I implement with C++: one virutal base class—*class virtualmem*, which has the generic interface of 6 derived classes---*class optimal, class fifo, class lfu,class lru\_clock, class lru\_stack,class lru\_ref8*.

**class fifo:**

store the pages in list--- “*list<int> frames”*, if the page number in link list reach the maxium frame number, and need to find a victim, remove(page out) the “tail” page from link list, and push new page to the “head” of linklist.

**class lfu:**

Except the list of pages---“ *list<int> frames”*, another data strucutre— “*multiset<int> page* “ is added, which store all pages’ reference. When need to find a victim, find the page which is referened least, and repace the page with the new page.

**class optimal:**

data structure ---“ *list<int> frames”*, is used to hold the frames. When need to find a vitcim, scan all future pages which need to be referenced, and find in “*list<int> frames*”, the most “late” page which will be reference in the future, replace it with the new page.

**class lru\_stack:**

data structure ---“ *list<int> frames”*, is used to hold the frames. When a page is referenced, it is pull to the head of linklist. When need a victim, find the page at the tail of linklist, replace it with the new page.

**class lru\_clock:**

New data structure—“ *list<lru\_clock\_enty> frames*” is used to hold the frame and reference bit. When put a new page to link list, set the reference bit of the page to 0; when one page in the link list is referenced again, set the reference bit to 1; when need a victim, find the page which reference\_bit is 0,one by one from “*current\_ptr*” , if the refence\_bit is 1, continue the lookup.

**class lru\_ref8:**

Data strucure-- “*list<lru\_clock\_enty> frames*”, is used to hold the frame and 8 bit history reference record.When reference a new page, firstly all frames’ 8 bit history reference are right-shift one bit; then the newly reference page’s highest bit of 8 bit history reference will be set to 1.When need a victim, find the least from all 8 bit hitory references of link list.

Appendix 1-README

1.compile the image

>make

2.test

suppose the input text is "a.txt

1>FIFO

>./virtualmem -r FIFO -i a.txt

2>LFU

>./virtualmem -r LFU -i a.txt

3>LRU-STACK

>./virtualmem -r LRU\_STACK -i a.txt

4>LRU-CLOCK

>./virtualmem -r LRU\_CLOCK -i a.txt

5>LRU-REF8

>./virtualmem -r LRU\_REF8 -i a.txt

6>FIFO + 7 available pages

>./virtualmem -r FIFO -i a.txt -f 7

7>input from stdin

>./virtualmem -r FIFO

3.clean all \*.o or binary code

>make clean

Appendix 2—Source Code:

-----main.cpp-----------------------------------------------

#include "fifo.h"

#include "lru\_stack.h"

#include "lru\_clock.h"

#include "lru\_ref8.h"

#include "lfu.h"

#include "optimal.h"

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/time.h>

#include <unistd.h>

#include <iostream>

#include <fstream>

#include <sstream>

void test()

{

fifo fifo(3);

fifo.access\_newpage(1);

fifo.access\_newpage(2);

fifo.access\_newpage(3);

fifo.access\_newpage(1);

cout <<"fifo:";

fifo.show();

fifo.access\_newpage(4);

fifo.show();

cout <<"lru\_stack:";

lru\_stack lru\_stack(3);

lru\_stack.access\_newpage(1);

lru\_stack.access\_newpage(2);

lru\_stack.access\_newpage(3);

lru\_stack.access\_newpage(1);

lru\_stack.show();

lru\_stack.access\_newpage(4);

lru\_stack.show();

cout <<"lru\_clock:";

lru\_clock lru\_clock(4);

lru\_clock.access\_newpage(1);

lru\_clock.access\_newpage(2);

lru\_clock.access\_newpage(3);

lru\_clock.access\_newpage(4);

lru\_clock.access\_newpage(1);

lru\_clock.access\_newpage(2);

lru\_clock.access\_newpage(5);

lru\_clock.access\_newpage(1);

lru\_clock.access\_newpage(2);

lru\_clock.access\_newpage(3);

lru\_clock.access\_newpage(4);

lru\_clock.access\_newpage(5);

cout <<"lru\_ref8:";

lru\_ref8 lru\_ref8(4);

lru\_ref8.access\_newpage(1);

lru\_ref8.access\_newpage(2);

lru\_ref8.access\_newpage(3);

lru\_ref8.access\_newpage(4);

lru\_ref8.access\_newpage(1);

