ระบบจัดการข้อมูลพนักงาน

นาย อลีฟ	แวหะยี	6706022510034
นาย วิสิฐศักดิ์	เพชรหนองชุม	6706022510123
นาย นนทวัฒน์	ประสพรัตน์	6706022510212
นางสาว วริศรา	สุขสวัสดิ์	6706022510221

โครงงานนี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมสารสนเทศและเครือข่าย ภาควิชาเทคโนโลยีสารสนเทศ
คณะเทคโนโลยีและการจัดการอุตสาหกรรม มหาวิทยาลัยเทคโนโลยีพระจอม
เกล้าพระนครเหนือ
ปีการศึกษา 2567

คำนำ

การจัดทำโครงงาน “ระบบจัดการข้อมูลพนักงาน” นี้เป็นส่วนหนึ่งของวิชา COMPUTER PROGRAMMIMG ของหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมสารสนเทศและเครือข่าย ภาควิชาเทคโนโลยีสารสนเทศคณะเทคโนโลยีและการจัดการอุตสาหกรรม มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ เพื่อให้นักศึกษาได้นำความรู้ที่เรียนมาทั้งหมดมาประยุกต์ใช้ในการพัฒนาโปรแกรมที่สามารถทำงานได้จริง โดยเน้นการออกแบบและเขียนโปรแกรมในภาษา Python ซึ่งเป็นภาษาที่เรียนมาในวิชา COMPUTER PROGRAMMING โดยโครงงานนี้จะช่วย การคิดวิเคราะห์และแก้ปัญหาทางเทคนิค เพื่อเตรียมความพร้อมในการประกอบอาชีพด้านวิศวกรรมสารสนเทศและเครือข่ายในอนาคต หากมีข้อผิดพลาดประการใด คณะผู้จัดทำต้องขออภัยไว้ ณ ที่นี้ด้วย

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญภาพ

ภาพที่ หน้า

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

บทที่ 1
บทนำ

1.1 วัตถุประสงค์ของโครงการ

1) เพื่อจัดเก็บและบริหารข้อมูลพนักงานได้อย่างเป็นระบบ

2) เพื่อค้นหาข้อมูลได้อย่างรวดเร็วและแม่นยำ

3) เพื่อสนับสนุนการตัดสินใจในการบริหารทรัพยากรบุคคล

4) เพื่อสร้างรายงานข้อมูลพนักงานในรูปแบบที่เข้าใจง่าย

5) เพื่อลดข้อผิดพลาดในการจัดการข้อมูลด้วยการตรวจสอบความถูกต้อง

6) เพื่อช่วยประหยัดเวลาและลดความยุ่งยากในการดำเนินงาน

1.2 ขอบเขตของโครงการ

1) ระบบจัดการข้อมูลพนักงานจะมีฟังก์ชันพื้นฐาน 8 ฟังก์ชัน ได้แก่ 1. Main Menu 2. Show data 3. Insert data 4. Edit data 5. Show specific 6. Delete Data 7. Exit Data

2) ระบบจัดการข้อมูลพนักงาน ประกอบด้วย 5 ฟิลด์ ได้แก่ 1. ID 2. Name 3. Score 4. Department 5. Salary

3) ระบบจัดการข้อมูลพนักงานมีการจัดเก็บข้อมูลพนักงานและคะแนนพนักงานไว้ใน Text file ชื่อ report.txt ซึ่งมี รหัสพนักงาน ชื่อพนักงาน แผนก คะแนนเฉลี่ย และเงินเดือน

1.3 ประโยชน์ที่ได้รับ

1) พัฒนาระบบจัดการพนักงานได้อย่างมีประสิทธิภาพ

2) พัฒนาทักษะการเขียนโปรแกรม

3) พัฒนาทักษะการทำงานเป็นทีม

4) พัฒนาทักษะการแก้ปัญหาและการจัดการข้อมูล

1.4 เครื่องมือที่ใช้งาน

1) ภาษา Python

2) Microsoft office

3) Visual Code Studio

4) Git Hub

5) Git copilot

บทที่ 2
การทำงานของระบบจัดการข้อมูลพนักงาน

2.1 ฟิลด์ในการจัดการข้อมูลพนักงาน

การจัดการข้อมูลพนักงานในระบบประกอบด้วย 5 ฟิลด์หลักๆ ซึ่งแต่ละฟิลด์มีรายละเอียดและความสำคัญดังนี้

2.1.1 ID (รหัสพนักงาน)

ID เป็นรหัสประจำตัวพนักงานที่ใช้ในการระบุรหัสพนักงานแต่ละคนอย่างเฉพาะเจาะจง ซึ่งจะเป็นตัวเลขที่ไม่ซ้ำกันในระบบ ซึ่งรูปแบบของฟิลด์ เป็นประเภทข้อมูล String เช่น 0001, 0002, 0003 เป็นต้น ความสำคัญของฟิลด์นี้คือ เปรียบเสมือน Primary key ที่แต่ละหมายเลขไม่สามารถซ้ำกันได้

2.1.2 Name (ชื่อพนักงาน)

Name คือ ชื่อของพนักงาน ซึ่งรูปแบบของฟิลด์ เป็นประเภทข้อมูล String (ข้อความ) เช่น “Peter”, “Bob”, “Alice” เป็นต้น ความสำคัญของฟิลด์นี้คือใช้ระบุชื่อของพนักงาน สามารถใช้ค้นหาและสามารถซ้ำกันได้

2.1.3 Department (แผนก)

Department คือ ชื่อแผนกของพนักงานที่ทำงานใช้ในการระบุว่าพนักงานคนนี้อยู่แผนกอะไรเป็นกลุ่มๆ เช่น “QC”, “IT”, “Finance” เป็นต้น ซึ่งรูปแบบของฟิลด์ เป็นประเภทข้อมูล String ความสำคัญของฟิลด์นี้คือ หมวดหมู่ช่วยในการจัดกรองข้อมูลเป็นแผนกๆสามารถหาข้อมูลได้ง่ายขึ้น

2.1.4 Score (คะแนน)

Score คือ คะแนนที่พนักงานสอบวัดผลทุกๆ 4 ไตรมาส 1 ไตรมาสจะจัดเก็บ 1 ครั้ง ไว้ประเมินพนักงาน ถ้าพนักงานเข้ามาใหม่จะไม่มีการจัดเก็บคะแนนจะปล่อยให้คะแนนเป็น 0 แต่สามารถเพิ่มหรือแก้ไขได้ จัดเก็บข้อมูลเป็น String แล้วค่อย Convert เป็น List แล้วนำข้อมูลใน List Convert เป็น Float และหาค่าเฉลี่ย ของ 4 ไตรมาส เช่น [“90.0”, “80.0”, “70.0”, “99.0”] เป็นต้น ความสำคัญของฟิลด์นี้คือ ช่วยจัดเก็ยคะแนนของพนักงานเพื่อประเมินหรือขึ้นเงินเดือนให้แก่พนักงาน เพื่อเพิ่มประสิทธิภาพในองค์กร

2.1.5 Salary (เงินเดือน)

Salary คือ เงินเดือนของพนักงาน จัดเก็บข้อมูลเป็น String แล้วค่อย Convert เป็น Float มีไว้จัดกลุ่มว่าพนักงานที่มีเงินเดือนมากกว่าหรือเท่ากับ 15000 หรือพนักงานที่มีเงินเดือนน้อยกว่า 30000 เป็นต้น

2.2 ฟังก์ชันการทำงานของระบบจัดการพนักงาน

2.2.1 ฟังก์ชัน main

ฟังก์ชัน main เป็นฟังก์ชันหลักของโปรแกรมระบบจัดการพนักงาน ทำหน้าที่แสดงเมนูการจัดการข้อมูลพนักงาน และรับคำสั่งจากผู้ใช้เพื่อนำไปเรียกใช้ฟังก์ชันย่อยอื่นๆ ตามที่เลือกจากเมนู ฟังก์ชันนี้ทำงานวนซ้ำในลูปจนกว่าผู้ใช้จะเลือกออกจากโปรแกรม โดยรายละเอียดฟงัก์ชันมีดังนี้

1) โครงสร้างการทำงานของฟังก์ชัน main

แสดงเมนูหลักเมื่อโปรแกรมเริ่มต้น ฟังก์ชัน main() จะทำการแสดงเมนูหลักที่มีตัวเลือกต่างๆ ให้กับผู้ใช้ ผู้ใช้สามารถเลือกตัวเลือกตามหมายเลขที่ต้องการได้ ดังนี้

1.1) กด 1 เพื่อแสดงข้อมูลพนักงานทั้งหมด

1.2) กด 2 เพื่อค้นหาข้อมูลแบบเฉพาะเจาะจง

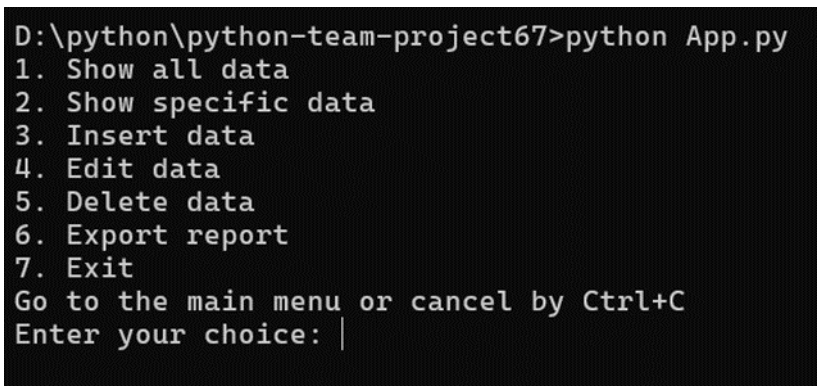
1.3) กด 3 เพื่อแทรกข้อมูล

1.4) กด 4 เพื่อแก้ไขข้อมูล

1.5) กด 5 เพื่อลบข้อมูล

1.6) กด 6 เพื่อแสดงผลข้อมูลเป็นไฟล์ .txt

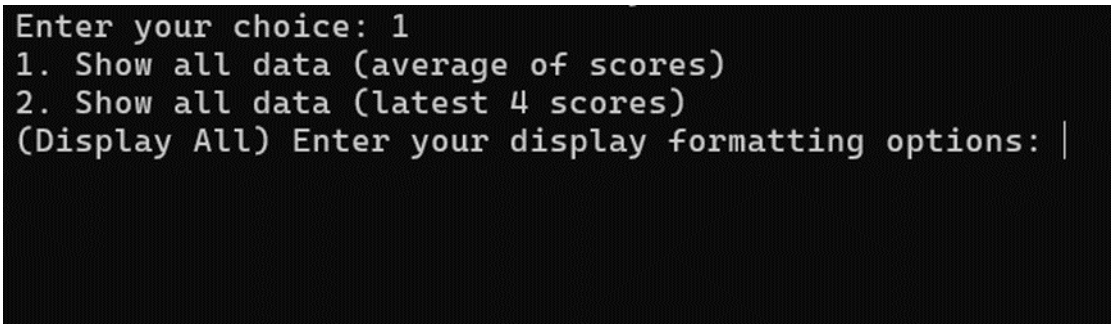
1.7) กด 7 เพื่อออกจากโปรแกรม



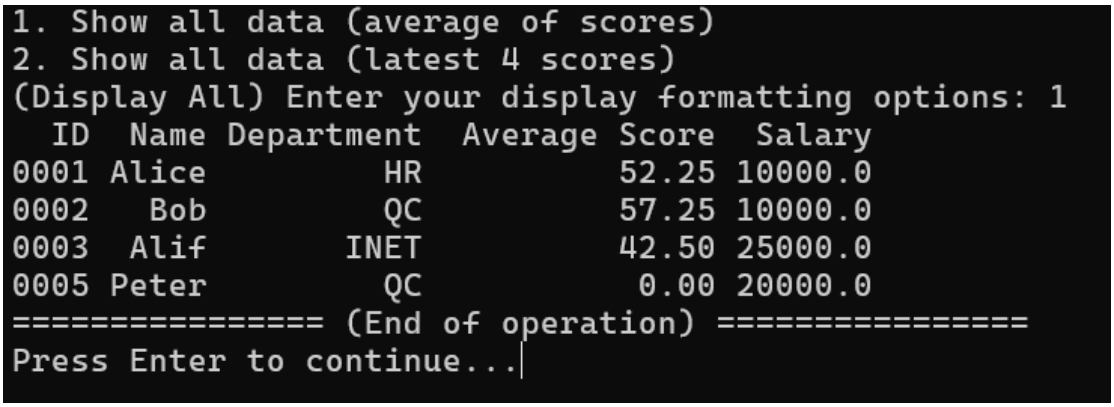
ภาพที่ 2-1 หน้าแสดงผลฟังก์ชัน Main

2.2.2 ฟังก์ชัน Show all data

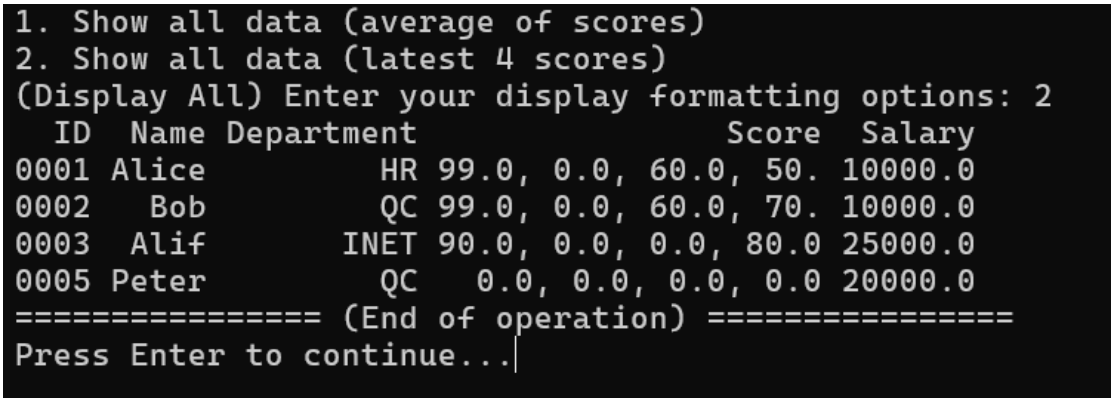
เป็นฟังก์ชันการแสดงผลข้อมูลทั้งหมด โดยสามารถแสดงผลข้อมูลเป็นค่าเฉลี่ยคะแนนของพนักงานหรือจะแสดงผลคะแนนออกมาทั้ง 4 ไตรมาสก็ได้



ภาพที่ 2-2 การแสดงผลข้อมูลแบบตัวเลือก



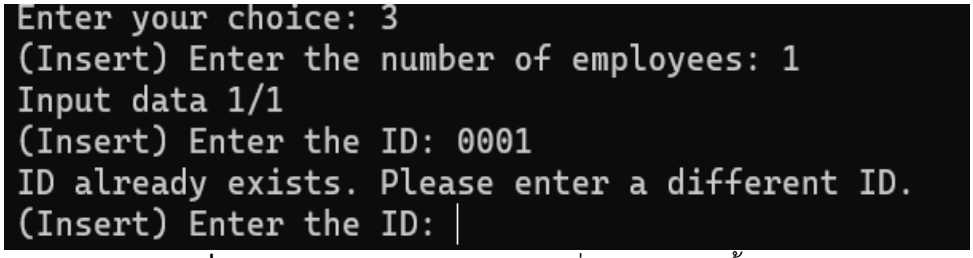
ภาพที่ 2-3 การแสดงผลข้อมูลทั้งหมดแบบค่าเฉลี่ย



