NATIONAL INSTITUTE OF TECHNOLOGY

KARNATAKA, SURATHKAL

CS202M

PROJECT NAME:

Student Hostel Mess Management System

SUBMITTED TO:

Prof. Annappa B

SUBMITTED BY:

Ashmita Das - 231EC208

Sahasra Pulumati - 231EE247

DETAILS

NAME: ASHMITA DAS

ROLL NO: 231EC208

PHONE NO: 7892120620

EMAIL ID: ashmitadas.231ec208@nitk.edu.in

NAME: SAHASRA PULUMATI

ROLL NO: 231EE247

PHONE NO: 8123900157

EMAIL ID: sahasrapulumati.231ee247@nitk.edu.in

Table Of Contents:

Abstract:

Student Hostel Mess Management System is a customised mess management system that efficiently manages and keeps a track of the students and their meals in different messes within the college. This system includes facilities to manage and update student details, their meal history and the mess menus. It can be used to replace the current inefficient method of using mess cards.

Student Hostel Mess Management System

1. DESIGN:

- **Student information management –**
  - Stores details of students such as Roll No, Reg No, Name, Mess name.
  - Stores Serial No, Reg No, Date and the Meal consumed such as Breakfast, Lunch, Snacks and Dinner.
  - During a meal, the Reg No of a student is read. This is used to update the Meal History.
  - Updating Student Mess Details (only accessible by Hostel Office) –
    - Insertion – New students join college: At the beginning of a new academic year, the batch of new students will be added to the database.
    - Deletion – Students leave college: At end of the year, the leaving batch will be removed from the database.
    - Mess change – Students' mess can be updated.
- **Verification –**
  - Ensures that no invalid registration number can be entered. When the Reg No is read for a student, it will check if it exists.
  - Similarly, it also checks whether the student already had the specified meal or not.
- **Multi-user access –**
  - Login page for different messes and hostel office – Each mess can only access the data of the students belonging to the respective mess. Further, they can update only their respective mess menus.

    The hostel office can access the features to update the Student Mess Details.
- **Menu management –**
  - Creating, updating, and deleting daily menus.
- **Graphical User Interface –**
  - Simple user interface for ease of operations.

2.IMPLEMENTATION

Data structures used:

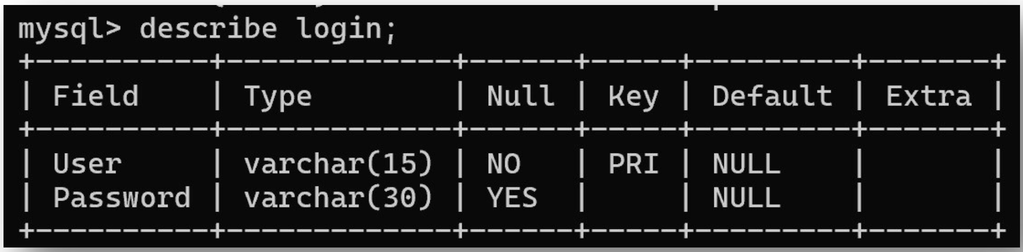
Linked list: stacks

Circular doubly linked list

Arrays

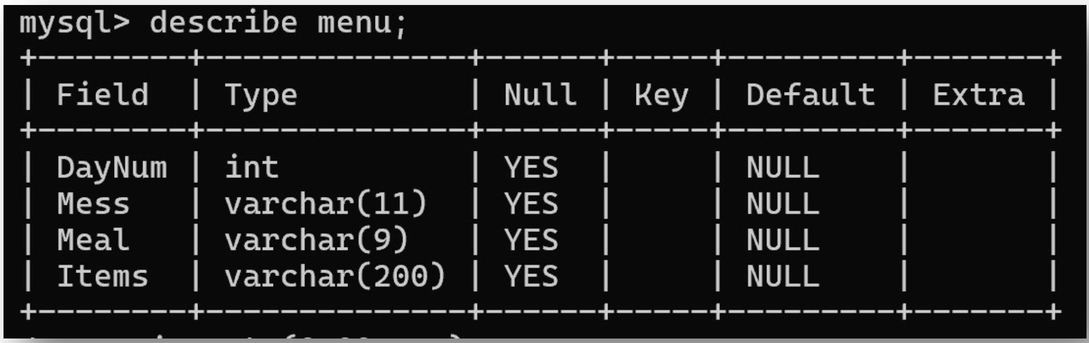
Tables used:

- Login:



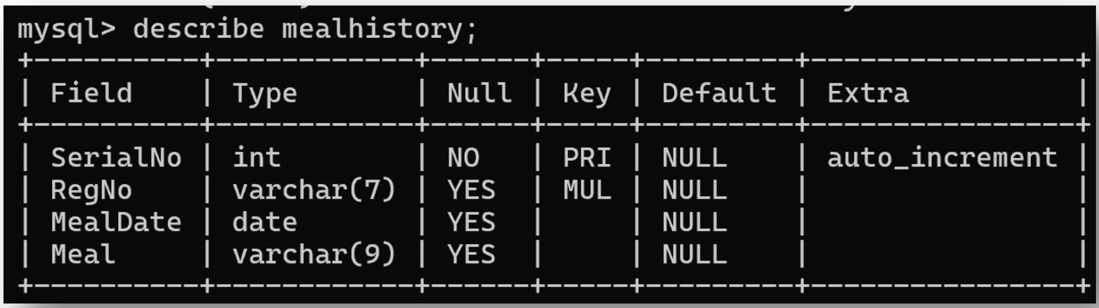
```
mysql> describe login;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| User     | varchar(15) | NO   | PRI | NULL    |       |
| Password | varchar(30) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

- Menu:



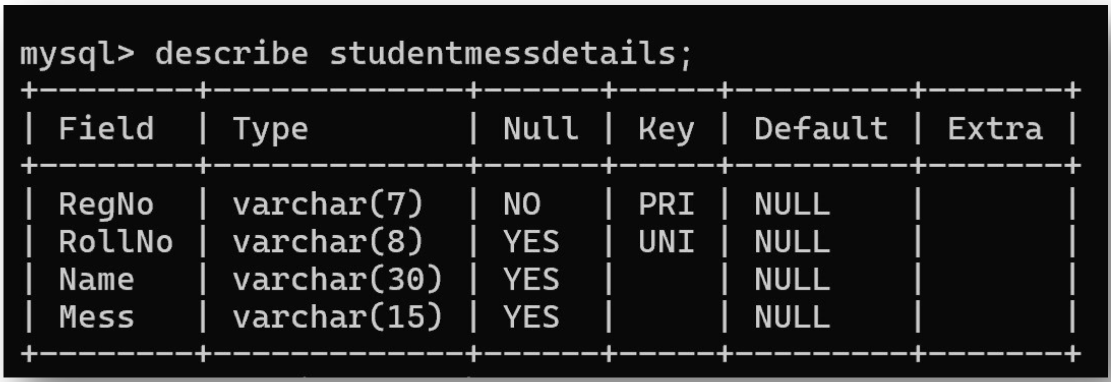
```
mysql> describe menu;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| DayNum | int          | YES  |     | NULL    |       |
| Mess   | varchar(11)  | YES  |     | NULL    |       |
| Meal   | varchar(9)   | YES  |     | NULL    |       |
| Items  | varchar(200) | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
```

- MealHistory:



```
mysql> describe mealhistory;
+----------+------------+------+-----+---------+----------------+
| Field    | Type       | Null | Key | Default | Extra          |
+----------+------------+------+-----+---------+----------------+
| SerialNo | int        | NO   | PRI | NULL    | auto_increment |
| RegNo    | varchar(7) | YES  | MUL | NULL    |                |
| MealDate | date       | YES  |     | NULL    |                |
| Meal     | varchar(9) | YES  |     | NULL    |                |
+----------+------------+------+-----+---------+----------------+
```

- StudentMessDetails:



```
mysql> describe studentmessdetails;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| RegNo  | varchar(7)  | NO   | PRI | NULL    |       |
| RollNo | varchar(8)  | YES  | UNI | NULL    |       |
| Name   | varchar(30) | YES  |     | NULL    |       |
| Mess   | varchar(15) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

LinkedList functions:

- Insertion:

  Inserts the node at the beginning.

- Deletion(data):

  Deletes the first node.

  Deletes the node with the specified data.

- Search(data):

  Finds the node with the specified data.

- SizeOfLL:

  Prints the number of nodes present in linked list.

- printLL:

  Prints the data present in a node of the linked list.

Linked list functions that interact with SQL:

dataLL class: (parent class: linked list)

- makeMessRegNoLL(user):

Selects the all the values in the column RegNo from the table StudentMessDetails where the user is as specified.

Inserts it at the beginning of the linked list, created using the class dataLL.

- makeMealCompletedLL (meal, date):

  Selects all the values in the column RegNo from the table MealHistory where, the MealDate and Meal are as specified.

  Inserts it at the beginning of the linked list, created using the class dataLL.

- writeStudentMessDetails:

  Inserts the data from the linked list into the StudentMessDetails table if there are no duplicate RegNo and RollNo.

  Checking for duplicates-

  Selects the count of rows where the RegNo, RollNo is the same as RegNo, RollNo in the current node of the linked list.

  If the count greater than one - duplicates exists, displays an error message.

