Software Engineering: CS20006/CS20202 Assignment – 2: Classes and Object-oriented Programming

Rohit Ranjan — 20CS30066

14th February 2022

1 Question 1.

```
//include directives
#include <iostream>

using namespace std;

//utility function that uses binary search to find element in a given array
//arr - pointer to array to search in
//left - left index of arr to start search from
//right - right index of arr to search till
//key - the element being searched for
//returns -1 if element not found or index in arr
int binary_search(int *arr, int left, int right, int key)
{
    if (right >= left)
    {
        int mid = left + (right - left) / 2;
        if (arr[mid] == key)
            return mid;
        if (arr[mid] > key)
            return binary_search(arr, left, mid - 1, key);
        return binary_search(arr, mid + 1, right, key);
    }
    return -1;
}

//class definition
class PrimalityTest
{
    static PrimalityTest* _myTest;
    int nStored;// number of Stored Primes
    int *primes;// pointer to Buffer
    int bufsize;// size of the buffer
    PrimalityTest(int bufsize);
    ~PrimalityTest();

    public:

    static PrimalityTest& newTest(int bufsize = 100);
    void test(int n);
};

//constructor function to initialise object
//buffersize - size of the buffer
PrimalityTest::PrimalityTest(int buffersize)
{
    bufsize = buffersize;
    primes = new int[bufsize];
    for(int i=1;i<bufsize;i++)
    {
        primes[i] = 0;
    }
    primes[0]=2;
    nStored = 1;
}
```

```
//destructor
PrimalityTest::~PrimalityTest()
{
    delete(primes);
    free(_myTest);
}

//method to create object if not created yet and to resize buffer array if buffer size is changed
//bufsize - size of the buffer
//returns reference to only object of PrimalityTest class
PrimalityTest& PrimalityTest::newTest(int bufsize)
{
    if(_myTest == NULL)
    {
        _myTest= new PrimalityTest(bufsize);
    }
    else if(_myTest->bufsize!=bufsize)
    {
        if(bufsize<_myTest->nStored)
        {
            cout<<"Buffer␣Overflow!"<<endl;
            exit(0);
        }
        //new buffer array created
        int *new_primes = new int[bufsize];
        //copying elements to new buffer
        for(int i=0;i<bufsize;i++)
        {
            if(i<_myTest->nStored)
            {
                new_primes[i] = _myTest->primes[i];
            }
            else
            {
                new_primes[i] = 0;
            }
        }
        _myTest->primes = new_primes;
    }
    return *_myTest;
}

//method to test for primality
//n - integer being tested
//return - none, output is printed
void PrimalityTest::test(int n)
{
    //flag variable which turns 0 when element is surely non-prime
    int prime=1;

    //largest prime stored in buffer till now
    int last_prime_now=0;
    if(nStored!=0)
    {
        last_prime_now = primes[nStored-1];
    }

    //quick check to determine non-primality
    if(n%2==0 || n%3==0 || ((n%6)*(n%6))%6!=1)
    {
        prime=0;
    }
    //buffer is filled with atleast all primes <= sqrt(n)
    else if(last_prime_now*last_prime_now<=n)
    {
        for(int i =last_prime_now+1;i*i<=n;i++)
        {
            int flag=1;
            for(int j=0;j<nStored;j++)
            {
                if(i%primes[j]==0)
                {
                    flag=0;
                    break;
                }
            }
            if(flag==0)
            {
                continue;
            }
            else
            {
                if(nStored>=bufsize)
                    {
                        cout<<”Buffer␣Overflow!”<<endl;
                        exit(0);
                    }
                    else
                    {
                        primes[nStored] = i;
                        nStored++;
                    }
            }
        }
    }
    //if largest prime stored in buffer is greater than n then binary search for n
    else if(last_prime_now>=n)
    {
        if(binary_search(primes,0,nStored-1,n)==-1)
        {
            prime=0;
        }
        else
        {
            prime=2;
        }
    }

    //buffer contains all primes <=sqrt(n) and n must be divisible by atleast one of these if non-prime
    if(prime==1)
    {
        for(int j=0;j<nStored;j++)
        {
            if(n%primes[j]==0)
            {
                prime=0;
                break;
            }
        }
    }

    //printing required output using flag variable
    if(prime==0)
    {
        cout<<n<<”␣is␣not␣a␣prime␣number.”<<endl;
        return;
    }
    else
    {
        cout<<n<<”␣is␣a␣prime␣number.”<<endl;
        return;
    }
}

//initialising the static variable of class
PrimalityTest* PrimalityTest::_myTest = NULL;

int main()
{
    //test cases
    PrimalityTest::newTest().test(2958);
    PrimalityTest::newTest().test(823);
    PrimalityTest::newTest().test(83479);
    return 0;
}
```

