```
/**********************************
 * FILE NAME: Application.h
 *
 * DESCRIPTION: Header file of all classes pertaining to the Application Layer
 **********************************/

#ifndef _APPLICATION_H_
#define _APPLICATION_H_

#include "stdincludes.h"
#include "MP1Node.h"
#include "Log.h"
#include "Params.h"
#include "Member.h"
#include "EmulNet.h"
#include "Queue.h"

/**
 * global variables
 */
int nodeCount = 0;

/*
 * Macros
 */
#define ARGS_COUNT 2
#define TOTAL_RUNNING_TIME 700

/**
 * CLASS NAME: Application
 *
 * DESCRIPTION: Application layer of the distributed system
 */
class Application{
private:
        // Address for introduction to the group
        // Coordinator Node
        char JOINADDR[30];
        EmulNet *en;
    Log *log;
        MP1Node **mp1;
        Params *par;
public:
        Application(char *);
        virtual ˜Application();
        Address getjoinaddr();
        int run();
        void mp1Run();
        void fail();
};

#endif /* _APPLICATION_H__ */
```

```
/**********************************
 * FILE NAME: EmulNet.h
 *
 * DESCRIPTION: Emulated Network classes header file
 **********************************/

#ifndef _EMULNET_H_
#define _EMULNET_H_

#define MAX_NODES 1000
#define MAX_TIME 3600
#define ENBUFFSIZE 30000

#include "stdincludes.h"
#include "Params.h"
#include "Member.h"

using namespace std;

/**
 * Struct Name: en_msg
 */
typedef struct en_msg {
        // Number of bytes after the class
        int size;
        // Source node
        Address from;
        // Destination node
        Address to;
}en_msg;

/**
 * Class Name: EM
 */
class EM {
public:
        int nextid;
        int currbuffsize;
        int firsteltindex;
        en_msg* buff[ENBUFFSIZE];
        EM() {}
        EM& operator = (EM &anotherEM) {
                this->nextid = anotherEM.getNextId();
                this->currbuffsize = anotherEM.getCurrBuffSize();
                this->firsteltindex = anotherEM.getFirstEltIndex();
                int i = this->currbuffsize;
                while (i > 0) {
                        this->buff[i] = anotherEM.buff[i];
                        i--;
                }
                return *this;
        }
        int getNextId() {
                return nextid;
        }
        int getCurrBuffSize() {
                return currbuffsize;
        }
        int getFirstEltIndex() {
                return firsteltindex;
        }
        void setNextId(int nextid) {
                this->nextid = nextid;
        }
        void settCurrBuffSize(int currbuffsize) {
                this->currbuffsize = currbuffsize;
        }
        void setFirstEltIndex(int firsteltindex) {
                this->firsteltindex = firsteltindex;
        }
        virtual ~EM() {}
};

/**
 * CLASS NAME: EmulNet
 *
 * DESCRIPTION: This class defines an emulated network
 */
class EmulNet
{
private:
        Params* par;
        int sent_msgs[MAX_NODES + 1][MAX_TIME];
        int recv_msgs[MAX_NODES + 1][MAX_TIME];
        int enInited;
        EM emulnet;
public:
        EmulNet(Params *p);
        EmulNet(EmulNet &anotherEmulNet);
        EmulNet& operator = (EmulNet &anotherEmulNet);
        virtual ˜EmulNet();
        void *ENinit(Address *myaddr, short port);
        int ENsend(Address *myaddr, Address *toaddr, string data);
        int ENsend(Address *myaddr, Address *toaddr, char *data, int size);
        int ENrecv(Address *myaddr, int (* enq)(void *, char *, int), struct timeval *t,
int times, void *queue);
        int ENcleanup();
};

#endif /* _EMULNET_H_ */
```

```
/**********************************
 * FILE NAME: Log.h
 *
 * DESCRIPTION: Header file of Log class
 **********************************/

#ifndef _LOG_H_
#define _LOG_H_

#include "stdincludes.h"
#include "Params.h"
#include "Member.h"

/*
 * Macros
 */
// number of writes after which to flush file
#define MAXWRITES 1
#define MAGIC_NUMBER "CS425"
#define DBG_LOG "dbg.log"
#define STATS_LOG "stats.log"

/**
 * CLASS NAME: Log
 *
 * DESCRIPTION: Functions to log messages in a debug log
 */
class Log{
private:
        Params *par;
        bool firstTime;
public:
        Log(Params *p);
        Log(const Log &anotherLog);
        Log& operator = (const Log &anotherLog);
        virtual ~Log();
        void LOG(Address *, const char * str, ...);
        void logNodeAdd(Address *, Address *);
        void logNodeRemove(Address *, Address *);
};

#endif /* _LOG_H_ */
```

```
/**********************************
 * FILE NAME: MP1Node.cpp
 *
 * DESCRIPTION: Membership protocol run by this Node.
 *                              Header file of MP1Node class.
