Name: Sayantan Biswas Roll: 00191050157

- Assignment 3 :-

Question 13 :- (linked list, Node class)

```
class LL        /* Class Linked List*/
{  public:
       int data;
       LL *next;
};
class NODE : public LL     /* For basic NODE
                              activities*/
{  private:
       LL *head, *tail;
   public:
     NODE()                /*constructor*/
   {  head = NULL;
      tail = NULL;
   }
   void add(int n)         /* For adding new
                              node to the linked
                              list */, assign
                              next pointer to given
                              address*/
   {
   }
   LL* gethead()           /* Getting head */
   {  return head;
   }
static void display(LL *head)
{                          /* For displaying
                              the linked list*/
}
```

```
static void concatenate ( LL *a, LL *b)
{                                  /* For concatenating two
                                      linked list */
}

void front (int n)                 /* Adding new node at the
{                                     front */
}
void after( LL* a , int value)
{                                  /* Adding new node
                                      after a particular
}                                     location */

void del (LL* before_del)
{                                  /* For deleting a node */
}
};
```

Assignment 3:-

Question 14:- (item, item_list class)

```
class item          /* class item is for storing
{                      code, name, rate, quantity,
   char code[50];      price */
   char name[50];
   float rate;
   int quantity;
   float price;
   public:
        friend class item_list;
};

class item_list     /* item_list contains item code,
{                      name, rate, quantity for
   ~~char code_list~~     several items, ~~and~~ and
                          uniquely stores count */
   char code_list[1000][50];
   item[1000]
   static int count;
   public:
      void add_item();      /* for adding an item */
      void rate();          /* for rating an item */
      void issue_item();    /* To ~~recive~~ receive item */
      void item_details();  /* prints the
                               details of the
                               item ~~in item_list~~
                               using code */
};
```

```
int position_item(char arr[][50], char
                  char it_code[50], int limit)
{
        /* Function to get position of
           item in list. Return
           position if code is found */
}

bool search_item(char arr[][50], char it_code
                                  [50],
                 int limit)
{
        /* To check if the item_code
           is available or not */
        /* return true if code is
           found */
        /* return false if cod is
           not found */
}

int item_list :: count = 0;  /* initializing
                                static variable count. */
```

Assignment 3:-

Question 15:- (balance , transaction class)

```
#define ll long long int
                    // defining long long int as ll //

class balance ;      // forward declaration of class
                        balance //
class transaction
                    // This class holds all important
                       information like account
                       number, date of transaction,
                       amount and transaction
                          type //
{   ll acc_num;
    int date;
    ll amount;
    char tran_type;

  public:
    bool check_balance (balance &); // To check
                                       balance //

    void transaction_data (ll , int, ll , char,
                           balance &);
                    // Function for transaction and
                       update balance //
};

class balance       // For neccessary activities
                       like display balance,
{   ll acc_num;        update date , checking
    ll balance;           balance , member'
    int update_date;   " functions etc. //

                    // last update date //

  public:
    int ret_acc_num();  // returns account
                           number //
```

```
balance ( ll ac = 0 , ll b = 0, int date = 0)
{                                    // constructor //
    acc_num = ac;
    balance = b;
    update_date = date;
}
friend bool transaction :: check_balance
                              ( balance & ) ;

friend ~~bool~~ void transaction :: transaction_data
      ( ll , int , ll , char , balance & ) ;
                                    // Friend member functions
                                                        //
void check_balance( ) ;  // For checking balance//
~~void get_balance ( ) ; //~~
void get_balance ( ll , ll , int );
                                    // Function to take details
                                       of balance objects input//
} ;
bool check_account_number ( ll arr[] ,
                              int acc_num , int limit)
{
}                                   // Global function to check
                                       account number if it is used
                                       before //
```

```
int return_position(ll arr[], int acc_num,
                    int limit)
{
    // Global function that returns
       the position of an account number
       in account number list //
}
```

