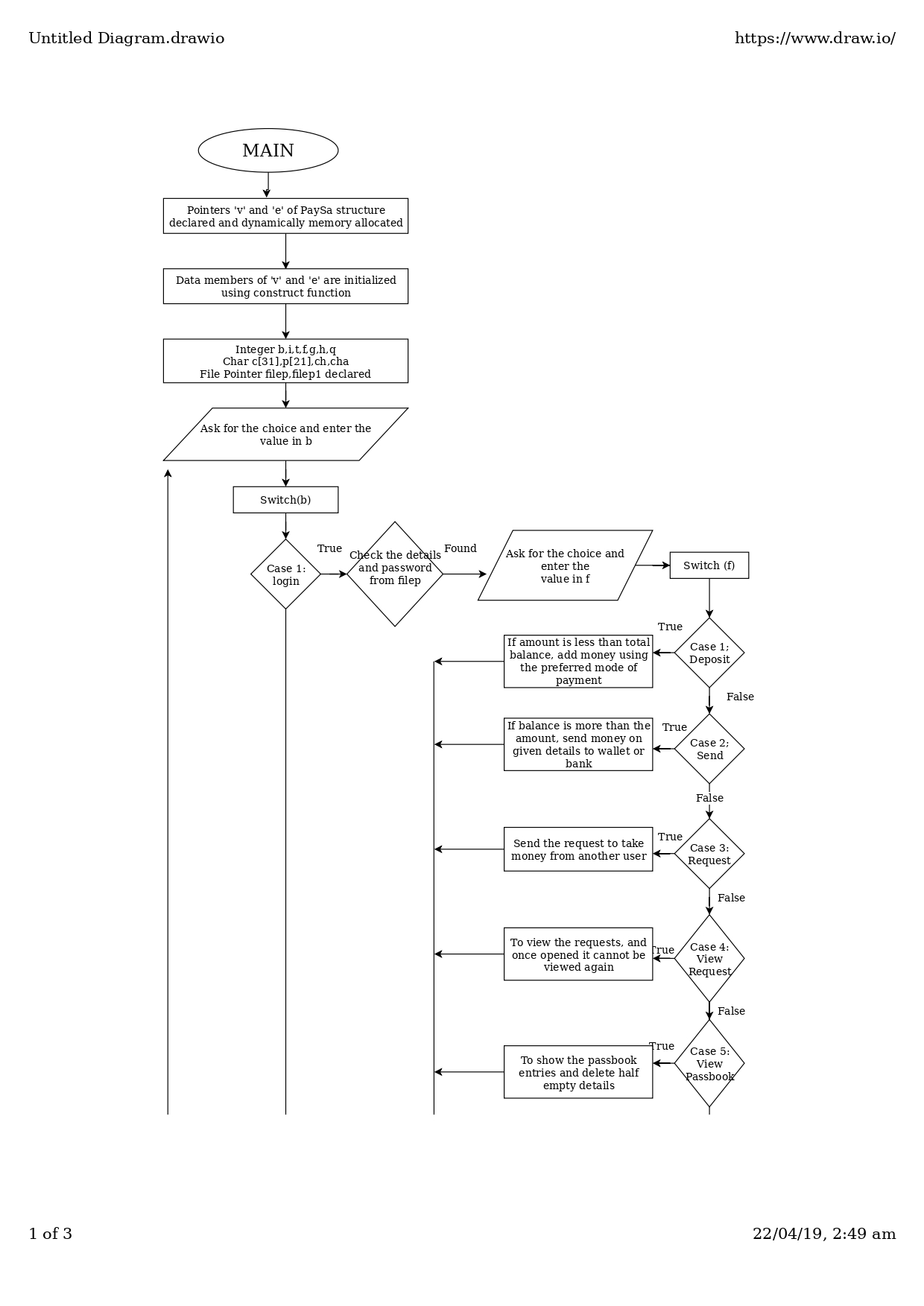
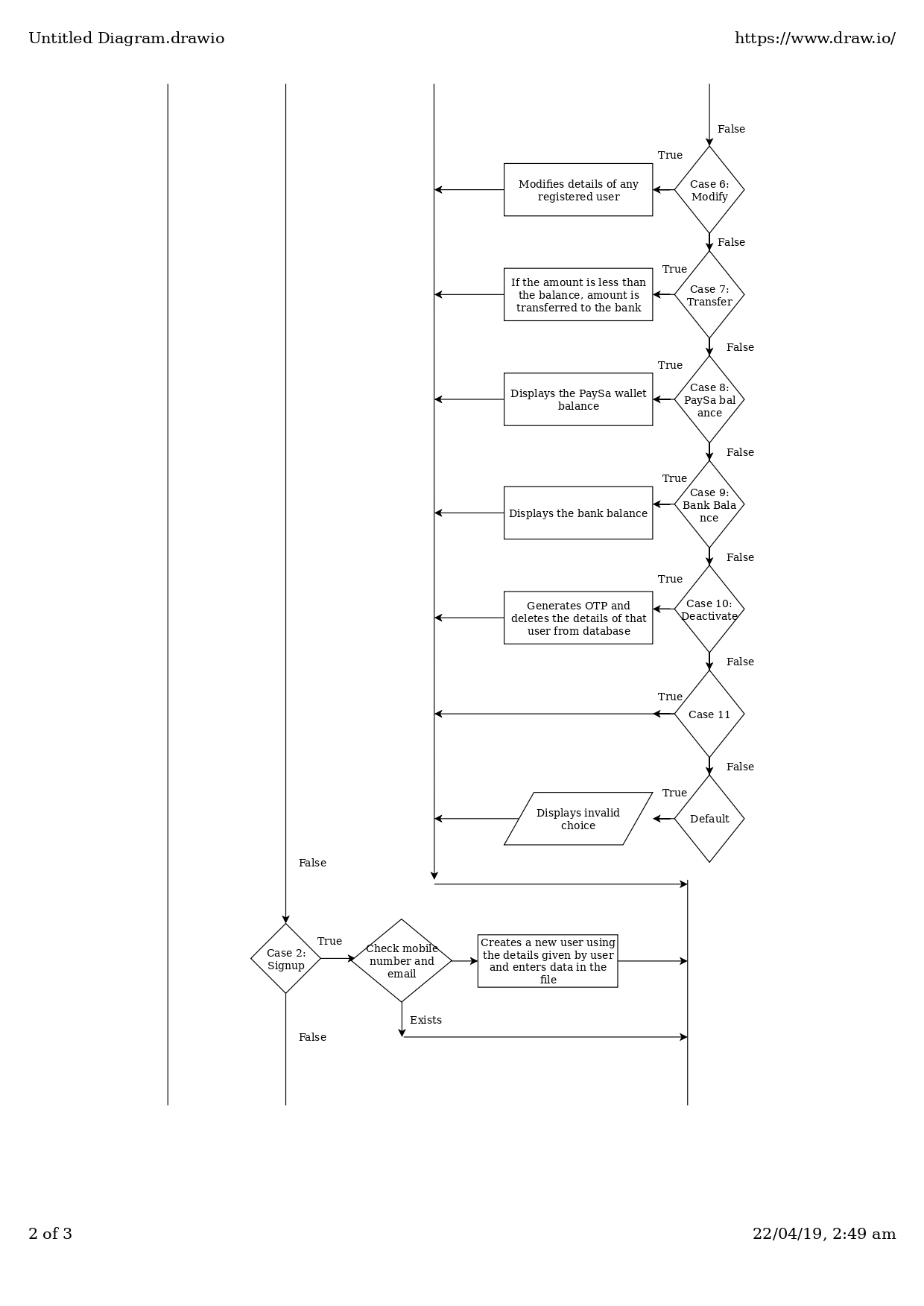
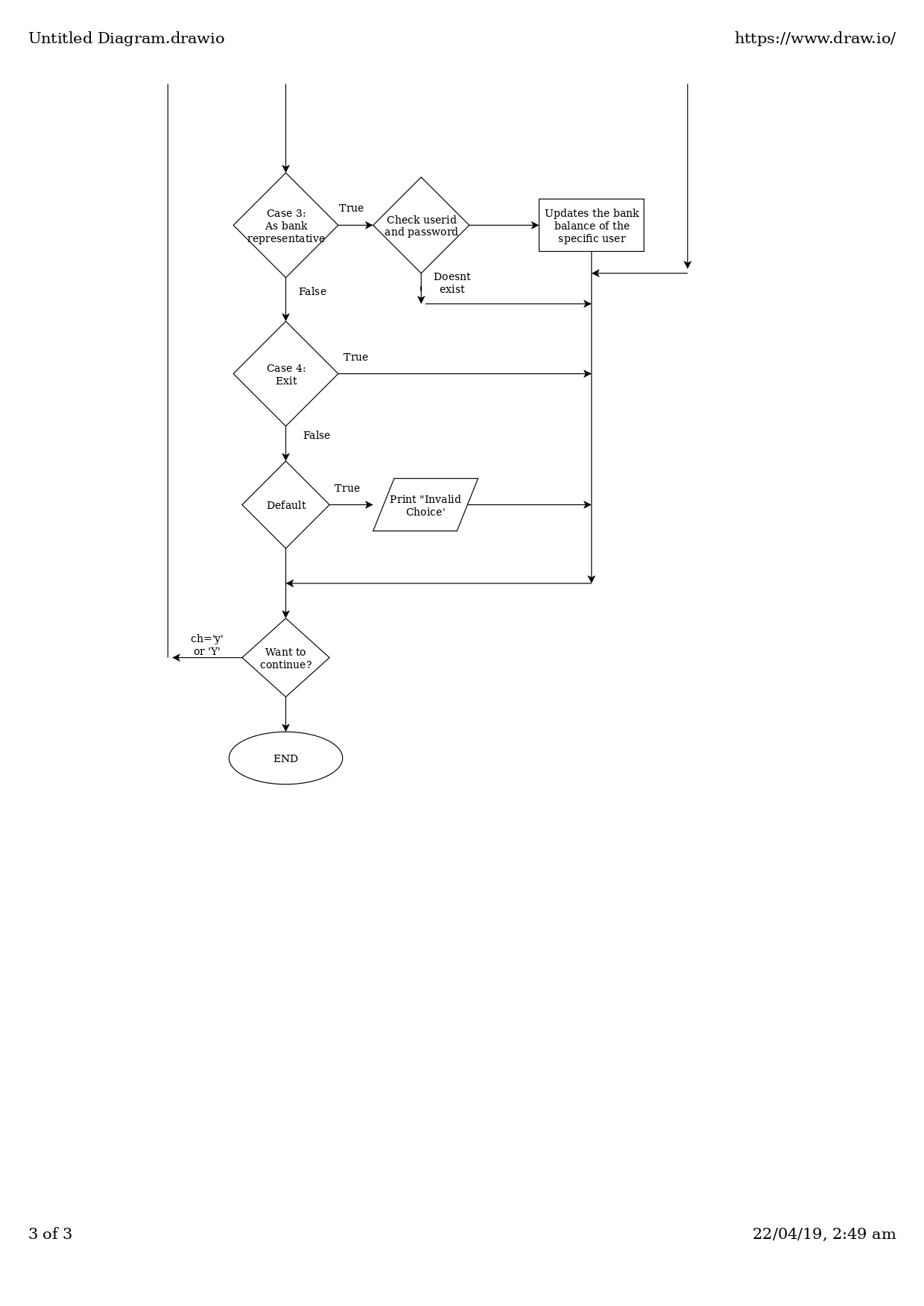
**PAY-SA**

This project aims at developing a digital payment platform and creating a payment gateway that offers comprehensive payment services for its customers. Pay-sa is a mobile commerce platform, offering its customers a digital wallet to store money and make quick payments. It requires users to create password protected accounts; that store information such as their name, mobile number, E-mail address, wallet balance, bank account details and net banking details; in order to carry out day to day transactions in a safe and convenient environment. The program stores a database of all its customers which contains all information stored by each account, with the application of file handling. The customer can edit most of this information, barring some invariable details, at any point in time. Due to lack of connectivity to the actual bank database at the current stage, the program creates a virtual bank database which requires the user to input his/her bank balance manually. The bank balance of the registered customers can be monitored and updated by a bank employee by logging in with the unique Bank User ID ( currently set as “bank1234” ) and password ( currently set as “1234bank” ). The customers can transfer money from their bank account to their digital wallet and vice-versa or from their digital wallet to any other customer’s digital wallet or bank account. Payments can be made conveniently using this program by transferring money from one person’s digital wallet to the intended recipient’s digital wallet or bank account, provided both the sender and the recipient are Pay-sa account holders. In addition to providing a platform for easy payment, Pay-sa also allows its users to request money/payment from other users by sending request messages. The program also stores a list of transactions made in the past ( up to a certain maximum limit ) by the customer, for future reference, in the form of a passbook. Each passbook entry includes the amount transferred and the details of the recipient ( name, mobile number, etc. ). Such a program can help minimize the use of cash in monetary transactions and in turn minimize the flow of black money in the market and promote transparency in finances.

**FLOW CHART**





**DATA STRUCTURE USED**

1. **CIRCULAR QUEUE :**

* A static circular queue ( of maximum size 10 ) has been used to store and display the payment requests received from other Pay-sa account holders.
* According to the logic of the program, the payment requests need to be viewed and deleted in chronological order ( i.e. the first request received will also be the first request to be viewed and deleted ). At any point of time, the request list is required to hold a maximum of 10 payment requests and a request once viewed is automatically deleted.
* In order to implement this logic, a First In First Out ( FIFO ) Data Structure was required.
* Considering the above mentioned conditions and the optimal use of memory space, a static circular queue ( of maximum size 10 ) has been used.

**DESCRIPTION OF IMPORTANT FUNCTIONS**

1. **genOTP() :**

* genOTP() takes PaySa structure pointer ( x ) as argument and generates an OTP
* It asks the user to enter a password. If this password matches completely with the user’s Pay-sa account password ( x->pass ), then a random number between 100 and 999 is generated and assigned as the value of x->OTP. The random number generated is also printed on the screen ( “Your OTP is …” ).
* In case the password entered does not match the user’s Pay-sa account password ( x->pass ), then the function genOTP() is called recursively ( with the same pointer x passed as argument ) till the correct password is entered.

1. **input() :**

* input() accepts a PaySa structure pointer ( x ) as its argument and is used for the creation of a new account.
* It asks and accepts input of all details required for the creation of an account such as name, mobile number, E-Mail ID, Date Of Birth, Netbanking User ID and password, and bank balance, and asks the user to create a password for his/her Pay-sa account.
* input() also places calls to checkmobile(), checkemail() and genOTP() in order to check the validity of various details entered by the user.
* If any of the above mentioned details are invalid then the input() fails.
* If account creation is successful, input() appends the pointer variable containing the details of the new account to the end of the file “PaySa.dat”.

1. **modify() :**

* modify() takes a PaySa structure pointer ( x ) as argument and offers a choice to the user to modify one of the following details :

1. Name
2. E-Mail ID
3. Password
4. Date Of Birth
5. Exit

* First the current data of the datamember to be modified is displayed and the function asks the user whether he/she wants to modify the current data or not. If yes, then the new value for the datamember is scanned from the console and previous value of the datamember stored in x is replaced with the new value.
* The function asks the user if he/she wants to modify other details as well. If yes, the switch-case statement is executed again, else the function terminates.

1. **pay() :**

* pay() takes a PaySa structure pointer ( x ) as argument and is used for making payments from current user’s Pay-sa wallet to another user’s Pay-sa wallet or bank account.
* It asks the user to input the amount to be paid. If this amount is greater than the user’s Pay-sa wallet balance ( x->balance ) the transaction fails and the function is terminated. If not then, it asks the user to input the date and the receiver’s mobile number.
* The function reads the file “PaySa.dat” account by account, if the mobile number entered by the user is present in the file, then depending on the user’s choice the receiver’s bank balance or Pay-sa wallet balance is incremented by the specific amount input by the user and both the current user’s and the receiver’s passbooks are updated with the required details. If the mobile number entered is not found in the file, the transaction fails.
* The function also updates the receiver’s account data in the file “PaySa.dat”.

1. **deposit() :**

* deposit() takes a PaySa structure pointer ( x ) as argument and is used for adding money to the user’s Pay-sa wallet from his/her bank account.
* It asks the user to input the amount to be paid. If this amount is greater than the user’s bank balance ( x->bank.bal ) the transaction fails and the function is terminated. If not then, it asks the user to choose his/her mode of payment :

1. Credit Card
2. Debit Card
3. Net Banking
4. Exit

* Depending on the user’s choice, the function asks for the card number and the CVV code and calls the genOTP() ( in case of credit card or debit card ) or asks for the username and password ( in case of net banking ).
* If the OTP/Password entered is found to be correct, the specified amount is deducted from the user’s bank balance ( x->bank.bal ) and added to his/her Pay-sa wallet balance ( x->balance ). Otherwise, the transaction fails.

1. **dequeue() :**

* dequeue() takes a PaySa structure pointer ( x ) as argument and is used to display and delete the request present at the front of the circular queue.
* First, the function checks if the queue is empty. If yes, then the message “No requests to be shown” is displayed on the screen and the function terminates.
* If not empty, the details of the request present at the front index ( x->zf ) of the circular queue are displayed and then this request is deleted.

1. **enqueue() :**

* enqueue() takes a PaySa structure pointer ( x ) as argument and is used to append a request to the rear of the circular request queue.
* The function checks if the queue is full. If yes, then the message "Cannot make request, receiver's inbox is currently full" is displayed and function terminates.
* If not full, the details of the request to be sent are scanned from the console and appended to the rear end ( x->zr ) of the circular request queue.

1. **request() :**

* request() takes a PaySa structure pointer ( x ) as argument and is used to make payment request from other users.
* First, the function asks the user for the receiver’s mobile number.
* Data is read from “PaySa.dat” record by record. If the mobile number entered by the user is present in the file, enqueue() is called and the record whose mobile number matches that entered by the user is passed as the argument.
* The function also updates the receiver’s data in the file.

1. **deactivate() :**

* deactivate() takes a PaySa structure pointer as argument and is used to deactivate the account of the current user.
* First the genOTP() is called. The function requires user to enter correct OTP.
* User’s Pay-sa wallet balance is added to his/her bank balance and then set to 0.
* The function deletes the user’s account details from the file “PaySa.dat”

1. **tbank() :**

* tbank() takes a PaySa structure pointer ( x ) as argument and is used to transfer money from the user’s Pay-sa wallet to his/her bank account.
* The function asks the user to input the transaction amount. If this amount is greater than the user’s Pay-sa wallet balance then the transaction fails.
* Otherwise , the specified amount is deducted from the user’s Pay-sa wallet balance ( x->balance ) and added to his/her bank balance ( x->bank.bal ) and the user’s passbook is appropriately updated.

1. **updatebankbalance() :**

* updatebankbalance() accepts no arguments and is used to monitor/update the bank balance of registered users ( made for use by bank representatives ).
* The function asks for the Bank User ID and Password from the user. If the details entered are incorrect, the function is terminated.
* Otherwise, the user ( bank representative ) is required to update the bank balance of each registered user.
* The function appropriately updates the details stored in the file “PaySa.dat”.

1. **main() :**

* Using switch-case statement ( placed inside a do-while loop ), the user is given a choice between the following :

1. Login
2. Sign up ( input(v) called )
3. Login a bank representative ( updatebankbalance() called )
4. Exit

* If the user selects Login, he/she is asked to enter E-Mail ID/Mobile number and Pay-sa account password. If the details entered do not match any record in the file “PaySa.dat”, the flow of control breaks out of switch-case statement. Otherwise the user is presented with a choice between the following (using switch-case statement placed inside a do-while loop ) :

1. Add money to your PaySa wallet ( deposit(v) called )
2. Pay or Send ( pay(v) called )
3. Request money ( request(v) called )
4. Check requests ( viewr(v) called )
5. Access Passbook ( viewpb(v) called )
6. Modify your registered information ( modify(v) called )
7. Transfer money from your PaySa wallet to your bank account ( tbank(v) called )
8. Check your current PaySa wallet balance ( v->balance displayed )
9. Check your current bank balance ( v->bank.bal displayed )
10. Deactivate your PaySa account ( deactivate(v) called )
11. Exit ( function terminated )

The file “PaySa.dat” is appropriately updated.

**ALGORITHMS FOR IMPORTANT FUNCTIONS**

1. **genOTP() :**

STEP 1 : Start

STEP 2 : Take a PaySa structure pointer ( x ) as argument.

STEP 3 : Declare a character array s of size 21

STEP 4 : Display “Enter your PaySa password” and take input of s from user

STEP 5 : Call cscreen()

STEP 6 : If the value of s matches completely with the value of x->pass, generate

a random number between 100 and 999, assign it as the value of x->OTP

and print x->OTP

Else, display “Wrong password” and call genOTP(x)

STEP 7 : Stop

1. **input() :**

STEP 1 : Start

STEP 2 : Accept a PaySa structure pointer as argument

STEP 3 : Declare a FILE pointer filep and open the file “PaySa.dat” in append

mode. If the file is not successfully opened then terminate the function

STEP 4 : Declare two integer variables p and i and set i=0

STEP 5 : Repeat the following steps till i<100 :

5.1 : Call cscreen()

5.2 : Print “Enter your mobile number” and scan x->mobile

5.3 : Call checkmobile(x)

5.4 : If value returned by checkmobile(x)!=0, break out of the loop

Else, increment value of i by 1

STEP 6 : Call checkmobile(x). If the value returned by checkmobile(x)=0 then

terminate the function.

STEP 7 : Set i=0 and repeat the following steps till i<100 :

7.1 : Call cscreen()

7.2 : Print “Enter your E-Mail ID” and scan x->email

7.3 : Call checkemail(x)

7.4 : If value returned by checkemail(x)!=0, break out of the loop

Else, increment value of i by 1

STEP 8 : Call checkemail(x). If the value returned by checkemail(x)=0 then

terminate the function.

