C初级

笔记本： 我的第一个笔记本

创建时间： 2020/4/17 18:41 **更新时间：** 2020/5/17 17:54

作者： 1272209351@qq.com

1、地址：从下往上增长的　　栈：向下增长　　栈帧：存放临时变量，被调用时创建，调用结束释放空间

2、字符串：有双引号引起来的，结束标志：\0 ；字符串默认 \0 作为结束标志

″Abc″　字符串（常量）；　‘A’字符；　“”　空串；　″1234″　字符串；

3、回车换行：\n　　；　　转义采用 \　；　\″特殊转字面；　\n　字面含义特殊化

退格：\t　　；\a警告，蜂鸣　；　\f 进纸符　；\r 回车；　\t 水平制表　　\ddd　其中ddd表示1-3个八进制数；

\xdd 其中dd表示两个十六进制数

表 3.1　格式控制符

格式控制符	说　明	举　例	输出结果
%d	输出十进制的整型数据	printf("%d",123);	123
%f	输出单精度和双精度的实型数据	printf("%f",123.456);	123.456000
%c	输出一个字符	printf("%c",'M');	M
%s	输出一个字符串	printf("%s","hello world");	hello world
%o	输出八进制的整型数据	printf("%o",12);	14
%x(或%X)	输出十六进制的整型数据	printf("%x",15);	f
%e(或%E)	以指数形式输出实型数据	printf("%e",456.789);	4.567890e+002
%p	输出变量的内存地址	printf("%p",&a);	变量 a 的地址

表中，对于八进制和十六进制，后面的输出项是十进制，根据进制之间的转换规则相应的输出数值，而对于指数形式而言，e(或者 E)后面有四个有效位置，第一个位置为符号位。

在使用格式控制符的过程中，有时还需要进行进一步的设置。具体包含以下几个方面。

%d整型输出，%ld长整型输出，

%o以八进制数形式输出整数，

%x以十六进制数形式输出整数，或输出字符串的地址。

%u以十进制数输出unsigned型数据(无符号数)。注意：%d与%u有无符号的数值范围，也就是极限的值，不然数值打印出来会有误。

%c用来输出一个字符，

%s用来输出一个字符串，

%f用来输出实数，以小数形式输出，默认情况下保留小数点6位。

%.100f用来输出实数，保留小数点100位。

%e以指数形式输出实数，

%g根据大小自动选f格式或e格式，且不输出无意义的零。

4、strlen()　计算字符串内容长度　　sizeof () 计算字符串空间大小

5、注释方式：//：注释一行；　　/**/ ：两个*之间内容注释掉，但是不支持嵌套注释；　if(0){} 可以达到注释效果，但是不推荐使用

6、C语言当中：0为假，非0为真；　　shift+f5　系统报错处理

7、while(1){}　可以形成死循环；for(;;)死循环；　　sleep()　毫秒计时

8、C语言中的三种循环：while() {} 循环(初始化，条件判定，条件更新)；　for(; ;)循环（三个条件同在一起）；do{}while() 循环:先循环在判定

9、函数：面向过程编写的最重要的语法结构；从工程上讲：函数可以让我们的代码具有结构性；从维护性：提升代码的可靠性

函数：返回值（该函数是否调用成功）；函数名（见名知意）；形参列表；函数体
C函数可以不写返回值！而且默认返回值是int； void 空类型没有返回值
形参，实参：形参实例化；传值传参（改变形参不会改变实参）要发生数据的临时拷贝， 只要调用就会发生拷贝
传址传参：* &
10、scanf :#pragma warning(disable:4996)
11、数组：一组相同类型元素的集合，数组名+[]； 数组的初始化{}； 数组的起始下标为0
12、操作符：+ - * / (移位操作符)<< >>
将整数右移一位相当于对整数除以2，向右移两位相当于两次除以2
按位与： & 1&1=1 0&1=0 1&0=0 0&0=0 同真为真
按位或： | 1|1=1 0|1=1 1|0=1 0|0=0 一个为真结果为真
异或： ^ 相同为0不同为1
赋值运算符 必须有“=”
初始化 VS 赋值：
定义变量： 1、开辟空间，大小由类型决定 2、向空间内容放置数据
逻辑反：!
13、变量在内存中是有地址的 &
数据前加~ 表示取反
14、自增自减：i++（先使用，后自增） i-- ;前置 ++i(先自增，后赋值) ；后置 --i
逻辑与：&& 逻辑或：||
条件操作符（三目运算符、逗号运算符）//赋值运算符 必须有“=”