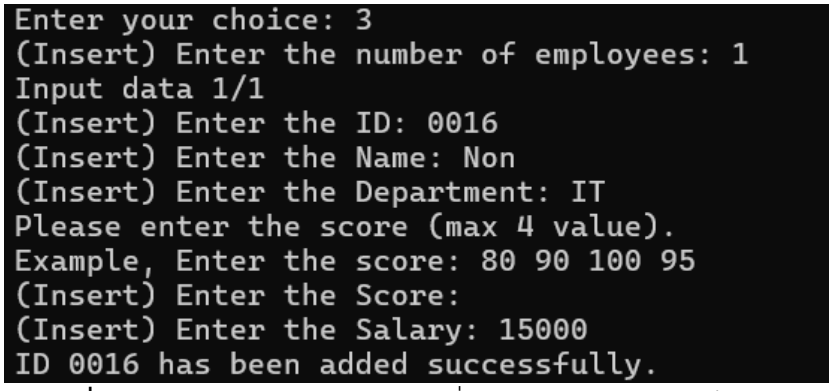
ภาพที่ 2-4 การแสดงผลข้อมูลทั้งหมดแบบแสดงทั้ง 4 ไตรมาส

2.2.3 ฟังก์ชัน Insert Data

เป็นฟังก์ชันในการแทรกข้อมูลสามารถเลือกแทรกจำนวนข้อมูลพนักงานทั้งหมดกี่คน โดยถ้า ID พนักงานซ้ำจะไม่สามารถแทรกข้อมูลพนักงานได้ โดยข้อมูลที่จะให้แทรก ประกอบด้วย 1. รหัสพนักงาน 2. ชื่อพนักงาน 3. แผนกของพนักงาน 4. คะแนนของพนักงาน (กรณีพนักงานเข้ามาใหม่ไม่ต้องใส่ก็ได้) 5. เงินเดือนของพนักงาน

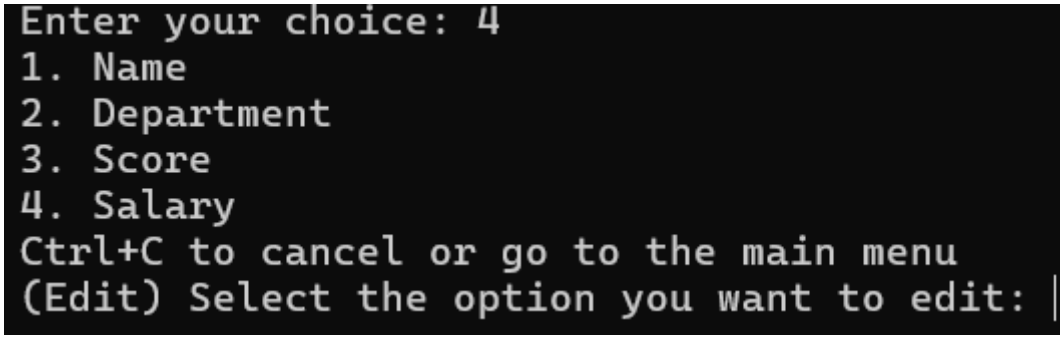


ภาพที่ 2-5 การ Insert data ในกรณีที่ ID พนักงานซ้ำ



ภาพที่ 2-6 การ Insert data ในกรณีที่สามารถ Insert ได้สำเร็จ

2.2.4 ฟังก์ชัน Edit data



ภาพที่ 2-7 หน้า Menu การแก้ไขข้อมูลของพนักงาน

เป็นฟังก์ชันที่ใช้สำหรับแก้ไขข้อมูลของพนักงาน โดยจะอ้างอิงจาก ID ซึ่งไม่สามารถแก้ไขได้ โดยสิ่งที่สามารถแก้ไขได้ มีดังนี้

1) ชื่อของพนักงาน (Name)

การแก้ไขชื่อพนักงาน สามารถแก้ไขได้โดย ใส่ ID พนักงานที่ต้องการจะแก้ไข และแก้ไขชื่อหรือเปลี่ยนชื่อใหม่ ถ้าต้องการแก้ไขข้อมูลพนักงานหลายคนหลังจากใส่เลข ID ของพนักงานคนแรกเสร็จให้เคาะ Space Bar 1 ครั้งและใส่ ID พนักงานคนถัดไป

```
(Edit) Select the option you want to edit: 1
Please enter a single ID or multiple IDs to edit.
Example. [A single ID] Enter ID: 0001
Example. [Multiple IDs] Enter ID: 0001 0002 0003
(Edit) Enter ID: 0001
Please enter new name to edit. (Must be equal to the number of IDs)
Example. Enter the new name for ID('001', '002'): John Peter
(Edit) Enter the new name for ID('0001',): Nick
================ (End of operation) ================
```

ภาพที่ 2-8 ตัวอย่างการแก้ไขชื่อของพนักงาน

2) แผนกของพนักงาน (Department)

การแก้ไขแผนกของพนักงาน สามารถแก้ไขได้โดย ใส่ ID พนักงานที่ต้องการจะแก้ไข และแก้ไขชื่อแผนกหรือเปลี่ยนชื่อแผนกใหม่ ถ้าต้องการแก้ไขข้อมูลพนักงานหลายคนหลังจากใส่เลข ID ของพนักงานคนแรกเสร็จให้เคาะ Space Bar 1 ครั้งและใส่ ID พนักงานคนถัดไป

```
(Edit) Select the option you want to edit: 1
Please enter a single ID or multiple IDs to edit.
Example. [A single ID] Enter ID: 0001
Example. [Multiple IDs] Enter ID: 0001 0002 0003
(Edit) Enter ID: 0001
Please enter new name to edit. (Must be equal to the number of IDs)
Example. Enter the new name for ID('001', '002'): John Peter
(Edit) Enter the new name for ID('0001',): Nick
================ (End of operation) ================
```

ภาพที่ 2-9 ตัวอย่างการแก้ไขแผนกของพนักงาน

3) คะแนนของพนักงาน (Score)

การแก้ไขข้อมูลคะแนนของพนักงาน สามารถแก้ไขได้โดย ใส่ ID พนักงานที่ต้องการจะแก้ไข และแก้ไขคะแนน การแก้ไขคะแนนมี 2 แบบคือ แบบที่ 1 คือ การเพิ่มคะแนนเข้าไปใหม่โดยจะเพิ่มไปที่ Index หลังสุดและคะแนนข้างหน้าก็จะหายไป แบบที่ 2 คือ แก้ไขคะแนนที่มีอยู่โดยระบุ Index ที่ต้องการจะแก้ไขและใส่คะแนนเข้าไปใหม่ ถ้าต้องการแก้ไขข้อมูลพนักงานหลายคนหลังจากใส่เลข ID ของพนักงานคนแรกเสร็จให้เคาะ Space Bar 1 ครั้งและใส่ ID พนักงานคนถัดไป

```
(Edit) Select the option you want to edit: 3
Please enter a single ID or multiple IDs to edit.
Example. [A single ID] Enter ID: 0001
Example. [Multiple IDs] Enter ID: 0001 0002 0003
(Edit) Enter ID: 0001 0002
1. Add new score
2. Edit existing score
(Edit) Enter your choice: 1
  ID  Name                    Score
0001  Nick [92.0, 93.0, 90.0, 91.0]
0002 Alice [80.0, 81.0, 75.0, 79.0]
Please enter new score to edit. (Max 4 value)
Example. Enter the new score: 80 90 100 95
(Edit>Add new>ID0001) Enter the new score: 90 95
Please enter new score to edit. (Max 4 value)
Example. Enter the new score: 80 90 100 95
(Edit>Add new>ID0002) Enter the new score: 90 99
```

ภาพที่ 2-10 การแก้ไขคะแนนพนักงานโดยการเพิ่มเข้าไปใหม่

```
  ID    Name  Department                    Score
0001    Nick          IT 90.0, 91.0, 90.0, 95
0002   Alice          QC 75.0, 79.0, 90.0, 99
```

ภาพที่ 2-11 ตัวอย่างการแก้ไขคะแนนพนักงานโดยการเพิ่ม

```
1. Add new score
2. Edit existing score
(Edit) Enter your choice: 2
  ID Name                   Score
0001 Nick [90.0, 91.0, 90.0, 95.0]
Please enter the index of score to edit. (0-3)
Example. Enter the index: 0 1 2 3
(Edit>Edit existing>ID0001) Enter the index: 0
Please enter new score to edit. (According to position of idex)
Example. Enter the new score for index ('0', '2'): 80 90
(Edit>Edit existing>ID0001) Enter the new score for index ('0',): 99
```

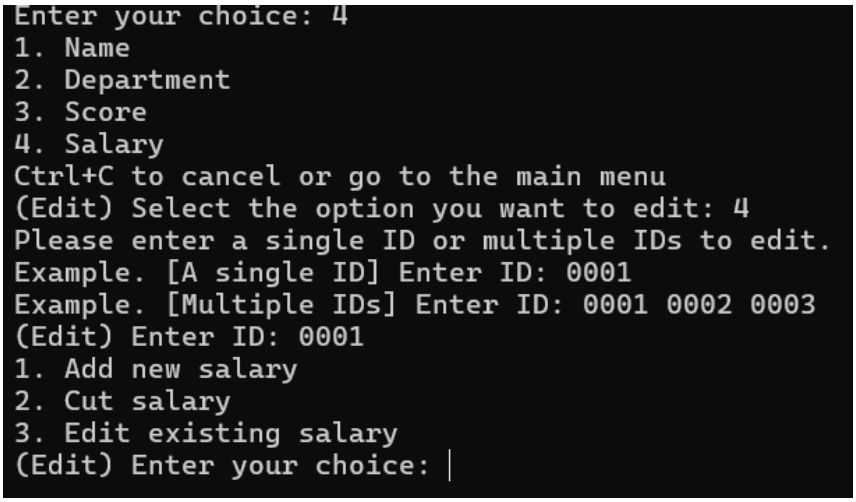
ภาพที่ 2-12 การแก้ไขคะแนนพนักงานโดยการแทนที่คะแนนที่มีอยู่

```
  ID    Name  Department                    Score
0001    Nick          IT 99.0, 91.0, 90.0, 95
0002   Alice          QC 75.0, 79.0, 90.0, 99
```

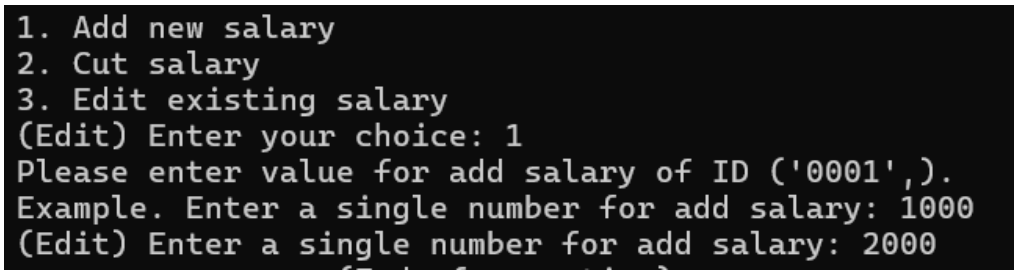
ภาพที่ 2-13 ตัวอย่างการแก้ไขคะแนนพนักงานโดยการแทนที่

4) เงินเดือนของพนักงาน (Salary)

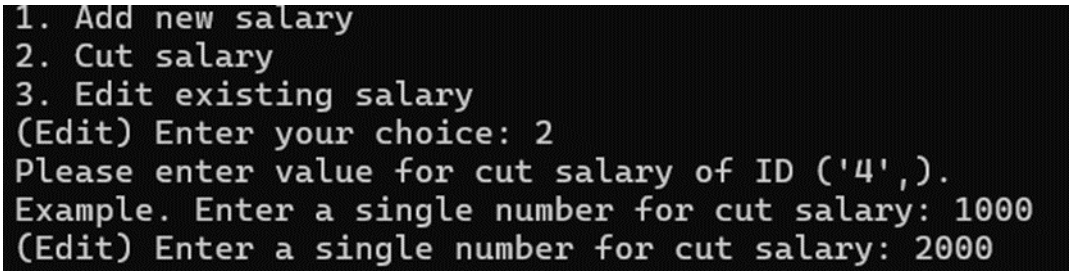
การแก้ไขข้อมูลเงินเดือนของพนักงาน สามารถแก้ไขได้โดย ใส่ ID พนักงานที่ต้องการจะแก้ไข และแก้ไขข้อมูลเงินเดือน การแก้ไขเงินเดือนประกอบด้วย 3 ฟังก์ชัน คือ 1. การเพิ่มเงินเดือน 2.การลดเงินเดือน 3.การแทนที่เงินเดือนที่มีอยู่ ถ้าต้องการแก้ไขข้อมูลพนักงานหลายคนหลังจากใส่เลข ID ของพนักงานคนแรกเสร็จให้เคาะ Space Bar 1 ครั้งและใส่ ID พนักงานคนถัดไป



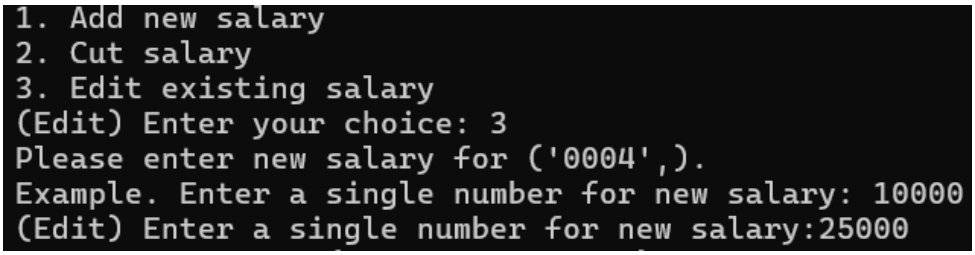
ภาพที่ 2-14 หน้าเมนูการแก้ไขเงินเดือน



ภาพที่ 2-15 ตัวอย่างการเพิ่มเงินเดือน (ถ้าใส่ไป 2000 ก็จะบวกเพิ่มเงินเดือนไป 2000)