STEP 9 : Set i=0 and repeat the following steps till i<100 :

9.1 : Call cscreen()

9.2 : Print “Create your PaySa password” and scan x->pass

9.3 : If length of x->pass=0, print “PaySa password is

necessary” and increment i by 1

Else, break out of the loop

STEP 10 : If length of x->pass=0, then terminate the function.

STEP 11 : Call genOTP(x)

STEP 12 : Set i=0 and repeat the following steps till i<100 :

12.1 : Call cscreen()

12.2 : Print “Enter the OTP or Press 1 to resend OTP” and scan p

12.3 : If p=1, call genOTP(x) and increment i by 1

Else if p!=x->OTP, print “Wrong OTP” and increment i by 1

Else, break out of the loop

STEP 13 : If p!=x->OTP, then terminate the function.

STEP 14 : Set i=0 and repeat the following steps till i<100 :

14.1 : Call cscreen()

14.2 : Print “Enter full name” and scan x->name.fname and x->

name.lname

14.3 : If length of x->name.fname=0, print “Your name is

necessary” and increment i by 1

Else, break out of the loop

STEP 15 : If length of x->name.fname=0, then terminate the function.

STEP 16 : Print “Enter your Date Of Birth” and scan x->DOB

STEP 17 : Set i=0 and repeat the following steps till i<100 :

17.1 : Call cscreen()

17.2 : Print “Enter net banking username” and scan x->bank.user

17.3 : If length of x->bank.user=0, print “Net banking username is

necessary” and increment i by 1

Else, break out of the loop

STEP 18 : If length of x->bank.user=0, then terminate the function.

STEP 19 : Set i=0 and repeat the following steps till i<100 :

19.1 : Call cscreen()

19.2 : Print “Enter net banking password” and take input of x->

bank.pass from the user

19.3 : If length of x->bank.pass=0, print “Net banking password is

necessary” and increment i by 1

Else, break out of the loop

STEP 20 : If length of x->bank.pass=0, then terminate the function.

STEP 21 : Print “Enter your bank balance” and scan x->bank.bal

STEP 22 : Append data of x to the end of file “PaySa.dat” and close the file.

STEP 23 : Print “Account created” and call display(x)

STEP 24 : Stop

1. **modify() :**

STEP 1 : Start

STEP 2 : Take a PaySa structure pointer ( x ) as argument

STEP 3 : Declare two integer variables a and j, two character variables b and c

and two character arrays p and s of size 21 each

STEP 4 : Execute the following steps once and then repeat till the value of b is

‘y’ or ‘Y’ :

4.1 : Call cscreen ()

4.2 : Print “Enter choice\n1.Change name\n2.Change E-Mail ID\n3.

Change Password\n4.Change Date Of Birth\n5.Exit” and scan a

4.3 : switch(a)

4.3.1 : For cases 1 to 4 :

4.3.1.1 : Print the value of the datamember that is to be

changed then print “Are you sure you want to

change the details?(y/n)” and scan c

4.3.1.2 : If c=’n’ or c=’N’, then break out of the switch-case

4.3.2 : For cases 1, 2 and 4 :

4.3.2.1 : Call cscreen()

4.3.2.2 : Scan new value of datamember to be modified

4.3.2.3 : Break out of the switch-case

4.3.3 : For case 3 :

4.3.3.1 : Set j=0 and repeat the following steps till j<100 :

4.3.3.1.1 : Call cscreen()

4.3.3.1.2 : Print “Enter new password” and scan s

4.3.3.1.3 : Print “Re-enter password” and scan p

4.3.3.1.4 : If p and s are same, copy s into x->pass

and break loop. Else, increment j by 1

4.3.3.2 : Break out of the switch-case

4.3.4 : Case 5, terminate the function

4.3.5 : Default, print “Invalid choice”

4.4 : Call cscreen()

4.5 : Print “Do you want to modify other details?” and scan b

STEP 5 : Stop

1. **pay() :**

STEP 1 : Start

STEP 2 : Take a PaySa structure pointer ( x ) as argument

STEP 3 : Declare two integers b and i and a long unsigned integer a and set i=0

STEP 4 : Call cscreen()

STEP 5 : Print "Enter amount to be paid" and scan a

STEP 6 : If a>x->balance, print "Transaction not possible (PaySa balance is

low)") and terminate the function

STEP 7 : Assign x->passbook[x->no].amount=a;

STEP 8 : Print "Enter date" and scan x->passbook[x->no].date

STEP 9 : Print “Enter mobile number of receiver" and scan x->passbook[x->no].

number

STEP 10 : Declare two FILE pointers filep and filep1 and open file “PaySa.dat”

in read mode using filep. If file does not open terminate the function

STEP 11 : Declare a PaySa structure pointer e and assign it memory dynamically

STEP 12 : Call construct(e)

STEP 13 : Create a file “temp.dat” using filep1. If the file is not created,

terminate the function

STEP 14 : Read the data of file “PaySa.dat” into e record by record and repeat

the following steps until the file ends :

14.1 : If value of x->passbook[x->no].number matches e->mobile :

14.1.1 : Call cscreen()

14.1.2 : Assign x->passbook[x->no].name.fname the value

of e->name.fname, x->passbook[x->no].name. lname the value of e->name.lname, e->passbook

[e->no].name.fname the value of x->name.fname, e->passbook[e->no].name.lname the value of x->

name.lname, e->passbook[e->no].amount=a, e->

passbook[e->no].func as "Received from", e->

passbook[e->no].date as x->passbook[x->no].date and e->passbook[e->no].number as x->mobile

14.1.3 : Print "Transfer to\n1.PaySa wallet\n2.Bank

account” and scan b

14.1.4 : switch(b)

case 1, add a to e->balance and break out

case 2, add a to e->bank.bal and break out

default, print "Invalid choice"

14.1.5 : Increment values of both e->no and i by 1

14.2 : Write e onto “temp.dat”

STEP 15 : Free the memory allocated to e and close both files

STEP 16 : Delete file “PaySa.dat” and rename “temp.dat” as “PaySa.dat”

STEP 17 : If i=0, print "Account not found" and terminate the function

STEP 18 : Assign x->passbook[x->no].func as "Sent to"

STEP 19 : Call cscreen() and print details of the payment

STEP 20 : Decrement a from x->balance and increment value of x->no by 1

STEP 21 : Stop

1. **deposit() :**

STEP 1 : Start

STEP 2 : Take a PaySa structure pointer ( x ) as argument

STEP 3 : Declare four integers b, s, i and t, a long unsigned integer a and a

character array sa of size 21. And set t=0.

STEP 4 : Call cscreen()

STEP 5 : Print "Enter amount to be added" and scan a

STEP 6 : If a>x->bank.bal, print "Transaction not possible (Bank balance is

low)") and terminate the function

STEP 7 : Assign x->passbook[x->no].amount=a;

STEP 8 : Print “Choose mode of payment\n1.Credit card\n2.Debit card\n3.Net

Banking\n4.Exit” and scan b

STEP 9 : Call cscreen()

STEP 10 : Switch(b)

10.1 : For case 1, assign x->passbook[x->no].name.fname as "Credit"

and x->passbook[x->no].name.lname as "card"

10.2 : For case 1 and 2,

10.2.1 : Print “Enter card number” and scan x->passbook[x->no

].number

10.2.2 : Print “Enter date” and scan x->passbook[x->no].date

10.2.3 : Print “Enter CVV/Security code” and scan sa

10.2.4 : If b=2, assign x->passbook[x->no].name.fname as

“Debit” and x->passbook[x->no].name.lname as "card”

10.2.5 : Call genOTP(x) and cscreen()

10.2.6 : Set i=0 and repeat the following steps till i<100 :

10.2.6.1 : Call cscreen()

10.2.6.2 : Print “Enter the OTP or Press 1 to resend

OTP” and scan s

10.2.6.3 : If s=1, call genOTP(x) and increment i by 1

Else if s!=x->OTP, print “Wrong OTP” and

increment i by 1

Else, break out of the loop

10.2.7 : If s=x->OTP, then increment x->balance by a

10.2.7.1 : If b=2, decrement x->bank.bal by a

10.2.7.2 : Set t=1

10.2.8 : Break out of swiych-case

10.3 : For case 3,

10.3.1 : Print “Enter name of bank” and scan x->passbook[x->

no].name.fname and x->passbook[x->no].name.lname

10.3.2 : Print “Enter Date” and scan x->passbook[x->no].date

10.3.3 : Print “Enter username” and scan x->passbook[x->no].

number

10.3.4 : If x->passbook[x->no].number is not same as x->bank.

user, print “Incorrect username” and break out

10.3.5 : Print “Enter Password” and scan sa

10.3.6 : If sa is not same as x->bank.pass, print “Incorrect

Password” and break out

10.3.7 : Increment x->balance by a and decrement x->bank.bal

by a and set t=1 and break out

10.4 : Case 4, terminate the function

10.5 : default, print “Invalid choice”

STEP 11 : Assign x->passbook[x->no].func as "Added from" and call cscreen()

STEP 12 : If t=0, print the details of the transaction and increment x->no by 1

Else, print “Transaction unsuccessful”

STEP 13 : Stop

1. **dequeue() :**

STEP 1 : Start

STEP 2 : Take a PaySa structure pointer ( x ) as argument

STEP 3 : If x->zf=-1, print “No requests to be shown” and return 0

STEP 4 : Print the details of the request

STEP 5 : If x->zf=x->zr, set x->zr=x->zf=-1

Else, x->zf=( ( x->zf+1 )%10 )

STEP 6 : Return 1

STEP 7 : Stop

1. **enqueue() :**

STEP 1 : Start

STEP 2 : Take a PaySa structure pointer ( x ) and a character array ( mobile[11]

)as argument

STEP 3 : Call cscreen()

STEP 4 : If ( x->zr=9 and x->zf=0 ) or ( x->zf=z->zr+1 ), return 0

Else if x->zf=-1, x->zf=0

STEP 5 : x->zf=( ( x->zf+1 )%10 )

STEP 6 : Print "Enter amount to be requested" and scan x->req[x->zr].amount

STEP 7 : Print "Enter date" and scan x->req[x->zr].date

STEP 8 : Assign x->req[x->zr].from as mobile

STEP 9 : Print "Payment requested for" and scan x->req[x->zr].reason

STEP 10 : Print "Request sent” and return 1

STEP 11 : Stop

1. **request() :**

STEP 1 : Start

STEP 2 : Take a PaySa structure pointer ( x ) as argument

STEP 3 : Declare two integers i and j, a character array m[11] and set i=0, j=1 STEP 4 : Print “Enter mobile number to request money from” and scan m

STEP 5 : Declare two FILE pointers filep, filep1 and open file “PaySa.dat” in read

mode using filep. If file does not open, terminate the function.

STEP 6 : Declare a PaySa structure pointer ( e ), assign it memory dynamically

and call construct(e)

STEP 7 : Create a file “temp.dat” using filep1. If file is not creater, end function

STEP 8 : Read the data from file “PaySa.dat” record by record into e and repeat

the following steps until the file ends :

* 1. : If m is same as e->mobile, call enqueue(e,x->mobile) and assign

value returned by enqueue() to j and set i=1

* 1. : Write e onto “temp.dat”

STEP 9 : Free the memory allocated to e and close both the files.

STEP 10 : Delete “PaySa.dat” and rename “temp.dat” as “PaySa.dat”

STEP 11 : If j=0, terminate function

If i=0, print “Account not found” and terminate function

STEP 12 : Stop

1. **deactivate() :**

STEP 1 : Start

STEP 2 : Take a PaySa structure pointer ( x )as argument

STEP 3 : Call genOTP(x)

STEP 4 : Declare two integer p and i

STEP 5 : Set i=0 and repeat the following steps till i<100 :

* 1. : Call cscreen()
  2. : Print “Enter OTP or press 1 to resend OTP” and scan p
  3. : If p=1, call genOTP() and increment i by 1

Else if p!=x->OTP, print “Wrong OTP” and increment i by 1

Else, break out of the loop

STEP 6 : If p!=x->OTP, terminate the function

STEP 7 : Declare a PaySa structure pointer ( e ) and assign it memory

Dynamically and call construct(e)

STEP 8 : Declare two FILE pointers filep and filep1. Open file “PaySa.dat” in

read mode using filep. If file does not open, terminate the function.

STEP 9 : Create file “temp.dat” using filep1. If file isn’t created, end function.

STEP 10 : Add x->balance to x->bank.balance and set x->balance=0

STEP 11 : Read data from “PaySa.dat” record by record into e and repeat the

following steps until the file ends :

* 1. : If x->mobile is not same as e->mobile, write e onto “temp.dat”

STEP 12 : Free memory allocated to e and close both the files.

STEP 13 : Delete “PaySa.dat” and rename “temp.dat” as “PaySa.dat”

STEP 14 : Call cscreen() and print “Account deactivated”

STEP 15 : Stop

1. **tbank() :**

STEP 1 : Start

STEP 2 : Take a PaySa structure pointer ( x ) as argument

STEP 3 : Declare a long unsigned integer a

STEP 4 : Call cscreen() and print “Enter amount to be transferred” and scan a

STEP 5 : If a>x->balance, print “Transaction not possible” and end function

STEP 6 : Set x->passbook[x->no].amount=a, x->passbook[x->no].number as x-

>mobile, x->passbook[x->no].name.fname as x->name.fname, x->

passbook[x->no].name.lname as x->name.lname, x->passbook[(x->no)

+1].name.fname as x->name.fname, x->passbook[(x->no)+1].name.

lname as x->name.lname), x->passbook[(x->no)+1].amount=a, x->

passbook[(x->no)+1].func as "Received from", x->passbook[(x->no)+1].

number as x->mobile and x->passbook[x->no].func as "Sent to"

STEP 7 : Print "Enter date" and scan x->passbook[x->no].date

STEP 8 : Set x->passbook[(x->no)+1].date as x->passbook[x->no].date,

STEP 9 : Call cscreen() and print the details of the transaction

STEP 10: Decrement x->balance by a, increment x->bank.bal by a and

increment x->no by 2

STEP 11 : Stop

1. **updatebankbalance() :**

STEP 1 : Start

STEP 2 : Declare a character array c of size 9

STEP 3 : Print “Enter User ID” and scan c

STEP 4 : If c is not “bank1234”, print “Wrong ID” and terminate the function

STEP 5 : Print “Enter password” and scan c

STEP 6 : If c is not “1234bank”, print “Wrong Password” and terminate function STEP 7 : Declare a PaySa structure pointer ( e ), assign it memory dynamically

and call construct(e)

STEP 8 : Declare two File pointer filep and filep1. Open “PaySa.dat” in read

mode using filep. If the file does not open, terminate the function.

STEP 9 : Create a file “temp.dat” using filep1. If file isn’t created, end function

STEP 10 : Read data from “PaySa.dat” into e record by record and repeat the

following steps until the end of file :

10.1 : Call cscreen () and print “Enter the updated bank balance of “

and the account details of the record and scan e->bank.bal

10.2 : Write e onto “temp.dat”

STEP 11 : Free the memory allocated to e and close both the files

STEP 12 : Delete “PaySa.dat” and rename “temp.dat” as “PaySa.dat”

STEP 13 : Call cscreen() and print “Updation complete”

STEP 14 : Stop

1. **main() :**

STEP 1 : Start

STEP 2 : Declare two PaySa structure pointers ( v and e ), assign them memory

dynamically and call construct(v) and construct(e)

STEP 3 : Declare seven integers b, i, t, f, g, h and q, two characters ch and cha,

two character arrays c[31] and p[21], and two FILE pointers filep,filep1

STEP 4 : Execute the followings steps once and then repeat them till ch is ’y’/‘Y’

* 1. : Call cscreen(), print “Enter your choice\n1.Login to PaySa\n2.Sign

Up\n3.Login as bank representative(Update bank balance of

registered users)\n4.Exit\n:” and scan b

* 1. : switch(b)

4.2.1 : Case 1,

4.2.1.1 : Call cscreen(), print “Enter E-Mail ID/Mobile

number” and scan c

4.2.1.2 : Open file “PaySa.dat” in read mode using filep. If

file does not open, terminate function

4.2.1.3 : Set t=0 and read the data of the file into v and

repeat the following steps until end of file :

4.2.3.1 : If c is same as v->email or v->mobile, t=1

4.2.1.4 : Close file

4.2.1.5 : If t=0, print “Account not found” and break out

4.2.1.6 : Print “Enter Password” and scan p

4.2.1.7 : If p is not same as v->pass, print “Wrong

password” and break out

4.2.1.8 : Execute the following steps once and then repeat

till cha=’y’/’Y’ :

4.2.1.8.1 : Call cscreen() and print "What do you

want to do?\n1. Add money to your

PaySa wallet\n2.Pay or Send\n3.

Request money\n4.Check requests\n5.

Access Passbook\n6.Modify your

registered information\n7.Transfer

money from your PaySa wallet to your

bank account\n8.Check PaySa wallet

balance\n9.Check bank balance\n10.

Deactivate your PaySa account\n11.

Exit\n" and scan f, call cscreen()

4.2.1.8.2 : switch(f)

Case 1, call deposit(v)

Case 2, call pay(v)

Case 3, call request(v)

Case 4, call viewr(v)

Case 5, call viewpb(v)

Case 6, call modify(v)

Case 7, call tbank(v)

Case 8, print v->balance

Case 9, print v->bank.bal

Case 10, call deactivate(v)

Case 11, terminate function

Default, print “Invalid choice”

4.2.1.8.3 : Open file “PaySa.dat” in read mode

using filep and create “temp.dat” using

filep1. If either file does not open,

terminate the function

4.2.1.8.4 : Read data from “PaySa.dat” into e and

repeat the following until file end:

4.2.1.8.4.1 : If v->mobile is same as e->

mobile, write v onto

“temp.dat”

Else, write e onto

“temp.dat”

4.2.1.8.5 : Close both files, delete “PaySa.dat” and

rename “temp.dat” as “PaySa.dat”

4.2.1.8.6 : Print “Do you want to continue using

your account?” and scan cha

4.2.2 : Case 2, call input(v)

4.2.3 : Case 3, call updatebankbalance()