lru\_ref8.access\_newpage(2);

lru\_ref8.access\_newpage(5);

lru\_ref8.access\_newpage(1);

lru\_ref8.access\_newpage(2);

lru\_ref8.access\_newpage(3);

lru\_ref8.access\_newpage(4);

lru\_ref8.access\_newpage(5);

cout <<"lfu:";

lfu lfu(4);

lfu.access\_newpage(1);

lfu.access\_newpage(2);

lfu.access\_newpage(3);

lfu.access\_newpage(4);

lfu.access\_newpage(1);

lfu.access\_newpage(2);

lfu.access\_newpage(5);

lfu.access\_newpage(1);

lfu.access\_newpage(2);

lfu.access\_newpage(3);

lfu.access\_newpage(4);

lfu.access\_newpage(5);

}

static void display\_usage()

{

cout << "virtualmem [−h] [-f available-frames] [−r replacement-policy] [−i input\_file]"<<endl;

return;

}

enum REPL\_POLICY

{

FIFO = 0,

LFU = 1,

LRU\_STACK = 2,

LRU\_CLOCK = 3,

LRU\_REF8 = 4

};

int main(int argc, char \*argv[])

{

//test();

int c;

unsigned int available\_frames=5;

int policy = FIFO;

string filename,policy\_str("FIFO");

ifstream file;

string line;

unsigned int num;

list<unsigned int> pagelist;

int line\_number =0, word\_number;

struct timeval org\_tv,later\_tv;

int actual\_ms,optimal\_ms,actual\_replace=0,optimal\_replace;

if(argc==1)

{

display\_usage();

exit(0);

}

while ((c = getopt (argc, argv, "hf:r:i:")) != -1)

{

switch(c)

{

case 'f':

available\_frames = atoi(optarg);

if(available\_frames<=0)

{

display\_usage();

exit(1);

}

break;

case 'r':

policy\_str = optarg;

if(strcmp(optarg,"FIFO")==0)

{

policy = FIFO;

}

else if(strcmp(optarg,"LFU")==0)

{

policy = LFU;

}

else if(strcmp(optarg,"LRU\_STACK")==0)

{

policy = LRU\_STACK;

}

else if(strcmp(optarg,"LRU\_CLOCK")==0)

{

policy = LRU\_CLOCK;

}

else if(strcmp(optarg,"LRU\_REF8")==0)

{

policy = LRU\_REF8;

}

else

{

display\_usage();

exit(1);

}

break;

case 'i':

filename = optarg;

break;

case 'h':

default:

display\_usage();

exit(0);

}

}

if(!filename.empty())

{

file.exceptions ( ifstream::failbit | ifstream::badbit );

try {

file.open (filename.c\_str());

}

catch (ifstream::failure e) {

cout << "Exception opening/reading file";

exit(0);

}

file.exceptions ( ifstream::goodbit);

//while (!file.eof()) file.get();

while (getline(file, line))

{

istringstream iss(line,istringstream::in);

line\_number++;

word\_number=1;

while ((iss >> num))

{

word\_number++;

//cout<<num <<" ";

pagelist.push\_back(num);

}

if(iss.eof())

{

//cout <<"only end of line";

}

else if(iss.fail())

{

cout <<"line"<< line\_number<<" word" <<word\_number <<" :invalid page number";

pagelist.clear();

exit(1);

}

}

file.close();

}

else

{

getline(cin, line);

istringstream iss(line,istringstream::in);

word\_number =1;

while ((iss >> num))

{

word\_number++;

//cout<<num <<" ";

pagelist.push\_back(num);

}

if(iss.eof())

{

//cout <<"only end of line";

}

else if(iss.fail())

{

cout <<"word" <<word\_number <<" :invalid page number";

pagelist.clear();

exit(0);

}

}

{

fifo fifo(available\_frames);

lru\_stack lru\_stack(available\_frames);

lru\_clock lru\_clock(available\_frames);

lru\_ref8 lru\_ref8(available\_frames);

lfu lfu(available\_frames);

optimal optimal(available\_frames);

gettimeofday(&org\_tv, NULL);

switch(policy)