15、C语言中的常见关键字（32个）
typedef ：类型重命名
#define 宏定义 只做简单的文本替换(便于代码的维护性、方便阅读能够做到见名知意)
16、局部变量==自动变量 %p 取地址 &
static 修饰局部变量只会改变其生命周期，让静态局部变量出了作用域依然存在，到程序结束，生命周期才结束，但是它的作用域没有变化
变量只能被初始化一次但可以多次赋值
定义只能定义（开辟空间+放置内容）一次，声明可以声明多次
17、全局变量是支持跨文件访问的，static 修饰全局变量代表该全局变量只在本文件内有效，只能在本文件访问，但可以被本文件中的函数进行间接访问
函数访问也是支持跨文件的，static 修饰函数表示该函数只在本文件被调用或访问，不能跨文件访问
extern int g_value （跨文件访问） 声明
18、.exe 是一个文件（硬盘上），定义变量全都是在内存中定义或开辟的，双击：把 .exe 加载到内存
计算机访问内存的基本单元是字节，以32计算机为例：2^32*1字节=（2^10）*（2^10）*（2^10）*（2^2）* 1字节 = 4GB
19、内存中空间的编号称为地址（空间连续编址），提高查找效率，编址是硬件电路自动完成的
20、指针：地址 指针变量：变量，该变量里边放的内容是地址
对指针解引用代表的就是指针所指向的变量,但具体用的是变量空间还是内容取决于是左值还是右值
一个变量：变量的空间，变量的内容
*a=20 用的是a的空间；int b=*a 用的是a的内容 *A：作为左值表示使用地址，作为右值表示使用内容

21、if 条件判定，有分支（else） if ... else ..语句（多选一）
switch 只能放整型表达式 int short char long long long
switch 判断没有分支，需用break 一个switch 语句可以有多个break case ,但是只能有一个 default
当 switch表达式的值并不匹配所有case标签的值时，这个default子句后面的语句就会执行。所以，每个switch语句中只能出现一条default子句。
break 跳出当前循环 continue 结束本次循环
22、一个数字是否是奇数可以看最后一个Bit位，如果最后一位bit位是1 则为奇数，0则为偶数(按位与 &)
23、 while循环中 : break 结束循环，跳出循环 while 中的 break 永久终止循环
while循环中 ：continue 结束本次（一次）循环
getchar 获取字符 putchar 输出一个字符
// #define EOF
for 循环：for(条件初始化；条件判定；条件更新)
for循环中： break 结束循环 continue 结束本次循环
do..while..循环：先执行后判断（至少执行一次）

24、 ctrl+ z 表示在对话框输入完毕

25、memset 把内存条按字节清成固定值　　(memset)是按照字节赋值的
　　%c :字符　%s :字符串　%p :取地址
　F10 逐过程调试 ； F11 逐语句调试；　取消调试：shift + f5

26、用指针所指向的字符串不能被修改 : char *str="abcd1234" ，所以逆序输出会出错
27、char []表述数组：被当作普通的char 数组　/ 被整体当作字符串使用
　　char arr[]={'a','b','c','d'.'\0'}；　char类型数组要有\0结束标志 ； 字符串数组输出：printf("%s\n",arr);
　　数组的空间是在对于函数的栈帧内部开辟的，换言之是在栈上开辟空间的（一般情况）
　　数组只能被整体初始化，不能被整体赋值

