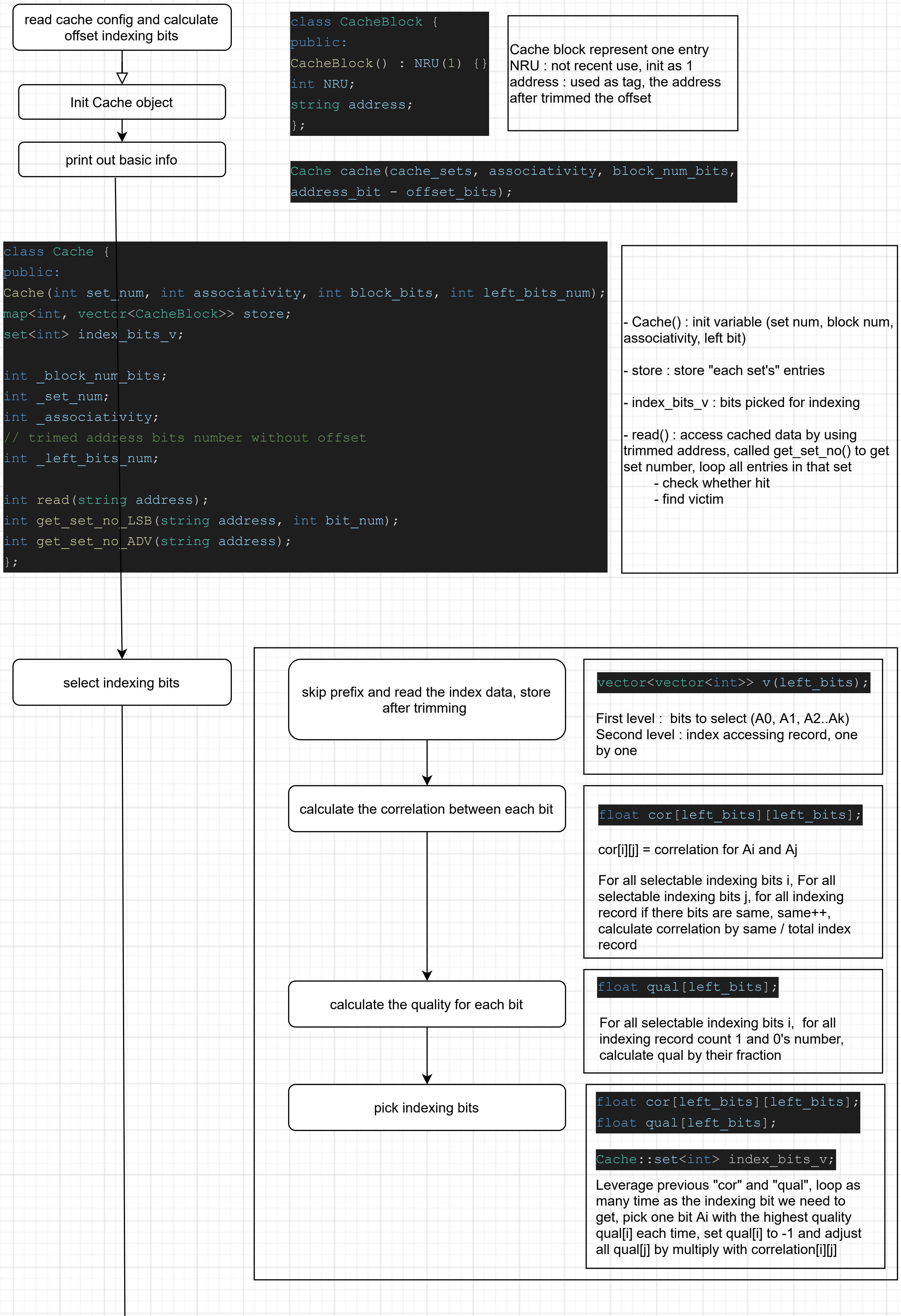
Final Project - The cache behavior simulation

