Group Member: Eric Simmons , Yijie Sun

**Instructions on how to run the project:**

**Part 1:**

Place your input file in the same folder. Use terminal to run the program by command:

$ ./LC <filename>

The input file example:

a s1 r3 b

c r2 s3

r1 d s2 e

The ouput:

p0: 1 2 8 9

p1: 1 6 7

p2: 3 4 5 6

**Part 2:**

Place your input file in the same folder as the executable and run the executable or run the cpp file through a compiler.

The input file should be formated so that instructions are separated by spaces and a new line represents a new process.

Example:

Input file format:

1 2 8 9

1 6 7 0

3 4 5 6

Output:

Names of activity

A s1 r3 e

B r2 s3 null

R1 c s2 d

**Explanation of Code**

**Part 1 (Explained in PsuedoCode):**

Input: some processes with different event

Example:

P0: a s1 r3 b

P1: c r2 s3

P2: r1 d s2 e

Output: logic clock for each process

//we create a class to holds event types

Class event{

String name;

Int type; // internal = 0, send = 1, recv = 2

Int numEvent; // only for send and recv event

Int logicClock;

}

map<int, int> srMap;

Int globalLogicClock = 1;

Make queues for each process

Make vectors for each process to store the logic clock for each event

do{

For each process:

curEvent = the front of queue(process)

If (curEvent is not equal to NULL) {

if(curEvent.type != recv) {

// internal and send event

curEvent.logicClock = globalLogicClock;

if( curEvent.type == send) {

srMap[curEvent.numEvent] = curEvent.logicClock;

}

Vector.push\_back the logic clock

queue(process).pop the front one;

} else {

// receive event

if( srMap[curEvent.numEvent] exist && srMap[curEvent.numEvent] +1 == globalLogicClock) {

curEvenvt.logicClock = globalLogicClock;

Vector.push\_back the logic clock

queue(process).pop the front one;

}

}

}

globalLogicClock ++;

}while(all the queue is empty)

Output the logic clock(vector) for each event in the process.

**Part2 (explained using the actual code plus comments):**  
  
#include <iostream>  
#include <queue>  
#include <vector>  
#include <sstream>  
#include <string>  
  
using namespace std;

/\*orderAllActivities() is where the majority of the action occurs and logic occurs. Everything else is more or less a data structure to keep track of what activity each process holds or to keep track of what letter of the alphabet we are currently at/send and receive pair.\*/

//class used to store activity information  
class Activity{  
 private:  
 string name;  
 int time;  
 public:  
 void setname(string n){  
 name=n;  
 }  
 string getname(){  
 return name;  
 }  
 int gettime(){  
 return time;  
 }  
 void settime(int t){  
 time=t;  
 }  
  
};  
  
  
//class that stores activities for each process  
class Process{  
 private:  
 vector<Activity> activities;  
 int p\_No;  
 public:

//we push activities into queue to be read

void pushActivity(string s){  
 istringstream iss(s);  
 while (iss){  
 string sub;  
 iss >> sub;  
 if(sub!=""){  
 Activity n;  
 n.settime(atoi(sub.c\_str()));  
 activities.push\_back(n);  
 }  
 }  
 }

//getters and setters  
 void setPno(int i){  
 p\_No=i;  
 }  
 int getPno(){  
 return p\_No;  
 }  
 int getSize(){  
 return (int)activities.size();  
 }  
 int getTime(int j){  
 return activities[j].gettime();  
 }  
 void setName(int j, string name){  
   
 activities[j].setname(name);  
 }  
 string getName(int j){  
 return activities[j].getname();  
 }  
 void printActivityTime(){  
 int len=getSize();  
 for(int i=0;i<len;i++){  
 cout<<activities[i].gettime()<<" ";  
 }  
 cout<<endl;  
 }  
 void printAName()  
 { for(int i=0; i<getSize();i++)  
 {  
 cout<<getName(i)<<" ";  
 }  
 cout<<endl;  
   
 }  
   
   
};  
  
