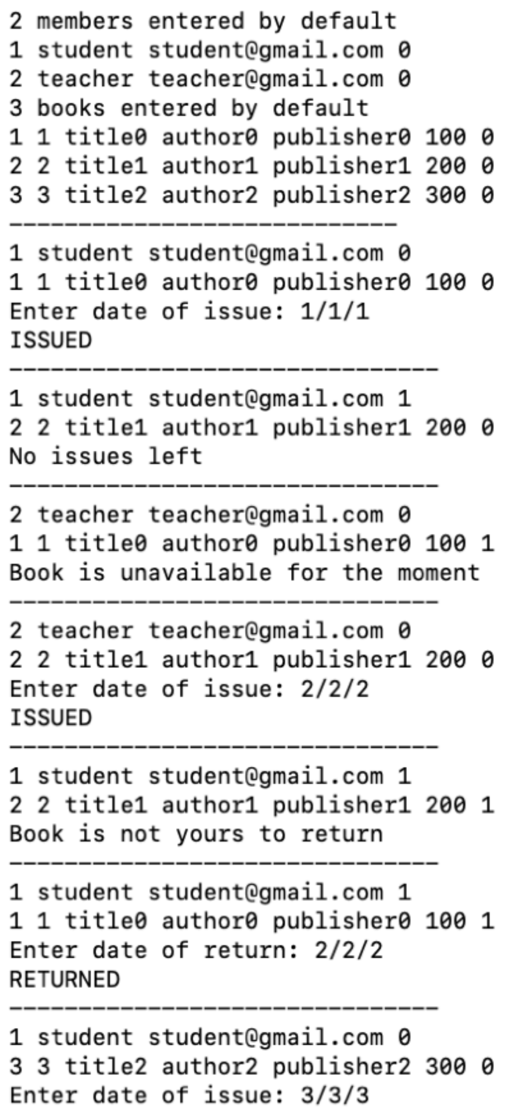
Ritabroto Ganguly 001910501090 BCSE-II

OOP Assignment 6

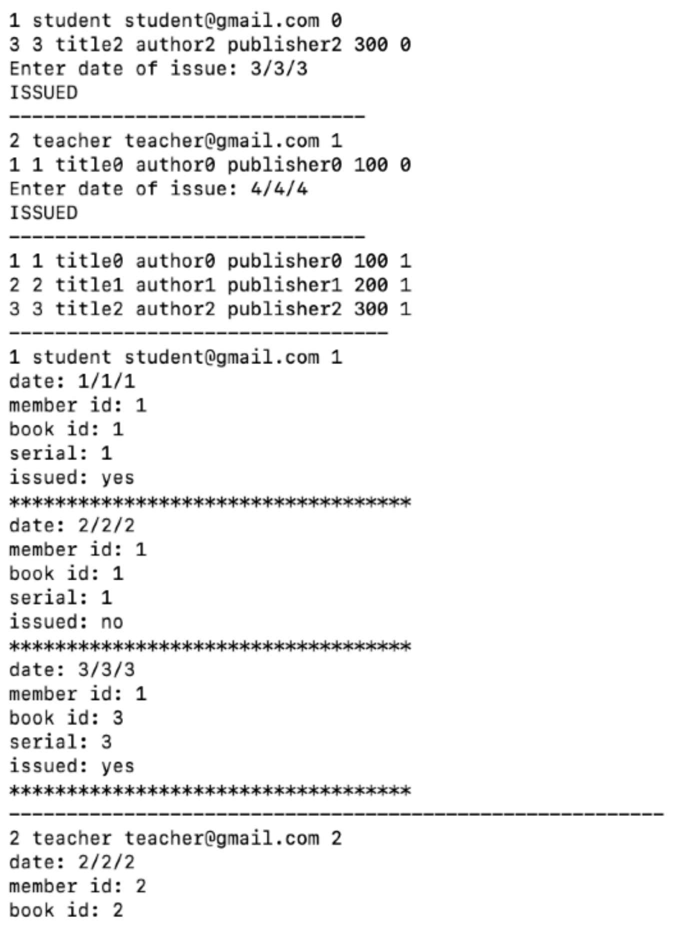
1. In a library, for each book book-id, serial number (denotes copy number of a book), title, author, publisher and price are stored. Book-id and serial number together will be unique identifier for a book. Members are either student or faculty. Each member has unique member-id. Name, e-mail, address are also to be stored. For any transaction (book issue or return), members are supposed to place transactions slip. User will submit member-id, book-id, and serial number (only for book return). While processing a transaction, check the validity of the member. While issuing, availability of a copy of the book is to be checked. While returning a book, it is to be checked whether this copy was issued to the member or not. A student member can have 2 books issued at a point of time. For faculty members it is 10. Transaction information is to be stored like date of transaction, member-id, book-id, serial number, returned or not. An entry is made when book is issued and updated when the book is returned. For storing the information consider files. Design the classes and implement.