4.2.4 : Case 4, terminate function

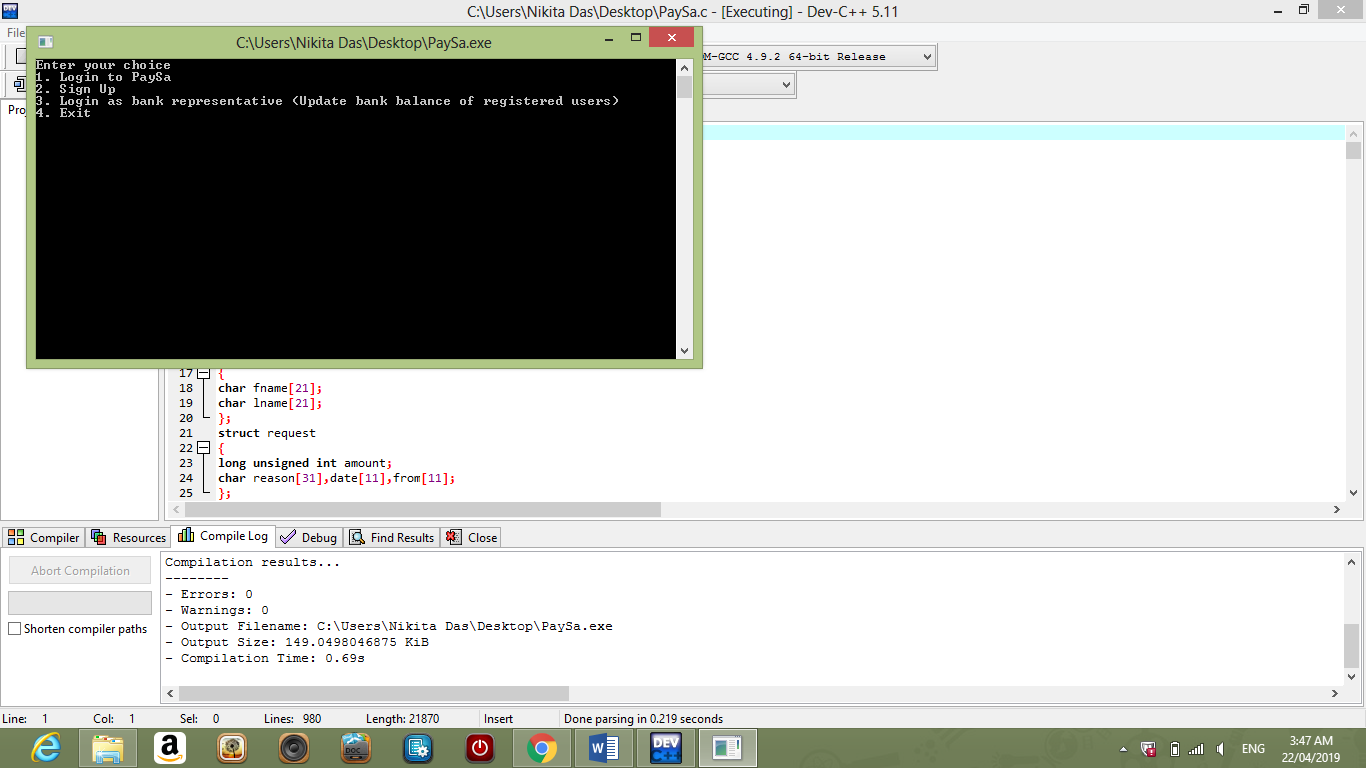
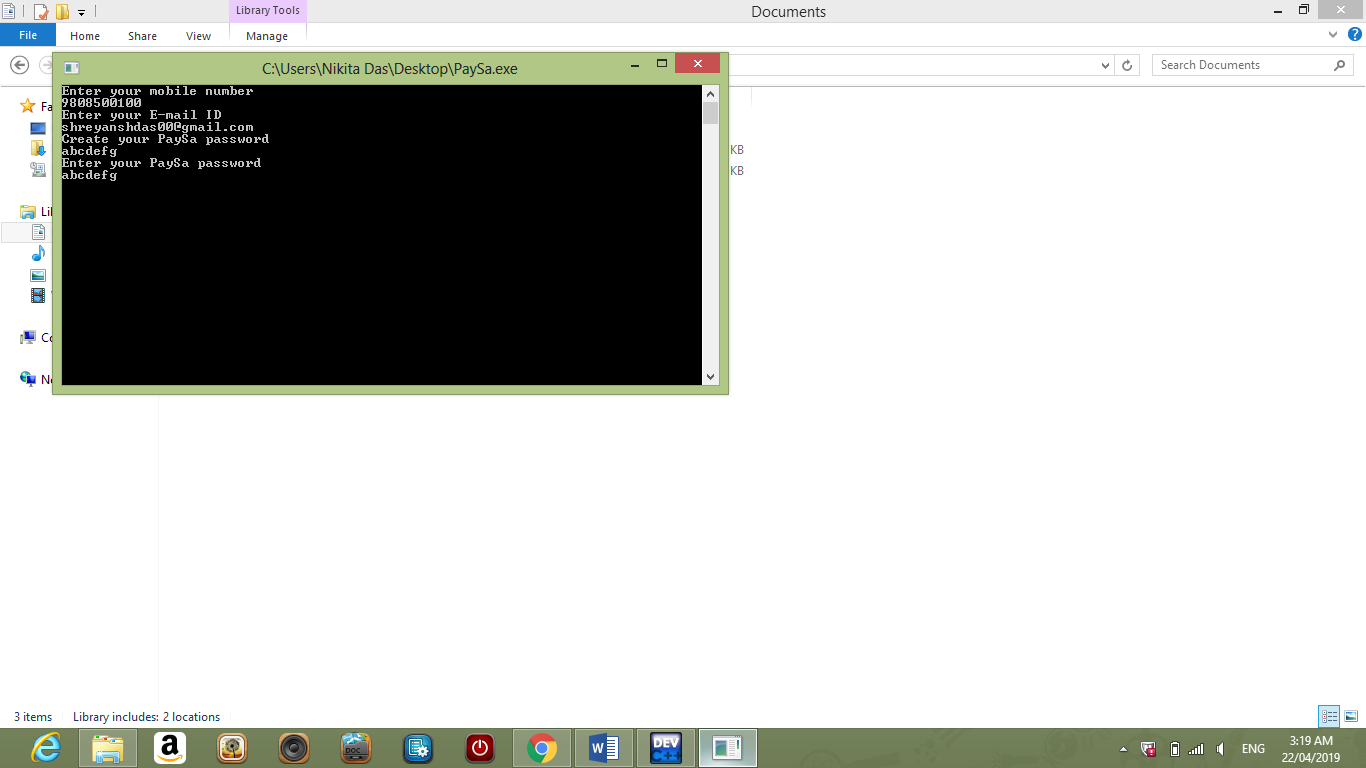
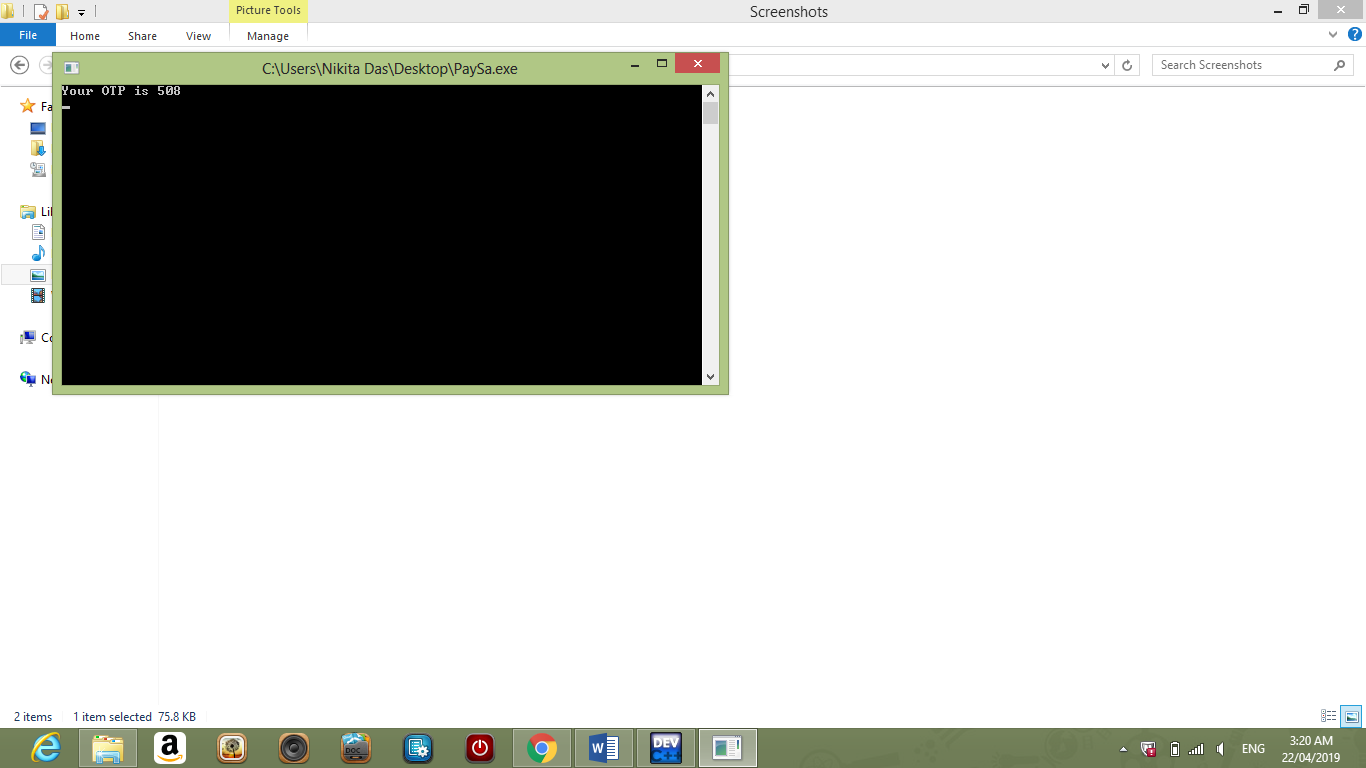
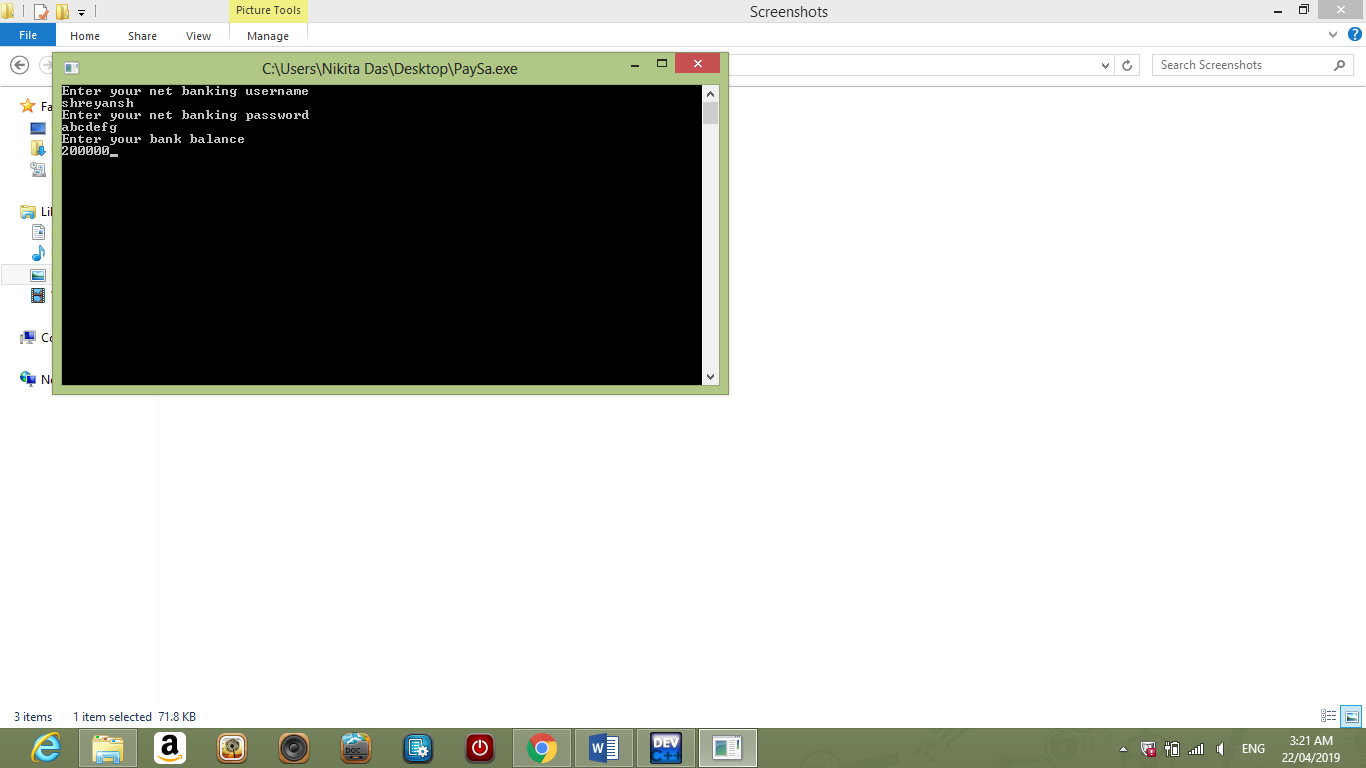
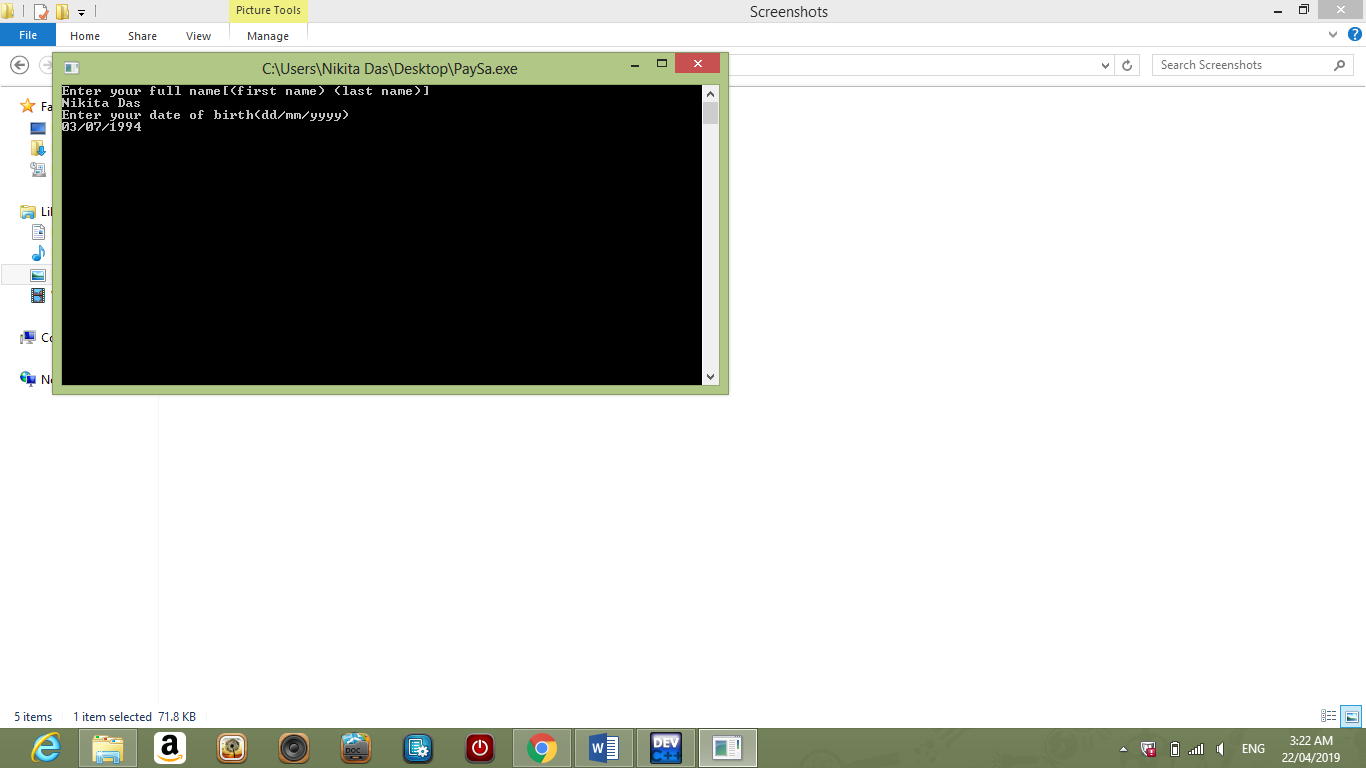
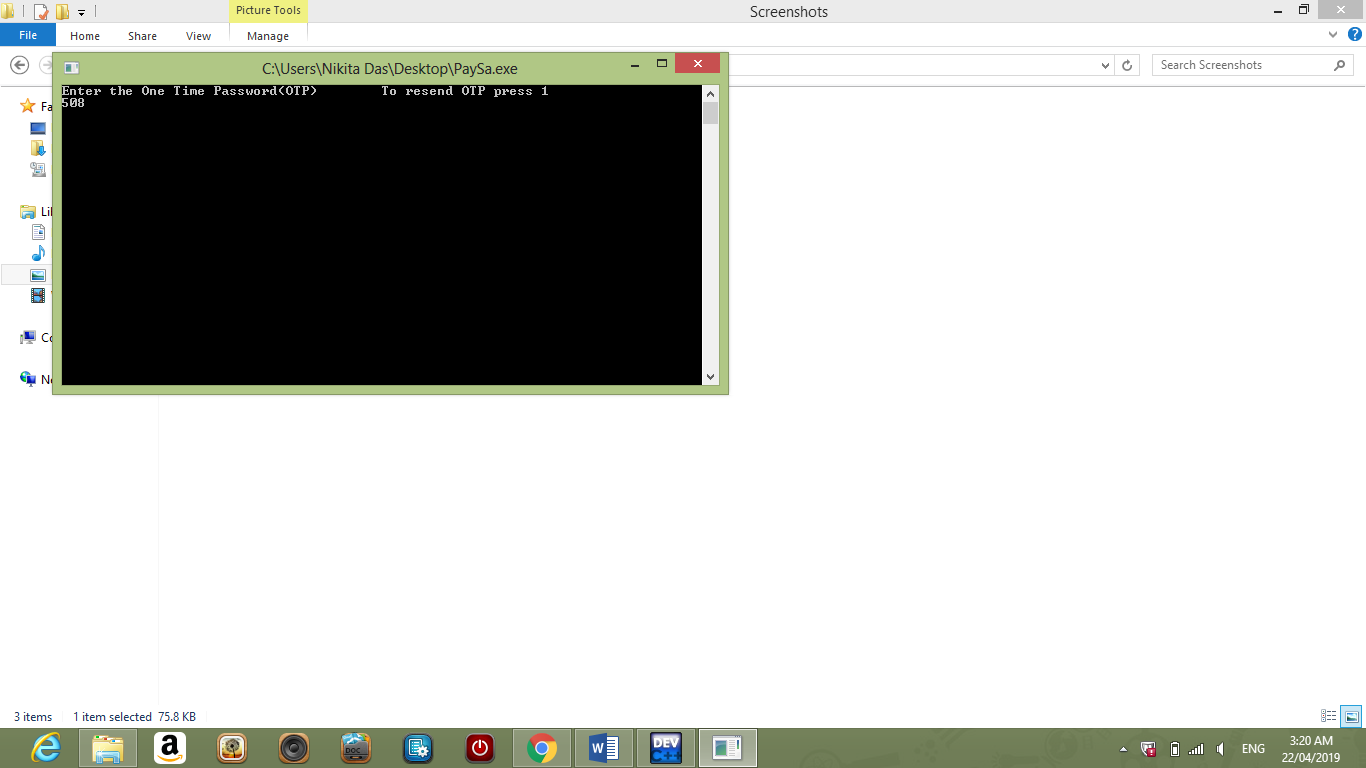
4.2.5 : Print “Invalid choice”

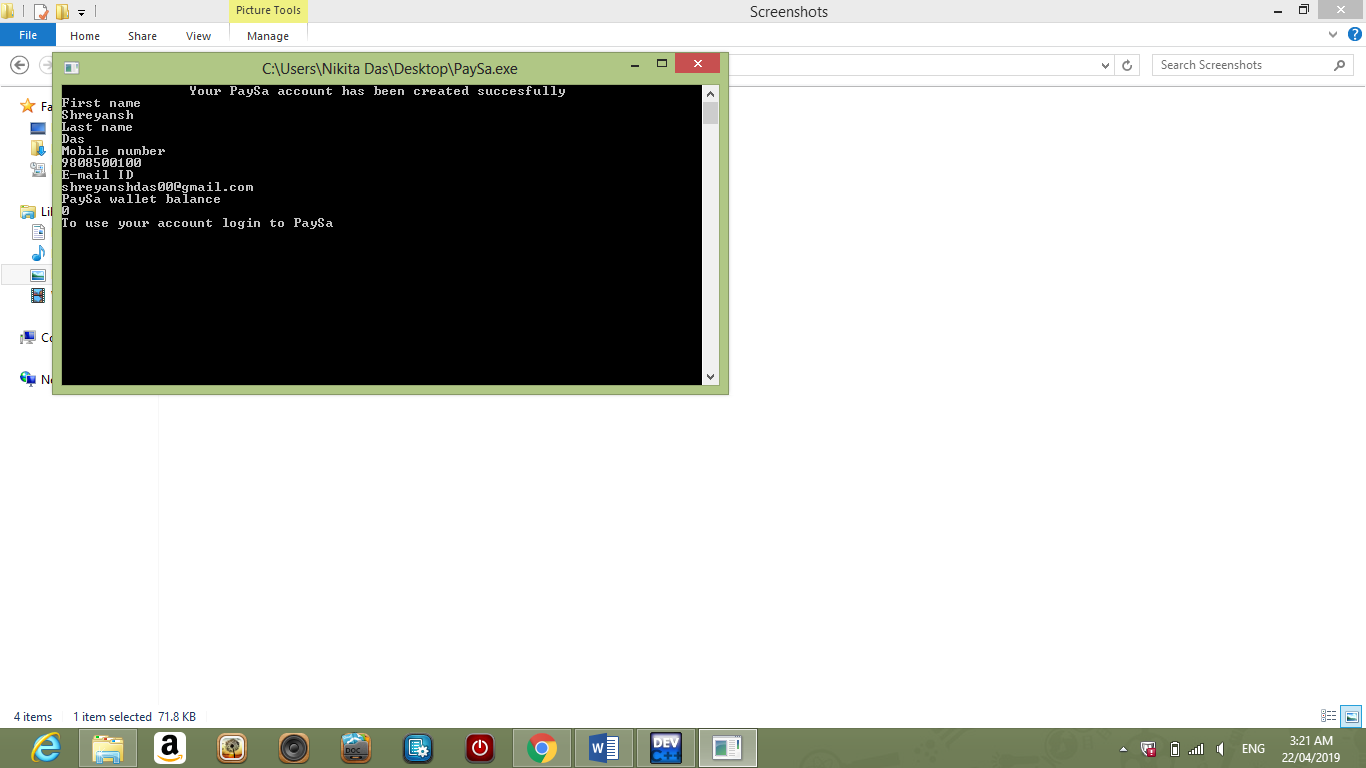
4.3 : Print “Do you want to continue using PaySa?” amd scan ch

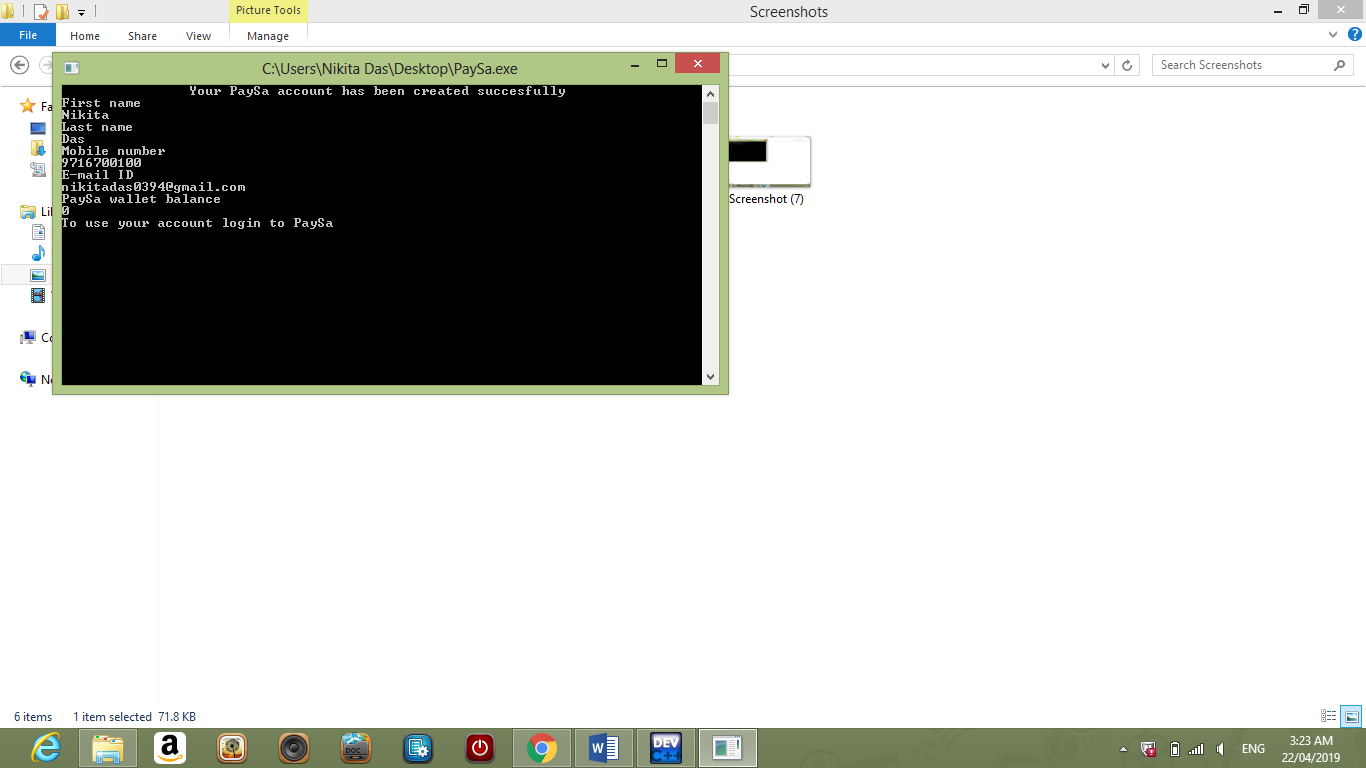
STEP 5 : Free the memory allocated to v and e