The code file is also available here - Q1.cpp

2 Question 2.

```
//include statements
#include <iostream>
#include <list>
#include <vector>

using namespace std;

//class definition for Customer
class Customer
{
    //name of the customer
    string name;

    //unique id of customer
    int id;

    //NumCustomer stores the current number of instances of the class
    //it is static because counting is to be done by class
    static int NumCustomer;

    public:

    //constructor to initialise data members and assign id
    Customer(string name = ”NA”);

    //destructor to decrement current number of available instances of class
    ~Customer();

    //operator overloading for output operator
    friend ostream& operator << (ostream &os, const Customer & cust);

    //operator overloading for input operator
    friend istream& operator >> (istream &is, Customer & cust);
};

//class definition for ProductItem
class ProductItem
{
    //title of product
    string title;

    //unique id of the product
    int id;

    //NumProductItem stores the current number of instances of the class
    //it is static because counting is to be done by class
    static int NumProductItem;

    //price of product
    float price;

    //number of copies of product
    int copies;

    public:

    //constructor to initialise data members and assign id
    ProductItem(string title = ”NA”, float price = 0);

    //destructor to decrement current number of available instances of class
    ~ProductItem();

    //operator overloading for output operator
    friend ostream& operator << (ostream &os, const ProductItem & pi);

    //operator overloading for input operator
    friend istream& operator >> (istream &is, ProductItem & pi);

    //operator overloading for * operator which changes the number of copies of the product
    //a - new number of copies of the product
    //returns a reference to edited product
    ProductItem& operator* (int a);

    //operator overloading for = operator which acts as copy constructor
    //pi - object to copy data members into
    //returns a reference to edited product
    ProductItem& operator= (ProductItem & pi);
};

//class definition for Order
class Order
{
    //unique id of the order
    int id;

    //flag variable that holds 1 if order has been finalised and can no longer be edited
    int flag;

    //NumOrder stores the current number of instances of the class
    //it is static because counting is to be done by class
    static int NumOrder;

    //customer associated with the order
    Customer c;

    //vector of all products in the order
    vector<ProductItem> prods;

    public:

    //constructor to initialise data members and assign id
    Order(Customer &in_c);

    //destructor to decrement current number of available instances of class
    ~Order();

    //utility function that returns id of instance
    int getid();

    //utility function that returns flag of instance
    int getflag();

    //utility function that sets the flag status of the instance
    void setflag();

    //operator overloading for output operator
    friend ostream& operator << (ostream &os, const Order & o);

    //operator overloading for input operator
    friend istream& operator >> (istream &is, Order & o);

    //operator overloading for + operator which adds products to order
    //p - product to be added
    //returns a reference to edited order
    Order & operator+ (ProductItem &p);

    //operator overloading for = operator which acts as copy constructor
    //o - object to copy data members into
    //returns a reference to edited order
    Order & operator= (Order &o);
};

//class definition for ShoppingBasket
class ShoppingBasket
{
    //unique id of the basket
    int id;

    //NumBasket stores the current number of instances of the class
    //it is static because counting is to be done by class
    static int NumBasket;

    //customer associated with the basket
    Customer c;

    //list of all orders in the basket
    list<Order> orders;

    public:
```

```
    //constructor to initialise data members and assign id
    ShoppingBasket(Customer &in_c);

    //destructor to decrement current number of available instances of class
    ~ShoppingBasket();

    //operator overloading for output operator
    friend ostream& operator << (ostream &os, const ShoppingBasket & sb);

    //operator overloading for input operator
    friend istream& operator >> (istream &is, ShoppingBasket & sb);

    //operator overloading for + operator which adds order to basket
    //p - order to be added
    //returns a reference to edited basket
    ShoppingBasket & operator+ (Order &p);

    //operator overloading for - operator which removes order from basket
    //orderid - id of order to be removed
    //returns a reference to edited basket
    ShoppingBasket & operator- (int orderid);

    //operator overloading for = operator which acts as copy constructor
    //sb - object to copy data members into
    //returns a reference to edited basket
