UNIVERSITY IBN TOFAIL

Algorithms II

Problem Set III

Exercise 1: Employee Management

We want to create a Python program to manage a list of employees. Each employee is modeled as a dictionary containing the following fields:

- id: integer (unique identifier of the employee)
- name: string
- age: integer
- position: string

All employees will be stored in a list of dictionaries. The data can also be saved and loaded from a text file named "employees.txt", where each line represents an employee, with fields separated by commas.

Required work:

1. Define the structure of an employee using a dictionary.
2. Write a function `input_employee()` that returns a dictionary representing an employee entered by the user.
3. Write a function `display_employee(employee)` that displays the information of a given employee.
4. Write the main program that:
   - Asks the user to enter information for n employees ($n \leq 10$).
   - Stores these employees in a list.
   - Displays the names of employees whose age is strictly greater than 30 years.
5. Write a function `save_employees(list, filename)` that saves the list of employees to a text file ("employees.txt"), with each employee on a line in the format: id, name, age, position.
6. Write a function `load_employees(filename)` that reads data from the text file and returns the list of employees (as dictionaries).
7. Write a function `search_employee(list, id)` that searches for an employee by id in the list and displays their information if they exist.

8. Write a function `update_position(list, id, new_position)` that updates the position of an employee identified by their id.

9. Write a function `delete_employee(list, id)` that removes the employee with the given id from the list, then updates the "employees.txt" file.

10. Write a function `sort_employees_by_age(list)` that sorts the list of employees by ascending age and displays the sorted list.

Correction

Algorithm

Require: Operations for employee management (add, display, save, load, search, update, delete, sort)
Ensure: Manage employee data
Define employee dictionary with fields: id, name, age, position
Define function input_employee():
Read id, name, age, position from input
return employee dictionary
Define function display_employee(*employee*):
print employee details
Define function save_employees($list, filename$):
for each employee in list **do**
employee data to file in CSV format
end for
Define function load_employees($filename$): file line by line CSV into employee dictionaries
return list of employees
Define function search_employee($list, id$):
for each employee in list **do**
if employee.id == id **then**
return employee
end if
end for
return None
Define function update_position($list, id, new\_position$):
for each employee in list **do**
if employee.id == id **then**
employee.position = new_position
end if
end for
Define function delete_employee($list, id$): employee with matching id from list
Define function sort_employees_by_age(*list*): list by employee.age in ascending order

Python Implementation

Correction

```python
def input_employee():
    return {
        'id': int(input("Enter ID: ")),
        'name': input("Enter name: "),
        'age': int(input("Enter age: ")),
        'position': input("Enter position: ")
    }

def display_employee(emp):
    print(f"ID: {emp['id']}, Name: {emp['name']}, Age: {emp['age']}, Position: {emp['position']}")

def save_employees(employees, filename):
    with open(filename, 'w') as f:
        for emp in employees:
            f.write(f"{emp['id']},{emp['name']},{emp['age']},{emp['position']}\n")

def load_employees(filename):
    employees = []
    with open(filename, 'r') as f:
        for line in f:
            parts = line.strip().split(',')
            employees.append({
                'id': int(parts[0]),
                'name': parts[1],
                'age': int(parts[2]),
                'position': parts[3]
            })
    return employees

def search_employee(employees, emp_id):
    for emp in employees:
        if emp['id'] == emp_id:
            return emp
    return None

def update_position(employees, emp_id, new_pos):
    for emp in employees:
        if emp['id'] == emp_id:
            emp['position'] = new_pos

def delete_employee(employees, emp_id):
    return [emp for emp in employees if emp['id'] != emp_id]

def sort_employees_by_age(employees):
    return sorted(employees, key=lambda x: x['age'])
```

Correction

```python
# Main program
n = int(input("Enter number of employees (n     10): "))
employees = [input_employee() for _ in range(n)]
print("\nEmployees over 30:")
for emp in employees:
    if emp['age'] > 30:
        display_employee(emp)
save_employees(employees, "employees.txt")
loaded = load_employees("employees.txt")
search_id = int(input("Enter ID to search: "))
found = search_employee(loaded, search_id)
if found:
    display_employee(found)
update_position(loaded, search_id, "Manager")
loaded = delete_employee(loaded, search_id)
sorted_employees = sort_employees_by_age(loaded)
print("\nSorted Employees:")
for emp in sorted_employees:
    display_employee(emp)
```

Listing 1: Python Code for Exercise 1

Exercise 2: Student Management

We want to develop a Python application to manage a list of students. Each student is represented by a dictionary with the following fields:

- code: integer (unique identifier of the student)
- last_name: string
- first_name: string
- gender: character ('M' for male or 'F' for female)
- average: float (average grade out of 20)

Students are stored in a list of dictionaries and can be saved or loaded from a JSON file named "students.json".