Student Id : 1101158S

Main Flow


read cache config and calculate offset indexing bits
Init Cache object
print out basic info
select indexing bits
skip prefix and read the index data, store after trimming
calculate the correlation between each bit
calculate the quality for each bit
pick indexing bits

```
class CacheBlock {
public:
CacheBlock() : NRU(1) {}
int NRU;
string address;
};
```

Cache block represent one entry
NRU : not recent use, init as 1
address : used as tag, the address after trimmed the offset

```
Cache cache(cache_sets, associativity, block_num_bits,
address_bit - offset_bits);
```

```
class Cache {
public:
Cache(int set_num, int associativity, int block_bits, int left_bits_num);
map<int, vector<CacheBlock>> store;
set<int> index_bits_v;

int _block_num_bits;
int _set_num;
int _associativity;
// trimed address bits number without offset
int _left_bits_num;

int read(string address);
int get_set_no_LSB(string address, int bit_num);
int get_set_no_ADV(string address);
};
```

- Cache() : init variable (set num, block num, associativity, left bit)

- store : store "each set's" entries

- index_bits_v : bits picked for indexing

- read() : access cached data by using trimmed address, called get_set_no() to get set number, loop all entries in that set
    - check whether hit
    - find victim

```
vector<vector<int>> v(left_bits);
```

First level : bits to select (A0, A1, A2..Ak)
Second level : index accessing record, one by one

```
float cor[left_bits][left_bits];
```

cor[i][j] = correlation for Ai and Aj

For all selectable indexing bits i, For all selectable indexing bits j, for all indexing record if there bits are same, same++, calculate correlation by same / total index record

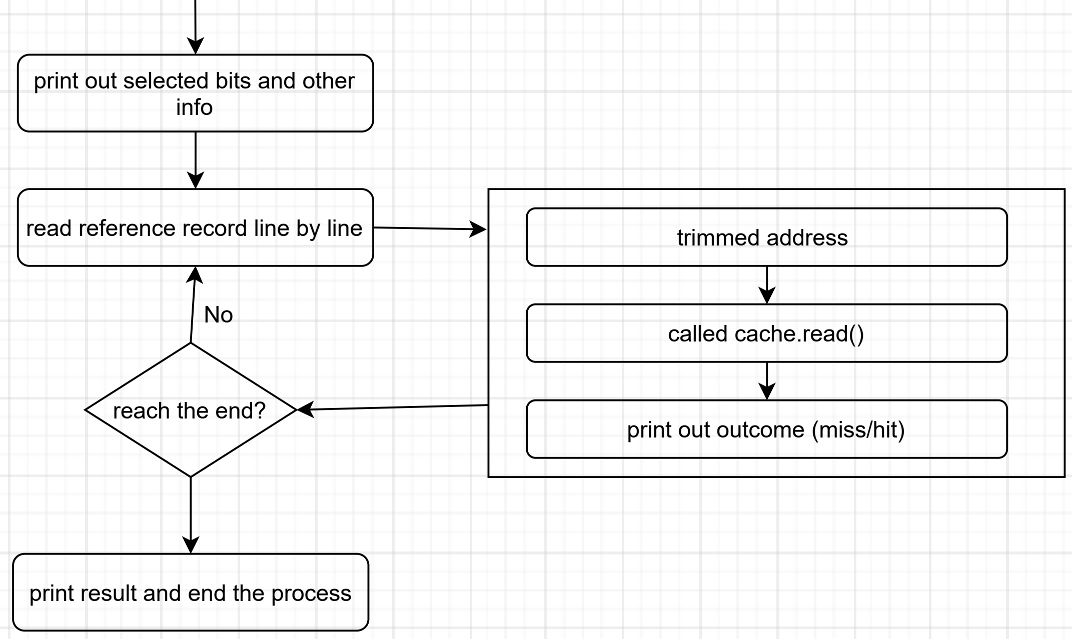
```
float qual[left_bits];
```

For all selectable indexing bits i, for all indexing record count 1 and 0's number, calculate qual by their fraction

