```
#include <map>
#include <set>
#include <pthread.h>
#include <stdlib.h>
#include <string.h>
using namespace std;
//
// structure: _data_iterative_access_log, used to encapsulate the use history of a shared data,
//           item: it's writers, readers and previous values. It is through this
//           structure that data dependence violation is tracked for iterative speculation.
//
struct _data_iterative_access_log{
        unsigned int _size;//size of data
        set <int> _readers;
        vector <int> _writers;
        vector <void*> _previous_values; //this will be used to solve anti-dependence
                                         //and output dependence violations (WAR and WAW)
                                         //restoring the data to a value that is safe.

        pthread_mutex_t _data_mutex;


        _data_iterative_access_log(){
                pthread_mutex_init (&_data_mutex, NULL);
        };


        bool _output_violations_are_false(void* data_to_write, int pos){ //this function determines if the output dependence
        //violations are false, that is, if the value to write is the same that is already written by later iterations.
        //Note that in this version this check to avoid unnecessary thread-resetting is only possible for WAW violations, since
        //there is book-keeping of the values that are being written, but not of the values that are being read.

                void* prev_value= (void*) malloc (_size);

                int first_reader=-2;;
                for (int i=0; i<_writers.size(); i++){
                        if (_writers[i]==-1){
                                memcpy((void*)prev_value, _previous_values[i], _size);
                        }
                        else if (_writers[i]>pos){
                                if (memcmp((void*)data_to_write,(void*)_previous_values[i],_size)==0){
                                        if (first_reader==-2){
                                                if (memcmp((void*)_previous_values[i],(void*)prev_value,_size)!=0){
                                                        free (prev_value);
                                                        return false;
                                                }
                                                first_reader=_writers[i];
                                        }
                                        else if (first_reader==_writers[i]){
                                                if (memcmp((void*)_previous_values[i],(void*)prev_value,_size)!=0){
                                                        free (prev_value);
                                                        return false;
                                                }
                                        }
                                        else{
                                                free (prev_value);
                                                return false;
                                        }
                                }
                        }
                }



                if (first_reader==-2){
                        _writers.push_back(pos);
                        memcpy ((void*)prev_value, (void*)data_to_write, _size);
                        _previous_values.push_back((void*)prev_value);
                        return true;
                }
/*TEMPORALMENTE QUITADO PQ NO ESTOY SEGURO DE LO QUE HACE:
                for (int i=0; i<_writers.size(); i++){
                        if (_writers[i]==first_reader){
                                memcpy((void*)_previous_values[i], (void*)data_to_write, _size);
                        }
                }*/
                _writers.push_back(pos);
                _previous_values.push_back((void*)prev_value);

                return true;

        };

        vector <int> _cancel_higher_readers(int pos){
        //if used before a write_data and the return vector is not empty, a true-dependence violation has ocurred (RAW).
        //and the higher readers have to be restarted.
```

```
        vector <int> to_cancel;
        if (_readers.empty())
                return to_cancel;
        set <int>:: iterator it=_readers.begin();
        while ((!_readers.empty()) && it!=_readers.end()){
                if ((*it)>pos){
                        to_cancel.push_back(*it);
                        _readers.erase(*it);
                        it=_readers.begin();
                }
                else {
                        it++;
                }
        }
        if ((!to_cancel.empty())&&(!_writers.empty())){
                for (unsigned int i=0; i<to_cancel.size(); i++){
                        unsigned int j=0;
                        while((!_writers.empty())&& j<_writers.size()){
                                if (to_cancel[i]==_writers[j]){
                                        _writers.erase(_writers.begin()+j); //all the writings of the higher readers on this data
                                        realloc (_previous_values[j], 0);
                                        _previous_values.erase(_previous_values.begin()+j);
                                        j=0;
                                }
                                else
                                        j++;
                        }
                }
        }
        return to_cancel;
};

void* _get_previous_value(int pos){
        void* value;
        int currPos=-2;
        //The logic to find the previous value is to get the previous value from the lesser of the higher writers,
        //if that fails, then it is necessary to find the previous value of the latest of the lower writers.

        for (int i=(static_cast<int>(_writers.size())-1); i>=0; i--){ //this order matters because if a thread has written several
                                                //it's first should be restored

                if (static_cast<int>(_writers[i])>pos){
                        if (currPos==-2){
                                value=_previous_values[i];
                                currPos=_writers[i];
                                //cout<<"Aqui..."<<endl;
                        }
                        else if (static_cast<int>(_writers[i])<=currPos){
                                value=_previous_values[i];
                                currPos=_writers[i];
                                //cout<<"O aca..."<<endl;
                        }
                }

        }
        if (currPos==-2){
                for (int i=(static_cast<int>(_writers.size())-1); i>=0; i--){
                        if (static_cast<int>(_writers[i])<=pos){
                                if (currPos==-2){
                                        value=_previous_values[i];
                                        //cout<<"Valor es: "<<_previous_values[i]<<" "<<*(double*)_previous_values[i]<<" "<<*new_v
                                        currPos=_writers[i];
                                        //cout<<"O acaaa..."<<endl;
                                }
                                else if (static_cast<int>(_writers[i])>currPos){
                                        value=_previous_values[i];
                                        currPos=_writers[i];
                                        //cout<<"O este..."<<endl;
                                }
                        }

                }

        }

        if (currPos!=-2){
//                      cout<<"Retorna: "<<*(double*)value<<" "<<value<<endl;
                        return (void*)value;
        }
        return (void*)value;
};

        vector <int> _cancel_higher_writers(int pos){
        //if used on a write function and the return vector is not empty, an output dependence violation has ocurred (WAW) and the
        //value has to be restored to it's state before pos, so it can be written over.
```

```
        //if used on a read function and the return vector is not empty, an anti-dependence violation has ocurred (WAR) and the
        //value has to be restored to it's state before pos, so it can be read.
                set <int> to_cancel_set;
                vector <int> to_cancel;
                if (_writers.empty())
                        return to_cancel;
                unsigned int i=0;
                while ((!_writers.empty())&& (i<_writers.size())){
                        if (_writers[i]>pos){
                                to_cancel_set.insert(_writers[i]);
                                _writers.erase(_writers.begin()+i);
                                realloc (_previous_values[i], 0);
                                _previous_values.erase(_previous_values.begin()+i);
                                i=0;
                        }
                        else
                                i++;

                }
                to_cancel.insert(to_cancel.end(), to_cancel_set.begin(), to_cancel_set.end());
                if ((!to_cancel.empty())&&(!_readers.empty())){
                        for (unsigned int i=0; i<to_cancel.size(); i++){
                                _readers.erase(to_cancel[i]); //all the readings of the higher writers on this data item have to be unlogg
                        }
                }
                return to_cancel;
        };
};
```