```
============== error.h ===============
#ifndef _ERROR_
#define _ERROR_

class Error {
      public:
            static int NO_USER;
            static int NO_SERVER;
            static int OK_CODE;
            static int NO_MESSAGE;
            static int SERVER_DOWN;
            static int INVALID_USER;
            static int NO_SERVER_SPACE;
};
#endif

============= error.cpp ===============
#include "error.h"

int Error::NO_USER = 0;
int Error::NO_SERVER = 1;
int Error::OK_CODE = 2;
int Error::NO_MESSAGE = 3;
int Error::SERVER_DOWN = 4;
int Error::INVALID_USER = 5;
int Error::NO_SERVER_SPACE = 6;

============ email.h ==================
#ifndef _EMAIL_
#define _EMAIL_

#define MAX_ADDRESS 64
#define MAX_TEXT 1000
#define MAX_NAME 20

#include <iostream.h>

class Email {
      char from[MAX_ADDRESS];
      char to[MAX_ADDRESS];
      char text[MAX_TEXT];
    public:
      EMail(char[], char[], char[]);
      EMail(EMail&);

      char* getFrom();
      char* getTo();
      char* getMessage();

      void print(ostream&);
};
#endif

============== email.cpp =================
#include <string.h>
#include "email.h"

EMail::EMail(char from[], char to[], char text[]) {
  strcpy(this->from, from);
  strcpy(this->to, to);
  strcpy(this->text, text);
```

```
}

EMail::EMail(EMail& email) {
  strcpy(from, email.from);
  strcpy(to, email.to);
  strcpy(text, email.text);
}

char* EMail::getFrom() {
  return from;
}

char* EMail::getTo() {
  return to;
}

char* EMail::getMessage() {
  return text;
}

void EMail::print(ostream& out) {
  out << "[From:] " << from << endl;
  out << "[To:] "   << to << endl;
  out << "[Message:] " << text << endl << endl;
}
```

`================== mailsrv.h ==============`

```
#ifndef _MailServer_
#define _MailServer_

#include "email.h"

#define MAX_ITEMS 1000

class MailServer {
      static int INVALID_INDEX;

      EMail* elements[MAX_ITEMS];
      int lastElementIndex;

      char name[MAX_NAME];

      int checkUser(char[]);
      int getNextEmailIndex(char[], int);
      int deleteEmailByIndex(int);
     public:
      MailServer(char[]);

      int getEmailsNumber(char[]);
      int getNextEmail(char[], EMail*& );

      int post(EMail* email);
};
#endif
```

