Семетровий проєкт з C/C++

University

Студентки механіко-математичного факультету
Київського національного університету імені Тараса Шевченка
Соловій Катерини
2 курс, комп'ютерна математика, 1 група

2024

Зміст

1 Вступ

Задача проєкту полягає в тому, щоб створити базу даних для внесення інформації про студентів та працівників університету на C, використовуючи структури, на C++, використовуючи класи та застосувати інфтерфейс. Це включає внесення імені, прізвища студента, а також освітній ступінь, спеціальність, рік навчання, середній бал та імена і прізвище викладачів, які викладають у даного студента.

Для працівника база даних має містити такі відомості: ім'я, прізвище, посада, базова зарплата, стаж.

Програма повинна обчислювати зарплату працівників, враховуючи всі характеристики, зазначені вище. Також необхідно підрахувати середній бал студентів, які навчаються у відповідного викладача, та визначити студентів, які отримують звичайну стипендію, підвищену стипендію або не отримують стипендію взагалі. Уся ця інформація зберігається у форматі JSON.

2 Реалізація на C

2.1 Структура проєкту

Нижче наведена структура проєкту:

```
project_c
 cJSON
 headers
    university.h
    input_validation.h
 src
    university.c
    input_validation.c
 test
 main.c
```

cJSON завантажено з GitHub з офіційного репозиторія. Ось посилання на репозиторій cJSON на GitHub.

2.2 Функції

У цьому розділі будуть описані функції, реалізовані в проєкті.

Функції в university.h:

- `int get_number_of_students();`
- `void input_student_data(Student* students, int num_students);`
- `void sorted_students(Student* students, int num_students, Record_book* recordBook, char specialties[][200]);`
- `void calculate_scholarships(Student* students, int num_students, Record_book* recordBook, char specialties[][200]);`
- `void generate_filename(const char* ind, const char* form, char* filename);`
- `void save_students_to_json(Student* students, int num_students);`
- `char* read_file(const char* filename);`
- `cJSON* load_json_from_file(const char* filename, const char* key);`
- `void display_students_from_json(cJSON* students);`
- `void calculate_salary(Employee* employee);`
- `int get_number_of_employees();`
- `void input_data(Employee* employee);`
- `void input_employees_data(Employee* employees, int num_of_employees);`
- `void save_employees_to_json(Employee* employees, int num_of_employees, const char* filename);`
- `void save_lecturers_to_json(Employee* employees, int num_of_employees, const char* filename);`
- `float calculate_average_grade_for_lecturer(const cJSON *students_json, const char *lecturer_first_name, const char *lecturer_last_name);`
- `void update_lecturers_file(const char *filename, cJSON *lecturers_json);`
- `void update_lecturers(const char *students_file, const char *lecturers_file);`
- `void display_lecturers_from_json(cJSON *lecturers);`
- `void display_employees_from_json(cJSON *employees);`

- `int find_specialty_index(char specialties[][200], int num_specialties, const char* specialty);`
- `int is_valid_string(const char* str);`
- `void get_valid_string(const char* m, char* str, size_t max_length);`
- `int is_valid_degree(const char* str);`
- `void get_valid_degree(const char* m, char* deg, size_t max_length);`
- `double get_valid_double(const char* num);`
- `int get_valid_int(const char* m);`
- `int is_profession(const char* profession);`
- `void get_valid_profession(const char* m, char* prof, size_t size);`
- `int file_exists(const char *filename);`

Функції в input validation.h:

- `int find_specialty_index(char specialties[][200], int num_specialties, const char* specialty);`
- `int is_valid_string(const char* str);`
- `void get_valid_string(const char* m, char* str, size_t max_length);`
- `int is_valid_degree(const char* str);`
- `void get_valid_degree(const char* m, char* deg, size_t max_length);`
- `double get_valid_double(const char* num);`
- `int get_valid_int(const char* m);`
- `int is_profession(const char* profession);`
- `void get_valid_profession(const char* m, char* prof, size_t size);`
- `int file_exists(const char* filename);`

У документації опис усіх нище наведних функцій описано.

