本题B选项考查转义字符，有如下格式，但八进制数字是0-7，没有8，故B选项中'\8'是错误的
\ddd　　ddd表示1到3个八进制数　　如：\130　　转义为　　字符X
\xhh　　hh表示1到2位十六进制数　　如：\x30　　转义为　　字符0

3、test.c文件中包括如下语句，文件中定义的四个变量中，是指针类型的变量为【多选】（）

```
#define INT_PTR int*
typedef int* int_ptr;
INT_PTR a, b;
int_ptr c, d;
```

A:a B:b C:c D:d

答案解析:
正确答案：ACD
因为#define是宏定义，仅仅是直接替换,INT_PTR a, b; 进行宏替换后代码是这样的：int *a, b;这里的int *是a的类型，b的类型是int，故此次b只是int类型。而typedef是把该类型定义一个别名，别名是一个独立的类型了，使用这个类型创建的变量都是这个类型的。
所以 a，c，d才是指针类型

一、动态建立数组

代码：

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

int main()
{
    int n = 10;
    int *p = (int*)malloc(n * sizeof(int));//建立可以存放是个int型数据的数组
    //数组赋值
    for (int i = 0; i<n; i++)
    {
        p[i] = i;
    }
    //数组打印
    for (int i = 0; i<n; i++)
    {
        printf("%d ", p[i]);
    }
    system("pause");
    return 0;
}
```

函数说明(stdlib.h)

1、malloc()函数的形参是要分配的字节数，注意是字节数，不是单元数。

2、malloc 函数返回的是一个无类型的首地址，因此必须在 malloc 函数前加上类型强转，转换为自己需要的数据类型。

3、因为 malloc 函数只负责分配内存，所以我们需要自己定义相应的类型变量来接收 malloc 分配的内存，注意 malloc 函数返回的是分配内存的首地址，所以接收的变量也应该是相应的指针类型。

4、在 C 语言中，sizeof() 是一个判断数据类型或者表达式长度的运算符。

注意：动态数组内存是在堆区分配，堆区有 2G 内存

原文链接：https://blog.csdn.net/famur/article/details/104973909

地址和数组应该一样的比如 int *arr，这是一个指针放的地址，但是可以用 arr[i];

三、数组扩容

代码：

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

int main()
{
    int n = 10;
    int *p = (int*)malloc(n * sizeof(int));
    //数组赋值
    for (int i = 0; i<n; i++)
    {
        p[i] = i;
    }
    //数组打印
    for (int i = 0; i<n; i++)
    {
        printf("%d ", p[i]);
    }

    p = (int*)realloc(p, sizeof(int)* 20);
    //数组打印
    for (int i = 0; i<n; i++)
    {
        printf("%d ", p[i]);
    }
    system("pause");
    return 0;
```

示例：

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 请按任意键继续. . .
```

函数说明：

1、realloc函数第一个形参是：对以前指定的指针内存块

2、realloc函数第二个形参是：新的大小 (以字节为单位)。

3、realloc函数返回的也是无类型的地址，因此自己定义相应的类型变量来接收realloc分配的内存

4、realloc函数申请的扩充空间原数据依次覆盖。

注意：

1）如果当前内存段后面有需要的内存空间，则直接扩展这段内存空间，realloc()将返回原指针。

2）如果当前内存段后面的空闲字节不够，那么就使用堆中的第一个能够满足这一要求的内存块，将目前的数据复制到新的位置，并将原来的数据块释放掉，返回新的内存块位置。

3）如果申请失败，将返回NULL，此时，原来的指针仍然有效。

下面记录一下各种位运算符操作：

- `&` 与运算 两个位都是 1 时，结果才为 1，否则为 0，如

```
  1 0 0 1 1
& 1 1 0 0 1
-----------------------------
  1 0 0 0 1
```

- `|` 或运算 两个位都是 0 时，结果才为 0，否则为 1，如

```
  1 0 0 1 1
| 1 1 0 0 1
-----------------------------
  1 1 0 1 1
```

- `^` 异或运算，两个位相同则为 0，不同则为 1，如

```
  1 0 0 1 1
^ 1 1 0 0 1
-----------------------------
  0 1 0 1 0
```

- `~` 取反运算，0 则变为 1，1 则变为 0，如

```
~ 1 0 0 1 1
-----------------------------
  0 1 1 0 0
```

- 按位取反技巧

> 无论正负数，先对数值本身加1，再改变符号位

```
1  12   符号位取反：数值加1（12+1=13），  正变负，  即-13；
2  -12  符号位取反：数值加1（-12+1=-11），负变正，  即11；
```

- `<<` 左移运算，向左进行移位操作，高位丢弃，低位补 0，如

```
int a = 8;
a << 3;
移位前：0000 0000 0000 0000 0000 0000 0000 1000
移位后：0000 0000 0000 0000 0000 0000 0100 0000
```

- `>>` 右移运算，向右进行移位操作，对无符号数，高位补 0，对于有符号数，高位补符号位，如

```
unsigned int a = 8;
a >> 3;
移位前：0000 0000 0000 0000 0000 0000 0000 1000
移位后：0000 0000 0000 0000 0000 0000 0000 0001

int a = -8;
a >> 3;
移位前：1111 1111 1111 1111 1111 1111 1111 1000
移位前：1111 1111 1111 1111 1111 1111 1111 1111
```

.0.