C 语言总结

wybuhui
wybuhui@linuxstory.org

目录

1 C 初识

1.1 引言

C 语言的重要我在这里也就不做过多的讲述了，本文档重在 C 语言的知识总结，所以，就直接讲重点好了。本文档是作者本人以个人的经验进行分类整理的，如有建议请联系作者邮箱。

1.2 导入

相信看到本文档的读者都是学过数学的，所以，我们这里就从数学引入吧。

$$y = ax^2 + bx + c \tag{1}$$

相信大家都认识上面这个数学函数，是一个简单的一元二次函数，毫无争议。其中的 x 我们称之为自变量，而 y 我们称之为因变量，相应的，a, b, c 我们都称为常数。当我们的函数不是特指，而是一系列很多个二次函数的时候，也就是说当我们上面的函数是一个通式的时候，我们的每一个二次函数的 a, b, c 对应的值也都不同。这个时候，我们就说 a, b 是一个常变量，而我们的 c 因为是一个常数，和自变量 x 没有任何关系，所以一般都不太考虑，我们一般都是把 c 认为是一个固定的常数而用字符 c 代替。在这里，我们就把 c 称为常量。

好了，到了这里就到了我们的重点所在了。在 c 语言中，我们也存在着各种各样的变量，用来表示数字，字符等等。而作为最古老的高级编程语言，c 语言在这个方面尤其丰富和完善。当然，也就是详细和底层。

2 C 基础

2.1 变量

在 C 语言中有如下变量 (Variables)：

表 1: Variables in C language

Type	Explanation	Format specifier
char	计算机中表示字符的最小单元。这个类型在计算机中存储的类型实际是整型。可以是 signed 或者 unsigned 类型。	%c
signed char	同样是字符型，不过是有符号位。表示的大小范围在 [−127, +127] 内。	%c (or use %hhi for a numerical output)
unsigned char	同样是字符型，不过是无符号位。表示的大小范围在 [0, 255] 内。	%c (or use %hhu for a numerical output)
short short int signed short signed short int	有符号短整型，表示的大小范围在 [−32,767, +32,767] 内; 一般最小有 16 位。	%hi

Variables in C language

unsigned short unsigned short int	有符号短整型，表示的大小范围在 [0, 65,535] 内。	%hu
long long int signed long signed long int	有符号长整型，表示大小范围在 [−2,147,483,647, +2,147,483,647] 内。一般有 32 位大小。	%li
unsigned long unsigned long int	无符号长整型，表示大小范围在 [0, 4,294,967,295] 内;	%lu
long long long long int signed long long signed long long int	有符号长长整型，表示大小范围在 [−9,223,372,036,854,775,807, +9,223,372,036,854,775,807] 内; 一般至少有 64 位大小，只在 C99 版本及以后中使用。	%lli
unsigned long long unsigned long long int	无符号长整型，表示的大小范围在 [0, +18,446,744,073,709,551,615] 内; 只在 C99 版本及以后中使用。	%llu
float	浮点型类型，通常是单精度浮点型，实际上是没有准确定义的。在大多数系统上是使用的 IEEE 754 的单精度浮点型（32 位）格式。	%f %F for digital notation, or %g %G, or %e %E %a %A for scientific notation

以上就是我们在 C 语言中所有的变量类型了，可能你的书上没有列出来，但是，相信我，你的书写的太浅薄了。一般我们常用的类型只有：int, char, long int, float 这几个。具体别的呢，大家可以参考上面的解释吧。

这里呢，我给大家看一段代码，进行详细解释一下。

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{

        int integer = 10; // define a integer
```

```
6         char a = 's';           // define a char
7
8         // print  variables
9         printf("%d : %c\n", integer, a);
10
11        return 0;
12  }
```

这里，我就给大家详细介绍一下我们上面的代码[1]。代码第一行，是我们的 C 语言的库文件包含，也就是我们通常所听到的头文件。在这里呢，就是我们 C 语言的标准库。只有写上我们的这句代码，我们下面写的代码调用的代码才会有用。换句话说，我们 C 语言一般用到的东西都在这个库文件中进行了定义说明。第二行代码是我们的 C 语言框架，也是我们 C 语言程序执行代码开始的地方。这句代码和第十一行的代码相对应，具体为什么这样写，大家可以参考我下面函数 (3.5) 部分。这里就不再进行过多的介绍。在我们代码的第四和第五行分别定义了两个变量，分别是整型和字符型。后面的则是笔者添加的注释[2] 。在我们代码中的第九行调用了我们的 printf 函数把我们的变量内容进行输出。可能细心的读者就会发现在函数中有一些字符和本文档上面的表格的内容有关联了，这个函数的使用方法后续再进行讲解，这里大家就先认识一下吧。

在这里呢，大家就先记住上面这个模板吧。中间 5-9 行是我们自己的代码。如果你想写自己的代码，就写在这一块吧。

2.2 静态变量

我们在前面介绍了变量，这里呢，就给大家介绍一下静态变量，静态变量大家可以理解为公式 (1) 中的 a 或者 b。

在这里因为静态变量不是特别常用，所以这里就不过多的进行介绍了。当我们声明静态变量的时候，就用 static 关键词。类似与上面我们讲解的变量，不同的是在变量前加上这个关键词。

2.3 常量

常量则是我们前面介绍的公式 (1) 中的 c ，换句话说，也就是一般我们数学中的确定的一个实数，不过我们在程序中给它取了一个外号。当声明常量的时候，我们使用 const 这个关键词。

Δ：常量已经声明，在程序中就不允许再次改变。

2.4 printf() 函数

printf() 函数是我们 C 语言中很常用的函数，我这里就提前给读者进行一些基本的讲解。

[1] C 语言中所有的代码语句是以分号为结束，一般没有特殊说明的话，符号都是英文的

[2] C 语言中的注释有单行注释符号：// 和多行注释符号/* 注释内容，可以多行 */。

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{
        /*              #include <stdio.h>
         *      int printf(const char *format, ...);
         *
         *      The functions printf() write output to stdout under format
         *      string that specifies how subsequent arguments are converted
         *      for ouput.
         *
         *      The arguments must correspond properly with the conversion
         *      specifier. By default,the arguments are used in the order
         *       given,
         */

        printf("test\n");

        return 0;
}
```

在上面的代码中，5-13 行是笔者写的代码注释，有兴趣的读者可以多去学习一下第一手的资料。这里呢，我就以注释为例给大家进行一下讲解。首先我们 C 语言的框架我就不说了。我们代码的第 15 行是我们整个程序的主体。当然了，就只有一行代码，就是调用 printf() 函数进行输出。当我们调用这个函数时候，毫无疑问的是首先要写出是函数的名字，后面紧跟一对圆括号，括号里面就是我们传给该函数的参数。这里呢，我们传的参数是用双引号括起来的一串字符。表示我们要程序把双引号中的东西输出来。

