Objective:

- It will help in understanding the purpose of constructor and destructor
- It will help you understand and thoroughly practice the object passing by value or by references.
- Last part of it will also help you understand the issues related to shallow copy and method to deal with such issues.

Task-1:

Your Quiz-4: You are a programmer for the Home Software Company. You have been assigned to develop a class named 'BankAccount' that models the basic workings of a bank account. The class should perform the following tasks:

- Save the account balance.
- Save the Email-ID of the Account Holder
- Save the number of transactions performed on the account.
- Allow deposits to be made to the account.
- Allow withdrawals to be taken from the account.
- Calculate interest for the period.
- Report the current account balance at any time.
- Report the current number of transactions at any time.

In order to provide the above said features in *BankAccount* class, we need to provide following data members and member function:

Data Members:

balance	A double that holds the current account balance.
interestRate	A double that holds the interest rate for the period.
interest	A double that holds the interest earned for the current period.
transactions	An integer that holds the current number of transactions.
email	A character array of size 100, which hold email of account holder.

Member Function

Setter/getter function	Provide setter/getter functions for the data members with appropriate names and prototypes.
makeDeposit	Takes a double argument, which is the amount of the deposit. This argument is added to balance.
withdraw	Takes a double argument, which is the amount of the withdrawal. This value is subtracted from the balance, unless the withdrawal amount is greater than the balance. If this happens, the function reports an error.
calcInterest	Takes no arguments. This function calculates the amount of interest for the current period, stores this value in the interest member, and then adds it to the balance member.

Note: Apply all the validation checks like; the account balance must never become less than zero. Take care of transaction counter, it must also be >= 0. The transaction data member should be incremented only on a successful transaction.

After implementing the *BankAccount* class, create its objects, put values in them and check/validate the functionality of the provided member functions.

Quiz Solution in the absence of constructor

```
class BankAccount
{
    double balance;
    double interestRate;
    double interest;
    int transactions;
    char email[100];
    void incrementTransaction()
    {
        transactions++;
    }

public:
    double getBalanace()
    {
        return balance;
    }
    void setBalanace(double amount)
    {
        if (amount>0)
            balance = amount;
    }
    double getInterestRate()
    {
        return interestRate;
    }
    void setInterestRate(double rate)
    {
        interestRate = rate;
    }
    double getInterest()
    {
        return interest;
    }
    void setInterest(double amount)
    {
        interest = amount;
    }
    void setTransactions(int count)
    {
        transactions = count;
    }
    int getTransactions()
    {
        return transactions;
    }
    const char * getEmail()
    {
        return email;
    }
    void setEmail( char * em)
    {
        strcpy(email, em);
    }
    void makeDeposit(double amount)
    {
        balance = balance + amount;
        incrementTransaction();
    }
    bool withdraw(double amount)
    {
        if (amount > balance)
            return false;
        balance = balance - amount;
        incrementTransaction();
        return false;
    }
    void calcInterest()
    {
        interest = interestRate / 100 *
balance;
        makeDeposit(interest);
    }
};
```

```
int main()
{
    BankAccount ba;
    //Lot of issues in our BankAccount class because of
    //the absence of constructor
    //e.g transaction seter must never be exposed to client
    //balance must be changed through makeDeposit
    ba.setBalanace(100);
    ba.setEmail("ahmed@pucit.edu.pk");
    ba.setInterestRate(10);
    ba.setTransactions(1);
    return 1;
}
```

Mush Safer in the presence of Constructor
It is not necessary to provide getter/setter for each data member

```
class BankAccount
{
    double balance;
    double interestRate;
    double interest;
    int transactions;
    char email[100];
    void incrementTransaction()
    {
        transactions++;
    }
public:
    BankAccount()
    {
        balance = 0;
        interestRate = 0;
        interest = 0;
        transactions = 0;
        strcpy(email, "");
    }
    double getBalanace()
    {
        return balance;
    }
    double getInterestRate()
    {
        return interestRate;
    }
    void setInterestRate(double rate)
    {
        interestRate = rate;
    }
    double getInterest()
    {
        return interest;
    }
    int getTransactions()
    {
        return transactions;
    }
    const char * getEmail()
    {
        return email;
    }
    void setEmail( char * em)
    {
        strcpy(email, em);
    }
    void makeDeposit(double amount)
    {
        balance = balance + amount;
        incrementTransaction();
    }
    bool withdraw(double amount)
    {
        if (amount > balance)
            return false;
        balance = balance - amount;
        incrementTransaction();
        return false;
    }
    void calcInterest()
    {
        interest = interestRate / 100 *
balance;
        makeDeposit(interest);
    }
};
```

```
int main()
{
    BankAccount ba;
    cout<<ba.getBalanace();//no garbage and no need for client to
                           //call setter: object is preperly initialized
    ba.setInterestRate(10);
    ba.makeDeposit(100);
    ba.calcInterest();
    cout<<"\n"<<ba.getBalanace();
    cout<<"\n"<<ba.getTransactions();
    return 1;
}
```

