학생/교사 정보 관리 시스템 보고서

1. 프로젝트 개요

본 프로젝트는 학생과 교사의 정보를 효율적으로 관리하는 프로그램을 개발하는 것을 목표로 하였습니다. 객체지향 프로그래밍(OOP)의 핵심 개념인 클래스, 상속, 다형성, 함수 오버로딩, 연산자 오버로딩, 템플릿을 활용하여 실용적인 정보 관리 시스템을 구현하였습니다.

2. 개발 환경

- **언어:** C++
- **개발 도구:** Visual Studio Code
- **운영 체제:** Windows 10

3. 역할 분담

역할 팀원 담당 업무

팀장 A	코드 설계 및 Git 관리
팀원 B	Student 클래스 및 연산자 오버로딩 구현
팀원 C	Teacher 클래스 구현 및 보고서 작성

4. 핵심 기능

(1) 학생 및 교사 정보 관리

- 학생과 교사의 이름, 나이, 고유 ID를 관리
- Person 클래스를 부모 클래스로 두고 Student, Teacher 클래스를 상속

(2) 학생 성적 관리

- 과목별 성적 추가 기능 (addScore() 함수)

- 평균 성적 계산 기능 (getAverage() 함수)

(3) 연산자 오버로딩 활용

- operator+ 오버로딩을 활용하여 전체 학생 평균 계산
- operator/ 오버로딩을 활용하여 특정 학생의 평균을 정규화

(4) 템플릿을 활용한 범용성 확장

- Student 클래스에서 템플릿을 적용하여 다양한 자료형의 점수 관리 가능

(5) 사용자 인터페이스

- 콘솔 기반 메뉴 시스템을 통해 직관적인 조작 가능

5. 보완할 점

- 현재 Teacher 클래스는 학생과 달리 성적을 관리하지 않음. 교사가 담당하는 학생 목록을 추가할 수 있도록 확장 가능
- Student 클래스의 성적 저장 방식을 동적 할당을 통해 확장 가능하도록 개선
- 파일 입출력을 추가하여 프로그램 종료 후에도 데이터가 유지되도록 구현 필요

6. 프로젝트의 성공 요인

1. **객체지향 개념의 적극 활용**: 상속, 다형성, 템플릿, 연산자 오버로딩을 효과적으로 적용하여 유지보수성을 향상시킴
2. **팀원 간의 원활한 협업**: 역할을 명확히 분담하고 Git을 활용한 코드 관리를 통해 충돌을 최소화
3. **확장성을 고려한 설계**: 추가 기능 개발이 용이하도록 설계 구조를 체계적으로 구성

7. 어려웠던 점 및 해결 방안

(1) 다형성 구현

- 문제: Person 클래스를 추상 클래스로 설정하여 순수 가상 함수 (displayInfo())를

정의하는 과정에서 객체 생성 제한 문제 발생

- 해결: Person 클래스를 직접 생성하지 않고 Student, Teacher 클래스에서만 상속받아 사용하도록 설계 변경

(2) 연산자 오버로딩

- 문제: operator+ 오버로딩을 통해 여러 학생의 평균을 구할 때, 첫 번째 학생의 점수를 초기값으로 설정하는 방식이 직관적이지 않음
- 해결: operator+ 연산을 적용할 때 첫 번째 학생을 별도로 처리하여 연산의 일관성을 유지하도록 수정

(3) 템플릿 적용

- 문제: Student 클래스에 템플릿을 적용하는 과정에서 vector<Student<T>>의 동작 방식에 대한 이해 부족
- 해결: 다양한 자료형(int, double)을 사용한 테스트 코드를 작성하여 템플릿의 동작을 검증하고, 필요한 경우 명시적 특수화를 적용
-

8. 결론

이번 프로젝트를 통해 객체지향 프로그래밍의 주요 개념을 실습하고 팀 협업을 경험할 수 있었습니다. 다음과 같은 추가 기능도 고려해볼 수 있을 것

- **파일 입출력 지원**
- **GUI 기반 인터페이스 추가:** 콘솔이 아닌 그래픽 사용자 인터페이스를 통해 보다 직관적인 조작 가능
- **학생/교사 검색 기능 강화:** 특정 조건(예: 평균 성적, 담당 과목)으로 검색하는 기능 추가