  If no duplicates exist – inserts the values in the current node into the StudentMessDetails table and then deletes the current node.

- writeMealHistory:

  Inserts RegNo, MealDate, Meal from the linked list into the MealHistory table if there are no duplicate RegNo.

  Checking for duplicates-

  Selects the count of rows where the RegNo is the same as RegNo in the current node of the linked list.

  If the count greater than one - duplicates exists, displays an error message.

  If no duplicates exist – insert the values in the current node into the MealHistory table and then deletes the current node.

- removeFromStudentMessDetails:

  Removes the row in the table StudentMessDetails where the RegNo is same as the RegNo of the current node in the linked list and then deletes the current node.

- updateStudentMessDetails:

  Updates the mess in the table StudentMessDetails with the mess in the current node where the RegNo is same as the RegNo of the current node in the linked list and then deletes the current node.

CircularDoublyLinkedList:

- Insertion:
  Inserts the node at the beginning.
  Inserts the node at the end.
- Deletion(node):
  Deletes the specified node.

Circular doubly linked list functions that interact with SQL:

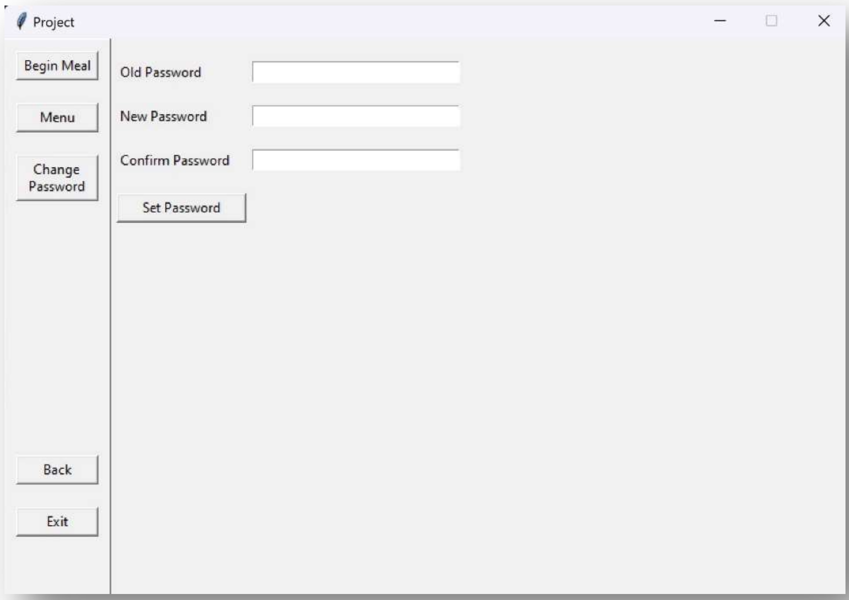
dataCDLL class: (parent class: CircularDoublyLinkedList)

- readMenu(user):
  Selects daynum, meal, items from Menu table where mess is as specified.
  Selects 4 consecutive rows from the Menu table (breakfast, lunch, snacks and dinner)
  Insert them into the linked list created using the dataCDLL.

- UpdateMenu (daynum, user, meal, items):
  Updates the items in the table Menu, where the daynum and mess are as specified, with the new items.

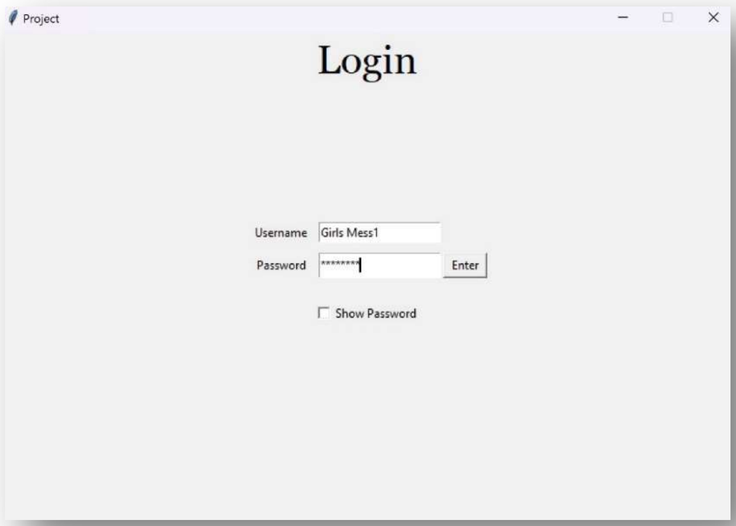
GUI:

- changePage(n):
  Helps to navigate between 3 pages namely - loginPage, messPage, hostelOfficePage
- changePassword(user):



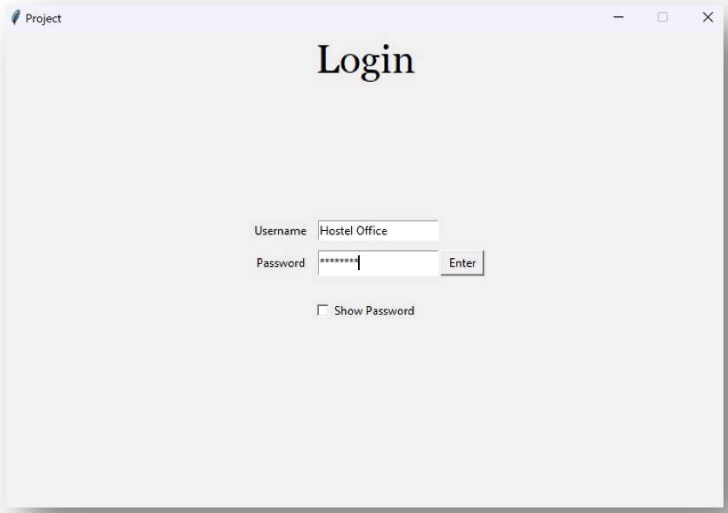
Allows the users to update their respective passwords.
Displays an error message if the old password doesn’t match or if the password exceeds 30 characters.

- Loginpage:
  - Mess login-



Username Girls Mess1
Password ********
Enter
Show Password

  - Hostel Office login-

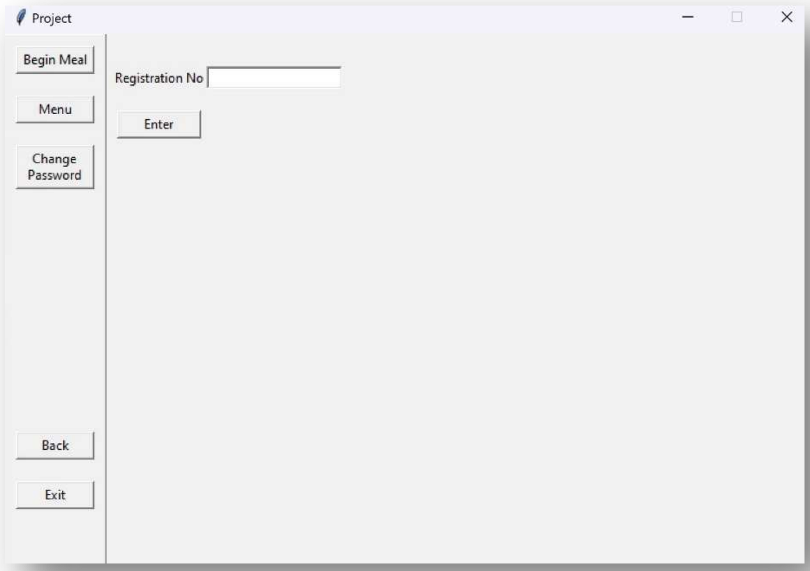


Username Hostel Office
Password ********
Enter
Show Password

Checks if the username matches with the corresponding password using the data stored in the login table.

Else an error message displayed.

- MessPage(user):
    - Meal:

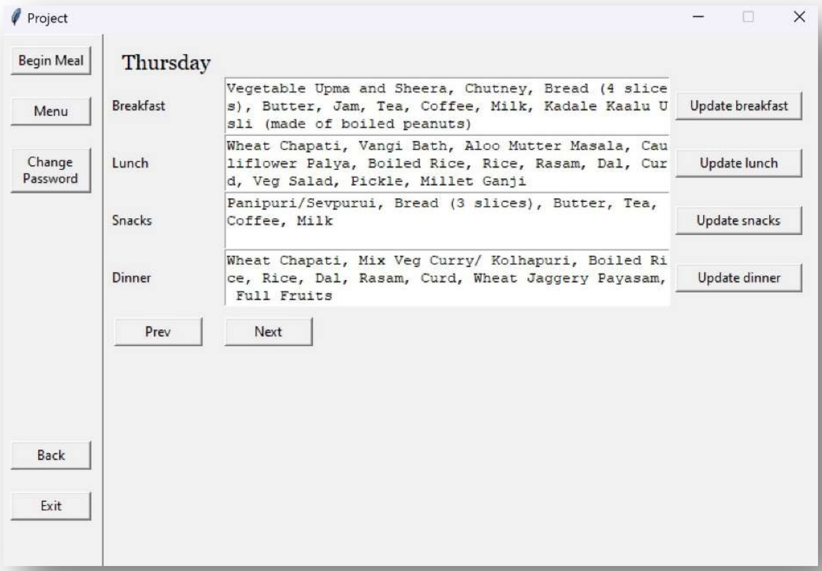


RegNos can be scanned only during the specified meal timings.

