2015网易游戏校园招聘笔试题 游戏插件研发岗

一. 单项选择题

1. 用命令（ ）可以查看mysql数据库中user表的表结构？

A desc user;

B show create table user;

C show columns for user;

D describe user;

2. tcp三次握手创建连接，双方交互的报文中SYN和ACK的序列是什么样的（ ）

A SYN，SYN+ACK，ACK

B SYN，ACK，SYN，ACK

C SYN+ACK，ACK，SYN

D SYN，SYN，ACK

3. 函数参数使用的空间是在（ ）中申请的，malloc或new是在（ ）中申请空间的？

A 堆，栈

B 栈，堆

C 栈， 栈

D 堆，堆

4. 有B+Tree/Hash_Map/STL Map三种数据结构。对于内存中数据，查找性能较好的数据结构是（ ），对于磁盘中数据，查找性能较好的数据结构是（ ）。

A Hash_Map/B+Tree

B STL_Map/B+Tree

C STL_Map/Hash_Map

D B+Tree/Hash_Map

5. 由源代码生成可执行文件需要经过预编译，编译，汇编，链接等阶段，错误：unresolved external symbol BeginScene属于[$##$]阶段错误。

A 预编译

B 编译

C 汇编

D 链接

二. 多选选择题

6. 下面属于进程间通信的有？

A 管道

B 消息队列

C 内存共享

D 套接字

7. 下面关于ISO网络参考模型分层及每一层功能描述错误的有？

A 物理层，在此层将数据分帧，并处理流控制

B 数据链路层，为物理层提供连接，以便透明的传送比特流

C 网络层，本层通过寻址来建立两个节点之间的连接，为源端的运输层送来的分组，选择合适的路由和交换节点

D 运输层，常规数据递送－面向连接或无连接

E 会话层，在两个节点之间建立端连接。

F 表示层，主要用于处理两个通信系统中交换信息的表示方式。

8. 以下对TCP和UDP区别的描述哪些是正确的（ ）

A TCP是无序数据传输，UDP不是

B TCP重发丢失的IP包，UDP不是

C TCP是传输流的协议，而UDP不是

D TCP面向连接，而UDP不是

三. 填空题

9.

```
char *p1;int 64_t *p2;
p1=(char *)0x800000;
p2=(int 64_t *)0x800000;
char *a=p1+2
int 64_t *b=p2+2
```

那么a=(),b=()

10. 有一个数组（53,83,18,59,38,35），依次将其存储在hash表中，其中哈希函数为h(k)=k%7,如采用线性探测（每次向后查找1位）的方式解决冲突，则该hash表上查找38,35,53访问hash表的表项次数分别为()，()，()。

11. 32位系统上

```
char c1[]={'a','b','\0','d','e'};
char c2[]="hello";
```

sizeof(c1),strlen(c1),sizeof(c2),strlen(c2)值分别是()()()()。

四. 问答题

12.
使用C/C++语言写一个函数，实现字符串的反转，要求不能用任何系统函数，且时间复杂度最小。
函数原型是：char *reverse_str(char *str)

13. 在SQL中，一个表的定义如下：

```
CREATE TABLE t_account(
    account varchar(100),
    account_type TINYTEXT,
    PRIMARY KEY (account),
};
```

account为账号，account_type为该账号的类型，写出一个sql，统计账号数累计超过5000个账号类型，并显示对应的账号数，即结果中每行是（账号类型，账号数）

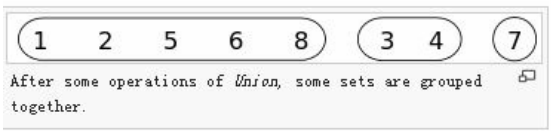
14. http状态码中，()表示访问成功，()表示坏请求，()表示服务不可用

15. 现有/home/script/check.sh脚本，要求每周一到周五14点内每三分钟运行一次，相应的crontab配置是()

