알고리즘개론 as3 Report 20163107003 남득윤

[**1.** **What is this Program?.** 2](#_Toc88761812)

[**2.** **User Interface Description** 2](#_Toc88761813)

[**2.1** **/index 페이지 with invalid credential** 2](#_Toc88761814)

[**2.2** **/login 페이지** 2](#_Toc88761815)

[**2.3** **/index 페이지with valid credential** 3](#_Toc88761816)

[**2.4** **/transcript-entire 페이지** 3](#_Toc88761817)

[**2.5** **/transcript-request, /transcript-semester 페이지** 4](#_Toc88761818)

[**2.6** **/delete-all 페이지** 4](#_Toc88761819)

[**2.7** **검증 미흡사항** 5](#_Toc88761820)

[**3.** **Program Architecture** 6](#_Toc88761821)

[**3.1** **Project Structure** 6](#_Toc88761822)

[**3.2** **Entity** 6](#_Toc88761823)

[**3.3** **rb\_tree** 8](#_Toc88761824)

[**3.4** **Repository** 9](#_Toc88761825)

[**3.5** **Service** 9](#_Toc88761826)

[**3.6** **Model** 11](#_Toc88761827)

[**3.7** **View** 14](#_Toc88761828)

[**3.8** **Util** 14](#_Toc88761829)

[**3.9** **Test** 14](#_Toc88761830)

[**4.** **Data Initialization Policy.** 15](#_Toc88761831)

[**4.1** **Logic and Implementation.** 15](#_Toc88761832)

[**4.2 Test**. 18](#_Toc88761833)

[**5.** **Conclusions** 18](#_Toc88761834)

1. **What is this Program?.**

* Coffee 대학교의 GLS(학생-수강 관리) 프로그램입니다.
* 18 ~ 21학번 학생에 대해서 각 학생의 수강 과목을 조회, 삭제할 수 있습니다.
* 조회는 전체 조회, 학기 별 조회 기능을 제공합니다.
* 삭제는 수강 과목 별 삭제, 학생 및 학생의 수강 과목 전체 삭제 기능을 제공합니다.

1. **User Interface Description**

이 프로그램은 input에 따라 페이지를 이동하는 방식으로 동작합니다. 웹과 비슷한 방식으로 구현하는 것을 지향하였기에 각 페이지는 url의 path style의 이름을 두었습니다. 각 페이지는 Client(appService)로부터 authentication객체를 전달받습니다. authentication객체는 view에 접근하는 사용자의 인가여부와 사용자 정보(student)를 포함합니다. View는 authentication에 따라 다른 화면을 제공하고 client는 이를 rendering하는 방식으로 구현하고자 하였습니다. 이 장에서는 Input에 따른 페이지의 변화 및 동작을 설명하겠습니다.

* 1. **/index 페이지 with invalid credential**



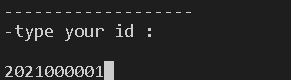
Capture 1. /index page with invalid credential

|  |  |
| --- | --- |
| Option | Move to |
| 1 | /login |
| 2 | 프로그램 종료 |

Table 1 /index options

프로그램 시작 시 등장하는 화면입니다.

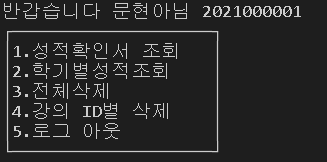
* 1. **/login 페이지**



Capture 2. /login

학번을 입력하여 로그인 할 수 있습니다. 구현시 GLS라고 생각하여 로그인이라고 표현하였지만 관리 프로그램이라면 관리(조회, 삭제)대상 학생의 ID를 입력하는 것으로 이해할 수 있습니다. 저는 기본적으로 이를 학생이 사용자인 GLS와 같은 프로그램으로 생각하고 구현하였으며 보고서를 작성하겠습니다. Id는 기본적으로 10자리의 숫자를 활용하였으며 자세한 내용은 Entity를 설명하는 곳에서 담겠습니다.

