小型代码分析系统的实现方式

题目

一个程序中有 26 个对象，每个对象有 26 个成员指针变量。同时还有 26 个普通的指针变量。给定 n 条赋值语句，询问在以任意顺序执行每条语句无限多次的过程中，每个指针变量可能指向的对象集合。

指针分析（Pointer analysis）是静态程序分析的基本组成部分之一，它的目的是找出在程序执行过程中通过特定指针变量访问哪些对象。现在我们希望您对测试数据执行上下文无关指针分析。

一个程序包含26个用小写字母表示的对象，每个对象也有26个用小写字母表示的成员变量（又称字段，可能指向某些对象的指针）。同时，程序中有26个用大写字母指定的全局指针。

程序中有四种语句。我们使用[Variable]表示指针的名称，[Field]表示成员变量的名称，[Object]表示对象。

A = x分配：指针A可以指向对象x，即可以通过A访问x。

A = B转让：指针A可以指向通过B访问的每个对象

A.f = B贮存：对于通过A访问的每个对象o，o的成员变量f可以指向通过B访问的每个对象

A = B.f装载：对于通过B访问的每个对象o，A可以指向通过o的成员变量f访问的每个对象

上下文无关指针分析假设程序的语句将以任何顺序执行足够的次数。例如，在下面两个程序中，A和B都可以指向对象x和对象o。原因是在现实世界中，语句的确切执行顺序和执行时间很难预测。

```
A = o
A = x
B = A

B = A
A = x
A = o
```

现在，您需要对由N个语句组成的给定程序执行上下文无关指针分析，对于每个指针，输出它可以指向的对象。

输入的第一行包含一个整数N（1≤N≤200），表示程序中的语句数。等号“ = ”前后只有一个空格。

以下N行中的每一行都包含一个语句。

输出应该包含26行。

在第i行中，输出第i个指针的名称（第i个大写字母），后跟冒号“ : ”和空格，然后按字母顺序列出可通过该指针访问的对象。

样例

样例一：

```
5
B.f = A
C = B.f
C = x
A = o
B = o

A: o
B: o
C: ox
D:
E:
F:
G:
H:
I:
J:
K:
L:
M:
N:
O:
P:
Q:
R:
S:
T:
U:
V:
W:
X:
Y:
Z:
```

样例二：

```
4
A = o
B.f = A
C = B.f
C = g

A: o
B:
C: g
D:
E:
F:
G:
H:
```

```
I:
J:
K:
L:
M:
N:
O:
P:
Q:
R:
S:
T:
U:
V:
W:
X:
Y:
Z:
```

样例三：

```
3
A = o
B = A
A = x

A: ox
B: ox
C:
D:
E:
F:
G:
H:
I:
J:
K:
L:
M:
N:
O:
P:
Q:
R:
S:
T:
U:
V:
W:
X:
Y:
```

```
Z:
```

题解

直接暴力求解即可。本题难点可能只在于处理读入。

这里给出一个简单的暴力框架：令 pt(x) 为指针 x 可能指向的对象集合。

```
Let worklist be a set
For every allocation statement A = x:
    insert x into pt(A)
    If pt(A) has been changed, add A into worklist
While worklist is not empty:
    While worklist is not empty:
        select one element X from worklist
        delete X from worklist
        For every assignment statement like Y = X:
            merge pt(X) into pt(Y)
            If pt(Y) has been changed, add Y into worklist
    For every store statement Y.f = X:
        For every object o in pt(Y):
            merge pt(X) into pt(o.f)
    For every load statement Y = X.f:
        For every object o in pt(X):
            merge pt(o.f) into pt(Y)
            If pt(Y) has been changed, add Y into worklist
```

样例2是说B没法指向任何对象，是空指针。

这题的重点是，要分清对象和指针：

所有的 a, b, c, d, e, f... z 都是对象A, B, C, ..., Z 和 A.a, A.b, ..., A.z 和 o.a, o.b, o.c, ...o.z 这些是指针。