 **********************************/

#ifndef _MP1NODE_H_
#define _MP1NODE_H_

#include "stdincludes.h"
#include "Log.h"
#include "Params.h"
#include "Member.h"
#include "EmulNet.h"
#include "Queue.h"

/**
 * Macros
 */
#define TREMOVE 20
#define TFAIL 5

/*
 * Note: You can change/add any functions in MP1Node.{h,cpp}
 */

/**
 * Message Types
 */
enum MsgTypes{
    JOINREQ,
    JOINREP,
    DUMMYLASTMSGTYPE
};

/**
 * STRUCT NAME: MessageHdr
 *
 * DESCRIPTION: Header and content of a message
 */
typedef struct MessageHdr {
        enum MsgTypes msgType;
}MessageHdr;

/**
 * CLASS NAME: MP1Node
 *
 * DESCRIPTION: Class implementing Membership protocol functionalities for failure detect
ion
 */
class MP1Node {
private:
        EmulNet *emulNet;
        Log *log;
        Params *par;
        Member *memberNode;
        char NULLADDR[6];

public:
        MP1Node(Member *, Params *, EmulNet *, Log *, Address *);
        Member * getMemberNode() {
                return memberNode;
        }
        int recvLoop();
        static int enqueueWrapper(void *env, char *buff, int size);
        void nodeStart(char *servaddrstr, short serverport);
        int initThisNode(Address *joinaddr);
        int introduceSelfToGroup(Address *joinAddress);
        int finishUpThisNode();
        void nodeLoop();
        void checkMessages();
        bool recvCallBack(void *env, char *data, int size);
        void nodeLoopOps();
        int isNullAddress(Address *addr);
        Address getJoinAddress();
        void initMemberListTable(Member *memberNode);
        void printAddress(Address *addr);
        virtual ~MP1Node();
};

#endif /* _MP1NODE_H_ */
```

```
/**********************************
 * FILE NAME: MP1Node.cpp
 *
 * DESCRIPTION: Membership protocol run by this Node.
 *                              Header file of MP1Node class.
 **********************************/

#ifndef _MP1NODE_H_
#define _MP1NODE_H_

#include "stdincludes.h"
#include "Log.h"
#include "Params.h"
#include "Member.h"
#include "EmulNet.h"
#include "Queue.h"

/**
 * Macros
 */
#define TREMOVE 20
#define TFAIL 5

/*
 * Note: You can change/add any functions in MP1Node.{h,cpp}
 */

/**
 * Message Types
 */
enum MsgTypes{
    JOINREQ,
    JOINREP,
    DUMMYLASTMSGTYPE
};

/**
 * STRUCT NAME: MessageHdr
 *
 * DESCRIPTION: Header and content of a message
 */
typedef struct MessageHdr {
        enum MsgTypes msgType;
}MessageHdr;

/**
 * CLASS NAME: MP1Node
 *
 * DESCRIPTION: Class implementing Membership protocol functionalities for failure detect
ion
 */
class MP1Node {
private:
        EmulNet *emulNet;
        Log *log;
        Params *par;
        Member *memberNode;
        char NULLADDR[6];

public:
        MP1Node(Member *, Params *, EmulNet *, Log *, Address *);
        Member * getMemberNode() {
                return memberNode;
        }
        int recvLoop();
        static int enqueueWrapper(void *env, char *buff, int size);
        void nodeStart(char *servaddrstr, short serverport);
        int initThisNode(Address *joinaddr);
        int introduceSelfToGroup(Address *joinAddress);
        int finishUpThisNode();
        void nodeLoop();
        void checkMessages();
        bool recvCallBack(void *env, char *data, int size);
        void nodeLoopOps();
        int isNullAddress(Address *addr);
        Address getJoinAddress();
        void initMemberListTable(Member *memberNode);
        void printAddress(Address *addr);
        virtual ~MP1Node();
};

#endif /* _MP1NODE_H_ */
```

```
/**********************************
 * FILE NAME: Member.h
 *
 * DESCRIPTION: Definition of all Member related class
 **********************************/

#ifndef MEMBER_H_
#define MEMBER_H_

#include "stdincludes.h"

/**
 * CLASS NAME: q_elt
 *
 * DESCRIPTION: Entry in the queue
 */
class q_elt {
public:
	void *elt;
	int size;
	q_elt(void *elt, int size);
};

/**
 * CLASS NAME: Address
 *
 * DESCRIPTION: Class representing the address of a single node
 */
class Address {
public:
	char addr[6];
	Address() {}
	// Copy constructor
	Address(const Address &anotherAddress);
	 // Overloaded = operator
	Address& operator =(const Address &anotherAddress);
	bool operator ==(const Address &anotherAddress);
	Address(string address) {
		size_t pos = address.find(":");
		int id = stoi(address.substr(0, pos));
		short port = (short)stoi(address.substr(pos + 1, address.size()-pos-1));
		memcpy(&addr[0], &id, sizeof(int));
		memcpy(&addr[4], &port, sizeof(short));
	}
	string getAddress() {