上面的代码很简单。然后我们开始看我们的注释的地方，这里讲的更加详细一些。首先，是我们的函数库的说明，每次我们调用这个函数都要包含头文件 stdio.h 这个库文件。然后是我们函数的定义。首先是 int ，表示我们函数的返回值是 int 类型。然后函数名和括号里面的参数。参数第一个是我们之前讲过的 char 类型。逗号表示分隔，最后面的省略号表示参数不缺定，可以有很多个。下面是一些进一步的说明文字，这里就不再过多介绍。

在这里需要大家注意的地方是：我们调用函数时候传进去的参数，我们只传进去了双引号括起来的一串字符，而这一串字符中，还有一个反斜杠和一个字符 n 。这个就表示我们输出字符的时候进行换行。而当我们输出我们前面学的变量的值的时候，我们该怎么输出呢？

在这里我就直接写出来了，读者只需要想一下，看看自己是否是真的掌握了。

这里我给大家写了一个简单的例子。

```c
int var = 10;
```

```
printf(" variable is : %d", var);
```

我们程序运行结果是：

variable is : 10

通过观察上面的字符大家可以看到，我们的%d 在这里表示的是我们的 int 类型的变量。如果读者还有印象的话，应该就记得我之前将变量的那个表格最后一列中对应的字符和这个是一样的。而我们上面的表格最后一列的作用，也确实就是这个用处。说到这里，相信大家都明白 printf 函数该怎么用了吧。因为我们的参数个数是不确定的，所以我们参数个数理论上来说没有限制，可以往里面传进去任意多个，只要我们在第一个双引号格式化中添加上我们变量对应的表示符号，后面就可以以逗号为分隔符进行传参了。

因为我们 printf 函数还有一些格式化字符输出的符号，我这里整理了一下，大家可以参考一下：

表 2: printf format

%e	科学计数
%s	字符串
%o	无符号八进制
%x	十六进制
%X	大写十六进制
\t	输出一个制表符号
\n	输出一个回车符号

2.5 scanf() 函数

说过了 printf() 这个输出函数，我们这里就顺便说一下输入函数吧。毕竟是好事成双嘛不是。老规矩，先看一下我们的代码：

```
#include <stdio.h>

int main(int argc, const char *argv[])
{
        /*      #include <stdio.h>
         *
         *      int scanf(const char *format, ...);
         */
```

```
9
10          int var = 0;
11
12          printf("Please input a integer:\n");
13
14          scanf("%d", &var);
15          printf("var = %d\n", var);
16
17          return 0;
18  }
```

运行结果是:

```
1           Please input a integer:
2           23
3           var = 23
```

这里我们来看一下我们的代码。首先我们的代码的框架就不再多说了，然后紧接着就是我们的注释。老规矩，首先是头文件包含，然后是我们函数的源定义。基本上和我们之前讲过的 printf 函数类似。然后就是我们主要的代码。第一个是我们定义了一个变量，然后是我们调用 printf 函数进行提示信息的输出。紧接着就是我们这里介绍的函数 scanf 函数。这里大家仔细看一下，首先是我们字符的格式，和 printf 函数一样，然后是我们的变量。不过和 printf 函数不同的地方是，在变量前面有一个 & 符号，这个符号在这里表示取地址。这里大家先这样记住吧，后面函数部分进行详细介绍。

这里就需要大家要注意了，我们 scanf 函数和 printf 函数唯一的区别的地方就是这里，我们传的变量参数是地址。

2.6 运算符

既然我们从数学入手，而我们程序代码更多的时候给人们的印象就是计算，那当然少不了我们的计算方面的运算符。

在我们的数学中，相信大家都已经很熟悉了，我们代码中也基本上是一样的。我这里就做一个简单的总结吧。

表 3: C 运算符

+ -	加减	*	乘
/	除	%	求余数

好了，目前呢，我们基本上算是入门了。目前的话读者还可以看得懂，下面我就做一个整体的比较全的总结。看不懂没有关系，之后慢慢懂了再会头看也可以，或者当作一个参考也行。话不多说，先给图：

表 4: Operators in C language

操作符名		操作符	语法	C++ 是否相同	是否包含在 C 中
Arithmetic Operators					
Basic assignment		=	a = b	Yes	Yes
Addition		+	a + b	Yes	Yes
Subtraction		-	a - b	Yes	Yes
正		+	+a	Yes	Yes
负		-	-a	Yes	Yes
Multiplication		*	a * b	Yes	Yes
Division		/	a / b	Yes	Yes
Modulo (integer remainder)		%	a % b	Yes	Yes
Increment	Prefix	++	++a	Yes	Yes
	Postfix	++	a++	Yes	Yes
Decrement	Prefix	– –	– –a	Yes	Yes
	Postfix	– –	a– –	Yes	Yes
Comparison operators/relational operators					
Equal to		==	a == b	Yes	Yes
Not equal to		!=	a != b	Yes	Yes
Greater than		>	a > b	Yes	Yes
Less than		<	a < b	Yes	Yes
Greater than or equal to		>=	a >= b	Yes	Yes
Less than or equal to		<=	a <= b	Yes	Yes
Logical operators					
Logical negation (NOT)		! not	!a not a	Yes	Yes
Logical AND		&& and	a && b a and b	Yes	Yes
Logical OR		|| or	a || b a or b	Yes	Yes
Bitwise operators(位操作符)					
Bitwise NOT		~	~a	Yes	Yes
Bitwise AND		&	a & b	Yes	Yes
Bitwise OR		|	a | b	Yes	Yes
Bitwise XOR		^	a ^ b	Yes	Yes
Bitwise left shift		<<	a << b	Yes	Yes
Bitwise right shift		>>	a >> b	Yes	Yes
Compound assignment operators					

Operators in C language

Name	Symbols	Syntax	Meaning	Included in C++	Included in C
Addition assignment	+=	a += b	a = a + b	Yes	Yes
Subtraction assignment	-=	a -= b	a = a - b	Yes	Yes
Multiplication assignment	*=	a *= b	a = a * b	Yes	Yes
Division assignment	/=	a /= b	a = a / b	Yes	Yes
Modulo assignment	%=	a %= b	a = a % b	Yes	Yes
Bitwise AND assignment	&=	a &= b	a = a & b	Yes	Yes
Bitwise OR assignment	|=	a |= b	a = a | b	Yes	Yes
Bitwise XOR assignment	^=	a ^= b	a = a ^ b	Yes	Yes
Bitwise left shift assignment	<<=	a <<= b	a = a << b	Yes	Yes
Bitwise right shift assignment	>>=	a >>= b	a = a >> b	Yes	Yes

好了，以上基本上就是我们 C 语言中所有的操作符了。还有几个，这里就不列出来了，随着下面的学习大家再慢慢掌握吧。在这里给大家解释一下，一本笔者会给大家适当的引入一些英语，因为写代码是需要会一些英文的。当然了，我也会对大家比较陌生的单词进行注释，如果没有注释的话，那就是基本上认为你适合掌握的。基本上说到操作符号就必须给大家提一下笔者认为很有用的一个操作符。上面的操作符基本上都是**双目运算符**，而这个操作符则是**三目操作符**：

