选择20题答案.md

1. (2, 0, 1, 2)。

2. (15)。

3. (i > j)。

解释如下:

表达式会包含隐式类型转换,它由编译器自动执行,不需程序员介入。

- 何时发生隐式类型转换:

(1) 在混合类型的表达式中,操作数会被转换为相同类型,如:

```
int ival;
double dval;
ival >= dval; // ival converted to double
```

(2) 条件表达式会被转换为bool类型,如:

```
int ival;
if (ival) // ival converted to bool
```

条件操作符(? :)中的第一个操作数,逻辑非(!)、逻辑与(&&)、逻辑或(||)的操作数都是条件表达式。if、while、do while、以及for的第2个表达式都是条件表达式。

(3) 初始化和赋值,如:

```
int ival = 3.14 // 3.14 converted to int
int *ip;
ip = 0; // the int 0 converted to a null pointer of type int *
```

(4) 在函数调用时,所传递的参数也可能发生隐式类型转换。

- 如何转换:

(1) 算术转换

算术转换保证在执行操作前,将二元操作符的两个操作数转换为同一类型,并使表达式的值也具有相同的类型。

算术转换通常的是做提升(integral promotion),对于所有比int小的整型,包括char、signed char、unsigned char、short和unsigned short,如果该类型的所有可能的值都能包含在int内,它们就会被提升为int,否则被提升为unsigned int。如果将bool值提升为int,则false转换为0,true转换为1。

包含short和int类型的表达式,short转换为int。如果int足以表示所有unsigned short类型的值,则将unsigned short转换为int,否则两个操作数均转换为unsigned int。long和unsigned int的转换也一样。只要机器上的long足够表示unsigned int类型所有的值,就将unsigned int转换为long,否则两个操作数都转换为unsigned long。在32位的机器上,long和int通常用一个字长表示,此时如果表达式包含unsigned int和long,两者都转换为unsigned long。

如果表达式包含signed和unsigned int, signed会被转换为unsigned。如果int 操作数的值恰为负数, 其转换为unsigned int可能会变为一个很大的正数（转换结果是该负值对unsigned int的取值个数求模）。所以最好避免对int和unsigned int的两个操作数进行比较。

转换示例：

```
bool     flag;        char          cval;
short    sval;        unsigned short usval;
int      ival;        unsigned int   uival;
long     lval;        unsigned long  ulval;
float    fval;        double         dval;
3.14159L + 'a'; // promote 'a' to int, then convert to long double
dval + ival;    // ival converted to double
dval + fval;    // fval converted to double
ival = dval;    // dval converted (by truncation) to int
flag = dval;    // if dval is 0, then flag is false, otherwise true
cval + fval;    // cval promoted to int, that int converted to float
sval + cval;    // sval and cval promoted to int
cval + lval;    // cval converted to long
ival + ulval;   // ival converted to unsigned long
usval + ival;   // promotion depends on size of unsigned short and int
uival + lval;   // conversion depends on size of unsigned int and long
```

（2）其他隐式转换

1）数组名转换为指向其第一个元素的指针, 如：

```
int ia[10]; // array of 10 ints
int *ip = ia; // convert ia to pointer to first element
```

另外, 任意数据类型的指针都可转换为void *, 整形数值常量0可以转换为任意类型指针。

2）指针值可转换为bool

如果指针为0, 转换为false, 否则转换为true。如：

```
if (cp) // true if pointer cp is not zero
```

3）算术类型与bool的转换

算术类型转换为bool时, 0转换为false, 其他值（包括负值）转换为true。将bool转换为算术类型时, true转换为1, false转换为0。

4）转换与枚举类型

枚举类型对象或枚举成员将自动转换为整型, 其转换结果可以用于任何需要使用整数值的地方。具体会被转换为哪种整型, 依赖于枚举成员的最大值和机器。enum对象或枚举成员至少提升为int, 如果int无法表示枚举成员的最大值, 则提升到能表示所有枚举成员值的、大于int型的最小类型（unsigned int 、long或unsigned long）。

4. (2, 5)。

解释如下：

数组名就是数组0号元素的地址, 即a = &a[0]。

&a是指向一个有5个整型元素的数组的地址。a是一维指针, &a相当于是二维指针。&a+1就是从a向后跳过一个完整的数组所占用的内存空间。整型5个元素的数组占用5*sizeof(int)=5*4=20, 所以&a+1应该从a向后跳20字节。正好指到a[4]的后面。ptr是int *, 减1就是向前跳4个字节, ptr-1正好指向a[4]。