STEP 6 : Stop

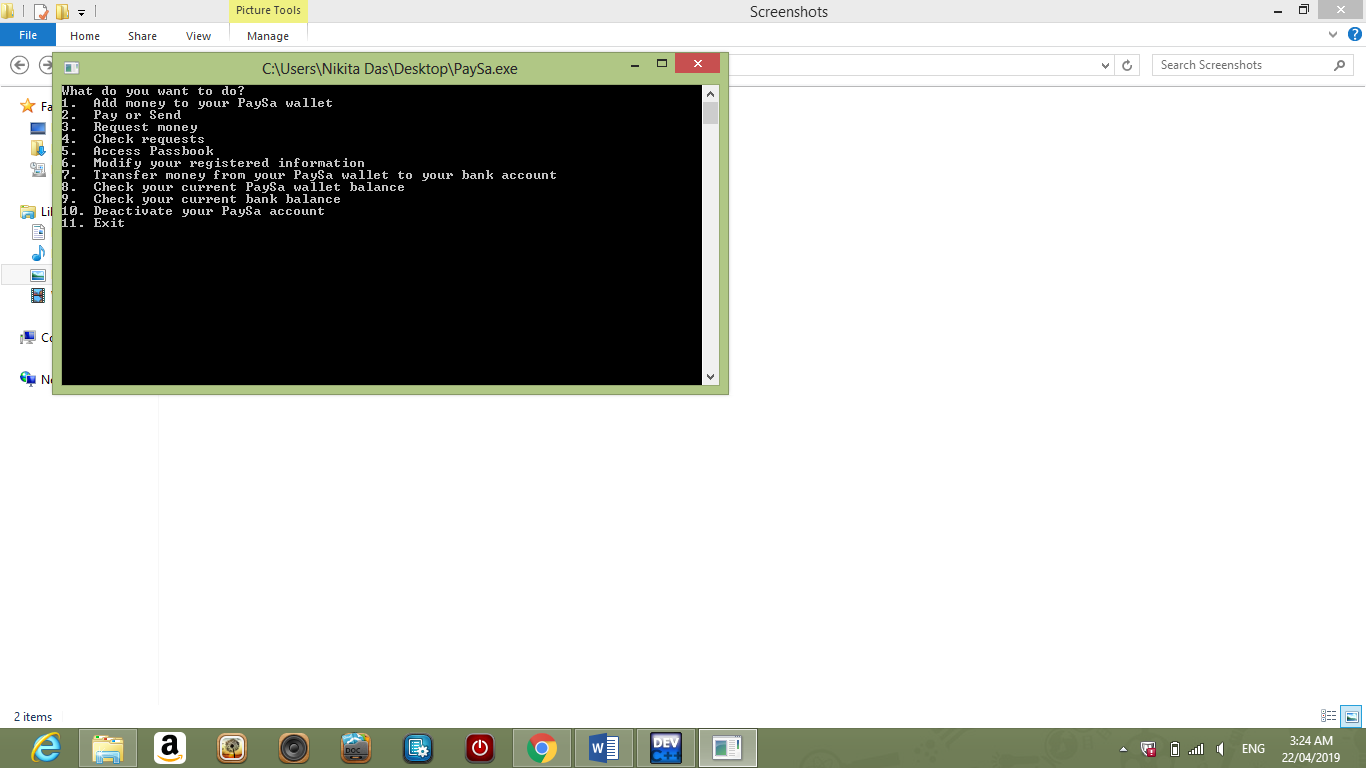
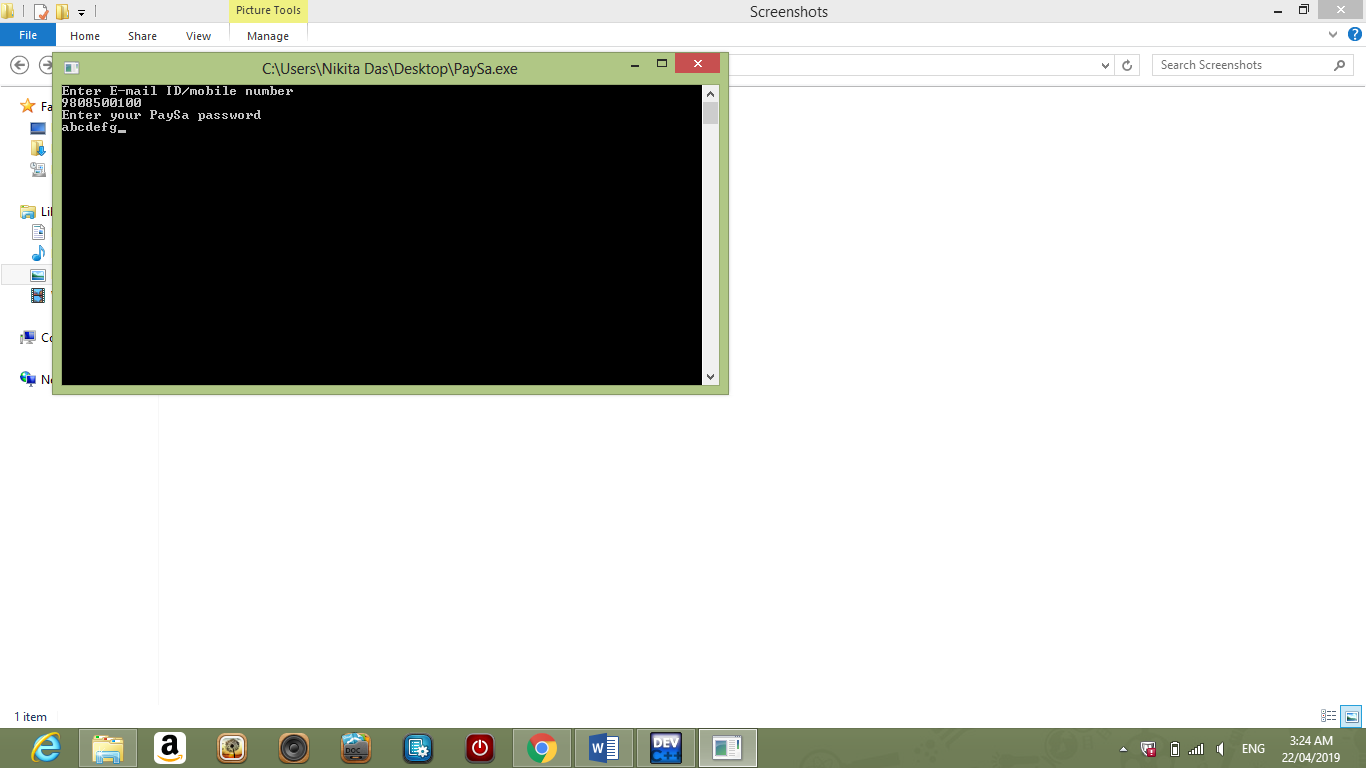
**SCREENSHOTS OF RESULTS**

* **MAIN MENU –**
* **SIGN UP –**

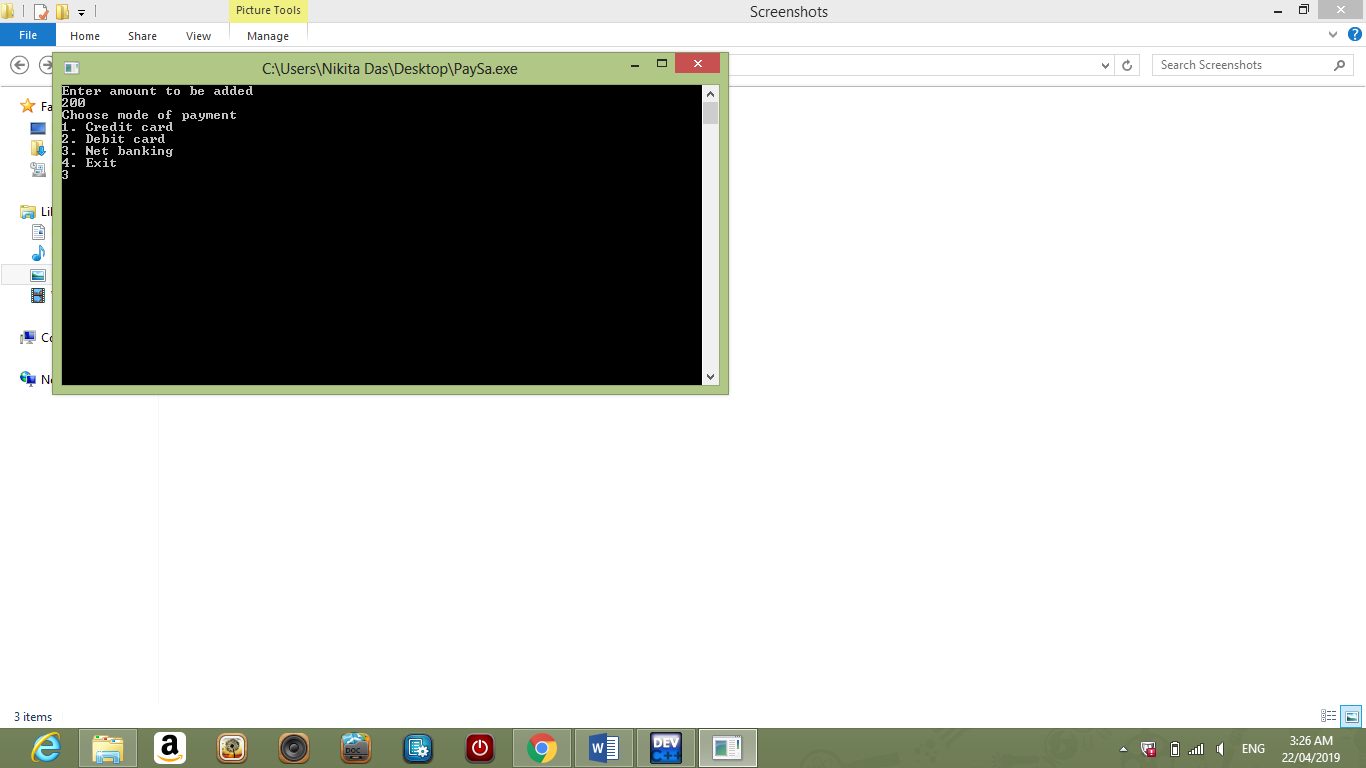
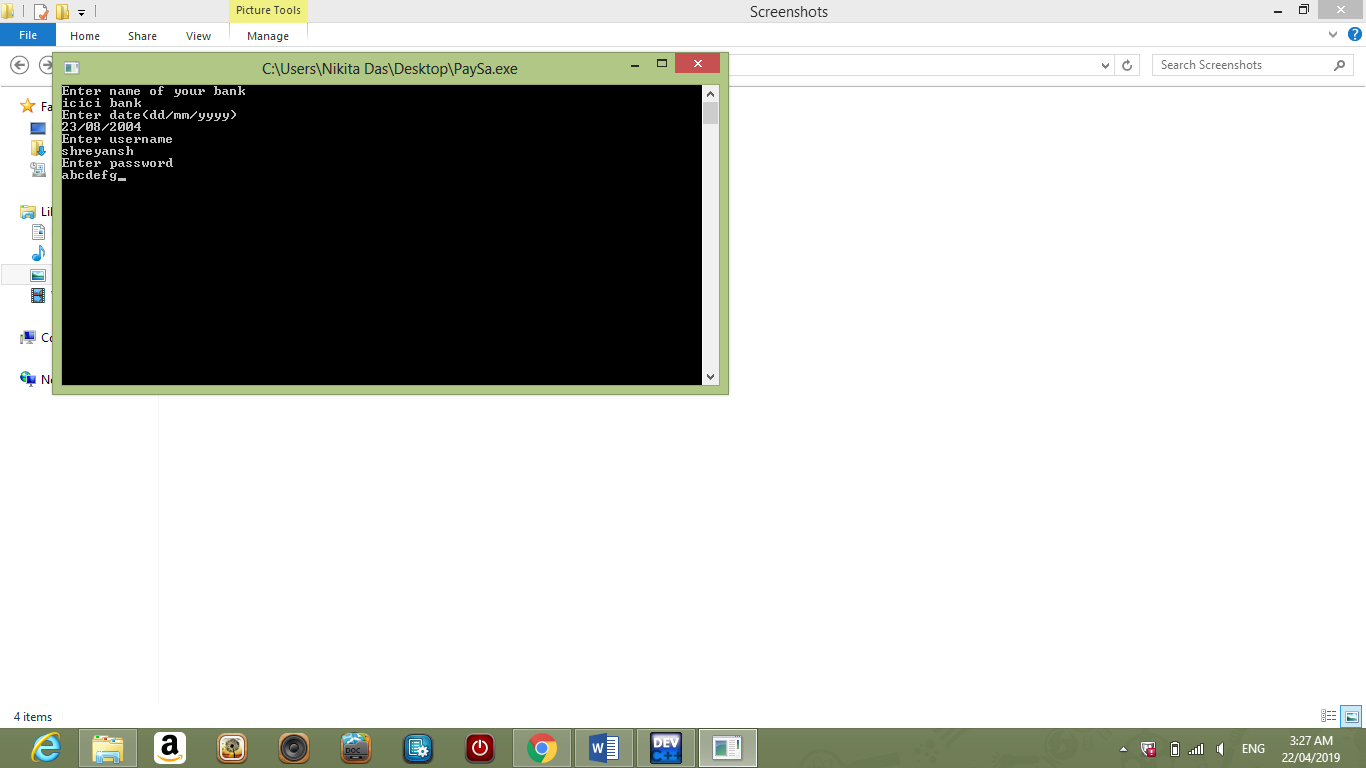
****