    ShoppingBasket & operator= (ShoppingBasket &sb);
};

Customer::Customer(string name)
{
    this->name = name;
    this->id = ++NumCustomer;
}

Customer::~Customer()
{
    NumCustomer--;
}

ostream& operator << (ostream &os, const Customer & cust)
{
    os << "Customer␣name:␣"<<cust.name<<"\nCustomer␣id:␣"<<cust.id;
    return os;
}

istream& operator >> (istream &is, Customer & cust)
{
    cout<<"Enter␣Customer␣Name:␣";
    is>>cust.name;
    return is;
}

ProductItem::ProductItem(string title, float price)
{
    this->title = title;
    this->id = ++NumProductItem;
    this->price = price;
    this->copies = 0;
}

ProductItem::~ProductItem()
{
    NumProductItem--;
}

ostream& operator << (ostream &os, const ProductItem & pi)
{
    os<<"Product␣Title:␣"<<pi.title<<"\nProduct␣id:␣"<<pi.id<<"\nProduct␣price:␣"<<pi.price<<"\nProduct␣co
    return os;
}

istream& operator >> (istream &is, ProductItem & pi)
{
    cout<<"Enter␣Product␣Title:␣";
    is>>pi.title;
    cout<<"Enter␣Product␣Price:␣";
    is>>pi.price;
    cout<<"Enter Product Copies: ";
    is>>pi.copies;
    return is;
}

ProductItem& ProductItem::operator*(int a)
{
    this->copies = a;
    return *this;
}

ProductItem& ProductItem::operator=(ProductItem & pi)
{
    pi.title = this->title;
    pi.price = this->price;
    pi.copies = this->copies;
    return pi;
}

Order::Order(Customer &in_c)
{
    this->id = ++NumOrder;
    this->c = in_c;
    this->flag = 0;
}

Order::~Order()
{
    NumOrder--;
}

ostream& operator << (ostream &os, const Order & o)
{
    os<<"Order id: "<<o.id<<"\nCustomer Details:\n"<<o.c<<"\nProduct List:";
    for(auto itr = o.prods.begin(); itr != o.prods.end(); itr++)
    {
        os<<"\n"<<*itr;
    }
    return os;
}

istream& operator >> (istream &is, Order & o)
{
    if(o.getflag()!=1)
    {
        cout<<"Enter Customer Details:"<<endl;
        is>>o.c;
        cout<<"Enter number of Products: ";
        int size;
        is>>size;
        for(int i=0;i<size;i++)
        {
            cout<<"Enter Product "<<i+1<<" details:\n";
            ProductItem temp;
            is>>temp;
            o.prods.push_back(temp);
        }
    }
    else
    {
        cout<<"Order is finalised and cannot be edited now!";
    }
}

Order& Order::operator+ (ProductItem &p)
{
    if(this->flag!=1)
    {
        this->prods.push_back(p);
        return *this;
    }
    else
    {
        cout<<"Order is finalised and cannot be edited now!";
    }
```

```
}

Order& Order::operator= (Order &o)
{
    if(this!=&o)
    {
        o.c = this->c;
        o.prods.clear();
        for(auto prod:this->prods)
        {
            o.prods.push_back(prod);
        }
    }
    return o;
}

int Order::getid()
{
    return this->id;
}

int Order::getflag()
{
    return this->flag;
}

void Order::setflag()
{
    this->flag=1;
}

ShoppingBasket::ShoppingBasket(Customer &in_c)
{
    this->id = ++NumBasket;
    this->c = in_c;
}

ShoppingBasket::~ShoppingBasket()
{
    NumBasket--;
}

ostream& operator << (ostream &os, const ShoppingBasket & sb)
{
    os<<"Shopping␣Basket␣id:␣"<<sb.id<<"\nCustomer␣details:\n"<<sb.c<<"\nOrders␣List:";
    for(auto itr = sb.orders.begin(); itr != sb.orders.end(); itr++)
    {
        os<<"\n"<<*itr;
    }
    return os;
}

istream& operator >> (istream &is, ShoppingBasket & sb)
{
    cout<<"Enter␣Customer␣Details:"<<endl;
    is>>sb.c;
    cout<<"Enter␣number␣of␣Orders:␣";
    int size;
    is>>size;
    for(int i=0;i<size;i++)
    {
        cout<<"Enter␣Object␣"<<i+1<<"␣details:\n";
        Order temp(sb.c);
        is>>temp;
        sb.orders.push_back(temp);
    }
}

ShoppingBasket& ShoppingBasket::operator+ (Order &p)
{
    this->orders.push_back(p);
    return *this;
}

ShoppingBasket& ShoppingBasket::operator- (int orderid)
{
```

```cpp
    for (auto itr = this->orders.begin(); itr != this->orders.end(); itr++)
    {
        if((*itr).getid() == orderid)
        {
            this->orders.erase(itr);
            break;
        }
    }
    return *this;
}

ShoppingBasket& ShoppingBasket::operator= (ShoppingBasket &sb)
{
    if(&sb!=this)
    {
        sb.NumBasket = this->NumBasket;
        sb.c = this->c;
        sb.orders.clear();
        for(auto order:this->orders)
        {
            sb.orders.push_back(order);
        }
    }
    return sb;
}

//initialising the static data members
int Customer::NumCustomer = 0 ;
int ProductItem::NumProductItem = 0;
int Order::NumOrder = 0;
int ShoppingBasket::NumBasket = 0;

int main()
{
    //create a customer
    Customer *c = new Customer("Nikhil");

    // create a product
    ProductItem *p = new ProductItem("Something");

    // create am Order
    Order *o = new Order(*c);

    // add 10 copies of p to order o
    Order &oref = *o;
    oref = oref + *p * 10;

    // create a shopping basket
    ShoppingBasket *s = new ShoppingBasket(*c);
    ShoppingBasket &shop = *s;
    shop = shop + oref;
    shop = shop - oref.getid();

    return 0;
}
```

The code file is also available here - Q2.cpp