122022010 Shrayank Mistry
Description: Simulate recovery using undo, redo and undo/redo logging.
Programming Language: Python

```python
from tabulate import tabulate

#* GLOBAL VARIABLES
transactions_dict = {}
temp_values = {}
main_disk = {
    'A': 0,
    'B': 0,
    'C': 0,
    'D': 0,
    'E': 0,
    'F': 0,
}
cache_disk = {}
class transaction_data:
    def __init__(self, transaction_id):
        self.transaction_id = transaction_id
        self.data_items = list()

def get_value(ins, a, b):
    if (ins.find('+') > -1): return a + b
    if (ins.find('-') > -1): return a - b
    if (ins.find('*') > -1): return a * b
    if (ins.find('/') > -1): return a / b

def load_data_to_cache(main_disk, cache_disk):
    for key, value in main_disk.items():
        cache_disk[key] = value

def read_transaction_file(filename):
    file = open(filename, "r")
    data = []

    for line in file:
        data.append(line.strip())

    instrus_cnt = int(data[0])
    instructions = data[1:]
    return instrus_cnt, instructions
```

```python
if __name__ == "__main__":
    filename = "transaction-undo-redo.txt"
    instrus_cnt, instructions = read_transaction_file(filename)

    load_data_to_cache(main_disk, cache_disk)

    #* Deferred Updation (0), Immediate Updation (1)
    type_of_updation = 1
    create_log(instrus_cnt, instructions, type_of_updation)

    cnt = 1
    data = []

    data_items = list(cache_disk.keys())
    cache_values = list(cache_disk.values())
    disk_bc = list(main_disk.values())

    log_filename = "log-undo-redo.txt"
    simulate_crash(log_filename)

    disk_ac = list(main_disk.values())

    while cnt <= len(data_items):
        data.append([cnt, data_items[cnt - 1], cache_values[cnt - 1], disk_bc[cnt - 1], disk_ac[cnt - 1]])
        cnt += 1

    str = ''
    if type_of_updation == 0:
        str = "Defered Update"
    if type_of_updation == 1:
        str = "Immediate Update"

    print("Type of Updation Policy = " + str)
    print (tabulate(data, headers=["No.", "Data Item", "Cache", "Disk [Before Crash]", "Disk[After Crash]"]))
```

```python
def create_log(instrus_cnt, instructions, type):
    log_file = open("log-undo-redo.txt", "w")
    for i in range(instrus_cnt):
        ins = instructions[i]

        #* Normal Operations
        if (ins.find('(') != -1):
            operation = ins[0:ins.find('(')]
            if (operation == "BEGIN"):
                transaction = ins[ins.find('(') + 1:ins.find(')')]
                log_file.write(f"<START {transaction}>\n")
                transactions_dict[transaction] = transaction_data(transaction)

            if (operation == 'READ'):
                data_item = ins[ins.find('(') + 1:ins.find(',')]
                transaction = ins[ins.find(' ') + 1:ins.find(')')]
                # log_file.write(f"<{transaction} READ {data_item}>\n")
                temp_values[data_item] = cache_disk[data_item]

            if (operation == 'WRITE'):
                data_item = ins[ins.find('(') + 1:ins.find(',')]
                transaction = ins[ins.find(' ') + 1:ins.find(')')]
                log_file.write(f"<{transaction} {data_item} {cache_disk[data_item]}
{temp_values[data_item]}>\n")
                cache_disk[data_item] = temp_values[data_item]
                if type == 1:
                    main_disk[data_item] = cache_disk[data_item]
                transactions_dict[transaction].data_items.append(data_item)

            if (operation == 'COMMIT'):
                transaction = ins[ins.find('(') + 1:ins.find(')')]
                log_file.write(f"<COMMIT {transaction}>\n")
                data_list = transactions_dict[transaction].data_items

                if type == 0:
                    for d in data_list:
                        main_disk[d] = cache_disk[d]
```

```
#* Arithematic Operations
        else:
            updating_data = ins[0]
            a = ins[ins.find('[') + 1]

            #* Code to Refactor
            b = ''
            index = ins.find(']') - 1
            while (ins[index] != ' '):
                b += ins[index]
                index -= 1
            b = b[::-1]

            try:
                temp_b = int(b)
                b = temp_b
                a = cache_disk[a]
            except:
                #* GET BOTH a and b FROM MAIN DISK
                a = cache_disk[a]
                b = cache_disk[b]
            current_value = cache_disk[updating_data]
            updated_value = get_value(ins, a, b)
            # log_file.write(f"<{ins[ins.find('_') + 1:]} {updating_data} {curre
