Programowanie Strukturalne

-

Laboratorium 05 - Python lab02

Adam Szajgin s319821 gr. 109

Politechnika Warszawska

12 grudnia 2023

Spis treści

1.1 Temat programu

Poniższy program to proste narzędzie demonstracyjne umożliwiające użytkownikowi wykonywanie różnorodnych działań na niewielkiej bazie danych, program jest mała symulacją elektronicznego dziennika elektronicznego. Stworzone z myślą o prezentacji różnych funkcji, program oferuje możliwość wyboru operacji na zainicjalizowanej bazie danych.

1.2 Opis programu

Program został zaprojektowany w charakterze bazodanowym przy wykorzystaniu bilbioteki sqlite3. Program ten, pozwala użytkownikowi, poprzez komunikację w terminalu, na dokonywanie różnych operacji na zainicjalizowanej małej bazie danych. Poprzez spredefiniowane polecenia w języku SQL, użytkownik wykonuje je z wybranymi parametrami. Program wykorzystuje listy strukturalne jak i różne funkcje, w tym dekomponujących zadania do innych funkcji.

Oto pomysł przyświecający programowi:
"Program stworzony do zarządzania bazą danych studentów. Każdy student jest identyfikowany poprzez imię, nazwisko, wiek, średnią ocen, kierunek studiów oraz unikalne ID. Dzięki temu programowi użytkownik może przeglądać różne informacje dotyczące studentów: wyświetlać listę studentów, którzy spełniają określone warunki, uzyskiwać konkretne informacje o danym studencie oraz wyszukiwać studentów posiadających takie same dane, na przykład imię. Dodatkowo, użytkownik ma możliwość modyfikowania, usuwania i dodawania nowych studentów do bazy danych. Program oferuje funkcje dekompozycji, na przykład możliwość porównania studentów na podstawie określonych kryteriów, gdzie funkcja usunięcia studenta przyjmuje odpowiednie dane i porównuje je, np. średnią ocen, imiona."

Ten program został utworzony w celu demonstracyjnym, dlatego też wykorzystałem przykładowe dane, aby pokazać działanie różnych funkcji. Nie był optymalizowany pod kątem uniwersalności czy efektywności.

1.3 Prezentacja działania programu

UWAGA: Prezentację działania programu podzieliłem na dwie części; formalną i praktyczną. W formalnej opisuję jak wygląda wejście do programu, jak inicjowałem bazę danych, itp. W skrócie; mało znaczące z punktu widzenia laboaratorium rzeczy, a dużo znaczące z punktu widzenia ogólno pojętej "dokumentacji". Część praktyczna - odnosi się do wymagań laboratorium.

I CZĘŚĆ FORMALNA

Po uruchomieniu programu, w back-endzie otwierana jest baza danych i wypełniana zestawem 10 przykładowych rekordów:

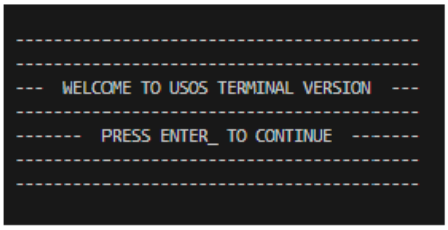
```
create_table()

input_student("Aleksandra", "Pilsner", 21, 123456, "Electrical Engineering", 3.75)
input_student("Bartosz", "Stout", 19, 234567, "Computer Science", 4.00)
input_student("Celina", "Porter", 20, 345678, "Business Administration", 3.90)
input_student("Dawid", "Hefeweizen", 22, 456789, "Mechanical Engineering", 3.60)
input_student("Eliza", "Amber", 18, 567890, "Biochemistry", 4.20)
input_student("Feliks", "Bock", 21, 678901, "Graphic Design", 3.80)
input_student("Gabriela", "Lager", 20, 789012, "Environmental Science", 3.95)
input_student("Henryk", "Ale", 19, 890123, "Psychology", 3.85)
input_student("Izabela", "Saison", 22, 901234, "Civil Engineering", 3.70)
input_student("Jacek", "Porter", 18, 102345, "English Literature", 4.10)
```