”? :” 如果只是看这个字符或许大家还看不懂，我这里就给大家写一段代码，让大家体会一下。

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{

        int a = 3;

        a = a>4 ? 5 : 6; // if a>4 then a = 5 else a = 6
        printf("a = %d\n", a);

        return 0;
}
```

在这里呢，我们这个小程序的代码输出结果是 “a = 6”，通过这段代码和上面的注释相信大家就很熟悉这个操作符的作用了吧。这里就不做过多的介绍了。

2.7 逻辑结构

经过我们前面的介绍，相信大家已经对 C 语言有了一个初步的认识了。在这一小节中我们就算是 C 语言的真正入门了。本小节，就给大家介绍一下我们程序语言中的流程控制语句。

2.7.1 if 语句

对于 if 语句呢，这个就是程序中最基本的语句了。这个语句就和我们经常开玩笑时候说的话类似的：你不是傻是什么？虽然这个是平常我们和朋友开玩笑的话，但是基本逻辑还是一样的。我们先来看看我们的代码语句：

```
        if( 判断条件 )
        {
                put your code here;
        }
        else
        {
                put your code here for other conditions;
        }
```

这里我们的举例就是这样，很简单，括号里面是判断条件，else 后面就是对应的不满足判断条件的时候执行的代码。而判断的条件怎么写，就需要大家去看我在前面放的表格 4 中的 Comparison Operators/relational Operators 那一个模块。在这里，当然我们的 if 语句也有扩展，我们这里就给大家再看一下我们 if 语句的扩展形式。

```
        if( 判断条件1 )
        {
                put your code here;
        }
        elif( 判断条件2 )
        {
                put your code here;
        }
        ...
        ...
        ...
        else
        {
                put your code here for other conditions;
        }
```

中间的省略号就是表示很多，就不再列举了。当然了，如果你愿意，当然想写多少就可以这样写多少了。

2.7.2 for 语句

在 for 语句中呢，这个就是属于循环结构了，下面的也都是一样的。除了最后一个是分支语句。好了，我们直接上代码讲解。

```
        for( 初始条件; 判断条件; 每次执行改变量)
        {
                put your code here;
        }
```

在我们写代码中，初始条件相信大家应该没有什么问题吧。判断条件就是和上面的 if 语句一样的。每次执行改变量就是我们每次经过一次循环我们需要改变的地方，只有这样我们才能不会让计算机一直在循环里面反反复复，才可以最后退出循环最后停止执行。说了这么多，可能读者还是不是很明白，我们直接上代码讲解。

```
#include <stdio.h>

int main(int argc, const char *argv[])
{

        for( int i=0; i < 10; i++ )
        {
                printf("i = %d\n", i);
        }

        return 0;
}
```

代码执行的结果是：

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
```

相信经过上面的代码大家应该会有很好的理解了吧。如果对上面我们的代码有疑问的地方，我估计也就是 for 语句最后一个参数吧。至于是什么，大家就参考我之前整理的表格4吧。

2.7.3 while 语句

对于 while 语句呢，其实是和我们之前的 for 语句是类似的，同样的都是循环语句，不过还是有一点点的不同。我们先看一般使用 while 语句的格式。

```
while( 判断条件 )
{
        put your code here;
}
```

和我们 for 语句一样，也是判断条件符合的时候才会执行，不符合的时候就跳出循环。和 for 语句不同的是，这个语句是把初始条件放在了前面，而每次执行改变量则是放在我们花括号里面的代码的地方。这就是 while 语句和 for 不同的地方吧。其他的都和 for 语句相同的。

2.7.4 do while 语句

说到 do while 语句，额，先不多说，先给格式给大家瞅瞅：

```
do{
        put your code here;

}while( 判断条件 );
```

把 do while 语句拿过来和我们的 while 语句相比一下，大家应该就可以看出来有什么区别了。while 语句是把判断条件放到前面，而我们的 do while 语句是放到后面，这样次序经过改变后，会有什么区别？这个问题就先留给读者自己思考一下啦。我们这里就不再过多介绍了。

2.7.5 switch 语句

说到 switch 语句，这个语句还是有很大作用的。我们先看看它的格式：

```
switch( 变量 )
{
        case 变量值1:
                put your code ;
                break;
        case 变量值2:
                put your code ;
                break;
        default:
                put your code ;
}
```

在这里我们就需要详细介绍一下这个语句了。圆括号里的变量是我们需要判断的对象，case 后面和冒号之间的是我们判断变量是否符合的值，只有当我们判断的变量符合我们 case 中的变量值的时候，才会执行我们 case 语句后面的代码。而我们代码的最后要记得放上后面的 break; 这几个字符，表示执行完我们的代码后就跳出我们的 switch 语句。不然就会从我们变量值和我们变量相等的地方开始执行，一直到最后整个语句结束，这样的话就完全没有这个语句该有的作用了。而在这个语句最后的地方呢，有一个 default 分支，因为当我们不能把所有的情况列举出来的时候，它就表示没有列举出来的所有情况。这样，到这里，相信大家已经对这个语句有很深刻的认识了吧。

在这里呢，我们给大家再看一下 switch 的一个示例程序：

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{

        int test = 3;

        switch(test)
        {
                case 1:
                        printf("test1 = %d\n", test);
                        break;

                case 2:
                        printf("test2 = %d\n", test);
                        break;

                case 3:
                        printf("test3 = %d\n", test);
                        break;

                default:
                        printf("test\n");
        }

        return 0;
}
```

代码的执行结果是：

```
                test3 = 3
```

3 C 进阶

3.1 枚举

枚举类型就是给我们的一些常量起了一个可以让我们一眼看出来有什么作用的类型。不过这个类型允许我们去定义一些我们自己的约定。这个类型声明的关键字是:(enum) 。声明的方式如下:

```
                enum colors { Red, Green, Blue = 5, Yellow };
                enum colors{
                        Red,
                        green,
                        Blue = 5,
                        Yellow
                        };
```

在我们枚举类型的定义中，第一个是我们的关键字，空格，然后是我们的枚举变量的名字，然后是用花括号括起来的常量，这些常量我们随意取一个名字，中间用逗号进行分隔。C 语言的编译器默认分配的编号是从 0 开始，依次往后递增 1 。当然了，C 语言中也允许我们进行特别说明，就比如我上面的例子中，在 Blue 这个常量就给予了特殊说明，这样的话，该常量就会被分配给 5 这个常数。然后在这个常量后面的常量是在 5 这个基础上再依次递增 1 的。在我们枚举类型中的所有常量，在程序中实际存储中是以 int 类型存储的。

在我们的示例中我们给大家写了两种不同风格的代码。效果都是一样。在我们 C 语言代码中，任何东西都是先给出说明，然后再进行调用的。既然我们已经实现了枚举类型的定义，那我们该怎么去进行调用呢？下面就给大家一个示例先看看。

