```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");
```

```
54
55          num *= symbol;
56
57          if (nxt != NULL)
58              return num + nxt->eval();
59          return num;
60      }
61  };
62
63  struct Instruction
64  {
65      int type;
66      Initer *init;
67      Expression *exp1, *exp2, *exp3;
68      int judgetype;
69      Instruction *subins;
70      Instruction *nxt;
71  };
72
73  void Run(Instruction *ins);
74
75  void _vars(Instruction *ins)
76  {
77      for (Initer *i = ins->init; i != NULL; i = i->nxt)
78      {
79          if (i->type == 0)
80              inttable[i->name] = 0;
81          else if (i->type == 1)
82              arrtable[i->name] = new arr(i->begin, i->end);
83          else
84              throw("RE: Something wrong with the type of a Initer.");
85      }
86  }
87
88  void _set(Instruction *ins)
89  {
90      Expression *exp1 = ins->exp1, *exp3 = ins->exp3;
91
92      if (exp1->type == 1)
93          inttable[exp1->val] = exp3->eval();
94      else if (exp1->type == 2)
95          arrtable[exp1->val]->aset(exp1->arre->eval(), exp3->eval());
96      else
97          throw("CE: exp1 is not a variable");
98  }
99
100 void _yosoro(Instruction *ins)
101 {
102     printf("%d ", ins->exp1->eval());
103 }
104
105 bool _gif(Instruction *ins)
106 {
```

```
107         Expression *exp1 = ins->exp1, *exp2 = ins->exp2;
108 
109         switch (ins->judgetype)
110         {
111         case 0: // lt
112             if (exp1->eval() >= exp2->eval())
113                 return false;
114             break;
115         case 1: // gt
116             if (exp1->eval() <= exp2->eval())
117                 return false;
118             break;
119         case 2: // le
120             if (exp1->eval() > exp2->eval())
121                 return false;
122             break;
123         case 3: // ge
124             if (exp1->eval() < exp2->eval())
125                 return false;
126             break;
127         case 4: // eq
128             if (exp1->eval() != exp2->eval())
129                 return false;
130             break;
131         case 5: // neq
132             if (exp1->eval() == exp2->eval())
133                 return false;
134             break;
135         default:
136             throw("RE: Something wrong with the type of a judge");
137         }
138 
139         Run(ins->subins);
140         return true;
141     }
142 
143     void _gor(Instruction *ins)
144     {
145         Expression *exp1 = ins->exp1, *exp2 = ins->exp2, *exp3 = ins->exp3;
146 
147         _set(ins);
148         while (_gif(ins))
149         {
150             if (exp1->type == 1)
151                 ++inttable[exp1->val];
152             else if (exp1->type == 2)
153             {
154                 int i = exp1->arre->eval();
155                 arrtable[exp1->val]->aset(i, arrtable[exp1->val]->aget(i) + ↵
     1);
156             }
157             else
158                 throw("CE: exp1 is not a variable");
```

```
159         }
160     }
161
162     void _ghile(Instruction *ins)
163     {
164         while (_gif(ins))
165             ;
166     }
167
168     void Run(Instruction *ins)
169     {
170         while (ins != NULL)
171         {
172             switch (ins->type)
173             {
174             case 0:
175                 _vars(ins);
176                 break;
177             case 1:
178                 _set(ins);
179                 break;
180             case 2:
181                 _yosoro(ins);
182                 break;
183             case 3:
184                 _gif(ins);
185                 break;
186             case 4:
187                 _gor(ins);
188                 break;
189             case 5:
190                 _ghile(ins);
191                 break;
192             default:
193                 throw("RE: Something wrong with the type of a instruction.");
194             }
195             ins = ins->nxt;
196         }
197     }
198
199     Instruction *Main;
200     char c;
201
202     void ReadInstruction(char endc, Instruction *ins);
203
204     void readiniter(Initer *init)
205     {
206         while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
207             c = _getc();
208         if (!((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')))
209             throw("CE: Unexpected character. ");
210
211         std::string name;
```

```
212         while ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
213             name.push_back(c), c = _getc();
214 
215         init->name = name;
216         while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
217             c = _getc();
218         if (c != ':')
219             throw("CE");
220 
221         c = _getc();