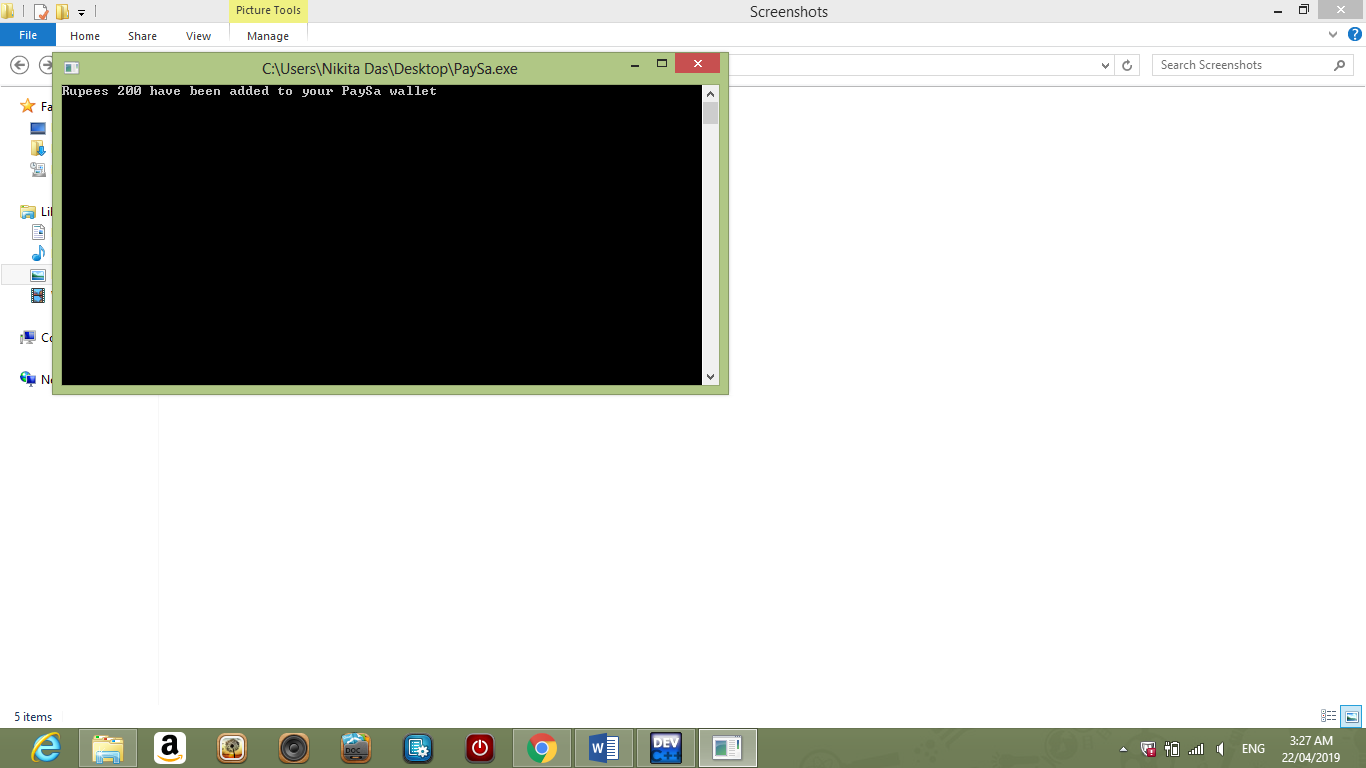
****

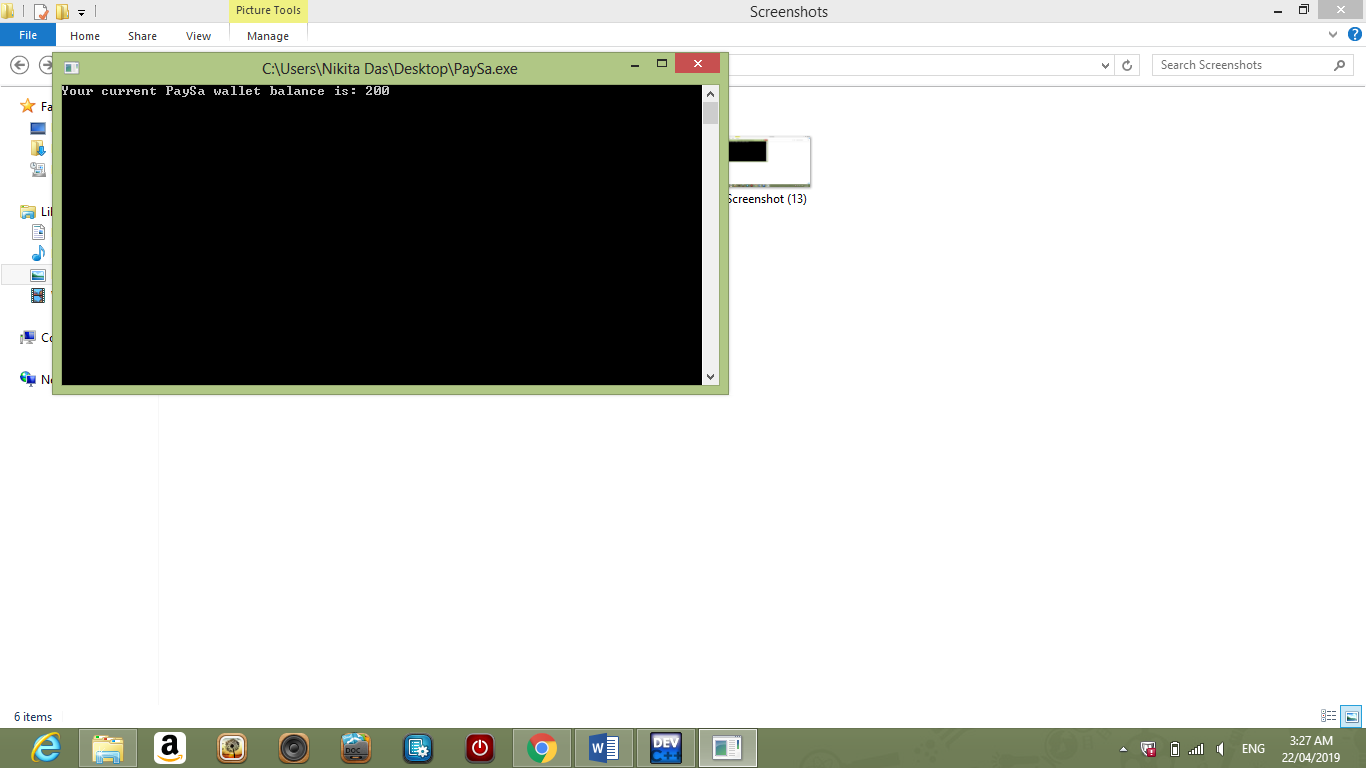
* **LOGIN –**

****

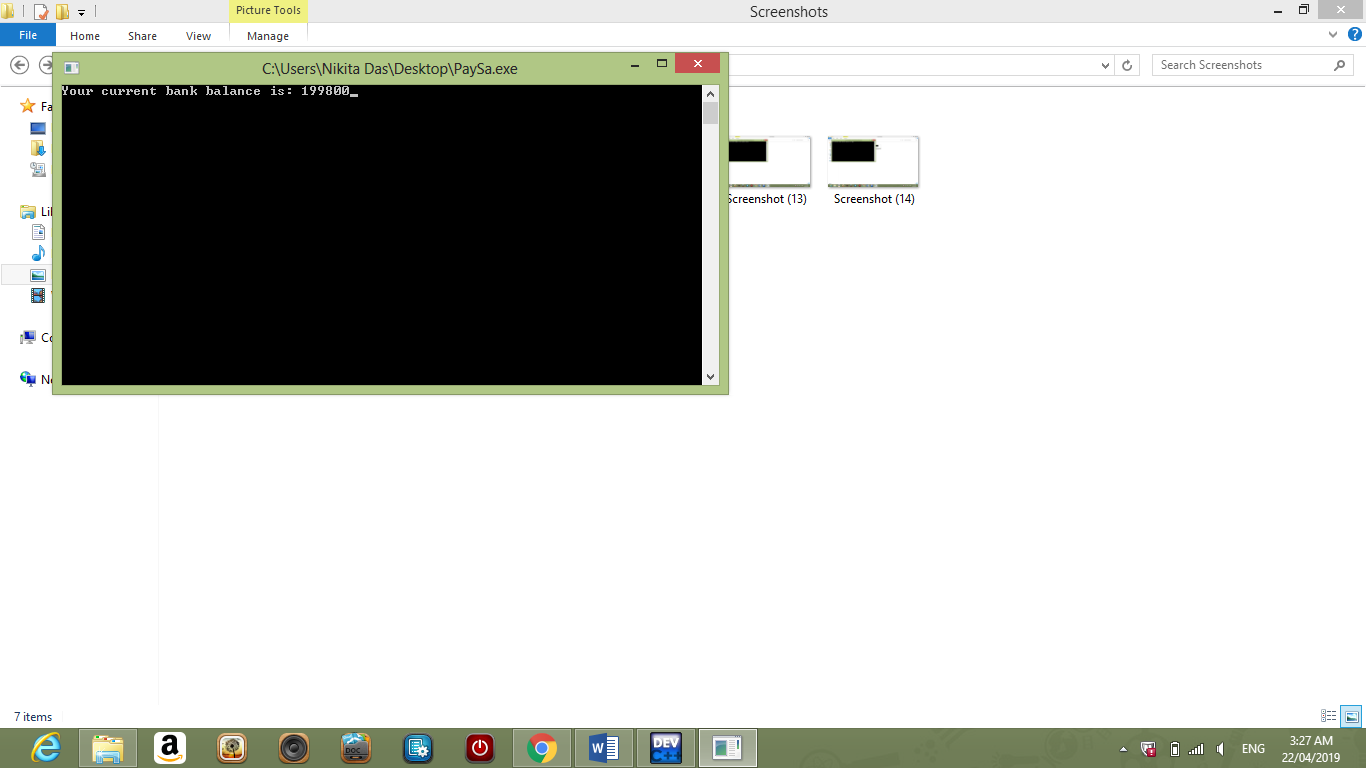
* **deposit() –**

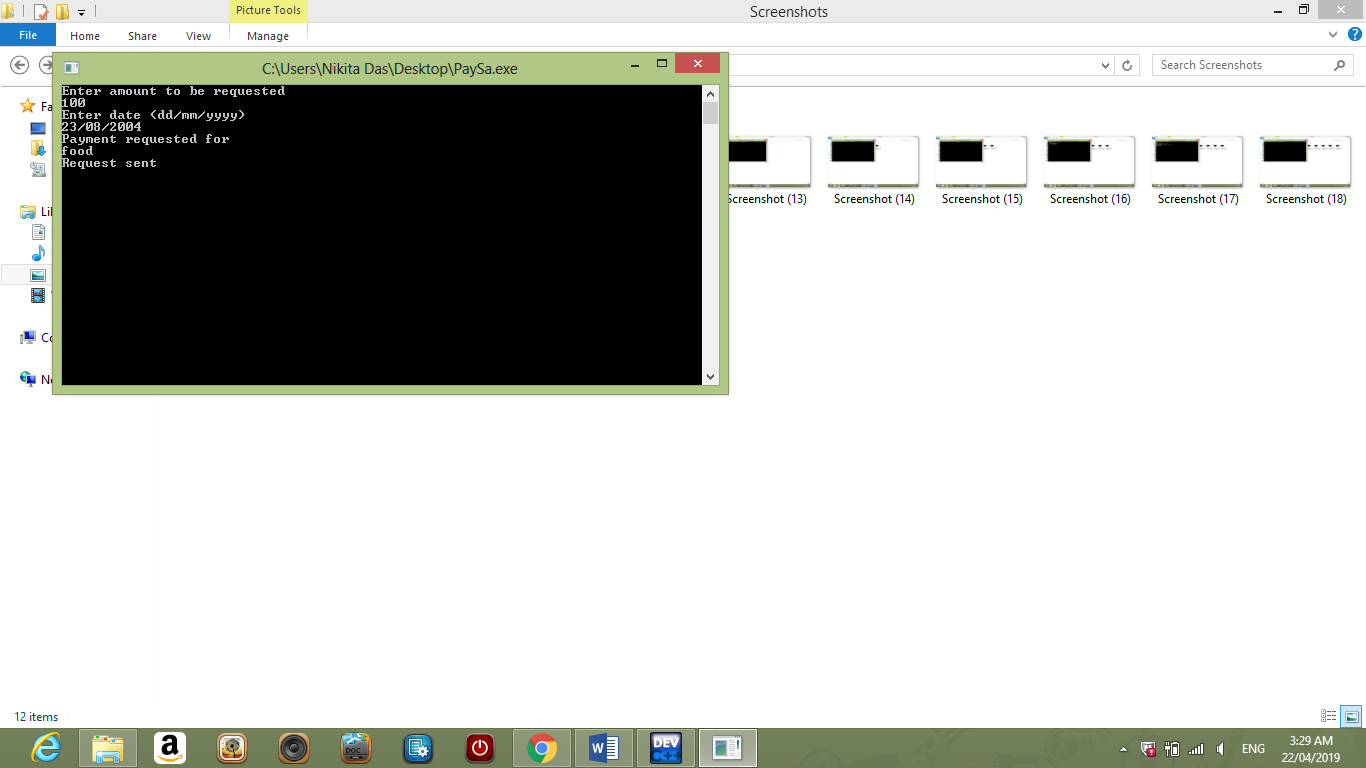
****

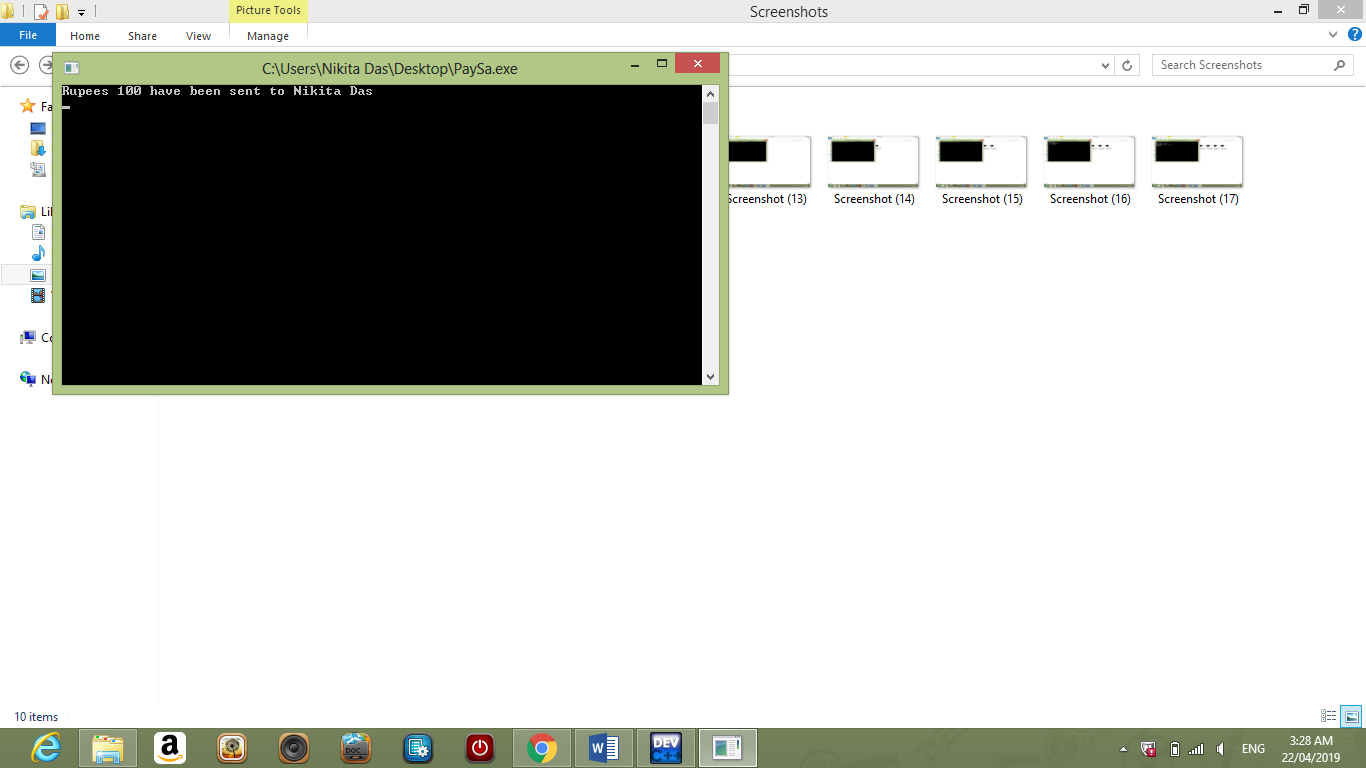
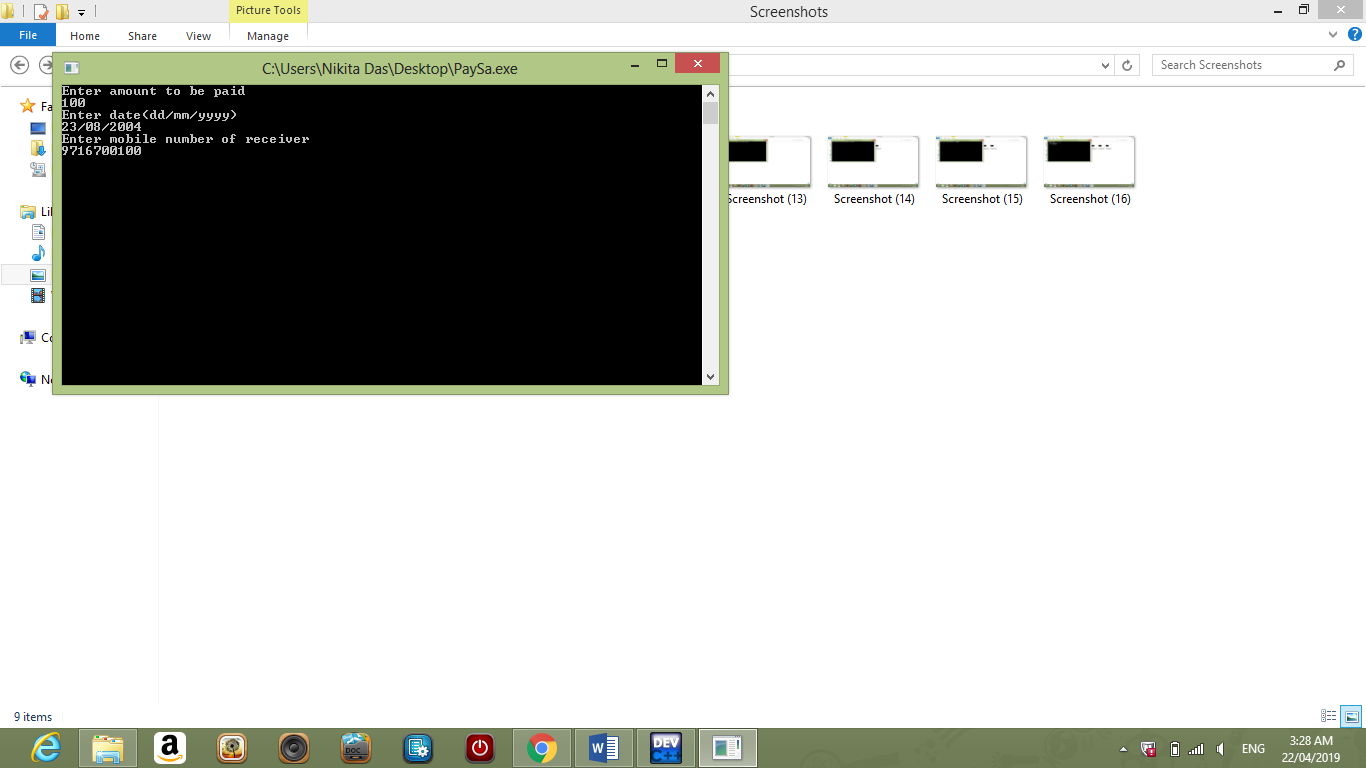
****

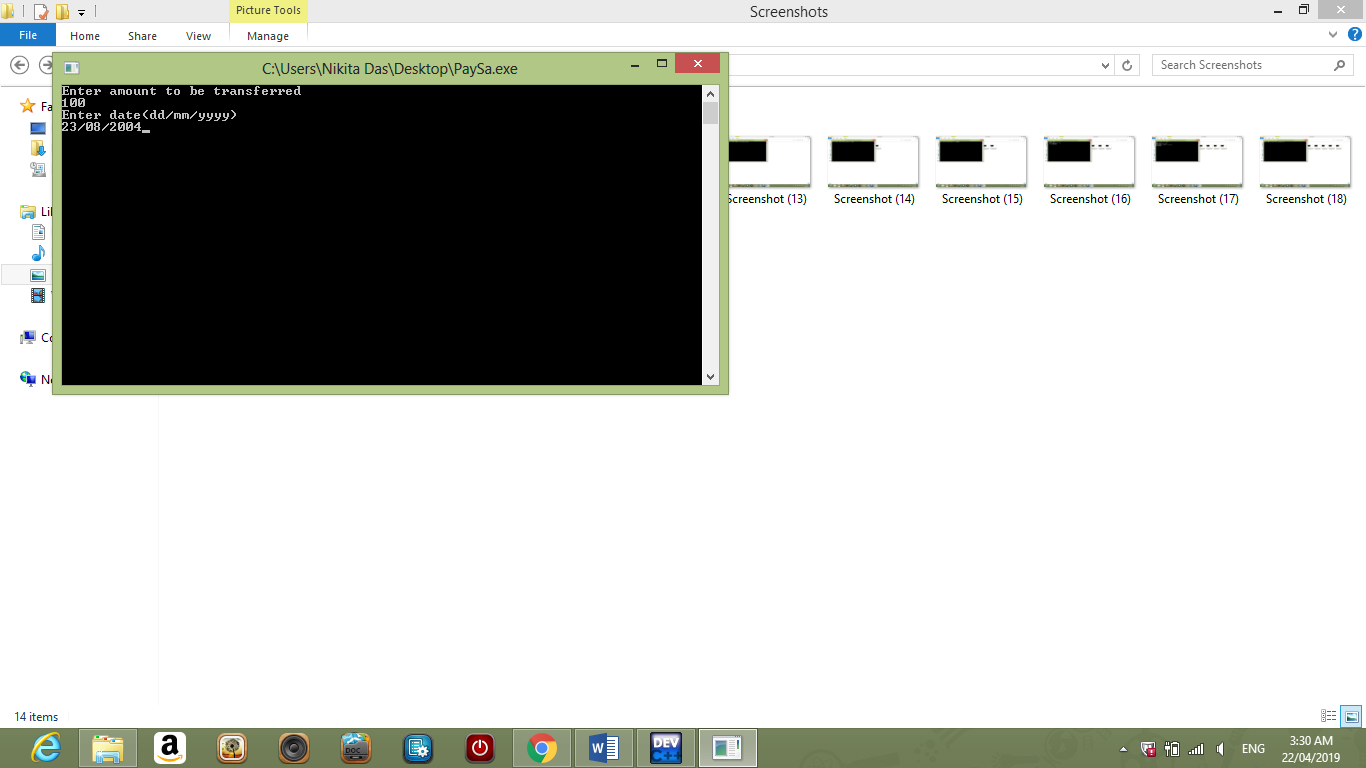
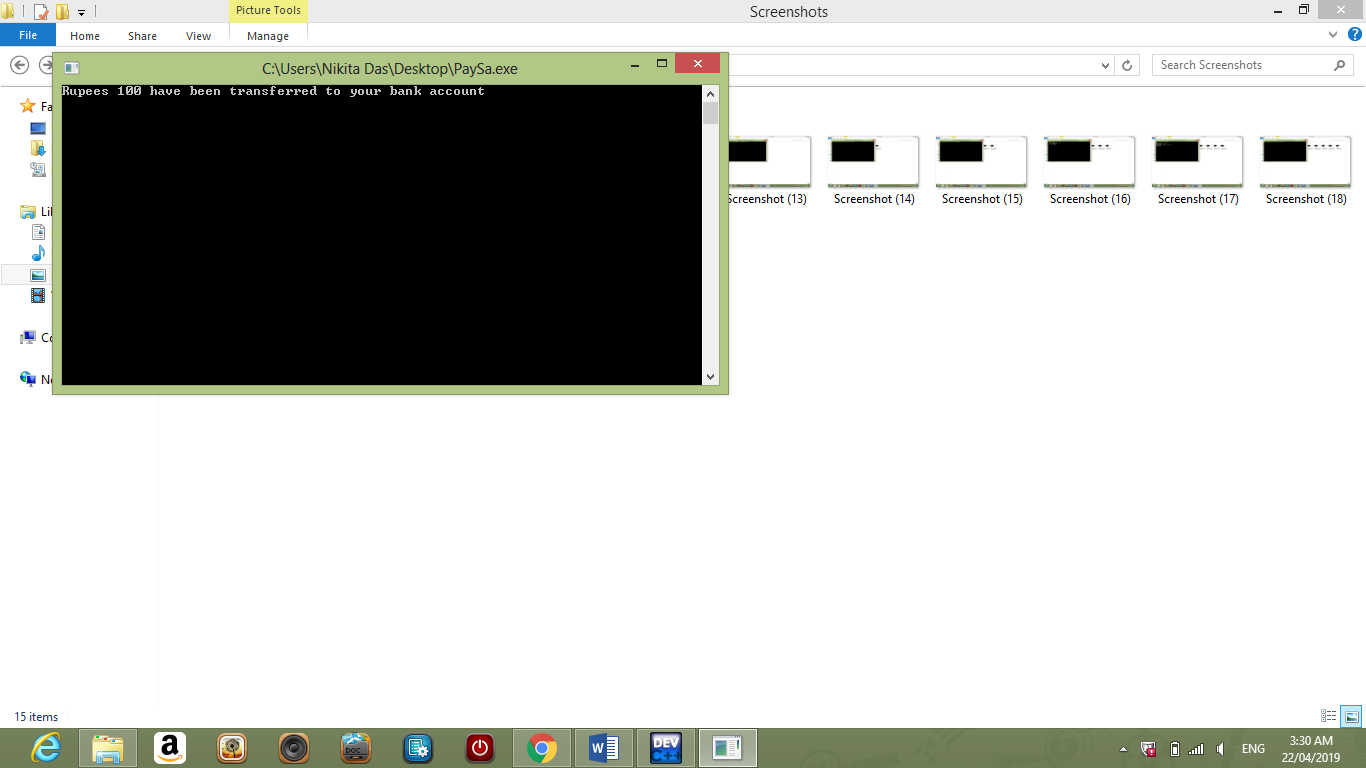
****

