Lyft 60 min round

1. Key-Value Database:
   Input is a file
   Command line
2. **KeyVersionStore**
3. **Autocomplete**
4. **Image LRU**
5. **Excel**
6. **map digit to letter**

Lyft Onsite

In-Memory Key-Value Database

You are to build a data structure for storing integers. You will not persist the database to disk, you will store the data in memory.

For simplicity's sake, instead of dealing with multiple clients and communicating over the network, your program wil

Your database should accept the following commands.

SET <name> <value>
Set the variable name to the value value. Neither variable names nor values will contain spaces.

GET <name>
Print out the value of the variable name, or `NULL` if that variable is not set.

UNSET <name>
Unset the variable name, making it just like that variable was never set.

NUMWITHVALUE <value>
Print out the number of variables that are currently set to value. If no variables equal that value, print 0.

END
Exit the program. Your program will always receive this as its last command.

PART 2:
Once your database accepts the above commands and is tested and works, implement commands below

Open a new transaction block. Transaction blocks can be nested (`BEGIN` can be issued inside of an existing

block) but you should get non-nested transactions working first before starting on nested. **A `GET` within a transaction returns the latest value by any command**. **Any data command that is run outside of a transaction block should commit immediately.**

- ROLLBACK

Undo all of the commands issued in the most recent transaction block, and close the block. Print nothing if successful, or print `NO TRANSACTION` if no transaction is in progress.

- COMMIT

Close all open transaction blocks, permanently applying the changes made in them. Print nothing if successful, or print `NO TRANSACTION` if no transaction is in progress.

Your output should contain the output of the GET and NUMWITHVALUE commands. GET will print out the value of the specified key, or NULL. NUMWITHVALUE will return the number of keys which have the specified value.

申请的是2019软件工程实习 找地里的大哥内推的 我分到的hr特别好 基本秒回邮件
一面是lc tag题 小哥就问了一道秒了聊了会儿天30分. more info on 1point3acres
终面是30min behavioral 90 min programming 60min 正常lc 也是问的tag题 但让我写了三个function细微改条件
签了nda我就不说lc是哪道题了 怕被认出来 好好刷面经就行 但是我把programming的原题贡献出来了
我个人认为我写的代码readability和correctness都是很好的 但是。。因为时间不够 某个function在list是空的时候忘记加return以至于
接下来会有idx out of range exception 也没来得及写comment 这里要骂一下我的面试官 30分钟我连题都没读懂问他一些example cases.
他说他也不知道 迟到了5分钟打来不说 还用了10+分钟问我简历 因为他找不到google docs的链接了 我最后快交的时候都急哭了 可定是negative signal
其实这道题地里我见过 但当时好多帖子就说的是什么kv store 而且只有一个帖子是提到了rollback,begin block功能我根本没看懂 还是希望大家贡献面经的时候说清楚点吧.
准备lyft准备了一周都没有去上课 最后就因为这个少加一句return跪了 哪怕再多给我2分钟我就查到了 心情堵的慌.

Command line

```
"""
1. Start program: python myprogram.py
2. Type commands: "BEGIN", "ROLLBACK","COMMIT", "SET A 3", "UNSET A","GET A"
* Output will be printed to console.
* Add return statements for unit testing,not necessary but save code.

"""
class Database(object):
    def __init__(self):
        """
        :param self.store: database
        :param self.transaction_stack: a stack storing transaction blocks
        :param self.has_open_block: True if at least one transaction is open
        """
        self.store = {}
        self.transaction_stack = []
        self.has_open_block = False

