进程控制块是描述进程状态和特性的数据结构，一个进程() 。
正确答案: D 你的答案: 空 (错误)

可以有多个进程控制块；
可以和其他进程共用一个进程控制块；
可以没有进程控制块；
只能有惟一的进程控制块。

在操作系统中,把逻辑地址转变为内存的物理地址的过程称作()。
正确答案: D 你的答案: 空 (错误)

编译；
连接；
运行；
重定位。

某计算机系统中有 8 台打印机,由 K 个进程竞争使用,每个进程最多需要 3 台打印机。该系统可能会发生死锁的 K 的最小值是()
正确答案: C 你的答案: 空 (错误)

2
3
4
5

下面关于线程的叙述中，正确的是( )。
正确答案: C 你的答案: 空 (错误)

不论是系统支持线程还是用户级线程，其切换都需要内核的支持
线程是资源的分配单位，进程是调度和分配的单位
不管系统中是否有线程，进程都是拥有资源的独立单位
在引入线程的系统中，进程仍是资源分配和调度分派的基本单位

程序动态链接发生时刻是在()
正确答案: B 你的答案: 空 (错误)

编译时
装入时
调用时
程序执行时

以下哪些进程状态转换是正确的
正确答案: A B C E 你的答案: 空 (错误)

就绪到运行

运行到就绪
运行到阻塞
阻塞到运行
阻塞到就绪

IP 地址分类中,C 类地址的范围为：
正确答案: C 你的答案: 空 (错误)

以 0 开头， 第一个字节范围：0~127
以 10 开头， 第一个字节范围：128~191；
以 110 开头， 第一个字节范围：192~223；
以上答案都不正确

FTP 服务和 SMTP 服务的端口默认分别是（）
正确答案: C 你的答案: 空 (错误)

20 与 25
21 与 25
20，21 与 25
20 与 21

一个 B 类网的子网掩码是 255.255.240.0，这个子网能拥有的最大主机数是：
正确答案: C 你的答案: 空 (错误)

240
255
4094
65534

如果在一个建立了 TCP 连接的 socket 上调用 recv 函数，返回值为 0，则表示（）
正确答案: B 你的答案: 空 (错误)

对端发送了一段长度为 0 的数据
对端关闭了连接
还没有收到对端数据
连接发生错误

面有关 http keep-alive 说法错误的是？
正确答案: D 你的答案: 空 (错误)

在 HTTP1.0 和 HTTP1.1 协议中都有对 KeepAlive 的支持。其中 HTTP1.0 需要在 request 中增加“Connection： keep-alive” header 才能够支持，而 HTTP1.1 默认支持当使用 Keep-Alive 模式时，Keep-Alive 功能使客户端到服 务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive 功能避免了建立或者重新建立连接

可以在服务器端设置是否支持 keep-alive
当你的 Server 多为动态请求，建议开启 keep-alive 增加传输效率

某台路由器有两个以太网接口，分别与不同网段的以太网相连，请问：该路由器最多可有几组？（ ）
正确答案: E 你的答案: 空 (错误)

1
2
3
4
>4

a 边长为 n 的正方形可以分成多个边长为 1 的正方形，如边长为 2 的正方形有 2×2 个边长为 1 的正方形和 1 个边长为 2 的正方形；问边长为 5 的正方形有几个正方形。
正确答案: C 你的答案: 空 (错误)

25
30
55
100

A 市 B，C 两个区，人口比例为 3：5，据历史统计 B 区的犯罪率为 0.01%，C 区为 0.015%，现有一起新案件发生在 A 市，那么案件发生在 B 区的可能性有多大？（）
正确答案: C 你的答案: 空 (错误)

37.5%
32.5%
28.6%
76.9%

下面程序的功能是输出数组的全排列,选择正确的选项,完成其功能。

```
1    void perm(int list[],  int k,  int m)
2    {
3    if (      )
4    {
5            copy(list,list+m,ostream_iterator<int>(cout," "));
6            cout<<endl;
7            return;
8    }
9    for (int i=k; i<=m; i++)
10   {
11           swap(&list[k],&list[i]);
12           (      );
```

```
13            swap(&list[k],&list[i]);
14    }
15    }
```

正确答案: B 你的答案: 空 (错误)

```
k!=m 和 perm（list，k+1，m）
k==m 和 perm（list，k+1，m）
k!=m 和 perm（list，k，m）
k==m 和 perm（list，k，m）
```