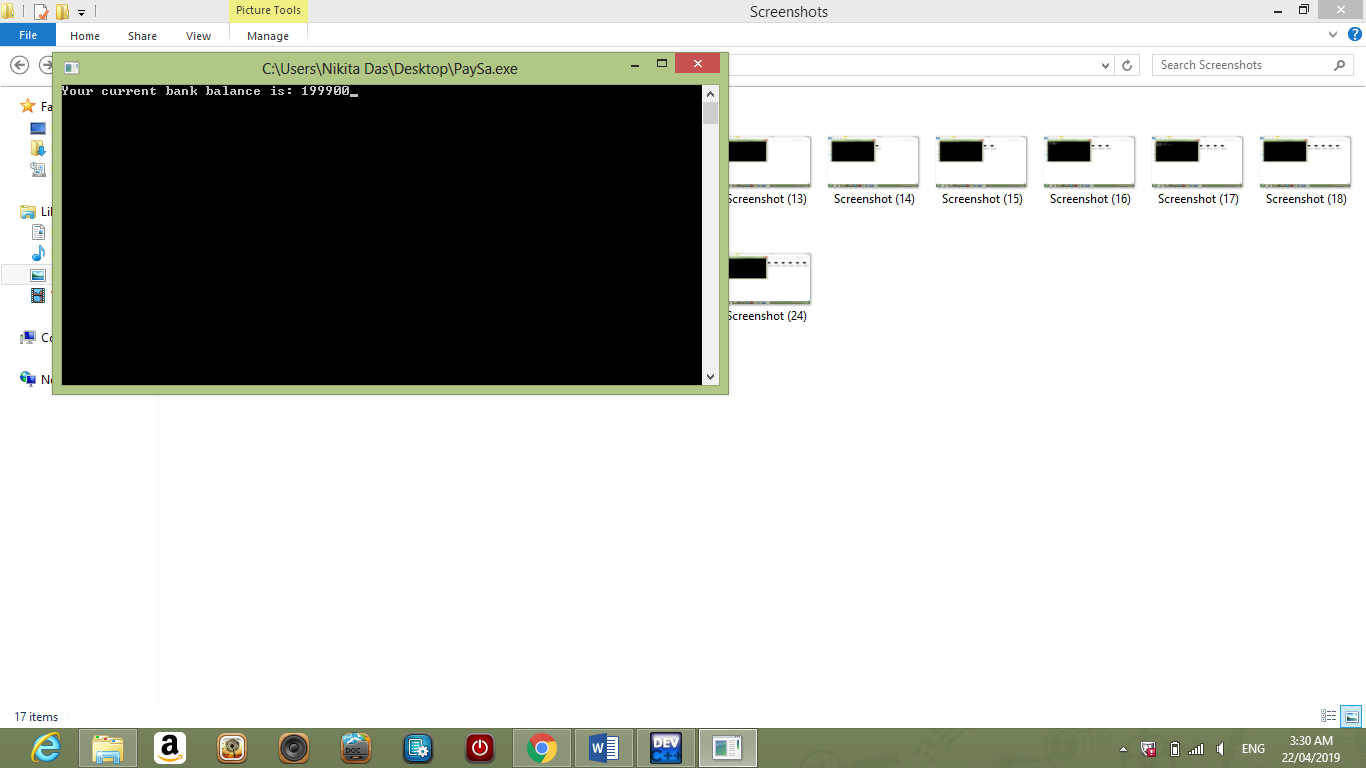
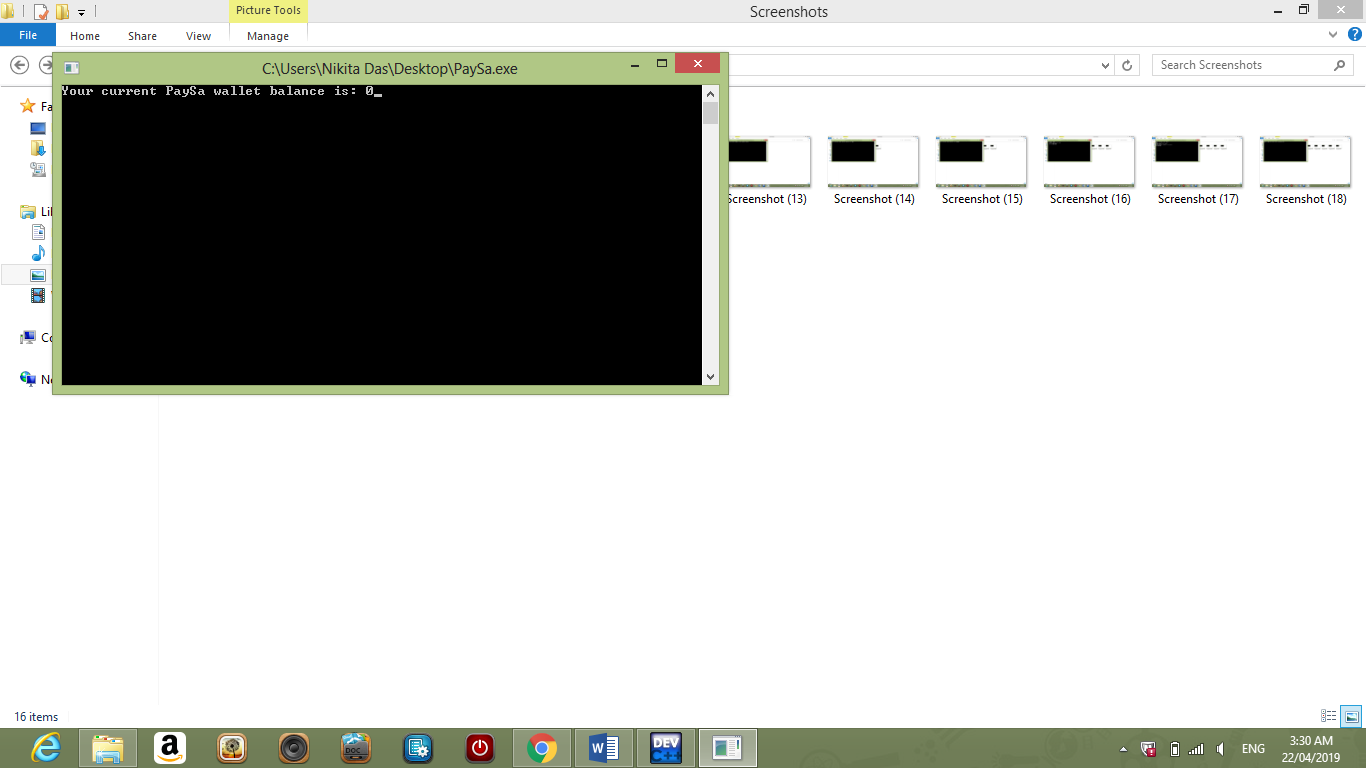
* **CHECK CURRENT PAY-SA WALLET BALANCE –**

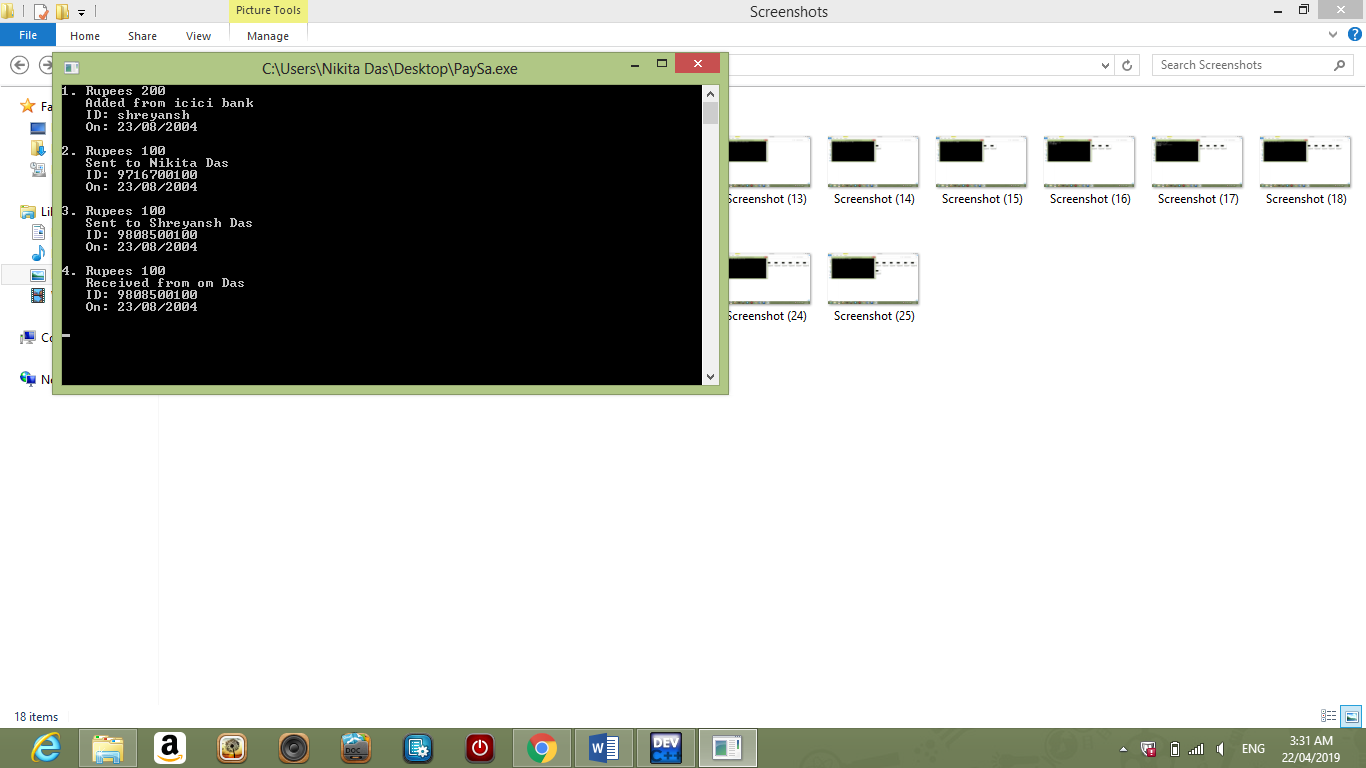
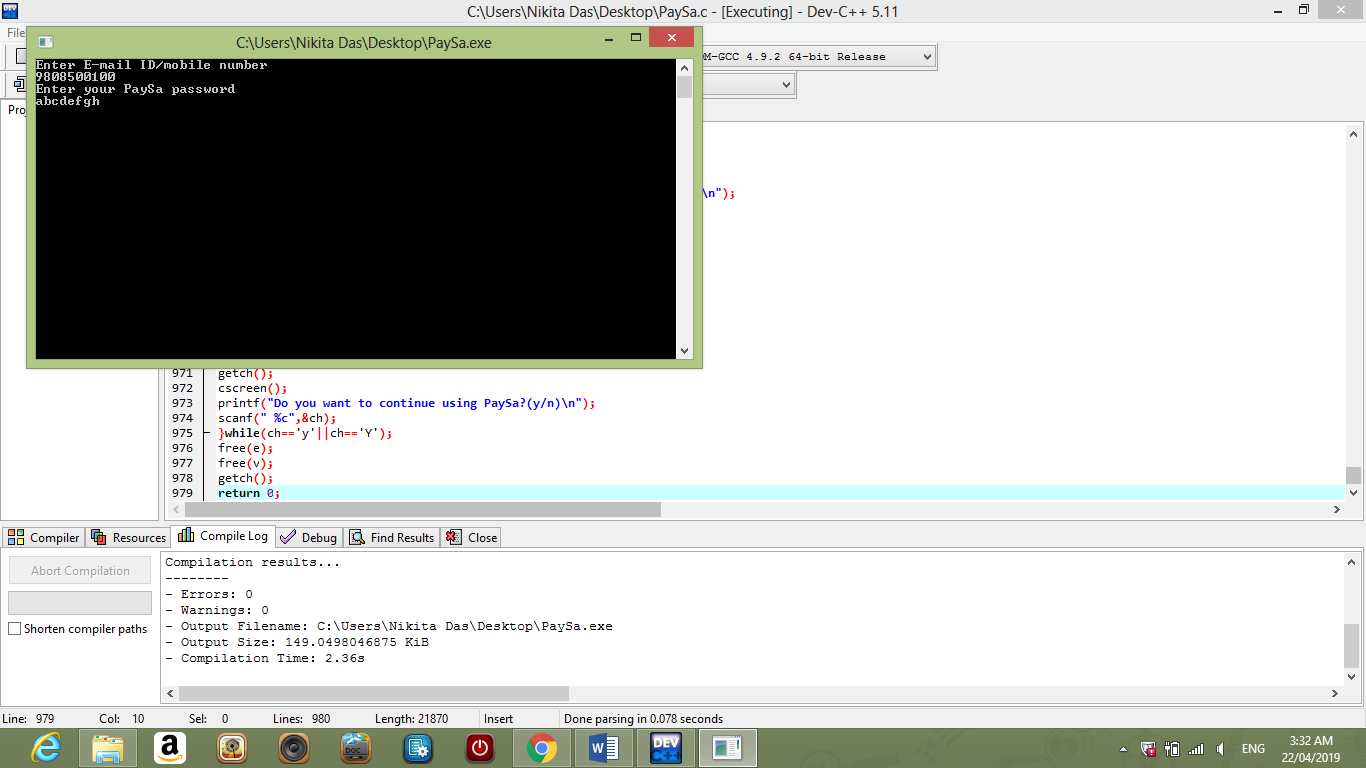
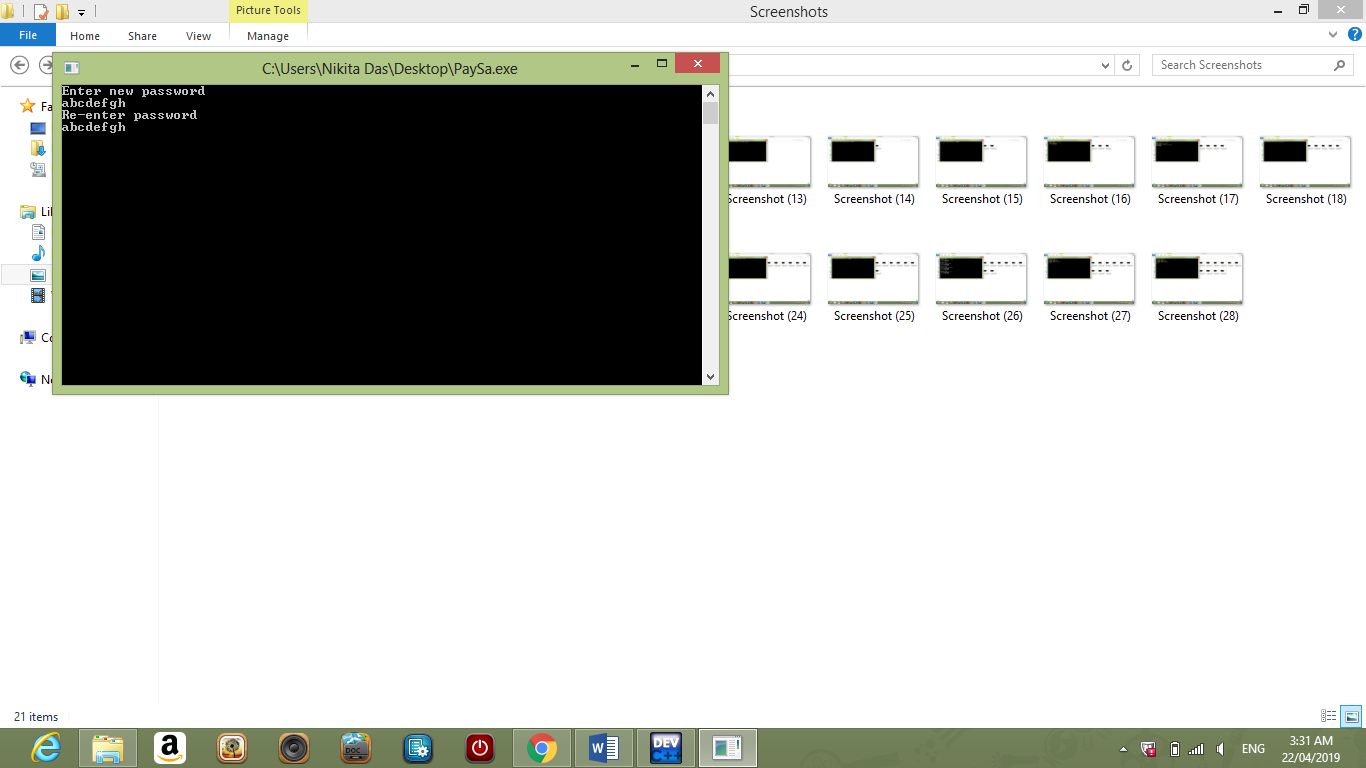
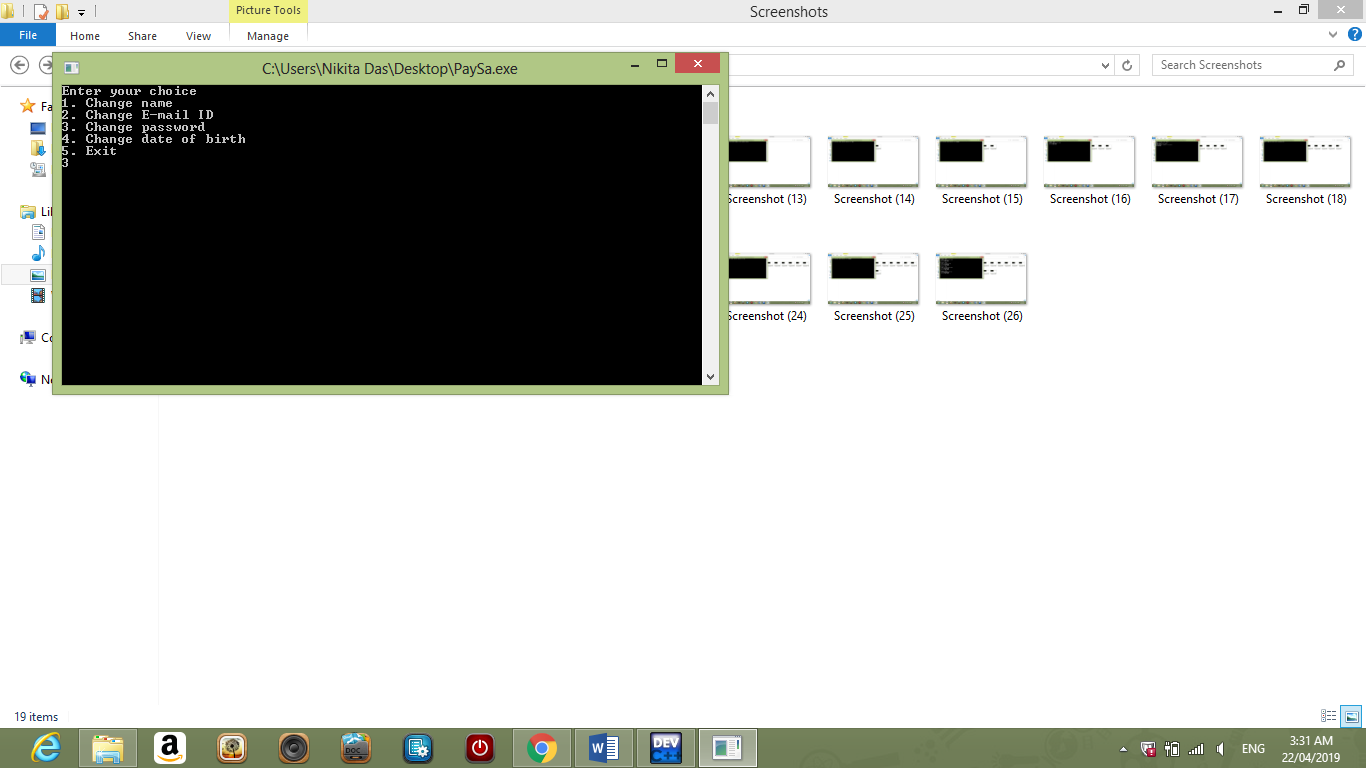
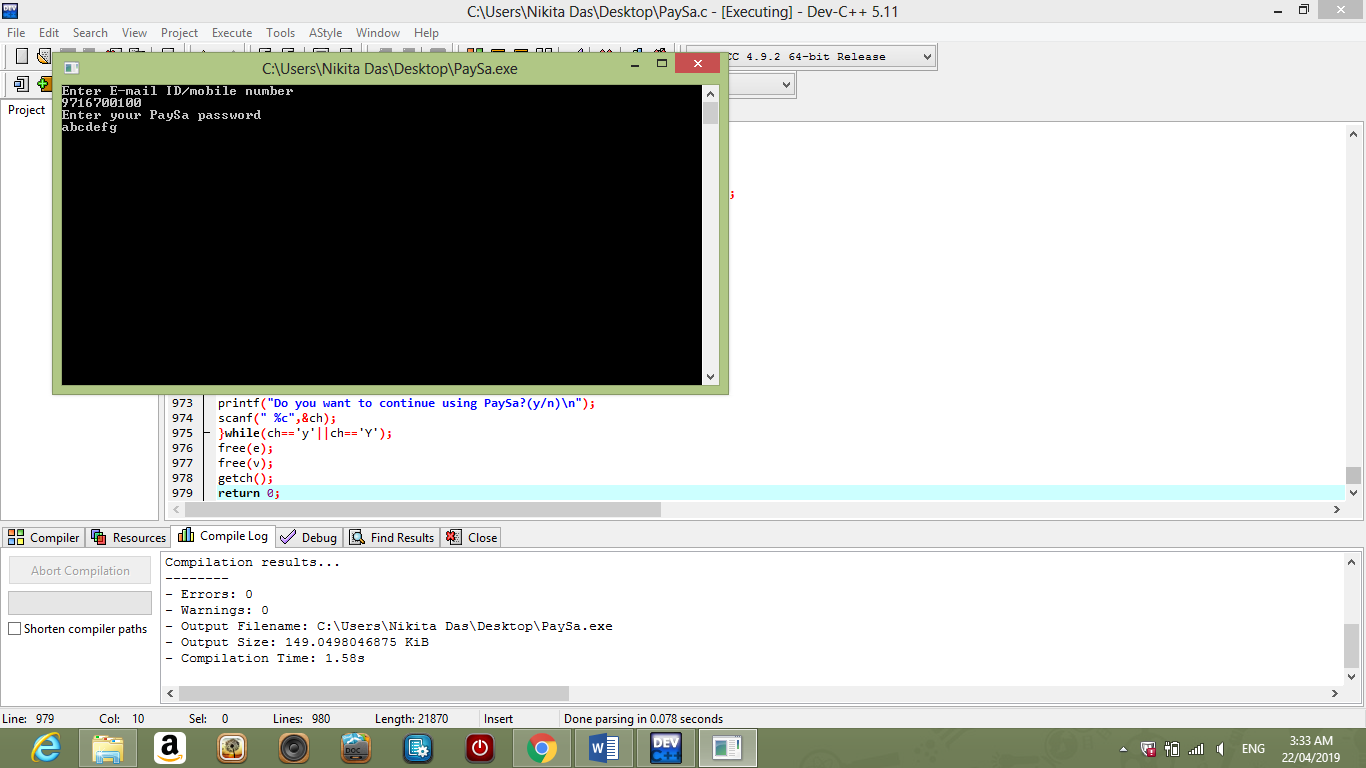
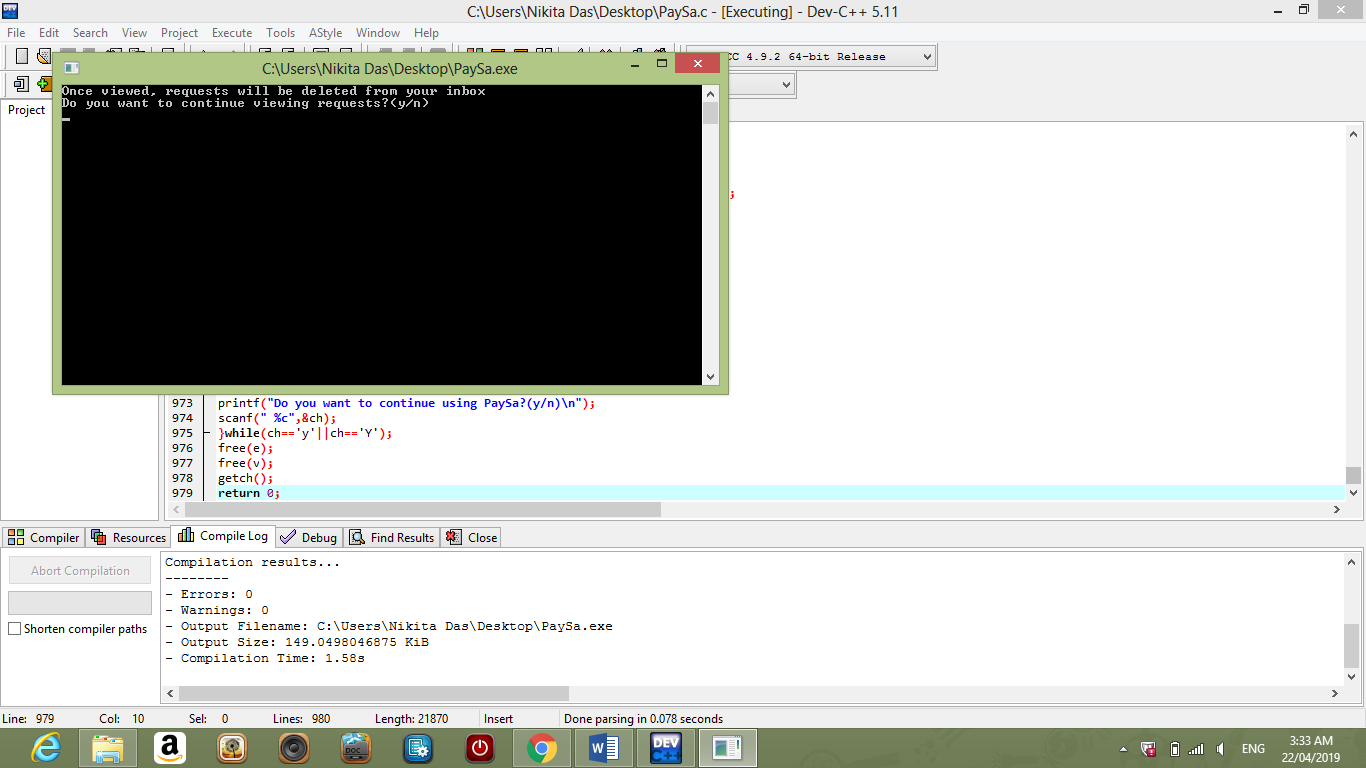
****