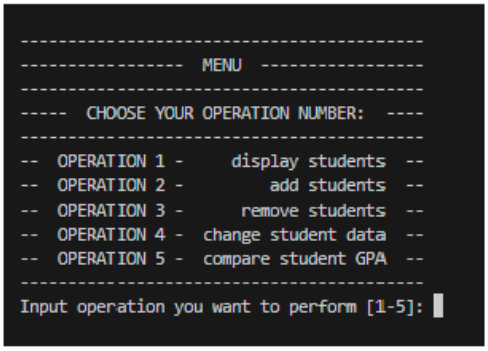
Rys. 1. Inicjalizacja bazy danych i uzupełnienie jej przykładowymi danymi

```
def create_table():
    cursor.execute(
        "CREATE TABLE students(name TEXT, surname TEXT, age INTEGER, id INTEGER, major TEXT, gpa DOUBLE)"
    )


def input_student(name, surname, age, id, major, gpa):
    cursor.execute(
        " INSERT INTO students (name, surname, age, id, major, gpa) values (?, ?, ?, ?, ?, ?)",
        (name, surname, age, id, major, gpa),
    )
```

Rys. 2. Funkcje odpowiadające za wykonane operacje

Natomiast na front-endzie aplikacji użytkownik napotyka ekran "WELCOME" a następnie ekran "MENU" wyboru operacji na bazie:



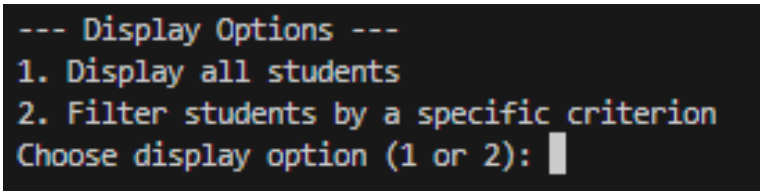


Rys. 3. Wiadomosc "welcome" oraz "Menu" programu

I CZĘŚĆ PRAKTYCZNA:

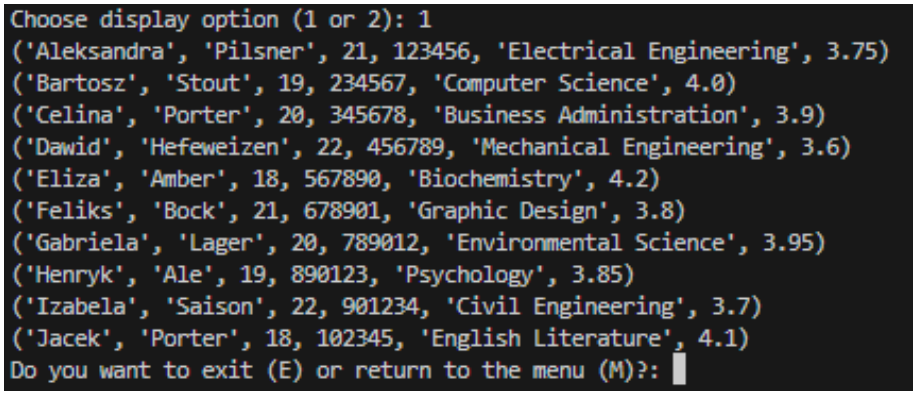
WAŻNE: Komunikacja z użytkownikiem realizowana jest poprzez CLI i wpisywanie odpowiednich komend. Pierwsze 3 operacje są przykładami komunikacji użytkownika z bazą danych.
Każda operacja jest przykładem działania funkcji przetwarzania danych.
Natomiast ostatnia - 5 operacja jest przykładem funkcji dekomponującej zadania, na pomniejsze funkcje. Również niektóre funkcje bezpośrednio wykorzystują listy strukturalne w swoim działaniu, jednak na dobrą sprawę każda je wykorzystuje - wystarczy spojrzeć na sposób zwracania danych z bazy danych - dane układane są w swoiste listy strukturalne.