* 1. **/index 페이지with valid credential**



Capture 3 /index page with valid credential

|  |  |  |
| --- | --- | --- |
| Option | Move to | 기능 |
| 1 | /transcript-entire | 전체 조회 |
| 2 | /transcript-request | 학기 별 조회 |
| 3 | /delete-all-request | 전체 삭제 |
| 4 | /delete-one-request | 수강 상제 |
| 5 | /index | 로그인 정보 초기화 |

Table 2 /index options

전체 조회는 바로 해당학생의 전체 수강 정보를 출력하는 페이지로 이동합니다. 그 외의 페이지에 대해서는 해당 기능을 수행하기 전에 사용자의 입력을 받는 과정이 필요하므로 요청 페이지로 먼저 요청 페이지로 이동합니다.

* 1. **/transcript-entire 페이지**

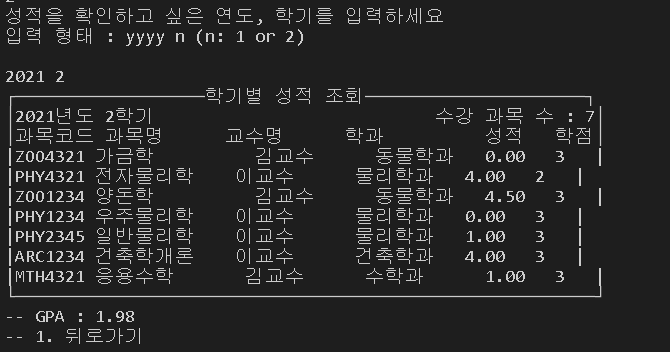
텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

Capture 4 /transcript-entire page

학생의 입학 년도 ~ 2021년도 2학기 까지의 수강정보를 출력합니다. 1을 누르면 index페이지로 이동합니다.

* 1. **/transcript-request, /transcript-semester 페이지**



Capture 5 /transcript-request, /transcript-semester page

/transcript-request 페이지에서 성적을 확인하고 싶은 연도, 학기를 입력하면

해당 학기가 사용자의 입학 년도 ~ 2021년도 2학기의 valid한 학기인지 확인 후 요청한 학기의 수강 정보를 출력합니다. 역시 1을 누르면 index페이지로 이동합니다.

* 1. **/delete-all 페이지**

텍스트이(가) 표시된 사진

자동 생성된 설명

Capture 6. /delete-all page

전체 삭제이니 만큼 한번 더 사용자의 선택을 확인 하기 위한 요청 페이지를 추가로 두었으며 1을 입력할 시 삭제 이후 credential을 invalid로 설정하고 /index 페이지로 redirect합니다.

* 1. **/delete-one-request, /delete-one 페이지**

텍스트이(가) 표시된 사진

자동 생성된 설명

Capture 7 /delete-one page (success)

텍스트이(가) 표시된 사진

자동 생성된 설명

Capture 8 /delete-one page (fail)

삭제할 수강과목의 courseId를 입력 받아 수강(Register) 정보를 삭제합니다. 이때 서로 다른 학기에 대해 여러 번 같은 Course를 수강할 수 있지만(재수강) 같은 학기에 대해서는 불가능합니다. 이에 정확한 수강 정보를 입력 받기위해 courseId와 함께 수강 년도와 학기를 입력 받습니다.

**2.7검증 미흡사항**

1. Input에 대해 숫자를 기대하는 과정에 대해서는 scanf(“%d,n)”를 활용하였는데 이때 문자를 입력하면 무한루프에 빠집니다.
2. Login시 임의의 id를 입력하면 dummyStudent에 해당하는 “덤학생“으로 로그인이 됩니다.
3. **Program Architecture**
   1. **Project Structure**

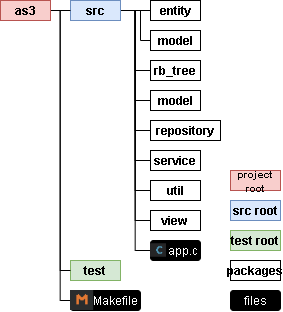


Figure 1 project structure

객체지향적으로 프로젝트를 구성하고 구현하고자 하였습니다. 이에 위와 같은 프로젝트 구조를 가지게 되었습니다. 객체지향적으로 구현을 한 만큼 Class와 Method라는 용어를 사용하여 보고서를 작성하였습니다. app의 구현에는 객체지향의 MVC 패턴을 적용하였습니다.