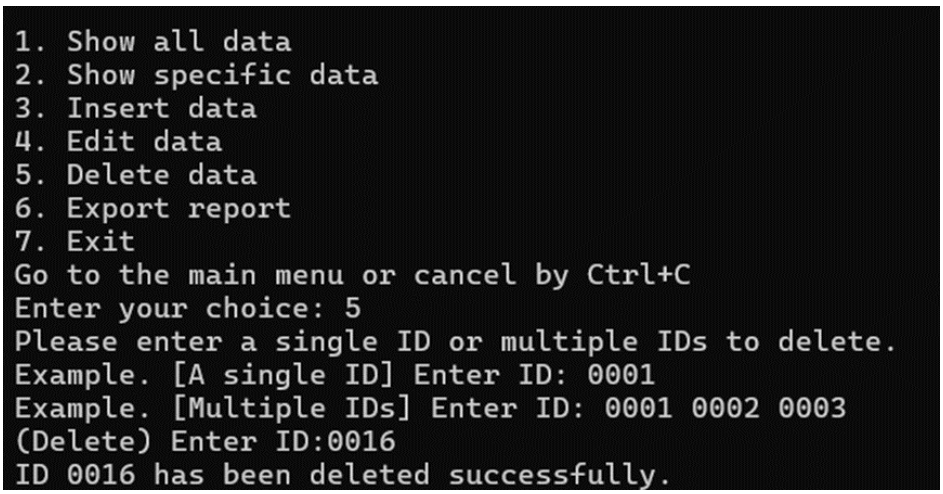
ภาพที่ 2-16 ตัวอย่างการลดเงินเดือน (ถ้าใส่ไป 2000 ก็จะลดเงินเดือนไป 2000)



ภาพที่ 2-17 ตัวอย่างการแทนที่เงินเดือน (ถ้าใส่ไป 25000 ก็จะแทนที่เงินเดือนเดิม)

2.2.5 ฟังก์ชัน Delete

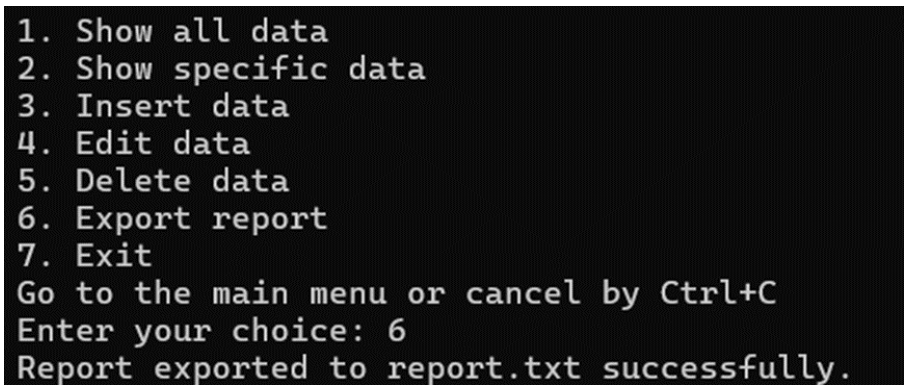
เป็นฟังก์ชันในการลบข้อมูลจะทำการลบข้อมูลของพนักงานโดยระบุ ID ของพนักงาน และทำการลบข้อมูลภายในฟิลด์พนักงานคนนั้นทั้งหมด



ภาพที่ 2-18 ตัวอย่างการลบข้อมูลพนักงาน

2.2.6 ฟังก์ชัน Export

เป็นฟังก์ชันในการ Export ข้อมูลออกมาเป็นไฟล์ชื่อว่า Report.txt



ภาพที่ 2-19 การ Export ข้อมูล

```
Summary Report:
Department: IT
  Nick: Average Score = 93.75
Top Performer: Nick Average score = 93.75
Department: QC
  Alice: Average Score = 85.75
Top Performer: Alice Average score = 85.75
Department: Marketing
  John: Average Score = 82.00
  Jessica: Average Score = 82.00
  Andrew: Average Score = 95.25
Top Performer: Andrew Average score = 95.25
Department: Finance
  Emily: Average Score = 95.25
  Daniel: Average Score = 78.75
  Olivia: Average Score = 91.50
Top Performer: Emily Average score = 95.25
Department: HR
  David: Average Score = 95.25
  Sophia: Average Score = 95.25
  James: Average Score = 82.00
Top Performer: David Average score = 95.25
Department: Engineering
  Matthew: Average Score = 78.75
  Sarah: Average Score = 95.25
Top Performer: Sarah Average score = 95.25
Department: Sales
  Emma: Average Score = 91.50
  Michael: Average Score = 78.75
Top Performer: Emma Average score = 91.50

Best Department: IT Average score = 93.75
```

ภาพที่ 2-20 ตัวอย่างข้อมูลที่ทำการ Export

2.2.7 ฟังก์ชัน Exit

เป็นฟังก์ชันสำหรับการออกจากโปรแกรม โดยเมื่อเลือกฟังก์ชันแล้วตัวโปรแกรมจะทำการถามว่าต้องการออกจากโปรแกรมหรือไม่ถ้าต้องการออกให้เลือก Y เพื่อทำการออกจากโปรแกรม

```
1. Show all data
2. Show specific data
3. Insert data
4. Edit data
5. Delete data
6. Export report
7. Exit
Go to the main menu or cancel by Ctrl+C
Enter your choice: 7
Do you want to exit? (y/[n]): y

D:\python-team-project67>
```

ภาพที่ 2-21 ตัวอย่างการออกจากโปรแกรม

บทที่ 3
อธิบายการทำงานของ code ส่วนการประมวลผล

3.1 การเตรียมความพร้อมก่อนการเขียนโปรแกรม

```
1   import struct
2   import numpy as np
3   import pandas as pd
4
5   bin_path = "data.bin"
```

ภาพที่ 3-1 การเตรียมความพร้อมก่อนการเขียนโปรแกรม

3.1.1 การนำเข้าโมดูลและการกำหนดค่าเริ่มต้น

1) โปรแกรมได้ทำการนำเข้าโมดูล struct, numpy และ pandas เพื่อใช้ในการจัดการข้อมูลและการประมวลผลต่างๆ ในโปรแกรม

2) โมดูล struct: ใช้สำหรับการอ่านและเขียนข้อมูลในรูปแบบไบนารี (Binary Format)

3) โมดูล numpy: ใช้สำหรับการจัดการข้อมูลในลักษณะอาเรย์ (Array) และการประมวลผลข้อมูลเชิงตัวเลข

4) โมดูล pandas: ใช้สำหรับการจัดการข้อมูลในรูปแบบ DataFrame ซึ่งช่วยให้สามารถแสดงผลและจัดการข้อมูลได้สะดวกขึ้น

3.1.2 การกำหนดเส้นทางของไฟล์ไบนารี

bin_path = "data.bin" เป็นการกำหนดชื่อไฟล์และเส้นทางสำหรับการจัดเก็บข้อมูลในรูปแบบไบนารี ซึ่งข้อมูลพนักงานทั้งหมดจะถูกจัดเก็บและเข้าถึงผ่านไฟล์ data.bin นี้

3.2 การอ่านค่าจากไฟล์ไบนารี

ใช้ฟังก์ชันที่ชื่อว่า readDataFromBinFile ทำหน้าที่อ่านข้อมูลจากไฟล์ไบนารีและแปลงข้อมูลให้อยู่ในรูปแบบของลิสต์ (list) ซึ่งฟังก์ชันจะคืนค่าลิสต์ของเรคคอร์ดทั้งหมดที่ถูกอ่านจากไฟล์

```python
def readDataFromBinFile() -> list:
    list_records = []
    try:
        with open(bin_path, "rb") as file:
            record_size = struct.calcsize("20s20s20s20sf")
            while True:
                data = file.read(record_size)
                if not data:
                    break
                records = struct.unpack("20s20s20s20sf", data)
                record_3 = [float(score) for score in records[3].decode().strip('\x00').split()]
                if len(record_3) < 4:
                    record_3 += [0.0] * (4 - len(record_3))
                records = [records[0].decode().strip('\x00'), records[1].decode().strip(
                    '\x00'), records[2].decode().strip('\x00'), record_3, records[4]]
                list_records.append(records)
    except FileNotFoundError:
        with open(bin_path, "wb") as file:
            file.write(b'')
    return list_records
```

ภาพที่ 3-2 การอ่านค่าจาก binary file

3.2.1 การประกาศตัวแปรไว้เก็บข้อมูลไปประมวลผล

list_records: เป็นลิสต์ที่จะเก็บเรคคอร์ดที่ถูกอ่านมาจากไฟล์

3.2.2 การเปิดไฟล์

ฟังก์ชันนี้ใช้ with open(bin_path, "rb") เพื่อเปิดไฟล์ไบนารีในโหมดการอ่าน (rb ย่อมาจาก read binary) ซึ่งจะป้องกันไม่ให้เกิดการลืมปิดไฟล์

3.2.3 การอ่านข้อมูลก่อนนำไปประมวลผล

1) record_size = struct.calcsize("20s20s20s20sf") คำนวณขนาดของเรคคอร์ดแต่ละเรคคอร์ดในไฟล์ โดยมีรูปแบบข้อมูลเป็น string ขนาด 20 bytes ทั้งหมด 4 ตัว และตัวเลขแบบ float 1 ตัว (20s20s20s20sf)

2) วนลูปอ่านข้อมูลด้วย file.read(record_size) เพื่ออ่านข้อมูลทีละเรคคอร์ดตามขนาดที่คำนวณได้

3) หากอ่านได้ข้อมูลไม่ครบ (ไม่มีข้อมูลเหลืออยู่ในไฟล์) การอ่านจะหยุดทันทีด้วย if not data: break

```python
            record_size = struct.calcsize("20s20s20s20sf")
            while True:
                data = file.read(record_size)
                if not data:
                    break
```

ภาพที่ 3-3 การอ่านข้อมูล

3.2.4 การประมวลผลข้อมูลที่อ่านได้

1) struct.unpack("20s20s20s20sf", data) ทำการแปลงข้อมูลที่อ่านจากไฟล์ให้เป็นรูปแบบ tuple โดยแต่ละสมาชิกของ tuple จะสอดคล้องกับรูปแบบข้อมูลที่กำหนดไว้ในฟอร์แมต

2) record_3 ทำการแปลงข้อมูลใน field ที่ 3 (ตัวที่สี่ใน tuple) จาก string ไปเป็นลิสต์ของตัวเลข float (ในกรณีที่ข้อมูลไม่ครบ 4 ตัว จะเติมตัวเลข 0.0 ให้ครบ 4 ตัว)

3) นำข้อมูลที่แปลงได้มาจัดรูปแบบใหม่เป็นลิสต์โดยทำการตัดสัญลักษณ์ '\x00' ที่ติดมากับ string ในข้อมูลแต่ละส่วน และเก็บไว้ในลิสต์ list_records

```
16          records = struct.unpack("20s20s20s20sf", data)
17          record_3 = [float(score) for score in records[3].decode().strip('\x00').split()]
18          if len(record_3) < 4:
19              record_3 += [0.0] * (4 - len(record_3))
20          records = [records[0].decode().strip('\x00'), records[1].decode().strip(
21              '\x00'), records[2].decode().strip('\x00'), record_3, records[4]]
22          list_records.append(records)
```

ภาพที่ 3-4 การประมวลผลข้อมูลที่อ่านได้

3.2.5 การจัดการข้อผิดพลาด

หากไฟล์ที่ต้องการอ่านไม่พบ (FileNotFoundError) ฟังก์ชันจะสร้างไฟล์ใหม่ขึ้นมาและทำการเขียนไฟล์เปล่าๆ ลงไป ซึ่งจำเป็นต้องใช้คำสั่ง try ถึงจะสามารถใช้คำสั่ง except ได้

```
23      except FileNotFoundError:
24          with open(bin_path, "wb") as file:
25              file.write(b'')
```

ภาพที่ 3-5 การจัดการข้อผิดพลาด

3.2.6 ผลลัพธ์ที่ได้จากการเรียกใช้ฟังก์ชัน

ฟังก์ชันจะคืนค่าลิสต์ list_records ที่เก็บเรคคอร์ดทั้งหมดในรูปแบบที่แปลงเสร็จแล้ว

3.2.7 ตัวอย่างการเรียกใช้

การเรียกใช้ฟังก์ชันนี้คือ readDataFromBinFile() ซึ่งจะคืนค่าลิสต์ของข้อมูลที่อ่านจากไฟล์

3.3 การเขียนลงในไฟล์ไบนารี

ฟังก์ชันนี้ชื่อว่า writeDataToBinFile มีหน้าที่เขียนข้อมูลที่ได้รับมาในรูปแบบของลิสต์ (list) ลงในไฟล์ไบนารี โดยทำการบันทึกข้อมูลแต่ละเรคคอร์ดที่มีรูปแบบคงที่

```python
def writeDataToBinFile(list_records: list):
    if len(list_records) == 0:
        with open(bin_path, "wb") as file:
            file.write(b'')
    for index, record in enumerate(list_records):
        if index == 0:
            with open(bin_path, "wb") as file:
                data = struct.pack("20s20s20s20sf", record[0].encode(), record[1].encode(
                ), record[2].encode(), ' '.join(map(str, record[3])).encode(), record[4])
                file.write(data)
        else:
            with open(bin_path, "ab") as file:
                data = struct.pack("20s20s20s20sf", record[0].encode(), record[1].encode(
                ), record[2].encode(), ' '.join(map(str, record[3])).encode(), record[4])
                file.write(data)
```

ภาพที่ 3-6 การเขียนลงในไฟล์ไบนารี

3.3.1 การตรวจสอบความยาวของลิสต์ข้อมูล

ฟังก์ชันจะรับข้อมูล list_records ที่เป็นลิสต์ของเรคคอร์ด ถ้าลิสต์นี้ไม่มีข้อมูล (ความยาวของลิสต์เป็น 0) ฟังก์ชันจะเปิดไฟล์ในโหมดเขียนไบนารี (wb) แล้วทำการเขียนข้อมูลเปล่า (file.write(b'')) ลงไปเพื่อเคลียร์ข้อมูลเดิมในไฟล์ จากนั้นจะจบการทำงาน

```python
    if len(list_records) == 0:
        with open(bin_path, "wb") as file:
            file.write(b'')
```

ภาพที่ 3-7 การตรวจสอบความยาวของลิสต์ข้อมูล

3.3.2 การวนลูปเขียนข้อมูล

ฟังก์ชันจะวนลูปผ่านแต่ละเรคคอร์ดใน list_records โดยใช้ฟังก์ชัน enumerate() เพื่อให้สามารถเข้าถึงทั้งดัชนี (index) และเรคคอร์ด (record) ได้ ดังนี้:

1) ถ้า index == 0 (เรคคอร์ดแรก):

1.1) ฟังก์ชันจะเปิดไฟล์ในโหมดเขียนไบนารี (wb) เพื่อเขียนทับข้อมูลเดิมทั้งหมด

1.2) จากนั้นจะทำการบรรจุข้อมูล (pack) เรคคอร์ดในรูปแบบไบนารีด้วย struct.pack("20s20s20s20sf", ...) ซึ่งแต่ละสมาชิกของเรคคอร์ดจะถูกแปลงเป็นข้อมูลชนิดไบนารีที่กำหนดไว้ (string ขนาด 20 bytes ทั้งหมด 4 ตัว และ float 1 ตัว)

1.3) สุดท้ายจะทำการเขียนข้อมูลที่ได้ลงไปในไฟล์ด้วย file.write(data)

```python
    if index == 0:
        with open(bin_path, "wb") as file:
            data = struct.pack("20s20s20s20sf", record[0].encode(), record[1].encode(
            ), record[2].encode(), ' '.join(map(str, record[3])).encode(), record[4])
            file.write(data)
```