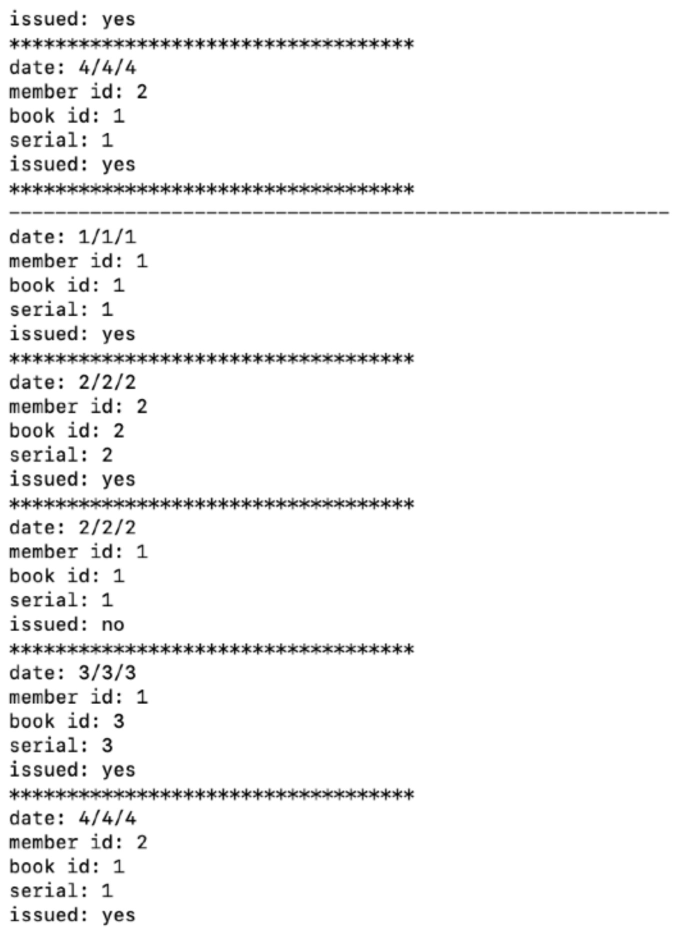
```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
using namespace std;

class book{
  static int ids;
  static int serials;
  int id;
  int serial;
  string title,author,publisher;
  float price;
  int issued;
  int memid;// needed to track which member has issued it currently for
member::return_book()
  public:
  book(const string& title,const string& author,const string& pub,float
price):title(title),author(author),publisher(pub),id(ids),serial(serials),price(price),issued
(0),memid(0){ids++;serials++;}

  int get_memid() const {return memid;}
  void set_issued(int i,int m=0){issued = i; memid = m;}
  int get_issued() const {return issued;}
  int get_id() const {return id;}
  int get_serial() const {return serial;}

  int operator==(const book& b){
    if(id==b.id && serial==b.serial)
      return 1;
    return 0;
  }

  friend ostream& operator<<(ostream& o,const book& t){
      o<<t.id<<" "<<t.serial<<" "<<t.title<<" "<<t.author<<" "<<t.publisher<<"
"<<t.price<<" "<<t.issued<<endl;
      return o;
  }
};
int book::ids = 1;
int book::serials = 1;
vector<book*> all_books;
book* find_book(const book& b){
  for(int i=0;i<all_books.size();i++)
    if(*all_books[i]==b)
      return all_books[i];
  return nullptr;
}

class Key;
class transaction{
  string dot;
  int memid;
  int bookid;
  int serial;
  int issued;
  transaction(const string& dot,int memid,int bookid,int serial,int issued):
  dot(dot),memid(memid),bookid(bookid),serial(serial),issued(issued){
    ofstream f(fname,ios::app);
    if(!f){
      cout<<"File non-existent in transaction cst"<<endl;
      exit(0);
    }
    //cout<<"In trans cst\n"<<*this;
    f.write((char*)this,sizeof(transaction));
    f.close();
  }
  public:
  static const string fname;
/*check_member is used in issue_book() an return_book()*/
  static transaction* check_member(const Key&,const string& dot,int memid,int
bookid,int serial,int issued);
  friend ostream& operator<<(ostream& o,const transaction& t){
      o<<"date: "<<t.dot<<endl;
      o<<"member id: "<<t.memid<<endl;
      o<<"book id: "<<t.bookid<<endl;
```

```
       o<<"serial: "<<t.serial<<endl;
       o<<"issued: "<<(t.issued ? "yes" : "no")<<endl;
       o<<"***********************************"<<endl;
       return o;
   }
   static void show_all_transactions() {
      cout<<"All Transactions in Library Ledger"<<endl;
      ifstream f(fname);
      if(!f){
        cout<<"File non-existent in show_all_transactions()"<<endl;
        exit(0);
      }
      transaction *t = new transaction("",-1,-1,-1,-1);
      f.read((char*)t,sizeof(transaction));
      while(!f.eof() && t->serial!=-1){
         cout<<*t;
         f.read((char*)t,sizeof(transaction));
      }
      f.close();
   }
};
const string transaction::fname = "Library_ledger.dat";

class member{
  protected:
  virtual void make_class_abstract() = 0;
  static int ids;
  int max_books;
  int id;
  int books;
  string name;
  string email;
  vector<transaction> trans_slip;
  member(const string& name,const string& email,int
m):name(name),email(email),books(0),max_books(m){id=ids;ids++;}
  public:
  void show_slip(){
    for(int i=0;i<trans_slip.size();i++)
        cout<<trans_slip[i];
  }
  int get_id() const {return id;}
  string get_name() const {return name;}
  string get_email() const {return email;}
  void set_email(const string& s){email = s;}
  int get_books() const {return books;}
  int get_max_books() const {return max_books;}
  void issue_book(book&);
  void return_book(book&);
  friend ostream& operator<<(ostream& o,const member& t){
       o<<t.id<<" "<<t.name<<" "<<t.email<<" "<<t.books<<endl;
       return o;
  }
};
int member::ids = 1;
vector<member*> all_members;

/*check_member is used in issue_book() an return_book()*/
transaction* transaction::check_member(const Key&,const string& dot,int
memid,int bookid,int serial,int issued){
      for(int i=0;i<all_members.size();i++)
         if(all_members[i]->get_id()==memid)
            return new transaction(dot,memid,bookid,serial,issued);