Required work:

1. Declare the Student record using a dictionary.
2. Write a function `input_student()` that allows filling in the fields of a student entered by the user, and returns a dictionary.
3. Write a function `display_student(student)` that displays the information of a given student.
4. Write a main program that:
   - Asks the user to enter information for n students (with $n \leq 100$), and stores them in a list.
   - Displays the names of students with an average greater than or equal to 10.
   - Displays the information of the student with the highest average.
5. Write a function `save_students(list, filename)` that saves the list of students to a JSON file named "students.json".
6. Write a function `load_students(filename)` that reads data from the JSON file and reconstructs the list of students as dictionaries.
7. Write a function `search_student_by_code(code, filename)` that searches for a student in the JSON file based on their code, and returns their information if found.
8. Write a function `update_average(code, new_average, filename)` that updates the average of a student searched by their code, then saves the data to the JSON file.
9. Write a function `delete_student(code, filename)` that deletes a student identified by their code, then saves the remaining students to a new JSON file named "students_updated.json".

Correction

Algorithm

Require: Operations for student management (add, display, save, load, search, update, delete)
Ensure: Manage student data
Define student dictionary with fields: code, last$_n ame$, $first_n ame$, $gender$, $average Define function$input_student():
Read code, last$_n ame$, $first_n ame$, $gender$, $average from input$**return** $student dictionary$
Define function display_student($student$):
print student details
Define function save_students($list$, $filename$): list to JSON file
Define function load_students($filename$): JSON file and parse into student dictionaries
Define function search_student_by_code($code$, $filename$): JSON file
for each student in file **do**
  if student.code == code **then**
    return student
  end if
end for
return None
Define function update_average($code, new\_avg, filename$): JSON file student.average for matching code updated data back to JSON
Define function delete_student($code$, $filename$): JSON file student with matching code remaining data to "students$_u pdated.json$"

Python Implementation

Correction

```python
import json

def input_student():
    return {
        'code': int(input("Enter code: ")),
        'last_name': input("Enter last name: "),
        'first_name': input("Enter first name: "),
        'gender': input("Enter gender (M/F): "),
        'average': float(input("Enter average (0-20): "))
    }

def display_student(student):
    print(f"Code: {student['code']}, Name: {student['first_name']} {student['last_name']}, Gender: {student['gender']}, Average: {student['average']}")

def save_students(students, filename):
    with open(filename, 'w') as f:
        json.dump(students, f)

def load_students(filename):
    with open(filename, 'r') as f:
        return json.load(f)

def search_student_by_code(code, filename):
    students = load_students(filename)
    for student in students:
        if student['code'] == code:
            return student
    return None

def update_average(code, new_avg, filename):
    students = load_students(filename)
    for student in students:
        if student['code'] == code:
            student['average'] = new_avg
    save_students(students, filename)

def delete_student(code, filename):
    students = load_students(filename)
    updated = [s for s in students if s['code'] != code]
    save_students(updated, "students_updated.json")
```

Correction

```python
# Main program
n = int(input("Enter number of students (n     100): "))
students = [input_student() for _ in range(n)]
print("\nStudents with average     10:")
for s in students:
    if s['average'] >= 10:
        display_student(s)
print("\nStudent with highest average:")
top = max(students, key=lambda x: x['average'])
display_student(top)
save_students(students, "students.json")
code = int(input("Enter code to search: "))
found = search_student_by_code(code, "students.json")
if found:
    display_student(found)
new_avg = float(input("Enter new average: "))
update_average(code, new_avg, "students.json")
delete_student(code, "students.json")
```

Listing 2: Python Code for Exercise 2

Exercise 3: Complex Number Manipulation

A complex number Z is fully defined by its real part a and its imaginary part b, and is written in the form: $Z = a + bi$

In this exercise, complex numbers will be represented using dictionaries in Python. They will be stored in a list, which will allow various mathematical operations and manipulations. This data can also be saved or loaded from a CSV file named "complex.csv".

Required work:

1. Declare a complex number using a Python dictionary.
2. Write the following functions:
   - `realPart(Z)`: returns the real part of the complex number Z.
   - `imaginaryPart(Z)`: returns the imaginary part of the complex number Z.
   - `Modulus(Z)`: returns the modulus of the complex number Z, defined by: $|Z| = \sqrt{a^2 + b^2}$.
3. Implement the following arithmetic functions:
   - `addition(Z1, Z2)`: returns the sum of Z1 and Z2.
   - `subtraction(Z1, Z2)`: returns the difference Z1-Z2.
   - `multiplication(Z1, Z2)`: returns the product of Z1 and Z2, according to the formula: $(a + bi)(c + di) = (ac - bd) + (ad + bc)i$
4. Write a function `conjugate(Z)` that returns the conjugate of a complex number Z, which is $a - bi$.
5. Write a function `inverse(Z)` that returns the inverse of a complex number Z, if it is non-zero: $Z^{-1} = \frac{a-bi}{a^2+b^2}$
6. Write a function `equality(Z1, Z2)` that tests if two complex numbers are equal (same real and imaginary parts).
7. Write a procedure `display(Z)` that displays a complex number in text form (e.g., 3 + 4i or 2 - 5i, or just 7 if the imaginary part is zero...).