{

case FIFO:

fifo.access\_pagelist(pagelist);

actual\_replace= fifo.replace\_nbr;

break;

case LFU:

lfu.access\_pagelist(pagelist);

actual\_replace= lfu.replace\_nbr;

break;

case LRU\_STACK:

lru\_stack.access\_pagelist(pagelist);

actual\_replace= lru\_stack.replace\_nbr;

break;

case LRU\_CLOCK:

lru\_clock.access\_pagelist(pagelist);

actual\_replace= lru\_clock.replace\_nbr;

break;

case LRU\_REF8:

lru\_ref8.access\_pagelist(pagelist);

actual\_replace= lru\_ref8.replace\_nbr;

break;

default:

break;

}

gettimeofday(&later\_tv, NULL);

actual\_ms = (later\_tv.tv\_sec - org\_tv.tv\_sec)\*1000000+(later\_tv.tv\_usec-org\_tv.tv\_usec);

gettimeofday(&org\_tv, NULL);

optimal.access\_pagelist(pagelist);

optimal\_replace = optimal.replace\_nbr;

gettimeofday(&later\_tv, NULL);

optimal\_ms = (later\_tv.tv\_sec - org\_tv.tv\_sec)\*1000000+(later\_tv.tv\_usec-org\_tv.tv\_usec);

cout<<endl;

cout<<"# of page replacements with "<< policy\_str<<" :" << actual\_replace<<endl;

cout<<"# of page replacements with "<<"Optimal"<<" :" << optimal\_replace<<endl;

if(0!= optimal\_replace)

{

cout.precision(3);

cout << "% page replacement penalty using " << policy\_str <<" "<<100.0\*(actual\_replace-optimal\_replace)/optimal\_replace <<"%"<<endl;

}

cout<<endl;

cout<<"Total time to run " <<policy\_str<<" algorithm: "<< actual\_ms<<"ms"<<endl;

cout<<"Total time to run " <<"Optimal"<<" algorithm: "<< optimal\_ms<<"ms"<<endl;

if(0!= optimal\_ms)

{

cout.precision(3);

if(actual\_ms < optimal\_ms)

{

cout << policy\_str <<" is "<<100.0\*(optimal\_ms-actual\_ms)/optimal\_ms<<"%"<<" faster than Optimal algorithm."<<endl;

}

else

{

cout << policy\_str <<" is "<<100.0\*(actual\_ms-optimal\_ms)/actual\_ms<<"%"<<" slower than Optimal algorithm." <<endl;

}

}

}

pagelist.clear();

return 0;

}

virtutalmem.h

#ifndef \_\_X\_H\_VIRTUAL\_\_

#define \_\_X\_H\_VIRTUAL\_\_

#include <iostream>

#include <list>

using namespace std;

class virtualmem

{

public:

unsigned int max\_frame\_nbr;

unsigned int replace\_nbr;

public:

virtualmem (int framenumber);

virtual int access\_newpage(int page\_number) = 0;

virtual void access\_pagelist(list<unsigned int> pagelist)=0;

virtual void show()=0;

};

#endif

-------------------------------virtualmem.cpp---------------------------------------

#include "virtualmem.h"

virtualmem::virtualmem(int framenumber)

{

max\_frame\_nbr = framenumber;

replace\_nbr = 0;

}

----------------------------------------------fifo.h-----------------------------------------------------

#ifndef \_\_X\_H\_FIFO\_\_

#define \_\_X\_H\_FIFO\_\_

#include <iostream>

#include <list>

#include "virtualmem.h"

//using namespace std;

class fifo: public virtualmem

{

private:

list<int> frames;

public:

fifo(int framenumber);

void access\_pagelist(list<unsigned int> pagelist);

int access\_newpage(int page\_number);

void show();

~fifo();

};

#endif

---------------------------------fifo.cpp--------------------------

#include "fifo.h"

using namespace std;

fifo::fifo(int framenumber)

:virtualmem(framenumber)

{

}

void fifo::access\_pagelist(list<unsigned int> pagelist)

{

list<unsigned int>::iterator it;

for(it=pagelist.begin(); it!= pagelist.end();++it)

{

access\_newpage(\*it);

}

return;

}

int fifo::access\_newpage(int page\_number)

{

list<int>::iterator it;

if(frames.size()> max\_frame\_nbr)

{

return -1;

}

else if(frames.size()== max\_frame\_nbr)

{

bool found = false;

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(page\_number == \*it)

{

//found

found = true;

break;

}

}

// find a viticm, replace it

if (false == found)

{

// fifo

frames.pop\_back();

frames.push\_front(page\_number);

replace\_nbr++;

}

}

else

{

bool duplicate = false;