2.3 UML-даграма

У цьому розділі представлено діаграму UML, що представляє структуру та зв'язки функцій у проєкті.

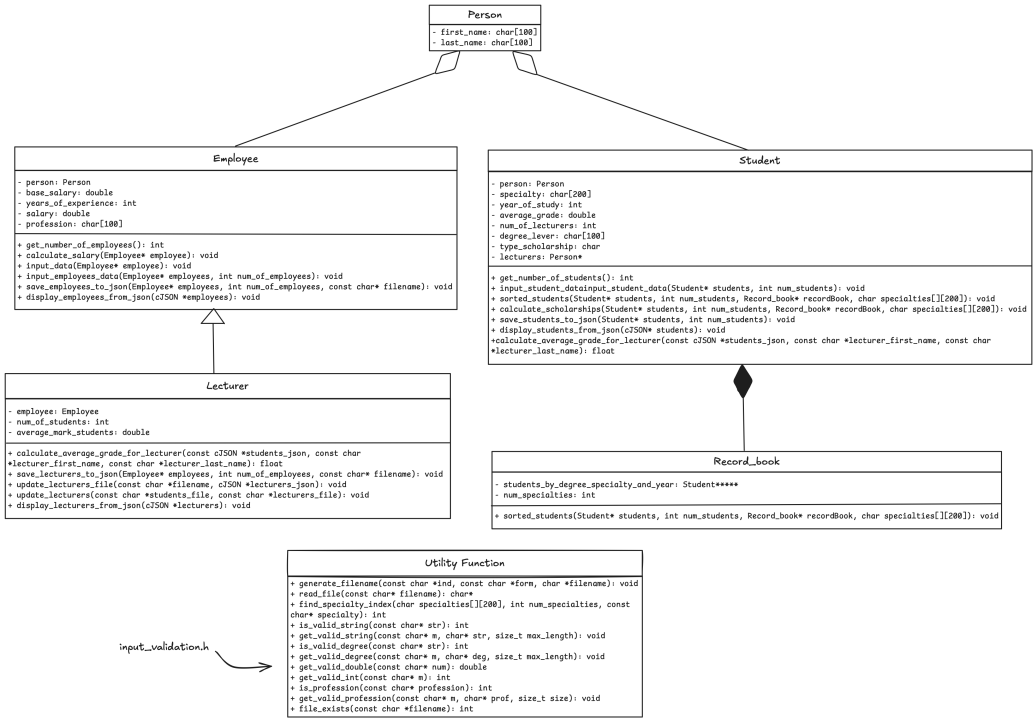

Рис. 1: UML Diagram

Структура `Employee` включає `Person` як складову частину (поле `person`). Це агрегація, оскільки `Person` є частиною `Employee`.

`Lecturer` будується на основі `Employee`, тобто всі поля `Employee` стають частиною `Lecturer` — наслідування.

Структура `Student` також має поле `Person` як частину даних про студента (`person`). Це також агрегація.

`Record_book` зберігає ієрархію студентів через багатовимірний масив вказівників на `Student`. Це композиція, оскільки студенти логічно є частиною записної книжки.

2.4 Текстовий інтерфейс - ncurses

Ncurses (new curses) — бібліотека, що дозволяє створювати текстовий інтерфейс користувача для керуванням вводом-виводом на термінал в режимі консольного застосування.

2.4.1 У цьому проєкті використовуються такі основні функції:

- `initscr()` — ініціалізація бібліотеки.
- `raw()` — режим raw input, без відображення натисканих клавіш.
- `keypad(stdscr, TRUE)` — активація спеціальних клавіш.
- `noecho()` — заборона на відображення введених символів.
- `mvprintw(row, col, string)` — виведення тексту в заданій позиції.
- `refresh()` — оновлення екрану після виведення.
- `getch()` — очікування натискання клавіші.
- `endwin()` — завершення роботи з ncurses.

Ці функції дозволяють ефективно відображати таблиці та взаємодіяти з користувачем через консоль.

