2.数据类型（22道）

1

2.1 用变量a给出下面的定义

（1）一个整型数： int a。

（2）一个指向整型数的指针（一重指针）： int *a。

（3）一个指向指针的的指针，它指向的指针是指向一个整型数的指针（二重指针）： int **a。

（4）一个有10个整型数的数组： int a[10]。

（5）一个有10个指针的数组，这10个指针是指向整型数的（指针数组）： int *a[10]。

（6）一个指向有10个整型数数组的指针（数组指针）：int (*a)[10]。

（7）一个指向函数的指针，该函数有一个整型参数并返回一个整型数（函数指针）：int (*a)(int)。

（8）一个有10个指针的数组，这10个指针均指向函数，该函数有一个整型参数并返回一 个整型数（函数指针数组）： int (*a[10])(int)。

2.2下面的代码输出是什么，为什么?

```
void foo(void)
{
    unsigned int a = 6;
    int b = -20;
    (a + b > 6)? printf("> 6") : printf(" <= 6");
}
```

答案: 输出是 “>6”。

解读: 当运算表达式中存在有符号数和无符号数时，有符号数隐式转换成了无符号数（即底层的补码不变，但是此数从有符号数变成了无符号数）。注意，正数的补码为其本身，负数的补码为其反码+1。因此-20变成了一个非常大的正整数，所以该表达式计算出的结果 ”>6”。

2.3写出float x与“零值”比较的if语句。

答案:

```
if(x > -0.000001 && x < 0.000001);
```

解读：因为计算机在处理浮点数的时候是有误差的，所以不能将浮点型变量用“==”或“！=”与数字比较，应该设法转化成“>”或“<”此类形式。

1

2.4下面代码有什么错误?

```
#include<stdio.h>
void main()
{
    char *s = "AAA";
    s[0] = 'B';
    printf("%s", s);
}
```

（1）"AAA"是字符串常量，s是指针，指向这个字符串常量，所以声明s的时候就有问题，应该是cosnt char* s="AAA"。

（2）然后又因为是常量，所以对是s[0]的赋值操作是不合法的。

2.5下面代码输出是什么?

```
#include<stdio.h>
void main()
{
    int *a = (int *)2;
    printf("%d", a + 3);
}
```

答案：输出是14。

解读：代码将数值2强制类型转换为int类型指针，int类型指针加3 相当于指向后面第三个int类型变量的首地址，一个int类型变量占4个字节，所以加3相当于指针往后移了12个字节，指向地址14处。

2.6下面代码运行后会是什么现象?

```
#include<stdio.h>
#define N 500
void main()
{
    unsigned char count;
    for(count = 0; count < N; count++)
    {
        printf("---%d---\n", count);
    }
```

```
}
```

答案：进入不断打印count值的死循环。

解读：因为unsigned char 类型变量的最大值为255，所以count只能从0一直增加到255，然后又恢复为0，无法退出for循环。

2.7下面函数的返回值是?

```
int foo(void)
{
    int i;
    char c = 0x80;
    i = c;
    if(i > 0)
        return 1;
    return 2;
}
```

答案：返回值为2。

解读：因为0x80 == 128，超出了char类型变量c的表示范围（-128~127），所以c == -128，进而i == -128，i < 0。

2.8结构体内存对齐原则?

答案：

（1）第一个成员的首地址（地址偏移量）为0。

（2）成员对齐：以4字节对齐为例，如果自身类型小于4字节，则该成员的首地址是自身类型大小的整数倍；如果自身类型大于等于4字节，则该成员的首地址是4的整数倍。若内嵌结构体，则内嵌结构体的首地址也要对齐，只不过自身类型大小用内嵌结构体的最大成员类型大小来表示。数组可以拆开看做n个数组元素，不用整体看作一个类型。

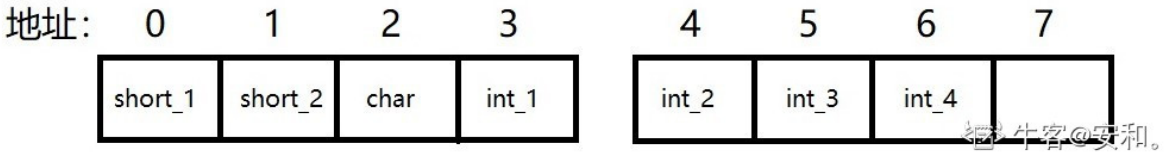
（3）最后结构体总体补齐：以4字节对齐为例，如果结构体中最大成员类型小于4字节，则大小补齐为结构体中最大成员类型大小的整数倍；如果大于等于4字节，则大小补齐为4的整数倍。内嵌结构体也要补齐。

注意：32位编译器，一般默认对齐方式是4字节。

2.9结构体内存对齐的原因?

（1）平台原因（移植原因）：不是所有的硬件平台都能访问任意地址上的任意数据。

（2）性能原因：数据结构（尤其是栈）应该尽可能地在自然边界上对齐，因为访问未对齐的内存，处理器需要做两次内存访问，而访问对齐的内存仅需要一次。如下图所示，访问对齐的short变量只需要一次，而访问未对齐的int变量则需要访问两次。



2.10给定结构体，它在内存中占用多少字节（32位编译器）？

```
struct A
{
     char t : 4;         // 4位
     char k : 4;         // 4位
     unsigned short i : 8;  // 8位
     unsigned long m;      // 4字节
};
```

根据结构体内存对齐原则，共占用8字节。

2.11在32位系统中，有如下结构体，那么sizeof(fun)的数值是?

```
#pragma pack(1)
struct fun
{
    int i;    // 4字节
    double d;  // 8字节
    char c;  // 1字节
};
```

答案：sizeof(fun)得到的结果是13。

解读：因为预处理语句 ”#prama pack(1)” 将编译器的字节对齐数改为1了，根据结构体内存对齐原则，该结构体占用的字节数为13。

2.12数组首元素地址和数组地址的异同?