```
#include <stdio.h>

int main(int argc, const char *argv[])
{
        enum week { Monday,
                                Tuesday,
                                Wednesday,
                                Thursday,
                                Friday,
                                Saturday,
                                Sunday };

        enum week day;
```

```
14
15          printf("Please input a day:\n");
16
17          scanf("%d", &day );
18          switch(day)
19          {
20                  case Monday:
21                          printf("Today is Monday\n");
22                          break;
23
24                  case Tuesday:
25                          printf("Today is Tuesday\n");
26                          break;
27
28                  case Wednesday:
29                          printf("Today is Wednesday\n");
30                          break;
31
32                  case Thursday:
33                          printf("Today is Thursday\n");
34                          break;
35
36                  case Friday:
37                          printf("Today is Friday\n");
38                          break;
39
40                  case Saturday:
41                          printf("Today is Saturday\n");
42                          break;
43
44                  case Sunday:
45                          printf("Today is Sunday\n");
46                          break;
47
48                  default:
49                          printf("Input Error!\n");
50                          break;
51          }
52
53
54          return 0;
55  }
```

运行结果是:

```
Please input a day:
3
Today is Thursday
```

相信通过上面的代码大家就明白了该怎么定义枚举类型的一个变量并怎么去使用吧。在看上面这个代码的时候，大家有没有觉得代码一眼就能看出来代码是在做什么的呢？当然了，这可不是笔者的自恋啊。只是笔者在这里给大家一点建议，大家平常写代码的时候一定要注意代码的缩进排版以及代码变量的名字取名问题，尽量做到一眼就可以看出来自己代码是什么作用的最好。千万不要写完代码过两天回头再来看的时候看不懂自己写的代码哦。

3.2 联合

在我们的 C 语言中，联合的关键词是: (union) 。先给大家看一下我们联合的定义:

```
union <name>
{
        <datatype> <1st variable name>;
        <datatype> <2nd variable name>;
        ...
        ...
        ...
        <datatype> <nth variable name>;
} <union variable name>;
```

以上就是我们的类型的定义了，当然，如果只是定义的话，也可以把花括号后面的都删除，只保留一个分号。我们上面的定义就是首先需要我们联合的关键词，然后是我们这种关键词的名字，紧接着便是我们花括号里面的类型定义。花括号里面的格式是先是类型关键词，然后是我们对应类型的变量名字。最后以分号结尾。而我们类型的关键词，就是我们前面讲的变量，大家可以参考之前的章节 2.1 。而这里花括号里面的变量我们这里有了另外一个名字，叫成员变量。虽然我们上面写了那么多，但是我们这里需要强调一点的是，联合只可以使用它的成员变量中的一个。

到目前为止，我们的定义就搞定了，那么我们接下来怎么使用呢？我们下面就以代码为例进行讲解吧。

```
#include <stdio.h>

int main(int argc, const char *argv[])
{
```

```
5
6           union Data{
7                   int addr;
8                   char var;
9           };
10
11          union Data data;
12          printf("Please input a char to save:\n");
13
14          scanf(" %c", &data.var);
15
16          printf("Data is : %c\n", data.var);
17
18          return 0;
19  }
```

运行结果是：

```
1               Please input a char to save:
2               c
3               Data is : c
```

通过上面的程序，大家应该已经有比较好的体会了吧。在上面的例程中，我们定义了一个 Data 的联合，它里面有两个成员变量，一个是整型，一个是字符型。最后我们用这个联合定义了一个联合的变量，并对它的 var 成员变量进行了赋值以及输出。我们联合只能使用一个成员变量，如果使用了多个，那最后一个会覆盖掉前面所有的。联合的物理存储的形式是多个成员变量共同占用同一块内存。

3.3 数组

好了，通过前面的学习，相信大家都对我们 C 语言中的变量有一定的了解了，也算是有一定的基础了。但是如果我们遇到了大量的数据需要操作呢？我们该怎么做呢？

随着这个问题的出现，我们这里就出现了数组来进行解决这一个问题。首先，数组是什么呢？其实数组就是我们单个的变量进行翻倍，然后进行按顺序依次存储的这么一种结构。数组的话，我们就把单个的变量一次进行多个的处理了。需要注意一点的是，我们的数组只能是同一种类型的变量进行操作。

首先，我们从一维数组开始。先给大家看一下我们数组定义的格式：

```
1   <datatype> name[nums];
```

在我们的格式中，第一个是我们需要说明的我们数组的变量类型。具体可以参考我们之前的章节 (2.1) 。这里就不再多说了。紧随其后的是我们数组的名字，名字后面是一对中括号，括号里面是我们想要定义的变量的个数。这样的话就定义了 nums 个数量的 datatype 类型的一个叫 name 的变量数组。这个数组中，name 可以用来表示存储这个数组的内存的地址。当我们使用的时候，name[index] 用来表示我们这个数组中第 index 的变量。这里需要给大家说明的是，我们数组的 nums 表示我们数组的大小，但是我们数组的第一个元素[1] 的编号是从 0 开始的，一直编号到我们的 nums - 1 总共 nums 个变量。而我们对数组进行操作的时候，只能一个元素一个元素进行操作，不可以一次性对所有元素进行操作。而我们每一个元素就是有我们元素的编号进行寻找的。说了这么多，可能大家有点晕，我们还是上一段代码给大家进行讲解吧。

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{

        int int_array[10];

        for(int i=0; i < 10; i++)
                int_array[i] = i;

        for(int i=0; i < 10; i++)
                printf("array[%d] = %d\n", i, int_array[i]);

        return 0;
}
```

我们的运行结果是：

```
array[0] = 0
array[1] = 1
array[2] = 2
array[3] = 3
array[4] = 4
array[5] = 5
array[6] = 6
array[7] = 7
array[8] = 8
array[9] = 9
```

[1]元素：即数组中的单个的变量

我们现在来给大家分析一下我们的代码。首先我们代码的框架我这里就不再多说了。我们代码的第 6 行代码定义了一个有 10 个元素的整型数组。在我们第 8-9 行，我们用 for 循环给我们的数组进行了赋值，从 0 依次到 9 。下面的 for 循环我们再次调用了 for 循环把我们数组的每一个数值进行了输出，大家通过看我们程序运行的结果就可以体会到了。这里我们需要给大家说的是：首先我们 for 语句的运用，我们 for 语句循环的主体只有一条语句的时候可以这样不用花括号，但是当有多个语句的时候还是需要按照我们之前讲过的需要用花括号。然后就是我们的数组，我们的数组是需要对每一个元素进行操作，而每一个元素的编号，则更加重要。如果我们使用数组的时候没有加上编号，那么我们操作的对象就是这个数组的地址了。这样就会造成系统错误。另外，如果我们给数组元素的编号大于我们定义的大小的时候，系统也不会报错，但是会造成溢出，简单的理解就是会把内存中别的不属于我们数组的内存给覆盖掉，造成别的变量的错误。

好了，到目前为止，我们的数组算是简单入门了。我们数组的类型可以是我们之前提到的变量类型中的任意一种。而我们数组中的元素的存储则是按顺序依次排列的。下面呢，我们就开始介绍我们的二维数组以及多维。

如果说我们的一维是一条直线的话，那么我们二维就是我们的面了。所以我们的二维数组呢，则是按照面的形式来理解的。下面先来看看我们二维数组的定义：

```
<datatype> name[row][col];
```