2.4.2 Приклад таблиці

2.5 Застосування масивів професій у проєкті

У рамках реалізації проєкту, що спрямований на створення бази даних для обробки інформації про студентів та працівників університету, необхідно враховувати різні професії працівників. Для цього були визначені масиви, що містять різні категорії професій. Кожен масив є набором рядків, що описують конкретну категорію працівників, які можуть бути присутні в університеті. Це дозволяє зручно організовувати і працювати з даними, що стосуються посад та ролей співробітників.

Ось приклади таких масивів:

2.5.1 Масив адміністративного управління

Масив `ADMINISTRATIVE_MANAGEMENT` містить професії, що відносяться до адміністративного управління університету. Це посадові категорії, що відповідають за управлінські функції та організацію навчального процесу на високому рівні:

Цей масив включає наступні посади:

- **Dean** – Ректор або декан факультету, який має найвищу адміністративну роль у навчальному закладі.

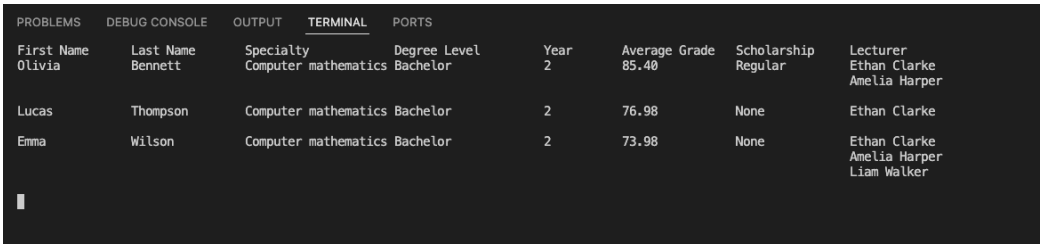

Рис. 2: результат застосування ncurses: students

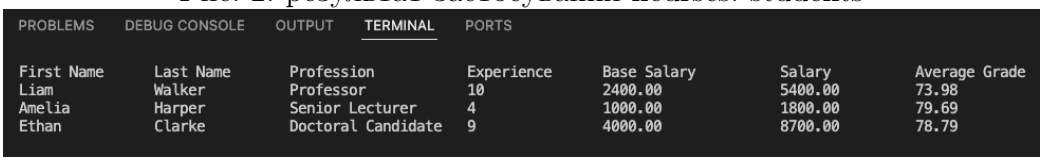

Рис. 3: результат застосування ncurses: lecturers

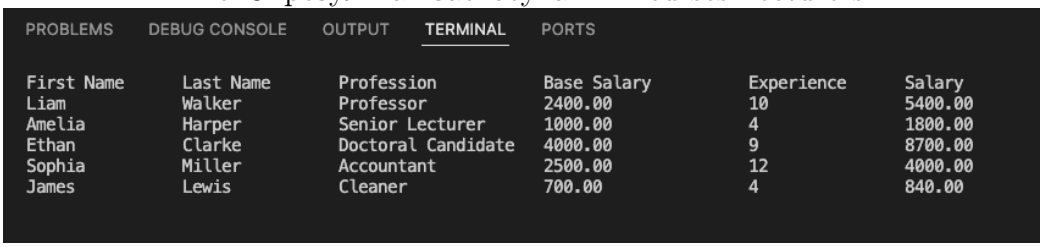

Рис. 4: результат застосування ncurses: employees

- **Deputy Dean** – Заступник декана, відповідальний за організаційні питання факультету.
- **Head of Department** – Керівник кафедри, що займається організацією навчального процесу на кафедрі та управлінням викладацьким складом.

2.5.2 Масив науково-педагогічного складу

Масив `SCIENTIFIC_PEDAGOGICAL` включає професії, що відносяться до науково-педагогічного складу університету. Ці професії стосуються викладачів та науковців:

Цей масив містить наступні професії:

- **Professor** – Професор, вищий рівень академічного складу, який має досвід у науковій та педагогічній діяльності.
- **Associate Professor** – Доцент, науковець та викладач, що має певний досвід в академічній сфері.