在 Linux 中 crontab 文件由 6 个域组成，每个域之间用空格分隔，下列哪个排列方式是正确的？

正确答案: B 你的答案: 空 (错误)

```
MIN HOUR DAY MONTH YEAR COMMAND
MIN HOUR DAY MONTH DAYOFWEEK COMMAND
COMMAND HOUR DAY MONTH DAYOFWEEK
COMMAND YEAR MONTH DAY HOUR MIN
```

在 Linux 系统中哪个文件定义了服务搜索顺序？ 。

正确答案: C 你的答案: 空 (错误)

```
/etc/services
/etc/nsorder
/etc/nsswitch.conf
/etc/hosts
```

下列关于链接描述，正确的的是（）

正确答案: A C D 你的答案: 空 (错误)

硬链接就是让链接文件的 i 节点号指向被链接文件的 i 节点
硬链接和符号连接都是产生一个新的 i 节点
链接分为硬链接和符号链接
硬连接不能链接目录文件

在 Linux 系统中,下列哪个命令是用来统计一个文件中的行数?

正确答案: B 你的答案: 空 (错误)

```
lc
wc - l
cl
count
```

Linux 执行 ls，会引起哪些系统调用（）

正确答案: B C　你的答案: 空 (错误)

nmap
read
execve
fork

卢卡斯的驱逐者大军已经来到了赫柏的卡诺萨城，赫柏终于下定决心,集结了大军,与驱逐者全面开战。
卢卡斯的手下有 6 名天之驱逐者，这 6 名天之驱逐者各赋异能，是卢卡斯的主力。
为了击败卢卡斯，赫柏必须好好考虑如何安排自己的狂战士前去迎战。
狂战士的魔法与一些天之驱逐者的魔法属性是相克的，第 i 名狂战士的魔法可以克制的天之驱逐者的集合为 Si(Si 中的每个元素属于[0,5])。
为了公平，两名狂战士不能攻击同一个天之驱逐者。
现在赫柏需要知道共有多少种分派方案。
例：
S1={01},S2={23}，代表编号为 0 的狂战士的魔法可以克制编号为 0 和编号为 1 的天之驱逐者，编号为 1 的狂战士的魔法可以克制编号为 2 和编号为 3 的天之驱逐者，共有四种方案：02,03,12,13。
02---代表第一个狂战士负责编号为 0 的驱逐者，第二个狂战士负责编号为 2 的驱逐者；
03---代表第一个狂战士负责编号为 0 的驱逐者，第二个狂战士负责编号为 3 的驱逐者；
12---代表第一个狂战士负责编号为 1 的驱逐者，第二个狂战士负责编号为 2 的驱逐者;
13---代表第一个狂战士负责编号为 1 的驱逐者，第二个狂战士负责编号为 3 的驱逐者;
S1={01},S2={01}，代表编号为 0 的狂战士的魔法可以克制编号为 0 和编号为 1 的天之驱逐者，编号为 1 的狂战士的魔法可以克制编号为 0 和编号为 1 的天之驱逐者，共有两种方案：01,10。

```
1    //非递归解法和递归解法，有兴趣的可以参考下
2    // letv1.cpp : 定义控制台应用程序的入口点。
3    //这道题的非递归解法，有兴趣的可以参考一下
4
5    //#include "stdafx.h"
6    #include <iostream>
7    #include <vector>
8    #include <string>
9    using namespace std;
10   struct node
11   {
12           string * ps;
13           int i;
14   };
15   int findNum(vector<string> data)
16   {
```

```
17          if (data.size() == 1)
18          {
19                  return data[0].size();
20          }
21          vector<struct node *> stk;
22          vector<struct node *>::iterator it;
23          struct node * pNode =  new struct node;
24          struct node * pTmp;
25          pNode->ps = &data[0];
26          pNode->i = 0;
27          int j = 1;
28          string * now = &data[1];
29          int k = 0;
30          int num = 0;
31          stk.push_back(pNode);
32          int m;
33          while (true)
34          {//while1
35                  for (m = k; m < now->size(); m++)
36                  {
37                          bool flag =  true;
38                          for (it = stk.begin(); it != stk.end(); it++)
39                          {
40                                  if ((*((*it)->ps))[(*it)->i] == (*now)[m])
41                                  {
42                                          flag =  false;
43                                          break;
44                                  }
45                          }
46                          if (flag ==  true)
47                          {
48                                  pNode =  new struct node;
49                                  pNode->ps = now;
50                                  pNode->i = m;
51                                  stk.push_back(pNode);
52                                  break;
53                          }
54                  }
55                  if (m < now->size())
56                  {
57                          if (j < data.size() - 1)
58                          {
59                                  j++;
60                                  k = 0;
```

```
61                              now = &data[j];
62                      }
63                      else
64                      {
65                              if (m == now->size() - 1)
66                              {
67
68                                      it = stk.end() - 1;
69                                      pTmp = *it;
70                                      stk.erase(it);
71                                      delete pTmp;
72                                      it = stk.end() - 1;
73                                      pTmp = *it;
74                                      k = (*it)->i + 1;
75                                      now = (*it)->ps;
76                                      stk.erase(it);
77                                      delete pTmp;
78                                      j--;
79                                      num++;
80                                      if (stk.size() == 0 && k==now->size())
81                                      {
82                                              break;
83                                      }
84                              }
85                              else
86                              {
87                                      k=m+1;
88                                      it = stk.end() - 1;
89                                      pTmp = *it;
90                                      stk.erase(it);
91                                      delete pTmp;
92                                      num++;
93                              }
94
95                      }
96              }
97              else
98              {
99                      it = stk.end() - 1;
100                     pTmp = *it;
101                     now = (*it)->ps;
102                     k = (*it)->i+1;
103                     stk.erase(it);
104                     delete pTmp;
```

