变量定义 */
    int a = 100;
    int b = 200;
    int ret;

    /* 调用函数来获取最大值 */
    ret = max(a, b);

    printf( "Max value is : %d\n", ret );

    return 0;
}

/* 函数返回两个数中较大的那个数 */
int max(int num1, int num2)
{
    /* 局部变量声明 */
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

5、函数参数

- 实际参数（实参）
- 形式参数（形参）

(1) 实际参数

是在调用时传递给函数的参数。实参可以是常量、变量、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值，以便把这些值传送给形参。

(2) 形式参数

形参就像函数内的其他局部变量，在进入函数时被创建，退出函数时被销毁。在调用函数时，实参将赋值给形参。因而，必须注意实参的个数，类型应与形参——对应。

比较实参和形参：

- 形参只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只有在函数内部有效。

函数调用结束返回主调函数后则不能再使用该形参变量。

- 实参可以是常量、变量、表达式、函数等。

无论实参是何种类型的量，在进行函数调用时，它们都必须具有**确定的值**，以便把这些值传送给形参。因此应预先用赋值等方法使实参获得确定值。

注：

在参数传递时，实参和形参在数量上，类型上，顺序上应严格一致，否则会发生**类型不匹配**的错误。

6、函数调用练习——递归函数

概念：递归就是一个函数在它的**函数体内调用它自身**。

执行递归函数将反复调用其自身，每调用一次就进入新的一层。

注意递归函数必须有结束条件

eg：有5个人坐在一起，问第5个人多少岁？他说比第4个人大2岁。问第4个人岁数，他说比第3个人大2岁。问第3个人，又说比第2人大两岁。问第2个人，说比第1个人大两岁。最后问第1个人， he 说是10岁。请问第5个人多大？

程序分析：

利用递归的方法，递归分为回推和递推两个阶段。要想知道第5个人岁数，需知道第4人的岁数，依次类推，推到第1人（10岁），再往回推。

```
#include <stdio.h>
int age(int n)
{
    return n == 1 ? 10 : age(n - 1) + 2;
}
int main()
{
    printf("第5个人的年龄是%d岁", age(5));
    return 0;
}
```

六、数组

1、什么是数组

程序中也需要容器，只不过该容器有点特殊，它在程序中是一块**连续的，大小固定并且里面的数据类型一致的内存空间**，它还有个好听的名字叫数组。可以将数组理解为大小固定，所放物品为同类的一个收纳盒，在该收纳盒中的物品是按一定顺序放置的。

2、声明一维数组

```
数据类型 数组名称[长度];
```

eg:

```
int a[3];
```

3、一维数组初始化

数组不能只声明，来看一下数组是如何初始化的。说到初始化，C语言中的数组初始化是有三种形式的，分别是：

1. 数据类型 数组名称[长度n] = {元素1,元素2...元素n};
2. 数据类型 数组名称[] = {元素1,元素2...元素n};
3. 数据类型 数组名称[长度n];
 数组名称[0] = 元素1;
 数组名称[1] = 元素2;
 数组名称[n-1] = 元素n;

eg:

```
int a[3] = {1,2,3};
```

其中，a[0]就是元素1，a[1]为元素2，a[2]为元素3；

注：

1. 数组的下标均以**0开始**；
2. 数组在初始化的时候，数组内元素的个数不能大于声明的数组长度；
3. 如果采用第一种初始化方式，**元素个数小于数组的长度时，多余的数组元素初始化为0**；

4、一维数组的遍历

数组就可以采用循环的方式将每个元素遍历出来，而不用人为的每次获取指定某个位置上的元素。

例如我们用for循环遍历一个数组：

```
int a[3] = {1,2,3};
int i;
for (i = 0; i < 3; i++)
{
    printf("%d\n",a[i]);
}
```

注:

