C语言：指针易错

第一题（数组地址的存放）：

下面哪个代码是错误的？（）

```c
#include <stdio.h>
int main()
{
  int *p = NULL;
  int arr[10] = {0};
  return 0;
}
```

A. p = arr;

B. int (*ptr)[10] = &arr;

C. p = &arr[0];

D. p = &arr;

解析：

我们看到代码首先给了p一个空指针，之后定义了一个数组。我们知道单独的数组名（arr）在&和sizeof后面是表示整个数组，在其他地方表示数组首元素的地址。

A：将arr数组首元素的地址给p，没有问题。

B：int (*ptr)[10]这是一个数组指针，代码的意思是将整个的数组地址（&arr）放进这个数组指针里面，没有问题。

C：代码的意思是将arr这个数组的首元素地址放进p里面，没有问题。

D：前面说了&arr表示整个数组的地址，把整个数组的地址放进一个指针里面是不可行的。

第二题（assert宏）：

下面关于assert宏的描述哪个是正确的？（）

A. assert宏总会终止程序的执行。

B. assert宏仅在调试模式下起作用。

C. assert宏只能用于整数类型的表达式。

D. assert宏可以通过编译选项进行启用和禁用。

解析：

关于assert宏，它是一个断言，在写代码的过程中，如果要使用assert，就要包括头文件<assert.h>。在代码中加入assert(exp)，若exp为真，程序继续运行。否则就终止程序，这在代码的调试中很常用，我们通过定义NDEBUG宏来进行assert的禁用。

A: 当exp为真时程序不会终止。

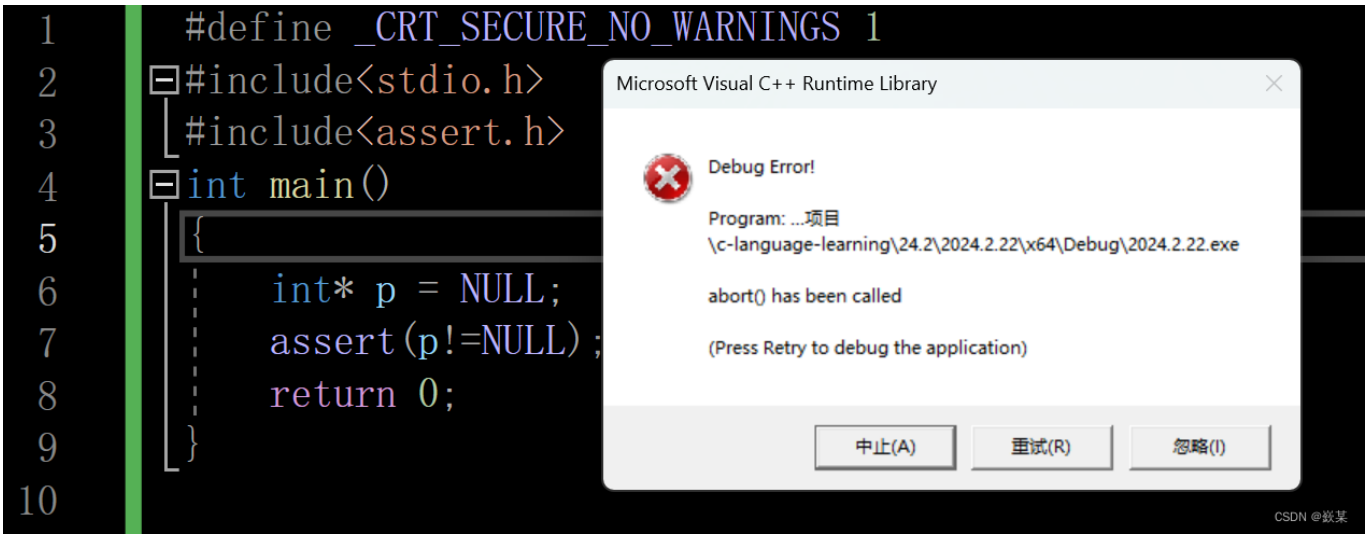
B: assert在Linux系统release版本下也可以起作用的（这个超纲了）

C: assert宏还能用于指针类型的表达式。

D: 前面说过定义NDEBUG宏可以禁用assert。（这里要注意的是#define NDEBUG必须写在所有头文件的前面！）

```c
#define _CRT_SECURE_NO_WARNINGS 1
#define NDEBUG
#include<stdio.h>
#include<assert.h>

int main()
{
    int* p = NULL;
    assert(p!=NULL);
    return 0;
}
```



编辑

第三题（整型提升）：

下面代码的结果是？（）

```c
#include <stdio.h>
int i;
int main()
{
    i--;
    if (i > sizeof(i))
    {
        printf(">\n");
    }
    else
    {
        printf("<\n");
    }
    return 0;
}
```

A. >

B. <

C. 不输出

D. 程序有问题

解析：

首先定义了一个全局变量i，并且无初始值，编译器会默认将其初始化为0。之后i--，i变为-1。sizeof(i)的值为4，而i为-1，此时如果以为选择B就万事大吉了的话，那也太小看此题了。我们知道sizeof的返回值是size_t类型也就是无符号整型（unsigned int），而i为整型（int）。两边数据类型不一样，就要将整型提升为无符号整型，-1就变成了0xFF FF FF，这是一个比4大得多的数。所以最后的答案应该选择A。

第四题（野指针）：

下面哪些描述是正确的？（）

A. 内存中每个bit位都有唯一的地址。

B. 内存中每个字节都有地址，地址可以唯一标识一个内存单元的。

C. C语言中地址就是指针，指针就是地址。

D. C语言中只要有了地址就可以随意读写内存空间。

解析：

A: 内存中每个字节都有唯一的地址，说法错误。

B: 正确。

C: 正确。

D: 有了地址但是没有给你分配空间就成了野指针，不能随意读写内存空间（非法访问）。