对于二维数组int array[M][N], 说明如下：

1. M和N都必须为常数，
2. M代表数组有M行，N代表每行中有N个元素
3. 其中M可以省略，省略后必须给出初始化表达式，编译器从初始化结果中推断数组有多少行
4. N一定不能省略，因为N省略了就不能确定一行有多少个元素，也不能确定数组有多少行

28、二维数组：a[][]　数组的行可以省略列不能省略
　　a[3][4]={1 , 2, 3, 4, 5, 6, 7, 8, 9 ,0 , 11, 12}
　　a[][4]={ {1, 3, 4} , {5 ,6 ,7, 8} ,{0} }　　花括号里边的花括号表示某一行
二维数组本质可以看做一维数组，只不过该数组内放的元素是一个一维数组！
任何数组，线性连续且递增　　数组名代表数组首元素的地址
数组名在两种情况下代表整个数组：&arr / sizeof(arr) ； 其他情况，都代表数组首元素的地址
（arr： 数组首元素地址　arr : 数组的地址　sizeof(arr) : 表示整个数组的大小　sizeof(arr+1) : 表示数组中元素的大小）
29、移位操作：
算术右移：如果是负数右移左边要补 “1” （符号位），右边补“0”
逻辑右移：补 0
逻辑反：！
按位反：~　二进制形式每一位取反
30、& 按位与，进行数值计算：a&b
　　&& 逻辑与，链接两个表达式：a>10 && b<2　　|| 逻辑或　----> 进行条件判断的
　　(，，，) 逗号表达式，其值为最后一个逗号后边的部分
31、typedef 类型重定义
访问结构体的成员：　结构体名. 类型名
　　结构体指针 -> 类型名
32、
为甚莫要进行整形提升?
因为运算是在CPU内进行的 2. 可能进行不对等赋值，如：int = char
整形提升是按照变量的数据类型的符号位来提升的
如果是无符号数，整型提升直接全部补0；若果是有符号数，补符号位（看变量本身的类型）

33、算术转换
int --> unsigned int --> long int --> unsigned long int --> float --> double --> long double
优先级：
　　() 优先级最高
　　后缀自增自减优先级大于前置自增
　　-> 优先级高于 * (间接访问)
　　++ -- （前置/后置） 大于 *

《C和指针》 《C深度剖析》

34、结构体内容分访问： . 操作符（结构体变量） -> 操作符（结构体指针）

结构体的初始化：

```
//打印结构体信息
struct Stu s = {"张三", 20, "男", "20180101"};

//.为结构成员访问操作符
printf("name = %s age = %d sex = %s id = %s\n", s.name, s.age, s.sex, s.id);
//->操作符
struct Stu *ps = &s;
printf("name = %s age = %d sex = %s id = %s\n", ps->name, ps->age, ps->sex, ps->id);
```

35、指针： type + * 变量 （空间 内容 地址）

（1）野指针（悬垂指针）：int *p （指向未知）

int *p=NULL

指针是用来存放地址的，地址是唯一标示一块地址空间的。

指针的大小在32位平台是4个字节，在64位平台是8个字节。

（2）对指针 +1 ：实际上是加上指针所指向的类型的大小

* +(sizeof(type))