//we create a log that keeps track of the time and whether something is dependent on another //time, an example of a dependency is how r1 is dependent on the time of s1, see //orderAllActivity() for more information on how this is exactly is used  
class Log{  
 int pnum;  
 int anum;  
 int time;  
 int dtag; //0 independ 1 depend  
public:  
 void setLog(int pNum, int t, int aNum ){  
 pnum=pNum;  
 time=t;  
 anum=aNum;  
 dtag=-1; //default  
 }  
 void setDtag(int d){  
 dtag=d;  
 }  
 int getT(){  
 return time;}  
 int getP(){  
 return pnum;}  
 int getA(){  
 return anum;}  
 int getD(){  
 return dtag;}  
   
};

//we create two queue whose only purpose is to hold letters of the alphabet and to number //different receive and send events. As letters of the alphabet and send and receive are used they //are popped from the queue

queue<string> abc;  
queue<string> sendRec;

// This just assume we only have one send and receive event pair  
void initNameQueue(int num)  
{  
 for(int i=0;i<num; i++)  
 { string c;  
 c=97+i;  
 abc.push(c);  
 }  
 sendRec.push("s1");  
 sendRec.push("r1");  
 sendRec.push("s2");  
 sendRec.push("r2");  
 sendRec.push("s3");  
 sendRec.push("r3");  
 sendRec.push("s4");  
 sendRec.push("r4");  
 sendRec.push("s5");  
 sendRec.push("r5");  
 sendRec.push("s6");  
 sendRec.push("r6");  
 sendRec.push("s7");  
 sendRec.push("r7");  
 sendRec.push("s8");  
 sendRec.push("r8");  
 sendRec.push("s9");  
 sendRec.push("r9");  
 sendRec.push("s10");  
 sendRec.push("r10");  
 sendRec.push("s11");  
 sendRec.push("r11");  
 sendRec.push("s12");  
 sendRec.push("r12");  
}  
  
  
/\*The psuedocode behind the idea of orderAllActivities

orderAllActivity(vector<Process> processes)

{ add all activities’ info to log;

Order the log array by time

if( log.time is not sequentially)

Print out “ the process is invalid” then exit

Check the dependency, (if all the previous values are in the same process,it is dependent)

Mark the dtag as 0 independent or 1 dependent

Assign name the independent activities in abc queue

Assign name the dependent activities in sendRec queue

Print out all activity name by process

}\*/

void orderAllActivity(vector<Process> processes)  
{ vector<Log> log;  
 //load log  
 int len=(int)processes.size();  
 for( int i=0;i<len;i++ )  
 {  
 for(int j=0;j<processes[i].getSize();j++)  
 {  
 if( processes[i].getTime(j)!= 0)  
 {  
 Log l;  
 l.setLog(processes[i].getPno(), processes[i].getTime(j),j );  
 log.push\_back(l);  
 }  
 else  
 {  
 processes[i].setName(j,"null");  
  
 }  
 }  
 }  
   
 //order it  
 int seq=1;  
 for(int i=0; i<log.size();i++)  
 {  
 for(int j=i; j<log.size();j++)  
 { if (log[i].getT()>log[j].getT())  
 {  
 Log tem=log[i];  
 log[i]=log[j];  
 log[j]=tem;  
 }  
 }  
 if(log[i].getT()<=seq)  
 { if(log[i].getT()==seq)  
 {seq+=1;  
 }  
 }  
 else{cout<<"incorect";  
 exit(0);  
 }  
  
 }  
   
 int p;  
 for(int i=0; i<log.size();i++)  
 {  
 p=log[i].getP();  
 if(log[i].getT()==1)  
 {  
 log[i].setDtag(0);  
 continue;  
 }  
 for (int j=0; j<i;j++)  
 { if(log[j].getP()==p)  
 { if(log[j].getT()==log[i].getT())  
 {  
 cout<<"incorrect"<<endl;  
 exit(0);  
 }  
 if( log[j].getT()+1==log[i].getT())  
 {  
 log[i].setDtag(0);  
 break;  
 }  
 }  
 }  
 if(log[i].getD()==-1)  
 { for(int j=i-1; j>0;j--)  
 { if(log[j].getT()+1==log[i].getT())  
 { if(log[j].getD()==0)  
 {  
 log[j].setDtag(1);  
 log[i].setDtag(1);  
 break;  
 }  
 }  
 }  
 if(log[i].getD()==-1)  
 {cout<<"incorrect"<<endl;  
 exit(0);  
 }  
 }  
 }  
   