- **Senior Lecturer** – Старший викладач, який має великий досвід у викладанні та науковій роботі.
- **Assistant** – Асистент, помічник викладача, який може допомагати у проведенні лекцій, семінарів і лабораторних робіт.
- **Lecturer** – Викладач, основна роль якого полягає в проведенні лекцій та семінарів.
- **Postgraduate Student** – Аспірант, студент, який проходить наукову підготовку на ступінь кандидата наук.
- **Doctoral Candidate** – Кандидат наук, особа, яка здобуває науковий ступінь.

.

2.5.3 Масив допоміжного персоналу

Масив `SUPPORTING_STAFF` містить професії, що належать до допоміжного персоналу університету. Ці посади не є безпосередньо пов'язаними з навчальним процесом, але забезпечують функціонування університету:

Цей масив містить наступні посади:

- **Librarian** – Бібліотекар, відповідальний за забезпечення доступу до літератури та навчальних матеріалів.
- **Laboratory Assistant** – Асистент лабораторії, який допомагає в проведенні лабораторних робіт та організації досліджень.
- **Methodologist** – Методист, спеціаліст, який займається розробкою методичних рекомендацій для викладачів та студентів.
- **IT Specialist** – IT-спеціаліст, який займається технічною підтримкою, обслуговуванням комп'ютерних систем та мереж.

2.5.4 Масив адміністративно-господарського персоналу

Масив `ADMINISTRATIVE_HOUSEHOLD` містить професії, що відносяться до адміністративно-господарських служб університету. Це професії, що займаються безпосередньо обслуговуванням та технічною підтримкою установи:

Цей масив включає наступні посади:

- **HR Staff** – Персонал відділу кадрів, який займається набором і управлінням кадрами.
- **Accountant** – Бухгалтер, який відповідає за фінансові операції та облік в університеті.
- **Chief Accountant** – Головний бухгалтер, який керує фінансовими процесами в установі.
- **Cleaner** – Прибиральник, відповідальний за чистоту на території університету.
- **Electrician** – Електрик, спеціаліст, який обслуговує електричні мережі та обладнання університету.
- **Plumber** – Сантехнік, який займається обслуговуванням водопостачання та каналізації.
- **Security Guard** – Охоронець, що забезпечує безпеку на території університету.

.

2.5.5 Використання масивів професій у проєкті

Масиви, визначені вище, використовуються для класифікації працівників університету за професіями. Це дозволяє:

- Легко працювати з різними категоріями працівників при введенні та обробці даних.
- Використовувати ці дані для перевірки правильності введених професій в базу даних.
- Групувати працівників за категоріями для подальшого розрахунку зарплати або іншої інформації, що стосується конкретної професії.
- Здійснювати фільтрацію та сортування працівників за їх професійною категорією для зручності аналізу.

Ці масиви значно спрощують формування професій працівників, забезпечують коректність та достовірність введення даних, а також відіграють важливу роль у "сортуванні"викладачів. Це, у свою чергу, дозволяє точно обчислювати середній бал студентів, які навчаються у кожного викладача окремо.

2.6 Формат JSON

Нижче наведені приклади (частинки файлів) зберіганя іфомації про студентів, працівників і викладачі у форматі JSON.

3 Реалізація на C++

Реалізація проєкту на C++ схожа за логікою коду C, але відрізняється у деяких аспектах, які наведені нижче.

3.1 Різниця між реалізації C та C++

3.1.1 JSON

Реалізація на C++ відрізняється у таких аспектах, як робота з JSON. Однією з ключових особливостей є необхідність перетворення об'єктів типу `std::string` у тип `const char*`, оскільки бібліотека cJSON працює з рядками у форматі C-рядків.

Для цього в C++ використовується метод `c_str()` класу `std::string`. Він повертає вказівник на C-рядок, який представляє той самий рядок у пам'яті. Це особливо корисно під час передачі даних до функцій бібліотеки cJSON, які очікують аргументи типу `const char*`.