1. Pierwsza operacja - "display students"
   Po wybraniu tej operacji użytkownik napotyka wybór:



Rys. 4. Symulacja wyboru opcji: "display students"

I wybór odpowiednio 1 (wyświetlenie wszystkich rekordów z bazy danych) i 2 (wyświelenie rekordów po filtracji)opcji:



Rys. 5. wyświetlenie wszystkich rekordów z bazy danych

```
Choose display option (1 or 2): 2
Provide the column name by which you'd like to filter students: age
Provide the value you want the 'age' to be equal to: 20
('Celina', 'Porter', 20, 345678, 'Business Administration', 3.9)
('Gabriela', 'Lager', 20, 789012, 'Environmental Science', 3.95)
Do you want to exit (E) or return to the menu (M)?:
```

Rys. 6. wyświelenie rekordów po filtracji

Oraz kod odpowiedzialny za powyższe operacje:

```python
def display_student(display_option=None, disp_where=None, disp_equal=None):
    if display_option == "filter":
        query = f"SELECT * FROM students WHERE {disp_where} = ?"
        cursor.execute(query, (disp_equal,))
        output = cursor.fetchall()
        return output
    elif display_option == "all":
        cursor.execute("SELECT * FROM students")
        output = cursor.fetchall()
        return output
    else:
        print("Invalid display option. Please choose 'filter' or 'all'.")
```

Rys. 7. funkcja wywoływana, modelująca zapytanie do bazy danych

```python
match op_choice:
    # First case, displaying the students already in the database
    case 1:
        print("--- Display Options ---")
        print("1. Display all students")
        print("2. Filter students by a specific criterion")

        display_choice = input("Choose display option (1 or 2): ")

        if display_choice == "1":
            result = display_student("all")
            if not result:
                print("No students found.")
            else:
                for student in result:
                    print(
                        student
                    )
        elif display_choice == "2":
            input_where = input(
                "Provide the column name by which you'd like to filter students: "
            )
            input_what = input(
                f"Provide the value you want the '{input_where}' to be equal to: "
            )
            result = display_student("filter", input_where, input_what)
            if not result:
                print("No students found with the provided criteria.")
            else:
                for student in result:
                    print(
                        student
                    )
        else:
            print("Invalid choice.")

        if end_option():
            continue
```

Rys. 8. Wykonanie operacji 1 w switch case

II CZĘŚĆ FORMALNA:

2. Druga operacja - ”add students”
   Po wybraniu tej operacji użytkownik napotyka wiele wierszy wprowadzenia argumentów dla wszystkich kolumn:

```
--- You are now adding students into the database ---
Provide the student's name: Student
Provide the student's surname: Testowski
Provide the student's age: 22
Provide the student's ID: 021370
Provide the student's major: Gender studies (?)
Provide the student's GPA: 1.2
Do you want to exit (E) or return to the menu (M)?:
```

Rys. 9. Symulacja wyboru opcji: ”add students”

```
--- Display Options ---
1. Display all students
2. Filter students by a specific criterion
Choose display option (1 or 2): 1
('Aleksandra', 'Pilsner', 21, 123456, 'Electrical Engineering', 3.75)
('Bartosz', 'Stout', 19, 234567, 'Computer Science', 4.0)
('Celina', 'Porter', 20, 345678, 'Business Administration', 3.9)
('Dawid', 'Hefeweizen', 22, 456789, 'Mechanical Engineering', 3.6)
('Eliza', 'Amber', 18, 567890, 'Biochemistry', 4.2)
('Feliks', 'Bock', 21, 678901, 'Graphic Design', 3.8)
('Gabriela', 'Lager', 20, 789012, 'Environmental Science', 3.95)
('Henryk', 'Ale', 19, 890123, 'Psychology', 3.85)
('Izabela', 'Saison', 22, 901234, 'Civil Engineering', 3.7)
('Jacek', 'Porter', 18, 102345, 'English Literature', 4.1)
('Student', 'Testowski', 22, 21370, 'Gender studies (?)', 1.2)
Do you want to exit (E) or return to the menu (M)?:
```