      return nullptr;
}

class student: public member{
  void make_class_abstract(){cout<<"Memeber is abstract, student is not"<<endl;}
  public:
  student(const string& name,const string& email):member(name,email,1){}
};

class faculty: public member{
  void make_class_abstract(){cout<<"Memeber is abstract, faculty is not"<<endl;}
  public:
  faculty(const string& name,const string& email):member(name,email,2){}
};

/* Key class following passkey idiom to restrict access to
transaction::all_transactions vector from transaction::check_member()*/
class Key{
   Key(){}
   Key(const Key&){}

   friend void member::issue_book(book&);
   friend void member::return_book(book&);
};
  void member::issue_book(book& b){
  //   cout<<"books="<<books<<" max_books="<<max_books<<"
issued="<<b.get_issued()<<endl;
    cout<<*this;
    if(books<get_max_books()){
      if(find_book(b)==nullptr){
        cout<<"Invalid book id"<<endl;
        cout<<"------------------------------"<<endl;
        return;
      }
      cout<<b;
      if(b.get_issued()==1){
        cout<<"Book is unavailable for the moment"<<endl;
        cout<<"-------------------------------"<<endl;
        return;
      }
      string s;
      cout<<"Enter date of issue: ";
      cin>>s;
      transaction* t =
transaction::check_member(Key(),s,id,b.get_id(),b.get_serial(),1);//check if member
id is valid
      if(t){
        trans_slip.push_back(*t);
        free(t);
        b.set_issued(1,this->id);
        books++;
        cout<<"ISSUED"<<endl;
      }else
        cout<<"Invalid member id"<<endl;
    }else{
      cout<<b;
      cout<<"No issues left"<<endl;
    }
    cout<<"-------------------------------"<<endl;
  }
  void member::return_book(book& b){
    cout<<*this;
    if(books>0){
      if(find_book(b)==nullptr){
        cout<<"Invalid book id"<<endl;
        cout<<"-------------------------------"<<endl;
        return;
      }
      cout<<b;
      if(b.get_issued()==0){
        cout<<"Book has not been issued"<<endl;
        cout<<"-------------------------------"<<endl;
        return;
      }
      if(b.get_memid() != id){
        cout<<"Book is not yours to return"<<endl;
        cout<<"-------------------------------"<<endl;
        return;
      }
      string s;
      cout<<"Enter date of return: ";
      cin>>s;
      transaction* t =
transaction::check_member(Key(),s,id,b.get_id(),b.get_serial(),0);//check if member
id is valid
      if(t){
        trans_slip.push_back(*t);
        free(t);
        b.set_issued(0);
        books--;
        cout<<"RETURNED"<<endl;
      }else
        cout<<"Invalid member id"<<endl;
    }else{
      cout<<b;
      cout<<"Can't return book"<<endl;
    }
    cout<<"------------------------------"<<endl;
  }

int main() {
  ofstream f(transaction::fname);
  if(!f){
    cout<<"File could not be opened"<<endl;
    exit(0);
  }
  f.close();

  //transaction::show_all_transactions();
  member* s1 = new student("student","student@gmail.com");
  member* f1 = new faculty("teacher","teacher@gmail.com");
  all_members.push_back(s1);
  all_members.push_back(f1);
  cout<<*s1<<*f1;

  book *b;
  for(int i=0;i<3;i++){
     string t = "title" + to_string(i);
     string a = "author" + to_string(i);
     string p = "publisher" + to_string(i);
     b = new book(t,a,p,(i+1)*100);
     all_books.push_back(b);
     cout<<*b;
  }

  s1->issue_book(*all_books[0]);
  s1->issue_book(*all_books[1]);
  f1->issue_book(*all_books[0]);
  f1->issue_book(*all_books[1]);
  s1->return_book(*all_books[1]);
  s1->return_book(*all_books[0]);
  s1->issue_book(*all_books[2]);
  f1->issue_book(*all_books[0]);

  for(int i=0;i<all_books.size();i++)
     cout<<*all_books[i];
  cout<<"--------------------------------"<<endl;
  for(int i=0;i<all_members.size();i++){
     cout<<*all_members[i];
     all_members[i]->show_slip();
     cout<<"--------------------------------------------------------"<<endl;
  }
  transaction::show_all_transactions();
  cout<<"-----------------------------------------------------------"<<endl;
  return 0;
}
```