ภาพที่ 3-8 หากเข้าเงื่อนไขของเรคคอร์ดแรกในการเขียนข้อมูลลงไฟล์

2) ถ้า index > 0 (เรคคอร์ดอื่น ๆ ที่ไม่ใช่เรคคอร์ดแรก):

2.1) ฟังก์ชันจะเปิดไฟล์ในโหมดเพิ่มข้อมูล (ab) ซึ่งจะทำการเขียนข้อมูลต่อท้ายจากข้อมูลที่มีอยู่ในไฟล์

2.2) กระบวนการแปลงข้อมูล (pack) และการเขียนข้อมูลลงในไฟล์จะเหมือนกับกรณีของเรคคอร์ดแรก

```
39          else:
40              with open(bin_path, "ab") as file:
41                  data = struct.pack("20s20s20s20sf", record[0].encode(), record[1].encode(
42                  ), record[2].encode(), ' '.join(map(str, record[3])).encode(), record[4])
43                  file.write(data)
```

ภาพที่ 3-9 ถ้าเรคคอร์ดอื่น ๆ ที่ไม่ใช่เรคคอร์ดแรกในการเขียนข้อมูลลงไฟล์

3.3.3 การจัดรูปแบบข้อมูลก่อนเขียน

ข้อมูลแต่ละเรคคอร์ดจะถูกแปลงให้เป็นข้อมูลไบนารีผ่านฟังก์ชัน struct.pack โดย:

1) ข้อมูลตัวอักษร (strings) จะถูกเข้ารหัส (encode) เป็นข้อมูลไบนารี

2) ข้อมูลที่เป็นลิสต์ของตัวเลข (record[3]) จะถูกแปลงเป็น string โดยใช้ join(map(str, record[3])) แล้วเข้ารหัสเป็นไบนารี

3) ข้อมูล float (record[4]) จะถูกเก็บโดยตรงในรูปแบบ float ในไบนารี

3.3.4 ผลลัพธ์ที่ได้

ฟังก์ชันจะทำการเขียนข้อมูลที่ได้รับมาในรูปแบบไบนารีลงในไฟล์ตามรูปแบบที่กำหนด โดยการเขียนข้อมูลเรคคอร์ดจะเขียนทีละเรคคอร์ดต่อเนื่องกันในไฟล์

3.3.5 ตัวอย่างการเรียกใช้

ฟังก์ชันนี้สามารถเรียกใช้ด้วย writeDataToBinFile(list_records) โดยที่ list_records เป็นลิสต์ของเรคคอร์ดที่ต้องการเขียนลงในไฟล์

3.4 การแก้ไขข้อมูลในไฟล์ไบนารี

ฟังก์ชันนี้ชื่อว่า editData มีหน้าที่แก้ไขข้อมูลในไฟล์ไบนารีตามคอลัมน์ที่ต้องการแก้ไข โดยรับข้อมูลระบุว่าจะแก้ไขคอลัมน์ไหน (pointed_col), ข้อมูลระบุผู้ใช้ที่จะแก้ไข (id), ข้อมูลใหม่ที่จะแก้ไข (new_data), ตัวเลือกการแก้ไข (choice_edit), และคอลัมน์ที่เลือก (selected_col)

```python
def editData(pointed_col: str, id: tuple, new_data, choice_edit:str=None, selected_col:list=None):
    list_records = readDataFromBinFile()
    match pointed_col:
        case 'Name':       You, 6 days ago • # ...
        case 'Department':...
        case 'Score':...
        case 'Salary':...
        case _:
            print("Invalid choice. Please try again.")
            return
    writeDataToBinFile(list_records)
```

ภาพที่ 3-10 ฟังก์ชันของการแก้ไขข้อมูลในไฟล์ไบนารี

3.4.1 การอ่านข้อมูลก่อนแก้ไขข้อมูล

ฟังก์ชันเริ่มต้นด้วยการอ่านข้อมูลทั้งหมดจากไฟล์ไบนารีด้วยการเรียกใช้ฟังก์ชัน readDataFromBinFile() และเก็บข้อมูลไว้ในตัวแปร list_records ซึ่งเป็นลิสต์ของเรคคอร์ด

3.4.2 การเลือกคอลัมน์ที่จะแก้ไขข้อมูล

```python
48      match pointed_col:
```

ภาพที่ 3-11 ใช้ฟังก์ชัน match เพื่อตรวจสอบว่าผู้ใช้ต้องการแก้ไขคอลัมน์ใด

ใช้ฟังก์ชัน match เพื่อตรวจสอบว่าผู้ใช้ต้องการแก้ไขคอลัมน์ใด โดย pointed_col จะระบุคอลัมน์ที่ต้องการแก้ไข ซึ่งสามารถเลือกได้ระหว่าง:

1) 'Name' (ชื่อ)
2) 'Department' (แผนก)
3) 'Score' (คะแนน)
4) 'Salary' (เงินเดือน)

3.4.3 กรณีการแก้ไขตามคอลัมน์

```python
    case 'Name':
        for i in range(len(id)):
            for index, record in enumerate(list_records):
                if record[0] == id[i]:
                    try:
                        list_records[index][1] = new_data[i]
                    except IndexError:
                        raise Exception(f"Input data is not enough for {id[i]}.(required: {len(id)}, got: {len(new_data)})")
```

ภาพที่ 3-12 การแก้ไขคอลัมน์ 'Name'

1) การแก้ไขคอลัมน์ 'Name'

1.1) ฟังก์ชันจะวนลูปตามจำนวนผู้ใช้ (id) และค้นหาเรคคอร์ดที่มีรหัส (record[0]) ตรงกับ id ที่ได้รับมา

```python
            for i in range(len(id)):
                for index, record in enumerate(list_records):
                    if record[0] == id[i]:
```

ภาพที่ 3-13 การวนลูปเพื่อแก้ไขในคอลัมน์ 'Name'

1.2) ถ้าพบ เรคคอร์ดนั้นจะถูกแก้ไขชื่อ (คอลัมน์ 1 หรือ record[1]) ด้วยข้อมูลใหม่ (new_data[i])

```
52                  if record[0] == id[i]:
53                      try:
54                          list_records[index][1] = new_data[i]
```

ภาพที่ 3-14 แก้ไขเมื่อพบเรคคอร์ดที่ตรงกับไอดีผู้ใช้

1.3) มีการตรวจสอบหากข้อมูลใหม่ไม่เพียงพอ จะส่งข้อผิดพลาดกลับมา

```
55              except IndexError:
56                  raise Exception(f"Input data is not enough for {id[i]}.(required: {len(id)}, got: {len(new_data)})")
```

ภาพที่ 3-15 การตรวจสอบหากข้อมูลใหม่มีจำนวนไม่สอดคล้องกับจำนวนไอดี

2) การแก้ไขคอลัมน์ 'Department'

```
57          case 'Department':
58              for i in range(len(id)):
59                  for index, record in enumerate(list_records):
60                      if record[0] == id[i]:
61                          try:
62                              list_records[index][2] = new_data[i]
63                          except IndexError:
64                              list_records[index][2] = new_data[-1]
```

ภาพที่ 3-16 การแก้ไขคอลัมน์ 'Department'

2.1) ฟังก์ชันจะทำงานคล้ายกับการแก้ไขชื่อ โดยแก้ไขข้อมูลในคอลัมน์ 2 หรือ record[2] (แผนก)

```
57          case 'Department':
58              for i in range(len(id)):
59                  for index, record in enumerate(list_records):
60                      if record[0] == id[i]:
61                          try:
62                              list_records[index][2] = new_data[i]
```

ภาพที่ 3-17 การแก้ไขแผนกจะทำงานคล้ายกับการแก้ไขชื่อ

2.2) ถ้าข้อมูลไม่เพียงพอ ฟังก์ชันจะใช้ข้อมูลสุดท้ายที่เหลืออยู่

```
64                              list_records[index][2] = new_data[-1]
63                          except IndexError:
62                              list_records[index][2] = new_data[i]
61                          try:
60                      if record[0] == id[i]:
59                  for index, record in enumerate(list_records):
58              for i in range(len(id)):
57          case 'Department':
```

ภาพที่ 3-18 หากข้อมูลแผนกไม่เพียงพอจะใช้ข้อมูลแผนกสุดท้ายที่เหลืออยู่

3) การแก้ไขคอลัมน์ 'Score'

```
65      case 'Score':
66          match choice_edit:
67              case '1':
68                  for i in range(len(id)):
69                      for index, record in enumerate(list_records):
70                          if record[0] == id[i]:
71                              for score in new_data[i]:
72                                  list_records[index][3].append(score)
73                              while len(list_records[index][3]) > 4:
74                                  list_records[index][3].pop(0)
75              case '2':
76                  for i in range(len(id)):
77                      for index, record in enumerate(list_records):
78                          if record[0] == id[i]:
79                              for col in selected_col[i]:
80                                  list_records[index][3][int(col)] = new_data[i][selected_col[i].index(col)]
```

ภาพที่ 3-19 การแก้ไขคอลัมน์ 'Score'

ฟังก์ชันตรวจสอบการแก้ไขด้วยตัวเลือก choice_edit ดังนี้:

3.1) '1': เพิ่มคะแนนใหม่เข้าไปในลิสต์ของคะแนน (record[3]) โดยเพิ่มไปเรื่อย ๆ จนเกิน 4 คะแนน จากนั้นจะลบคะแนนแรกๆ ออกเพื่อให้เหลือ 4 คะแนน

```
71                              for score in new_data[i]:
72                                  list_records[index][3].append(score)
73                              while len(list_records[index][3]) > 4:
74                                  list_records[index][3].pop(0)
```

ภาพที่ 3-20 เพิ่มคะแนนใหม่เข้าไปในลิสต์ของคะแนนเดิม

3.2) '2': แก้ไขคะแนนตามคอลัมน์ที่เลือก (selected_col) โดยใช้ข้อมูลใหม่จาก new_data[i] และอัปเดตคะแนนในคอลัมน์ที่เลือก

```
79                              for col in selected_col[i]:
80                                  list_records[index][3][int(col)] = new_data[i][selected_col[i].index(col)]
```

ภาพที่ 3-21 แก้ไขคะแนนตามคอลัมน์ที่เลือก

4) การแก้ไขคอลัมน์ 'Salary'

```
82              match choice_edit:
83                  case '1':
84                      for i in range(len(id)):
85                          for index, record in enumerate(list_records):
86                              if record[0] == id[i]:
87                                  list_records[index][4] += new_data
88                  case '2':
89                      for i in range(len(id)):
90                          for index, record in enumerate(list_records):
91                              if record[0] == id[i]:
92                                  list_records[index][4] -= new_data
93                  case '3':
94                      for i in range(len(id)):
95                          for index, record in enumerate(list_records):
96                              if record[0] == id[i]:
97                                  list_records[index][4] = new_data[i]
```

ภาพที่ 3-22 การแก้ไขคอลัมน์ 'Salary' ด้วยตัวเลือก

ฟังก์ชันตรวจสอบการแก้ไขด้วยตัวเลือก choice_edit ดังนี้:

4.1) '1': เพิ่มเงินเดือน

```
83              case '1':
84                  for i in range(len(id)):
85                      for index, record in enumerate(list_records):
86                          if record[0] == id[i]:
87                              list_records[index][4] += new_data
```

ภาพที่ 3-23 การเพิ่มเงินเดือน

4.2) '2': ลดเงินเดือน

```
88              case '2':
89                  for i in range(len(id)):
90                      for index, record in enumerate(list_records):
91                          if record[0] == id[i]:
92                              list_records[index][4] -= new_data
```

ภาพที่ 3-24 การลดเงินเดือน

4.3) '3': ตั้งค่าเงินเดือนใหม่

```
93              case '3':
94                  for i in range(len(id)):
95                      for index, record in enumerate(list_records):
96                          if record[0] == id[i]:
97                              list_records[index][4] = new_data[i]
```

ภาพที่ 3-25 การตั้งค่าเงินเดือนใหม่

3.4.4 การจัดการข้อมูลที่ไม่ถูกต้อง

หาก pointed_col ไม่ตรงกับตัวเลือกที่รองรับ ฟังก์ชันจะพิมพ์ข้อความแจ้งเตือนว่าทางเลือกไม่ถูกต้อง และจะสิ้นสุดการทำงาน

3.4.5 การเขียนข้อมูลกลับลงไฟล์

เมื่อแก้ไขข้อมูลเสร็จแล้ว ฟังก์ชันจะเขียนข้อมูลที่ถูกแก้ไขกลับลงไปในไฟล์ไบนารีด้วยการเรียกใช้ฟังก์ชัน writeDataToBinFile(list_records)

3.4.6 ผลลัพธ์ที่ได้

ฟังก์ชันนี้จะทำการแก้ไขข้อมูลในไฟล์ตามคอลัมน์ที่ผู้ใช้ระบุ โดยผลลัพธ์คือข้อมูลที่ถูกแก้ไขแล้วจะถูกเขียนกลับลงไปในไฟล์ไบนารี

3.4.7 ตัวอย่างการเรียกใช้

ฟังก์ชัน editData(pointed_col: str, id: tuple, new_data, choice_edit:str=None, selected_col:list=None) สามารถเรียกใช้ได้ดังนี้:

1) การแก้ไขคอลัมน์ 'Name':

editData('Name', ['0001', '0002', '0003'], ['Silfy', 'Vyne', 'Buck'])

2) การแก้ไขคอลัมน์ 'Department'

editData('Department', ['0001', '0002', '0003'], ['IT', 'HR', 'IT'])

3) การแก้ไขคอลัมน์ 'Score'

editData('Score', ['0001', '0002', '0003'], [[97, 98, 99], [45, 46, 47], [50, 60, 70]], '2', [['1', '3', '2'], ['1', '2', '3'], ['3', '2', '1']])

4) การแก้ไขคอลัมน์ 'Salary'

editData('Salary', ['0001', '0002', '0003'], [10000, 20000, 30000], '3')

3.5 การเพิ่มข้อมูลใหม่ลงในไฟล์ไบนารี

ฟังก์ชันนี้ชื่อว่า addData มีหน้าที่เพิ่มเรคคอร์ดใหม่เข้าไปในไฟล์ไบนารี โดยรับข้อมูลใหม่ 5 ชุด ได้แก่ id (รหัส), name (ชื่อ), department (แผนก), score (คะแนนในรูปแบบลิสต์), และ salary (เงินเดือน)

```
103  def addData(id: str, name: str, department: str, score: list, salary: float):
104      try:
105          list_records = readDataFromBinFile()
106          for record in list_records:
107              if record[0] == id:
108                  raise Exception(f"ID {id} already exists.")
109          list_records.append([id, name, department, score, salary])
110      except Exception as e:
111          print(e)
112      else:
113          print(f"ID {id} has been added successfully.")
114          with open(bin_path, "ab") as file:
115              data = struct.pack("20s20s20s20sf", id.encode(), name.encode(
116              ), department.encode(), ' '.join(map(str, score)).encode(), salary)
117              file.write(data)
```