```

```
    def operate(self):
        """
        Process user commands
        """
        while True:
            commands = input()
            commands = commands.strip()
            if not commands:
                print("Please enter a valid command.")
            else:
                commands = commands.split()
                if commands[0] == "BEGIN":
                    self.begin_block()
                elif commands[0] == "ROLLBACK":
                    self.rollback()
                elif commands[0] == "COMMIT":
                    self.commit()
                elif commands[0] == "SET" and len(commands) == 3:
                    name, value = commands[1], commands[2]
                    self.set(name, value)

                elif commands[0] == "GET" and len(commands) == 2:
                    name = commands[1]
                    self.get(name)

                elif commands[0] == "UNSET" and len(commands) == 2:
                    name = commands[1]
                    self.unset(name)

                elif commands[0] == "NUMWITHVALUE" and len(commands) == 2:
                    value = commands[1]
                    self.num_with_value(value)

                elif commands[0] == "END":
                    return
                else:
                    return "Please enter a valid command."

    def get(self, name):
        if name not in self.store:
            print(None)
            return None
        else:
            print(self.store[name])
            return self.store[name]

    def set(self, name, value, undo = True):
        """
        If name in db and value != previous value,set to new value and record "set, previous value" for future rollback
        If name not in db, set value and record "unset"
```

```
        """
        if name in self.store:
            prev_val = self.store[name]
            self.store[name] = value
            if prev_val != value and undo:
                self.prepare_rollback("set", (name, prev_val))
        else:
            self.store[name] = value
            self.prepare_rollback("unset", (name))

    def unset(self, name, undo = True):
        """
        If name in db,remove it and record "(set, previous value)" for future rollback
        """
        if name in self.store:
            prev_val = self.store[name]
            if undo:
                self.prepare_rollback("set", (name, prev_val))
            del self.store[name]

    def prepare_rollback(self, operation, info):
        """
        Append undo commands to the latest transaction block on the stack
        :param operation: "set"/"unset"
        :param info: (name,previous value) for "set"; (name) for "unset"
        """
        if self.has_open_block and self.transaction_stack:
            last_block = self.transaction_stack[-1]
            last_block.append((operation, info))


    def num_with_value(self, val):
        """
        Print out the number of variables that are currently set to value
        """
        count = 0
        for k, v in self.store.items():
            if v == val:
                count += 1
        print(count)
        return count

    def rollback(self):
        """
        Undo all commands issued in the latest transaction block, and close the block
        Print nothing if successful, or print `NO TRANSACTION` if no transaction is in
progress
        """
        if not self.has_open_block:
            print("NO TRANSACTION")
            return "NO TRANSACTION"
```

```
        else:
            last_block = self.transaction_stack.pop()
            for (operation, info) in last_block:
                if operation == "set":
                    # Set undo to False, we don't want to undo rollback
                    # info: (name,previous value)
                    self.set(*info, undo = False)
                elif operation == "unset":
                    # info: name
                    self.unset(info[0], undo = False)

    def begin_block(self):
        self.transaction_stack.append([])
        self.has_open_block = True

    def commit(self):
        """
        Close all open transaction blocks, permanently applying the changes made in them.
        Print nothing if successful, or print `NO TRANSACTION` if no transaction is in
progress.
        """
        if not self.transaction_stack:
            print("NO TRANSACTION")
            return "NO TRANSACTION"
        else:
            self.transaction_stack = []
            self.has_open_block = False



import unittest

class Test(unittest.TestCase):
    def test_set(self):
        db = Database()
        db.set("A",10)
        self.assertEqual(db.get("A"), 10)
        db.set("A",20)
        self.assertEqual(db.get("A"), 20)

    def test_unset(self):
        db = Database()
        db.set("A",10)
        db.unset("A")
        self.assertEqual(db.get("A"), None)

    def test_num_with_value(self):
        db = Database()
        db.set("A",10)
        db.set("B",10)
```

```
        self.assertEqual(db.num_with_value(10), 2)
        db.set("B",5)
        self.assertEqual(db.num_with_value(10), 1)


    def test_rollback(self):
        db = Database()
        db.set("A",10)
        db.begin_block()
        db.set("A",20)
        db.begin_block()
        db.set("A",30)
        self.assertEqual(db.get("A"), 30)
        db.rollback()
        self.assertEqual(db.get("A"), 20)
        db.rollback()
        self.assertEqual(db.get("A"), 10)

    def test_commit(self):
        db = Database()
        db.set("A",10)
        db.begin_block()
        db.set("A",20)
        db.commit()
        self.assertEqual(db.get("A"), 20)

    def test_commit_rollback(self):
        db = Database()
        db.set("A",10)
        db.begin_block()
        db.set("A",20)
        db.begin_block()
        db.set("A",30)
        db.commit()
        self.assertEqual(db.get("A"), 30)
        self.assertEqual(db.rollback(), "NO TRANSACTION")
        self.assertEqual(db.commit(), "NO TRANSACTION")

    def test_rollback_commit(self):
        db = Database()
        db.set("A",10)
        db.begin_block()
        db.set("A",20)
        db.begin_block()
        db.set("A",30)
        db.rollback()
        self.assertEqual(db.get("A"), 20)
        db.rollback()
        db.commit()
        self.assertEqual(db.get("A"), 10)
        self.assertEqual(db.commit(), "NO TRANSACTION")
```

```
    
    
if __name__ == '__main__':
    db = Database()
    unittest.main()
```