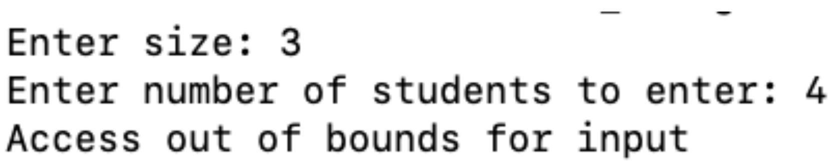
```
2 members entered by default
1 student student@gmail.com 0
2 teacher teacher@gmail.com 0
3 books entered by default
1 1 title0 author0 publisher0 100 0
2 2 title1 author1 publisher1 200 0
3 3 title2 author2 publisher2 300 0
----------------------------
1 student student@gmail.com 0
1 1 title0 author0 publisher0 100 0
Enter date of issue: 1/1/1
ISSUED
-------------------------------
1 student student@gmail.com 1
2 2 title1 author1 publisher1 200 0
No issues left
-------------------------------
2 teacher teacher@gmail.com 0
1 1 title0 author0 publisher0 100 1
Book is unavailable for the moment
-------------------------------
2 teacher teacher@gmail.com 0
2 2 title1 author1 publisher1 200 0
Enter date of issue: 2/2/2
ISSUED
-------------------------------
1 student student@gmail.com 1
2 2 title1 author1 publisher1 200 1
Book is not yours to return
-------------------------------
1 student student@gmail.com 1
1 1 title0 author0 publisher0 100 1
Enter date of return: 2/2/2
RETURNED
-------------------------------
1 student student@gmail.com 0
3 3 title2 author2 publisher2 300 0
Enter date of issue: 3/3/3
```


```
1 student student@gmail.com 0
3 3 title2 author2 publisher2 300 0
Enter date of issue: 3/3/3
ISSUED
-------------------------------
2 teacher teacher@gmail.com 1
1 1 title0 author0 publisher0 100 0
Enter date of issue: 4/4/4
ISSUED
-------------------------------
1 1 title0 author0 publisher0 100 1
2 2 title1 author1 publisher1 200 1
3 3 title2 author2 publisher2 300 1
---------------------------------
1 student student@gmail.com 1
date: 1/1/1
member id: 1
book id: 1
serial: 1
issued: yes
***********************************
date: 2/2/2
member id: 1
book id: 1
serial: 1
issued: no
***********************************
date: 3/3/3
member id: 1
book id: 3
serial: 3
issued: yes
***********************************
---------------------------------------------------------
2 teacher teacher@gmail.com 2
date: 2/2/2
member id: 2
book id: 2
```


```
issued: yes
***********************************
date: 4/4/4
member id: 2
book id: 1
serial: 1
issued: yes
***********************************
---------------------------------------------------------
date: 1/1/1
member id: 1
book id: 1
serial: 1
issued: yes
***********************************
date: 2/2/2
member id: 2
book id: 2
serial: 2
issued: yes
***********************************
date: 2/2/2
member id: 1
book id: 1
serial: 1
issued: no
***********************************
date: 3/3/3
member id: 1
book id: 3
serial: 3
issued: yes
***********************************
date: 4/4/4
member id: 2
book id: 1
serial: 1
issued: yes
```