Assignment 4:-

Question 1 :- (account, account_list classes)
(without using friend functions)

```
#define ll long long int

class account        // contains account number and
                        baldnce //
{
  ll acc_num;
  ll balance;
  public:
    void getacc(ll);          // takes account number//
    void getbalance(ll);      // takes balance //
    ll return_acc_num();      // returns account
                                 number //
    ll return_balance();      // returns balance //
};

class account_list    // Account list containing
                         10000 account objects //
{
  account arr[10000];
  ll acc_num_list[10000];
```

```
public:
  static int count;  // Counts total number of
                        accounts //

  void add_account();  // Adds account in account
                          list //

  bool find_account();  // Finds account from
                           account list //

  void display();  // display all account data //

  friend class withdraw;

};

int account_list :: count=0 ;  // initilizing
                                  static int count //

class withdraw  // contains account number
                   (must exist) and amount
                   (will not exceed balance
                    amount of corresponding
                    account) //
{ ll acc_number;
  ll amount;

  public:
    void getwithdraw(account_list &);

};
                  // For withdrawing a
                     certain amount of
                     money //
```

```
int main()
{
    .
    .
    .
    withdraw w;          // withdraw object //
    cout<<"\n Withdrawing money \n";
    w.getwithdraw(a);
    return 0;
}
```

Assignment 4 :- Question 6 :-

Assignment 5 :- Question 1 :-

```
class SUBJECT;
class STUDENT;
struct choice
{
    STUDENT *std;
    SUBJECT *sub;
    choice *next_sub, *next_std;
    choice(STUDENT &st, SUBJECT &sb): std(&st),
        sub(&sb), next_std('\0'),
        next_sub('\0') { }
};

class STUDENT   // Contains number of students,
                   roll (unique), name etc. //
{ char name[31], ph_no[11];
  choice *sub_list;  // Stores location of a
                        choice object //
public:
                                  constant
    const uint roll;  // non-ve roll //
    STUDENT(uint, const char*, ~~const~~
            const char* = "--NOTSET--");
            // ph_no default --NOTSET--//
```

```
void changePh(const char *ph) { strcpy(ph_no, ph); }
                                    // changing ph_no. //

void changeName(const char *nm) { strcpy(name, nm); }
                                    // changing name of a
                                       student //

void print();        // prints students' name, roll //

void printDetails(); // prints full details of
                        student //

bool study(SUBJECT &); // Tells if a student
                          studies a particular
                          subject //

void ~~list~~ subject_list(); // Lists of subject that
                                 has been chosen //

void choose(SUBJECT &); // For choosing a
                           subject //

};
```

(11)

```
class SUBJECT    // Contains subject code, name
                    for each subject //
{  char name[31];
   choice *sid_list;  // Stores location of a
                         choice object //

  public:
   char sub_code;
   SUBJECT(char, const char* = "--NOTSET--");

   void changeName(const char*); // To change
                                    name of a subject //
  void list_students(); // Gives list of students
                           who studies a
                           particular subject //
  bool study_(STUDENT &); // Tells if a student
                             studies a particular
                             subject //
  void print();   // Prints subject code and
                     subject name //

   friend void STUDENT::choose(SUBJECT &);
                  // Friend function to take
                     arguments from class STUDENT
                     and choosing a subject //

};
```