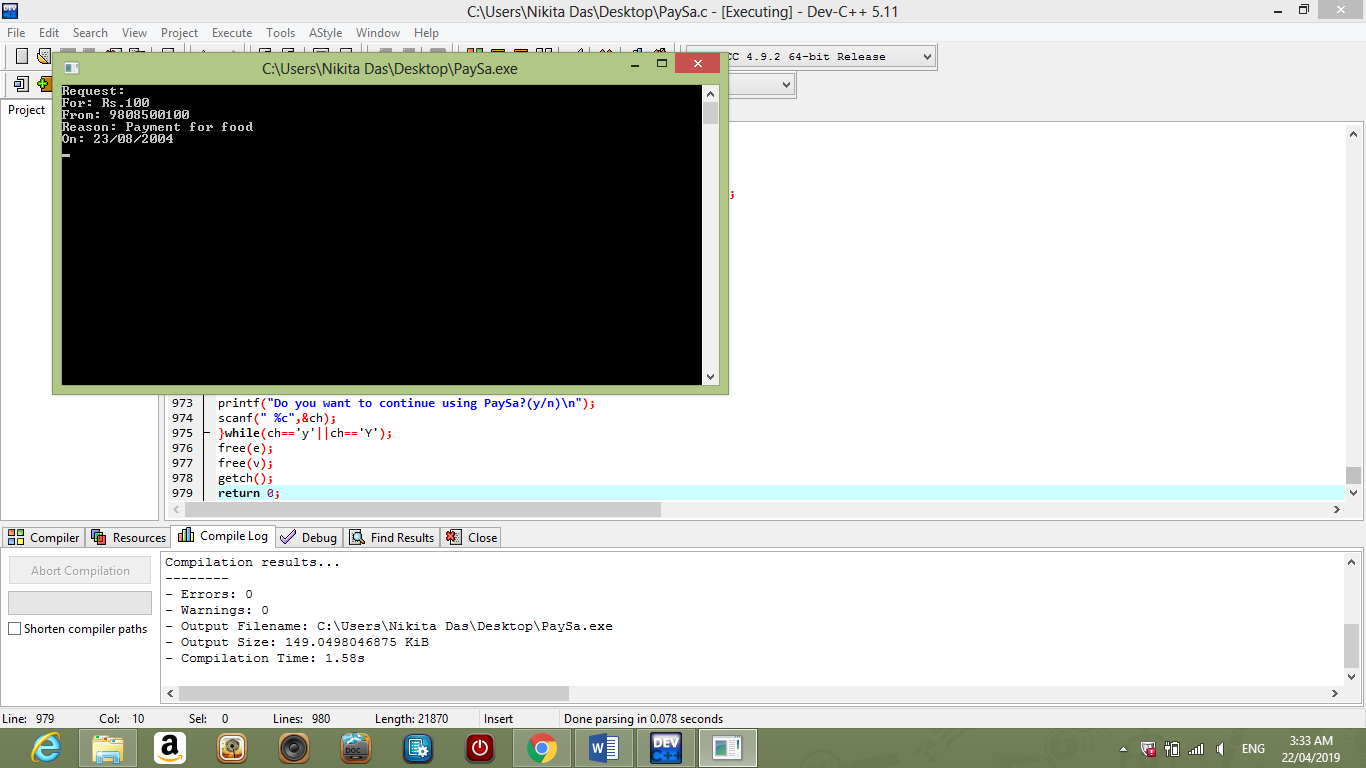
* **CHECK CURRENT BANK BALANCE –**
* **request() -**
* **pay() –**

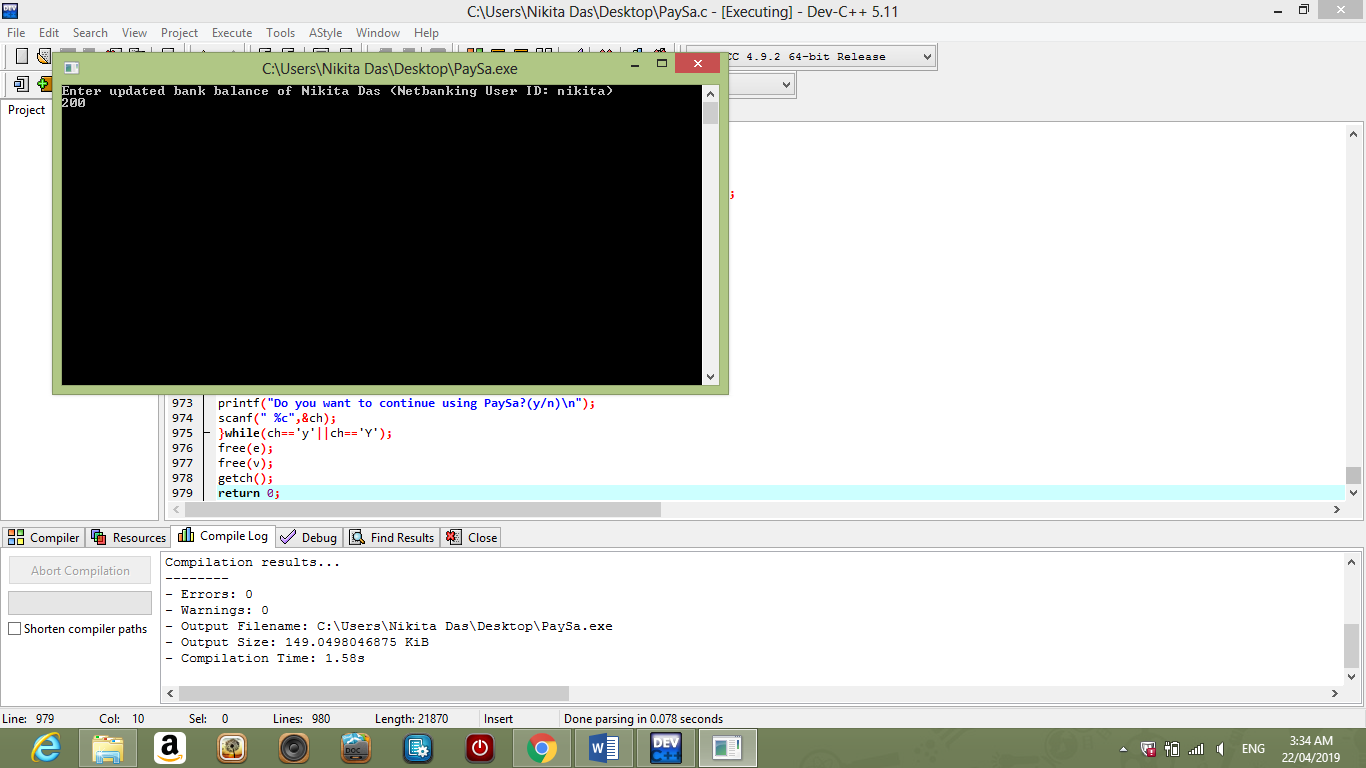
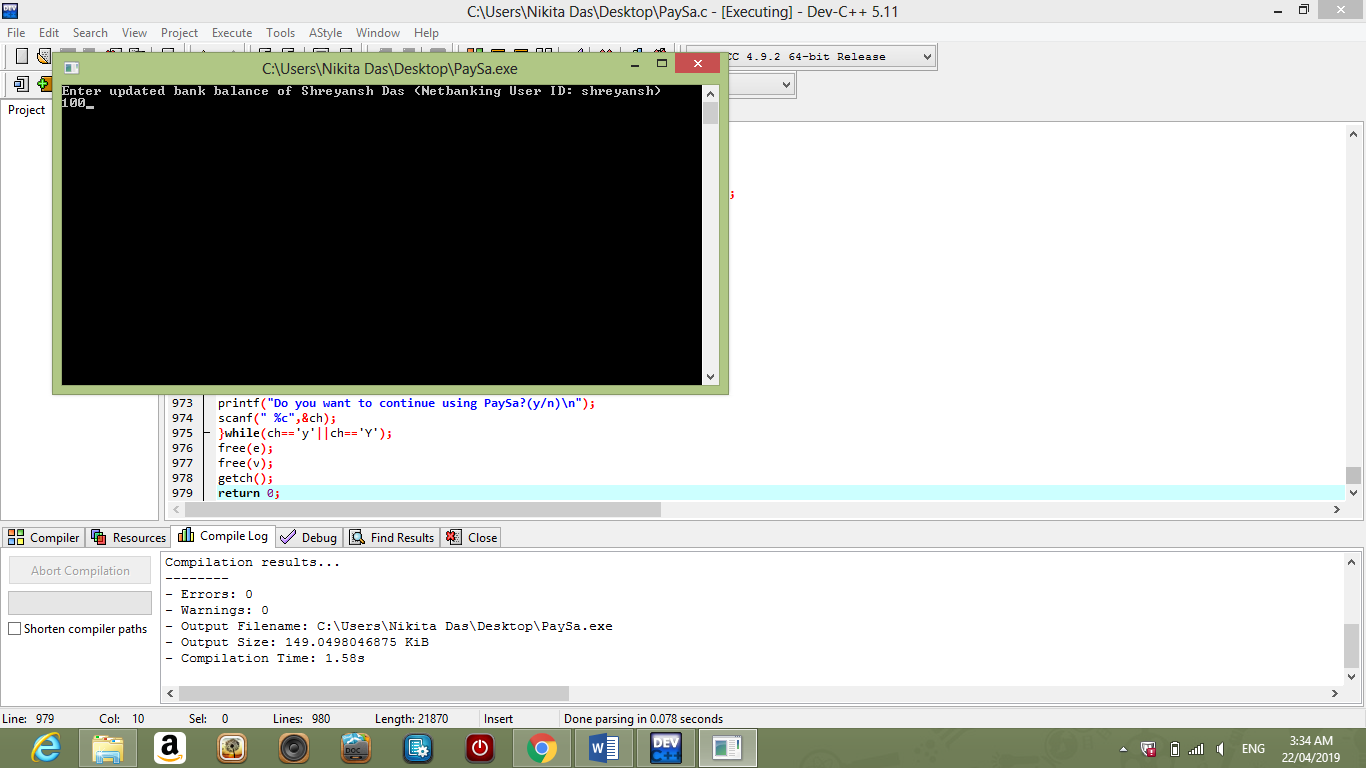
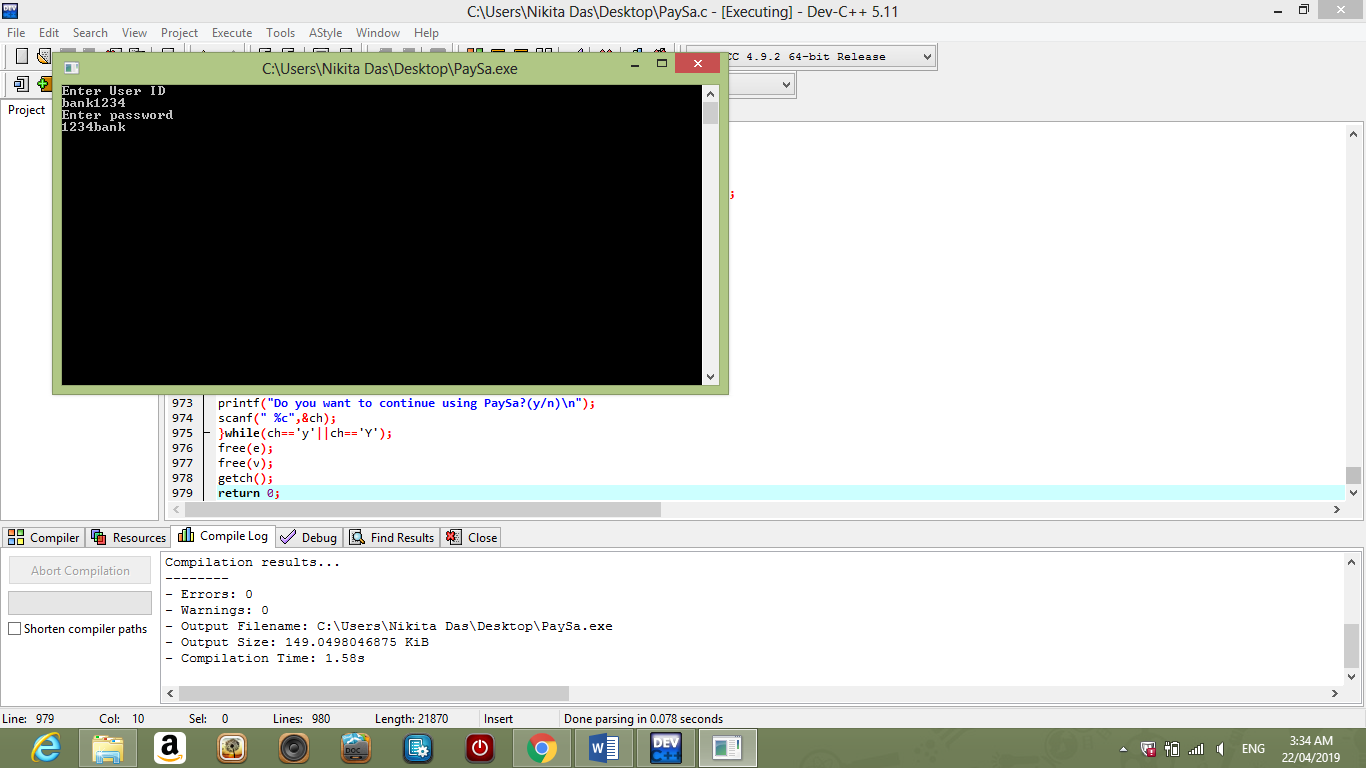
****

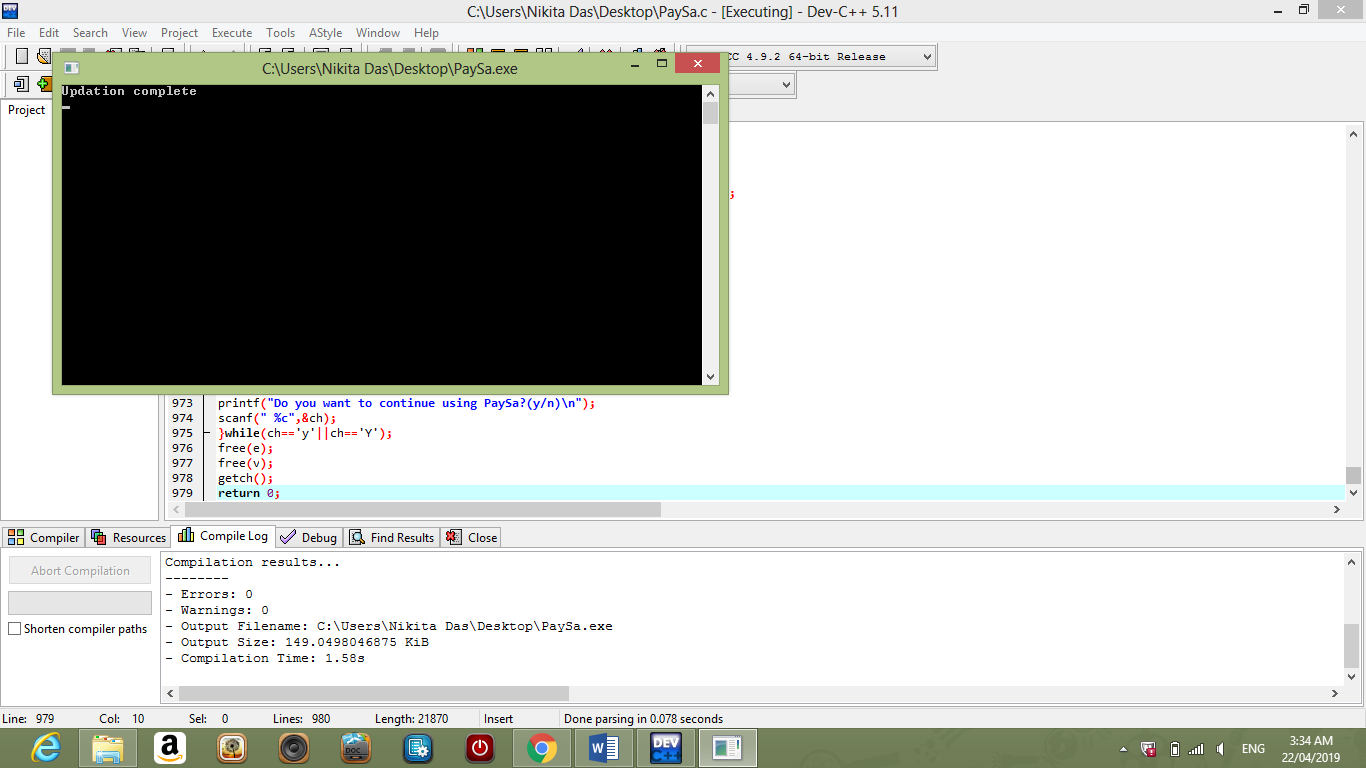
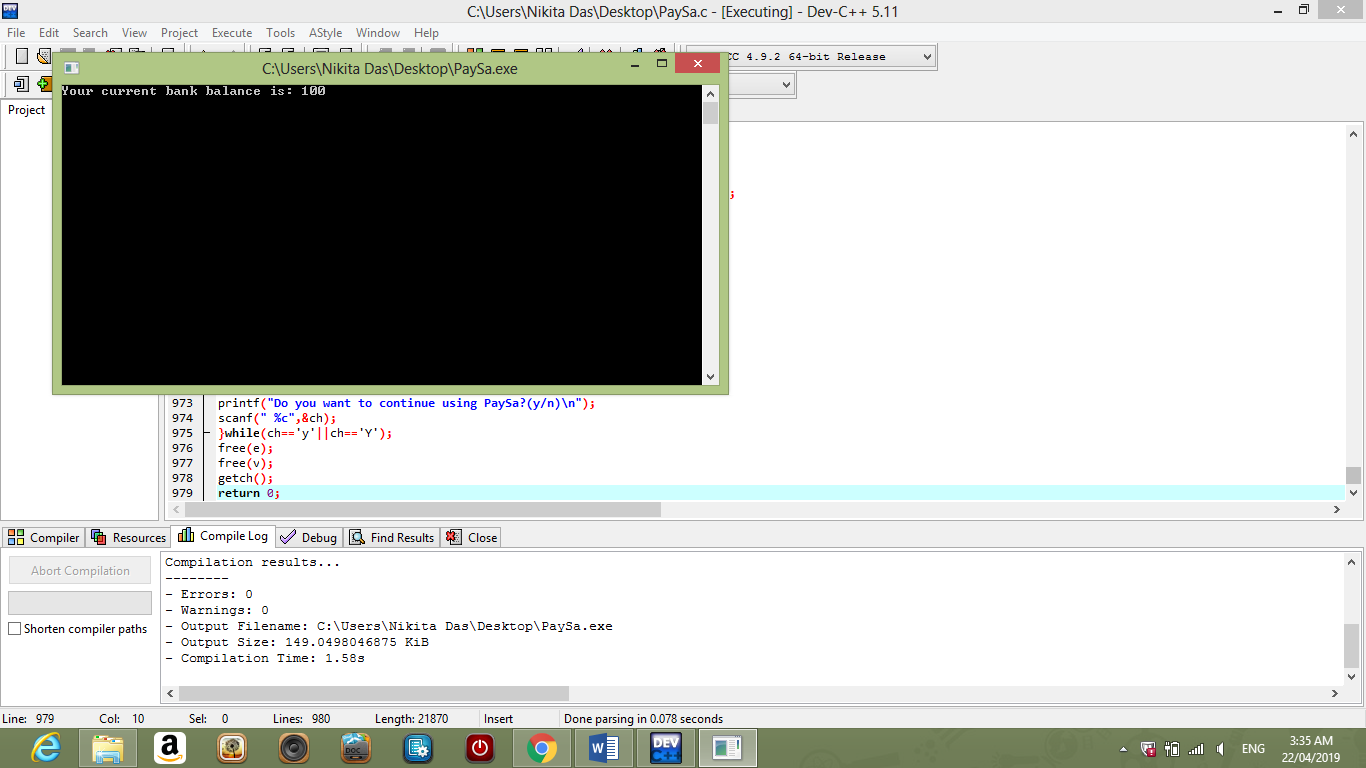
* **tbank() -**