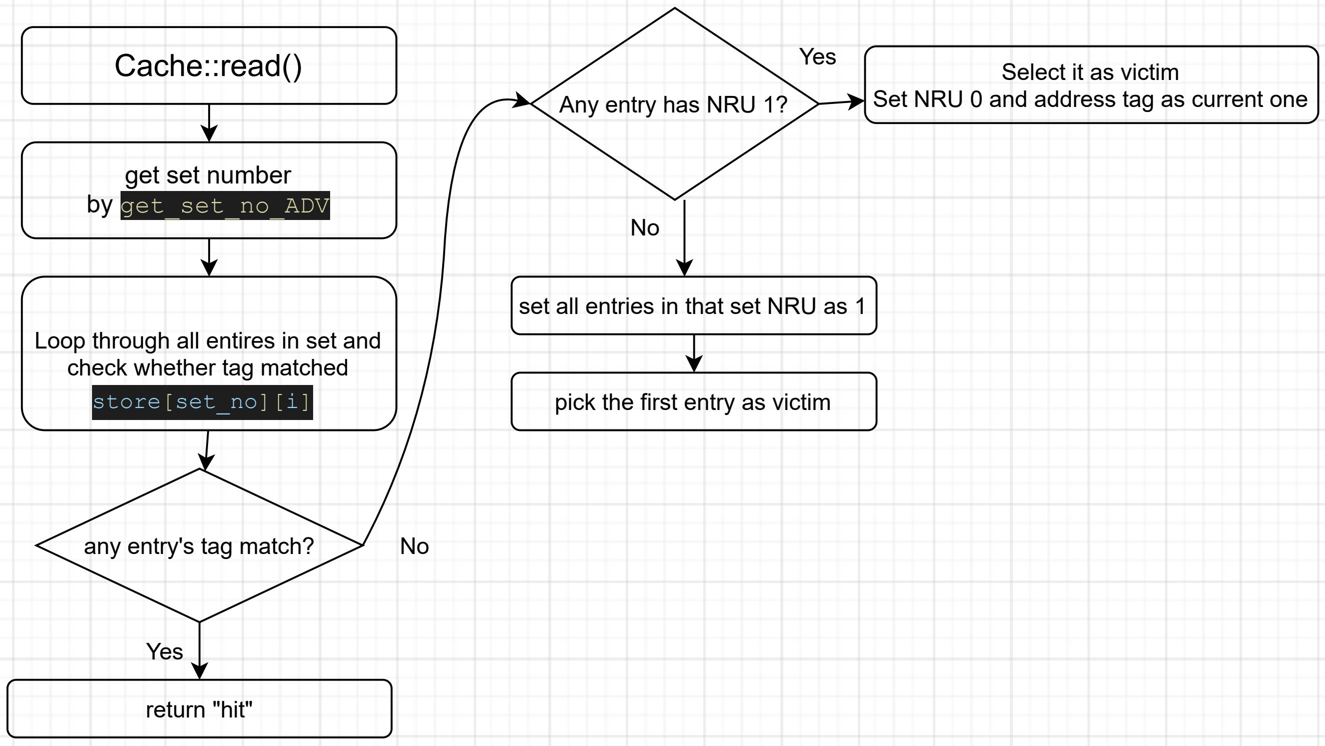
```
float cor[left_bits][left_bits];
float qual[left_bits];
```

```
Cache::set<int> index_bits_v;
```

Leverage previous "cor" and "qual", loop as many time as the indexing bit we need to get, pick one bit Ai with the highest quality qual[i] each time, set qual[i] to -1 and adjust all qual[j] by multiply with correlation[i][j]


print out selected bits and other info
read reference record line by line
trimmed address
called cache.read()
print out outcome (miss/hit)
No
reach the end?
print result and end the process

Detail of Cache Object


Cache::read()
get set number by get_set_no_ADV
Loop through all entires in set and check whether tag matched store[set_no][i]
any entry's tag match?
No
Yes
return "hit"
Any entry has NRU 1?
Yes
Select it as victim
Set NRU 0 and address tag as current one
No
set all entries in that set NRU as 1
pick the first entry as victim