Task-2:
Considering the way we solved Task-1, redo the Practice file-5 questions and must show it to TA/me.

Task-3: Array ADT

The Array ADT that we discussed today.

Private Members:

- int *data; /* pointer to an array of integers
- int capcity; /* size/capacity of array pointed by data
- int isValidIndex(int index)
  return 1 if index is within bounds otherwise 0.

Public Members:

The class 'Array' should support the following operations

- Array(int cap = 5);
  Sets 'cap' to 'capacity' and initializes rest of the data members accordingly.
  If user sends any invalid value then sets the cap to zero.

- ~Array()
  Free the dynamically allocated memory.

- int & getSet(int index);
  insert value at given index of array.

- int getCapacity()
  returns the size of array.

- void reSize (int newcapacity)
  resize the array to new capacity. Make sure that elements in old array should be preserved in the new array if possible.

You should look at the following function after lecture No. 8.

- Array (const Array &)
  makes a deep copy of the received object.

Task-4:
Design an ADT 'Matrix' whose objects should be able to store a matrix of integers. Matrix ADT should also perform specified operations listed below related to matrices.

Data Members:

- int * * data; /* pointer to an array of pointers whose each location points to an array of integers */
- int row; // number of rows in matrix
- int col; // number of columns matrix

Supported Operations:

- Matrix ();
  Set row and col to 0.
- Matrix (int r, int c);
  Set the r to row and c to col and creates matrix structure appropriately. If user sends invalid value in r or c or both then set them to 0.
- ~Matrix ();
  Free the dynamically allocated memory.
- int& at(int r, int c);
  For setting or getting some value at a particular location of matrix
- Matrix Transpose ();
  *Find Transpose of the *this object(calling object) and returns the transpose in a new Matrix object by value.*
- Matrix add (Matrix & m2);
  Add this(calling object) and m2 object and return the result in a new Matrix object.
- void reSize (int newrow, int newcol);
  resize the matrix according to new row and column. Make sure that the old matrix elements should be preserved in the new resized matrix if possible.

  Do the following tasks after Lecture No. 8.
- Matrix (const Matrix &)
  The sizzling copy ctor ☺

Task-5:
You are to design an application, which will store sayings/quotations of different renowned philosophers/writers etc.

This application should support the following operations.

- Let the user search the quotation on the basis of different keywords.
- Let the user search the quotation on the basis of author/writer name.
- It allows the user to add quotes along with author name (if author is not know then name it as anonymous).

For the above application you will design an ADT 'Quotation' with the members listed below:

Data Members:

- char * * quote; /* pointer to an array of char pointers whose each location points to a quotation (array of characters) */
- int numOfQuotations;
- char * * author; /* pointer to an array of char pointers whose each location points to an author name(array of characters) So the quotation at quote[i] has an author at author[i] */
- int capacity; // size of array pointed by quote ***which should be resizable***

Supported Operations:

1. `Quotation (int = 10);`
   Receives the initial capacity of quote array
2. `void addQuotation(char * quote, char * author);`
3. `void displayAuthorWise (char *)`
   receives author name and display all the quotation by the author on console.
4. `void displayQuotation (char *)`
   display on console all those quotation(s) containing the received string/word.
5. `void removeQuotation (char *)`
   it displays all the quotations containing the received string/word and then ask the user for the quotation number on console which he wants to delete.
6. `~Quotation ();`
   You know what to do.

Do the following tasks after Lecture No. 8.

7. `Quotation (const Quotation &)`
   You know what to do.