RegNo of students who have already eaten and those of students from other messes cannot be scanned and instead displays an error message.

The RegNos are added to the database once number of elements in the stack reaches 20 or when the meal is over.

- Menu:



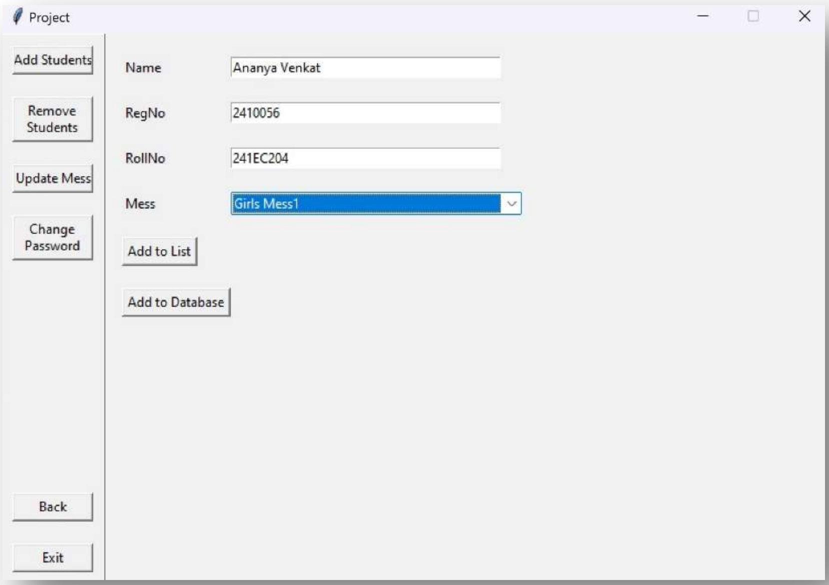
Thursday

Breakfast	Vegetable Upma and Sheera, Chutney, Bread (4 slices), Butter, Jam, Tea, Coffee, Milk, Kadale Kaalu Usli (made of boiled peanuts)	Update breakfast
Lunch	Wheat Chapati, Vangi Bath, Aloo Mutter Masala, Cauliflower Palya, Boiled Rice, Rice, Rasam, Dal, Curd, Veg Salad, Pickle, Millet Ganji	Update lunch
Snacks	Panipuri/Sevpurui, Bread (3 slices), Butter, Tea, Coffee, Milk	Update snacks
Dinner	Wheat Chapati, Mix Veg Curry/ Kolhapuri, Boiled Rice, Rice, Dal, Rasam, Curd, Wheat Jaggery Payasam, Full Fruits	Update dinner

Displays the menu of the mess using a doubly linked list.

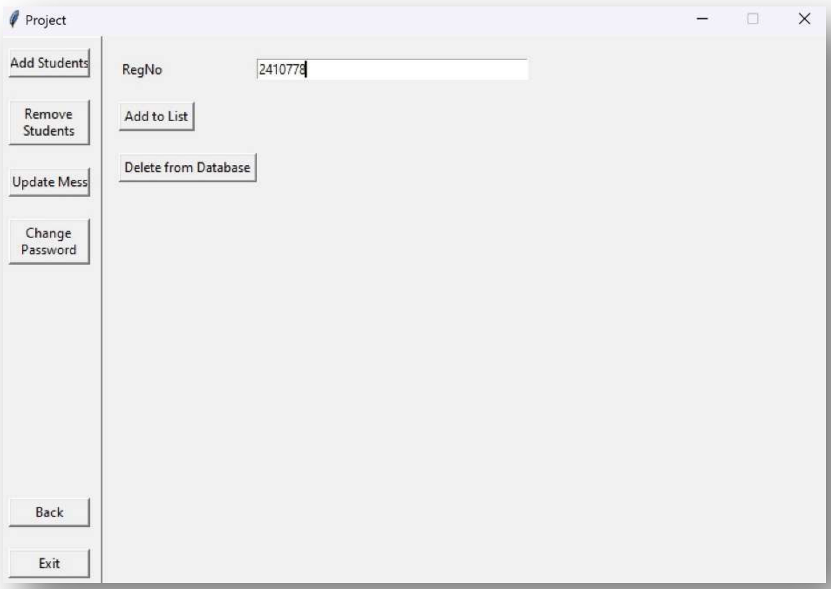
Updates the menu in the node of the linked list when called and then the changes are made into the menu table.

- HostelOfficePage:
    - AddStudent:



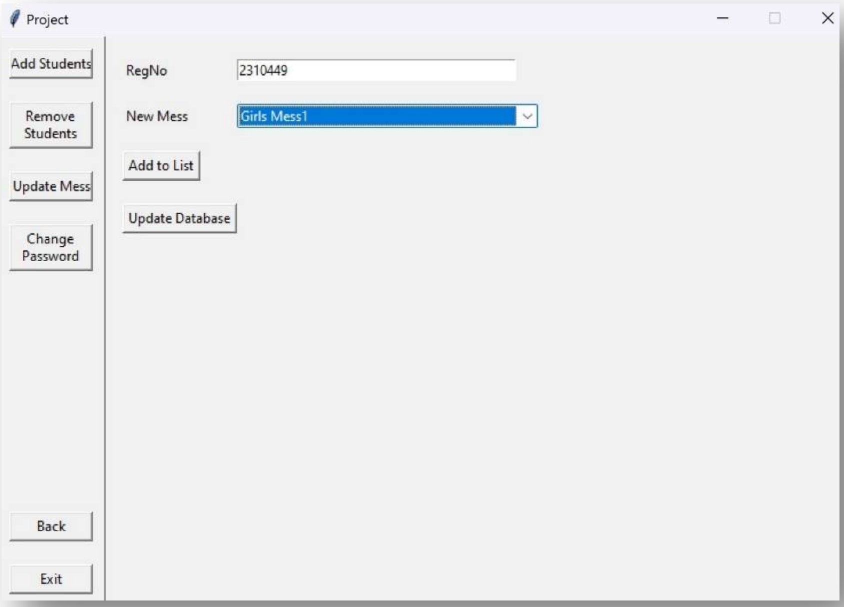
Name: Ananya Venkat
RegNo: 2410056
RollNo: 241EC204
Mess: Girls Mess1

- DeleteStudents:



RegNo 2410778

- Updatemess:



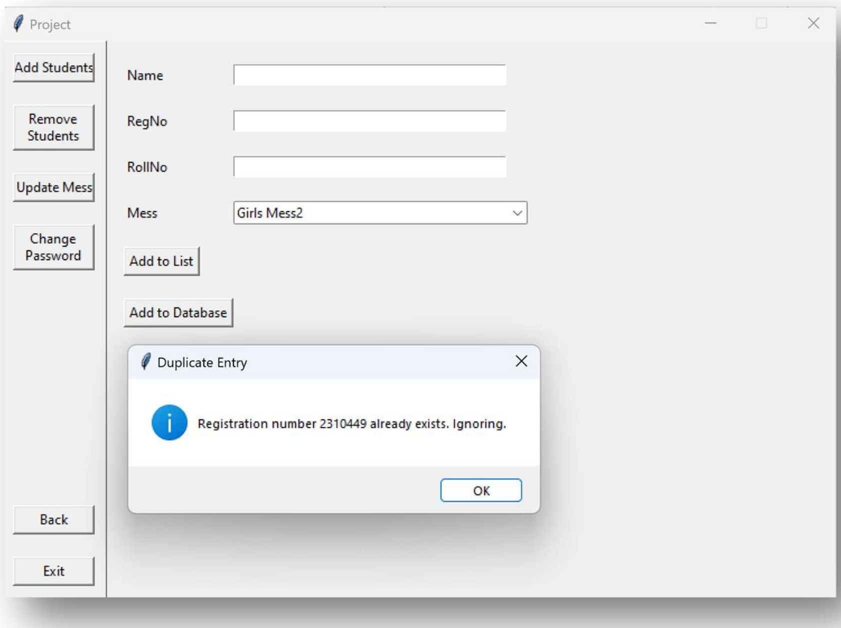
RegNo 2310449

New Mess Girls Mess1

3.TESTING:

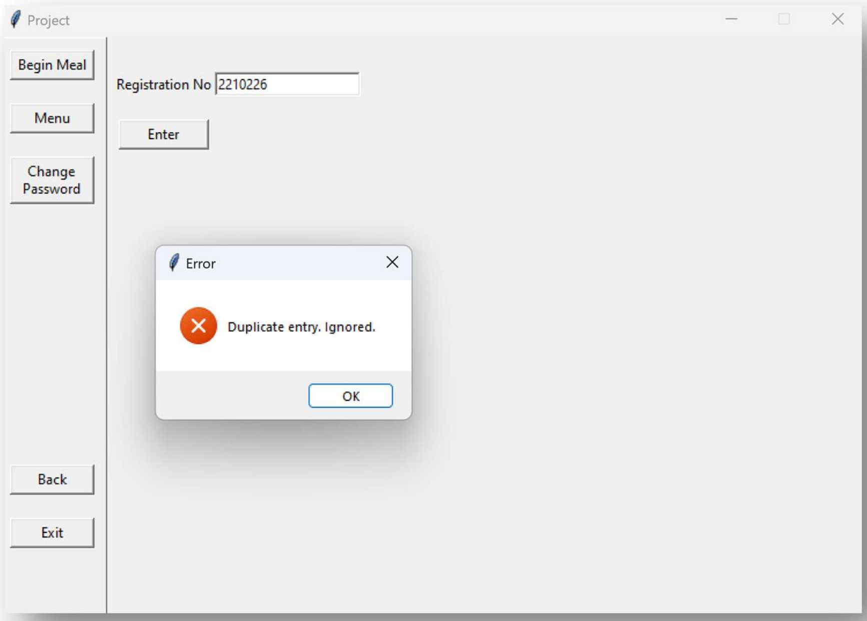
- **UNIQUE ENTRIES:**

While entering data in to StudentMessDetails table if an existing RegNo is added, an error message will be displayed.