5. (10, 20, 30)。

解释如下：

int (*p)[3]，这里首先确定：p是一个指针，一个指向数组的指针。

p = &(p[0])，p是二维指针。

p[0] = &(p[0][0])，p[0]是一维指针。

p[0] + 1表示在列上移动。e.g.，p[0] + 1 = &p[0][0] + 1 = &p[0][1]。

p + 1表示在行上移动。e.g.，p + 1 = &(p[0]) + 1 = &p[1]。

因此：

*(p[0]+1) = p[0][1] = 20。

(*p)[2] = p[0][2] = 30。

6.（0）。

解释如下：

char str[] = "ABCD"; 相当于：

str[0] = 'A';

str[1] = 'B';

str[2] = 'C';

str[3] = 'D';

str[4] = '\0';

而*(p + 4) == str[4]，'\0'的ASCII为0，故输出0。

7.（2）。

解释如下：

注意短路求值。逻辑与或者逻辑或的表达式，先是判断一边，若一边可以判断整个表达式为真假时，另一边不再执行。

8.（10，4）。

解释如下：

sizeof返回数组所占的字节数，'wang' 'miao'共占8字节，显式'\0'占1字节，字符串末尾隐式'\0'占1字节，共10字节。

strlen返回字符串的长度，以遇到'\0'结束符为准，因此为4。

另外，对于指针，sizeof操作符返回这个指针占的空间，一般是4个字节；而对于一个数组，sizeof返回这个数组所有元素占的总空间,包括结束符'\0'。char*与char[]容易混淆，一定要分清。

strlen不区分是数组还是指针，就读到'\0'为止返回长度。而且strlen是不把'\0'计入字符串的长度的 。

9.（00801005, 00810014）。

解释如下：

p1指向字符型，一次移动一个字符型，1个字节；p1+5后移5个字节，16进制表示为5；p2指向长整型，一次移动一个长整型，4个字节，p2+5后移20字节，16进制表示为14。

另外，char每次移动1个字节；short移动2个字节 ；int、long、float移动4个字节 ；double移动8个字节。

10.（7）。

解释如下：

该函数为递归调用。

需注意i为静态变量。

f(1)：n=2; i=2;调用f(2)；

f(2)：n=4; i=3;调用f(4)；

f(4)：n=7; i=4;调用f(7)；

f(7)：返回7。

即最终函数返回结果为7。

11.（1）。

12.（0，1）。

13.（D）。

解释：

++的优先级比*高，所以D 中p++先执行，p 指向的地址改变了，所以year 的内容没有变化。

14.（9）。

15.（C）。

解释：

- 知识点1：函数指针变量。

函数指针变量的声明方法为：返回值类型 (* 指针变量名) ([形参列表]);

根据定义，

```
int (*pf)(float);
int (*p)(float) = &f1;
```

pf，p 都是函数指针变量。

- 知识点2:函数地址。

C 在编译时,每一个函数都有一个入口地址,该入口地址就是函数指针所指向的地址。

函数地址的获取,可以是函数名,也可以在函数名前加取地址符&。

(C) 错误是因为函数形参类型不匹配。

16. (Bejing)。

17. (D)。

解释:

str 为一个指针,但实际上为int 类型,传入函数内部并不会发生任何改变。

GetMemory 函数执行完成后,str 仍然指向NULL,所以赋值时崩溃。

正确的做法应该使用双指针,如下:

```
void GetMemeory(char **p) {
        *p = (char *)malloc(100);
}
void Test() {
        char *str = NULL;
        GetMemeory(&str);
        strcpy(str, "Thunder");
        strcat(str + 2, "Downloader");
        printf("%s", str);
}
```

18. (B)。

解释:

```
typedef char T[10] ;
T * a ;
```

这里T 是什么类型呢,把名字抹去不就是类型了吗?char [10],T 为一个char 数组。

那么T * a 中a是什么类型呢,T * ,T 为数组,是一个整体,a 为指向这个数组的一个指针喽。也就是a 为指向一个10个元素的数组的指针。

首先,a是指针,不是数组,而且a是指向数组的指针,不仅仅是指针。所以(A),(C) 排除了。那么剩下两项(B),(D).

```
char (*a) [10] ;
char *a [10] ;
```

这里就是一个优先级的问题了,[]优先级要高于解引用运算符。所以第一个a 为指针,指向具有10个char 元素的指针。第二个为数组,每个元素都是指针,每个指针指向一个char 变量。

这就是函数指针和函数也有类似的问题。分析优先级即可游刃而解。

19. (14)。

解释：

X & (X - 1); 统计X 的二进制中1 的个数；

X | (X + 1); 统计X 的二进制中0 的个数；

20.（AB）。

解释：

- 知识点1：C 编译器不做数组越界检查。

MAX 为255。

数组A 的下标范围为：0..MAX-1。

- 知识点2：当i 循环到255 时,循环内执行：A[255] = 255;

这句本身没有问题。但是返回for (i = 0; i<= MAX; ++i)语句时,

由于unsigned char 的取值范围在[0, 255], ++i 以后i 又为0 了。无限循环下去。