1. 避免出现数组越界访问，循环变量不要超出数组的长度。
2. C语言的数组长度一经声明，长度就是固定，无法改变。

5、一维数组作为函数参数

数组可以由整个数组当作函数的参数，也可以由数组中的某个元素当作函数的参数。

(1) 整个数组当作函数参数，即把数组名称传入函数，例如：

```
#include <stdio.h>
void temp(int arr[])
{
    int i;
    for(i = 0; i < 5; i++)
    {
        printf("%d\n", arr[i]);
    }
}
int main()
{
    int arr[5] = {1,2,3,4,5};
    temp(arr);
    return 0;
}
```

(2) 数组中的元素当作函数参数，即把数组中的参数传入函数中，例如：

```
#include <stdio.h>
void temp(int arrValue)
{
    printf("%d\n", arrValue);
}
int main()
{
    int arr[5] = {1,2,3,4,5};
    temp(arr[3]);
    return 0;
}
```

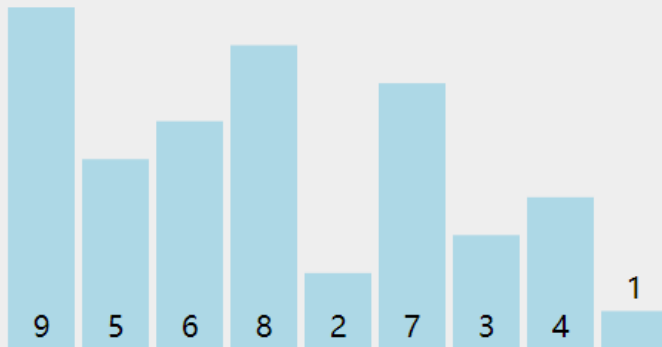
数组作为函数参数时注意以下事项:

1. 数组名作为函数实参传递时，函数定义处作为接收参数的数组类型形参既可以指定长度也可以不指定长度。
2. 数组元素作为函数实参传递时，数组元素类型必须与形参数据类型一致。

6、一维数组应用

【冒泡排序】

以升序排序为例冒泡排序的思想：相邻元素两两比较，将较大的数字放在后面，直到将所有数字全部排序。就像小学排队时按大小个排一样，将一个同学拉出来和后面的比比，如果高就放后面，一直把队伍排好。



```
#include <stdio.h>
int main()
{
    double arr[]={1.78, 1.77, 1.82, 1.79, 1.85, 1.75, 1.86, 1.77, 1.81, 1.80};
    int i,j;
    printf("\n*****排队前*****\n");
    for(i=0;i<10;i++)
    {
        if(i != 9)
            printf("%1.2f, ", arr[i]); // %1.2f表示小数点前一位，小数点后精确到两位
        else
            printf("%1.2f", arr[i]); // %1.2f表示小数点前一位，小数点后精确到两位
    }
    for(i=8; i>=0; i--)
    {
        for(j=0;j<=i;j++)
        {
            if( arr[j]>arr[j+1]) //当前面的数比后面的数大时
            {
                double temp; //定义临时变量temp
                temp=arr[j]; //将前面的数赋值给temp
                arr[j]=arr[j+1]; //前后之数颠倒位置
                arr[j+1]=temp; //将较大的数放在后面
            }
        }
    }
    printf("\n*****排队后*****\n");
    for(i=0;i<10;i++)
    {
        if(i != 9)
            printf("%1.2f, ", arr[i]); // %1.2f表示小数点前一位，小数点后精确到两位
        else
            printf("%1.2f", arr[i]); // %1.2f表示小数点前一位，小数点后精确到两位
    }
}
```

```
        printf("%1.2f", arr[i]);    // %1.2f表示小数点前一位，小数点后精确到两位
    }
    return 0;
}
```

课后作业

Level 0

编写一个C语言程序，输入a,b,c三个值，输出其最大值。

Level 1

写一个判断素数的函数，在主函数输入一个整数，输出是否是素数的消息。

输入

一个数

输出

如果是素数输出prime，如果不是输出not prime

样例输入

97

样例输出

prime

Level 3

求一个3x3矩阵对角线元素之和。

输入

矩阵

输出

主对角线 副对角线 元素和

样例输入

1 2 3
1 1 1
3 2 1

样例输出

3 7