Rys. 10. wyświetlenie zaktualizowanej bazy studentów - powiększonej o nowo-dodany rekord (ostatni)

Oraz kod odpowiedzialny za powyższe operacje:

```python
# Second case, inputting a student of chosen attributes
case 2:
    print("--- You are now adding students into the database ---")

    name = input("Provide the student's name: ")
    surname = input("Provide the student's surname: ")
    age = int(input("Provide the student's age: "))
    id = int(input("Provide the student's ID: "))
    major = input("Provide the student's major: ")
    gpa = float(input("Provide the student's GPA: "))
    input_student(name, surname, age, id, major, gpa)

    if end_option():
        continue
```

Rys. 11. Wykonanie operacji 2 w switch case

Natomiast funkcja odpowiedzialna za dodawanie rekordów została uwieczniona już na Rys.4.

3. Trzecia operacja - ”remove students”

```
--- You are now removing students from the database ---
Provide the column name by which you'd like to remove students: name
Provide the value you want the 'name' to be equal to for removal: Student
Do you want to exit (E) or return to the menu (M)?:
```

Rys. 12. Symulacja wyboru opcji: ”remove students”

```
--- Display Options ---
1. Display all students
2. Filter students by a specific criterion
Choose display option (1 or 2): 1
('Aleksandra', 'Pilsner', 21, 123456, 'Electrical Engineering', 3.75)
('Bartosz', 'Stout', 19, 234567, 'Computer Science', 4.0)
('Celina', 'Porter', 20, 345678, 'Business Administration', 3.9)
('Dawid', 'Hefeweizen', 22, 456789, 'Mechanical Engineering', 3.6)
('Eliza', 'Amber', 18, 567890, 'Biochemistry', 4.2)
('Feliks', 'Bock', 21, 678901, 'Graphic Design', 3.8)
('Gabriela', 'Lager', 20, 789012, 'Environmental Science', 3.95)
('Henryk', 'Ale', 19, 890123, 'Psychology', 3.85)
('Izabela', 'Saison', 22, 901234, 'Civil Engineering', 3.7)
('Jacek', 'Porter', 18, 102345, 'English Literature', 4.1)
Do you want to exit (E) or return to the menu (M)?:
```

Rys. 13. Stan bazy danych po usunięciu rekordu dodanego w poprzedniej sytiacji

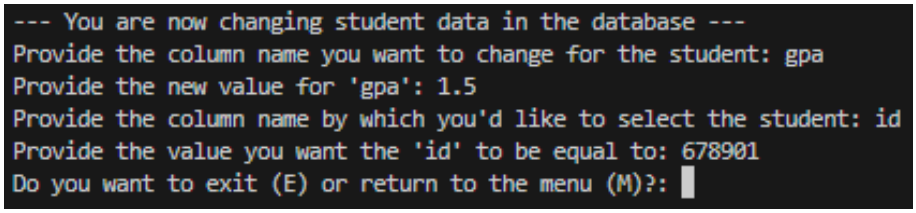
```python
def remove_student(rem_where, rem_equal):
    query = f"DELETE FROM students WHERE {rem_where} = ?"
    cursor.execute(query, (rem_equal,))
```

Rys. 14. funkcja wywoływana, modelująca zapytanie do bazy danych

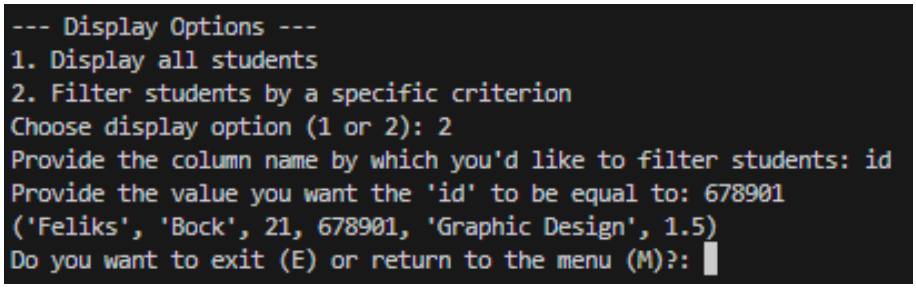
```python
# Third case, removing a student based on given criteria
case 3:
    print("--- You are now removing students from the database ---")

    rem_where = input(
        "Provide the column name by which you'd like to remove students: "
    )
    rem_equal = input(
        f"Provide the value you want the '{rem_where}' to be equal to for removal: "
    )

    remove_student(rem_where, rem_equal)

    if end_option():
        continue
```