ภาพที่ 3-26 การเพิ่มข้อมูลใหม่ลงในไฟล์ไบนารี

3.5.1 การอ่านข้อมูลจากไฟล์เพื่อตรวจสอบ ID ซ้ำซ้อน

ฟังก์ชันเริ่มต้นด้วยการอ่านข้อมูลจากไฟล์ไบนารีโดยเรียกใช้ฟังก์ชัน readDataFromBinFile() และเก็บข้อมูลทั้งหมดไว้ในตัวแปร list_records

3.5.2 การตรวจสอบ ID ซ้ำ

ฟังก์ชันจะตรวจสอบว่ามี id ที่ได้รับมาอยู่ใน list_records แล้วหรือไม่

1) ใช้การวนลูปเพื่อเช็กว่า record[0] (ซึ่งเก็บ ID ของแต่ละเรคคอร์ด) ตรงกับ id ที่ได้รับมา

2) ถ้าพบว่ามี ID ซ้ำ ฟังก์ชันจะส่งข้อผิดพลาดโดยการ raise Exception และพิมพ์ข้อความแจ้งว่ามี ID ซ้ำแล้ว ซึ่งจะหยุดการทำงานของฟังก์ชัน

```
104     try:
105         list_records = readDataFromBinFile()
106         for record in list_records:
107             if record[0] == id:
108                 raise Exception(f"ID {id} already exists.")
```

ภาพที่ 3-27 การตรวจสอบ ID ซ้ำ

3.5.3 การเพิ่มข้อมูลใหม่

ถ้าไม่มีข้อผิดพลาดเกิดขึ้น (ไม่มี ID ซ้ำ)

1) ข้อมูลใหม่จะถูกเพิ่มเข้าไปในลิสต์ list_records ในรูปแบบของลิสต์ที่มี 5 คอลัมน์ ได้แก่ id, name, department, score, และ salary

```
109         list_records.append([id, name, department, score, salary])
```

ภาพที่ 3-28 การเพิ่มข้อมูลใหม่เข้าไปในลิสต์ list_records

2) ฟังก์ชันจะแจ้งข้อความว่า ID ที่ได้รับมาได้ถูกเพิ่มสำเร็จแล้ว (ID {id} has been added successfully.)

```
113         print(f"ID {id} has been added successfully.")
```

ภาพที่ 3-29 การจะแจ้งข้อความว่า ID ที่ได้รับมาได้ถูกเพิ่มสำเร็จแล้ว

3.5.4 การเขียนข้อมูลใหม่ลงไฟล์

1) ฟังก์ชันจะเปิดไฟล์ในโหมดเพิ่มข้อมูล (ab) ซึ่งเป็นการเขียนข้อมูลต่อท้ายไฟล์โดยไม่ลบข้อมูลเดิม

```
114         with open(bin_path, "ab") as file:
```

ภาพที่ 3-30 การเปิดไฟล์ในโหมดเพิ่มข้อมูลเพื่อเพิ่มข้อมูลใหม่ลงไฟล์

2) ใช้ struct.pack("20s20s20s20sf", ...) ในการบรรจุข้อมูลใหม่ในรูปแบบไบนารี:

2.1) id, name, และ department จะถูกแปลงเป็น string แล้วเข้ารหัสเป็นไบนารีด้วย .encode()

2.2) score ซึ่งเป็นลิสต์ของคะแนนจะถูกแปลงเป็น string ด้วยการใช้ join(map(str, score)) แล้วเข้ารหัสเป็นไบนารี

2.3) salary จะถูกบรรจุเป็นตัวเลข float ตามรูปแบบที่กำหนด

```
115         data = struct.pack("20s20s20s20sf", id.encode(), name.encode(
116         ), department.encode(), ' '.join(map(str, score)).encode(), salary)
```

ภาพที่ 3-31 การใช้ struct.pack ในการบรรจุข้อมูลใหม่ในรูปแบบไบนารี

3) ข้อมูลที่ถูกบรรจุเป็นไบนารีจะถูกเขียนลงไปในไฟล์ด้วย file.write(data)

3.5.5 ผลลัพธ์ที่ได้

ฟังก์ชันนี้จะเพิ่มข้อมูลใหม่เข้าไปในไฟล์ไบนารี โดยจะทำการตรวจสอบว่า ID นั้นมีอยู่แล้วหรือไม่ ถ้าไม่มี ID ซ้ำ ข้อมูลใหม่จะถูกเพิ่มลงไฟล์

3.5.6 ตัวอย่างการเรียกใช้

ตัวอย่างนี้จะเพิ่มเรคคอร์ดใหม่สำหรับผู้ใช้ที่มี ID "002" ชื่อ "Jane Smith" ทำงานในแผนก HR มีคะแนน และเงินเดือนตามที่ระบุ: addData("002", "Jane Smith", "HR", [80.5, 90.0, 85.0, 70.0], 50000.0)

3.6 การลบข้อมูลในไฟล์ไบนารี

ฟังก์ชันนี้ชื่อว่า deleteData มีหน้าที่ลบเรคคอร์ดในไฟล์ไบนารีตามรหัส (ID) ที่ได้รับจากลิสต์ multi_id ซึ่งเป็นลิสต์ของ ID ที่ต้องการลบ

```
119 def deleteData(multi_id: list):
120     list_records = readDataFromBinFile()
121     all_user_id = [user_data[0] for user_data in list_records]
122     for id in multi_id:
123         if id not in all_user_id:
124             print(f"ID {id} not found.")
125     list_index = []
126     for index, record in enumerate(list_records):
127         if record[0] in multi_id:
128             list_index.append(index)
129     list_index.sort(reverse=True)
130     list_deleted = []
131     for index in list_index:
132         list_deleted.append(list_records.pop(index))
133     writeDataToBinFile(list_records)
134     for record in list_deleted:
135         print(f"ID {record[0]} has been deleted successfully.")
```

ภาพที่ 3-32 การลบข้อมูลในไฟล์ไบนารี

3.6.1 การอ่านข้อมูลก่อนลบข้อมูล

ฟังก์ชันเริ่มต้นด้วยการอ่านข้อมูลทั้งหมดจากไฟล์ไบนารีโดยใช้ฟังก์ชัน readDataFromBinFile() และเก็บข้อมูลทั้งหมดไว้ในตัวแปร list_records ซึ่งเป็นลิสต์ของเรคคอร์ด

3.6.2 การตรวจสอบ ID ที่จะลบ

1) สร้างลิสต์ all_user_id ซึ่งประกอบด้วย ID ของผู้ใช้ทั้งหมดใน list_records โดยใช้การสร้างลิสต์อย่างย่อ (list comprehension) จาก record[0] ซึ่งเป็นคอลัมน์ ID ของแต่ละเรคคอร์ด

```
121     all_user_id = [user_data[0] for user_data in list_records]
```

ภาพที่ 3-33 สร้างลิสต์ all_user_id ซึ่งประกอบด้วย ID ของผู้ใช้ทั้งหมด

2) วนลูปตรวจสอบ ID ที่ได้รับจาก multi_id ว่ามีอยู่ใน all_user_id หรือไม่:

หาก ID ไม่พบใน list_records ฟังก์ชันจะแสดงข้อความว่า ID {id} not found. แต่จะยังคงดำเนินการลบ ID อื่นๆ ที่พบต่อไป

```
122     for id in multi_id:
123         if id not in all_user_id:
124             print(f"ID {id} not found.")
```

ภาพที่ 3-34 การวนลูปตรวจสอบ ID ที่ได้รับจาก multi_id ว่ามีอยู่ใน all_user_id หรือไม่

3.6.3 การลบเรคคอร์ด

1) สร้างลิสต์ list_index ซึ่งเก็บตำแหน่งของเรคคอร์ดที่มี ID ตรงกับ ID ที่ได้รับจาก multi_id

ใช้การวนลูปเพื่อค้นหาตำแหน่งของแต่ละเรคคอร์ด (index) ที่มี ID ตรงกัน และเก็บค่า index ไว้ใน list_index

```
125     list_index = []
126     for index, record in enumerate(list_records):
127         if record[0] in multi_id:
128             list_index.append(index)
```

ภาพที่ 3-35 การวนลูปเพื่อค้นหาตำแหน่งของแต่ละเรคคอร์ด (index) ที่มี ID ตรงกัน

2) จัดเรียง list_index ในลำดับจากมากไปน้อย (จาก index สูงไปต่ำ) โดยใช้ sort(reverse=True) เพื่อป้องกันปัญหาในการลบเรคคอร์ดหลายตัวจากลิสต์ ซึ่งอาจทำให้ตำแหน่งของเรคคอร์ดที่เหลือเปลี่ยนไป

```
129     list_index.sort(reverse=True)
```

ภาพที่ 3-36 จัดเรียง list_index ในลำดับจากมากไปน้อย

3) ลบเรคคอร์ดจาก list_records ตามตำแหน่งที่บันทึกใน list_index โดยใช้ pop() และเก็บเรคคอร์ดที่ถูกลบไว้ใน list_deleted

```
130     list_deleted = []
131     for index in list_index:
132         list_deleted.append(list_records.pop(index))
```

ภาพที่ 3-37 ลบเรคคอร์ดจาก list_records ตามตำแหน่งที่บันทึกใน list_index

3.6.4 การเขียนข้อมูลกลับลงไฟล์

ฟังก์ชันเขียนข้อมูลที่เหลืออยู่ใน list_records หลังจากการลบ กลับลงไปในไฟล์ไบนารี โดยเรียกใช้ฟังก์ชัน writeDataToBinFile(list_records)

3.6.5 การแสดงผล ID ที่ถูกลบ

ฟังก์ชันจะแสดงข้อความว่ารหัส ID แต่ละตัวที่ถูกลบสำเร็จ (ID {record[0]} has been deleted successfully.) โดยใช้ข้อมูลจากลิสต์ list_deleted ซึ่งเก็บเรคคอร์ดที่ถูกลบออก

```
134     for record in list_deleted:
135         print(f"ID {record[0]} has been deleted successfully.")
```

ภาพที่ 3-38 การแสดงผล ID ที่ถูกลบ

3.6.6 ผลลัพธ์ที่ได้

ฟังก์ชันนี้จะทำการลบเรคคอร์ดในไฟล์ตาม ID ที่ระบุ หาก ID ไม่พบ ฟังก์ชันจะแสดงข้อความแจ้งเตือน แต่ยังคงลบเรคคอร์ดที่มี ID ตรงกับที่ระบุไว้ต่อไป

3.6.7 ตัวอย่างการเรียกใช้

ตัวอย่างนี้จะทำการลบเรคคอร์ดที่มี ID "001", "003", และ "005" จากไฟล์ไบนารี หากพบว่า ID ใดไม่อยู่ในไฟล์ ฟังก์ชันจะแจ้งข้อความว่าหาไม่พบ: deleteData(["001", "003", "005"])

3.7 การสร้างรายงานสรุปข้อมูลและบันทึกเป็นไฟล์ข้อความ

ฟังก์ชันนี้ชื่อว่า exportReport ทำหน้าที่สร้างรายงานสรุปผลการทำงานของพนักงานตามแผนก และบันทึกรายงานนี้ลงในไฟล์ข้อความชื่อ report.txt

```python
def exportReport():
    data_bin_file = np.array(readDataFromBinFile(), dtype="object")
    departments = {}
    for record in data_bin_file:
        _, name, dept, scores, _ = record
        avg_score = sum(scores) / len(scores)
        if dept not in departments:
            departments[dept] = {
                'employees': [], 'total_score': 0.0, 'count': 0, 'top_performer': (None, 0)}
        departments[dept]['employees'].append((name, avg_score))
        departments[dept]['total_score'] += avg_score
        departments[dept]['count'] += 1
        if avg_score > departments[dept]['top_performer'][1]:
            departments[dept]['top_performer'] = (name, avg_score)
    best_department = (None, 0)
    str_report = ""
    str_report += "Summary Report:\n"
    for dept, info in departments.items():
        avg_dept_score = info['total_score'] / info['count']
        str_report += f"Department: {dept}\n"
        for name, avg_score in info['employees']:
            str_report += f"  {name}: Average Score = {avg_score:.2f}\n"
        top_performer, top_score = info['top_performer']
        str_report += f"Top Performer: {top_performer} Average score = {top_score:.2f}\n"
        if avg_dept_score > best_department[1]:
            best_department = (dept, avg_dept_score)
    str_report += f"\nBest Department: {best_department[0]} Average score = {best_department[1]:.2f}\n"
    with open("report.txt", "w") as file:
        file.write(str_report)
    print("Report exported to report.txt successfully.")
```

ภาพที่ 3-39 การสร้างรายงานสรุปข้อมูลและบันทึกเป็นไฟล์ข้อความ

3.7.1 การอ่านข้อมูลจากไฟล์

ฟังก์ชันเริ่มต้นด้วยการอ่านข้อมูลทั้งหมดจากไฟล์ไบนารีโดยเรียกใช้ฟังก์ชัน readDataFromBinFile() และแปลงข้อมูลเป็นอาเรย์ของ NumPy โดยใช้ np.array() และกำหนดชนิดข้อมูลเป็น dtype="object" เพื่อให้รองรับข้อมูลประเภทต่าง ๆ ที่เก็บอยู่ในไฟล์

```python
    data_bin_file = np.array(readDataFromBinFile(), dtype="object")
```

ภาพที่ 3-40 การอ่านข้อมูลจากไฟล์โดยใช้ numpy

3.7.2 การจัดกลุ่มตามแผนก

1) สร้างตัวแปร departments ในรูปแบบของดิกชันนารี เพื่อเก็บข้อมูลของพนักงานที่จัดกลุ่มตามแผนก

2) วนลูปในแต่ละเรคคอร์ดใน data_bin_file เพื่อดึงชื่อ (name), แผนก (dept), และคะแนนเฉลี่ย (avg_score) ของพนักงานแต่ละคน โดยคำนวณคะแนนเฉลี่ยจากลิสต์ scores:

```python
    for record in data_bin_file:
        _, name, dept, scores, _ = record
        avg_score = sum(scores) / len(scores)
```

ภาพที่ 3-41 วนลูปในแต่ละเรคคอร์ดใน data_bin_file เพื่อดึงข้อมูล

2.1) ถ้าแผนกของพนักงานไม่อยู่ในดิกชันนารี departments ให้สร้างแผนกนั้นขึ้นมา และกำหนดค่าเริ่มต้น เช่น รายชื่อพนักงาน (employees), คะแนนรวม (total_score), จำนวนพนักงาน (count), และพนักงานที่ได้คะแนนสูงสุด (top_performer)

```
154         if dept not in departments:
155             departments[dept] = {
156                 'employees': [], 'total_score': 0.0, 'count': 0, 'top_performer': (None, 0)}
```

ภาพที่ 3-42 การสร้างแผนกของพนักงานไม่อยู่ในดิกชันนารี departments

2.2) เพิ่มชื่อและคะแนนเฉลี่ยของพนักงานในรายการของแผนกนั้น

```
157             departments[dept]['employees'].append((name, avg_score))
```