`================= mailsrv.cpp =================`

```
#include <string.h>
#include "mailsrv.h"
#include "error.h"

int MailServer::INVALID_INDEX = -1;
```

```
MailServer::MailServer(char name[]) {
  lastElementIndex = MailServer::INVALID_INDEX;
  strncpy(this->name, name, MAX_NAME);
}

int MailServer::getEmailsNumber(char user[]) {
  int i = 0;
  int counter = 0;

  while ((i = getNextEmailIndex(user, i)) != MailServer::INVALID_INDEX){
    counter++;
    i++;
  }

  return counter;
}

int MailServer::checkUser(char user[]) {
  return Error::OK_CODE;
}

int MailServer::getNextEmailIndex(char user[], int startIndex) {
  if (startIndex < 0)
    return MailServer::INVALID_INDEX;

  for (int i = startIndex; i <= lastElementIndex; i++)
    if (strcmp(elements[i]->getTo(), user) == 0)
      return i;

  return MailServer::INVALID_INDEX;
}

int MailServer::deleteEmailByIndex(int index) {
  for (int i = index; i < lastElementIndex; i++)
    elements[i] = elements[i + 1];
  lastElementIndex--;

  return Error::OK_CODE;
}

int MailServer::getNextEmail(char user[], EMail*& item) {
  int code = checkUser(user);

  if (code == Error::OK_CODE){
     int index = getNextEmailIndex(user, 0);

     if (index != MailServer::INVALID_INDEX){
       item = elements[index];
       deleteEmailByIndex(index);

       return Error::OK_CODE;
     }
     else
       return Error::NO_MESSAGE;
  }
  else
    return code;
}

int MailServer::post(EMail* email) {
```

```
  if (lastElementIndex < MAX_ITEMS) {
    if (checkUser(email->getTo()) != Error::OK_CODE)
      return Error::NO_USER;
    else {
      elements[++lastElementIndex] = email;

      return Error::OK_CODE;
    }
  }
  else
    return Error::NO_SERVER_SPACE;
}
```

```
==================== mailc.h ==============
#ifndef _MAILCLIENT_
#define _MAILCLIENT_

#include "mailsrv.h"
#include "email.h"

class MailClient {
      static char NO_USER[];

      MailServer* server;
      char user[MAX_NAME];
    public:
      MailClient();

      int getNextEmail(EMail*&);

      int sendMessage(char[], char[]);

      void setServer(MailServer* );
      MailServer* getServer();

      void setUser(char*);
      char* getUser();
};
#endif
```

```
=============== mailc.cpp ============
#include <string.h>
#include "mailc.h"
#include "error.h"

char MailClient::NO_USER[] = "@@@";

MailClient::MailClient() {
  server = NULL;
  strcpy(user, MailClient::NO_USER);
}

int MailClient::getNextEmail(EMail*& item) {
  if (server == NULL)
    return Error::NO_SERVER;
  else
    if (strcmp(user, MailClient::NO_USER) == 0)
      return Error::NO_USER;
    else
      return server->getNextEmail(user, item);
}
```

```
int MailClient::sendMessage(char to[], char text[]) {
  if (server == NULL)
    return Error::NO_SERVER;
  else
    if (strcmp(user, MailClient::NO_USER) == 0)
      return Error::NO_USER;
    else {
      EMail* item = new EMail(user, to, text);

      int result = server->post(item);

      if (result != Error::OK_CODE)
        delete item;

      return result;
    }
}

void MailClient::setServer(MailServer* server) {
  this->server = server;
}

MailServer* MailClient::getServer() {
  return server;
}

void MailClient::setUser(char* str) {
   strcpy(user, str);
}

char* MailClient::getUser() {
  return user;
}

================= mailer.cpp =============
#include <ctype.h>

#include "email.h"
#include "mailc.h"
#include "mailsrv.h"
#include "error.h"

void readUserName(MailClient* client) {
  char str[MAX_NAME];

  cout << "User name: "; cin >> str;
  client->setUser(str);
}

void sentEmail(MailClient* client) {
  char str[MAX_NAME];
  char buffer[MAX_TEXT];

  cout << "To: "; cin >> str;
  cout << "Message: "; cin >> buffer;
  if (client->sendMessage(str, buffer) == Error::OK_CODE)
    cout << "Message succesfully sent! \n";
  else
    cout << "Message could not be sent! \n";
}
```

```
void receiveEmails(MailClient* client) {
  EMail* item;

  while (client->getNextEmail(item) == Error::OK_CODE){
    item->print(cout);

    delete item;
    cout << " -------------------------\n";
  }
}

void newSession(MailClient* client) {
  char ch;

  while (1){
    cout << "0. Return \n";
    cout << "1. Set user \n";
    cout << "2. Send email \n";
    cout << "3. Receive emails \n";
    cin >> ch;

    switch (toupper(ch)){
      case '0' : return;
      case '1' : readUserName(client);
             break;
      case '2' : sentEmail(client);
             break;
      case '3' : receiveEmails(client);
             break;
    }
  }

}

void main(void) {
  char ch;

  MailServer* server1 = new MailServer("Samba");

  MailClient* outlook = new MailClient();
  outlook->setServer(server1);

  while (1){
    cout << "0. Return \n";
    cout << "1. New Session \n";
    cin >> ch;

    switch (toupper(ch)){
      case '0' : return;
      case '1' : newSession(outlook);
             break;
    }
  }
}
```

Temă

1. Concurs de admitere la facultate (se specifică numele facultății şi data de susținere a concursului).

Pentru fiecare concurent se cunosc următoarele informații de intrare:

- nume şi prenume
- data naşterii
- numărul şi seria buletinului de identitate
- pentru fiecare din cele două probe, trei note ale celor trei corectori.

Operații:

- sortare concurenți după nume
- calcul notă la fiecare probă
- calcul medie
- sortare concurenți după medie, iar pentru medii egale după nume
- separarea în 2 liste (admişi, respinşi)
- determinarea numărului de studenți admişi şi respinşi
- afişarea studenților admişi şi respinşi în ordinea descrescătoare a mediilor
- rezolvarea contestațiilor
- modificarea notelor la candidații cărora li s-au rezolvat contestațiile
- inserarea (după contestații) a studenților cu o medie mai mare sau egală cu cea a ultimului admis în lista cu studenții admişi pe poziția corespunzătoare
- ştergerea din lista studenților respinşi a celor admişi după contestație
- schimbarea poziției fiecărui student admis, căruia după contestație i s-a modificat media

Cerințe:

- cel puțin 4 clase;
- posibilitatea salvării tutror informațiilor într-un fişier, precum şi citirea datelor din fişier (supraâncărcarea operatorilor << şi >> pentru clasa ConcursAdmitere)
- dialogul cu utilizatorul prin intermediul unei interfețe grafice.