nt_value} {updated_value}>\n")
            temp_values[updating_data] = updated_value
            continue
    log_file.close()
```

```
def simulate_crash(log_file):
    commited_transactions = dict()
    uncommited_transactions = dict()
    started_transactions = dict()

    commit_set = set()
    uncommit_list = list()
    started_set = set()

    #* Get all started transactions
    file = open(log_file, "r")
    for line in file:
        if line.find('START') == 1:
            started_transactions[line[line.find(' ') + 1:line.find('>')]] = list()
            started_set.add(line[line.find(' ') + 1:line.find('>')])
    file.close()

    #* Get all commited transactions
    file = open(log_file, "r")
    for line in file:
        if line.find('COMMIT') == 1:
            commited_transactions[line[line.find(' ') + 1:line.find('>')]] = list()
            commit_set.add(line[line.find(' ') + 1:line.find('>')])
    file.close()

    uncommit_list = list(started_set - commit_set)

    for t in uncommit_list:
        uncommited_transactions[t] = list()

    #TODO For all Commited Transactions (Redo)
    #TODO For all Uncommited Transactions (Undo)
```

```
file = open(log_file, "r")
    for line in file:
        line_temp = line.replace('>', '').replace('<', '').replace('\n', '').split(' ')
        if len(line_temp) == 4:
            transaction_id = line_temp[0]
            if commited_transactions.get(transaction_id) == None: #* Uncommited Ts
                uncommited_transactions[transaction_id].append(line.replace('\n', ''))
            else:
                commited_transactions[transaction_id].append(line.replace('\n', '')) #*
 Commited Ts

    log_file = open(log_file, "a+")
    #* Perform forward updates
    for tid, operations in commited_transactions.items():
        for op in operations:
            op = op.replace('>', '').replace('<', '').split(' ')
            transaction_id, data_item, old_value, new_value = op[0:]
            main_disk[data_item] = new_value
        log_file.write(f"<END {tid}>\n")

    #* Perform Backward updates
    for tid, operations in uncommited_transactions.items():
        operations = operations[::-1]
        for op in operations:
            op = op.replace('>', '').replace('<', '').split(' ')
            transaction_id, data_item, old_value, new_value = op[0:]
            main_disk[data_item] = old_value
        log_file.write(f"<ABORT {tid}>\n")

    file.close()
```

```
TODO -------------------------- OUTPUT ------------------------------ #
* Type of Updation Policy = Defered Update
*   No.  Data Item      Cache    Disk[Before Crash]    Disk[After Crash]
* -----  -----------  -------  --------------------  -------------------
*      1  A                 60                    60                    60
*      2  B                 50                    50                    50
*      3  C                 50                    50                    50
*      4  D                100                     0                     0
*      5  E                 20                     0                     0
*      6  F                 10                     0                     0

? Type of Updation Policy = Immediate Update
?   No.  Data Item      Cache    Disk[Before Crash]    Disk[After Crash]
? -----  -----------  -------  --------------------  -------------------
?      1  A                 60                    60                    60
?      2  B                 50                    50                    50
?      3  C                 50                    50                    50
?      4  D                100                   100                     0
?      5  E                 20                    20                     0
?      6  F                 10                    10                     0

TODO ---------------------------------------------------------------- #
```

```
29
BEGIN(T1)
READ(A, T1)
READ(B, T1)
A = [A + 50]_T1
BEGIN(T2)
WRITE(A, T1)
B = [B + A]_T1
WRITE(B, T1)
COMMIT(T1)
READ(A, T2)
A = [A + 10]_T2
READ(C, T2)
C = [C + 50]_T2
WRITE(A, T2)
WRITE(C, T2)
BEGIN(T3)
READ(D, T3)
COMMIT(T2)
D = [D + 100]_T3
WRITE(D, T3)
BEGIN(T4)
READ(E, T4)
E = [E + 10]_T4
WRITE(E, T4)
READ(F, T4)
F = [F + E]_T4
WRITE(F, T4)
E = [E + F]_T4
WRITE(E, T4)
```

```
<START T1>
<START T2>
<T1 A 0 50>
<T1 B 0 50>
<COMMIT T1>
<T2 A 50 60>
<T2 C 0 50>
<START T3>
<COMMIT T2>
<T3 D 0 100>
<START T4>
<T4 E 0 10>
<T4 F 0 10>
<T4 E 10 20>
<END T1>
<END T2>
<ABORT T4>
<ABORT T3>
```