2023 Spring OOP Assignment Report

과제 번호 : 2
학번 : 20220826
이름 : 김민서
Povis ID : kimminseo

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

1. 프로그램 개요

- 본 프로그램은 학생 데이터를 입력받아 관리하는 간단한 데이터베이스를 구현한 것으로, 학생 데이터의 추가, 삭제, 출력, 피벗 변환 기능을 지원한다.

- 프로그램의 디렉토리 구조는 src/에 cpp 파일(이하 소스 파일), include/에 hpp 파일(이하 헤더 파일)로 구성되어 있다.

  소스 파일의 main.cpp는 프로그램의 진입점을 정의한다. student_database.cpp는 학생 데이터를 linked list로 관리하여, 학생 정보를 추가, 삭제, 출력, pivot 연산을 수행하는 클래스를 정의한다. src/util 디렉토리에는 유틸리티 함수를 정의하는 소스 파일이 포함되어 있다. src/aggregator 디렉토리에는 Aggregator 클래스의 파생 클래스를 정의하는 소스 파일이 포함되어 있다.

  헤더 파일은 위의 소스 파일에 정의되어 있는 클래스/함수의 전방 선언을 포함하고 있다. 헤더 파일만으로 정의되는 클래스의 경우, dept.hpp, student.hpp는 각각 학과 및 학생을 나타낸다. list.hpp는 template를 사용한 linked list가 구현되어 있다. 각 파일에 대한 자세한 설명은 README.txt에 작성되어 있다.

- 실행 파일 생성을 위한 Makefile을 제공한다. make 명령을 수행하면 student_database라는 이름으로 실행 파일이 생성된다.

- 안내된 구현 요구사항에서 변경하여 구현한 내용은 다음과 같다.

  1. 리스트는 템플릿으로 구현하였음. int dept_cnt, string dept[9]가 사라진 대신, StudentDatabase 클래스를 정의하여 멤버로 Student의 리스트와 Dept의 리스트를 관리하도록 하였음.

  2. Student 클래스의 메서드 void input_info()를 클래스 밖의 util/get_student_info.cpp으로 분리하였음. 멤버 변수의 초기화가 객체의 생성 이후에 이루어지고, 그 메서드

가 stdin/out에서 직접 입출력을 행하는 것이 적절하지 않다고 판단하였음. 대신, 외부의 함수에서 사용자 입력을 받고 해당 함수를 main에서 호출하도록 하였음. 사용자 입력을 받은 후, 해당 값을 Student 생성자에 매개변수로 넘김으로써 객체의 생성 당시에 멤버 변수를 초기화하도록 하였음.

3. sort() 함수는 매개변수로 string metric 대신 비교 함수의 함수 포인터를 받음.

2. 프로그램의 구조 및 알고리즘

- 본 프로그램의 작성을 위해 구현된 class는 다음과 같다.
  - StudentDatabase
  - Aggregator
    - MinAggregator
    - MaxAggregator
    - AvgAggregator
  - List (template)
    - (templated) List<Student>
    - (templated) List<Dept>
  - Student
  - Dept

- 프로그램이 실행되면, main 함수는 StudentDatabase의 인스턴스를 생성한다. main 함수는 사용자의 입력을 받고 적절하게 StudentDatabase의 public method를 호출한다.

- StudentDatabase는 학생 데이터를 linked list로 관리하여, 학생 정보를 추가, 삭제, 출력, pivot 연산을 수행하는 public method를 제공한다. 내부적으로 학생 및 학과 데이터의 linked list를 멤버로 갖는다.

  후술할 List의 sort() 메서드의 인수로 넘기기 위한 comparator function으로 cmp_dept_()와 cmp_stu_()를 static member로 갖는다. dept list에서 이름으로 dept를 찾기 위한 메서드로 find_dept()를, student list에 특정 student가 존재하는지 판별하기 위한 메서드로 is_stu_exists()를 제공한다. dept list와 student list의 정렬을 위한 dept_sort() 및 stu_sort() 메서드를 제공한다.

  Public 메서드로 save_node(), delete_node(), print_list(), pivot_table()를 제공한다.
  pivot_table()은 인수로 PivotColumn 타입의 enum 변수 piv_col과 Aggregator 타입(후술)의 인스턴스를 받는다. PivotColumn은 pivot 시에 group by를 어느 컬럼 기준으로 할 것인지를 나타내며, Dept, Gender, DeptGender (각각 dept, gender, (dept, gender) 기준으로

group-by)의 값을 갖는다.

- List 클래스는 템플릿으로 정의되었으며, 삽입, 삭제, 정렬 기능을 제공한다. 리스트의 내부 구현을 감추기 위하여 head 포인터를 노출하지 않도록 private으로 접근을 제한하였다. 외부에서 List의 원소를 traverse하기 위한 방법으로 내부적으로 Iterator class를 정의하고 begin() 및 end() 메서드를 정의하였다. begin() 메서드는 리스트의 헤드 (리스트가 비어있는 경우, null)에 해당하는 Iterator 객체를 반환한다. end() 메서드는 내부적으로 node 포인터가 nullptr인 Iterator 객체를 반환한다. Iterator 객체는 prefix ++ 연산자를 오버로딩하여, 연산 결과로 자신이 가리키는 노드의 다음 노드를 가리키도록 정의하였다. Dereference 연산자를 오버로딩하여 Iterator가 가리키는 node의 data를 반환하도록 정의하고, != 연산자를 통해 가리키는 node의 주소를 비교하도록 정의하였다.