代码

标程的程序用了大量的assert来控制函数是否执行下去。

```
#include<stdio.h>
#include<assert.h>
#include<string.h>

#include<vector>
#include<set>

using namespace std;

vector<pair<int,int> > storeEdge, loadEdge, allocEdge;
```

```
vector<int> assignEdge[2 * 27];

int pt[(27 + 27) * 27];

int ObjectIndex(char ch)
{
    return ch - 'a';
}

int PointerIndex(char ch)
{
    assert('A' <= ch && ch <= 'Z');
    return ch - 'A' + 26;
}

int FieldIndex(char ch)
{
    assert('a' <= ch && ch <= 'z');
    return ch - 'a' + 1;
}

int makeField(int v, int f)
{
    return f * 52 + v;
}

int makeField(char v, char f)
{
    assert(('A' <= v && v <= 'Z') || ('a' <= v && v <= 'z'));
    assert('a' <= f && f <= 'z');
    if ('a' <= v && v <= 'z') return makeField(ObjectIndex(v),
FieldIndex(f));
    return makeField(PointerIndex(v), FieldIndex(f));
}

int makeField(int v, char f)
{
    assert(0 <= v && v <= 51);
    assert('a' <= f && f <= 'z');
    return makeField(v, FieldIndex(f));
}

int getBase(int v)
{
    int base = v % 52, field = v / 52;
    assert(0 <= base && base <= 52);
    assert(1 <= field && field <= 26);
    return base;
}

int getField(int v)
```

```
{
    int base = v % 52, field = v / 52;
    assert(0 <= base && base <= 52);
    assert(1 <= field && field <= 26);
    return field;
}

int checkFieldNode(const char *s)
{
    int n = strlen(s);
    assert(n == 1 || n == 3);
    if (n == 1)
    {
        return -1;
    }
    assert(s[1] == '.');
    return makeField(s[0], s[2]);
}

void propagate()
{
    set<int> worklist;
    vector<pair<int,int> >::iterator e;
    for(e=allocEdge.begin();e!=allocEdge.end();e++)
    {
        pt[e->first] |= (1 << e->second);
        worklist.insert(e->first);
    }
    while (!worklist.empty())
    {
        while (!worklist.empty())
        {
            int u = *worklist.begin();
            worklist.erase(worklist.begin());
            vector<int>::iterator v;
            for(v=assignEdge[u].begin();v!=assignEdge[u].end();v++)
            {
                if ((pt[*v] & pt[u]) != pt[u])
                {
                    pt[*v] |= pt[u];
                    worklist.insert(*v);
                }
            }
        }
        for(e=storeEdge.begin();e!=storeEdge.end();e++)
        {
            int p = e->second;
            int q = getBase(e->first);
            int f = getField(e->first);
            for (int i = 0; i < 26; i++)
                if ((pt[q] >> i) & 1)
```

```
                {
                    pt[makeField(i, f)] |= pt[p];
                }
        }
        for(e=loadEdge.begin();e!=loadEdge.end();e++)
        {
            int p = getBase(e->second);
            int f = getField(e->second);
            int q = e->first;
            for (int i = 0; i < 26; i++)
                if ((pt[p] >> i) & 1)
                {
                    int value = pt[makeField(i, f)];
                    if ((pt[q] & value) != value)
                    {
                        pt[q] |= value;
                        worklist.insert(q);
                    }
                }
        }
    }
    for(e=allocEdge.begin();e!=allocEdge.end();e++)
    {
        assert(pt[e->second] | (e->first));
    }
    for (int i = 'A'; i <= 'Z'; i++)
    {
        int src = PointerIndex(i);
        vector<int>::iterator dst;
        for(dst=assignEdge[src].begin();dst!=assignEdge[src].end();dst++)
        {
            assert((pt[*dst] & pt[src]) == pt[src]);
        }
    }
    for(e=storeEdge.begin();e!=storeEdge.end();e++)
    {
        int dst = e->first;
        int src = e->second;
        int base = getBase(dst);
        int f = getField(dst);
        for (int i = 0; i < 26; i++)
        {
            if ((pt[base] >> i) & 1)
            {
                assert((pt[makeField(i, f)] & pt[src]) == pt[src]);
            }
        }
    }
    for(e=loadEdge.begin();e!=loadEdge.end();e++)
    {
        int dst = e->first;
```

```
        int src = e->second;
        int base = getBase(src);
        int f = getField(src);
        for (int i = 0; i < 26; i++)
        {
            if ((pt[base] >> i) & 1)
            {
                assert((pt[makeField(i, f)] & pt[dst]) == pt[makeField(i,
f)]);
            }
        }
    }
}

void output(int x)
{
    for (int i = 0; i < 26; i++)
    {
        if ((x >> i) & 1)
        {
            printf("%c",(char)('a' + i));
        }
    }
    printf("\n");
}

int main()
{
    char receiver[10], sender[10];
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        scanf("%s = %s", receiver, sender);
        int R = checkFieldNode(receiver);
        int G = checkFieldNode(sender);
        if (R != -1 && G != -1)
        {
            assert(0);
        }
        if (R != -1 && G == -1)
        {
            storeEdge.push_back(pair<int,int>(R, PointerIndex(sender[0])));
        }
        if (R == -1 && G != -1)
        {
            loadEdge.push_back(pair<int,int>(PointerIndex(receiver[0]),
G));
        }
        if (R == -1 && G == -1)
        {
```

```
            if (sender[0] >= 'a' && sender[0] <= 'z')
allocEdge.push_back(pair<int,int>(PointerIndex(receiver[0]),
ObjectIndex(sender[0])));
            else
            {
assignEdge[PointerIndex(sender[0])].push_back(PointerIndex(receiver[0]));
            }
        }
    }
    propagate();
    for (char i = 'A'; i <= 'Z'; i++)
    {
        printf("%c: ",i);
        for (int j = 0; j < 26; j++)
            if ((pt[PointerIndex(i)] >> j) & 1)
            {
                printf("%c",(char)(j + 'a'));
            }
        printf("\n");
    }
    return 0;
}
```