ภาพที่ 3-43 การเพิ่มชื่อและคะแนนเฉลี่ยของพนักงานในรายการของแผนก

2.3) รวมคะแนนเฉลี่ยของพนักงานเข้าในคะแนนรวม (total_score) และเพิ่มจำนวนพนักงานในแผนก

```
158             departments[dept]['total_score'] += avg_score
159             departments[dept]['count'] += 1
```

ภาพที่ 3-44 การรวมคะแนนเฉลี่ยของพนักงานเข้าและเพิ่มจำนวนพนักงานในแผนก

2.4) ถ้าคะแนนเฉลี่ยของพนักงานคนนี้มากกว่าคะแนนเฉลี่ยสูงสุดเดิมของแผนก ฟังก์ชันจะบันทึกให้พนักงานคนนี้เป็นผู้ทำคะแนนสูงสุดในแผนก (top_performer)

```
160             if avg_score > departments[dept]['top_performer'][1]:
161                 departments[dept]['top_performer'] = (name, avg_score)
```

ภาพที่ 3-45 การบันทึกพนักงานที่ทำคะแนนสูงสุดในแผนก

3.7.3 การหาพนักงานของแผนกที่ดีที่สุด

1) หลังจากวนลูปเพื่อเก็บข้อมูลทุกแผนกแล้ว ฟังก์ชันจะเริ่มต้นหาผลงานของแผนกที่ดีที่สุด โดยคำนวณคะแนนเฉลี่ยของแผนก (avg_dept_score) จาก total_score หารด้วยจำนวนพนักงาน (count)

```
160             if avg_score > departments[dept]['top_performer'][1]:
161                 departments[dept]['top_performer'] = (name, avg_score)
```

ภาพที่ 3-46 การหาพนักงานของแผนกที่ดีที่สุด

2) ถ้าคะแนนเฉลี่ยของแผนกใดมากกว่าคะแนนเฉลี่ยสูงสุดเดิม ฟังก์ชันจะบันทึกแผนกนั้นเป็นแผนกที่ดีที่สุด

```
172         if avg_dept_score > best_department[1]:
173             best_department = (dept, avg_dept_score)
```

ภาพที่ 3-47 คะแนนเฉลี่ยของแผนกใดมากกว่าคะแนนเฉลี่ยสูงสุดเดิม

3.7.4 การสร้างข้อความรายงาน

```
    best_department = (None, 0)
    str_report = ""
    str_report += "Summary Report:\n"
    for dept, info in departments.items():
        avg_dept_score = info['total_score'] / info['count']
        str_report += f"Department: {dept}\n"
        for name, avg_score in info['employees']:
            str_report += f"  {name}: Average Score = {avg_score:.2f}\n"
        top_performer, top_score = info['top_performer']
        str_report += f"Top Performer: {top_performer} Average score = {top_score:.2f}\n"
        if avg_dept_score > best_department[1]:
            best_department = (dept, avg_dept_score)
    str_report += f"\nBest Department: {best_department[0]} Average score = {best_department[1]:.2f}\n"
```

ภาพที่ 3-48 การสร้างข้อความรายงาน

1) ฟังก์ชันสร้างสตริงรายงาน (str_report) โดยมีหัวข้อหลัก "Summary Report" และข้อมูลรายละเอียดของแต่ละแผนก:

1.1) แสดงชื่อแผนก และข้อมูลพนักงานแต่ละคนในแผนกพร้อมคะแนนเฉลี่ย (Average Score)

1.2) แสดงพนักงานที่ทำคะแนนสูงสุดในแผนก (Top Performer)

2) สุดท้าย ฟังก์ชันจะแสดงชื่อแผนกที่มีคะแนนเฉลี่ยดีที่สุด (Best Department) พร้อมกับคะแนนเฉลี่ยของแผนกนั้น

3.7.5 การบันทึกลงไฟล์

ฟังก์ชันบันทึกสตริงรายงานลงในไฟล์ข้อความชื่อ report.txt โดยใช้ with open("report.txt", "w") ซึ่งจะเปิดไฟล์ในโหมดเขียน (ถ้าไฟล์มีอยู่แล้วจะถูกเขียนทับ)

```
    with open("report.txt", "w") as file:
        file.write(str_report)
```

ภาพที่ 3-49 บันทึกสตริงรายงานลงในไฟล์ข้อความชื่อ report.txt

3.7.6 การแสดงผล

ฟังก์ชันพิมพ์ข้อความว่า "Report exported to report.txt successfully." เมื่อการเขียนรายงานลงไฟล์สำเร็จ

3.7.7 ผลลัพธ์ที่ได้

ฟังก์ชันนี้จะสร้างรายงานสรุปผลการทำงานของพนักงานในแต่ละแผนก โดยแสดงข้อมูลพนักงานในแผนก คะแนนเฉลี่ยของแต่ละคน และพนักงานที่ทำคะแนนสูงสุดในแผนกนั้น รวมถึงระบุแผนกที่มีคะแนนเฉลี่ยดีที่สุด และบันทึกรายงานทั้งหมดลงในไฟล์ report.txt

3.7.8 ตัวอย่างการเรียกใช้

เมื่อเรียกฟังก์ชันนี้ จะสร้างรายงานสรุปผลของพนักงานทุกคนตามแผนกและบันทึกลงไฟล์: exportReport()

3.8 แสดงข้อมูลของพนักงานทั้งหมด

ฟังก์ชันนี้ชื่อว่า showAllData ทำหน้าที่แสดงข้อมูลพนักงานทั้งหมดจากไฟล์ไบนารี ซึ่งสามารถเลือกได้ว่าจะให้แสดงเฉลี่ยคะแนนหรือแสดงคะแนนแบบเต็ม

```
180 def showAllData(choice=None):
181     list_records = readDataFromBinFile()
182     np.seterr(all='ignore')
183     choice = choice
184     match choice:
185         case '1':
186             list_records = [(record[0], record[1], record[2], np.mean(
187                 record[3]), record[4]) for record in list_records]
188             dtype = [('ID', 'U20'),
189                      ('Name', 'U20'),
190                      ('Department', 'U20'),
191                      ('Average Score', 'f4'),
192                      ('Salary', 'f4')]
193             data_arr = np.asarray(list_records, dtype=dtype)
194         case '2':
195             list_records = [(record[0], record[1], record[2], ', '.join(
196                 list(map(str, record[3]))), record[4]) for record in list_records]
197             dtype = [('ID', 'U20'),
198                      ('Name', 'U20'),
199                      ('Department', 'U20'),
200                      ('Score', 'U20'),
201                      ('Salary', 'f4')]
202             data_arr = np.asarray(list_records, dtype=dtype)
203         case _:
204             print("Invalid choice. Please try again.")
205             return
206     sorted_data = np.sort(data_arr, order='ID')
207     df_data_arr = pd.DataFrame(sorted_data)
208     if df_data_arr.empty:
209         print("Data not found.")
210         return
211     print(df_data_arr.to_string(index=False))
```

ภาพที่ 3-50 การแสดงข้อมูลของพนักงานทั้งหมด

3.8.1 การอ่านข้อมูลจากไฟล์

ฟังก์ชันเริ่มต้นด้วยการเรียกใช้ฟังก์ชัน readDataFromBinFile() เพื่ออ่านข้อมูลทั้งหมดจากไฟล์ไบนารี และเก็บข้อมูลไว้ในตัวแปร list_records ซึ่งเป็นลิสต์ของเรคคอร์ด

3.8.2 การเลือกโหมดการแสดงข้อมูล

```
184     match choice:
185         case '1':
186             list_records = [(record[0], record[1], record[2], np.mean(
187                 record[3]), record[4]) for record in list_records]
188             dtype = [('ID', 'U20'),
189                      ('Name', 'U20'),
190                      ('Department', 'U20'),
191                      ('Average Score', 'f4'),
192                      ('Salary', 'f4')]
193             data_arr = np.asarray(list_records, dtype=dtype)
194         case '2':
195             list_records = [(record[0], record[1], record[2], ', '.join(
196                 list(map(str, record[3]))), record[4]) for record in list_records]
197             dtype = [('ID', 'U20'),
198                      ('Name', 'U20'),
199                      ('Department', 'U20'),
200                      ('Score', 'U20'),
201                      ('Salary', 'f4')]
202             data_arr = np.asarray(list_records, dtype=dtype)
```

ภาพที่ 3-51 การเลือกโหมดการแสดงข้อมูล

ฟังก์ชันรับพารามิเตอร์ choice เพื่อระบุว่าจะแสดงข้อมูลในรูปแบบใด:

1) ถ้า choice คือ '1' ฟังก์ชันจะแสดงข้อมูลพร้อมคะแนนเฉลี่ยของแต่ละพนักงาน

2) ถ้า choice คือ '2' ฟังก์ชันจะแสดงข้อมูลพร้อมคะแนนของแต่ละพนักงานในรูปแบบลิสต์ที่คั่นด้วยเครื่องหมายจุลภาค (,)

3.8.3 การแสดงข้อมูลในกรณี choice = '1' (แสดงคะแนนเฉลี่ย)

```
185         case '1':
186             list_records = [(record[0], record[1], record[2], np.mean(
187                 record[3]), record[4]) for record in list_records]
188             dtype = [('ID', 'U20'),
189                      ('Name', 'U20'),
190                      ('Department', 'U20'),
191                      ('Average Score', 'f4'),
192                      ('Salary', 'f4')]
193             data_arr = np.asarray(list_records, dtype=dtype)
```

ภาพที่ 3-52 การแสดงข้อมูลในกรณี choice = '1' (แสดงคะแนนเฉลี่ย)

1) ฟังก์ชันจะสร้างลิสต์ใหม่จาก list_records โดยแทนที่ข้อมูลคะแนนของแต่ละพนักงาน (record[3]) ด้วยค่าเฉลี่ยของคะแนนเหล่านั้นโดยใช้ np.mean()

2) กำหนดชนิดข้อมูลของลิสต์นี้ในรูปแบบ dtype เพื่อให้รองรับฟิลด์ที่ต้องการ ได้แก่:

2.1) ID (ประเภท U20 สำหรับตัวอักษร Unicode ไม่เกิน 20 ตัว)

2.2) Name (ประเภท U20)

2.3) Department (ประเภท U20)

2.4) Average Score (คะแนนเฉลี่ย ประเภท f4 หรือจำนวนทศนิยมแบบ float)

2.5) Salary (ประเภท f4)

3) จากนั้นแปลงข้อมูลในลิสต์เป็นอาเรย์ NumPy (np.asarray()) เพื่อใช้จัดการต่อไป

3.8.4 การแสดงข้อมูลในกรณี choice = '2' (แสดงคะแนนแบบแจกแจ้ง)

```
194     case '2':
195         list_records = [(record[0], record[1], record[2], ', '.join(
196             list(map(str, record[3]))), record[4]) for record in list_records]
197         dtype = [('ID', 'U20'),
198                  ('Name', 'U20'),
199                  ('Department', 'U20'),
200                  ('Score', 'U20'),
201                  ('Salary', 'f4')]
202         data_arr = np.asarray(list_records, dtype=dtype)
```

ภาพที่ 3-53 การแสดงข้อมูลในกรณี choice = '2' (แสดงคะแนนแบบแจกแจ้ง)

1) ฟังก์ชันจะแปลงลิสต์ของข้อมูลพนักงาน โดยใช้ ', '.join() เพื่อรวมคะแนนแต่ละค่าที่อยู่ใน record[3] เป็นสตริงคั่นด้วยเครื่องหมายจุลภาค

2) กำหนดชนิดข้อมูล dtype ในรูปแบบที่คล้ายกับตัวเลือกแรก แต่ใช้ฟิลด์ Score (ประเภท U20) เพื่อแทนที่ฟิลด์ Average Score

3.8.5 การตรวจสอบและเรียงลำดับข้อมูล

```
206     sorted_data = np.sort(data_arr, order='ID')
207     df_data_arr = pd.DataFrame(sorted_data)
```

ภาพที่ 3-54 การตรวจสอบและเรียงลำดับข้อมูล

1) ฟังก์ชันจัดเรียงข้อมูลในอาเรย์โดยเรียงตามฟิลด์ ID โดยใช้ np.sort(data_arr, order='ID')

2) แปลงอาเรย์ที่จัดเรียงแล้วเป็น DataFrame ของ Pandas (pd.DataFrame (sorted_data)) เพื่อให้ง่ายต่อการจัดการและแสดงผล

3.8.6 การตรวจสอบข้อมูล

```
208     if df_data_arr.empty:
209         print("Data not found.")
210         return
211     print(df_data_arr.to_string(index=False))
```

ภาพที่ 3-55 การตรวจสอบข้อมูล

1) ถ้า DataFrame ว่างเปล่า (ไม่มีข้อมูล) ฟังก์ชันจะแสดงข้อความว่า "Data not found."

2) ถ้ามีข้อมูล ฟังก์ชันจะแสดงข้อมูลทั้งหมดในรูปแบบตารางโดยใช้คำสั่ง print(df_data_arr.to_string(index=False)) ซึ่งจะพิมพ์ข้อมูลในรูปแบบที่ไม่มีตัวเลขลำดับ (index=False)

3.8.7 ผลลัพธ์ที่ได้

ฟังก์ชันนี้จะพิมพ์ข้อมูลพนักงานทั้งหมดในรูปแบบตาราง โดยขึ้นอยู่กับตัวเลือก choice:

1) ถ้าเลือก '1' จะแสดง ID, ชื่อ, แผนก, คะแนนเฉลี่ย, และเงินเดือน จากภาพที่ 2-3

2) ถ้าเลือก '2' จะแสดง ID, ชื่อ, แผนก, คะแนนเต็มของแต่ละพนักงาน, และเงินเดือน จากภาพที่ 2-4

3.8.8 ตัวอย่างการเรียกใช้

ฟังก์ชันนี้สามารถเรียกใช้ได้โดยกำหนดตัวเลือกการแสดงข้อมูล เช่น:

1) showAllData('1') เป็นการแสดงข้อมูลพร้อมคะแนนเฉลี่ย

2) showAllData('2') เป็นการแสดงข้อมูลพร้อมคะแนนเต็ม

3.9 แสดงข้อมูลเฉพาะของพนักงานตามคอลัมน์ที่กำหนด

ฟังก์ชันนี้ชื่อว่า showSpecificData ทำหน้าที่แสดงข้อมูลเฉพาะของพนักงานตามคอลัมน์ที่กำหนด โดยสามารถกรองข้อมูลตามเงื่อนไขที่ต้องการ เช่น การค้นหาข้อมูลจากคอลัมน์ ID, Name, Department, Average Score, Score หรือ Salary

```python
def showSpecificData(col: int, list_search=None, choice=None):
    list_records = readDataFromBinFile()
    col = ['ID', 'Name', 'Department', 'Average Score',
           'Score', 'Salary'][int(col) - 1]
    if col == 'Score':
        list_records = [(record[0], record[1], record[2], ','.join(
            list(map(str, record[3]))), record[4]) for record in list_records]
        dtype = [('ID', 'U20'),
                 ('Name', 'U20'),
                 ('Department', 'U20'),
                 ('Score', 'U20'),
                 ('Salary', 'f4')]
        data_arr = np.asarray(list_records, dtype=dtype)
    else:
        list_records = [(record[0], record[1], record[2], np.mean(
            record[3]), record[4]) for record in list_records]
        dtype = [('ID', 'U20'),
                 ('Name', 'U20'),
                 ('Department', 'U20'),
                 ('Average Score', 'f4'),
                 ('Salary', 'f4')]
        data_arr = np.asarray(list_records, dtype=dtype)
    if col == 'Average Score' or col == 'Salary':
        choice = choice
        match choice:
            case '1':
                data_fltr = data_arr[data_arr[col] > float(list_search[0])]
            case '2':
                data_fltr = data_arr[data_arr[col] >= float(list_search[0])]
            case '3':
                data_fltr = data_arr[data_arr[col] < float(list_search[0])]
            case '4':
                data_fltr = data_arr[data_arr[col] <= float(list_search[0])]
            case _:
                list_search = [float(score) for score in list_search]
                data_fltr = data_arr[np.isin(data_arr[col], list_search)]
        sorted_data = np.sort(data_fltr, order=col)
    elif col == 'ID' or col == 'Name':      You, last week • #
        data_fltr = data_arr[np.isin(data_arr[col], list_search)]
        sorted_data = np.sort(data_fltr, order='ID')
    elif col == 'Department':
        data_fltr = data_arr[np.isin(data_arr[col], list_search)]
        sorted_data = np.sort(data_fltr, order='Department')
    elif col == 'Score':
        data_fltr = data_arr[np.isin(data_arr['ID'], list_search)]
        sorted_data = np.sort(data_fltr, order='ID')
    df_data_fltr = pd.DataFrame(data=sorted_data)
    if df_data_fltr.empty:
        print("No data found.")
        return
    print(df_data_fltr.to_string(index=False))
```

ภาพที่ 3-56 การแสดงข้อมูลเฉพาะของพนักงานตามคอลัมน์ที่กำหนด

3.9.1 รายละเอียดการทำงาน

ฟังก์ชันนี้รับพารามิเตอร์ 3 ตัว ได้แก่:

1) col: หมายเลขคอลัมน์ที่ต้องการค้นหา เช่น 1 (ID), 2 (Name), 3 (Department), 4 (Average Score), 5 (Score), และ 6 (Salary)

2) list_search: ลิสต์ของค่าที่ต้องการใช้ในการค้นหา

3) choice: เงื่อนไขการเปรียบเทียบค่าต่างๆ เช่น มากกว่า, น้อยกว่า, หรือเท่ากับ

3.9.2 การอ่านข้อมูลจากไฟล์

ฟังก์ชันเริ่มต้นด้วยการเรียกใช้ฟังก์ชัน readDataFromBinFile() เพื่ออ่านข้อมูลทั้งหมดจากไฟล์ไบนารี และเก็บข้อมูลไว้ในตัวแปร list_records ซึ่งเป็นลิสต์ของเรคคอร์ด

3.9.3 การกำหนดคอลัมน์ที่ต้องการค้นหา

```
216     col = ['ID', 'Name', 'Department', 'Average Score',
217            'Score', 'Salary'][int(col) - 1]
218     if col == 'Score':
219         list_records = [(record[0], record[1], record[2], ','.join(
220             list(map(str, record[3]))), record[4]) for record in list_records]
221         dtype = [('ID', 'U20'),
222                  ('Name', 'U20'),
223                  ('Department', 'U20'),
224                  ('Score', 'U20'),
225                  ('Salary', 'f4')]
226         data_arr = np.asarray(list_records, dtype=dtype)
227     else:
228         list_records = [(record[0], record[1], record[2], np.mean(
229             record[3]), record[4]) for record in list_records]
230         dtype = [('ID', 'U20'),
231                  ('Name', 'U20'),
232                  ('Department', 'U20'),
233                  ('Average Score', 'f4'),
234                  ('Salary', 'f4')]
235         data_arr = np.asarray(list_records, dtype=dtype)
```

ภาพที่ 3-57 การกำหนดคอลัมน์ที่ต้องการค้นหา

1) ฟังก์ชันจะแปลงค่าพารามิเตอร์ col เป็นชื่อของคอลัมน์ เช่น 'ID', 'Name', 'Department', 'Average Score', 'Score', หรือ 'Salary' โดยการแมปหมายเลขคอลัมน์เข้ากับชื่อคอลัมน์ในลิสต์ที่กำหนดไว้ล่วงหน้า

2) จากนั้นฟังก์ชันจะแบ่งการจัดการข้อมูลเป็นสองรูปแบบ ขึ้นอยู่กับคอลัมน์ที่เลือก

3.9.4 การจัดการข้อมูลในคอลัมน์ Score

ถ้าเลือกค้นหาคอลัมน์ Score ฟังก์ชันจะเปลี่ยน record[3] ซึ่งเป็นลิสต์ของคะแนน เป็นสตริงที่คั่นด้วยเครื่องหมายจุลภาค (,) และเก็บข้อมูลเป็นอาเรย์ NumPy พร้อมชนิดข้อมูล dtype ที่ประกอบไปด้วยฟิลด์ ID, Name, Department, Score, และ Salary

```
218     if col == 'Score':
219         list_records = [(record[0], record[1], record[2], ','.join(
220             list(map(str, record[3]))), record[4]) for record in list_records]
221         dtype = [('ID', 'U20'),
222                  ('Name', 'U20'),
223                  ('Department', 'U20'),
224                  ('Score', 'U20'),
225                  ('Salary', 'f4')]
226         data_arr = np.asarray(list_records, dtype=dtype)
```

ภาพที่ 3-58 การจัดการข้อมูลในคอลัมน์ Score

3.9.5 การจัดการข้อมูลในคอลัมน์อื่นๆ

ถ้าเลือกค้นหาคอลัมน์ที่ไม่ใช่ Score เช่น Average Score หรือ Salary ฟังก์ชันจะคำนวณค่าเฉลี่ยของคะแนนแต่ละพนักงาน (record[3]) โดยใช้ np.mean() และเก็บข้อมูลในอาเรย์ NumPy พร้อมชนิดข้อมูล dtype ที่ประกอบไปด้วยฟิลด์ ID, Name, Department, Average Score, และ Salary

```
227     else:
228         list_records = [(record[0], record[1], record[2], np.mean(
229             record[3]), record[4]) for record in list_records]
230         dtype = [('ID', 'U20'),
231                  ('Name', 'U20'),
232                  ('Department', 'U20'),
233                  ('Average Score', 'f4'),
234                  ('Salary', 'f4')]
235         data_arr = np.asarray(list_records, dtype=dtype)
```

ภาพที่ 3-59 การจัดการข้อมูลในคอลัมน์อื่นๆ

3.9.6 การเปรียบเทียบและกรองข้อมูล

1) ถ้าค้นหาคอลัมน์ที่เป็นตัวเลข เช่น Average Score หรือ Salary ฟังก์ชันจะใช้พารามิเตอร์ choice เพื่อกำหนดเงื่อนไขการกรองข้อมูล เช่น มากกว่า (>) หรือ น้อยกว่า (<) ตามค่าที่ผู้ใช้ระบุใน list_search

1.1) '1': มากกว่า (>)

1.2) '2': มากกว่าหรือเท่ากับ (>=)

1.3) '3': น้อยกว่า (<)

1.4) '4': น้อยกว่าหรือเท่ากับ (<=)

1.5) ถ้าไม่ได้เลือกเงื่อนไขเฉพาะ ฟังก์ชันจะใช้ค่าใน list_search เพื่อกรองข้อมูลที่ตรงกับค่าที่ค้นหา

```
236     if col == 'Average Score' or col == 'Salary':
237         choice = choice
238         match choice:
239             case '1':
240                 data_fltr = data_arr[data_arr[col] > float(list_search[0])]
241             case '2':
242                 data_fltr = data_arr[data_arr[col] >= float(list_search[0])]
243             case '3':
244                 data_fltr = data_arr[data_arr[col] < float(list_search[0])]
245             case '4':
246                 data_fltr = data_arr[data_arr[col] <= float(list_search[0])]
247             case _:
248                 list_search = [float(score) for score in list_search]
249                 data_fltr = data_arr[np.isin(data_arr[col], list_search)]
250         sorted_data = np.sort(data_fltr, order=col)
```

ภาพที่ 3-60 ค้นหาคอลัมน์ที่เป็นตัวเลข เช่น Average Score หรือ Salary

2) ถ้าค้นหาคอลัมน์ที่เป็นสตริง เช่น ID, Name, หรือ Department ฟังก์ชันจะใช้ np.isin() เพื่อกรองข้อมูลที่มีค่าในคอลัมน์เหล่านั้นตรงกับค่าใน list_search

```
251     elif col == 'ID' or col == 'Name':
252         data_fltr = data_arr[np.isin(data_arr[col], list_search)]
253         sorted_data = np.sort(data_fltr, order='ID')
254     elif col == 'Department':
255         data_fltr = data_arr[np.isin(data_arr[col], list_search)]
256         sorted_data = np.sort(data_fltr, order='Department')
257     elif col == 'Score':
258         data_fltr = data_arr[np.isin(data_arr['ID'], list_search)]
259         sorted_data = np.sort(data_fltr, order='ID')
```

ภาพที่ 3-61 ค้นหาคอลัมน์ที่เป็นสตริง เช่น ID, Name, หรือ Department

3.9.7 การเรียงลำดับข้อมูล

ข้อมูลที่ถูกกรองแล้วจะถูกจัดเรียงตามลำดับตามคอลัมน์ที่เกี่ยวข้อง:

1) ถ้าคอลัมน์ที่ค้นหาคือ ID หรือ Name ข้อมูลจะถูกจัดเรียงตาม ID ตามภาพที่ 3-61

2) ถ้าคอลัมน์ที่ค้นหาคือ Department ข้อมูลจะถูกจัดเรียงตาม Department ตามภาพที่ 3-61

3) ถ้าคอลัมน์ที่ค้นหาคือ Score ข้อมูลจะถูกจัดเรียงตาม ID ตามภาพที่ 3-61

4) ถ้าคอลัมน์ที่ค้นหาคือ Average Score หรือ Salary ข้อมูลจะถูกจัดเรียงตามคอลัมน์ที่ค้นหานั้น ตามภาพที่ 3-60

3.9.8 การแสดงผล

1) ฟังก์ชันจะนำข้อมูลที่ถูกจัดเรียงและกรองแล้วมาแปลงเป็น DataFrame ของ Pandas (pd.DataFrame(data=sorted_data)) เพื่อให้สามารถแสดงข้อมูลในรูปแบบตาราง

2) ถ้า DataFrame ว่างเปล่า (ไม่มีข้อมูลตรงตามที่ค้นหา) ฟังก์ชันจะแสดงข้อความว่า "No data found."

3) ถ้ามีข้อมูล ฟังก์ชันจะแสดงข้อมูลทั้งหมดในรูปแบบตารางโดยใช้คำสั่ง print(df_data_fltr.to_string(index=False)) ซึ่งจะพิมพ์ข้อมูลในรูปแบบที่ไม่มีตัวเลขลำดับ (index=False)

3.9.9 ผลลัพธ์ที่ได้

ฟังก์ชันนี้จะกรองและแสดงข้อมูลพนักงานเฉพาะที่ตรงกับเงื่อนไขที่ผู้ใช้กำหนด เช่น ค่าตัวเลขที่มากกว่าหรือน้อยกว่า หรือตรงกับค่าที่ค้นหาในคอลัมน์ที่ต้องการ

3.9.10 ตัวอย่างการเรียกใช้

ฟังก์ชันนี้สามารถเรียกใช้โดยกำหนดคอลัมน์และเงื่อนไขการค้นหา เช่น:

1) showSpecificData(1, ['12345']) เป็นการค้นหาข้อมูลจากคอลัมน์ ID

2) showSpecificData(4, [80], '1') เป็นการค้นหาข้อมูลที่มีคะแนนเฉลี่ยมากกว่า 80

บทที่ 4
อธิบายการทำงานของ code ส่วนการทำงาน

4.1 การรับคำสั่งจากผู้ใช้ (User Input Handling)

โค้ดที่ให้ไปเริ่มต้นด้วยการแสดงเมนูหลักให้ผู้ใช้เลือกการทำงานต่างๆ ซึ่งแต่ละเมนูจะสอดคล้องกับฟังก์ชันเฉพาะที่รับผิดชอบการทำงาน เช่น:

1) แสดงข้อมูลทั้งหมด (Show all data): ใช้สำหรับแสดงข้อมูลพนักงานทั้งหมด โดยสามารถเลือกรูปแบบการแสดงข้อมูลได้

2) แสดงข้อมูลเฉพาะเจาะจง (Show specific data): ใช้สำหรับค้นหาข้อมูลพนักงานตามเงื่อนไขที่ระบุ เช่น ค้นหาจาก ID, ชื่อ, แผนก หรือคะแนนเฉลี่ย

3) เพิ่มข้อมูล (Insert data): ใช้สำหรับเพิ่มข้อมูลพนักงานใหม่

4) แก้ไขข้อมูล (Edit data): ใช้สำหรับแก้ไขข้อมูลพนักงานที่มีอยู่แล้ว

5) ลบข้อมูล (Delete data): ใช้สำหรับลบข้อมูลพนักงานที่ระบุ

6) สร้างรายงาน (Export report): ใช้สำหรับสร้างรายงานสรุปข้อมูลพนักงาน

4.2 โครงสร้างการทำงานของแต่ละเมนู (Menu Functionality)

1) แสดงข้อมูลทั้งหมด (Show all data): โค้ดจะเรียกใช้ฟังก์ชัน showAllData ซึ่งจะรับพารามิเตอร์จากผู้ใช้เพื่อกำหนดรูปแบบการแสดงข้อมูลที่ต้องการ เช่น แสดงเฉลี่ยคะแนนหรือแสดงคะแนนล่าสุด

2) แสดงข้อมูลเฉพาะเจาะจง (Show specific data): ฟังก์ชัน showSpecificData จะรับพารามิเตอร์เพื่อทำการค้นหาข้อมูลเฉพาะตามที่ผู้ใช้ระบุ โค้ดจะเรียงลำดับข้อมูลที่ตรงกับเงื่อนไขก่อนแสดงผลออกมา

3) เพิ่มข้อมูล (Insert data): ฟังก์ชันนี้จะรับข้อมูลพนักงานใหม่จากผู้ใช้ เช่น ID, ชื่อ, แผนก, คะแนน, และเงินเดือน แล้วทำการเพิ่มข้อมูลใหม่ลงในไฟล์ข้อมูล

4) แก้ไขข้อมูล (Edit data): ผู้ใช้สามารถเลือกแก้ไขข้อมูลที่ต้องการ เช่น ชื่อ, แผนก, คะแนน หรือเงินเดือน โดยระบุ ID ของพนักงานที่ต้องการแก้ไข

5) ลบข้อมูล (Delete data): โค้ดจะรับ ID ของพนักงานที่ต้องการลบ แล้วทำการลบข้อมูลนั้นจากไฟล์

6) สร้างรายงาน (Export report): ฟังก์ชันนี้จะสร้างรายงานข้อมูลพนักงานในรูปแบบที่กำหนด

4.3 การจัดการข้อผิดพลาด (Error Handling)