Input is a file

```
class Database(object):
    def __init__(self):
        """
        :param self.store: database
        :param self.transaction_stack: a stack storing transaction blocks
        :param self.has_open_block: True if at least one transaction is open
        """
        self.store = {}
        self.transaction_stack = []
        self.has_open_block = False

    def operate(self, commands):
        """
        Process user commands
        """
        commands = commands.strip()
        if not commands:
            return "Please enter a valid command."

        else:
            commands = commands.split()
            if commands[0] == "BEGIN":
                return self.begin_block()
            elif commands[0] == "ROLLBACK":
                return self.rollback()

            elif commands[0] == "COMMIT":
                return self.commit()
            elif commands[0] == "SET" and len(commands) == 3:
                name, value = commands[1], commands[2]
                self.set(name, value)

            elif commands[0] == "GET" and len(commands) == 2:
                name = commands[1]
                return self.get(name)

            elif commands[0] == "UNSET" and len(commands) == 2:
                name = commands[1]
                self.unset(name)

            elif commands[0] == "NUMWITHVALUE" and len(commands) == 2:
                value = commands[1]
```

```
            return self.num_with_value(value)

        elif commands[0] == "END":
            return
        else:
            return "Please enter a valid command."

    def get(self, name):
        if name not in self.store:
            return "NULL"
        else:
            return self.store[name]

    def set(self, name, value, undo = True):
        """
        If name in db and value != previous value,set to new value and record "set, previous value" for future rollback
        If name not in db, set value and record "unset"
        """
        if name in self.store:
            prev_val = self.store[name]
            self.store[name] = value
            if prev_val != value and undo:
                self.prepare_rollback("set", (name, prev_val))
        else:
            self.store[name] = value
            self.prepare_rollback("unset", (name))

    def unset(self, name, undo = True):
        """
        If name in db,remove it and record "(set, previous value)" for future rollback
        """
        if name in self.store:
            prev_val = self.store[name]
            if undo:
                self.prepare_rollback("set", (name, prev_val))
            del self.store[name]

    def prepare_rollback(self, operation, info):
        """
         Append undo commands to the latest transaction block on the stack
        :param operation: "set"/"unset"
        :param info: (name,previous value) for "set"; (name) for "unset"
        """
        if self.has_open_block and self.transaction_stack:
            last_block = self.transaction_stack[-1]
            last_block.append((operation, info))


    def num_with_value(self, val):
        """
```

```
        Print out the number of variables that are currently set to value
        """
        count = 0
        for k, v in self.store.items():
            if v == val:
                count += 1
        return count

    def rollback(self):
        """
        Undo all commands issued in the latest transaction block, and close the block
        Print nothing if successful, or print `NO TRANSACTION` if no transaction is in
progress
        """
        if not self.has_open_block:
            return "NO TRANSACTION"

        else:
            last_block = self.transaction_stack.pop()
            for (operation, info) in last_block:
                if operation == "set":
                    # Set undo to False, we don't want to undo rollback
                    # info: (name,previous value)
                    self.set(*info, undo = False)
                elif operation == "unset":
                    # info: name
                    self.unset(info[0], undo = False)

    def begin_block(self):
        self.transaction_stack.append([])
        self.has_open_block = True

    def commit(self):
        """
        Close all open transaction blocks, permanently applying the changes made in them.
        Print nothing if successful, or print `NO TRANSACTION` if no transaction is in
progress.
        """
        if not self.transaction_stack:
            return "NO TRANSACTION"
        else:
            self.transaction_stack = []
            self.has_open_block = False



import unittest

class Test(unittest.TestCase):
    def test_set(self):
        db = Database()
```

```
        db.set("A",10)
        self.assertEqual(db.get("A"), 10)
        db.set("A",20)
        self.assertEqual(db.get("A"), 20)

    def test_unset(self):
        db = Database()
        db.set("A",10)
        db.unset("A")
        self.assertEqual(db.get("A"), "NULL")

    def test_num_with_value(self):
        db = Database()
        db.set("A",10)
        db.set("B",10)
        self.assertEqual(db.num_with_value(10), 2)
        db.set("B",5)
        self.assertEqual(db.num_with_value(10), 1)


    def test_rollback(self):
        db = Database()
        db.set("A",10)
        db.begin_block()
        db.set("A",20)
        db.begin_block()
        db.set("A",30)
        self.assertEqual(db.get("A"), 30)
        db.rollback()
        self.assertEqual(db.get("A"), 20)
        db.rollback()
        self.assertEqual(db.get("A"), 10)

    def test_commit(self):
        db = Database()
        db.set("A",10)
        db.begin_block()
        db.set("A",20)
        db.commit()
        self.assertEqual(db.get("A"), 20)

    def test_commit_rollback(self):
        db = Database()
        db.set("A",10)
        db.begin_block()
        db.set("A",20)
        db.begin_block()
        db.set("A",30)
        db.commit()
        self.assertEqual(db.get("A"), 30)
        self.assertEqual(db.rollback(), "NO TRANSACTION")
```

```
        self.assertEqual(db.commit(), "NO TRANSACTION")

    def test_rollback_commit(self):
        db = Database()
        db.set("A",10)
        db.begin_block()
        db.set("A",20)
        db.begin_block()
        db.set("A",30)
        db.rollback()
        self.assertEqual(db.get("A"), 20)
        db.rollback()
        db.commit()
        self.assertEqual(db.get("A"), 10)
        self.assertEqual(db.commit(), "NO TRANSACTION")

import sys
if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'],exit=False)
    if len(sys.argv) != 2:
        print("Please inform the file name")
        exit(1)
    f1 = sys.argv[1]
    with open(f1) as f1:
        db = Database()
        with open("output.txt", "w") as f2:
            for line in f1:
                result = db.operate(line)
                if result:
                    print(result, file= f2)
```