Наприклад:

```
std::string name = "John";
const char* c_name = name.c_str();
cJSON* json_name = cJSON_CreateString(c_name);
```

У цьому прикладі:

- `name` — це змінна типу `std::string`, яка містить ім'я.
- `c_name` — це змінна типу `const char*`, яка отримана за допомогою `c_str()`.
- `cJSON_CreateString` приймає `const char*` як аргумент і створює JSON-об'єкт.

Використання `c_str()` забезпечує сумісність між сучасним класом рядків `std::string`, який має безліч методів для зручної роботи з текстом, і функціями бібліотеки cJSON, які працюють із C-рядками. Таким чином, цей метод є необхідним інструментом для інтеграції C++ коду з бібліотеками на C.

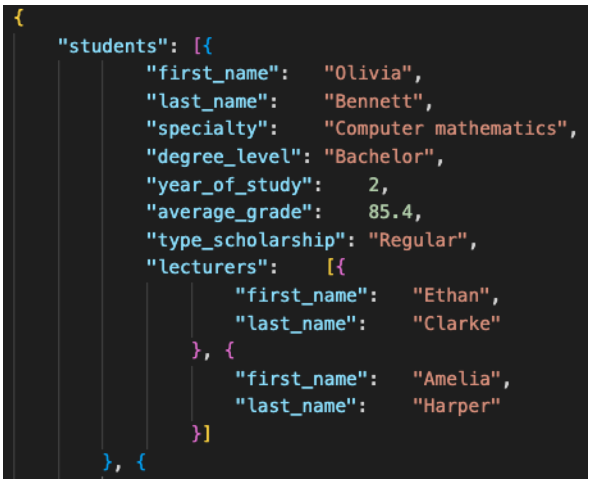

Рис. 5: Частинка json-файлу: students

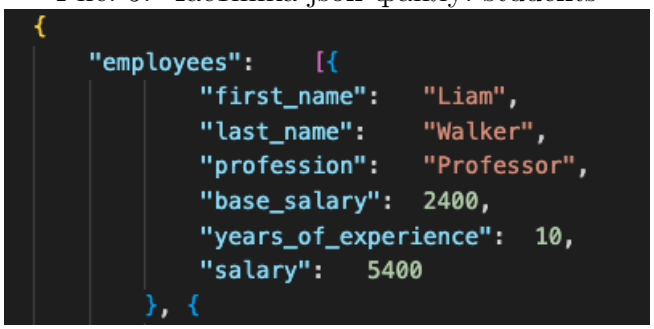

Рис. 6: Частинка json-файлу: employees

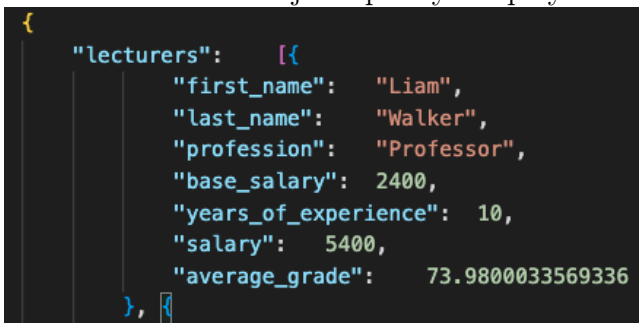

Рис. 7: Частинка json-файлу: lecturers

3.1.2 `sizeof()`

У мові програмування C масиви зберігають інформацію про свій розмір, якщо вони визначені в тій самій області видимості. Це дозволяє використовувати оператор `sizeof()` для отримання розміру масиву в байтах і подальшого обчислення кількості елементів:

```
int arr[] = {1, 2, 3, 4, 5};
size_t size = sizeof(arr) / sizeof(arr[0]); // Розмір масиву: 5
```

Однак у C++ масиви, передані в функції, автоматично перетворюються у вказівники. Це призводить до втрати інформації про їхній розмір, і `sizeof()` повертає розмір вказівника, а не масиву. Для роботи з розміром масивів у C++ зазвичай використовують додаткові константи або змінні, які вручну визначають кількість елементів:

```
const int ADMINISTRATIVE_MANAGEMENT_SIZE = 3;
const int SCIENTIFIC_PEDAGOGICAL_SIZE = 7;
const int SUPPORTING_STAFF_SIZE = 4;
const int ADMINISTRATIVE_HOUSEHOLD_SIZE = 7;
```

У цьому коді:

- Розміри масивів визначені як константи, наприклад,
  `ADMINISTRATIVE_MANAGEMENT_SIZE = 3`.
- Це забезпечує доступ до кількості елементів у будь-якому місці програми.