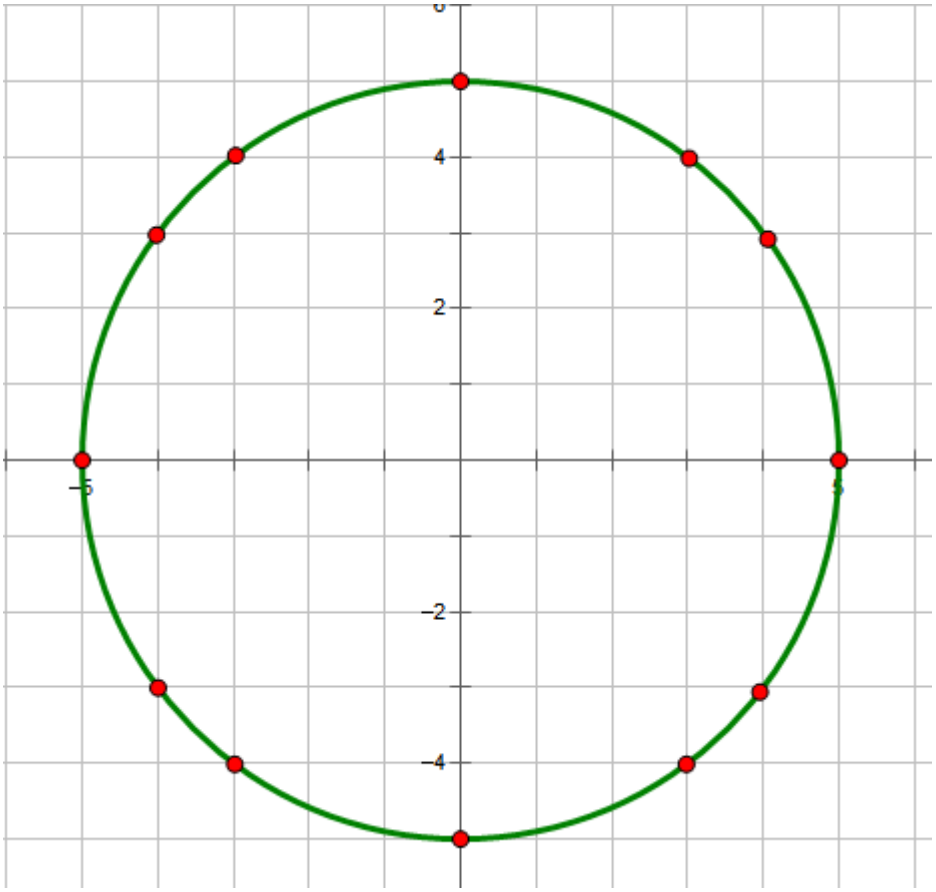
```
105                         j--;
106                         if (stk.size() == 0 && k == now->size())
107                         {
108                                 break;
109                         }
110                 }
111         }//while
112         return num;
113 }
114
115 int main()
116 {
117         vector<string> data;
118
119         int n;
120         string tmp;
121         while (cin >> n)
122         {
123                 for (int i = 0; i < n; i++)
124                 {
125                         string tmp;
126                         cin >> tmp;
127                         data.push_back(tmp);
128                 }
129
130                 cout << findNum(data) << endl;
131                 data.clear();
132         }
133
134         return 0;
135 }
136 //以下是递归的解法
137 // letv1Recursion.cpp : 定义控制台应用程序的入口点。
138 //
139
140 //#include "stdafx.h"
141 #include <iostream>
142 #include <vector>
143 #include <string>
144 using namespace std;
145
146 int getNum(vector<string> data,  int i, string &s)
147 {
148         int num = 0;
```

```
149         int j;
150         for (j = 0; j < data[i].size(); j++)
151         {
152                 char a = data[i][j];
153                 if (s.find(a)== string::npos)
154                 {
155                         s.push_back(a);
156                         if (i == data.size() - 1)
157                         {
158                                 num++;
159                                 s.pop_back();
160                         }
161                         else
162                         {
163                                 num+=getNum(data, i + 1, s);
164                                 s.pop_back();
165                         }
166                 }
167         }
168         //s.pop_back();
169         return num;
170 }
171
172 int main()
173 {
174         vector<string> data;
175
176         int n;
177         while (cin>>n)
178         {
179                 int i = 0;
180                 string s =  "";
181                 int num = 0;
182                 for (int j = 0; j < n; j++)
183                 {
184                         string tmp;
185                         cin >> tmp;
186                         data.push_back(tmp);
187                 }
188                 cout<<getNum(data, i, s)<<endl;
189                 data.clear();
190         }
191         return 0;
192 }
```