（1）异：数组首元素地址和数组地址是两个不同的概念。例如int a[10]，a的值是数组首元素地址，所以a+1就是第二个元素的地址，int类型占用4个字节，所以两者相差4。而&a是数组地址，所以&a+1就是向后移动(10*4)个单位，所以两者相差40。

（2）同：数组首元素地址和数组地址的值是相等的。

2.13下面代码输出是什么?

```
#include<stdio.h>
void main()
{
    int a[5] = {1, 2, 3, 4, 5};
    int *ptr = (int *)(&a + 1);
    printf("%d, %d", *(a + 1), *(ptr - 1));
}
```

答案：输出为2, 5。

解读： a是数组首元素地址，所以*(a + 1)就是第二个元素a[1]。&a是数组地址，所以&a + 1是整个数组结尾的下一个地址，*(ptr - 1)就是a[4]。

2.14判断下列表达式正确与否?

```
char str[2][3] = {“a”, “b”};          // 正确，str是一个可存放两个字符串的字符串数组
char str[2][3] = {{1, 2}, {3, 4}, {5, 6}};  // 错误，行列不匹配
char str[] = {“a”, “b”};            // 错误，字符数组不能存放字符串
char str[2] = {“a”, “b”};           // 错误，字符数组不能存放字符串
```

答案：以注释形式展示。

注意：在C语言中字符用’’括起来，而字符串用””括起来。

2.15查看下面代码，p[6]等于几?

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
int *p = &a[1];
```

答案：等于8。

解读：p是一个int类型指针，指向a[1]，p[6]表示p往后移了6个单位（每个单位4个字节）并解引用，因此p[6]等于8。

2.16下面代码的输出结果是什么?

```
#include<stdio.h>
void main()
{
```

```
    char *str[] = {"ab", "cd", "ef", "gh", "ij", "kl"};  //指针数组
    char *t;
    t = (str + 4)[-1];
    printf("%s", t);
}
```

答案：输出"gh"。

解读：str表示数组首元素地址，str + 4表示数组第五个元素地址，(str + 4)[-1]表示在第五个元素地址的基础上往前移一个元素并解引用，因此输出是第四个元素。

2.17变长数组是什么?

在C99标准中，允许定义数组时[]中的值是整型变量或整型表达式，比如说下面的代码在支持C99标准的编译器中可编译通过：

```
#include<stdio.h>
void main()
{
    int n;
    scanf(“%d”, &n);
    int array[n];
}
```

2.18 bool类型包含于哪个头文件?

答案：在C99标准中位于stdbool.h。

注意：C89标准中则不支持，需要自己定义，代码如下：

```
#define TRUE 1
#define FALSE 0
typedef int bool;
bool res = TRUE;
```

2.19结构体struct和联合体union的区别?

（1）两者最大的区别在于内存的使用。

（2）结构体各成员拥有自己的内存，各自使用且互不干涉，遵循内存对齐原则。

（3）联合体所有成员共用一块内存空间，并且同时只有一个成员可以得到这块内存的使用权。一个联合体变量的总长度应至少能容纳最大的成员变量，且需要进行内存补齐。

2.20给了一个地址a，分别强转类型为：int变量、int指针、数组指针、指针数组、函数指针。

int变量	(int) a;
int指针	(int *)a;
数组指针	(int (*)[])a;
指针数组	(int *[])a;
函数指针	(int (*)(int))a;

2.21 执行完下面代码，c的值是多少?

```
unsigned int a = 1;
int b = 0;
int c = 0;
c = a + b > 0 ? 1 : 2;
```

答案：c的值是1。

解读：有符号数与无符号数一起运算时，有符号数转为无符号数。有符号数int b = 0的在内存中的补码是0，因此转为无符号数之后依然是0，a + b == 1 > 0，c == 1。

2.22 C语言中不同数据类型之间的赋值规则?

答案：

（1）整数与整数之间（char, short, int, long）：

①长度相等：内存中的数据不变，只是按不同的编码格式来解析。

②长赋值给短：截取低位，然后按短整数的数据类型解析。

③短赋值给长：如果都是无符号数，短整数高位补0；如果都是有符号数，短整数高位补符号数；如果一个有符号一个无符号，那么先将短整数进行位数扩展，过程中保持数

据不变，然后按照长整数的数据类型解析数据。

（2）整数与浮点数之间

①浮点数转整数：截取整数部分。

②整数转浮点数：小数部分为0，整数部分与整数相等。

（3）float与double之间

①double转float会丢失精度。

②float转double不会丢失精度。

注意：整数在内存中都是以补码的形式存储的。

讨论 评论

意义非凡 1#

字符数组不能存放字符串吗，char str[]={“aaaaa”};这样没问题呀

发表于 2021-02-27 00:00:42 赞(0) 回复(2)

安和。 作者 ： 你是说2.4吗?字符数组可以放字符串，没问题，但题目2.4是字符串指针，也叫字符串常量，初始化之后就不能改变了

2021-02-27 08:09:06 赞(0) 回复(0)

队长_放下枪 ： 他说的是2.14吧，你的注释有问题

2021-02-28 12:10:13 赞(0) 回复(0)

请输入你的观点

回复