Отже,

- У C `sizeof()` працює коректно для локально визначених масивів, оскільки зберігається інформація про їхній розмір.
- У C++ для динамічних масивів і масивів, переданих у функції, необхідно вручну відслідковувати їхній розмір, використовуючи додаткові змінні або константи.

Таким чином, у C++ розробнику потрібно явно контролювати розмір масивів, якщо вони використовуються в різних областях видимості або передаються як параметри функцій.

3.2 UML-діаграма

У цьому розділі представлено діаграму UML, що представляє структуру та зв'язки функцій у проєкті.

- Класи Student і Employee успадковують базовий клас Person. Це означає, що всі поля та методи Person доступні в Student і Employee. Це вказує, що Person є базовим (батьківським) класом, а Student і Employee є похідними (дочірніми).

- Поле lecturers у класі Student є об'єктом типу vector, який містить посилання або копії об'єктів класу Person. Це вказує на те, що Student має список викладачів.

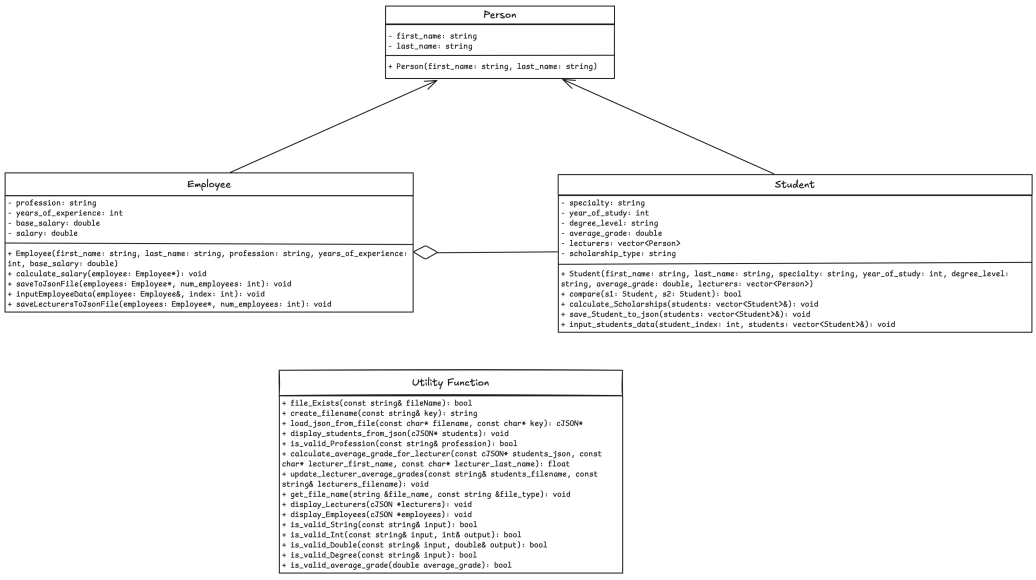

Рис. 8: UML Diagram

4 Висновок

У рамках цього проєкту було реалізовано систему бази даних для обробки інформації про студентів та працівників університету, використовуючи мови програмування C та C++. Створена програма дозволяє ефективно зберігати та обробляти дані про студентів, викладачів і працівників, а також обчислювати їх середній бал та зарплату. Проєкт включає

перевірку даних на коректність за допомогою масивів професій, що спрощує введення та обробку інформації.

Усі дані зберігаються у форматі JSON, що забезпечує зручність для подальшої обробки та використання. Використання бібліотеки ncurses дозволило створити текстовий інтерфейс для зручної взаємодії користувача з програмою.