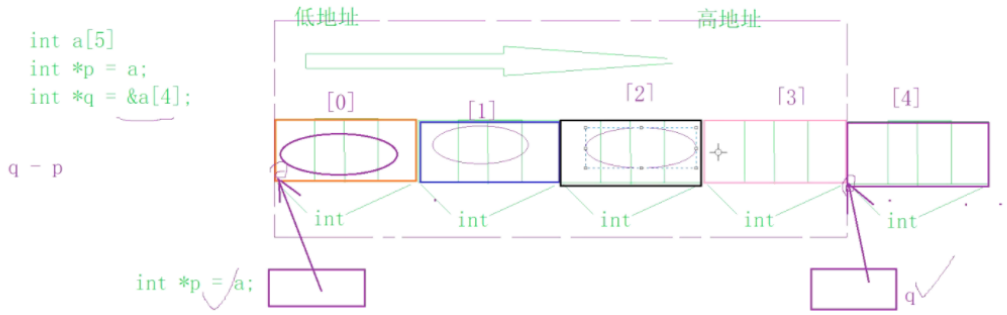
（3）对指针解引用，代表指针所指的变量（目标） （不包括强转）

指针变量也是变量，指针变量也有地址、内容、空间

指针进行解引用访问的时候，自己的类型决定了自己能访问到几个字节（type）

（4）指针指向一个变量一定指向的是这个变量的最低地址

为什么指针减指针，说的是所经历的“元素”，但是包括了第一个，没有包括最后一个，为什么呢？



（5）指针-指针：代表两个指针之间所经历的“元素”（不一定是1个字节）的个数（同类型指针相减，两个指针指向同一块内存）

```
#define N_VALUES 5
float values[N_VALUES];
float *vp = NULL;
int main()
{

    //指针+-整数；指针的关系运算
    for (vp = &values[0]; vp < &values[N_VALUES];)
    {
        *vp++ = 1.0;
    }
```

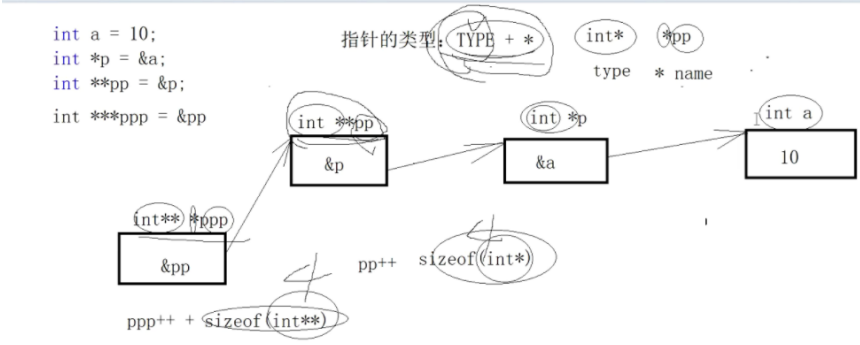
数组地址比较的时候，最后一个 元素 的下一个元素的地址是可以被拿来比较的。
但是，仅仅是拿来进行大小比较，不能有任何其他操作。如：写入

（6）数组的遍历：

```
int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
int num = sizeof(a)/sizeof(a[0]);
int *p = a;
int i = 0;
for (; i < num; i++){
    //printf("%d ", a[i]);
    //printf("%d ", *(a + i));
    //printf("%d ", *(p + i));
    printf("%d ", p[i]);
}
printf("\n");
```

