Software Engineering : CS20006 / CS20202

Assignment 2 : Classes and Object - Oriented Programming

Name : Rohit Ranjan
Roll no. : 20CS30066

Q1.

```
// 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 start search from
// 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
// bufferesize - size of the buffer
PrimalityTest :: PrimalityTest(int bufferesize)
{
    bufsize = bufferesize;
    primes = new int[bufsize];
```

Rohit Ranjan
20CS30066

```
    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);
    }
```

Rohit
20CS30066

```
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
```

Rohit
20CS30066

```
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;
    int (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++)
```

Rohit
20CS30066

```
{
    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 m.
else if ( last_prime_now >= n)
{
```

20C Rohit
20CS30066

```
    if (binary-search(primes, 0, nStored-1, n) == -1)
    {
        prime = 0;
    }
    else
    {
        prime = 2;
    }
}

// buffer contains all primes <= sqrt(n)
// n must be divisible by atleast one of these to be
  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;
}
```

Rohit
20CS30066

```
else
{
    cout << n << " is a prime number." << endl;
    return ;
}
}

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

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

Rohit
20CS30066

Q2.

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

using namespace std;

//class definition for Customer
class Customer
{

    //name of customer
    string name;
    //unique id of customer
    int id;

    // Num Customer 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");

    // default destructor is used
    // ~ Customer();
```

Rohit
20CS30066

```
// operator overloadding for output operator
friend ~~ostr~~ ostream& operator << (ostream &os,
const Customer & cust);

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

};

// class definition for Product Item
class Product Item
{
    // title of product
    string title;

    // unique id of product
    int id;

    // NumProduct Item stores 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
    int copies;
```

Rohit
20CS30066

public:

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

// default destructor is used

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

// utility function that increases copies of instance
void incrementcopies (ProductItem & p);

// operator overloading for >> and <<
friend ostream & operator << (ostream & os, const
ProductItem & pi);
friend istream & operator >> (istream & is,
ProductItem & pi);

// operator overloading for * operator which multiplies
copies of product
// a - new number of copies if previously empty or
the multiplicand
// 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

Rohit
20053006

```
ProductItem& operator = (ProductItem& pi);
};

// class definition for Order
class Order
{
    //unique id of order
    int id;
    // flag variable that holds 1 if order is finalised
    int flag;
    // NumOrder stores current number of instances of class
    // It is static because counting is to be done by class
    static int NumOrder;
    // customer associated with order
    Customer c;
    // vector of all products in order
    vector<ProductItem> prods;

    public :

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

    // default destructor is used

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

Rohit
20CS30066

~~utf~~ //utility function that returns flag status
int getflag ();

// utility function that sets the flag status
~~void set flag~~
void setflag ();

friend ostream& operator <<(ostream& os, const Order& o);
friend istream& operator >> (istream& is, Order& o);

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

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

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

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

```
static int NumBasket;

// customer associated with the basket
Customer c;

list <Order> orders;

public:

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

friend ostream & operator << (ostream & os, const
ShoppingBasket & sb);
friend ~~o~~istream & operator >> (istream & is,
ShoppingBasket & sb);

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

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

// operator overloading for = operator which acts as
copy constructor
// sb - object to copy data members into
ShoppingBasket & operator= (ShoppingBasket & sb);
};
```

```
Customer::Customer(string name)
{
    this->name = name;
    this->id = ++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 = -1;
}

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

Rohit
20CS30066

```
void ProductItem::incrementcopies (ProductItem& p)
{
    this->copies += p.copies;
}

ostream & operator << (ostream &os, const
ProductItem & pi)
{
os << "Product Title:" << pi.title << "\n Product Id: "
<< pi.id << "\n Product price: " << pi.price <<
"\n Product Copies: " << pi.copies;
return os;
}

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

ProductItem & ProductItem::operator* (int a)
{ // temporary object created in free store
ProductItem * temp = new ProductItem();
```

Rohit
20CS30066

```
// The count is decreased since only a copy is created
   not a new instance
NumProductItem--;
this -> operator =(*temp);
if (this -> copies == -1)
{
   temp -> copies = a;
}
else
{
   temp -> copies *= a;
}
temp -> id = this -> id;
return *temp;
}

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;
}
```

Rohit
20CS30066

```
ostream& operator << (ostream &os, const Order &o)
{
    os << " Order id : " << o.id << "\n Customer Details:\n
    << o.c << "\n Product 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);
        }
    }
```

Rohit
20CS30066

```
else
{
    cout << "Order is finalised and cannot be
            edited now!";
}
return is;
}


Order& Order :: operator+ (ProductItem &p)
{
    int size = this->prods.size();
    int i=0;
    for (; i<size ; i++)
    {
        if ((this->prods[i]).getid() == p.getid())
        {
            break;
        }
    }
    if (this->flag!=1)
    {
        if (i==size)
        {
            this->prods.push_back(p);
        }
        else
        {
            this->prods[i].incrementcopies(p);
        }
    }
```

Rohit
20CS30066

```
else
{
    cout << " Order is finalised and cannot
      be edited now! " << endl;
}
return * this ;
}

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 ;
}
```

Rohit
20CS30066

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

ShoppingBasket :: ShoppingBasket(Customer &in_c)
{   this->id = ++NumBasket;
    this->c = in_c;
}
ostream& operator << (ostream &os, const ShoppingBasket &sb)
{
 os << "Shopping Basket id: " << sb.id << "\n Customer Details: \n"
   << sb.c << "\n Orders 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;
```

Rohit
20053006

```
    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);
    }
    return is;
}

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

Shopping Basket& Shopping Basket::operator -(int orderid)
{
    for (auto itr = this -> orders.begin(); itr !=
     this -> orders.end(); itr ++)
    {
        if ((*itr).getid() == orderid)
        {
            this -> orders.erase(itr);
            break;
        }
    }
    return * this;
}
```

Rohit
20CS30066

```
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()
{
    //test code
    return 0;
}
```

Rohit
20CS30066