* 1. **Entity**

프로그램에 사용한 4개의 Entity에 대한 구현입니다. Entity구현사항은 ERD와 같으며 각 Class에는 생성자와 디버깅을 위한 toString method가 포함되어 있습니다.

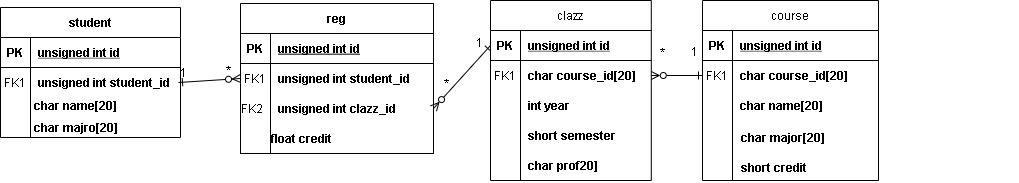


Figure 2 ER diagram

|  |  |  |
| --- | --- | --- |
| Entity | Description | Columns |
| Student | 학생, 인증객체의 내부에도 활용됨 | ID, 학번, 이름, 전공 |
| Reg | Student-Clazz의 관계 Table,  실질적인 수강의 의미를 담당 | ID, 학번, 강의정보 id, 취득성적 |
| Clazz | 학기별로 개설된 강의 정보 | ID, 학수번호, 연도, 학기, 교수명 |
| course | 강의 정보 (이름, 전공, 학점) | ID, 학수번호, 이름, 전공, 학점 |

Table 3 Entity Description

* 1. **rb\_tree**

Rb\_tree는 지난 과제인 as2를 적극 활용하여 구현하였습니다. As2와 비교하여 달라진 점은 3가지입니다.

1. Node의 data

as2에서는 각 node가 int하나의 data만 포함하였지만 이 프로그램에서는 위의 entity 객체를 node의 data로 하였습니다. ***unsigned int id***를 rb tree의 id로 활용하였습니다.

1. 버그 수정,

삭제 과정에서 Red Black tree의 성질이 유지되지 않는 버그를 발견하여 수정하였습니다.

1. Query문 수행을 위한 dfs logic 작성

Rb tree의 id로 탐색을 진행할 때와 달리 data의 column에 해당하는 값으로 검색을 수행하거나 전체 데이터를 모두 조회하는 작업에는 dfs를 적용하는 것이 효율적입니다. 각 Entity 별로 이 프로그램에서 dfs가 필요한 query문의 일반적인 형태는 아래와 같이 작성될 수 있습니다.

|  |  |  |
| --- | --- | --- |
| Entity | Query | 필요한 이유 |
| Student | Select \* from student | 디버깅, reg 초기화 |
|  | Select \* from student where student\_id = ? | 로그인 |
| reg | Select \* from reg where student\_id = ? | 학생의 학기별 수강 과목 정리 -API |
| clazz | Select \* from clazz where year = ? and semester = ? | 학생의 **학기별** 수강 과목 정리 -API |
| course | Select \* from course where course\_id = ? | 학생의 학기별 수강 **과목 정리** -API |

Table 4 Required Query for entities

Rb\_tree 패키지는 기본적으로 지난 과제의 형태를 그대로 활용하며 위와 같은 3가지 변경이 있었으며 tree level에서의 data 접근을 담당합니다. Api에 대해서는 **3.6 model**장에서 자세히 다루겠습니다.

또한, rb\_tree 패키지에 tree 정보을 위한 treeInfo class를 추가하였습니다. 각 tree 객체는 treeInfo를 가지며 service level에서 print를 요청할 수 있습니다. Data에 변화가 있는 요청을 수행한 이후 각 Entity에 해당하는 tree의 정보를 print해주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

Capture result of code 1

Code src>service>appService.c>initApp(appService\*) – print\_treeInfo\_...

* 1. **Repository**

Repository는 Service가 Entity의 Tree구현에 직접적으로 접근하는 것을 지양하기위해 도입하였으며 기본적으로 Service의 명령을 그대로 Tree level에 전달하는 역할을 수행합니다.