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(page\_number == \*it)

{

duplicate = true;

break;

}

}

if(false==duplicate)

{

frames.push\_front(page\_number);

}

}

return 0;

}

void fifo::show()

{

list<int>::iterator it;

cout << "physical frames:";

for (it=frames.begin(); it!=frames.end(); ++it)

{

cout << \*it << " ";

}

cout << endl;

}

fifo::~fifo()

{

frames.clear();

}

----------------------------lfu.h--------------------------------------------------------

#ifndef \_\_X\_H\_LFU\_\_

#define \_\_X\_H\_LFU\_\_

#include <iostream>

#include <list>

#include <set>

#include "virtualmem.h"

class lfu: public virtualmem

{

private:

list<int> frames;

multiset<int> page;

public:

lfu(int framenumber);

void access\_pagelist(list<unsigned int> pagelist);

int access\_newpage(int page\_number);

void show();

~lfu();

};

#endif

------------------------------------lfu.cpp-------------------------------------------

#include "lfu.h"

using namespace std;

lfu::lfu(int framenumber)

:virtualmem(framenumber)

{

}

void lfu::access\_pagelist(list<unsigned int> pagelist)

{

list<unsigned int>::iterator it;

for(it=pagelist.begin(); it!= pagelist.end();++it)

{

access\_newpage(\*it);

}

return;

}

//http://www.yolinux.com/TUTORIALS/CppStlMultiMap.html

int lfu::access\_newpage(int page\_number)

{

list<int>::iterator it;

bool duplicate = false;

if(frames.size()> max\_frame\_nbr)

{

return -1;

}

else

{

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(page\_number == (\*it))

{

duplicate = true;

break;

}

}

}

if(!duplicate)

{

if(frames.size()<max\_frame\_nbr)

{

frames.push\_back(page\_number);

}

else

{

//lookup a victim

list<int>::iterator least=frames.begin();

int counter = page.count(\*least);

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(counter>page.count(\*it))

{

least = it;

}

}

//replace

\*least = page\_number;

replace\_nbr++;

}

}

page.insert(page\_number);

//cout<<"debug info ";

//show();

}

void lfu::show()

{

list<int>::iterator it;

cout << "physical frames:";

for (it=frames.begin(); it!=frames.end(); ++it)

{

cout << \*it << ":counter "<<page.count(\*it)<<" ";

}

cout << endl;

}

lfu::~lfu()

{

frames.clear();

page.clear();

}

--------------------------optimal.h----------------------------------------

#ifndef \_\_X\_H\_OPTIMAL\_\_

#define \_\_X\_H\_OPTIMAL\_\_

#include <iostream>

#include <list>

#include "virtualmem.h"

//using namespace std;

class optimal: public virtualmem

{

private:

list<int> frames;

public:

optimal(int framenumber);

void access\_pagelist(list<unsigned int> pagelist);

int access\_newpage(int page\_number);

void show();

~optimal();

};

#endif

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
--------------------------optimal.cpp------------------------------------------------------------------------

#include "optimal.h"

using namespace std;

optimal::optimal(int framenumber)

:virtualmem(framenumber)

{

}

void optimal::access\_pagelist(list<unsigned int> pagelist)

{

list<unsigned int>::iterator page\_it;

list<int>::iterator it;

for(page\_it = pagelist.begin(); page\_it != pagelist.end();++page\_it)

{

//\*page\_it is new page

bool found = false;

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(\*it == \*page\_it)

{

//found

found = true;

break;

}

}

if(false == found)

{

if(frames.size() < max\_frame\_nbr)

{

frames.push\_back(\*page\_it);

}

else

{

list<unsigned int> locationlist;

list<unsigned int>::iterator location\_it;

unsigned int loc=0,farest\_loc=0;

//find a victim

//scan\_page\_it point to the future pages

list<unsigned int>::iterator scan\_page\_it;

list<int>::iterator replaced\_page;

for (it=frames.begin(); it!=frames.end(); ++it)

{

for(scan\_page\_it=page\_it,loc=0;scan\_page\_it!=pagelist.end(); ++scan\_page\_it,++loc)

{

if((\*scan\_page\_it) ==(\*it))

{

locationlist.push\_back(loc);

break;

}

}

if(pagelist.end()==scan\_page\_it)

{

locationlist.push\_back(0xFFFFFFFF);

break;