16. 请找出下面用于拷贝内存的代码中的逻辑错误，并修正。

```
void memcpy(const char* src,char* dest){
    int len=strlen(src);
    dest=(char*)malloc(len);
    char* d=dest;
    char* s=src;
    while(len--!=0){
        *d=*s;
        d++;
        s++;
    }
}
```

17. **Disjoint-set data structure**



In computing, a disjoint-set data structure, also called a union–find data structure or merge–find set, is a data structure that keeps track of a set of elements partitioned into a number of disjoint (nonoverlapping) subsets. It supports two useful operations:

Find: Determine which subset a particular element is in. Find typically returns an item from this set that serves as its "representative"; by comparing the result of two Find operations, one can determine whether two elements are in the same subset.
Union: Join two subsets into a single subset.
The other important operation, MakeSet, which makes a set containing only a given element (a singleton), is generally trivial. With these three operations, many practical partitioning problems can be solved (see the Applications section).

In order to define these operations more precisely, some way of representing the sets is needed. One common approach is to select a fixed element of each set, called its representative, to represent the set as a whole. Then, Find(x) returns the representative of the set that x belongs to, and Union takes two set representatives as its arguments.

Disjoint-set forests
Disjoint-set forests are data structures where each set is represented by a tree data structure, in which each node holds a reference to its parent node (see spaghetti stack).
In a disjoint-set forest, the representative of each set is the root of that set's tree. Find follows parent nodes until it reaches the root. Union combines two trees into one by attaching the root of one to the root of the other.

Question 1：According to the information above,implement three functions:MakeSet(),Fins(),Union().You can use C/C++/Python/Java.

In this naive form, this approach is no better than the linked-list approach, because the tree it creates can be highly unbalanced; however, it can be enhanced in two ways.

The first way, called union by rank, is to always attach the smaller tree to the root of the larger tree, rather than vice versa. Since it is the depth of the tree that affects the running time, the tree with smaller depth gets added under the root of the deeper tree, which only increases the depth if the depths were equal. In the context of this algorithm, the term rank is used instead of depth since it stops being equal to the depth if path compression (described below) is also used. One-element trees are defined to have a rank of zero, and whenever two trees of the same rank r are united, the rank of the result is r+1. Just applying this technique alone yields a worst-case running-time of O(\log n) per MakeSet, Union, or Find operation.

The second improvement, called path compression, is a way of flattening the structure of the tree whenever Find is used on it. The idea is that each node visited on the way to a root node may as well be attached directly to the root node; they all share the same representative. To effect this, as Find recursively traverses up the tree, it changes each node's parent reference to point to the root that it found. The resulting tree is much flatter, speeding up future operations not only on these elements but on those referencing them, directly or indirectly.

Question 2：Implement the above improvements.You can use C/C++/Python/Java.

Question 3：You can answer this question in Chinese.What are the real world applications of this data structures?

18. 有如下表结构：

```
CREATE TABLE 'game_stats'(
   'uid' int(11) NOT NULL COMMENT '玩家 uid',
```

```
    'host' int(11) NOT NULL COMMENT '玩家服务器',
    'username' varchar(128) NOT NULL COMMENT '玩家账号',
    'log_date' date NOT NULL COMMENT '消费日期',
    'login_long' int(11) NOT NULL DEFAULT '0' COMMENT '登录时长',
    'yuanbao_spend' int(11) NOT NULL DEFAULT '0' COMMENT '消费元宝',
    PRIMARY KEY('uid','host','log_date'),
}ENGINE=InnoDB DEFAULT CHARSET=UTF8

CREATE TABLE 'customer'(
    'id' int(11) NOT NULL AUTO_INCREMENT,
    'username' varchar(50) NOT NULL COMMENT '玩家账号',
    'product' int(8) NOT NULL DEFAULT '0' COMMENT '产品',
    'vipflag' tinyint(2) NOT NULL DEFAULT '1' COMMENT 'VIP等级',
    PRIMARY KEY('id'),
}ENGINE=innoDB DEFAULT CHARSET=utf8
```