   
   
 int a;  
   
 for(int i=0; i<log.size();i++)  
 { p=log[i].getP();  
 a=log[i].getA();  
 if(log[i].getD()==0)  
 { string ab=abc.front();  
 processes[p-1].setName(a, ab);  
 abc.pop();  
 }  
 else{  
 string sr=sendRec.front();  
 processes[p-1].setName(a, sr);  
 sendRec.pop();  
 }  
 }  
  
 for(int i=0; i<processes.size();i++)  
 {  
 processes[i].printAName();  
   
 }

/\*here we read a file

each line = a set of instructions for a process

after we’re done reading a line assign that process a number starting from 1 and push the set of instructions into a queue that to be read later

We then call initiate queue which populates 2 queue’s that hold letters of the alphabet and send and receive instructions that have yet to be assigned.

We then call OrderAllActivities() which decides whether the ordering is invalid/valid and what what type of instruction each number represents

\*/

int main(int argc, char\*argv[]){  
   
   
 vector<Process> processes;  
   
   
 // ifstream in(\*argv);  
 ifstream in(argv[0]);  
 string temp;  
 string str;  
 int count=1;  
 while(getline(in,temp))  
 {  
 Process px;  
 px.pushActivity(temp);  
 px.setPno(count);  
 processes.push\_back(px);  
 count++;  
 }  
   
 initNameQueue(20);  
   
 orderAllActivity(processes);  
}

**Output Examples:**

**Part 1:**

Input:

a s1 r3 b

c r2 s3

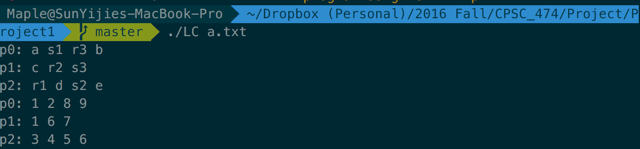
r1 d s2 e

Output:

p0: 1 2 8 9

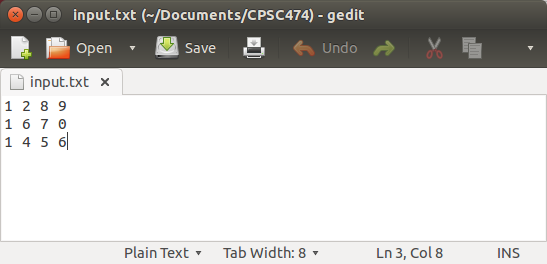
p1: 1 6 7

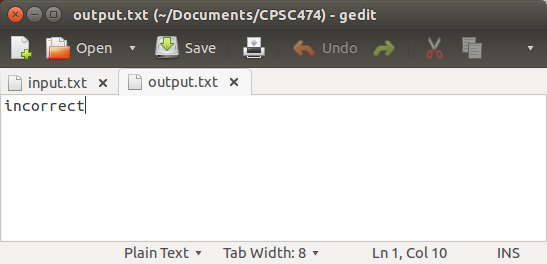
p2: 3 4 5 6



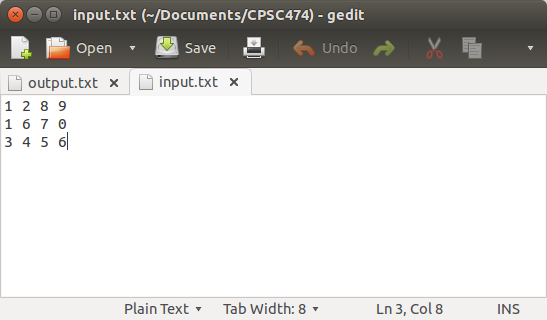
**Part 2:**

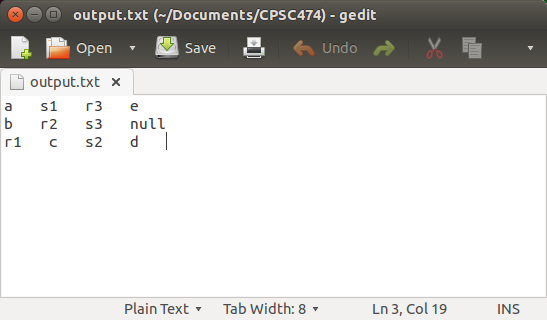
Test 1





Test 2





Test 3