Registration number 2310449 already exists. Ignoring.

While adding data into the MealHistory table the duplicate RegNos in the linked list will not be added and an error message will be displayed.


Project

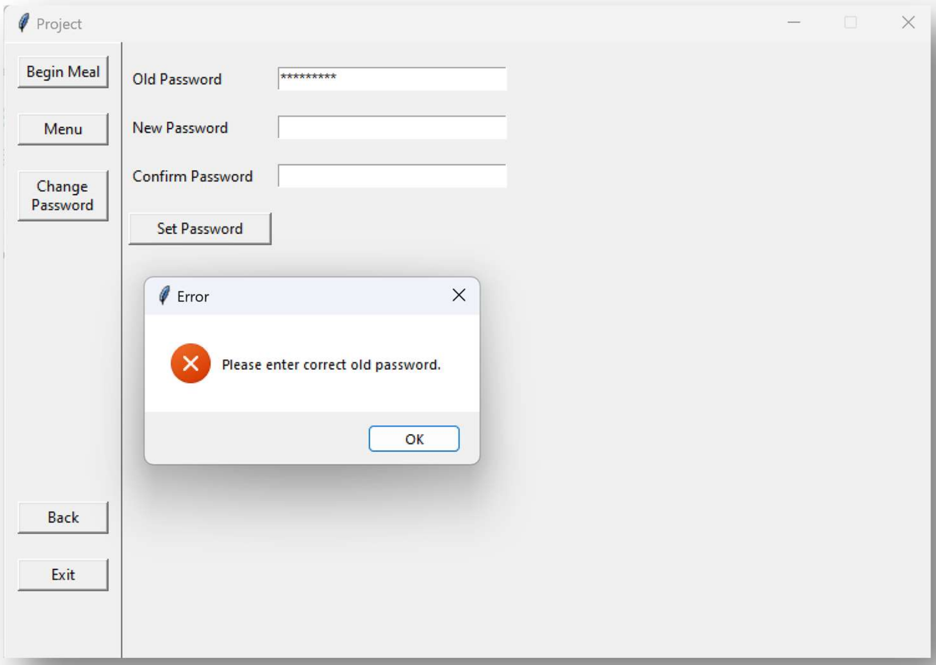
Registration No 2210226

Error

Duplicate entry. Ignored.

- **CHANGE PASSWORD:**

Error message when old password is incorrect.


Project

Old Password *********

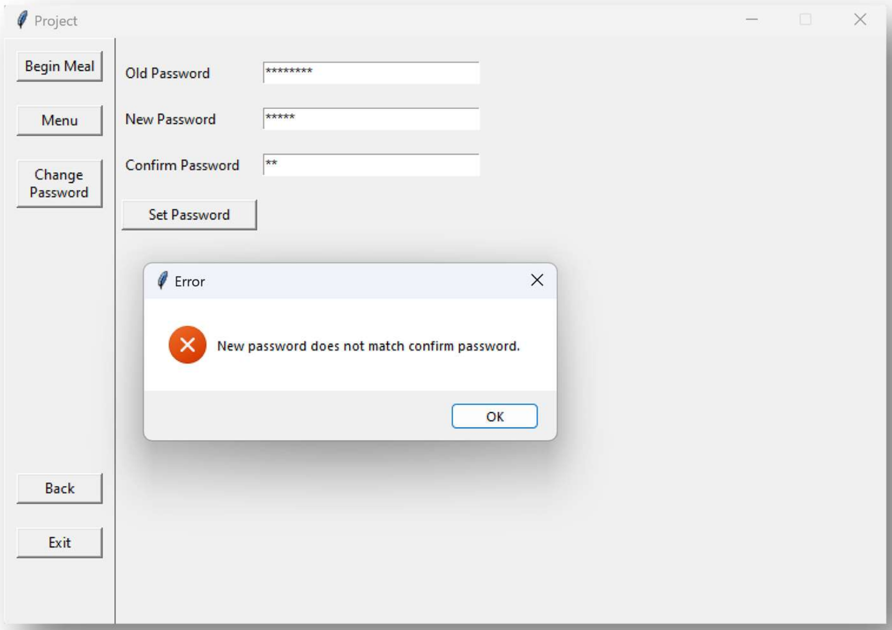
New Password

Confirm Password

Error

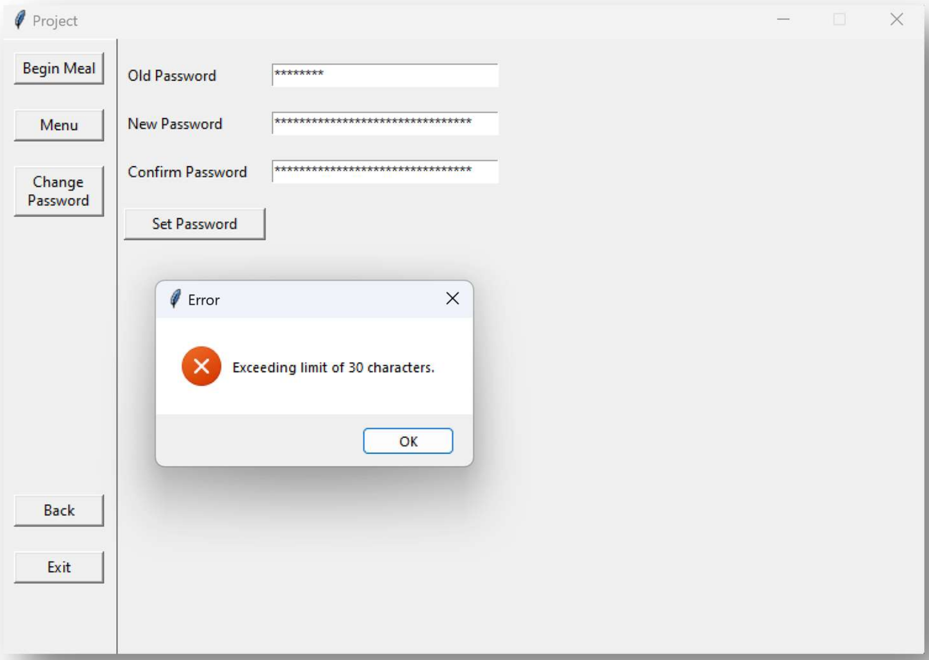
Please enter correct old password.

Error message if the new password doesn’t match the confirm password.



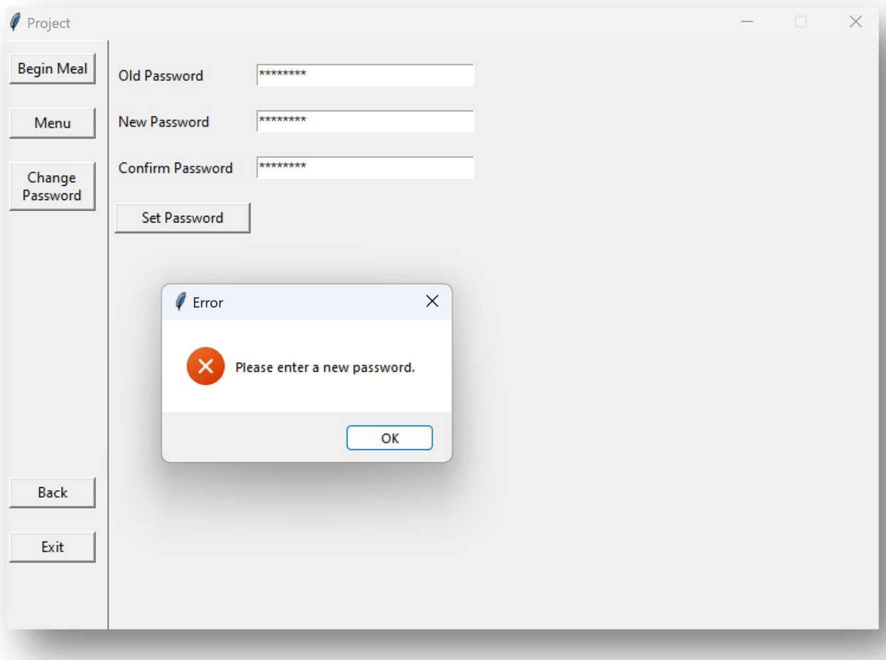
New password does not match confirm password.

Error message if the new password exceeds 30 characters.



Exceeding limit of 30 characters.

Error message if the new password is the same as the old password.