}

}

replaced\_page= frames.begin();

farest\_loc= \*locationlist.begin();

for(location\_it = locationlist.begin(),it=frames.begin();location\_it!=locationlist.end();++location\_it,++it)

{

if(\*location\_it> farest\_loc)

{

farest\_loc = \*location\_it;

replaced\_page = it;

}

}

(\*replaced\_page)=(\*page\_it);

replace\_nbr++;

}

}

//show();

}

}

int optimal::access\_newpage(int page\_number)

{

return 0;

}

void optimal::show()

{

list<int>::iterator it;

cout << "physical frames:";

for (it=frames.begin(); it!=frames.end(); ++it)

{

cout << \*it << " ";

}

cout << endl;

}

optimal::~optimal()

{

frames.clear();

}

---------------------------------------lru\_stack.h---------------------------------

#ifndef \_\_X\_H\_LRU\_STACK\_\_

#define \_\_X\_H\_LRU\_STACK\_\_

#include <iostream>

#include <list>

#include "virtualmem.h"

using namespace std;

class lru\_stack:public virtualmem

{

private:

list<int> frames;

public:

lru\_stack(int framenumber);

void access\_pagelist(list<unsigned int> pagelist);

int access\_newpage(int page\_number);

void show();

~lru\_stack();

};

#endif

----------------------------------lru\_stack.cpp---------------------

#include "lru\_stack.h"

using namespace std;

lru\_stack::lru\_stack(int framenumber)

:virtualmem(framenumber)

{

}

void lru\_stack::access\_pagelist(list<unsigned int> pagelist)

{

list<unsigned int>::iterator it;

for(it=pagelist.begin(); it!= pagelist.end();++it)

{

access\_newpage(\*it);

}

return;

}

int lru\_stack::access\_newpage(int page\_number)

{

list<int>::iterator it;

if(frames.size()> max\_frame\_nbr)

{

return -1;

}

else if(frames.size()== max\_frame\_nbr)

{

bool found = false;

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(page\_number == \*it)

{

//found

found = true;

frames.erase(it);

frames.push\_front(page\_number);

break;

}

}

// find a viticm, replace it

if (false == found)

{

// fifo

frames.pop\_back();

frames.push\_front(page\_number);

replace\_nbr++;

}

}

else

{

bool duplicate = false;

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(page\_number == \*it)

{

duplicate = true;

break;

}

}

if(false==duplicate)

{

frames.push\_front(page\_number);

}

}

return 0;

}

void lru\_stack::show()

{

list<int>::iterator it;

cout << "physical frames:";

for (it=frames.begin(); it!=frames.end(); ++it)

{

cout << \*it << " ";

}

cout << endl;

}

lru\_stack::~lru\_stack()

{

frames.clear();

}

----------------------------------lru\_clock.h--------------------------

#ifndef \_\_X\_H\_LRU\_CLOCK\_\_

#define \_\_X\_H\_LRU\_CLOCK\_\_

#include <iostream>

#include <list>

#include "virtualmem.h"

#include "lru\_clock\_entry.h"

//http://www.yolinux.com/TUTORIALS/LinuxTutorialC++STL.html

class lru\_clock:public virtualmem

{

private:

list<lru\_clock\_enty> frames;

list<lru\_clock\_enty>::iterator current\_ptr;

public:

lru\_clock(unsigned int framenumber);

void access\_pagelist(list<unsigned int> pagelist);

int access\_newpage(int page\_number);

void show();

~lru\_clock();

};

#endif

-----------------------------------lru\_clock.cpp--------------------------------------

#include "lru\_clock.h"

using namespace std;

lru\_clock::lru\_clock(unsigned int framenumber)

:virtualmem(framenumber)

{

}

void lru\_clock::access\_pagelist(list<unsigned int> pagelist)

{

list<unsigned int>::iterator it;

for(it=pagelist.begin(); it!= pagelist.end();++it)

{

access\_newpage(\*it);

}

return;

}

//http://www.cs.columbia.edu/~junfeng/os/lectures/l19-vm.pdf

int lru\_clock::access\_newpage(int page\_number)

{

list<lru\_clock\_enty>::iterator it;

lru\_clock\_enty node;

bool replaced = false;

if(frames.size()> max\_frame\_nbr)

{

return -1;

}

else if(frames.size()== max\_frame\_nbr)

{

bool found = false;

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(page\_number == (\*it).frame)

{

//found

found = true;

(\*it).reference = 1;

break;