  이를 활용하여, begin()으로 Iteration 객체를 생성하고, 이를 end()와 != 연산으로 비교하여 같지 않을 때까지 반복하며, 각 Iteration이 끝나면 ++ 연산으로 다음 node를 가리키도록 반복문을 작성함으로써 리스트의 내부 구현에 대한 정보 없이 모든 원소를 traverse할 수 있도록 하였다. 추가적으로, Iterator 클래스 및 begin(), end() 메서드를 구현함을 통해 range-based for loop 문법을 사용할 수 있게 되었으며, 실제로 본 프로그램에서 List에 대한 traverse는 모두 앞에서 제시한 방법 대신 range-based for loop으로 구현되었다. 단, 해당 문법의 사용을 배제하는 경우를 고려하여 begin(), end() 메서드와 Iterator 객체에 대해 오버로딩된 연산자만을 이용한 traverse 구현 또한 일부 주석으로 남겨두었다.

- List의 정렬에는 insertion sort를 사용하였다. 이는 간단한 in-place 방법으로, array가 아닌 linked list에서의 구현에서 보다 효율적인 장점이 있다. sort() 메서드는 매개변수로 comparator function의 포인터를 받는다. Comparator 함수는 같은 클래스의 서로 다른 인스턴스를 매개변수로 받아 내부적으로 비교 연산을 수행하여 0보다 작거나, 0이거나, 0보다 큰 3가지 경우로 구분되는 리턴 값을 반환한다. sort() 메서드는 내부적으로 정렬 과정에서 해당 comparator 함수를 통해 원소를 비교하는데, 왼쪽(head 쪽) 원소를 comparator의 left hand side에, 오른쪽(tail 쪽) 원소를 right hand side에 두었을 때 값이 항상 양수로 나오도록 정렬한다.

  그 외 List의 메서드로 전체 노드 개수를 반환하는 size(), 노드를 tail 뒤에 삽입하는 insert(), 데이터를 인수로 받아 값이 일치하는 노드 하나를 삭제하는 remove()가 구현되어 있다.

- 테이블의 피벗 변환은 StudentDatabase의 pivot_table() 메서드를 통해 이루어지는데, 이 메서드의 매개변수로 Aggregator 클래스의 인스턴스를 받도록 하였다. Aggregator 클래스는 Max, Min, Avg 연산에 대한 공통된 인터페이스를 제공한다. feed_data() 메서드를 통해 데이터를 누적하도록 하고, 연산 결과를 얻기 위해 evaluate() 메서드를 호출하도록 하였다. Init() 메서드를 통해 다음 Iteration에서 Aggregator 인스턴스의 내부 상태를 초기화

할 수 있도록 하였다. 또한, name 멤버 변수에 Aggregator의 이름을 저장하여 get_name() 메서드를 통해 이름을 얻을 수 있도록 하였다.

Aggregator의 child class에서 feed_data()와 evaluate()를 오버라이딩하여 구현한 예시는 다음과 같다. MaxAggregator는 내부적으로 maximum value를 저장하기 위한 멤버변수 max를 갖고 있다. feed_data()를 호출하며 age를 인수로 넘기면 max와 age를 비교하여 max를 업데이트한다. evaluate() 호출 시 max를 그대로 리턴한다. 다른 예시로, AvgAggregator는 내부적으로 입력받은 age 데이터의 합계와 개수를 저장하기 위한 변수 sum과 count를 멤버로 갖는다. feed_data()를 호출하면 sum과 count가 누적되어 업데이트되고, evaluate() 호출 시에 sum에 count를 나눈 결과를 반환함으로써 average value를 산출한다.

이와 같은 구현의 장점으로, 각 aggregate function (max, min, average)에 대한 pivot_table()의 구현을 다르게 작성할 필요가 사라진다. 또한, pivot table의 기능으로 다른 aggregate function을 추가하고 싶은 경우, 해당 function에 대한 Aggregator 파생 클래스만 작성하면 되므로 프로그램 기능 확장에 유리하다.

- Student 및 Dept는 class가 아닌 struct로 정의하였고, 모든 멤버는 public이다. Student는 멤버 변수로 dept, gender, name, age를 갖고, Dept는 dept_name, cnt (학생 수)를 갖는다.

  각각에 대해 비교 연산자 ==를 오버로딩하였으며, 두 인스턴스를 비교하여 모든 멤버가 서로 같을 때 참이 되도록 구현하였다.

3. 토론 및 개선

- 본 과제를 수행하면서 캡슐화, 상속, 다형성의 객체지향 개념을 익힐 수 있었다. List, StudentDatabase 클래스 구현 과정에서 내부 구현을 숨기고 public interface만 제공함을 통해 캡슐화의 개념을, Aggregator의 child class를 base class의 타입으로 다룸으로써 상속과 다형성의 개념을 사용하였다.

- Lvalue 및 rvalue를 모두 받기 위해서는 parameter type를 constant reference인 "const T &"로 받아야 했다. "T &"로 받을 경우에는 rvalue를 받을 수 없었다. rvalue와 lvalue의 차이를 수업에서 다룬 적이 없었기에 해결 방법을 찾기에 시간이 걸렸다. 또한, 본 프로그램에서는 non-const reference를 받을 필요가 없었기에 위의 방법으로 해결이 가능했지만, non-const로 다루어야 할 경우에는 다른 방법이 필요하다.

- 정렬 알고리즘으로 insertion sort를 사용했는데, 성능 개선을 위해서는 quick sort 및 merge sort 등 다른 알고리즘이 필요하다.

4. 참고 문헌

- https://www.internalpointers.com/post/writing-custom-iterators-modern-cpp
  : List의 range-based for loop를 사용하기 위한 Iterator 클래스 및 begin(), end() 메서드 작성 방법