```cpp
#include <bits/stdc++.h>

using namespace std;

char _getc()
{
    char ch = getchar();
    if (ch == '#')
        while (ch != '\n' && ch != EOF)
            ch = getchar();
    return ch;
}

struct arr
{
    int *val, start;
    arr(int s, int t) : start(s) { val = new int[t - s + 5]; }
    void aset(int i, int v) { val[i - start] = v; }
    int aget(int i) { return val[i - start]; }
};

map<std::string, int> inttable;
map<std::string, arr *> arrtable;

struct Initer
{
    int type;
    std::string name;
    int begin, end;
    Initer *nxt;
};

struct Expression
{
    int type;
    int symbol;
    Expression *arre;
    std::string val;
    Expression *nxt;

    int eval()
    {
        int num = 0;

        if (type == 0)
            for (int i = 0; i < val.size(); ++i)
                num = num * 10 + val[i] - '0';
        else if (type == 1)
            num = inttable[val];
        else if (type == 2)
            num = arrtable[val]->aget(arre->eval());
        else
            throw("RE");

        num *= symbol;

        if (nxt != NULL)
            return num + nxt->eval();
        return num;
    }
};

struct Instruction
{
    int type;
    Initer *init;
    Expression *exp1, *exp2, *exp3;
    int judgetype;
    Instruction *subins;
    Instruction *nxt;
};

void Run(Instruction *ins);

void _vars(Instruction *ins)
{
    for (Initer *i = ins->init; i != NULL; i = i->nxt)
    {
        if (i->type == 0)
            inttable[i->name] = 0;
        else if (i->type == 1)
            arrtable[i->name] = new arr(i->begin, i->end);
        else
            throw("RE: Something wrong with the type of a Initer.");
    }
}

void _set(Instruction *ins)
{
    Expression *exp1 = ins->exp1, *exp3 = ins->exp3;

    if (exp1->type == 1)
        inttable[exp1->val] = exp3->eval();
    else if (exp1->type == 2)
        arrtable[exp1->val]->aset(exp1->arre->eval(), exp3->eval());
    else
        throw("CE: exp1 is not a variable");
}

void _yosoro(Instruction *ins)
{
    printf("%d ", ins->exp1->eval());
}

bool _gif(Instruction *ins)
{
    Expression *exp1 = ins->exp1, *exp2 = ins->exp2;

    switch (ins->judgetype)
    {
    case 0: // lt
        if (exp1->eval() >= exp2->eval())
            return false;
        break;
    case 1: // gt
        if (exp1->eval() <= exp2->eval())
            return false;
        break;
    case 2: // le
        if (exp1->eval() > exp2->eval())
            return false;
        break;
    case 3: // ge
        if (exp1->eval() < exp2->eval())
            return false;
        break;
    case 4: // eq
        if (exp1->eval() != exp2->eval())
            return false;
        break;
    case 5: // neq
        if (exp1->eval() == exp2->eval())
            return false;
        break;
    default:
        throw("RE: Something wrong with the type of a judge");
    }

    Run(ins->subins);
    return true;
}

void _gor(Instruction *ins)
{
    Expression *exp1 = ins->exp1, *exp2 = ins->exp2, *exp3 = ins->exp3;

    _set(ins);
    while (_gif(ins))
    {
        if (exp1->type == 1)
            ++inttable[exp1->val];
        else if (exp1->type == 2)
        {
            int i = exp1->arre->eval();
            arrtable[exp1->val]->aset(i, arrtable[exp1->val]->aget(i) + 
1);
        }
        else
            throw("CE: exp1 is not a variable");
    }
}

void _ghile(Instruction *ins)
{
    while (_gif(ins))
        ;
}

void Run(Instruction *ins)
{
    while (ins != NULL)
    {
        switch (ins->type)
        {
        case 0:
            _vars(ins);
            break;
        case 1:
            _set(ins);
            break;
        case 2:
            _yosoro(ins);
            break;
        case 3:
            _gif(ins);
            break;
        case 4:
            _gor(ins);
            break;
        case 5:
            _ghile(ins);
            break;
        default:
            throw("RE: Something wrong with the type of a instruction.");
        }
        ins = ins->nxt;
    }
}

Instruction *Main;
char c;

void ReadInstruction(char endc, Instruction *ins);

void readiniter(Initer *init)
{
    while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
        c = _getc();
    if (!((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')))
        throw("CE: Unexpected character. ");

    std::string name;
```

```
        while ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
            name.push_back(c), c = _getc();

        init->name = name;
        while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
            c = _getc();
        if (c != ':')
            throw("CE");

        c = _getc();
        while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
            c = _getc();

        name.clear();
        if (!((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')))
            throw("CE: Unexpected character. TT");
        while ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
            name.push_back(c), c = _getc();

        if (name == "int")
            init->type = 0;
        else if (name == "array")
        {
            init->type = 1;
            if (c != '[')
                throw("CE: Unexpected character. ");

            name.clear();
            c = _getc();
            while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
                c = _getc();

            while ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
                name.push_back(c), c = _getc();

            if (name != "int")
                throw("CE: Unknow type. ");
            while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
                c = _getc();
            if (c != ',')
                throw("CE: Unexpected character. ");

            c = _getc();
            while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
                c = _getc();

            int num = 0;
            while (c >= '0' && c <= '9')
                num = num * 10 + c - '0', c = _getc();

            init->begin = num;
            if (c == '.')