}

}

//need to find a viticm, replace it

if (false == found)

{

for (it=current\_ptr; it!=frames.end(); ++it)

{

if(0 == (\*it).reference)

{

//replace

(\*it).frame = page\_number;

(\*it).reference =0 ;

replace\_nbr++;

current\_ptr = ++it;

if(frames.end()==current\_ptr)

{

current\_ptr = frames.begin();

}

replaced = true;

break;

}

else

{

(\*it).reference = 0;

}

}

if(false == replaced)

{

// continue to scan from begin

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(0 == (\*current\_ptr).reference)

{

//replace

(\*it).frame = page\_number;

(\*it).reference =0;

replace\_nbr++;

current\_ptr = ++it;

if(frames.end()==current\_ptr)

{

current\_ptr = frames.begin();

}

replaced = true;

break;

}

else

{

(\*it).reference = 0;

}

}

}

}

}

else

{

lru\_clock\_enty node;

bool duplicate = false;

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(page\_number == (\*it).frame)

{

duplicate = true;

(\*it).reference = 1;

break;

}

}

if(false==duplicate)

{

node.frame = page\_number;

node.reference = 0;

frames.push\_front(node);

}

current\_ptr = frames.begin();

}

//show();

return 0;

}

void lru\_clock::show()

{

list<lru\_clock\_enty>::iterator it;

cout << "lru clock current\_ptr:" <<(\*current\_ptr).frame<<endl;

cout << "physical frames:"<<endl;

for (it=frames.begin(); it!=frames.end(); ++it)

{

cout << \*it << " ";

}

cout <<endl;

}

lru\_clock::~lru\_clock()

{

frames.clear();

}

---------------------------------lru\_ref8.h--------------------------------

#ifndef \_\_X\_H\_LRU\_REF8\_\_

#define \_\_X\_H\_LRU\_REF8\_\_

#include <iostream>

#include <list>

#include "virtualmem.h"

#include "lru\_clock\_entry.h"

//http://www.yolinux.com/TUTORIALS/LinuxTutorialC++STL.html

class lru\_ref8:public virtualmem

{

private:

list<lru\_clock\_enty> frames;

public:

lru\_ref8(unsigned int framenumber);

void access\_pagelist(list<unsigned int> pagelist);

int access\_newpage(int page\_number);

void show();

~lru\_ref8();

};

#endif

-----------------------------------------------------------------lru\_ref8.cpp------------------------------

#include "lru\_ref8.h"

using namespace std;

lru\_ref8::lru\_ref8(unsigned int framenumber)

:virtualmem(framenumber)

{

}

void lru\_ref8::access\_pagelist(list<unsigned int> pagelist)

{

list<unsigned int>::iterator it;

for(it=pagelist.begin(); it!= pagelist.end();++it)

{

access\_newpage(\*it);

}

return;

}

int lru\_ref8::access\_newpage(int page\_number)

{

list<lru\_clock\_enty>::iterator it;

if(frames.size()> max\_frame\_nbr)

{

return -1;

}

else

{

lru\_clock\_enty node;

bool duplicate = false;

// first right shift 1 bit

for (it=frames.begin(); it!=frames.end(); ++it)

{

(\*it).reference /= 2;

}

//lookup, if duplicate

for (it=frames.begin(); it!=frames.end(); ++it)

{

if(page\_number == (\*it).frame)

{

duplicate = true;

break;

}

}

if(duplicate)

{

(\*it).reference = ((\*it).reference) | 0x80;

}

else

{

if(frames.size()<max\_frame\_nbr)

{

//then insert to the end

node.frame = page\_number;

node.reference = 0x80;

frames.push\_back(node);

}

else

{

//find a victim, and repalce it

list<lru\_clock\_enty>::iterator least=frames.begin();

for (it=frames.begin(); it!=frames.end(); ++it)

{

if((\*least).reference>(\*it).reference)

{

least = it;

}

}

(\*least).reference = 0x80;

(\*least).frame = page\_number;

replace\_nbr++;

}

}

}

//show();

return 0;

}

void lru\_ref8::show()

{

list<lru\_clock\_enty>::iterator it;

cout << "lru ref8:"<<endl;

cout << "physical frames:"<<endl;

for (it=frames.begin(); it!=frames.end(); ++it)

{

cout << \*it << " ";

}

cout <<endl;

}

lru\_ref8::~lru\_ref8()

{

frames.clear();

}