또한 Repository는 이 프로그램의 기본 id generation policy인 이용할 수 있는 id중 가장 순서가 빠른 id를 생성하는 정책을 담당합니다. 이에 Service는 input 명령을 수행할 때 db(rb tree) level에서의 id생성을 신경 쓰지 않고 entity의 의미를 가지는 부분에 대해서만 repository에게 전달하면 됩니다.

* 1. **Service**

Entity, Rb\_tree, Repository에서 구현한 level들을 엮어서 app이 사용할 수 있는 Service 객체를 구성하는 것이 Service 패키지의 역할입니다.

Service class들의 연관 관계는 아래와 같습니다.

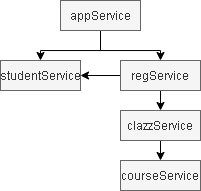


Figure 3 service dependencies

이때 regService는 studentService를 참조하지 않는 것이 논리적으로 옳지만 regService를 초기화 하는 과정에서 필요한 학생의 정보를 직접 모두 조회하여 얻는 방식으로 구현하였기에 regService도 studentService를 참조하도록 하였습니다.

appService에서 stududent의 모든 정보를 조회하여 regService에 전달하여 초기화 하는 방식으로 구현한다면 해당 참조를 끊을 수 있습니다.

이 프로그램에서 가장 중요한 역할을 하는 Class는 appService와 regService입니다. appService는 사용자로부터 input을 입력받아 regService에 전달하고 결과에 따른 화면을 사용자에게 보여줍니다. MVC 패턴을 적용하고자 viewTemplate 객체를 구현하였고 path에 따른 기능을 구분하는 과정에서 dispatcher함수를 구현하였습니다. 이때 실질적인 로직을 담당하는 Controller에 해당하는 부분을 구현하는 부분의 코드 if else문 내부에서 일관되지 않은 style로 작성된 것 같아 조금 아쉽습니다.

regService는 자신이 참조하는 service(regService-> clazzService, clazzService->courseService) 와의 협력을 통하여 표x의 아래 3가지 쿼리를 모두 실행하여 Service level에서 엮는 방식으로 학기별 조회를 구현하여 이를 API로 활용하여 나머지 요구사항은 모두 해당 API를 활용하는 방식으로 구현하였습니다.

|  |  |
| --- | --- |
| 요구사항 | Related method |
| 전체조회 | model1 entireSemester\_model1\_query(regService,student) |
| 학기별조회 | model1 singleSemester\_model1\_query(regService,student, int, short) |
| 전체삭제 | bool deleteById\_studentService(studentService \*, unsigned int)  bool delete\_regByCourseId(regService,student, char \*, int, short) |
| 선택삭제 | void deleteAllRegs\_relatedToStudent(regService,student) |

Table 5 related method for each requirement

* 1. **Model**

Model 패키지는 appService를 작성하는 과정에서 필요한 api객체인 model1 객체와 app이 현재 사용자를 구분하는 용도로 사용하는 authentication객체의 구현을 포함하는 패키지입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Code 2 src>model>model1.c>struct model1

텍스트이(가) 표시된 사진

자동 생성된 설명

Code 3 src>model>model1.c>struct \_semester\_register\_info

텍스트이(가) 표시된 사진

자동 생성된 설명

Code 4 src>model>model1.c>struct \_reg\_info

Model은 학기별 수강 정보를 정리하여 담을 수 있는 객체입니다. JSON처럼 활용할 수 있도록 단계적으로 구현하였습니다. 이를 그대로 view에 전달하는 model로 활용할 것이기 때문에 model이라고 명명하였습니다.

예를 들어 Capture 9과 같은 정보를 담기위해서는 Code n과 같은 JSON구조가 필요합니다. 이를 효율적으로 전달하기 위해 model 객체를 구현하였고 다양한 쿼리의 구현에 활용하였고 view에게 전달하는 model로써 활용하였습니다.

