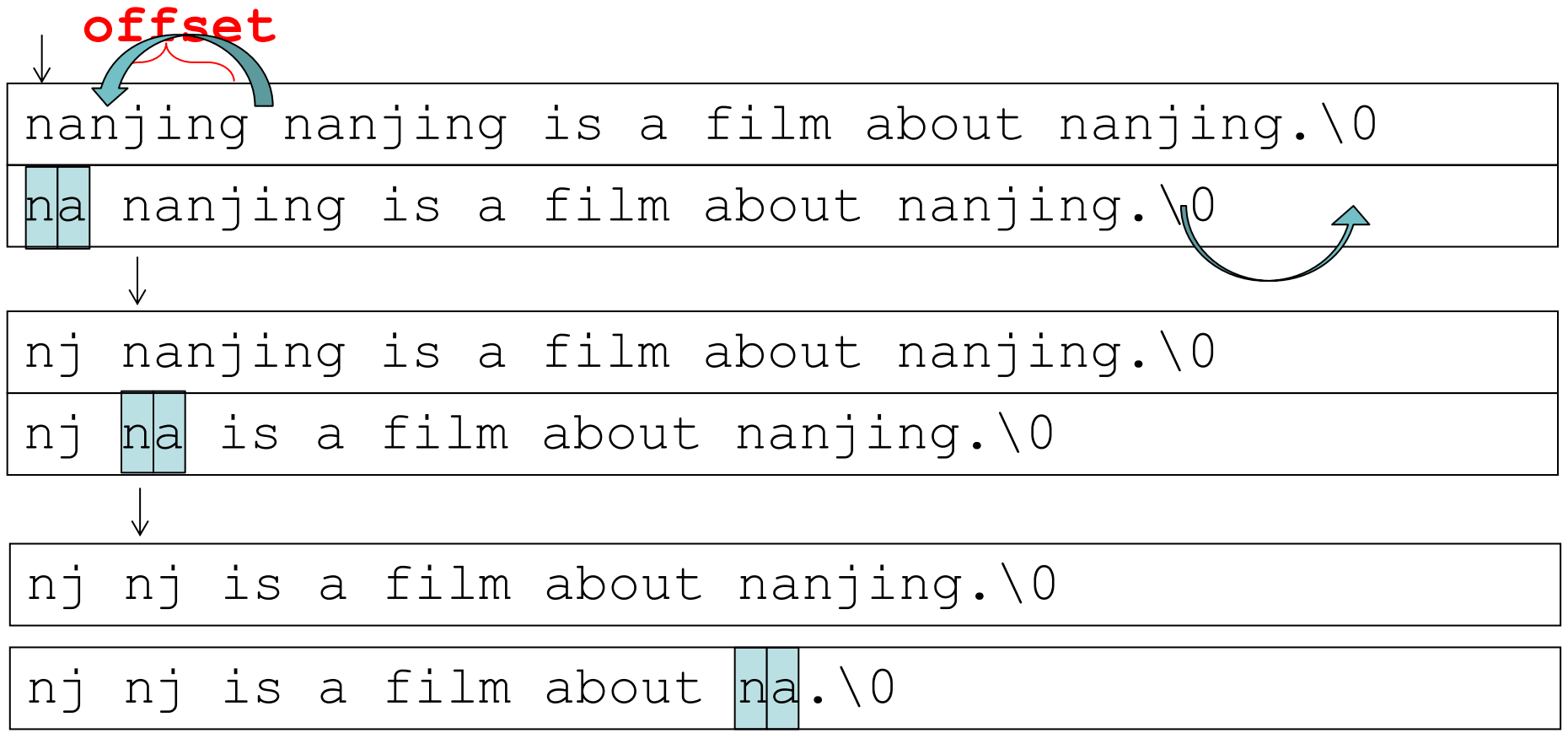
1. 替换子串

请你设计C/C++程序，替换字符串中的子串。其中，用一个函数实现将字符串str中的所有子串find_str都替换成字符串replace_str（其长度与find_str的长度不一定相等），返回值为替换的次数。（仅可以使用输入/输出库函数）

```
str :                nanjing nanjing is a film about nanjing.
find_str :           nanjing
replace_str :        nj
```



```
int find_replace_str(char str[ ], const char find_str[ ],
                                  const char replace_str[ ])
{
      int count = 0;

      int find_len = strlen(find_str);
      int replace_len = strlen(replace_str);
      int offset = find_len - replace_len;

      int index = 0;                                  //str中的当前处理位置
      while (strlen(str+index) >= find_len)
      { …… }

      return count;
}
```

```
…
while (strlen(str+index) >= find_len)
{
      if (strncmp(str+index, find_str, find_len) == 0)
      {
            ……
            ++count;
      }
      else
            ++index;
}
…
```

```
…
if (strncmp(str+index, find_str, find_len) == 0)
{
      int n = strlen(str+index) - find_len + 1;
            //剩余部分的字符个数+1（'\0'）

      if (offset < 0) //替换字符串 比较长
      {     //需把后部分字符后移 -offset 个位置
            for (int i=strlen(str); n>0;  --i, --n)
                  str[i+(-offset)] = str[i];
      }
      else if (offset > 0) //被替换字符串 比较长
      {     //需把后部分字符前移offset个位置
            for (int i=index+find_len; n>0;  ++i, --n)
                  str[i-offset] = str[i];
      }
      ……
```

```
…
if (strncmp(str+index, find_str, find_len) == 0)
{
      ……
      for (int i=0; i < replace_len; ++i) //复制替换串到str
            str[index+i] = replace_str[i];
      index += replace_len;

      ++count;
}
…
```

```
int strCmp(const char *src1, const char *src2)
{
      while(*src1 == *src2)
      {
            if (*src1 == '\0')
                  return 0;
            ++src1, ++src2;
      }
      return *src1 - *src2;

}
```