Old Password ********

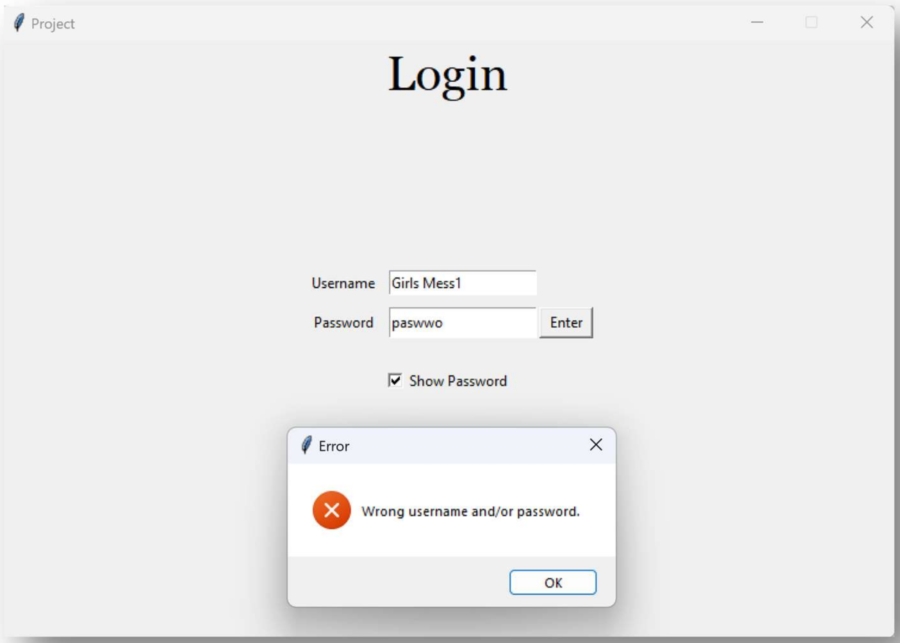
New Password ********

Confirm Password ********

Please enter a new password.

- **VALIDATION:**

Error message if incorrect password or username is entered.



Username Girls Mess1

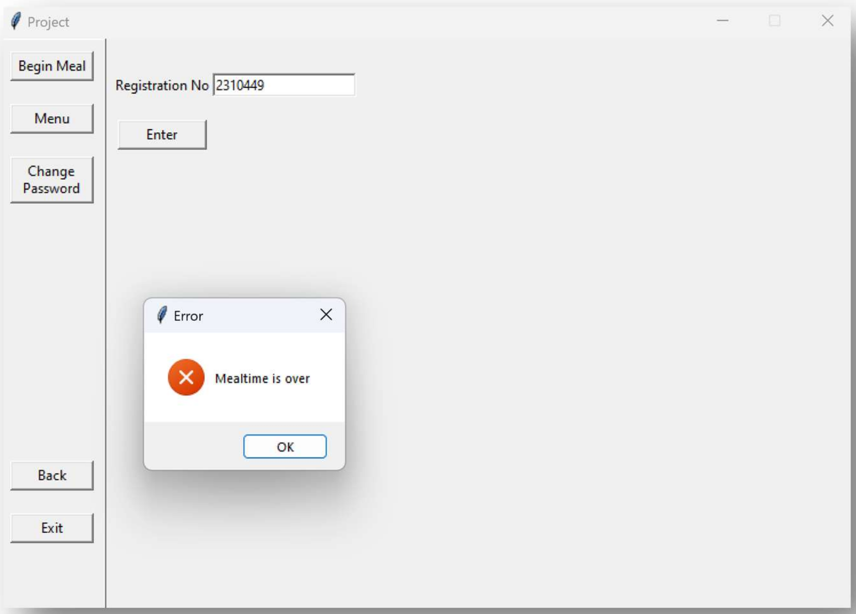
Password paswwo

☑ Show Password

Wrong username and/or password.

- **MEALTIME VALIDATION:**

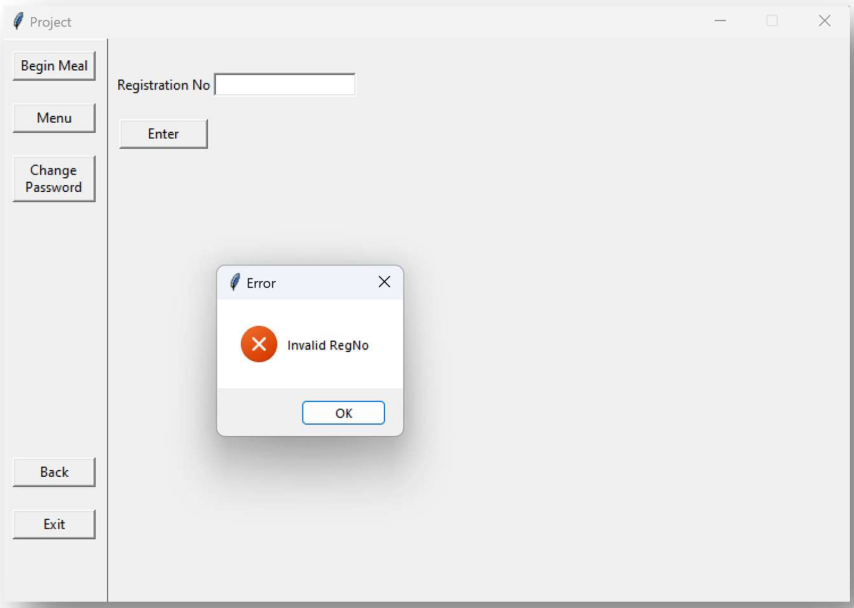
Error message if RegNo scanned outside meal timings.



Registration No 2310449

Mealtime is over

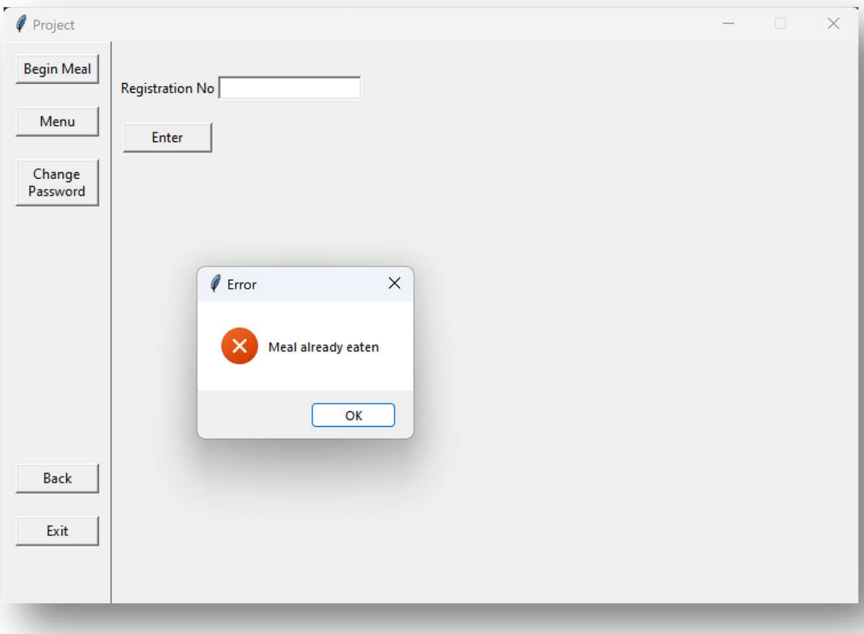
Error message if RegNo doesn’t exist in the mess.



Registration No

Invalid RegNo

Error message if the Regno has already had the respective meal.



Meal already eaten

CODE:

1. Creating Database using MySQL.

```
import mysql.connector as con

mycon = con.connect(host="localhost", user="root", passwd="mySQL123",
database="messdatabase")
if mycon.is_connected():
    print("Successfully connected to MySQL db")
else:
    print("Connection Unsuccessful")

cursor = mycon.cursor()

def execute(command):
    cursor.execute(command)

execute("Drop table MealHistory")
execute("Drop table StudentMessDetails")
execute("Drop table Menu")
execute("Drop table Login")
```

```
t1 = "CREATE TABLE StudentMessDetails(RegNo VARCHAR(7) PRIMARY KEY, RollNo VARCHAR(8)
UNIQUE, Name VARCHAR(30), Mess VARCHAR(15))"
t2 = "CREATE TABLE MealHistory(SerialNo INT AUTO_INCREMENT PRIMARY KEY, RegNo VARCHAR(7),
FOREIGN KEY (RegNo) REFERENCES StudentMessDetails(RegNo), MealDate DATE, Meal VARCHAR(9))"
t3 = "Create table Menu(DayNum INT, Mess VARCHAR(11), Meal VARCHAR(9), Items
VARCHAR(200))"
t4 = "CREATE TABLE Login(User VARCHAR(15) PRIMARY KEY, Password VARCHAR(30))"
execute(t1)
execute(t2)
execute(t3)
execute(t4)

studentInfo = ("('2210226', '221AI004', 'Swati M', 'Girls Mess1')",
               "('2310449', '231EC208', 'Ashmita Das', 'Girls Mess1')",
               "('2310465', '231EE247', 'Sahasra Pulumati', 'Girls Mess1')",
               "('2410226', '241MN023', 'Shreyas Kaluri', 'Boys Mess2')",
               "('2110006', '211CS107', 'Joshua Seth', 'Boys Mess1')")
mealInfo = ('("2210226", "2024-10-02", "Breakfast")',
            '("2310449","2024-10-03", "Lunch")')
loginInfo = ("('Girls Mess1', 'Password')",
             "('Girls Mess2', 'Password')",
             "('Boys Mess1', 'Password')",
             "('Boys Mess2', 'Password')",
             "('Hostel Office', 'Password')")

for i in studentInfo:
    x = 'insert into studentmessdetails values' + i
    execute(x)
for i in mealInfo:
    x = 'insert into mealhistory (RegNo, MealDate, Meal) values' + i
    execute(x)
for i in loginInfo:
    x = 'insert into login values' + i
    execute(x)


mess = ["Girls Mess1", "Girls Mess2", "Boys Mess1", "Boys Mess2"]
meals = ["Breakfast", "Lunch", "Snacks", "Dinner"]
for i in range(1,8,1):
    for j in mess:
        for k in meals:
            query = "Insert into Menu values ('{}','{}','{}','')".format(i,j,k)
            execute(query)

mycon.commit()
mycon.close()
```