Rys. 15. Wykonanie operacji 3 w switch case

4. Czwarta operacja - ”change student data”



Rys. 16. Symulacja wyboru opcji: "change student data"



Rys. 17. Stan bazy danych po zmiany rekordu

```python
# Fourth case, changing student data based on given criteria
case 4:
    print("--- You are now changing student data in the database ---")

    what_change = input(
        "Provide the column name you want to change for the student: "
    )
    new_value = input(f"Provide the new value for '{what_change}': ")
    where_change = input(
        "Provide the column name by which you'd like to select the student: "
    )
    old_value = input(
        f"Provide the value you want the '{where_change}' to be equal to: "
    )

    change_student(what_change, new_value, where_change, old_value)

    if end_option():
        continue
```

Rys. 18. funkcja wywoływana, modelująca zapytanie do bazy danych

```python
def change_student(what_change, new_value, where_change, old_value):
    query = f"UPDATE students SET {what_change} = ? WHERE {where_change} = ?"
    cursor.execute(query, (new_value, old_value))
```

Rys. 19. Wykonanie operacji 4 w switch case

II CZĘŚĆ PRAKTYCZNA:

5. Piąta operacja - ”compare student GPA”

```
--- You are now comparing student GPAs ---
Provide ID of first student: 345678
Provide ID of second student: 456789
[('Celina', 'Porter', 20, 345678, 'Business Administration', 3.9), ('Dawid', 'Hefeweizen', 22, 456789, 'Mechanical Engineering', 3.6)]
The student with a greater GPA is with id = 345678
Do you want to exit (E) or return to the menu (M)?:
```

Rys. 20. Symulacja wyboru opcji: ”compare student GPA”

W tym przypadku ciekawym jest sposób w jaki owa funkcja działa. Dekomponuje ona zadania pomiędzy siebie i inną funkcję. Pierwsza funkcja ekstraktuje dane z zapytania i oddelegowuje je do drugiej funkcji wewnętrznej, która je porównuje! Również wykorzystuje listy strukturalne.

```python
def comparing_students(stud_A, stud_B):
    def comparing(a, a_id, b, b_id):
        if a > b:
            return a_id
        elif a < b:
            return b_id
        else:
            return 0

    query = f"SELECT * FROM students WHERE id IN (?, ?)"
    cursor.execute(query, (stud_A, stud_B))
    output = cursor.fetchall()

    if len(output) != 2:
        print("Invalid student IDs provided.")
        return

    a = output[0][-1]
    b = output[1][-1]
    print(output)

    greater_id = comparing(a, stud_A, b, stud_B)
    if greater_id == 0:
        print("Both students have the same GPA, and it's equal to:", a)
    else:
        print("The student with a greater GPA is with id =", greater_id)
```