            {
                c = _getc();
                if (c != '.')
                    throw("CE: Unexpected character. ");
            }
            else
                throw("CE");

            c = _getc();
            num = 0;
            while (c >= '0' && c <= '9')
                num = num * 10 + c - '0', c = _getc();

            init->end = num;
            c = _getc();
        }
        else
            throw("CE: Unknow type. ");
}

void readexpression(char endc, Expression *&expr)
{
    expr = new Expression();
    while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
        c = _getc();

    if (c == '-')
    {
        expr->symbol = -1;
        c = _getc();
        while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
            c = _getc();
    }
    else if (c == '+')
    {
        expr->symbol = 1;
        c = _getc();
        while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
            c = _getc();
    }
    else
        expr->symbol = 1;

    if (c >= '0' && c <= '9')
    {
        expr->type = 0;
        std::string num;
        while (c >= '0' && c <= '9')
            num.push_back(c), c = _getc();
        expr->val = num;
    }
    else if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
    {
        std::string name;
```

```cpp
            while ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
                name.push_back(c), c = _getc();

            expr->val = name;
            if (c != '[')
                expr->type = 1;
            else
            {
                expr->type = 2;
                c = _getc();
                readexpression(']', expr->arre);
            }
        }

        while (c != endc && (c == ' ' || c == '\n' || c == '\t' || c == '\r'))
            c = _getc();

        if (c == '+' || c == '-')
            readexpression(endc, expr->nxt);
        else if (c == endc)
            c = _getc();
        else
            throw("CE: Unexpected character. ");
    }

    int readjudge()
    {
        while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
            c = _getc();

        if (c == 'l')
        {
            c = _getc();
            if (c == 't')
            {
                c = _getc();
                return 0; // lt
            }
            else if (c == 'e')
            {
                c = _getc();
                return 2; // le
            }
            else
                throw("CE: Unexpected character. ");
        }
        else if (c == 'g')
        {
            c = _getc();
            if (c == 't')
            {
                c = _getc();
                return 1; // gt
            }
            else if (c == 'e')
            {
                c = _getc();
                return 3; // ge
            }
            else
                throw("CE: Unexpected character. ");
        }
        else if (c == 'e')
        {
            c = _getc();
            if (c == 'q')
            {
                c = _getc();
                return 4; // eq
            }
            else
                throw("CE: Unexpected character. ");
        }
        else if (c == 'n')
        {
            c = _getc();
            if (c == 'e')
            {
                c = _getc();
                if (c == 'q')
                {
                    c = _getc();
                    return 5; // neq
                }
                else
                    throw("CE: Unexpected character. ");
            }
            else
                throw("CE: Unexpected character. ");
        }
        else
            throw("CE: Unexpected character. ");

        c = _getc();
    }

    void readvars(Instruction *ins)
    {
        ins->init = new Initer();

        Initer *init = ins->init;
        while (c != '}')
        {
            readiniter(init);
            init->nxt = new Initer();
```

```cpp
            init = init->nxt;
            while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
                c = _getc();
        }

        ins->type = 0;
        c = _getc();
    }

    void readset(Instruction *ins)
    {
        ins->type = 1;
        readexpression(',', ins->exp1);
        readexpression('\n', ins->exp3);
    }

    void readgyxsay(Instruction *ins)
    {
        ins->type = 2;
        readexpression('\n', ins->exp1);
    }

    void readgif(Instruction *ins)
    {
        ins->type = 3;
        ins->judgetype = readjudge();
        c = _getc();
        readexpression(',', ins->exp1);
        readexpression('\n', ins->exp2);
        ins->subins = new Instruction();
        ReadInstruction('}', ins->subins);
    }

    void readgor(Instruction *ins)
    {
        ins->type = 4;
        ins->judgetype = 2;
        readexpression(',', ins->exp1);
        readexpression(',', ins->exp3);
        readexpression('\n', ins->exp2);
        ins->subins = new Instruction();
        ReadInstruction('}', ins->subins);
    }

    void readwhile(Instruction *ins)
    {
        readgif(ins);
        ins->type = 5;
    }

    void ReadInstruction(char endc, Instruction *ins)
    {
        while (c != endc)
    {
        while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
            c = _getc();

        if (c != '{' && c != ':')
            throw("CE: Unexpected character. ");
        else
        {
            std::string opt;

            c = _getc();
            while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
                c = _getc();
            while (c >= 'a' && c <= 'z')
                opt.push_back(c), c = _getc();

            if (c != ' ' && c != '\t' && c != '\n' && c != '\r')
                throw("CE: Unexpected character. ");

            if (opt == "vars")
                readvars(ins);
            else if (opt == "set")
                readset(ins);
            else if (opt == "gyxsay")
                readgyxsay(ins);
            else if (opt == "gif")
                readgif(ins);
            else if (opt == "gor")
                readgor(ins);
            else if (opt == "ghile")
                readwhile(ins);
            else
                throw("CE: Unknow Instruciton. ");

            ins->nxt = new Instruction();
            ins = ins->nxt;

            while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
                c = _getc();
        }
    }

    c = _getc();
}

int main()
{
    c = _getc();
    Main = new Instruction();

    try
    {
        ReadInstruction(EOF, Main);
```

```
            Run(Main);
            printf("\n");
        }
        catch (const char *e)
        {
            cerr << e << endl;
        }

        return 0;
    }

    // g++ main.cpp -o main.exe -std=c++17 -Oz -lm -static

```