222         while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
223             c = _getc();
224 
225         name.clear();
226         if (!((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')))
227             throw("CE: Unexpected character. TT");
228         while ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
229             name.push_back(c), c = _getc();
230 
231         if (name == "int")
232             init->type = 0;
233         else if (name == "array")
234         {
235             init->type = 1;
236             if (c != '[')
237                 throw("CE: Unexpected character. ");
238 
239             name.clear();
240             c = _getc();
241             while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
242                 c = _getc();
243 
244             while ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
245                 name.push_back(c), c = _getc();
246 
247             if (name != "int")
248                 throw("CE: Unknow type. ");
249             while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
250                 c = _getc();
251             if (c != ',')
252                 throw("CE: Unexpected character. ");
253 
254             c = _getc();
255             while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
256                 c = _getc();
257 
258             int num = 0;
259             while (c >= '0' && c <= '9')
260                 num = num * 10 + c - '0', c = _getc();
261 
262             init->begin = num;
263             if (c == '.')
264             {
```

```
265             c = _getc();
266             if (c != '.')
267                 throw("CE: Unexpected character. ");
268         }
269         else
270             throw("CE");
271 
272         c = _getc();
273         num = 0;
274         while (c >= '0' && c <= '9')
275             num = num * 10 + c - '0', c = _getc();
276 
277         init->end = num;
278         c = _getc();
279     }
280     else
281         throw("CE: Unknow type. ");
282 }
283 
284 void readexpression(char endc, Expression *&expr)
285 {
286     expr = new Expression();
287     while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
288         c = _getc();
289 
290     if (c == '-')
291     {
292         expr->symbol = -1;
293         c = _getc();
294         while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
295             c = _getc();
296     }
297     else if (c == '+')
298     {
299         expr->symbol = 1;
300         c = _getc();
301         while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
302             c = _getc();
303     }
304     else
305         expr->symbol = 1;
306 
307     if (c >= '0' && c <= '9')
308     {
309         expr->type = 0;
310         std::string num;
311         while (c >= '0' && c <= '9')
312             num.push_back(c), c = _getc();
313         expr->val = num;
314     }
315     else if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
316     {
317         std::string name;
```

```
318             while ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
319                 name.push_back(c), c = _getc();
320 
321             expr->val = name;
322             if (c != '[')
323                 expr->type = 1;
324             else
325             {
326                 expr->type = 2;
327                 c = _getc();
328                 readexpression(']', expr->arre);
329             }
330         }
331 
332         while (c != endc && (c == ' ' || c == '\n' || c == '\t' || c == '\r ↵
    '))
333             c = _getc();
334 
335         if (c == '+' || c == '-')
336             readexpression(endc, expr->nxt);
337         else if (c == endc)
338             c = _getc();
339         else
340             throw("CE: Unexpected character. ");
341     }
342 
343     int readjudge()
344     {
345         while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
346             c = _getc();
347 
348         if (c == 'l')
349         {
350             c = _getc();
351             if (c == 't')
352             {
353                 c = _getc();
354                 return 0; // lt
355             }
356             else if (c == 'e')
357             {
358                 c = _getc();
359                 return 2; // le
360             }
361             else
362                 throw("CE: Unexpected character. ");
363         }
364         else if (c == 'g')
365         {
366             c = _getc();
367             if (c == 't')
368             {
369                 c = _getc();
```

```
370                 return 1; // gt
371             }
372             else if (c == 'e')
373             {
374                 c = _getc();
375                 return 3; // ge
376             }