KeyVersionStore

Everytime we add a new key or change the value of that key, we increment the version.Return the value of a specific key in a specific version.

1. `PUT <key> <value>`

   Set the key name to the value. Key strings will not contain spaces. Print out the version number, the key and the value as `PUT(#<version number>) <key> = <value>`. The first write in the file should be version number `1`, the second should be version number `2`, etc.

2. `GET <key>`

   Print out the key and the last value of the key, or `<NULL>` if that key has never been set as in: `GET <key> = <value>`

3. `GET <key> <version number>`

   Print out the key, the version number and the value of key as it was at the time of the version number,

or `<NULL>` if that key was not set at that time, as in `GET <key>(#version) = <value>`. If the version number has not yet been recorded, return the most recent value for the key. See below for examples of formatted output.

```
1
```

Image LRU

https://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=195179&highlight=lyft

```
import io
import os
import sys
import unittest
import wget

class LRUCache:
    def __init__(self, capacity):
        """
        :param self.LRUCache: a dictionary for storing url-image pairs in the LRU Cache
        :param self.queue: a double-ended queue with the most recently used item appended to the end,
                           an item in the queue denotes a usage of the item, not the item itself
        :param self.totalSize: the total size of images in LRUCache
        :param self.counter: a frequency counter for the urls in LRU Cache
        :param self.cacheSize: maximum capacity of the LRU Cache
        """
        from collections import deque, Counter
        self.LRUCache = {}
        self.queue = deque()
        self.totalSize = 0
        self.counter = Counter()
        self.cacheSize = capacity

    def getImage(self, url):
        """
        If url in cache, print IN CACHE
        Else call downloadImage(url)
        """
        url = url.strip()
        if not url:
            print("Empty url")
            return
        if url in self.LRUCache:
            # Append the most recently used item to the end of the self.queue
            self.queue.append(url)
            self.counter[url] += 1
            print(url + " IN CACHE",)
        else:
```

```
            self.downloadImage(url)
        

    def downloadImage(self, url):
        """
        Download the image from the url
        If the cache cannot hold the image, it will use LRU eviction policy to evict the existing images.
        If the image size is greater than the maximum cache size then output an error message
        Else print DOWNLOADED + size of image
        :type url: string
        :type image: image object
        :return: void
        """

        image = self.downloadHelper(url)
        size = self.getImgSize(image)
        if size > self.cacheSize:
            print("Image size exceeds cache size")
            return
        else:
            print(url + " DOWNLOADED " + str(size))

        self.LRUCache[url] = (image, size)
        self.queue.append(url)
        self.totalSize += size


        while self.totalSize > self.cacheSize:
            URL = self.queue.popleft()
            IMG, SIZE = self.LRUCache[URL]
            self.counter[URL] -= 1
            # If an item becomes the least recently used,remove it from self.LRUCache.
            if self.counter[URL] == 0:
                self.totalSize -= SIZE
                del self.LRUCache[URL]
                del self.counter[URL]
                    

    def downloadHelper(self, url):
        import wget
        try:
            image = wget.download(url)
        except Exception as e:
            print("Exception: ",e," at ",url)
        return image
    def getImgSize(self, image):
        return os.stat(image).st_size