텍스트, 영수증이(가) 표시된 사진

자동 생성된 설명

Capture 10 transcript-all capture

|  |
| --- |
| {      "size" : 2,                     //학기 수      "num\_reg": 10,                  //전체 과목 수      "score\_sum": 74.5,              //취득 성적 총합      "credit\_sum": 38,               //수강 학점 총합      "data":[                        //학기별 정보          {              "year":2021,              "semester":1,              "count":7,              //학기별 수강 과목 수              "score\_sum": 57.0,      //학기별 취득 성적 총합              "credit\_sum":21,        //학기별 수강 학점 총합              "infos":[                           //수강 정보                  {                      "course\_id": "ZOO\_1234",                      "name":"양돈학",                      "major":"동물학과",                      "credit":3,                      "year":2021,                      "semester":1,                      "prof":"김교수",                      "socre":1.0,                      "reg\_id": 7                  },                  {  ...                  }...              ]          }          {              "year":2021,              "semester":2,              "count":6,              "score\_sum": 19.5,              "credit\_sum":18,              "infos":[                  ...              ]          }      ]  } |

Code 5 corresponding json model for capture 9

* 1. **View**

View 패키지는 화면과 관련된 패키지입니다. View와 관련된 모든 로직은 기본적으로 viewTemplate이라는 클래스를 활용해서 동작하는 것을 지향하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Code 6 struct \_viewTemplate

App이 필요한 사용자 정보 혹은 model객체를 포함하여 화면을 요청하면 요청받은 url에 해당하는 view 함수는 출력내용을 2차원 배열 형태로 저장하여 app에게 돌려주면 app은 이를 render합니다.

기본적으로 위와 같은 방향성을 지향하였지만 transcript\_entireView를 구현하는 과정에서 model에 포함된 regInfo를 참조할 때 두번째 이상의 index에대해서 참조 에러가 발생하였는데 이를 해결하는 것이 어려워 그냥 view 함수에서 바로 printf를 호출하는 방식으로 구현하였습니다. 이러한 view도 상당히 많습니다.

* 1. **Util**

util패키지는 다양한 utility 기능을 담당하도록 하였습니다. 어쩔 수 없이 하드 코딩이 필요한 사람이름, 전공 이름, 과목 이름 등을 포함하는 dataSource와 간단한 stringUtil Class가 포함되어 있습니다.

* 1. **Test**

Entity 및 주요 Class에 대한 단위 test입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Figure 4 tests

┌────────────────────┐

│ make T=testName    │

│ make run           │

└────────────────────┘

으로 testName.c를 실행할 수 있습니다.

1. **Data Initialization Policy.**
   1. **Logic and Implementation.**

먼저 년도를 비롯한 많은 Data 초기화 관련 설정사항을 Hard Coding으로 구현한 것이 개인적으로 아쉽습니다.

이 시스템에서 제한사항을 만족시키면서도 가장 간단한 방식으로 구현하기 위해 스스로 추가한 제한사항은 다음과 같습니다.

1. 학생의 입학년도는 2018~2021중 하나이다.
2. 졸업생 및 휴학생은 고려하지 않는다.
3. 학생은 과목을 무작위로 선택한다.

전체 학생의 평균 총 수강학점을 맞추기 위해 학생의 학년을 고려하여 설정하였습니다. 학년에 따라 수강하는 학점의 기댓값을 유지하기 위해서는 학기 별 한 학생이 수강하는 학점의 기댓값을 일정하게 유지하는 것이 필요합니다. 이때 졸업에는 140학점이 필요하므로 매 학기당 17.5 학점을 수강한다면 8학기, 즉 4년만에 졸업이 가능합니다.

|  |  |  |
| --- | --- | --- |
| 입학년도 | 학년 | 수강학점 기댓값 |
| 2021 | 1 | 35 |
| 2020 | 2 | 70 |
| 2019 | 3 | 105 |
| 2018 | 4 | 140 |

Table 6 expected value of registered credit for each grade

하지만 이렇게만 설정하고 학생을 램덤 생성하게 되면 총 수강 기댓값은 70이 아닌 87.5((35+70+105+140)/4)가 될 것입니다. 따라서 평균 총 수강학점을 70으로 맞추기 위해 학년이 낮을수록 생성 횟수를 늘리도록 하였습니다. 아래는 적용된 가중치와 코드상 구현입니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 입학년도 | 학년 | 수강학점 기댓값 | 가중치 |
| 2021 | 1 | 35 | 3 |
| 2020 | 2 | 70 | 1 |
| 2019 | 3 | 105 | 1 |
| 2018 | 4 | 140 | 1 |

Table 7 weights for intended expected output

텍스트이(가) 표시된 사진

자동 생성된 설명

Code 7 src>repository>studentRepository.c>get\_student\_id(int,int,int) line 21

위와 같은 가중치를 통해 학생별 평균 수강학점이 70((35\*3+70\*1+105\*1+140\*1)/6)이 되도록 할 수 있습니다.

