

[TOC]

单选题

1. Linux中使用 **mkdir**命令创建新的目录时，在其父目录不存在时先创建父目录的选项是（ ）

A -m

B -d

C -f

D -p

D

mkdir [选项] [目录]

-m --mode=模式，建立目录的时候同时设置目录的权限。

-p --parents 若所建立的上层目录目前尚未建立，则会一并建立上层目录。

-v --verbose 每次创建新目录都显示信息。

-h --help 帮助信息。

2. 下面代码创建了多少个进程（不包含main进程本身）（ ）

```
int main(int argc, char* argv[])
{
    fork();
    fork() && fork() || fork();
    fork();
}
```

A 19

B 30

C 24

D 29

A

这道题主要考了两个知识点，一是逻辑运算符运行的特点；二是对fork的理解。

如果有一个这样的表达式：cond1 && cond2 || cond3 这句代码会怎样执行呢？

1、cond1为假，那就不判断cond2了，接着判断cond3。

2、cond1为真，这又要分为两种情况：

2.1 cond2为真，这就不需要判断cond3了。

2.2 cond2为假，那还得判断cond3。

fork调用的一个奇妙之处在于它仅仅被调用一次，却能够返回两次，它可能有三种不同的返回值：

1、在父进程中，fork返回新创建子进程的进程ID。

2、在子进程中，fork返回0。

3、如果出现错误，fork返回一个负值（题干中说明了不用考虑这种情况）。

在fork函数执行完毕后，如果创建新进程成功，则出现两个进程，一个是子进程，一个是父进程。在子进程中，fork函数返回0，在父进程中，fork返回新创建子进程的进程ID。我们可以通过fork返回的值来判断当前进程是子进程还是父进程。

有了上面的知识之后，下面我们来分析fork() && fork() || fork()会创建几个新进程。

很明显fork() && fork() || fork()创建了4个新进程。

总结：

第一行fork生成1个新进程。

第二行的三个fork生成4+4=8个新进程。

第三行的fork会生成10个新进程(这是因为前面总共有10个进程,调用一次fork生成10个新进程)。

所以一共会生成1+8+10=19个新进程。

3.如果下列公式成立： $3A \cdot 124 = 446C$ 。则采用的是（ ）进制

A 11

B 12

C 14

D 16

C

看个位。最后的结果446C个位为C，因此，A，B可以排除。

假设为14进制， $(A \cdot 4) \% 14 = 12$ ，结果正好为C。因此，答案为14进制。

4.下面关于字符数组的初始化，那个是正确的？（ ）

A `char b[2][3] = {"d", "e", "f"};`

B `char b[2][3] = {"d", "e"};`

C `char b[2][3] = {{ "d", "e", "f"}, {"a", "b", "c"} };`

```
Dchar b[2] ={"d","e"};
```

B

通常情况下，二维数组的每一行分别使用一个字符串进行初始化。例如：

```
char c[3][8]={{"apple"},"orange"},"banana"}};
```

等价于：

```
char c[3][8]="apple","orange","banana");
```

A:应改为char b[3][2] ={"d","e","f"};

C:应改为char b[2][3][2] ={{"d","e","f"},"a","b","c"}};

D:应改为 char b[2][2]={"d","e"};

5.在32位系统中，下列类型占用8个字节的为（ ）

A int

B unsigned long long

C char

D short int

B

32位操作系统

int：4字节

unsigned long long：8字节

char：1字节

short int：2字节

注意和64位操作系统的区别：64位系统中，指针变量和long以及unsigned long 都占8个字节，其他的和32位系统一样

简答

1.

```
int a[6] = {1,2,3,4,5,6};
printf("%d\n",*((int*)&a+1)-1));
```

那么打印结果是什么？

6。

将 `*((int*)&a+1)-1)` 化简为：`*(p-1)` 和 `p=(int*)&a+1`

`&a` 是一个指向 `int(*)[6]` 的指针。由于 `&a` 是一个指针，那么在32位机器上，`sizeof (&a) = 4`，但是 `**&a+1` 的值取决于 `&a` 指向的类型**，由于 `&a` 指向

`int(*)[6]`，所以 `&a+1 = &a + sizeof (int(*)[6]) = &a+24`。`&a` 是数组的首地址，由此 `&a+1` 表示 `a` 向后移动了 **6个int** 从而指向 `a[6]`（越界），所以 `p` 指向的是 `a[6]`。

由于 `p` 是 `int *` 类型，那么 `p-1` 就指向 `a[5]`，所以 `*(p-1) = a[5] = 6`。

2. 请写出常量指针和指针常量的代码形式，并简述他们的区别

`int const *p1;` `const` 在前，定义为常量指针

`int *const p2;` `*` 在前，定义为指针常量

常量指针 `p1`：指向的地址可以变，但内容不可以重新赋值，内容的改变只能通过修改地址指向后变换。

指针常量 `p2`：指向的地址不可以重新赋值，但内容可以改变，必须初始化，地址跟随一生。

3. 如何避免头文件被重复包含

1. 条件编译：

```
#ifndef _HEADERNAME_H
#define _HEADERNAME_H

...//(头文件内容)

#endif
```

2. #pragma once

指定当前文件在构建时只被包含(或打开)一次，这样就可以减少构建的时间，因为加入 `#pragma once` 后，编译器在打开或读取第一个 `#include` 模块后，就