2. Consider a class Student with roll, name and score as attributes. Support to take and display data is also there. One wants to works with array of Student objects. May collect data for array elements, display those. In case index goes out of bounds, exception is to be raised with suitable message.

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int size;

class student{
  static int rolls;
  int roll;
  string name;
  int score;
  public:
  student():roll(-1),name(""),score(-1){}
  void input(string n,int s){name=n;score=s;roll=rolls;rolls++;}
  friend ostream& operator<<(ostream& o,const student& s){
    if(s.roll!=-1)
      o<<"Roll: "<<s.roll<<" "<<"Name: "<<s.name<<" "<<"Score:
"<<s.score<<endl;
    return o;
  }
};
int student::rolls = 1;

void input(student* stu) throw(const char*){
   int s;
   cout<<"Enter number of students to enter: ";
   cin>>s;
   if(s>size)
     throw "Access out of bounds for input";

   for(int i=0;i<s;i++)
      stu[i].input("name"+to_string(i),(90+i));
}

void display_all(student* stu) throw() {
    for(int i=0;i<size;i++)
        cout<<stu[i];
}

int main() {
  cout<<"Enter size: ";
  cin>>size;
  student students[size];
  try{
    input(students);
  }catch(const char* s){
    cout<<s<<endl;
  }
  display_all(students);
}
```

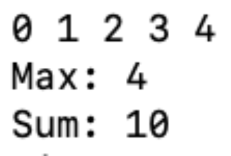


```
Enter size: 3
Enter number of students to enter: 4
Access out of bounds for input
```

3. Implement a class template for 1D array. Elements may be any basic data type. Provision to find maximum element, sum of the elements must be there.

```
#include <iostream>
using namespace std;

template<class X>
class Array;
template<class X>
ostream& operator<<(ostream&,const Array<X>&);

template<class X>
class Array{
  int size;
  X *arr;
  public:
  Array(int size):size(size){arr = new X[size];}

  int get_size() const {return size;}

  X& operator[] (int i) const {
    if(i<0 || i>size){
      cout <<"Index "<<i<<" is out of bounds"<<endl;
      exit(0);
    }
    return arr[i];
  }

  friend ostream& operator<< <>(ostream& o,const Array<X>& a){
     for(int i=0;i<a.size;i++)
```

```
        o<<a[i]<<" ";
     o<<endl;
     return o;
  }

  X max_element() const {
    X max;
    for(int i=0;i<size;i++)
       if(arr[i]>max)
         max = arr[i];
    return max;
  }

  X sum_of_elements() const {
     X sum;
     for(int i=0;i<size;i++)
        sum += arr[i];
     return sum;
  }
};

int main()
{
 Array<float> arr(5);
 for(int i=0;i<5;i++)
     arr[i] = i;

 cout<<arr;
 cout<<"Max: "<<arr.max_element()<<endl;
 cout<<"Sum: "<<arr.sum_of_elements()<<endl;
}
```



```
0 1 2 3 4
Max: 4
Sum: 10
```