```

```
89
90 class Test(unittest.TestCase):
91     def downloadHelper(self, url):
92         import wget
93         try:
94             image = wget.download(url)
95         except Exception as e:
96             print("Exception: ",e," at ",url)
97         return image
98
99     def getImgSize(self, image):
100        return os.stat(image).st_size
101
102    def test(self):
103        actual = io.StringIO()
104        sys.stdout = actual         #  and redirect stdout.
105
106        imageCache = LRUCache(580000)
107        urls = [ "http://i.imgur.com/xGmX4h3.jpg", "http://i.imgur.com/IUfsijF.jpg",
"http://i.imgur.com/xGmX4h3.jpg",
108                 "http://i.imgur.com/IUfsijF.jpg",
"http://i.imgur.com/xGmX4h3.jpg","https://i.imgur.com/HsUw6GN.jpg"]
109
110        imageCache.getImage(urls[0])
111        sys.stdout = sys.__stdout__                     # Reset redirect
112        size = self.getImgSize(self.downloadHelper(urls[0]))
113        self.assertEqual(actual.getvalue().strip(), urls[0] + " DOWNLOADED " + str(size))
114
115        actual = io.StringIO()
116        sys.stdout = actual         #  and redirect stdout.
117
118        imageCache.getImage(urls[1])
119        sys.stdout = sys.__stdout__                     # Reset redirect
120        size = self.getImgSize(self.downloadHelper(urls[1]))
121        self.assertEqual(actual.getvalue().strip(), urls[1] + " DOWNLOADED " + str(size))
122
123        actual = io.StringIO()
124        sys.stdout = actual         #  and redirect stdout.
125
126        imageCache.getImage(urls[1])
127        sys.stdout = sys.__stdout__                     # Reset redirect
128        self.assertEqual(actual.getvalue().strip(), urls[1] + " IN CACHE")
129
130
131
132
133 if __name__ == "__main__":
134     #unittest.main()
135     orig_stdout = sys.stdout
136     outputFile = open("output.txt", 'w')
137     sys.stdout = outputFile
```

```
    with open('url.txt',encoding="utf-8") as f1:
        capacity = next(f1).strip()
        imageCache = LRUCache(int(capacity))
        for line in f1:
            imageCache.getImage(line)
    sys.stdout = orig_stdout
    outputFile.close()
```

Autocomplete words

sentence: https://leetcode.com/problems/design-search-autocomplete-system/

```
class Autocomplete:
    def __init__(self, words, ranks):
        """
        Build the trie with using prefixes of the input words
        :param words: List[str]
        :param ranks: List[int]
        """
        from collections import defaultdict
        self.trie = defaultdict(list)
        self.rank = defaultdict(int)
        for i in range(len(words)):
            self.rank[words[i]] = ranks[i]
            for j in range(1, len(words[i])+1):
                self.trie[words[i][:j]].append(words[i])


    def input(self, word):
        """
        :param word: a test word string
        :rtype: a list of possible words that have the same prefix as the test word
        """
        result = []
        if word not in self.trie:
            result.append("This prefix doesn't exist in the autocomplete system.")
        else:
            # print candidate words in the order of their ranks
            self.trie[word].sort(key = lambda x: (self.rank[x], x))
            for i in range(len(self.trie[word])):
                result.append(self.trie[word][i] + " " + str(self.rank[self.trie[word][i]]))
        return result


if __name__ == '__main__':
    ranks, words = [], []
    with open("input.txt") as f1:
        for line in f1:
```

```
            line = line.strip()
            if line:
                word, rank = line.split(" ")
                ranks.append(int(rank))
                words.append(word)

    autocomplete = Autocomplete(words, ranks)
    with open("test.txt") as test:
        with open("output.txt", "w") as output:
            for line in test:
                word = line.strip()
                if word:
                    result = autocomplete.input(word)
                    print(word + ":", file=output)
                    for res in result:
                        print(res, file=output)
                    print(" ", file=output)
```

```
#版本2
class Autocomplete:
    def __init__(self, words):
        """
        Build the trie with using prefixes of the input words
        :param words: List[str]
        :param ranks: List[int]
        """
        from collections import defaultdict
        self.trie = defaultdict(list)
        for i in range(len(words)):
            for j in range(1, len(words[i])+1):
                self.trie[words[i][:j]].append(words[i])


    def input(self, word):
        """
        :param word: a test word string
        :rtype: a list of possible words that have the same prefix as the test word
        """
        result = []
        if word not in self.trie:
            result.append("This prefix doesn't exist in the autocomplete system.")
        else:
            # print candidate words in the order of their ranks
            for i in range(len(self.trie[word])):
                result.append(self.trie[word][i])
        return result


if __name__ == '__main__':
    words = []
    testWords = []
```

```
    with open("input.txt") as f1:
        numWords = int(next(f1))
        numTestWords = int(next(f1))
        for _ in range(numWords):
            word = next(f1)
            word = word.strip()
            if word:
                words.append(word)
        for _ in range(numTestWords):
            word = next(f1)
            word = word.strip()
            if word:
                testWords.append(word)
    autocomplete = Autocomplete(words)
    with open("output.txt", "w") as output:
        for word in testWords:
            result = autocomplete.input(word)
            print(result, file = output)
```