不会再打开或读取随后出现的相同 `#include` 模块。

4. 运行 `char name[] = "/dev/spdev"` 后，系统会分配几块内存，这些内存共占多少个字节？

11字节。

字符串最后以 `\0` 结尾，共占据11字节。

5. 如下代码，请设计宏定义 `STR(x)`，将 `USART_RATE` 转换成字符串并打印出来

```
#define USART_RATE 115200
#define STR(x)____?
printf("usart rate = %s\n", STR(USART_RATE));
```

```
#define STR(x) #x
```

#：会把参数转换为字符串

6.已知结构体成员d的地址为p1，请获取成员变量b的地址。

```
struct data{
    int a;
    char b;
    short c;
    short d;
    int e;
};
```

```
p1 - 4 * sizeof(p1)
```

结构体中成员变量在内存中存储的其实是偏移地址。也就是说结构体的首地址+成员变量的偏移地址 = 结构体成员变量的起始地址。具体如下图所示。



结构体在内存中的存储

因此，将指针p1向上移动4个单位即可。即 $p1 - 4 * \text{sizeof}(p1)$

7.请写出下列代码的输出结果

```
int main(int argc, char *argv[])
{
    char *buff[] = {"char", "int", "double"};
    printf("%c\n", *(buff+1)[1]);
    return 0;
}
```

d

buff是指针数组，一个3个元素的数组，数组里面是个字符串指针，这里执行buff+1时，则buff指向下一个数组元素，即int。

因此，*(buff+1)[0]指向int的地址，*(buff+1)[1]指向double的地址，而最后是输出一个字符。所以，输出d。

8.下面的代码输出什么？为什么？

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

>6

C中运算有规定，如果整型变量间进行数据运算，只要有一个变量是无符号的，结果就按无符号数据输出，因此 $a+b > a$ 结果会输出 >6

编程题

表一：人员信息表：(**info_table**)

序号 (num)	姓名 (name)	性别 (gender)	年龄 (age)
0	张三 (Bob)	男 (man)	60
1	李四 (Chris)	男 (man)	30
2	小红 (Colin)	女 (woman)	56

表二：人员职业表：(**work_table**)

序号 (num)	姓名 (name)	职业 (work)	等级 (level)
0	张三 (Bob)	司机 (driver)	9
1	李四 (Chris)	厨师 (chief)	3
2	小红 (Colin)	幼师 (teachers)	6

表三：技能成绩表：(**grade_table**)

序号 (num)	姓名 (name)	技能 (skill)	成绩 (grade)
0	张三 (Bob)	开车 (drive)	50
1	李四 (Chris)	烹饪 (cook)	64
2	小红 (Colin)	教学 (teaching)	55

如上所示有人员-职业-成绩的三个关系表

请尝试

- 1.使用结构体表示三个表格。
- 2.设计一个函数来依次录入人员信息

如add_personnel(info_table,work_table,grade_table);

```
/*
 * @Description: 兆易创新编程题
 * @Version:
 * @Autor: 公众号【嵌入式与Linux那些事】
 * @Date: 2021-04-03 21:46:16
```

```

* @LastEditors: 公众号【嵌入式与Linux那些事】
* @LastEditTime: 2021-04-03 22:03:38
*/
#include <stdio.h>
//人员信息表
typedef struct info_table{
    char name[6];
    char gender[5];
    int age;
}INFO;
//人员职业表
typedef struct work_table{
    char name[6];
    char work[10];
    int level;
}WORK;
//技能成绩表
typedef struct grade_table{
    char name[6];
    char skill[10];
    int grade;
}GRADE;
/**
 * @Description: 依次录入人员信息
 * @param {INFO} *pinfo
 * @param {WORK} *pwork
 * @param {GRADE} *pgrade
 * @Return: 无
 * @Author: 公众号【嵌入式与Linux那些事】
 */
void add_personnel(INFO *pinfo,WORK *pwork,GRADE *pgrade){
    int i;
    for(i = 0;i < 3;i++){
        printf("请依次输入第%d个人的信息：姓名·性别·年龄\n",i+1);
        scanf("%s%s%d",pinfo[i].name,pinfo[i].gender,&pinfo[i].age);
        printf("%s,%s,%d\n",pinfo[i].name,pinfo[i].gender,pinfo[i].age);
    }
    for(i = 0;i < 3;i++){
        printf("请依次输入第%d个人的信息：姓名·职业·等级\n",i+1);
        scanf("%s%s%d",pwork[i].name,pwork[i].work,&pwork[i].level);
        printf("%s,%s,%d\n",pwork[i].name,pwork[i].work,pwork[i].level);
    }
    for(i = 0;i < 3;i++){
        printf("请依次输入第%d个人的信息：姓名·技能·成绩\n",i+1);
        scanf("%s%s%d",pgrade[i].name,pgrade[i].skill,&pgrade[i].grade);
        printf("%s,%s,%d\n",pgrade[i].name,pgrade[i].skill,pgrade[i].grade);
    }
}
int main()
{
    INFO info_array[3];
    WORK work_array[3];
    GRADE grade_array[3];
    add_personnel(info_array,work_array,grade_array);
}

```

```
    return 0;  
}
```