Assignment 5 :-

Question 2 :-

```
class Book        // Contains information of book-id, number
{                    of copy of each books, their price,
                     title of the book, author's name,
                     publisher //
  protected:
    int book-id , num=0, price ;
    string title, author publisher ;
    int serial_number[ ] ;
  public:
    void get_data ( )
    {
        // stores every book details(like
           price, title, author's name;
           publisher//
    }

    bool check (int p)
    {
        // Checks if a book has same serial
        number in serial_number[ ] array.
        Returns true if that copy of the
        book already present in the library.
        If not then returns false. //
    }
    void add (int p)
    {
        // Adds a copy of a book by
           increasing the serial number //
    }
    void del (int p)
    {
        // If serial number p exists then
           changes its value to previous
           value and deletes a copy of a
           book //
    }
```

```
    friend class library;
};

class Library        // For storing books into the
{ protected:                  library //
    Book list [ ];
static int count ;

    public:
bool empty()
{       // checks if the book list is empty //
}

bool full()
{       // checks if the book list is full //
}

void new_book (int book_id)
{       // Adds a new book in the list of
           books and stores in the library //
}       // It calls library :: full() to check
           if any free space is left or not //

int position (int book_id)
{       // Returns position of a book from the
           book list //
}
```

```
bool present(int book_id, int serial_number)
{
    // Calls position[book_id] to check if
    any book is present according to its given
    book_id. Then calls Book::check() to
    check if given serial number of that
    book is present or not. If both conditions
    satisfy, then returns true. //
}

void add_copy(int book_id, int serial_number)
{
    // Adds a copy of book by given serial
       number to the book list //
}

void delete_copy(int book_id, int serial_number)
{
    // Deletes serial number of a book from the
       book list //
}
};
class member        // Contains all informations
                       about members (students and
                                      faculty) //
{
  protected:
    int id; ~~issue issue=0;~~
    ~~static~~ int issue;
    string name, address, email;
  public:
    void get_data()
    {
      // Stores informations about member
         for issuing or returning books //
    }
```

```
    friend class student;
    friend class faculty;
};

class student          // Contains information
    : public member    about students //
{
  protected:
    member arr_students[ ];
    int limit = 0;

  public:

  bool full( )
  {
      // Checks if a student has reached
         book issuing limit (2) //
  }

   void add_student( )
  {
      // Adds a student's information to
           arr_students[ ] //

  }

  int check_student(int id)
  {
      // If there is any student with given
         member id, then returns the position
         of that student from arr_students[] //

  }
};
```

```
class faculty: public member
                                    // Contains informations
{  protected:                          about faculty
                                       members //
   member arr_faculty[];
   int limit = 0;
  public:
     bool full()
     {   // checks if a faculty member has
         reached book issuing limit (10) //
     }
     void add_faculty()
     {   // Adds a faculty member's information
         to arr_faculty[] //
     }
     int check_faculty(int id)
     {   // If there is any faculty member
         with given member id, then returns
         the position of that faculty member
         from arr_faculty[] //
     }
};
class Transaction          // Contains trasaction
                              information of a book
{  private:                   when issuing or returning
                                by members //
int member_id, book_id, serial_number;
char member_type, transaction_type;
```

```
public:
    void get_data()
{       // stores every transaction details //

    }.
};
```

Assignment 5:~

Question 3:~

```
Class Employee      // Contains unique emp_id, name,
{                      designation and basic_pay //
  protected:
  int emp_id;
  float basic_pay;
  char name[31];
  char designation[50];
 public:
   Employee()   // Constructor //
   {
              // Tells if an employee is ~~contract~~
                 permanent or contractual
                 and also the information for
                 them //
   }

   float basic_pay()
   {
                 // Returns basic pay of an
                    employee through employee
                    id //
   }

   virtual calculate_salary() = 0
                 // Calculate salary for an
                 employee according to the
                 formula given in the
                 question //
};
```

```
class Permanent : public employee
{   public:
        calculate_salary()
        {       // Calculates salary for an
                employee according to the formula
                given ~~int~~ in the question,
                after calling Employee::basic_pay()
                                                //
        }
};
class Contractual : public Employee
    protected:
{   float allowance;  // Specially for contractual
                          Employee //
    public:
    Contractual()    // Constructor //
    {
            // initializes allowance //
    }
    calculate_salary()
    {       // Calculates salary for an
            employee according to the formula
            given in the question,
            after calling Employee::basic_pay()
                                                //
    }
};
```

Assignment 5 :-

Question 4 :-

```
class Player        // Contains information of
{ protected:                a cricketer (both batsman,
                            bowler) //
  char name[31], date_of_birth[8];
  public:
    player()  // Constructor //
    {
                // Initializes Stores name, date of
                   birth of a cricketer //
    }

    void display()      // Displays name, date of
    {                          birth of a cricketer //

    }
};
Class batsman : virtual public player
{ protected:
  int run;                  // Class for batsman's
                               information //
  float avg;
  public:
    batsman()   // Constructor //
    {
                // player() is called here //
    }
                // Stores total runs scored,
                   average runs scored //

  void display()
  {
                // display batsman's information
                                                  //
  }
};
```

```
class bowler : virtual public player
{   protected:                          // Class for bowler's
    int wicket ;                                information //
    float avg_economy ;

    public:
      bowler()      // Constructor //
    {
                    // player() is called here //
    }               // stores number of wickets taken,
                       average economy for a bowlar //

    void display()
    {       // display bowler's information //

    }
};
class allrounder : public bowler, public batsman
{   public ;                            // Class for allrounder.
                                           Bowler, Batsman both
    void display()                         arguments are
    {                                      included //.

    }
};
```