多益网络 2016 春招校园招聘笔试回顾（C++游戏后台开发）

原文地址：http://blog.csdn.net/k346k346/article/details/51253148

2016.04.16 晚中山大学大学城校区（东校区）参加了多益网络的 C++游戏后台开发的笔试。有几道笔试题还是值得斟酌和记录的，特记录如下。比较可惜，因为回老家了，未能参加多益网络的面试。

1.试题汇总

题目一：

给定代码段 int A[2][3]={1,2,3,4,5,6};那么 A[1][0]和*(*(A+1)+1)的值分别是什么?

答：

A[1][0]=4，*(*(A+1)+1)=5。

这里考察了对二维数组的理解和指针运算。A[1][0]=4 比较好理解。但是对二维数组 A 进行指针运算时，我们要知道二维数组 A 的类型是什么，考察如下代码：

```
int A[2][3]={1,2,3,4,5,6};
cout<<"sizeof(A):"<<sizeof(A)<< “ ” <<typeid(A).name()<<endl;
```

VS2012 中代码输出 sizeof(A):24 int [2][3]。可见二维数组 A 的类型是 int[2][3]，所以 sizeof(A)=sizeof(int)*6=24。

知道了 A 的类型是 int[2][3]之后，当我们对数组 A 进行指针运算时，那么 A 就会退化为指针，它的类型变为 int(*)[3]，验证代码如下：

```
cout<<"sizeof(A+1):"<<sizeof(A+1)<<"
"<<typeid(A+1).name()<<endl;
cout<<"sizeof(*(A+1)):"<<sizeof(*(A+1))<<endl;
```

输出结果为：

```
sizeof(A+1):4 int (*)[3]
sizeof(*A):12 int [3]
```

所以*(A+1)表示的是二维数组的第二行，其类型是 int[3]。可将*(A+1)取个别名，容易理解，*(A+1)=int a[3]，此时在对变量*(A+1)进行指针运算时，就相当于对一维数组 a 进行进行指针运算。那么*(a+1)的值就是二维数组 A 的第二行的第二个数 5。

是有点绕，不过一定要好好理解，才能掌握数组与指针之间的区别与联系。

这里有一点一定要记住：当对数组进行指针运算时，其会退化为指针。

题目二：

下面代码的作用是什么?

```
double x,ret=0;
for(int i=1;scanf("%lf",&x)==1;++i){
    ret+=(x-ret)/i;
}
```

答：

这段代码真的很精妙，其作用就是求标准输入双精度浮点数和的平均值。

按照顺序走几遍循环就可以了。比如输入的值为 a，那么结果 ret=a，第二次输入值为 b，那么:

ret=b−a2+a=a+b2

假如第三次输入的是 c，那么：

ret=a+b2+c−a+b23=a+b+c3

以此类推，可以知道上面的代码是求输入双精度浮点数和的平均值。

题目三：

在一个平面坐标系中，从方格（0,0）移动到方格（6,6），每次只能向上移动或者向右移动，且每次只能移动一个方格，且不能经过（2,3）和（4,4）两个方格，有多少种移动的方式。

答：

这道题本质是组合问题。解题思路：

（1）算出从方格（0,0）到方格（6,6）总共有多少种移动的方式；

（2）减去经过（2,3）和（4,4）的所有路径。

从方格（0,0）移动到方格（6,6）的移动次数是 12 次，每次都选择向右还是向上。因此向右只能选择 6 次，所以总的移动次数设为 countAll=C612=924 种。

按照上面的计算方式，（0,0）到（2,3）有 C25 种，再从（2,3）到（6,6）

有 C47 种。所以经过方格（2,3）从（0,0）移动到（6,6）的移动方式 countA=C25C47=350 种。

同理，经过方格（2,3）从（0,0）移动到（6,6）的移动方式 countB=C48C24=420 种。

同理，同时经过（2,3）和（4,4）的移动方式 countAB=C25C13C24=180 种。

因为经过（2,3）的路径中有可能经过（4,4），反之亦然。所以减去 countA 和 countB 时，会多减去一次同时经过（2,3）和（4,4）的移动方式数 countAB，所以最终结果是：

count=countAll - countA - countB+countAB=924 - 350 - 420+180=334$。

题目四：

这是一道代码理解题。给定如下代码片段：

```
void getmemoney(char** p,int num){
    *p=(char*)malloc(num);
}

void test(void){
    char* str=NULL;
    getmemoney(&str,1000);
    strcpy(str,"hello");
```

```
    printf(str);
}
```

问运行 test 函数有什么结果?

答： 这里考察了两点： 第一点：内存泄露； 第二点：strcpy 函数的作用于特点。

运行 test 函数会打印输出 hello，且出现内存泄露。strcpy 函数与是 C 语言标准库函数，把从 src 地址开始且含有 NULL 结束符的字符串复制到以 dest 开始的地址空间。这里要注意的是字符串拷贝结束后，会在目的地址空间最后添加空字符’\0’。

题目五：

这是一道编程题。题目如下： 第五套人民币，中华人民共和国的纸币有 1 元、5 元、10 元、20 元、50 元和 100 元。共 6 种，凑齐 100 元的一种组合是：五张 1 元+一张 5 元+两张 10 元+一张 20 元+一张 50 元。请写一个算法，计算凑齐 100 元的组合的种类数。

答：

方法一：穷举法

解题思路：

我们可以列举所有可能情况。全部用 1 元来凑齐的话，需要一百张；全部用 5 元来凑的话，需要二十张；全部用 10 元，需要十张；全部用 20 元，需要五张；全部用 50 元，需要两张，全部用 100 元，需要一张。