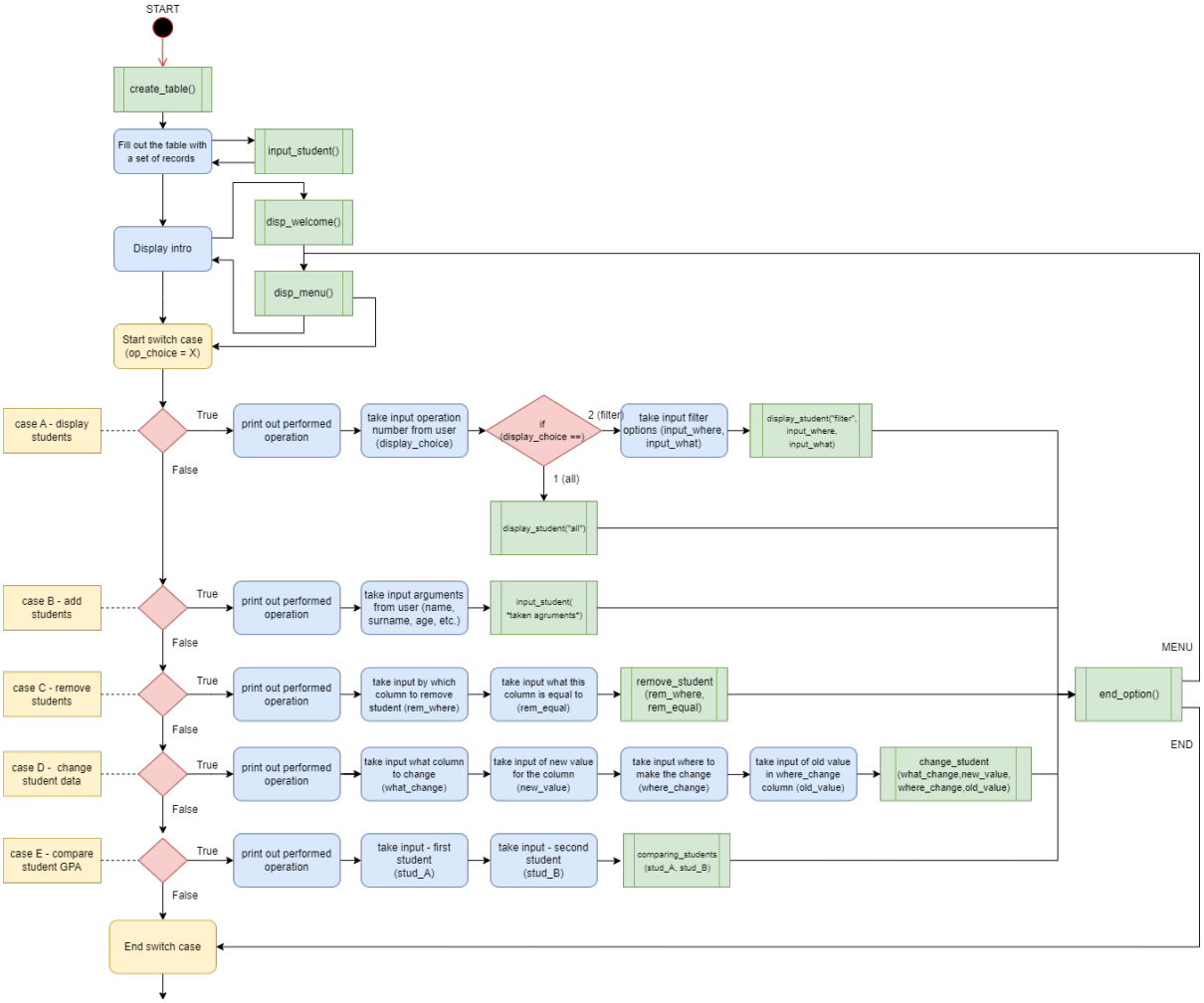
Rys. 21. funkcja wywoływana, dekomponująca zadania pomiędzy siebie i inną funkcję wewnętrzną

```
# Fifth case, comparing GPAs between two students
case 5:
    print("--- You are now comparing student GPAs ---")

    stud_A = int(input("Provide ID of first student: "))
    stud_B = int(input("Provide ID of second student: "))
    comparing_students(stud_A, stud_B)

    if end_option():
        continue
```