这里呢，就是我们的二维数组的定义。当我们以面的形式理解二维数组的时候，我们第一个中括号则表示有多少行，而我们第二个中括号则表示每一行有多少列。而我们的不过行还是列，都是从 0 还是排列，和一维数组是一样的。当我们使用其中一个元素的时候，我们就给出这个元素的位置，是以第几行，第几列的形式给出。而多维数组呢，依次类比。这里笔者建议大家结合我们生活或者说我们物理来理解，这样就会更好的理解，比如三维的话就以三维空间的角度来理解……

3.4 结构体

经过我们之前学了那么多的变量、枚举、联合等类型，大家现在对 C 语言有没有什么感觉呢？是不是已经可以上手了？笔者这里呢，也建议大家多动手去写写代码，毕竟学代码就是为了用的嘛。如果没有尝试用，是永远也不会发现问题的。

好了，我们废话就不多说了，下面我们就介绍一下我们的结构体吧。结构体和我们之前的联合类似，不同的地方是我们的结构体是所有的成员变量都有单独的空间存储，每个成员变量都可以使用。我们的声明和联合的类似，不同的地方是我们结构体的关键词是：(struct) 。我们这里就直接给大家一个程序示例吧。

```
#include <stdio.h>

int main(int argc, const char *argv[])
{
        enum sex{
                male,
                female
```

```
8           };
9
10          struct student{
11                  int stu_nu;
12                  enum sex stu_sex;
13                  char stu_name[10];
14          };
15
16          struct student student_wy;
17
18          printf("Please input student number:\n");
19          scanf(" %d", &student_wy.stu_nu);
20
21          printf("Please input student sex:\n");
22          scanf(" %d", &student_wy.stu_sex);
23
24          printf("Please input student name\n");
25          scanf(" %s", student_wy.stu_name);
26
27          printf("\n********************Student infor*****************************\n");
28          printf("Student name: %s \nStudent number: %d \nStudent sex:",
29                                                  student_wy.stu_name, student_wy.stu_nu);
30
31
32          if( student_wy.stu_sex == male )
33                  printf("male\n");
34          else
35                  printf("female\n");
36
37          return 0;
38  }
```

程序运行的结果是：

```
1                   Please input student number:
2                   20152020
3                   Please input student sex:
4                   0
5                   Please input student name
6                   nick
```

```
7
8                 ********************Student infor*****************************
9                 Student name: nick
10                Student number: 20152020
11                Student sex:male
```

通过以上的介绍，相信大家已经比较了解这个结构体的作用了吧。我们例程中定义了一个 student 的结构体并定义了一个对应的变量，然后对我们的变量进行操作。结构体在内存中的存储形式就是把成员变量依次存储，并把一个结构体放到一起存储。总的大小也是这个结构体所有的成员变量的大小之和。当我们调用结构体的成员变量的时候，我们是用 (.) 这个符号进行引用的。从这个个例子中，大家应该就可以看得到，当我们使用结构体的时候，使得我们的程序更具有模块化，更具有可读性和集成性。这里呢，我们就稍微介绍一下我们的例程吧，可能有些读者还是没有完全理解。大家有疑问的地方可能就是我在结构体中的定义的 char 类型的数组了吧。我们可以理解一下，我们数组的作用就是把多个同类型的变量放到一起进行操作。我们这里是多个 char 类型了，当我们一个 char 的时候表示的是单个的字符，那我们多个的时候呢？是不是就可以表示一个单词甚至一个句子？这里需要提醒一下的是，我们每个字符串的结尾会有一个 '\0' 的字符存储。具体详细内容大家可以参考我后续的字符相应章节 (3.7) 。

3.5 函数

这一节呢，我们就给大家介绍一下我们程序代码中最最重要核心的内容，那就是函数。或许很多读者在阅读这本总结的时候都是在怀着这样的疑问：为什么你的代码中 C 语言的框架和我遇到的都不同？为什么你的代码是这样写的？

好了，下面就让我慢慢给大家一一解答。首先我们来先看一下我们函数的定义以及调用。我们函数的定义格式如下：

```
1 <datatype> function_name( <datatype> parameters1, <datatype> parameters2, ... )
2 {
3
4         put your code here;
5
6         return variable;
7 }
```

在上面的格式中，首先就是我们函数的类型，和之前章节 (2.1) 中描述的一样。然后是我们函数的名字，紧接着是我们括号中的参数。我们的参数个数不限，以逗号进行分隔，并以类型之后紧跟参数名字的形式排列。函数下面就是我们的花括号，括号里面就是我们函数实现相应功能的代码的地方。在我们函数里面最后一行，一定是我们的函数的返回值语句，而这个返回就是有关键词 (return) 来进行说明的。在这个关键词后面紧跟的就是我们函数返回的变量。

在这里需要说明的一点是，有一个唯一的一个例外的类型是用关键词 (void) 声明的，该关键词表示的意义就是空类型，如果是变量就是在内存中占一个字节大小，类型没有限制的一个变量。

而在函数中的时候，如果函数类型是 void 类型的话，那这个函数最后面就不再需要 return 语句进行返回了。表示这个函数是没有返回值的。说到这里，或许大家应该就可以理解我们 C 语言的代码为什么会有那么多种形式了。

这里需要说的一点是，任何一个函数，都是先定义在声明的。我们代码执行的顺序是从上往下执行，所以，这就遇到问题了。难道我的函数定义只能写在 main 函数前面吗？当然不是的，还有一个种方法就是：我们把函数的定义放到 main 函数的后面，然后在 main 函数前面声明我们有这个一个函数是定义了的。声明的形式就是把我们函数的第一行放到 main 函数前面，末尾加上分号，表示一条语句。这样之后，我们就可以在 main 函数里面调用我们自己定义的函数了。

说了这么多，老规矩，上代码讲解：

```c
#include <stdio.h>

int my_function1( int parameters );

int my_function2( int parameters )
{
        printf("The parameters you have convert to is : %d\n", parameters);

        return 0;
}


int main(int argc, const char *argv[])
{

        int test = 5;

        test = my_function1( test );
        my_function2( test );


        return 0;
}

int my_function1( int parameters )
{
        printf("The parameters you have convert to is : %d\n", parameters);

        return 0;
}
```

运行结果是：

```
The parameters you have convert to is : 5
The parameters you have convert to is : 0
```

这里代码中我们定义了两个函数，都是把传入的参数输出，很简单。在这段代码中，我们的函数分别在 main 函数的前后进行了定义，最后在 main 函数中进行了调用。首先我们在调用的时候第一次因为我们的变量值是 5 ，然后把这个函数的返回值赋给变量。在我们第一个函数中的返回值由 return 关键字指出返回 0 ，所以我们在第二次调用我们的函数的时候，输出就是 0 了。在这里，我们大家要注意学习一下我们函数定义在 main 函数前后的形式，以及函数最后返回值的使用。

好了，但目前为止，我们函数算是基本讲到了。或许有基础的读者会有些疑问，为什么没有讲指针呢？笔者在这里进行说明一下，指针章节被笔者放到后面了。因为考虑到指针对于对硬件不理解的读者很难理解，就放到了后面进行讲解。当然，指针也是很重要的。毕竟，C 语言我是因为指针的存在才喜欢这个语言的。