1. 指针和数组没有关系，他们是两种不同的类型
2. 但是他们在访问数组元素的时候，是具有一定的相似性的

36、二级指针：指针变量 指针地址

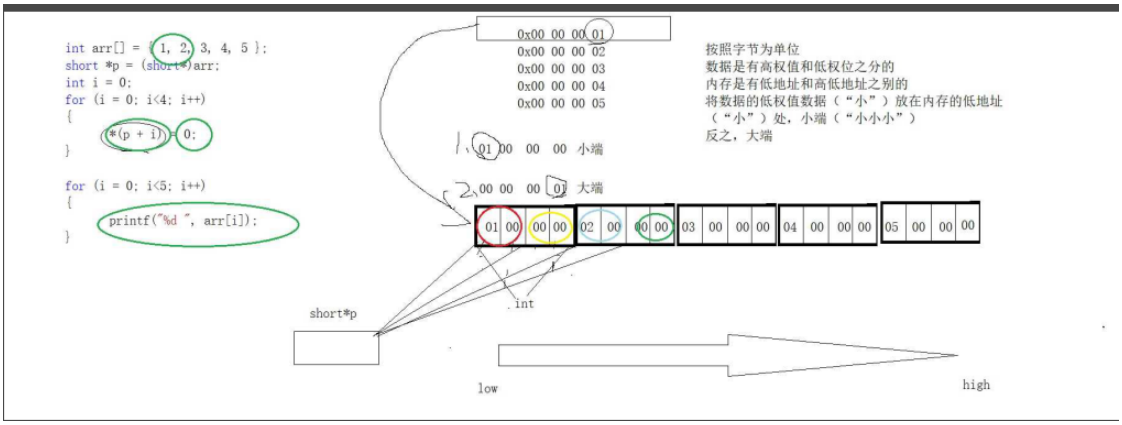



```c
int a = 10;
int *p = &a;
int **pp = &p;

//p = 20; //p变量的内容变成20（&a）
//pp = 20;//pp变量的内容变成20（&p）
//*p = 20;//a = 20
//*pp = 20; //p = 20
//**pp = 20;//a = 20;
```

37、int a[]　　整型数组
　　int *a[]　　指针数组
　　int (*a)[]　数组指针

38、大小端：
（按照字节为单位）
数据是有高权值和低权位之分的
内存是有低地址和高低地址之别的
将数据的低权值数据（“小”）放在内存的低地址（“小”）处-------小端（“小小小”）　，　反之----------大端



39、结构体：具有相同或不同类型元素的集合
（1）数组 或 结构体：在定义变量的同时进行整体初始化赋值 {}
（2）结构体赋值：要用（结构体变量名+. 变量）　进行访问，　“=”进行赋值
（3）初始化：在定义变量的同时进行初始化赋值　{}
　　赋值：在已经定义变量的前提下，在对变量进行赋值
（4）结构体初始化时候必须按照变量顺序，从左往右依次赋值不能省略或跳过
（5）结构体传参不谈“降维、退化”之类的问题，结构体会发生硬拷贝
函数调用结构体非常耗时效率低，所以可以人为的对变量进行降维（* 指针类型），所以对结构体变量进行访问需要用到 -> 操作

strcpy　只能对字符串（char 类型）进行拷贝

40、调试

（1）
Debug　：调试版本　（调试，占内存大） 可调试（F10 ， 监视等）
Release　：发布版本 （优化之后，占内存小） 不能进行调试（F10）

F9 断点：代码分块调试（遇到断点终止调试）

shift f5 ： 取消调试

（2）逻辑报错：程序正确结果出错
　　语法报错：标点符号等错误

（3）assert ()　------->>　头文件 #include<assert.h>
　　assert 检验传入参数是否合法 -------> 编译时会提示错误类型（Debug 状态下）

（4）const　变量不能直接被修改

```
int main()
{
    const int a = 10;
    const int *p = (int*)&a;
//  *p = 20;
    const char *s = "hello world";
    //*s = 'H';
```

int a=10;
const int *p=&a;
　　const 修饰 * ，代表不能通过该指针，对指向的目标进行更改
int const *p=&a　<--------->　const int *p=&a;　等价
int *const p=&a ; **const 修饰的是指针 [变量] ，指针变量不能被修改，也就是p 只想不能更改**

```
const int a = 10;
const int *p = &a;
//*p = 20;
//const char *const s = "hello world";

//const int *p = &a; //const 修饰*，代表不能通过该指针，对指向的目标进行修改！
//a = 20;
//*p = 20;
//int const *p = &a; //const 修饰*，代表不能通过该指针，对指向的目标进行修改！
//int *const p = &a;   //const 修饰的是指针[变量],指针变量不能被修改，也就是p的指向不能
//const int * const p = &a; //const 修饰的是* and p，代表指向目标不能改且p的指向不能改！
//p = 20;
```

*p =20 不能对变量 a 进行修改