2. Application Code

```
# Name: Ashmita Das
# Roll no: 231EC208
# Mobile no: 7892120620
# Email Id: ashmitadas.231ec208@nitk.edu.in

# Name: Sahasra Pulumati
# Roll no: 231EE247
# Mobile no: 8123900157
# Email Id: sahasrapulumati.231ee247@nitk.edu.in


from tkinter import *
from tkinter import messagebox, ttk
import mysql.connector as con
import datetime


# Node class for creating nodes in linked lists
class Node:
    def __init__(self, data):
        self.data = data
```

```python
        self.next = None


# Create a LinkedList class
class LinkedList:
    def __init__(self):
        self.head = None

    # Method to insert a node at the beginning of the linked list
    def insertAtBegin(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        else:
            new_node.next = self.head
            self.head = new_node

    # Method to remove first node of linked list
    def remove_first_node(self):
        if(self.head == None):
            return
        self.head = self.head.next

    # Method to delete the entire linked list
    def delete_entire_list(self):
        while self.head is not None:
            self.remove_first_node()  # Remove each node starting from the head

    # Method to search for a node with specific data and return True/False
    def search(self, data):
        current_node = self.head
        while current_node:
            if current_node.data == data:
                return True
            current_node = current_node.next
        return False


    # Method to return the size of linked list
    def sizeOfLL(self):
        size = 0
        if(self.head):
            current_node = self.head
            while(current_node):
                size = size+1
                current_node = current_node.next
            return size
        else:
            return 0

    # Method to print the linked list data
    def printLL(self):
        current_node = self.head
        while(current_node):
            print(current_node.data)
            current_node = current_node.next


# Double node class for creating nodes in a doubly linked list
class DoubleNode:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None


# Circular Doubly Linked List class
class CircularDoublyLinkedList:
    def __init__(self):
        self.head = None

    # Method to insert a node at the end of the list
    def insert_at_end(self, data):
        new_node = Node(data)
        if self.head is None:
            # List is empty, new_node points to itself
            self.head = new_node
            new_node.next = new_node
            new_node.prev = new_node
        else:
            tail = self.head.prev  # Last node in the list
            new_node.next = self.head  # New node's next is the head
            new_node.prev = tail  # New node's prev is the last node
            tail.next = new_node  # Last node's next is the new node
            self.head.prev = new_node  # Head's prev is the new node

    # Method to delete a specific node
    def delete_node(self, node):
        if self.head is None or node is None:
            return  # If the list is empty or node is None, do nothing

        if node == self.head and self.head.next == self.head:
            # Case when there is only one node in the list
            self.head = None
        else:
            prev_node = node.prev
            next_node = node.next
            prev_node.next = next_node
            next_node.prev = prev_node

            if node == self.head:
                self.head = next_node  # Update the head if the deleted node was the head

        node = None  # Delete the node


#Linked list object that interacts with sql
```

```python
class dataLL(LinkedList):
    def __init__(self):
        super().__init__()

    #Creates a linked list of students who belong to a specific mess(user)
    def makeMessRegNoLL(self,user):
        query = "Select RegNo from StudentMessDetails where Mess = '"+user+"'"
        cur.execute(query)
        regNos = cur.fetchall()
        for i in regNos:
            self.insertAtBegin(i[0])

    #Creates a linked list of all the students who already completed the current meal
    def makeMealCompletedLL(self,meal, date):
        query = "Select RegNo from MealHistory where MealDate='{}' and 
Meal='{}'".format(date,meal)
        cur.execute(query)
        regNos = cur.fetchall()
        for i in regNos:
            self.insertAtBegin(i[0])

    #Data from linked list of new students is added to StudentMessDetails, checks for 
duplicate RegNo and RollNo
    def writeStudentMessDetails(self):
        sql = "Insert into StudentMessDetails values (%s, %s, %s, %s)"
        current_node = self.head

        while current_node:
            row = current_node.data

            # Check for duplicate registration number
            check_reg = "SELECT COUNT(*) FROM StudentMessDetails WHERE RegNo = %s"
            cur.execute(check_reg, (row[0],))
            reg_exists = cur.fetchone()[0] > 0

            # Check for duplicate roll number
            check_roll = "SELECT COUNT(*) FROM StudentMessDetails WHERE RollNo = %s"
            cur.execute(check_roll, (row[1],))
            roll_exists = cur.fetchone()[0] > 0

            if reg_exists:
                messagebox.showinfo("Duplicate Entry", f"Registration number {row[0]} 
already exists. Ignoring.")
            elif roll_exists:
                messagebox.showinfo("Duplicate Entry", f"Roll number {row[1]} already 
exists. Ignoring.")
            else:
                cur.execute(sql, row)  # Only insert if no duplicates found

            # Remove node after processing the current one
            self.remove_first_node()
            current_node = current_node.next  # Move to the next node

        mycon.commit()  # Commit once after processing all records
    #Data from linked list of students who have eaten the current meal is added to
MealHistory table
    def writeMealHistory(self):
        sql = "Insert into MealHistory (RegNo, MealDate, Meal) values " #('','','','')
        current_node = self.head
        while current_node:
            row = current_node.data

            # Check for duplicate registration number
            check_reg = "SELECT COUNT(*) FROM MealHistory WHERE RegNo = '{}' and MealDate
= '{}' and Meal = '{}'".format(row[0], row[1], row[2])
            cur.execute(check_reg)
            reg_exists = cur.fetchone()[0] > 0
            if reg_exists == 0:
                rowText = "('{}','{}','{}')".format(row[0],row[1],row[2])
                cur.execute(sql+rowText)
            self.remove_first_node()
            current_node = current_node.next
        mycon.commit()

    def removeFromStudentMessDetails(self):
        current_node = self.head
        while current_node:
            delSQL = "DELETE FROM StudentMessDetails WHERE RegNo =
'{}'".format(current_node.data)
            cur.execute(delSQL)
            self.remove_first_node()
            current_node = current_node.next
        mycon.commit()

    def updateStudentMessDetails(self):
        current_node = self.head
        while current_node:
            updateSQL = "UPDATE StudentMessDetails SET mess = '{}' WHERE RegNo =
'{}'".format(current_node.data[1],current_node.data[0])
            cur.execute(updateSQL)
            self.remove_first_node()
            current_node = current_node.next
        mycon.commit()

#Circular doubly linked list object that interacts with sql
class dataCDLL(CircularDoublyLinkedList):
    def __init__(self):
        super().__init__()

    #Creates a doubly circular linked list of the menus of a week of the specified
mess(user)
    def readMenu(self, user):
        query = "Select daynum, meal, items from Menu where mess='{}'".format(user)
        cur.execute(query)
        menuRows = cur.fetchall()
        for i in range(0,len(menuRows),4):
            dayList = [menuRows[i], menuRows[i+1], menuRows[i+2],menuRows[i+3]]
            self.insert_at_end(dayList)

    def updateMenu(self,dayNum, user, meal, items):
        updatsql="UPDATE Menu set items='{}' where daynum={} and mess='{}' and 
meal='{}'".format(items,dayNum,user,meal)
        cur.execute(updatsql)
        mycon.commit()

# Used in making mess drop down box
def makeMessArray():
    query = "Select user from login where user != 'Hostel Office'"
    cur.execute(query)
    users = cur.fetchall()
    messArray = []
    for i in users:
        messArray.append(i[0])
    return messArray

#Changing the mainframes, called in next and back
def changePage(n, user="none"):
    for widgets in root.winfo_children():
        widgets.destroy()

    if n == 1:
        loginPage()

    elif n == 2 and user != "Hostel Office":
        messPage(user)

    elif n == 2 and user == "Hostel Office":
        hostelOfficePage()

def exitProgram():
    root.destroy()

def changePassword(user, actFrm):

    def set_pass():