6. map digit to letter

第二题是用trie 就是给一个file 里面上半部分是单词下半部分是一些数字的组合

AA
WOT
YOU
ME
968
63
12345

要求output是

968: you, wot
63: me
12345: no result
. more info on 1point3acres
就是把数字对应的字母能组成的单词找出来，数字对应的是电话号码key pad上的character

```
from collections import defaultdict
class Trie():
    def __init__(self,val):
        self.val = val
        self.children = defaultdict(Trie)
        self.number = ""

def check(text):
    dic = {"A":2,
"B":2,"C":2,"D":3,"E":3,"F":3,"G":4,"H":4,"I":4,"J":5,"K":5,"L":5,"M":6,"N":6,"O":6,"P"
:7,"Q":7,"R":7,"S":7,
           "T":8,"U":8,"V":8,"W":9,"X":9,"Y":9,"Z":9}
    words = set()
    numbers = set()
```

```
        with open(text) as f:
            content = f.readlines()
        content = [s.strip() for s in content if s]
        print(content)
        for w in content:
            if w.isupper():
                words.add(w)
            elif w.isdigit():
                numbers.add(w)
        root = Trie(0)
        res = defaultdict(list)
        print(words,numbers)
        for word in words:
            tmp = root
            for l in word:
                if l not in tmp.children:
                    node = Trie(l)
                    tmp.children[l] = node
                    node.number = tmp.number+str(dic[l])
                tmp = tmp.children[l]

            if tmp.number in numbers:
                res[tmp.number].append(word)
        return res
```

Column	Value
A1	1
A2	A1 + 4
A3	A1 + A2 + 10
A1	A3 - 5 (throw exception, because of circular reference. Remember: A3 was A1 + A2 + 10)

put('A1', '1')
put('A2', 'A1+4')
put('A3', 'A1+A2+10')
put('A3', 'A1 - 5') (throw exception, because of circular reference. Remember: A3 was A1 + A2 + 10)

Get('A1') => 1
Get('A2') => 5
Get('A3') => 16

第一轮90分钟上机coding：白人小哥，题目是设计excel，先讲思路，其实主要是处理dependency的问题，比如A1=B1+C1, B1=D1+1这种情况如何处理。我的想法跟小白有点不一样，但也是work的，结果这个小白非要和我追根问底，最后我就采用了小白的思路，尼玛这时候已经30分钟过去了。然后小白就走了，留下我一个人在那写code。C++选手这时候就吃亏了，parse传进来的公式什么的就比较麻烦，小白你知道什么库的话可以随便用，但我不知道，只能用string那些硬写，所以代码量还挺大的。到剩15分钟的小白又回来了，跟我讨论，让我解释我的code，到这时候还有一块没写完，更别说测试代码了。最后提一句，我是用的他们的loan laptop写的，上面的gcc

编译器都不支持C++11，非常坑爹。

是A=B+C或A=B-C的格式，支持+-两种就行了，主要是这里B和C可以是常数或者表达式的，A必须是表达式，然后要自己处理string parse