假设：
(uid,host)是表示一个游戏角色的唯一key

问题：
1.编写SQL,更新customer表的VIP等级，如果表内无对应账号，则插入。要求VIP等级只升不降。输入为：（玩家账号："u1",产品："p1",VIP等级："v1"）。（输入的VIP等级有可能比表内的对应VIP等级低）
2.编写SQL，列出在2014年8月，所有游戏角色中消费元宝最多的10个游戏角色及其总消费。
3.一个玩家账号下可能有一个或多个游戏角色，编写SQL，列出所有玩家账号及其在2014-08-01的账号登陆总时长，账号总购买元宝数。
4.为了满足上述各题的需求及提高查找性能，这两个表需要建什么索引？

19.
有pqueue.h如下

```
#ifndef HEADER_PQUEUE_H
#define HEADER_PQUEUE_H
typedef struct_pqueue{
    pitem *items;
     int count;
}pqueue_s;
typedef struct_pqueue *pqueue;
typedef struct_pitem{
    unsigned char priority[8];
    void *data;
    struct_pitem *next;
}pitem;
typedef struct_pitem *piterator;
pitem *pitem_new(unsigned char *prio64be,void *data);
void pitem_free(pitem *item);

pqueue pqueue_new(void);
void pqueue_free(pqueue pq);
pitem *pqueue_insert(pqueue pq,pitem *item);
```

```
pitem *pqueue_peek(pqueue pq);
pitem *pqueue_pop(pqueue pq);
pitem *pqueue_find(pqueue pq,unsigned char *prio64be);
pitem *pqueue_iterator(pqueue pq);
pitem *pqueue_next(piterator *iter);
int pqueue_size(pqueue pq);
#endif /*! HEADER_PQUEUE_H */
```

pq_test.c如下：

```
#include
#include
#include
#include"pqueue.h"
/*remember to change expected.txt if you change there values*/
unsigned char prio1[8]="supercal";
unsigned char prio2[8]="ifragili";
unsigned char prio3[8]="sticexpi";
static void
pqueue_print(pqueue pq)
{
    pitem *iter,*item;
    iter=pqueue_iterator(pq);
    for(item=pqueue_next(&iter);item!=NULL;
        item=pqueue_next(&iter)){
        printf("item\t%02x%02x%02x%02x%02x%02x%02x%02x\n",
            item ->priority[0],item->priority[1],
            item ->priority[2],item->priority[3],
            item ->priority[4],item->priority[5],
            item ->priority[6],item->priority[7],
        }
}
int main(void)
{
    pitem *item;
    pqueue pq;
    pq=pqueue_new();
    item=pitem_new(prio3,NULL);
    pqueue_insert(pq,item);

    item=pitem_new(prio1,NULL);
    pqueue_insert(pq,item);

    item=pitem_new(prio2,NULL);
    pqueue_insert(pq,item);
    item=pqueue_find(pq,prio1);
    fprintf(stderr,"found %p\n",item->priority);
    item=pqueue_find(pq,prio2);
    fprintf(stderr,"found %p\n",item->priority);

    item=pqueue_find(pq,prio3);
```

```
    fprintf(stderr,"found %p\n",item->priority);

    pqueue_print(pq);
    for(item=pqueue_pop(pq);item!=NULL;item=pqueue_pop(pq))
    pitem_free(item);

    pqueue_free(pq);
    return 0;
}
```

pq_test.sh如下：

```
#!/bin/sh
set -e
./pq_test | cmp $srcdir/pq_expected.txt-
```

pq_expected.txt如下：

```
    item 6966726167696c69
    item 7374696365787069
    item 737570657263616c
```

1.根据测试代码描述pqueue的工作原理。
2.请实现 pitem *pqueue_insert(pqueue pq,pitem *item);