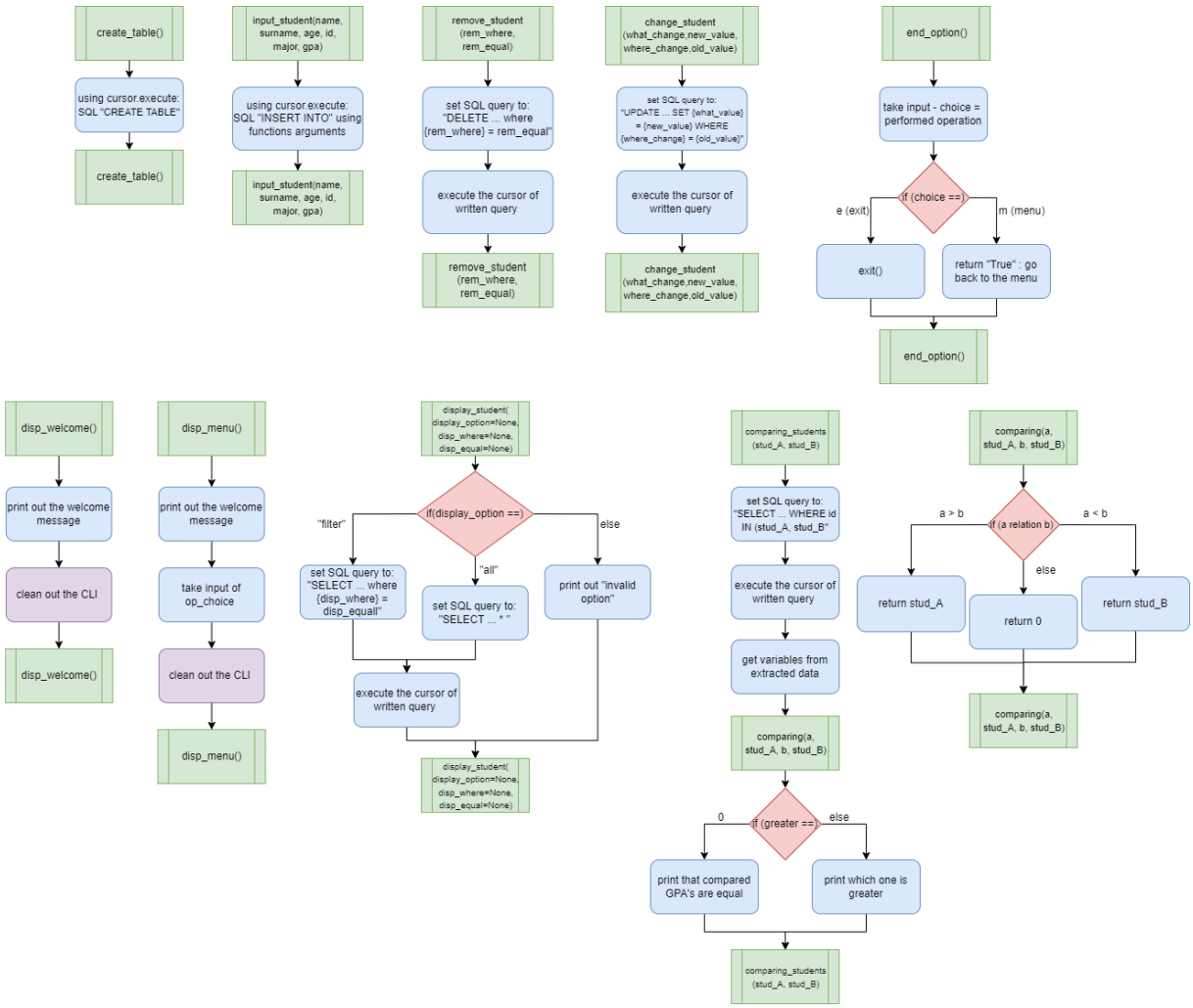
Rys. 22. Wykonanie operacji 5 w switch case

1.3 Algorytm działania w standardzie UML

Poniżej prezentuje diagram w standardzie UML programu. Pierwszy diagram prezentuje schemat działania strukturalnego programu, natomiast drugi prezentuje działanie każdej z użytych funkcji. Do wykonania diagramu został użyty program draw.io.



Rys. 23. diagram programu w standardzie UML



Rys. 24. diagramy funkcji używanych w programie w standardzie UML