이제 학기당 수강학점 기댓값이 17.5가 되도록 한다면. 평균 수강 학점이 70이 되도록 하는 것입니다. 이때 학생이 학기당 최대 3~7과목을 수강하므로 역시 단순 평균을 낼 시에는 5과목을 수강하게 되고 이렇게 되면 모든 과목을 3학점으로 수강하여도 17.5의 기댓값을 낼 수 없습니다. 따라서 이때도 가중치를 통한 기댓값의 조정이 필요합니다. 이 때 강의 역시 무작위로 선택을 하므로 이쯤에서 강의 당 학점의 기댓값을 설정한다면 17.5의 기댓값을 위해 필요한 수강 과목수의 기댓값을 얻을 수 있습니다. 이때 강의는 3학점 과목이 10개 2학점 과목이 2개 존재하며 학생은 수강 과목을 이 중에서 무작위로 선택하는 것으로 하겠습니다. 이때 강의 별 학점 기댓값은 2.83입니다. 따라서 학생은 학기당 평균 6.18(17.5/2.83) 과목을 수강한다면 학기당 17.5학점을 수강할 것으로 기대 할 수 있습니다.

요구사항에서는 최대 3~7개의 과목을 수강한다고 제한하였지만 그냥 3~7과목을 수강하는 것으로 생각하겠습니다. 이때 아래 표와 같은 가중치를 두면 학기당 수강 과목수의 기댓값을 6.18으로 맞출 수 있습니다.

|  |  |
| --- | --- |
| 학기 당 수강 과목수 | 가중치 |
| 3 | 1 |
| 4 | 2 |
| 5 | 3 |
| 6 | 15 |
| 7 | 17 |

Table 8 weihts for intended expected value of register per semester

텍스트이(가) 표시된 사진

자동 생성된 설명

Code 8 src>service>regService.c>initRegService(regService \*)

지금까지 제한사항을 만족하기 위하여 신경을 쓴 초기화 부분을 설명 드렸습니다. 전체적인 initialization flow는 initApp(AppService\*) 함수에 구현하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Code 9 src>service>appService.c>initApp(appService\*) (print function excluded)

언급한 내용 외의 초기화 과정에는 dataSource.c에서의 무작위 선택을 적용하였습니다. (e.g., clazz의 prof column은 service level에서 “김교수” or “이교수”로 무작위로 선택하였습니다.)

## **4.2Test**.

이부분에 대한 Test는 학생의 비율과 학년별 수강 학점 수가 **Table 7**과 **Table 8**의 가중치와 같이 생성이 되는 것을 확인해야 합니다. 이부분은 Code로는 진행하지 않았지만 학생의 학년별 생성 비율과 각 학년이 수강하는 학점의 분포가 의도한대로 등장하는 것을 구현된 app 상에서 확인할 수 있습니다.

1. **Conclusions**

코드 작성시에는 객체지향과 확장성에 가장 중점을 두고자 하였으며 데이터 초기화시에는 최대한 간단하게 주어진 요구사항을 맞추는 것에 중점을 두었습니다. C언어로 모든 것을 작성하고 엮으면서 평소에 별 생각없이 학습하던 객체지향, 데이터베이스, MVC framework의 소중함을 느낄 수 있었습니다.

저는

* overloading을 지원하지 않아 모든 함수에 \_entityName을 넣어서 구분한 것
* 단위 test를 C에서 실행해본 것
* C언어를 객체지향적으로 활용하는 것
* MVC를 console level에서 구현해본 것
* RB tree를 실제 DB와 같이 활용해본 것

의 경험들을 이번 과제를 통해 처음 겪었는데 개인적으로 굉장히 재미 있었고 의미 있게 진행한 것 같습니다.