377             else
378                 throw("CE: Unexpected character. ");
379         }
380         else if (c == 'e')
381         {
382             c = _getc();
383             if (c == 'q')
384             {
385                 c = _getc();
386                 return 4; // eq
387             }
388             else
389                 throw("CE: Unexpected character. ");
390         }
391         else if (c == 'n')
392         {
393             c = _getc();
394             if (c == 'e')
395             {
396                 c = _getc();
397                 if (c == 'q')
398                 {
399                     c = _getc();
400                     return 5; // neq
401                 }
402                 else
403                     throw("CE: Unexpected character. ");
404             }
405             else
406                 throw("CE: Unexpected character. ");
407         }
408         else
409             throw("CE: Unexpected character. ");
410 
411         c = _getc();
412     }
413 
414     void readvars(Instruction *ins)
415     {
416         ins->init = new Initer();
417 
418         Initer *init = ins->init;
419         while (c != '}')
420         {
421             readiniter(init);
422             init->nxt = new Initer();
```

```
423             init = init->nxt;
424             while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
425                 c = _getc();
426         }
427
428         ins->type = 0;
429         c = _getc();
430     }
431
432     void readset(Instruction *ins)
433     {
434         ins->type = 1;
435         readexpression(',', ins->exp1);
436         readexpression('\n', ins->exp3);
437     }
438
439     void readgyxsay(Instruction *ins)
440     {
441         ins->type = 2;
442         readexpression('\n', ins->exp1);
443     }
444
445     void readgif(Instruction *ins)
446     {
447         ins->type = 3;
448         ins->judgetype = readjudge();
449         c = _getc();
450         readexpression(',', ins->exp1);
451         readexpression('\n', ins->exp2);
452         ins->subins = new Instruction();
453         ReadInstruction('}', ins->subins);
454     }
455
456     void readgor(Instruction *ins)
457     {
458         ins->type = 4;
459         ins->judgetype = 2;
460         readexpression(',', ins->exp1);
461         readexpression(',', ins->exp3);
462         readexpression('\n', ins->exp2);
463         ins->subins = new Instruction();
464         ReadInstruction('}', ins->subins);
465     }
466
467     void readwhile(Instruction *ins)
468     {
469         readgif(ins);
470         ins->type = 5;
471     }
472
473     void ReadInstruction(char endc, Instruction *ins)
474     {
475         while (c != endc)
```

```
476         {
477             while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
478                 c = _getc();
479 
480             if (c != '{' && c != ':')
481                 throw("CE: Unexpected character. ");
482             else
483             {
484                 std::string opt;
485 
486                 c = _getc();
487                 while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
488                     c = _getc();
489                 while (c >= 'a' && c <= 'z')
490                     opt.push_back(c), c = _getc();
491 
492                 if (c != ' ' && c != '\t' && c != '\n' && c != '\r')
493                     throw("CE: Unexpected character. ");
494 
495                 if (opt == "vars")
496                     readvars(ins);
497                 else if (opt == "set")
498                     readset(ins);
499                 else if (opt == "gyxsay")
500                     readgyxsay(ins);
501                 else if (opt == "gif")
502                     readgif(ins);
503                 else if (opt == "gor")
504                     readgor(ins);
505                 else if (opt == "ghile")
506                     readwhile(ins);
507                 else
508                     throw("CE: Unknow Instruciton. ");
509 
510                 ins->nxt = new Instruction();
511                 ins = ins->nxt;
512 
513                 while (c == ' ' || c == '\n' || c == '\t' || c == '\r')
514                     c = _getc();
515             }
516         }
517 
518         c = _getc();
519     }
520 
521     int main()
522     {
523         c = _getc();
524         Main = new Instruction();
525 
526         try
527         {
528             ReadInstruction(EOF, Main);
```

```
529             Run(Main);
530             printf("\n");
531         }
532         catch (const char *e)
533         {
534             cerr << e << endl;
535         }
536 
537         return 0;
538     }
539 
540     // g++ main.cpp -o main.exe -std=c++17 -Oz -lm -static
541 
```