3.6 指针

好了，到目前为止，我们算是 C 语言已经基本可以用起来了。基本的编程已经是没有问题的，但是，难道编程就只有这么一点点吗？难道学到这里就可以做出来那些各种神奇的功能吗？答案是肯定的，肯定不是的。学到这里，我们现在只是基本的运算是没有问题的。但是不要忘了，我们还有文件存储以及网络还没有涉及。所以，下面我们就来介绍这些东西。给大家介绍一下，真正的代码编程中，我们用到的编程技术。

在 C 语言中，指针是一个很重要的概念，也算是一个抽象的概念。因为我们计算机中各种存储物质，如硬盘，U 盘，以及内存等。都涉及一个概念，我们是怎么把数据存储进去的。而我们存进去后我们是怎么知道存到哪里了？又该去哪里找我们存储的数据呢？

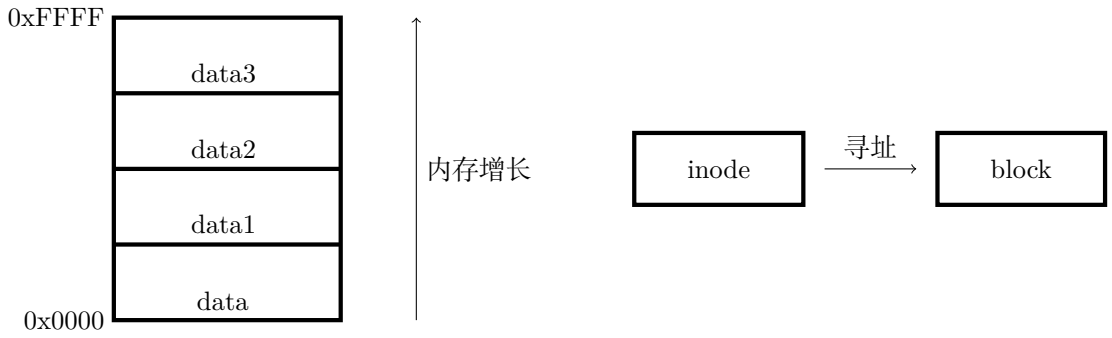

图 1: 内存存储

由上图大家可以看到，我们的存储设备是按顺序存储的。而且我们实际物理存储是按线性存储的，在每一个存储的地方都有地址相对应。只有这样，我们存储介质中的所有空间才可以被寻址也就是说找到，然后拿来供我们存储。如果找不到地址的话，就像小屁孩找不到家一样，肯定是不能用的咯。而线性存储又是如何转换到我们经常听说的扇区、页等的模式，有兴趣的读者可以去学习一下汇编语言。这里就不再进行介绍了。在我们操作系统的文件系统中，我们一般存储对应会有

一个 inode 对应我们的存储地址，而还有一个 block 对应我们数据存储的空间。而我们系统通过 inode 就可以找到我们数据 block 并进行基本的数据操作。

说到现在，我们基础知识背景就介绍完了。下面我们正式进入指针。笔者经常说，我们 C 语言是更加底层和强大的语言。而体现这些的地方，就是我们这里讲的指针。前面我们介绍了我们硬件存储的大致情况。我们指针就提供了我们直接操作底层的方式。指针我们就可以理解成我们之前说的 inode ，准确的来说，我们指针就是我们存储中的地址，而这个地址表示的就是我们数据存储的地址。通过我们指针中的地址，计算机就可以找到我们的数据。我们的数据就不是流浪街头的小屁孩，就是可以被父母找到的孩子。而我们指针在这里也是有类型的，表示我们的指针指向的数据类型、大小都是不同的。类型的话就和笔者之前变量章节 (2.1) 讲的一样，大家可以回头参考一下。

好了，老规矩，我们继续上代码讲解：

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{

        int integer = 4;
        int *int_pointer = &integer;
        // equal to : int *int_pointer;      int_pointer = &integer;

        printf("integer = %d\n", integer);
        printf("int_pointer = 0x%x\n", int_pointer);
        printf("*int_pointer = %d\n", *int_pointer);

        return 0;
}
```

运行结果是：

```
integer = 4
int_pointer = 0x4a321e2c
*int_pointer = 4
```

好了，大家这里看我写的代码很简单吧。到现在这个时候，相信大家对上面的代码已经很熟悉了吧。我就只给大家介绍一下我们本章节讲的东西吧。在我们代码中，我们在第 6 行定义了一个整型变量并初始化为 4 。在第 7 行，我们定义了一个整型的指针，并把我们的指针指向了我们上一行定义的整型变量。在这里需要给大家讲解一下，我们在 C 语言中，对一个变量取它的地址的操作是由符号 & 进行得到的。在代码中，我们使用注释给大家介绍了另一种操作方式。不过我们的指针变量不再加上星号 * 了。在我们的指针中，指针变量名加上星号表示我们指针指向的变量的值，而没有星号只有指针变量名的时候，表示我们指针变量指向的变量地址。所以，我们注释中的另一

种操作方式就是取得变量的地址，并把地址赋给我们的指针变量。

下面的代码中，我们就是把我们定义的变量和我们的指针进行输出了。首先我们输出的是定义的整型变量，然后是把我们指针地址以 16 进制形式进行输出。最后则是把我们指针指向的数据或者说我们的变量进行输出。应为我们数据的存储地址不一样，所以我们同样的代码，当在你的机器上运行的时候，就只有指针指向的地址是不一样的，其他的都是一样的。

好了，既然我们指针基本上已经带大家入门了。那就给大家多介绍一下。

首先，我们变量的时候虽然没有给大家介绍指针，但是我们的指针确实是一种变量，可以进行赋值等操作。因为我们指针是操作的底层，直接找到存储的数据它家门上去了，所以我们对指针的任何操作都会影响到我们的指针所对应的数据。我们的数据类型也和之前介绍的变量类型一样，也都是可以用来指明我们指针的类型。只要我们指针有一个星号 * ，那就表明这是一个指针变量。同样，既然我们函数中的参数可以是任何变量类型，那就可以是我们的指针。而当我们参数是指针的时候，因为指针直接操作的是底层对应的数据，所以当我们调用函数后，我们的传入的变量便会因为函数的使用改变。专业的术语来说，就是实参函数。当我们使用的参数直接是变量的时候，那就表示把传入的变量的值复制一份给我们的函数进行操作。专业的术语来说，就是形参。然后，我们函数返回值的类型，也是各种基本的变量类型。同样，也可以是指针，也就是说，我们函数返回值是指向某个变量的地址。

Ok, 说了这么多，我们还是给大家看一下代码吧。不然会让大家说的云里雾里的了。

```c
#include <stdio.h>

int swap( int a, int b );
int swap_p( int *a, int *b);

int main(int argc, const char *argv[])
{

        int a = 1;
        int b = 3;
        int *a_p = &a;
        int *b_p = &b;

        printf("\tInit value\n");
        printf("a = %d, b = %d\n", a, b);