在最近几场魔兽争霸赛中，赫柏对自己的表现都不满意。
为了尽快提升战力，赫柏来到了雷鸣交易行并找到了幻兽师格丽，打算让格丽为自己的七阶幻兽升星。
经过漫长的等待以后，幻兽顺利升到了满星，赫柏很满意，打算给格丽一些小费。
赫柏给小费是有原则的：
1.最终给格丽的钱必须是 5 的倍数；
2.小费必须占最终支付费用的 5%~10%之间（包含边界）。
升星总共耗费 A 魔卡，赫柏身上带了 B 魔卡，赫柏想知道他有多少种支付方案可供选择。
注：魔卡是一种货币单位，最终支付费用=本该支付的+小费

```
#include<iostream>
#include<math.h>
using namespace std;
int main(){
        int A,B,R;
        while(cin>>A&&cin>>B){
                R=0;
                int a=ceil(A/0.95),b=floor(A/0.9);
                if(a<=B){
                        if(b>B){
                                R=B/5-a/5;
                        }else{
                                R=b/5-a/5;
                        }
                        if(a%5==0)
                                R++;
        }
                cout<<R<<endl;
        }
        return 0;
}
```

赫柏在绝域之门击败鲁卡斯后，从鲁卡斯身上掉落了一本高级技能书，赫柏打开后惊喜地发现这是一个早已失传的上古技能---禁忌雷炎。
该技能每次发动只需扣很少的精神值，而且输出也非常高。
具体魔法描述如下：
把地图抽象为一个二维坐标，技能发动者位于(0,0)位置。以技能发动者为中心，做一个半径为 r 的圆，满足 r^2=S，如果敌人位于这个圆上，且位置为整点坐标，这个敌人将收到该技能的输出伤害。。
例如当 S=25 时，将有 12 个敌人受到该技能的输出伤害，如下图所示：



更厉害的是，禁忌雷炎可以通过改变魔法输入来控制 S 的大小，因此数学好的魔法师可以通过该技能攻击到更多的敌人。
赫柏想将这个技能学会并成为自己的主技能，可数学是他的硬伤，所以他请求你为他写一个程序，帮帮他吧，没准他就把禁忌雷炎与你分享了 :)

只需要按一个方向遍历一遍 利用勾股定理 x*x+y*y = r*r 即可解决四分之一圆周的点 *4

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<cmath>
using namespace std;
int n;
int main()
{
        while(cin>>n)
        {
                int ans = 0;
                for(int i = 0; i*i < n; i ++)
                {
                        int j = n - i*i;
                        int s =  sqrt(j);
                        if(s*s == j)ans++;
```

```
18                      }
19                      cout<<4*ans<<endl;
20              }
21      }
```