ในแต่ละส่วนของโค้ดได้มีการจัดการกับข้อผิดพลาดที่อาจเกิดขึ้น โดยใช้ try-except เพื่อดักจับข้อผิดพลาดที่อาจเกิดขึ้นในระหว่างการทำงาน เช่น การป้อนข้อมูลผิดพลาด การไม่พบข้อมูล หรือการยกเลิกการทำงาน (เช่น การกด Ctrl+C)

4.4 โครงสร้างข้อมูลและการจัดเก็บ (Data Structure and Storage)

ข้อมูลของพนักงานถูกจัดเก็บในไฟล์ไบนารีและเรียกใช้ผ่านโมดูล BinFileOperation ซึ่งเป็นการทำงานที่เกี่ยวข้องกับการอ่านและเขียนข้อมูลในรูปแบบไบนารีเพื่อประสิทธิภาพในการจัดเก็บข้อมูล โดย ข้อมูลที่จัดเก็บจะประกอบด้วย ID, ชื่อ, แผนก, คะแนน และเงินเดือน โดยจะใช้โครงสร้างแบบ numpy และ pandas ในการจัดการข้อมูลเพื่อความสะดวกในการค้นหาและแสดงผล

4.5 การใช้โมดูลเสริม (External Libraries)

ในโค้ดนี้มีการใช้โมดูล numpy และ pandas เพื่อช่วยในการจัดการข้อมูลที่เป็นตัวเลขและข้อความได้อย่างมีประสิทธิภาพ โดย numpy ช่วยในการสร้างอาร์เรย์ข้อมูลและการประมวลผลเชิงตัวเลข ส่วน pandas ช่วยในการสร้างตารางข้อมูล (DataFrame) เพื่อแสดงผลในรูปแบบที่อ่านง่าย

4.6 สรุป (Conclusion)

รายงานสามารถสรุปได้ว่าโปรแกรมนี้เป็นเครื่องมือที่มีประสิทธิภาพในการจัดการข้อมูลพนักงาน โดยใช้การป้อนข้อมูลจากผู้ใช้เพื่อตอบสนองการทำงานต่างๆ ทั้งการเพิ่ม ลบ แก้ไข และค้นหาข้อมูล อีกทั้งยังสามารถสร้างรายงานได้อย่างสะดวก โปรแกรมนี้เหมาะสมกับการจัดการข้อมูลในองค์กรหรือหน่วยงานที่ต้องการการบันทึกข้อมูลพนักงานในลักษณะที่เป็นระบบ

```
7       print("1. Show all data")
8       print("2. Show specific data")
9       print("3. Insert data")
10      print("4. Edit data")
11      print("5. Delete data")
12      print("6. Export report")
13      print("7. Exit")
14      print("Go to the main menu or cancel by Ctrl+C")
```

ภาพที่ 4-1 โค้ดแสดงเมนูหลัก

```
18          case '1':   # Show all data
19              print("1. Show all data (average of scores)")
20              print("2. Show all data (latest 4 scores)")
21              choice = input("(Display All) Enter your display formatting options: ")
22              bfo.showAllData(choice)
```

ภาพที่ 4-2 โค้ดแสดงเมนูของการแสดงข้อมูลทั้งหมด

```
        case '2':   # Show specific data
            print("1. ID")
            print("2. Name")
            print("3. Department")
            print("4. Average Score")
            print("5. Score")
            print("6. Salary")
            print("Ctrl+C to cancel or go to the main menu")
            col = input("(Display Specific) What do you want to search by? Choose 1-6: ")
            column_choice = ['ID', 'Name', 'Department', 'Average Score', 'Score', 'Salary'][int(col) - 1]
            if col == '1' or col == '5':
                print("Please enter a single ID or multiple IDs to search.")
                print("Example. [A single ID] Enter ID: 0001")
                print("Example. [Multiple IDs] Enter ID: 0001 0002 0003")
            elif col == '2':
                print("Please enter a single name or multiple names to search.")
                print("Example. [A single name] Enter Name: John")
                print("Example. [Multiple names] Enter Name: John Peter")
            elif col == '3':
                print("Please enter a single department or multiple departments to search.")
                print("Example. [A single department] Enter Department: Engineering")
                print("Example. [Multiple departments] Enter Department: Engineering IT")
            if col in ['4', '6']:
                print("1. [>] More than")
                print("2. [>=] More than or equal to ")
                print("3. [<] Less than")
                print("4. [<=] Less than or equal to")
                print("5. [=] Equal to")
                print("Ctrl+C to cancel or go to the main menu")
                choice = input(f"(Display Specific) Please enter your choice (1-5): ")
                compare_choice = ['More than', 'More than or equal to', 'Less than', 'Less than or equal to', 'Equal to'][int(choice) - 1]
                match choice:
                    case '5':
                        print(f"Please enter a single {column_choice} or multiple {column_choice}s to search.")
                        print(f"Example. [A single {column_choice}] Enter Name: 80")
                        print(f"Example. [Multiple {column_choice}s] Enter Name: 10 20")
                        list_search = input(f"(Display Specific) Enter value: ").strip().split()
                    case _:
                        list_search = input(f"(Display Specific) Enter Value for finding {column_choice} that is {compare_choice}: ").strip().split()
                bfo.showSpecificData(col, list_search, choice)
            elif col == '5':
                list_search = input(f"(Display Specific) Enter ID: ").strip().split()
                bfo.showSpecificData(col, list_search)
            elif col in ['1', '2', '3']:
                list_search = input(f"(Display Specific) Enter {column_choice}: ").strip().split()
                bfo.showSpecificData(col, list_search)
            else:
                print("Invalid choice. Please try again.")
```

ภาพที่ 4-3 โค้ดแสดงเมนูของการแสดงผลแบบกรองข้อมูล

```
        case '3':   # Insert data
            number_of_data = int(input("(Insert) Enter the number of employees: "))
            for i in range(number_of_data):
                print(f"Input data {i + 1}/{number_of_data}")
                while True:
                    id = input("(Insert) Enter the ID: ")
                    list_records = bfo.readDataFromBinFile()
                    all_user_id = [user_data[0] for user_data in list_records]
                    if id in all_user_id:
                        print("ID already exists. Please enter a different ID.")
                    else:
                        break
                name = input("(Insert) Enter the Name: ")
                department = input("(Insert) Enter the Department: ")
                print(f"Please enter the score (max 4 value). \nExample, Enter the score: 80 90 100 95")
                score = input("(Insert) Enter the Score:")
                score = list(map(float, score.strip().split()))
                salary = float(input("(Insert) Enter the Salary: "))
                bfo.addData(id.strip(), name.strip(), department.strip(), score, salary)
```

ภาพที่ 4-4 โค้ดแสดงเมนูของการเพิ่มข้อมูล

```
90              case '4':   # Edit data
91                  print("1. Name")
92                  print("2. Department")
93                  print("3. Score")
94                  print("4. Salary")
95                  print("Ctrl+C to cancel or go to the main menu")
96                  col = input("(Edit) Select the option you want to edit: ")
97                  column_choice = ['Name', 'Department', 'Score', 'Salary'][int(col) - 1]
98                  print("Please enter a single ID or multiple IDs to edit.")
99                  print("Example. [A single ID] Enter ID: 0001")
100                 print("Example. [Multiple IDs] Enter ID: 0001 0002 0003")
101                 multi_id = input("(Edit) Enter ID: ")
102                 multi_id = tuple(multi_id.strip().split())
103                 if len(multi_id) == 0:
104                     print("ID not found can't edit data.")
105                     input("Press Enter to continue...")
106                     continue
```

ภาพที่ 4-5 รับคำสั่งจากผู้ใช้สำหรับการแก้ไขข้อมูลพนักงานตามคอลัมน์ที่เลือกและตาม ID ที่ป้อน

```
108                 case 'Name':
109                     print(f"Please enter new name to edit. (Must be equal to the number of IDs)")
110                     print("Example. Enter the new name for ID('001', '002'): John Peter")
111                     new_name = input(f"(Edit) Enter the new name for ID{multi_id}: ")
112                     new_name = tuple(new_name.strip().split())
113                     bfo.editData(column_choice,multi_id,new_name)
```

ภาพที่ 4-6 การรับชื่อใหม่ที่ผู้ใช้ป้อนสำหรับพนักงานตาม ID ที่ต้องการแก้ไข

```
114                 case 'Department':
115                     print(f"Please enter new department to edit. (Must be equal to the number of IDs)")
116                     print("Example. Enter the new department for ID('001', '002'): John Peter")
117                     new_department = input(f"(Edit) Enter the new department for ID{multi_id}: ")
118                     new_department = tuple(new_department.strip().split())
119                     bfo.editData(column_choice,multi_id,new_department)
```

ภาพที่ 4-7 การแก้ไขข้อมูลแผนกของพนักงานตาม ID ที่ผู้ใช้เลือก

```
120             case 'Score':
121                 print("1. Add new score")
122                 print("2. Edit existing score")
123                 choice_edit_score = input("(Edit) Enter your choice: ")
124                 all_user_data = bfo.readDataFromBinFile()
125                 all_user_data = [(user_data[0],user_data[1] , f"[{', '.join(list(map(str, user_data[3])))}]") for user_data in all_user_data]
126                 dtype = [('ID', 'U20'),
127                          ('Name', 'U20'),
128                          ('Score', 'U30')]
129                 all_user_data_fltr = asarray(all_user_data, dtype=dtype)
130                 data_fltr = all_user_data_fltr[isin(all_user_data_fltr['ID'], multi_id)]
131                 sorted_data = sort(data_fltr, order='ID')
132                 if len(sorted_data) == 0:
133                     raise Exception("ID not found can't edit score.")
134                 df_data_fltr = DataFrame(data=sorted_data)
135                 print(df_data_fltr.to_string(index=False))
136                 multi_new_score = []
137                 match choice_edit_score:
138                     case '1':…
149                     case '2':…
177                     case _:
178                         print("Invalid choice. Please try again.")
```

ภาพที่ 4-8 การแก้ไขคะแนน (Score) ของผู้ใช้

```
                case '1':
                    for id in multi_id:
                        print(f"Please enter new score to edit. (Max 4 value)")
                        print(f"Example. Enter the new score: 80 90 100 95")
                        new_score = input(f"(Edit>Add new>ID{id}) Enter the new score: ")
                        new_score = list(new_score.strip().split())
                        if len(new_score) == 0:
                            print(f"Score not found can't edit score for {id}.")
                            continue
                        multi_new_score.append(new_score)
                    bfo.editData(column_choice,multi_id,multi_new_score,choice_edit_score)
```

ภาพที่ 4-9 การทำงานย่อยที่ 1 ของการแก้ไขคะแนน (score)

```
                case '2':
                    multi_index = []
                    all_user_data = bfo.readDataFromBinFile()
                    all_user_id = [user_data[0] for user_data in all_user_data]
                    for id in multi_id:
                        if id not in all_user_id:
                            print(f"ID {id} not found can't edit score for {id}.")
                            continue
                        print(f"Please enter the index of score to edit. (0-3)")
                        print(f"Example. Enter the index: 0 1 2 3")
                        index = input(f"(Edit>Edit existing>ID{id}) Enter the index: ")
                        index = tuple(index.strip().split())
                        if len(index) == 0:
                            print(f"Index not found can't edit score for {id}.")
                            continue
                        multi_index.append(index)
                        print(f"Please enter new score to edit. (According to position of idex)")
                        print(f"Example. Enter the new score for index ('0', '2'): 80 90")
                        new_score = input(f"(Edit>Edit existing>ID{id}) Enter the new score for index {index}: ")
                        new_score = tuple(new_score.strip().split())
                        if len(new_score) == 0:
                            print(f"Score not found can't edit score for {id}.")
                            continue
                        elif len(new_score) != len(index):
                            print(f"Score and index length not same can't edit score for {id}.")
                            continue
                        multi_new_score.append(new_score)
                    bfo.editData(column_choice,multi_id,multi_new_score,choice_edit_score, multi_index)
```

ภาพที่ 4-10 การทำงานย่อยที่ 2 ของการแก้ไขคะแนน (score)

```
                case 'Salary':
                    print("1. Add new salary")
                    print("2. Cut salary")
                    print("3. Edit existing salary")
                    choice_edit_salary = input("(Edit) Enter your choice: ")
                    match choice_edit_salary:
                        case '1':
                            print(f"Please enter value for add salary of ID {multi_id}.")
                            print("Example. Enter a single number for add salary: 1000")
                            try:
                                new_salary = float(input("(Edit) Enter a single number for add salary: "))
                            except ValueError:
                                print("Input 1 number only.")
                            else:
                                bfo.editData(column_choice,multi_id, new_salary, choice_edit_salary)
                        case '2':
                            print(f"Please enter value for cut salary of ID {multi_id}.")
                            print("Example. Enter a single number for cut salary: 1000")
                            try:
                                new_salary = float(input("(Edit) Enter a single number for cut salary: "))
                            except ValueError:
                                print("Input 1 number only.")
                            else:
                                bfo.editData(column_choice,multi_id, new_salary, choice_edit_salary)
                        case '3':
                            print(f"Please enter new salary for {multi_id}.")
                            print(f"Example. Enter a single number for new salary: 10000")
                            new_salary = input("(Edit) Enter a single number for new salary:")
                            new_salary = tuple(map(float, new_salary.strip().split()))
                            if len(new_salary) == 0:
                                print(f"Salary not found can't edit salary for {multi_id}.")
                            elif len(new_salary) != len(multi_id):
                                print(f"Salary and ID length not same can't edit salary for {multi_id}.")
                            else:
                                bfo.editData(column_choice,multi_id, new_salary, choice_edit_salary)
                        case _:
                            raise Exception("Invalid choice. Please try again.")
```

ภาพที่ 4-11 การเพิ่ม, ตัด, และแก้ไขเงินเดือนและแสดงผลคำแนะนำเพื่อให้ผู้ใช้ป้อนข้อมูล

```
218             case '5':   # Delete data
219                 print("Please enter a single ID or multiple IDs to delete.")
220                 print("Example. [A single ID] Enter ID: 0001")
221                 print("Example. [Multiple IDs] Enter ID: 0001 0002 0003")
222                 list_search = input(f"(Delete) Enter ID:").strip().split()
223                 bfo.deleteData(list_search)
```

ภาพที่ 4-12 การรับค่า ID (หรือหลาย ID) จากผู้ใช้เพื่อลบข้อมูลที่มี ID เหล่านั้น

```
224             case '6':   # Export report
225                 bfo.exportReport()
226             case '7':   # Exit
227                 if input("Do you want to exit? (y/[n]): ").lower() == 'y':
228                     break
```

ภาพที่ 4-13 การทำรายงานและการแสดงข้อความยืนยันออกจากโปรแกรม

```
231     except Exception as e:
232         print(f"An error occurred: {e}")
233         input("Press Enter to continue...")
234     except KeyboardInterrupt:
235         print("Operation cancelled.")
236     else:
237         print("================ (End of operation) ================")
238         input("Press Enter to continue...")
239         print()
```

ภาพที่ 4-14 การจัดการข้อผิดพลาดที่อาจเกิดขึ้นในโปรแกรม