        // use swap function to swap two parameters
        swap( a, b );
        printf("\tswap value\n");
        printf("a = %d, b = %d\n", a, b);

        // use swap_p function to swap two parameters
        swap_p( a_p, b_p );
        printf("\tswap_p value\n");
```

```
25          printf("a = %d, b = %d\n", a, b);
26
27
28          return 0;
29  }
30
31  int swap( int a, int b )
32  {
33          int mid = 0;
34
35          mid = a;
36          a = b;
37          b = mid;
38
39          return 0;
40  }
41
42  int swap_p( int *a, int *b)
43  {
44          int mid = 0;
45
46          mid = *a;
47          *a = *b;
48          *b = mid;
49
50          return 0;
51  }
```

运行结果是：

```
1                   Init value
2               a = 1, b = 3
3                   swap value
4               a = 1, b = 3
5                   swap_p value
6               a = 3, b = 1
```

在我们上面的代码中，首先定义了两个函数，一个是形参，一个是指针的实参。根据我们上面的代码运行结果看到，我们只有用实参的调用才发生了变化。而调用形参的时候，却完全不影响我们之前的变量。所以这里大家就感受到 C 语言的强大的地方了吧。开个玩笑，相信大家更加理解我

们指针了吧。然后，我这里给大家布置一个任务，有兴趣的可以去查一下资料，看看我的代码中的 main 函数的第二个参数是表示什么意思。

好了，顺带提一句，我们之前讲过的数组，当时也提到了一次，数组的名字表示我们数据变量存储的第一个地址，也就是首地址。

3.7 字符串操作

好了，到了这里，我们就算是真正进入 C 语言的高级中的应用了。平常大家使用的语言我相信是句子而不是字符。所以，我们之前讲过了字符还是远远不够的。这里呢，我就给大家专门拿出一小节来进行字符串的说明。在前面的数组章节 (3.3) 中进行一点点的介绍过。当我们用字符型数组时是可以用来表示字符串的。所以，我们这一小节就在之前的基础上进行讲解吧。

首先，我们来回顾一下之前讲过的东西。我们的数组是可以用来表示同一类型的多个数据的，而我们基本的变量类型里面，有一个是用来表示字符的 (char) 类型。当我们声明多个字符的时候，这些字符连接起来就成了字符串。就好像我们单个字，多个连接起来的时候就成了话。而在我们用数组表示字符串的时候呢，需要注意几点。

1) 首先就是数组名表示数组首地址。

2) 其次是表示字符串的数组末尾会有一个 '\0' 表示结尾。

3) 最后是我们的数组必须一个一个的操作，而不能像其它变量直接赋值。

好了，基本上上面就是我们需要注意的地方了。我们在操作字符串的时候，一般有两种选择：一种是自己写函数去操作字符串数组。另一种则是调用系统中已经写好的函数。这里我给大家直接看代码吧。就从代码中进行感受吧。

```c
#include <stdio.h>
#include <string.h>

int strcopy( char str1[], char str2[] );

int main(int argc, const char *argv[])
{
        /*
         *         #include <string.h>
         * char *strcpy(char *dest, const char *src);
         *
         */

        char str1[10] = "who are u?";
        char str2[8] = "nick";
        char str3[8] = "tom";

        printf("The string before:\n");
        printf("%s\n", str1);
        printf("%s\n", str2);
        printf("%s\n", str3);
```

```
22
23
24          strcpy( str3, str2 );
25          printf("The string after with system function:\n");
26          printf("%s\n", str1);
27          printf("%s\n", str2);
28          printf("%s\n", str3);
29
30          strcopy( str1, str2 );
31          printf("The string after with system function:\n");
32          printf("%s\n", str1);
33          printf("%s\n", str2);
34          printf("%s\n", str3);
35
36
37          return 0;
38  }
39
40  int strcopy( char str1[], char str2[] )
41  {
42          int index = 0;
43
44          while( str2[index] != '\0' )
45          {
46                  str1[index] = str2[index];
47                  index++;
48          }
49
50          while( str1[index] != '\0' )
51          {
52                  str1[index] = NULL;
53                  index++;
54          }
55
56          return 0;
57  }
```

运行结果是：

```
The string before:
who are u?
nick
tom
The string after with system function:
who are u?
nick
nick
The string after with system function:
nick
nick
nick
```

在我们代码中注释部分是我们代码中调用的系统函数的 strcpy 的原型定义和头文件。有兴趣的读者可以去查一下资料。这里就不再进行过多的介绍。另外，对字符串进行操作的系统函数还有：strlen, strcmp, fprintf 等函数。这里只做一个提示，有兴趣的话还是多动动手去查查资料吧。毕竟，作为一个码农，虽然要懒，但是脑子可不能懒啦。

在上面我们的代码中，我还定义了一个最基本的字符串复制函数。如果这个函数都写不出的话，说明你的 C 语言的基础还没有过关哦。所以，要记得多思考，多读代码，多提高自己啊。

3.8 宏

Ok, 到了这里，基本上就是我们一般项目中才会用到的技术了。当你读到这里的时候，祝贺你坚持到了这里。因为现在有一个愿意静下心来好好读书学习的人已经很少了。笔者也很开心你能读到这里。好了，收拾好心态，我们继续开始学习吧。

首先我就直接把我们的宏定义的格式直接写出来吧。

```
#include <filename>

#define <name> <original name>

#if <Logical>
        <command>
#elif <Logical>
        <command>
#else
        <command>
#endif

#ifndef <Logical>
        <command>
```

```
15                  #endif
```

在上面的代码中，我们首先看一下第一行，这是我们 C 语言中的大家最经常接触到的东西：包含头文件。一般我们的头文件是指以.h 结尾的文件，里面放了各种函数和变量的定义。而我们第三行则是定义，也就是我们的宏定义。宏定义就是我们为某一个东西取一个别名。然后在我们下面的代码中直接使用别名。当我们的代码进行编译的时候，我们代码中的别名就会被编译器自动替换成我们取别名的这个东西。不管是一个变量、常数、还是表达式。

好了，接着看我们代码下面的。在我们第第 5 行到第 11 行是我们的一个条件宏定义。一般被用来进行代码调试或平台信息的判断等。在我们最下面的地方，则是表示如果没有定义我们判断的东西，则执行相应的代码。这里呢，我们就先来看一些代码给大家学习一下。