？

- **考虑不周**
  - find_str一定与replace_str一样长吗?
  - find_str一定比replace_str长吗?
- **考虑到了，但不知道怎么办**
  - 站在机器的角度（一步一步来）

?

```
…
while (strlen(str+index) >= find_len)
{
      if (strncmp(str+index, find_str, find_len) == 0)
      {
            ……
            ++count;
      }
      else
            ++index;

}
…
```

```
…
for (int index=0; strlen(str+index) >= find_len; ++index)
{
      if (strncmp(str+index, find_str, find_len) == 0)
      {
            ……
            ++count;
      }
}
…
```

问题：替换子串之后紧接着的子串没有替换
原因：对循环的执行过程理解不透彻
（第一次循环之后 index对应第二个子串的第二个字符，而不是第一个字符）
对策：**调试！**

?

```
int a = 0;
int b = 0;
int c = 0;
while(str[a] != '\0')
{   while(find[b] != '\0')
    {   if(find[b] == str[a])
        {   ++a;
            ++b;
            if(find[b] == '\0')
            {   b = 0;
                for(c=0; re[c] != '\0'; ++c)
                {  str[a+c]=re[c];
                   ++a;
                }
                break;
            }
        }
    }
```

每次只处理（搬迁）一个字符

即使调对，也是在强化 非结构化思维

对策：看例子规范代码，做每道题时有意识地 训练思维能力

```
else
{     b = 0;
      ++a;
      ++b;
      break;
}
```

循环流程的执行过程没完全理解：a在哪里？

2. 滚雪球

某日，在NJU的某个小山坡上，有一个m×n大小的网格，每个网格的单元格有三种类型：（1）空网格，用‘u’表示；（2）一个雪球，用‘o’表示；（3）一个障碍物，用‘#’表示。 所有的雪球会从上往下滚，直到碰到坡底（网格最下面一行），或碰到一个障碍物，或碰到其他不能再往下滚的雪球。请你设计C/C++程序，输入一个网格的初始状态，当雪球都不能再滚动时，输出网格最终会变成的样子。

题解

- 每一列可以分别处理
  - 我们从下往上处理，如果不考虑#的存在，对于每个雪球，将当前雪球可以滚到的最低位置记为lastEmpty（可能就是当前雪球位置，其初始值为最底部），这样只要将雪球移动到该位置，然后lastEmpty往高处移动一个位置，循环往上处理即可。
  - 如果考虑#的存在，则就是从下往上处理碰到#时，将lastEmpty赋值为比#高一格的地方。

输入：

```
7 11
.o.o....o..
.o........o
...#..o.#..
.o.o.#..o..
.....o...o.
...o.o...#.
.#......#.o
```

输出：

```
...........
...o....o..
...#....#..
.o...#.....
.o.......o.
.o.o.o..o#o
.#.o.oo.#.o
```

这是以前王豫老师的解法，供参考

题解

```cpp
int main() {
  int n, m;
  cin >> n >> m;
  char a[52][52];
  for (int i = 0; i < n; i++)
    cin >> a[i];
  for (int j = 0; j < m; j++) { // 针对每一列
    int lastEmpty = n - 1;
    for (int i = n - 1; i >= 0; i--) { // 从后往前处理
      if (a[i][j] == 'o') {
        a[i][j] = '.';
        a[lastEmpty][j] = 'o';
        lastEmpty--;
      } else if (a[i][j] == '#') {
        lastEmpty = i - 1;
      }
    }
  }
  for (int i = 0; i < n; i++) {
    cout << a[i] << '\n';
  }
  return 0;
}
```

这是以前王豫老师的解法，供参考

3. 分数之和

请你设计C/C++程序求 n 个分数的和。注意结果需要化简，如果结果为0，输出0 1。要求用结构体存储分数。 数据范围：每个分数≥0，1 ≤n≤10，0≤分子＜1e4，1≤分母≤1e4，分母的公倍数≤1e6。

```
int ans_numerator = numerators[0], ans_denominator = denominators[0];
int ans_gcd = gcd(ans_numerator, ans_denominator);
ans_numerator /= ans_gcd;
ans_denominator /= ans_gcd;
```

化简第一个分数

```
for (int i = 1; i < n; i++) {
    ans_numerator = ans_numerator * denominators[i] +
                    ans_denominator * numerators[i];
    ans_denominator *= denominators[i];
    int ans_gcd = gcd(ans_numerator, ans_denominator);
    ans_numerator /= ans_gcd;
    ans_denominator /= ans_gcd;
}
```

计算当前分数和第i个分数的相加并化简

这是以前王豫老师的解法，供参考