        query = "Select password from login where user='{}'".format(user)
        cur.execute(query)
        old = cur.fetchone()[0]
        if old != oldPass.get():
            messagebox.showerror('Error', 'Please enter correct old password.')
        elif newPass.get() != confirmPass.get():
            messagebox.showerror('Error', 'New password does not match confirm password.')
        elif (len(newPass.get()) > 20):
            messagebox.showerror('Error', 'Exceeding limit of 30 characters.')
        elif (len(newPass.get()) == 0):
            messagebox.showerror('Error', 'Enter new password,')
        else:
            update_pass = "Update login set password = '{}' where 
user='{}'".format(newPass.get(), user)
            cur.execute(update_pass)
            mycon.commit()
            messagebox.showinfo('Password', 'Password update is successful')
            oldPass.delete(0,END)
            newPass.delete(0,END)
            confirmPass.delete(0,END)

    Label(actFrm, text='Old Password').grid(row=0, column=0, sticky='w')
    Label(actFrm, text='New Password').grid(row=1, column=0, sticky='w')
    Label(actFrm, text='Confirm Password').grid(row=2, column=0, sticky='w')

    oldPass = Entry(actFrm, width=30, show='*')
    oldPass.grid(row=0, column=1, sticky='w', padx=5, pady=10)
    newPass = Entry(actFrm, width=30, show='*')
    newPass.grid(row=1, column=1, sticky='w', padx=5, pady=10)
    confirmPass = Entry(actFrm, width=30, show='*')
    confirmPass.grid(row=2, column=1, sticky='w', padx=5, pady=10)

    Button(actFrm, text='Set Password', command=lambda: set_pass(), width=15).grid(row=3,
column=0, pady=10, sticky='w')

def loginPage():
    F1 = Frame(root)
    F1.rowconfigure(0, weight=1, minsize=100)
    F1.rowconfigure(1, weight=1, minsize=150)
    F1.rowconfigure(2, weight=1, minsize=70)
    F1.rowconfigure(3, weight=1, minsize=180)
    F1.columnconfigure(0, weight=1, minsize=750)

    titleFrm = Frame(master=F1, borderwidth=1)
    Label(master=titleFrm, text='Login', font=('Georgia', 30)).pack()
    titleFrm.grid(row=0, column=0, sticky='nsew')

    passFrm = Frame(master=F1, borderwidth=1)
    Label(master=passFrm, text='Username', width=10).grid(row=0, column=0, sticky='nse',
pady=10)
    usernmTxt = Entry(master=passFrm, width=20)
    Label(master=passFrm, text='Password', width=10).grid(row=1, column=0, sticky='nse')
    passwdTxt = Entry(master=passFrm, width=20, show='*')
    usernmTxt.grid(row=0, column=1, sticky='nsw', pady=10)
    passwdTxt.grid(row=1, column=1, sticky='nsw')
    passFrm.grid(row=1, column=0, sticky='s')

    #  Checks if entered username and password are correct
    def checkPass():
        UN = usernmTxt.get()
        PW = passwdTxt.get()
        cur.execute('Select user, password from login')
        loginInfo = cur.fetchall()
        for i in loginInfo:
            if i[0] == UN and i[1] == PW:
                changePage(2, UN)
                break
        else:
            messagebox.showerror("Error", 'Wrong username and/or password.')
    Button(master=passFrm, text='Enter', width=5, command=checkPass).grid(row=1, column=2,
sticky='nsw', padx=2)

    #  Show password checkbox
    chkFrm = Frame(F1, borderwidth=1)
    chkFrm.grid(row=2, column=0, pady=5)

    def show():
        if var.get() == 1:
            passwdTxt.config(show='')
        else:
            passwdTxt.config(show='*')

    var = IntVar()
    Checkbutton(chkFrm, text='Show Password', variable=var, command=show).pack()
    F1.pack()

# Three sub pages:
# Meal - Takes student RegNo for every meal and validates it
# Menu - Displays each day's menu, provides function to update it
# Password - Change password
def messPage(user):
    F2 = Frame(root)
    F2.rowconfigure(0, weight=1, minsize=500)
    F2.columnconfigure(0, weight=1, minsize=100)
    F2.columnconfigure(1, weight=1, minsize=650)

    # Options frame
    optFrm = Frame(F2, bd=1, relief=RAISED)
    # Activity frame
    actFrm = Frame(F2)

    # List of students belonging to specified mess
    validStudentsLL=dataLL()

    # Helper function to write to database and then execute a given function: Meal, Menu,
Password, Back, Exit
    def execute_with_db_update(action):
        validStudentsLL.writeMealHistory()
        action()

    def clear_actFrm():
        for widgets in actFrm.winfo_children():
            widgets.destroy()

    def Meal():
        clear_actFrm()

        # Checks if the RegNo is valid, entered within the meal timings and has already
eaten the current meal or not
        def validateRegNo():
            meal = "invalid"
            currentTime = datetime.datetime.now()
            hour= int(currentTime.strftime("%H"))
            minutes=int(currentTime.strftime("%M"))
            date = currentTime.strftime('%Y-%m-%d')
            time_in_minutes = hour * 60 + minutes

            if 420 <=time_in_minutes <=600:
                meal = "Breakfast"
            elif 720 <= time_in_minutes <= 840:
                meal = "Lunch"
            elif 990 <= time_in_minutes < 1080:
                meal = "Snacks"
            elif 1170 <= time_in_minutes <= 1320:
                meal = "Dinner"

            #Updates the database if list size is 20 or the meal timing is over
            if validStudentsLL.sizeOfLL() == 20 or meal == "invalid":
                validStudentsLL.writeMealHistory()

            if meal == "invalid":
                messagebox.showerror("Error", 'Mealtime is over')
                return

            # Checks if student is registered to the mess
            regNo = regNoInput.get()
            regNoInput.delete(0,END)
            if not (studentsLL.search(regNo)):
                messagebox.showerror("Error", 'Invalid RegNo')
                return

            # Checks if the student has eaten the current meal or not
            alreadyEatenLL = dataLL()
            alreadyEatenLL.makeMealCompletedLL(meal,date)

            if not (alreadyEatenLL.search(regNo)):
                validStudentsLL.insertAtBegin([regNo, date, meal])
            else:
                messagebox.showerror("Error", 'Meal already eaten')


        studentsLL = dataLL()
        studentsLL.makeMessRegNoLL(user)

        Label(actFrm, text="Registration No").grid(row=0, column=0, sticky='w')
        regNoInput = Entry(actFrm, width=20, borderwidth=2)
        regNoInput.grid(row=0, column=1, pady=20, sticky='ew')
        Button(actFrm, text='Enter', command=lambda: validateRegNo(), width=9).grid(row=1,
column=0, padx=5, sticky='ew')


    def Menu():
        clear_actFrm()
        days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday"]

        def getInfoFromNode(Node):
```

```
        menuInfo = Node.data
        dayNum = menuInfo[0][0]
        day = days[dayNum-1]
        return [day, menuInfo]

    def updateMenuDisplay(node):
        nonlocal menuInfo
        menuInfo = getInfoFromNode(node)
        day_label.config(text=menuInfo[0])   # Update day

        #Update Items
        breakfast_items_text.delete(1.0, END)
        lunch_items_text.delete(1.0, END)
        snacks_items_text.delete(1.0, END)
        dinner_items_text.delete(1.0, END)

        breakfast_items_text.insert('1.0',menuInfo[1][0][2])
        lunch_items_text.insert('1.0',menuInfo[1][1][2])
        snacks_items_text.insert('1.0',menuInfo[1][2][2])
        dinner_items_text.insert('1.0',menuInfo[1][3][2])

    def nextDayMenu():
        nonlocal currentNode  # Access the currentNode from outer scope
        currentNode = currentNode.next
        updateMenuDisplay(currentNode)

    def prevDayMenu():
        nonlocal currentNode
        currentNode = currentNode.prev
        updateMenuDisplay(currentNode)

    def updateMenuInfo(meal):
        updatedItems = ""
        mealName = ""
        if meal == 0:
            updatedItems = breakfast_items_text.get('1.0', "end-1c")
            mealName = "Breakfast"
        elif meal == 1:
            updatedItems = lunch_items_text.get('1.0', "end-1c")
            mealName = "Lunch"
        elif meal == 2:
            updatedItems = snacks_items_text.get('1.0', "end-1c")
            mealName = "Snacks"
        elif meal == 3:
            updatedItems = dinner_items_text.get('1.0', "end-1c")
            mealName = "Dinner"

        dayNum = days.index(menuInfo[0])+1
        menuInfo[1][meal] = (dayNum, mealName, updatedItems)
        messMenuLL.updateMenu(dayNum,user,mealName,updatedItems)

    messMenuLL=dataCDLL()
    messMenuLL.readMenu(user)
```

```
        currentNode = messMenuLL.head

        menuInfo = getInfoFromNode(currentNode)

        # Create and display labels with menu info
        day_label = Label(actFrm, text=menuInfo[0], font=('Georgia', 14), width=9)
        day_label.grid(row=0, column=0, sticky='w')

        Label(actFrm, text="Breakfast").grid(row=1, column=0, sticky='w')
        Label(actFrm, text="Lunch").grid(row=2, column=0, sticky='w')