```
1  #include <stdio.h>
2
3  #define debug
4  #define pi 3.1415926
5
6  int main(int argc, const char *argv[])
7  {
8          float result = 0;
9          int semidiameter = 0;
10
11         // input the semidiameter of a circle
12         printf("Please input a semidiameter of a circle:\n");
13         scanf(" %d", &semidiameter);
14         // calculate the result
15         result = pi * semidiameter * semidiameter;
16
17 #ifdef debug
18         printf("\tdebug info\n");
19 #endif
20
21         // print the result to console
22         printf("The result is : %f\n", result);
23
24
25
26         return 0;
27 }
```

这里在我们的代码中，我们定义了一个宏变量用于调试，当我们注释掉我们的调试的宏的时候，就不再有调试信息输出。然后我们定义了一个宏用来表示我们的常数。在这里大家思考一下，为什么我们直接定义宏而不是直接在代码中写呢？这里我还是给大家一个答案咯。我们这里的代码之有 27 行，当我们的代码有几千上万行的时候呢？我们这个数字难道还要一个一个的去输入吗？如果有一天我们的这个数字变了，我难道还有几千上万行代码这样去一个一个的修改吗？而当我们定义了这个宏的时候，我们只需要修改这一个地方，是不是所有的地方都会改变了呢？所以才说程序员都懒啊。不过确实是很高效哦。

这样呢，大家应该已经有比较好的体会了吧。下面我就给大家介绍一个高大上的东西。就是项目管理中遇到的事情。先假设我们的项目非常大，代码是几万行，这个时候，我们还需要把所有代码放到一个文件吗？可以当然是可以，但是看你代码的人背后把你骂的要死的时候可别怪别人哦。这里我想说的就是，我们当一个项目很大的时候，我们就可以根据功能进行划分，并分别放到不同的文件中去。到底怎么放呢？都用到什么东西呢？这里我们就先给大家看一个很小的例子：

主程序：

```c
#include <stdio.h>
#include "sum.h"

int main(int argc, const char *argv[])
{

        int a = 1;
        int b = 2;
        int result = 0;


        printf("result = %d\n", result);
        result = sum( a, b );
        printf("result = %d\n", result);

        return 0;
}
```

sum.h 文件：

```c
#ifndef _sum_H
#define        _sum_H

#include <stdio.h>

int sum( int a, int b );
```

```
7
8  #endif
```

sum.c 文件：

```
1  #include "sum.h"
2
3  int sum( int a, int b )
4  {
5          return a+b;
6  }
```

运行结果是：

```
1                  result = 0
2                  result = 3
```

这里呢，我们代码就不再过多介绍了。首先看我们这个项目的框架，首先是我们的主程序 project.c 文件，C 语言的标准库文件包含，然后是我们自己写的头文件的包含。而我们编译器实际上在遇到包含头文件的包含操作的时候，进行的是把头文件里面的所有东西放到我们当前包含语句这里的。所以就会遇到一个问题，那就是一个文件重复包含，这样就会发生不可预知的错误。所以，我们在 sum.h 头文件中用到了我们之前学到的宏定义的框架。我们这个文件中的前两行和最后一行是为了避免重复定义而写的。一般大家写头文件的时候记住都要这样写哦。我们的宏定义中首先是如果没有定义我们的这个名字的宏，那我们就执行下面的操作，如果定义了，就不再执行。而这个名字，当然了，一般什么名字都可以。但是一般是头文件的名字大写。但是笔者喜欢小写，就这样坚持下来了。你们一定要记得这个格式哦。不然被同行鄙视了我可不管啊。其次是我们头文件中的函数的声明，或者说定义。我们的函数的声明放在我们的.h 文件中，而我们的.c 文件中则放我们函数的定义。就比如我上面的例子一样。如果大家还是不理解的话，大家可以先当做一个模板来记忆。

然后我们的代码的执行结果这里就不再多说了。

3.9 文件

到这里的话，我们就算是到了 C 语言的最后了。任何一种语言、程序或者系统操作的对象都是数据，可以说，没有数据就没有计算机这一行业。而虽然我们的数据的表示形式多种多样，但是归根结底都是一文件为存储单位进行存储的。这里呢，我们就给大家讲一下我们 C 语言中的文件操作。这里呢，我们就以标准库中的函数为标准：

```
FILE *fopen( const char *filename, const char *mode);

int fclose( FILE *fp );

//写入文件
int fputc( int c, FILE *fp );
int fputs( const char *s, FILE *fp );

// 读取文件
int getc( FILE *fp );
char *fgets( char *buf, int n, FILE *fp );

int fseek( FILE *fp, long offset, int where );
                                    where
                    SEEk_SET    : start of file
                    SEEk_CUR    : current of file
                    SEEk_END    : end of file
```

以上就是我们文件操作中标准库所提供的函数方法了。第一个函数是我们文件打开操作，第一个是我们的文件名，如果有路径就给一个路径，如果没有就表示程序所在的当前目录。第二个参数表示的是以什么形式打开，总共有如下模式：

表 5: File Mode

模式	描述
r	打开一个已有的文本文件，允许读取文件。
w	打开一个文本文件，允许写入文件。如果文件不存在，则会创建一个新文件。在这里，您的程序会从文件的开头写入内容。如果文件存在，则该会被截断为零长度，重新写入。
a	打开一个文本文件，以追加模式写入文件。如果文件不存在，则会创建一个新文件。在这里，您的程序会在已有的文件内容中追加内容。
r+	打开一个文本文件，允许读写文件。
w+	打开一个文本文件，允许读写文件。如果文件已存在，则文件会被截断为零长度，如果文件不存在，则会创建一个新文件。
a+	打开一个文本文件，允许读写文件。如果文件不存在，则会创建一个新文件。读取会从文件的开头开始，写入则只能是追加模式。

好了，这里呢，我们就不再多介绍了，我们这里呢，就直接给大家上代码了：

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{
        char read_buf;

        FILE  *file = NULL;

        // open a file
        file = fopen( "test.txt", "r" );

        // read file line by line
        while(EOF != ( read_buf = fgetc( file ) ))
                printf("%c", read_buf);

        fclose(file);

        return 0;
}
```

test.txt 文件内容：

```
test
write by wybuhui
```

运行结果是：

```
                test
                write by wybuhui
```

这里就给大家简单介绍一下吧。我们这里首先定义了一个变量用来存我们读取文件的数据，然后定义了一个文件指针，用来指向我们的文件标记符。然后就调用系统的函数打开了和代码一个目录下的 test.txt 文件，以只读的权限。然后把我们文件打开操作返回的文件标识符号赋给我们之前定义的文件标识符变量。先面就是我们循环读取我们的文件，然后就进行输出到终端。这里需要解释一下的是，我们读取文件操作读取到文件最后会返回一个 EOF 标志，表示 End Of File 。这样，当我们读取到文件结尾后跳出我们的循环，然后关闭我们打开的文件。然后程序结束退出。这里需要强调一点，每当我们打开一个文件，我们都需要关闭。否则你的电脑就是这样卡死的。从专业的

角度来说，就是我们系统的资源有限，如果你把所有资源都占了，那你的电脑不卡就奇怪了。

说到这里，就基本上差不多了。后续的就不再多说了。其他的文件操作，大家就自己多去写代码多去尝试啦。这里就不再多说了。写到目前位置，笔者基本上就把 C 语言的基本介绍完了。如果读者在读本文档遇到了一些问题的话，笔者还是建议大家多查文档，尤其是官方文档，或者说第一手资料。当然，本文档还存在很多的不足。有些方面还没有介绍到。但是，额，就先让笔者偷个小懒，等下次更新的时候再完善啦。谢谢大家的体谅哦。如果本文档有错误的地方，欢迎大家邮箱联系笔者哦。拜拜 ~