迭代实现：

因此我们可以采用多重循环迭代的方式来求出组成 100 元的所有可能性。

参考如下代码：

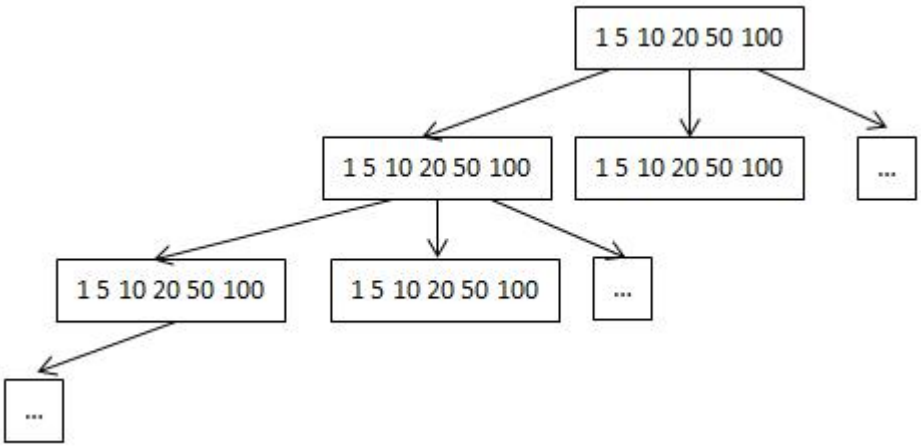
```
int main(){
    int count=0; //组合种类数
    for(int a=0;a<=100;++a){
        for(int b=0;b<=20;++b){
            for(int c=0;c<=10;++c){
                for(int d=0;d<=5;++d){
                    for(int e=0;e<=2;++e){
                        for(int f=0;f<=1;++f){
                            if(1*a + 5*b + 10*c + 20*d + 50*e +
100*f==100)
                                count++;
                        }//end f:100 元
                    }//end e：50 元
                }//end d:20 元
            }//end c:10 元
        }//end b：5 元
    }//end a:1 元

    cout<<"count:"<<count<<endl;
```

}

程序输出： count:344。表明有 344 种组合方式。

递归实现：

列举所有可能的组合，我们可以采用递归的方式来实现。将所有可能的组合可以列举成如下的六叉树形结构：



我们深度遍历这棵六叉树，来统计凑够 100 元的组合数。但是以递归的方式来深度遍历这棵六叉树时需要注意两点：

第一点：回溯。对于每种面值累加厚，在退出当前节点回到上一层节点时需要进行回溯，即减去这一层节点的纸币面值。

第二点：避免重复。在深度遍历时，如果全部遍历的话，会出现重复组合的情况。比如以面值 1 开始递归遍历，有一种组合方式是 1,1,1…1,5，从头结点开始再以 5 开始递归遍历会出现 5,1,1,1…1。这两种组合其实是同一种组合方式，如何避免这种重复计数呢?

以 1 开始遍历，其实是统计了所有包含 1 组成 100 的左右可能情况。这时候，再以 5 开始遍历的时候，我们就不应该再去遍历包含 1 的所有可能的组合。所以要给定节点内的下标，表示当前遍历时节点内的起始值是什

么。比如再以头结点的 5 开始遍历时，下面每一层节点内的遍历起点都是从 5 开始，而不能从 1 开始。

参考如下代码：

```
int rmb[6]={1,5,10,20,50,100};
int count=0;//组合数
//index：表示第几个纸币，即节点内下标
void getCombinationNum(int& sum,int index){
    for(int i=index;i<6;++i){
        sum+=rmb[i];
        if(sum<100)
            getCombinationNum(sum,i);
        if(sum==100){
            ++count;
        }
        sum-=rmb[i]; //回溯
    }
}

int main(){
    int sum=0; //币值累加和
    getCombinationNum(sum,0);
    cout<<"count:"<<count<<endl;
```

```
}
```

程序输出：count：344 种。

递归与迭代实现的对比：

使用递归的方式来实现穷举所有可能的组合，代码实现上较为简洁，但是递归带来的多重的函数调用增加了运行时开销，效率次于迭代实现，并且不太容易理解。所以建议使用迭代的方式来实现穷举。

方法二：动态规划法

考察组成 100 元的方式，可以从高面值往低面值开始拆分。对于 100 元面值的纸币，组成 100 元的方式要么包含 100 元面值的纸币，要么不包含这两种情况。

所以可以设 f(n,j)表示价值为 n 的金额由包含第 0 到第 j 种面值组成的所有情况数。那么 f(n,j)分为两种情况，包含第 j 种面值，和不包含第 j 种面值情况，那么 f(n,j)=f(n−v[j],j)+f(n,j−1)。其中 f[n,j−1]表示没有第 j 种纸币的情况的总和，f(n−v[j],j)表示去掉一张第 j 中纸币面值后剩余面值由第 0 到第 j 种面值组成的所有情况数。特别的，当 n=0 时，f(0,j)=1。

有了上面的递归式，我们知道 f(100,5)就是我们要求的组成 100 元由第 0 种纸币 1 元到第 5 种纸币 100 元组成的种类数。

实现参考如下代码：

```
const int v[6] = {1,5,10,20,50,100};


int f(int n, int w)
{
```

```
    if(n<0) return 0;
    if(n==0) return 1;
    if(w<0) return 0;

    return f(n, w−1) + f(n−v[w], w);
}

int main(){
    cout<<"count:"<<f(100,5)<<endl;
}
```

输出结果：count:344。