		int id = 0;
		short port;
		memcpy(&id, &addr[0], sizeof(int));
		memcpy(&port, &addr[4], sizeof(short));
		return to_string(id) + ":" + to_string(port);
	}
	void init() {
		memset(&addr, 0, sizeof(addr));
	}
};

/**
 * CLASS NAME: MemberListEntry
 *
 * DESCRIPTION: Entry in the membership list
 */
class MemberListEntry {
public:
	int id;
	short port;
	long heartbeat;
	long timestamp;
	MemberListEntry(int id, short port, long heartbeat, long timestamp);
	MemberListEntry(int id, short port);
	MemberListEntry(): id(0), port(0), heartbeat(0), timestamp(0) {}
	MemberListEntry(const MemberListEntry &anotherMLE);
```

```
        MemberListEntry& operator =(const MemberListEntry &anotherMLE);
        int getid();
        short getport();
        long getheartbeat();
        long gettimestamp();
        void setid(int id);
        void setport(short port);
        void setheartbeat(long hearbeat);
        void settimestamp(long timestamp);
};

/**
 * CLASS NAME: Member
 *
 * DESCRIPTION: Class representing a member in the distributed system
 */
// Declaration and definition here
class Member {
public:
        // This member’s Address
        Address addr;
        // boolean indicating if this member is up
        bool inited;
        // boolean indicating if this member is in the group
        bool inGroup;
        // boolean indicating if this member has failed
        bool bFailed;
        // number of my neighbors
        int nnb;
        // the node’s own heartbeat
        long heartbeat;
        // counter for next ping
        int pingCounter;
        // counter for ping timeout
        int timeOutCounter;
        // Membership table
        vector<MemberListEntry> memberList;
        // My position in the membership table
        vector<MemberListEntry>::iterator myPos;
        // Queue for failure detection messages
        queue<q_elt> mp1q;
        /**
         * Constructor
         */
        Member(): inited(false), inGroup(false), bFailed(false), nnb(0), heartbeat(0), pi
ngCounter(0), timeOutCounter(0) {}
        // copy constructor
        Member(const Member &anotherMember);
        // Assignment operator overloading
        Member& operator =(const Member &anotherMember);
        virtual ~Member() {}
};

#endif /* MEMBER_H_ */
```

```
/**********************************
 * FILE NAME: Params.h
 *
 * DESCRIPTION: Header file of Parameter class
 **********************************/

#ifndef _PARAMS_H_
#define _PARAMS_H_

#include "stdincludes.h"
#include "Params.h"
#include "Member.h"

enum testTYPE { CREATE_TEST, READ_TEST, UPDATE_TEST, DELETE_TEST };

/**
 * CLASS NAME: Params
 *
 * DESCRIPTION: Params class describing the test cases
 */
class Params{
public:
	int MAX_NNB;                // max number of neighbors
	int SINGLE_FAILURE;                     // single/multi failure
	double MSG_DROP_PROB;           // message drop probability
	double STEP_RATE;                   // dictates the rate of insertion
	int EN_GPSZ;                        // actual number of peers
	int MAX_MSG_SIZE;
	int DROP_MSG;
	int dropmsg;
	int globaltime;
	int allNodesJoined;
	short PORTNUM;
	Params();
	void setparams(char *);
	int getcurrtime();
};

#endif /* _PARAMS_H_ */
```

```
/**********************************
 * FILE NAME: Queue.h
 *
 * DESCRIPTION: Header file for std::<queue> related functions
 **********************************/

#ifndef QUEUE_H_
#define QUEUE_H_

#include "stdincludes.h"
#include "Member.h"

/**
 * Class name: Queue
 *
 * Description: This function wraps std::queue related functions
 */
class Queue {
public:
	Queue() {}
	virtual ~Queue() {}
	static bool enqueue(queue<q_elt> *queue, void *buffer, int size) {
		q_elt element(buffer, size);
		queue->emplace(element);
		return true;
	}
};

#endif /* QUEUE_H_ */
```

```
/**********************************
 * FILE NAME: stdincludes.h
 *
 * DESCRIPTION: standard header file
 **********************************/

#ifndef _STDINCLUDES_H_
#define _STDINCLUDES_H_

/*
 * Macros
 */
#define RING_SIZE 512
#define FAILURE -1
#define SUCCESS 0

/*
 * Standard Header files
 */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>
#include <stdarg.h>
#include <unistd.h>
#include <fcntl.h>
#include <execinfo.h>
#include <signal.h>
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <algorithm>
#include <queue>
#include <fstream>

using namespace std;

#define STDCLLBKARGS (void *env, char *data, int size)
#define STDCLLBKRET     void
#define DEBUGLOG 1

#endif  /* _STDINCLUDES_H_ */
```