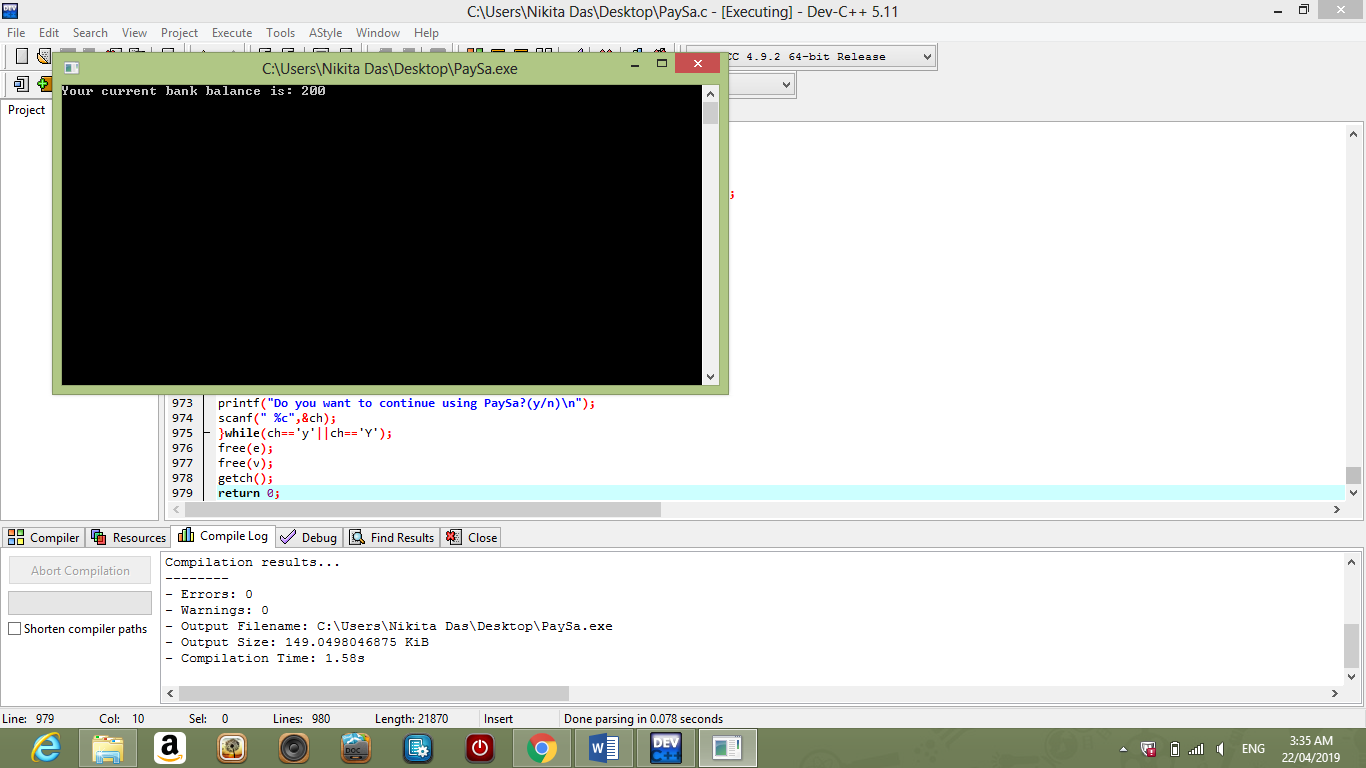
****

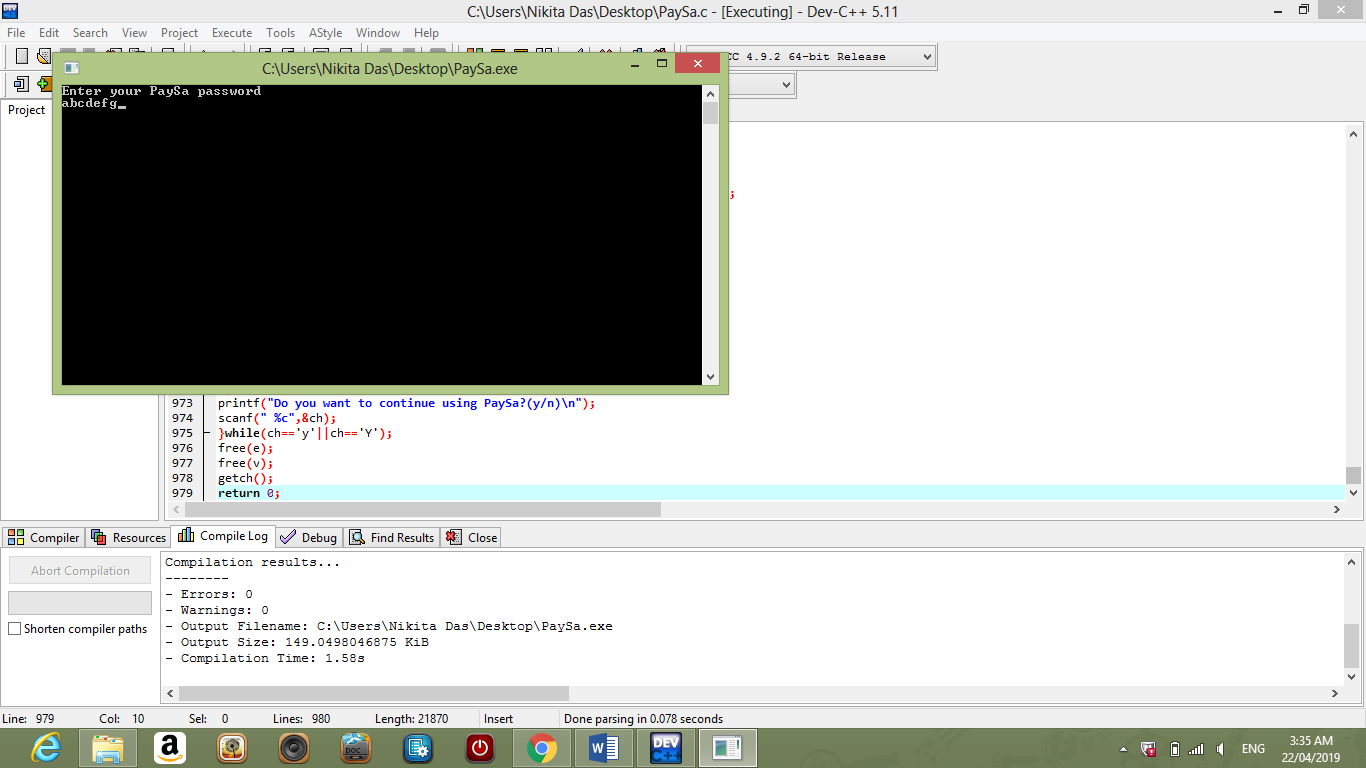
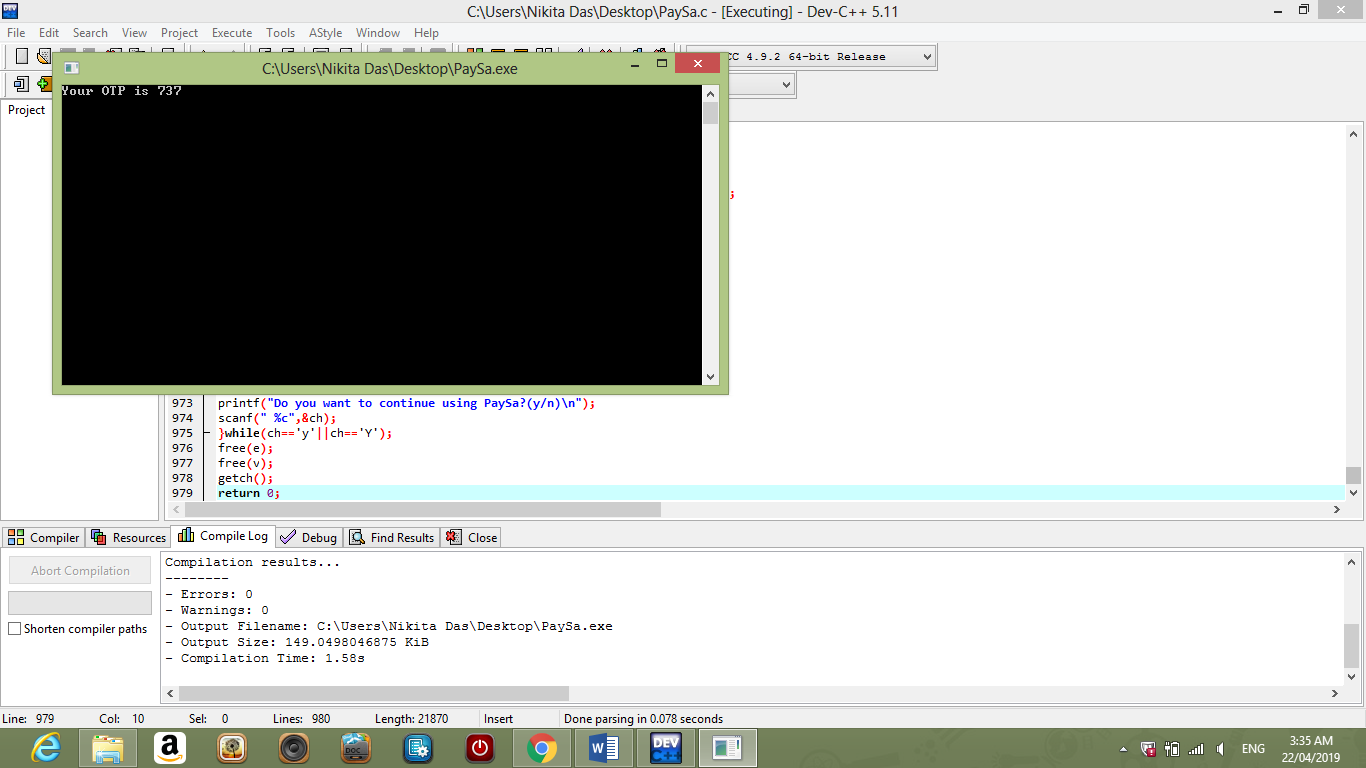
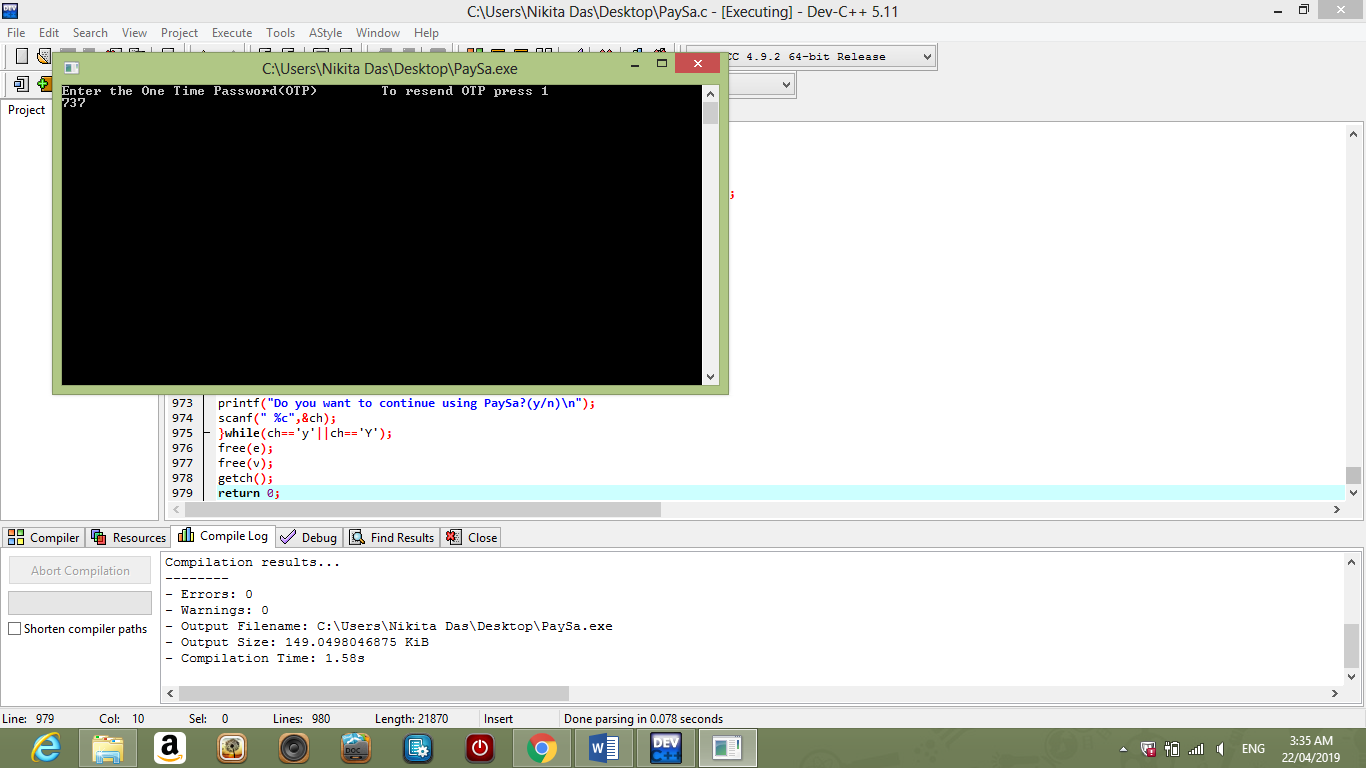
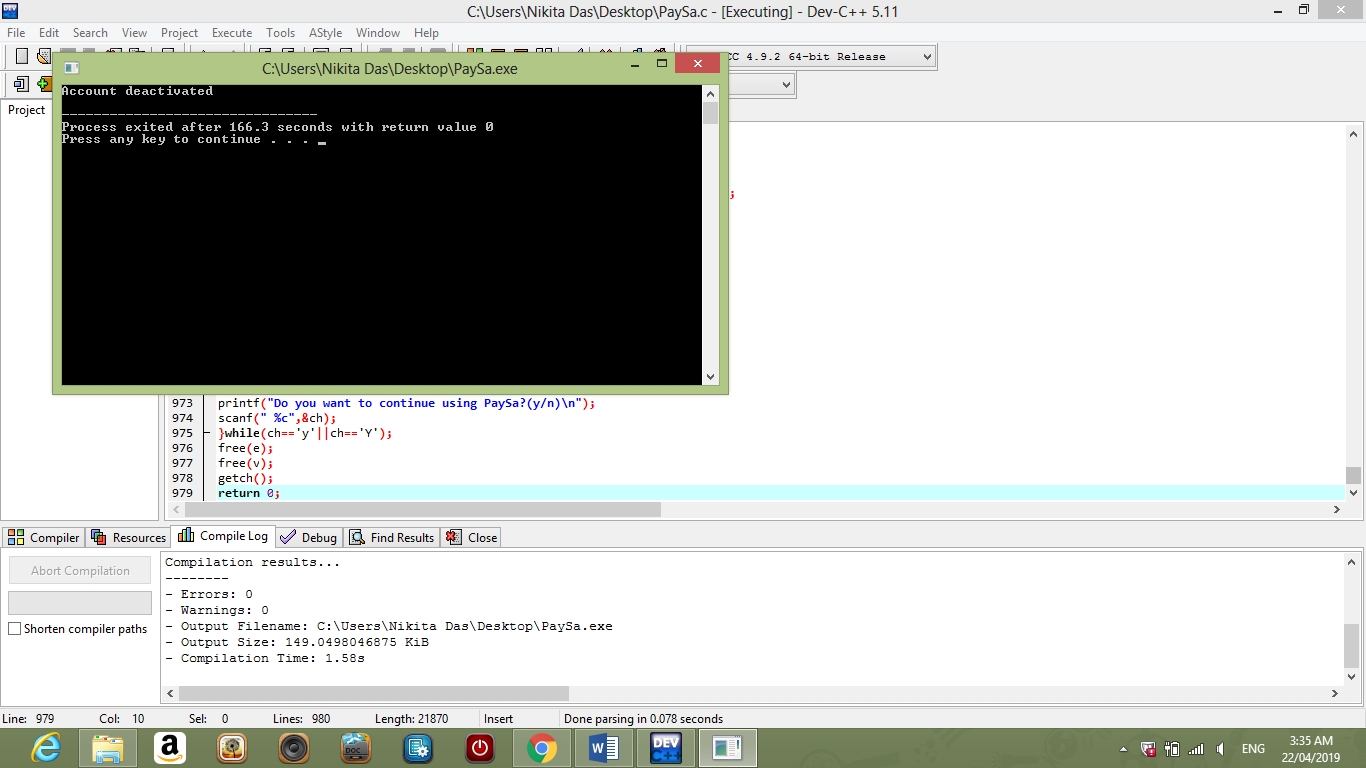
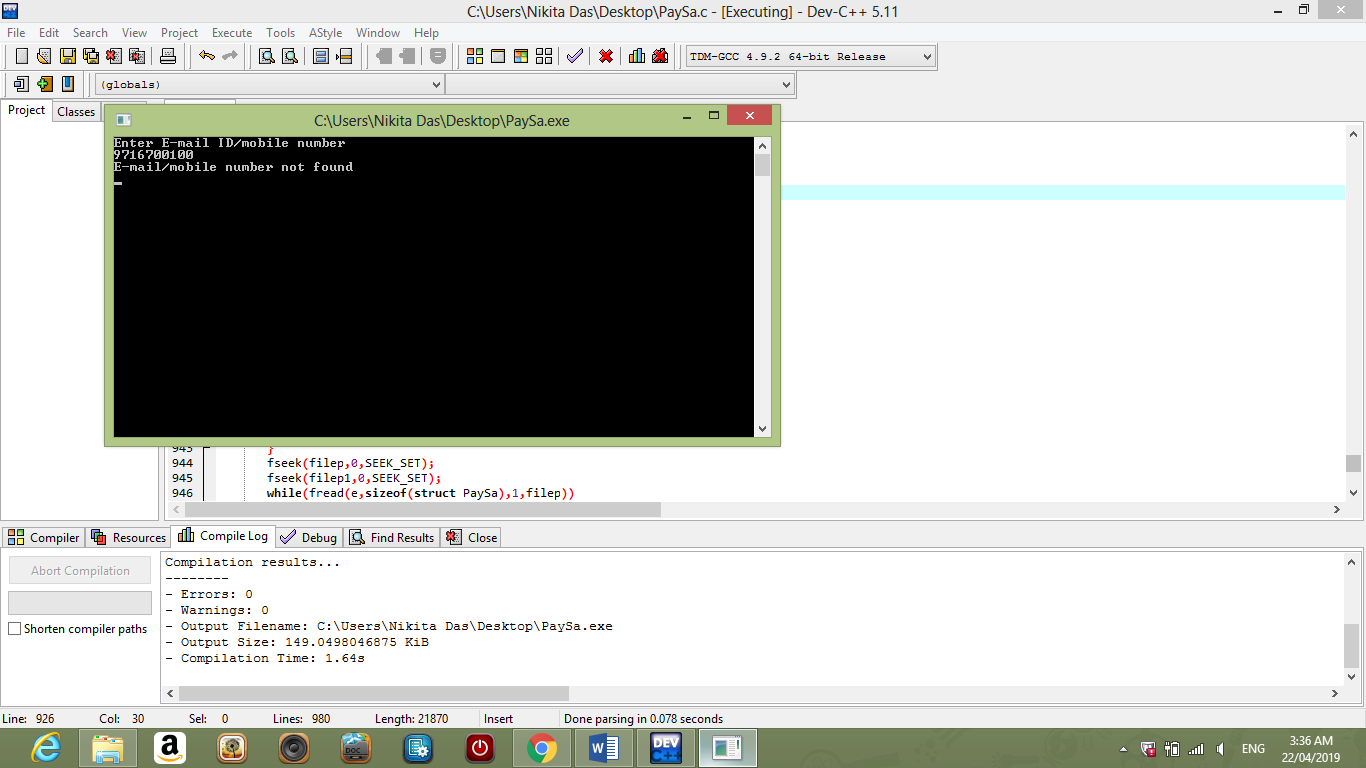
* **viewpb() -**
* **modify() –**
* **viewr() –**

****

* **updatebankbalance() –**

****

****

* **deactivate() -**