Now assume we have an array TC containing N complex numbers (N $\leq$ 100).

8. Display the complex number with the largest modulus in the array TC, then check if its conjugate is also present in this array.
9. Calculate:
   - The sum Zs of all elements in TC.
   - The product Zp of all non-zero elements of the array.
10. Calculate the difference Zs - Zp and display the result only if this difference is a pure imaginary number (its real part is zero).

11. Write a function `save_csv(TC, filename)` that saves the TC array to a CSV file named "complex.csv", each line containing two values: the real part and the imaginary part of the complex number.

12. Write a function `load_csv(filename)` that reads the "complex.csv" file, and returns a list of dictionaries representing the complex numbers read.

13. After reading the data from the "complex.csv" file, display all complex numbers whose modulus is strictly greater than a given value (entered by the user).

Correction

Algorithm

Require: Operations for complex numbers (add, subtract, multiply, conjugate, inverse, equality)
Ensure: Complex number arithmetic
Define complex number dictionary: real, imaginary
Define function realPart(Z):
return Z.real
Define function imaginaryPart(Z):
return Z.imaginary
Define function Modulus(Z):
return $\sqrt{Z.real^2 + Z.imaginary^2}$
Define function addition($Z1, Z2$):
return $\{real : Z1.real + Z2.real, imaginary : Z1.imaginary + Z2.imaginary\}$
Define function subtraction($Z1, Z2$):
return $\{real : Z1.real - Z2.real, imaginary : Z1.imaginary - Z2.imaginary\}$
Define function multiplication($Z1, Z2$):
return $\{real : Z1.real * Z2.real - Z1.imaginary * Z2.imaginary, imaginary : Z1.real * Z2.imaginary + Z1.imaginary * Z2.real\}$
Define function conjugate(Z):
return $\{real : Z.real, imaginary : -Z.imaginary\}$
Define function inverse(Z):
if $Z.real == 0$ and $Z.imaginary == 0$ **then**
"Zero complex number"
end if
$denom \leftarrow Z.real^2 + Z.imaginary^2$
return $\{real : Z.real/denom, imaginary : -Z.imaginary/denom\}$
Define function equality($Z1, Z2$):
return $Z1.real == Z2.real$ and $Z1.imaginary == Z2.imaginary$
Define function display(Z):
if $Z.imaginary == 0$ **then**
print $Z.real$
else if $Z.imaginary > 0$ **then**
print $f"Z.real + Z.imaginaryi"$
else
print $f"Z.real - -Z.imaginaryi"$
end if

Python Implementation

Correction

```python
import math
import csv

def real_part(z):
    return z['real']

def imaginary_part(z):
    return z['imaginary']

def modulus(z):
    return math.sqrt(real_part(z)**2 + imaginary_part(z)**2)

def addition(z1, z2):
    return {'real': real_part(z1) + real_part(z2), 'imaginary':
    imaginary_part(z1) + imaginary_part(z2)}

def subtraction(z1, z2):
    return {'real': real_part(z1) - real_part(z2), 'imaginary':
    imaginary_part(z1) - imaginary_part(z2)}

def multiplication(z1, z2):
    real = real_part(z1)*real_part(z2) - imaginary_part(z1)*
    imaginary_part(z2)
    imag = real_part(z1)*imaginary_part(z2) + imaginary_part(z1)*
    real_part(z2)
    return {'real': real, 'imaginary': imag}

def conjugate(z):
    return {'real': real_part(z), 'imaginary': -imaginary_part(z)}

def inverse(z):
    if real_part(z) == 0 and imaginary_part(z) == 0:
        raise ValueError("Cannot invert zero complex number")
    denom = modulus(z)**2
    return {'real': real_part(z)/denom, 'imaginary': -
    imaginary_part(z)/denom}

def equality(z1, z2):
    return real_part(z1) == real_part(z2) and imaginary_part(z1) ==
    imaginary_part(z2)

def display_complex(z):
    r = real_part(z)
    i = imaginary_part(z)
    if i == 0:
        print(r)
    elif i > 0:
        print(f"{r} + {i}i")
    else:
        print(f"{r} - {-i}i")
```

Correction

```python
# Additional operations for array TC
def save_csv(tc, filename):
    with open(filename, 'w', newline='') as f:
        writer = csv.writer(f)
        for z in tc:
            writer.writerow([real_part(z), imaginary_part(z)])

def load_csv(filename):
    tc = []
    with open(filename, 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            tc.append({'real': float(row[0]), 'imaginary': float(row[1])})
    return tc

# Example usage
z1 = {'real': 3, 'imaginary': 4}
z2 = {'real': 1, 'imaginary': -2}
z_sum = addition(z1, z2)
display_complex(z_sum)
tc = [z1, z2]
save_csv(tc, "complex.csv")
loaded = load_csv("complex.csv")
```

Listing 3: Python Code for Exercise 3