        Label(actFrm, text="Snacks").grid(row=3, column=0, sticky='w')
        Label(actFrm, text="Dinner").grid(row=4, column=0, sticky='w')

        breakfast_items_text = Text(actFrm, height=3, width=50)
        breakfast_items_text.insert('1.0',menuInfo[1][0][2])#,text=menuInfo[1][0][2])
        breakfast_items_text.grid(row=1, column=1, sticky='w')

        lunch_items_text = Text(actFrm, height=3, width=50)#text=menuInfo[1][1][2])
        lunch_items_text.insert('1.0',menuInfo[1][1][2])
        lunch_items_text.grid(row=2, column=1, sticky='w')

        snacks_items_text = Text(actFrm, height=3, width=50)#text=menuInfo[1][2][2])
        snacks_items_text.insert('1.0',menuInfo[1][2][2])
        snacks_items_text.grid(row=3, column=1, sticky='w')

        dinner_items_text = Text(actFrm, height=3, width=50) #text=menuInfo[1][3][2])
        dinner_items_text.insert('1.0',menuInfo[1][3][2])
        dinner_items_text.grid(row=4, column=1, sticky='w')

        Button(actFrm, text='Update breakfast', command=lambda: updateMenuInfo(0),
width=15).grid(row=1, column=2, sticky='w',pady=10, padx=5)
        Button(actFrm, text='Update lunch', command=lambda: updateMenuInfo(1),
width=15).grid(row=2, column=2, sticky='w',pady=10, padx=5)
        Button(actFrm, text='Update snacks', command=lambda: updateMenuInfo(2),
width=15).grid(row=3, column=2, sticky='w',pady=10, padx=5)
        Button(actFrm, text='Update dinner', command=lambda: updateMenuInfo(3),
width=15).grid(row=4, column=2, sticky='w',pady=10, padx=5)

        # Create Next and Previous buttons with corresponding commands
        Button(actFrm, text='Next', command=nextDayMenu, width=10).grid(row=5, column=1,
sticky='w', pady=10)
        Button(actFrm, text='Prev', command=prevDayMenu, width=10).grid(row=5, column=0,
sticky='w', pady=10, padx=5)

    def Password():
        clear_actFrm()
        changePassword(user,actFrm)

    actFrm.grid(row=0, column=1, sticky='nws', ipadx=50, pady=10)

    Button(optFrm, text='Begin Meal', command=lambda: execute_with_db_update(Meal),
width=9).grid(row=0, column=0, padx=10, pady=10)
    Button(optFrm, text='Menu', command=lambda: execute_with_db_update(Menu),
width=9).grid(row=1, column=0, pady=10)
```

```
    Button(optFrm, text='Change Password', command=lambda:
execute_with_db_update(Password), width=9, wraplength=90).grid(row=2, column=0, pady=10)

    bottomFrm = Frame(optFrm)
    Button(bottomFrm, text='Back', command=lambda: execute_with_db_update(changePage(1)),
width=9).grid(row=1, column=0, sticky='s', pady=10)
    Button(bottomFrm, text='Exit', command=lambda: execute_with_db_update(exitProgram),
width=9).grid(row=2, column=0, sticky='s', pady=10)

    bottomFrm.grid(row=4, column=0, sticky='s', pady=205)
    optFrm.grid(row=0, column=0, sticky='nsw')

    F2.pack()

# Four sub pages:
# AddStudents - Add new students and their details to the database
# RemoveStudents - Remove students from the database
# UpdateMess - Change the mess
# Password - Change password
def hostelOfficePage():
    F3 = Frame(root)
    F3.rowconfigure(0, weight=1, minsize=500)
    F3.columnconfigure(0, weight=1, minsize=100)
    F3.columnconfigure(1, weight=1, minsize=650)

    # Contains buttons
    optFrm = Frame(F3, bd=1, relief=RAISED)
    # Activity frame
    actFrm = Frame(F3)

    def clear_actFrm():
        for widgets in actFrm.winfo_children():
            widgets.destroy()

    # List used for drop down box
    messArray = makeMessArray()


    def AddStudents():
        newStudentsList = dataLL()
        clear_actFrm()
        Label(actFrm, text="Name").grid(row=0, column=0, sticky='w', pady=10)
        nameTxt = Entry(actFrm, width=40)
        Label(actFrm, text="RegNo").grid(row=1, column=0, sticky='w', pady=10)
        regNoTxt = Entry(actFrm, width=40)
        Label(actFrm, text="RollNo").grid(row=2, column=0, sticky='w', pady=10)
        rollNoTxt = Entry(actFrm, width=40)
        Label(actFrm, text="Mess").grid(row=3, column=0, sticky='w', pady=10)
        messTxt = ttk.Combobox(actFrm, width=40, values=messArray, state="readonly")
        nameTxt.grid(row=0, column=1, sticky='w')
        regNoTxt.grid(row=1, column=1, sticky='w')
        rollNoTxt.grid(row=2, column=1, sticky='w')
        messTxt.grid(row=3, column=1, sticky='w')
        def clear_text():
            nameTxt.delete(0, END)
            regNoTxt.delete(0, END)
            rollNoTxt.delete(0, END)
            messTxt.delete(0, END)

        #Add valid students to newStudentsList
        def addToLL():
            if (regNoTxt.get() != '' and rollNoTxt.get() != '' and nameTxt.get() != '' and
messTxt.get() != ''):
                llist = [regNoTxt.get(),rollNoTxt.get(),nameTxt.get(),messTxt.get()]
                newStudentsList.insertAtBegin(llist)
                newStudentsList.printLL()
            clear_text()

        def addToDB():
            addToLL()
            newStudentsList.writeStudentMessDetails()
            clear_text()

        Button(actFrm, text="Add to List", command=lambda: addToLL()).grid(row=4,
column=0, sticky='w', pady=10)
        Button(actFrm, text="Add to Database", command=lambda: addToDB()).grid(row=5,
column=0, sticky='w', pady=10)


    def RemoveStudents():
        toRemoveLL = dataLL()
        clear_actFrm()
        Label(actFrm, text="RegNo").grid(row=0, column=0, sticky='w')
        regNoTxt = Entry(actFrm, width=40)
        regNoTxt.grid(row=0, column=1, sticky='w', pady=10)
        def clear_text():
            regNoTxt.delete(0, END)

        def addToLL():
            if regNoTxt.get() != '':
                regNo = regNoTxt.get()
                toRemoveLL.insertAtBegin(regNo)
            clear_text()

        def removeFromDB():
            addToLL()
            toRemoveLL.removeFromStudentMessDetails()
            clear_text()

        Button(actFrm, text="Add to List", command=lambda: addToLL()).grid(row=1,
column=0, sticky='w',pady=10)
        Button(actFrm, text="Delete from Database", command=lambda:
removeFromDB()).grid(row=2, column=0, sticky='w',pady=10)

    def UpdateMess():
        toUpdateLL = dataLL()
        clear_actFrm()
        Label(actFrm, text="RegNo").grid(row=0, column=0, sticky='w')
        Label(actFrm, text="New Mess").grid(row=1, column=0, sticky='w',pady=10)
        regNoTxt = Entry(actFrm, width=40)
        newmessTxt = ttk.Combobox(actFrm, width=40, values=messArray, state="readonly")
        regNoTxt.grid(row=0, column=1, sticky='w',pady=10)
        newmessTxt.grid(row=1, column=1, sticky='w',pady=10)

        def clear_text():
            regNoTxt.delete(0, END)
            newmessTxt.delete(0, END)

        def addToLL():
            if (regNoTxt.get() != '' and newmessTxt != ''):
                updaterInfo = [regNoTxt.get(),newmessTxt.get()]
                toUpdateLL.insertAtBegin(updaterInfo)
                toUpdateLL.printLL()
            clear_text()

        def updateDB():
            addToLL()
            toUpdateLL.updateStudentMessDetails()
            clear_text()

        Button(actFrm, text="Add to List", command=lambda: addToLL()).grid(row=4,
column=0, sticky='w',pady=10)
        Button(actFrm, text="Update Database", command=lambda: updateDB()).grid(row=5,
column=0, sticky='w',pady=10)

    def Password():
        clear_actFrm()
        changePassword("Hostel Office", actFrm)

    actFrm.grid(row=0, column=1, sticky='nws', padx=10, pady=10)

    Button(optFrm, text='Add Students', command=lambda: AddStudents(), width=9,
wraplength=90).grid(row=0, column=0, padx=10, pady=10)
    Button(optFrm, text='Remove Students', command=lambda: RemoveStudents(), width=9,
wraplength=90).grid(row=1, column=0, pady=10)
    Button(optFrm, text='Update Mess', command=lambda: UpdateMess(), width=9,
wraplength=90).grid(row=2, column=0, pady=10)
    Button(optFrm, text='Change Password', command=lambda: Password(), width=9,
wraplength=90).grid(row=3, column=0, pady=10)

    bottomFrm = Frame(optFrm)
    Button(bottomFrm, text='Back', command=lambda: changePage(1), width=9).grid(row=1,
column=0, sticky='s', pady=10)
    Button(bottomFrm, text='Exit', command=lambda: exitProgram(), width=9).grid(row=2,
column=0, sticky='s', pady=10)

    bottomFrm.grid(row=4, column=0, sticky='s', pady=190)
    optFrm.grid(row=0, column=0, sticky='nsw')

    F3.pack()
```

```python
geometry = '750x500+500+10'
root = Tk()
root.geometry(geometry)
root.title('Project')
root.resizable(False, False)

mycon = con.connect(host="localhost", user="root", passwd="mySQL123", 
database="messdatabase")
if mycon.is_connected():
    cur = mycon.cursor()
    changePage(1) #login page
else:
    print("Connection Unsuccessful")

root.mainloop()
```

Contributions:

1. Ashmita Das –
   a) Database and interactions with database
   b) Login Page
   c) Mess Page
   d) Changing password feature
2. Sahasra Pulumati –
   a) Data Structures
   b) Hostel Office Page
   c) Project Report