|  |
| --- |
| 객체지향프로그래밍  실습과제보고서 #8 |
|  |

|  |
| --- |
| 2019년 5월 7일  김예원  소프트웨어학과 |

**실습과제 #8: 객체지향 방식의 성적처리프로그램 #1 작성**

크기 변형이 자유로운 vector 클래스를 이용해 단어의 빈도수를 확인하는 프로그램 작성

**1. Subject 구조체를 Class로 변환**

**\*소스코드**

|  |
| --- |
| class Subject { //과목 클래스  protected:  string m\_name;  int m\_hakjum;  string m\_grade;  float m\_GPA;  public:  void Initialize();  void Initialize(string, int, string);  void InputValue(int &);  void InputValue(string &);  void InputData();  void PrintTitle();  void PrintData();  void CalcGPA();  string GetName();  int GetHakjum();  string GetGrade();  float GetGPA();  }; |

Subject 구조체의 과목명, 과목당 학점, 등급, 학점을 protected 멤버변수로 지정했다. 이들을 관리하는 멤버함수를 public으로 지정했다. 멤버변수에 접근할 수 없으므로 string GetName() 등과 같은 접근자 함수를 이용해 변수의 값을 받을 수 있게 했다.

**\*접근자함수**

|  |
| --- |
| string GetName();  int GetHakjum();  string GetGrade();  float GetGPA(); |

**2. Student 구조체를 Class로 변환**

**\*소스코드**

|  |
| --- |
| class Student { //학생 클래스  protected:  string m\_name;  int m\_hakbun;  int m\_subnum;  Subject\* m\_sub;  float m\_aveGPA;  public:  void Initialize();  void Initialize(string, int, int, Subject\*);  void Remove();  void InputValue(int &);  void InputValue(string &);  void InputData();  void PrintData();  void CalcAveGPA();  string GetName(); //m\_name  int GetHakbun(); //m\_hakbun  int GetSubNum(); //m\_subnum 리턴  float GetAveGPA(); //m\_aveGPA  }; |

Student 구조체에 있던 학생 이름, 학번, 과목수, 과목 정보 동적할당, 평균학점을 멤버 변수로 지정했다. 또한 이들을 관리하는 멤버함수를 선언했다. Subject 클래스와 마찬가지로, protected로 지정된 멤버변수의 값을 받기 위해 접근자 함수를 만들었다.

**3. Subject 클래스의 InputData()**

**\*소스코드**

|  |
| --- |
| void Subject::InputData() { //정보를 입력 받는 함수  string name, grade;  int hakjum;  cout << "교과목명 : ";  InputValue(name); //교과목명 입력  cout << "과목학점수 : ";  InputValue(hakjum); //과목학점수 입력  cout << "과목등급<A+ ~ F> : ";  InputValue(grade); //과목 등급 입력  Initialize(name, hakjum, grade); //초기화  } |

오버로딩 된 함수 InputValue를 이용했다. 입력받은 값은 Initialize()함수로 인해 초기화의 과정을 거친 후 멤버변수에 저장된다.

**4. Student 클래스의 InputData()**

**\*소스코드**

|  |
| --- |
| void Student::InputData() { //정보를 입력 받는 함수  string name;  int hakbun;  int subnum = 0;  int j = 0;  cout << "이름 : ";  InputValue(name);  cout << "학번 : ";  InputValue(hakbun);  cout << "수강한 과목 수를 입력 : ";  cin >> subnum;  Subject\* sub = new Subject[subnum]; //과목 수 동적할당  while (j < subnum) {  sub[j].Initialize();  cout << "\n" << "\* 수강한 과목 " << subnum << "개와 각 교과목명, 과목학점, 과목등급을 입력하세요\n";  sub[j].InputData();  sub[j].CalcGPA(); //과목 평점 계산  cout << endl;  j++;  }  Initialize(name, hakbun, subnum, sub); //입력 받은 값으로 초기화  } |

수강한 과목 수를 입력 받은 뒤 과목수에 대해 동적할당을 한다. 동적할당을 통해 수강한 과목 수만큼의 과목 정보를 입력 받을 수 있게 되었다.

**5. Student의 PrintData()를 통한 전체 학생 정보 출력**

**\*소스코드**

|  |
| --- |
| void Student::PrintData() { //최종적인 출력  cout << "=========================================" << endl;  cout << " 이름 : " << m\_name << " 학번 : " << m\_hakbun << endl;  m\_sub->PrintTitle();  for (int i = 0; i < m\_subnum; i++) {  m\_sub[i].PrintData();  }  cout << endl;  cout.width(20);  cout << "평균평점 : ";  cout.width(10);  CalcAveGPA(); //평균평점 계산  cout << m\_aveGPA;  cout << endl;  } |

**\*main에서의 호출**

|  |
| --- |
| for (i = 0; i < studentnum; i++) {  St[i].PrintData();  } |

동적할당한 학생 수 배열에 따라 PrintData()를 호출했다. PrintData() 함수는 Student 클래스 내부에 있는 m\_sub를 통해 각 학생의 각각의 과목정보를 출력하도록 한다.

**6. ShowData라는 friend 함수를 사용한다.**

**\*소스코드**

|  |
| --- |
| void ShowData(const Student &s) {  cout << s.m\_name << "이 수강한 과목은 총" << s.m\_subnum << "개로";  cout << "수강한 과목의 평균평점은 " << s.m\_aveGPA << "입니다.\n";  } |

**\*Student 클래스에서의 호출**

|  |
| --- |
| class Student { //학생 클래스  friend void ShowData(const Student &); |

**\*main에서의 호출**

|  |
| --- |
| for (i = 0; i < studentnum; i++) {  St[i].PrintData();  ShowData(St[i]);  } |

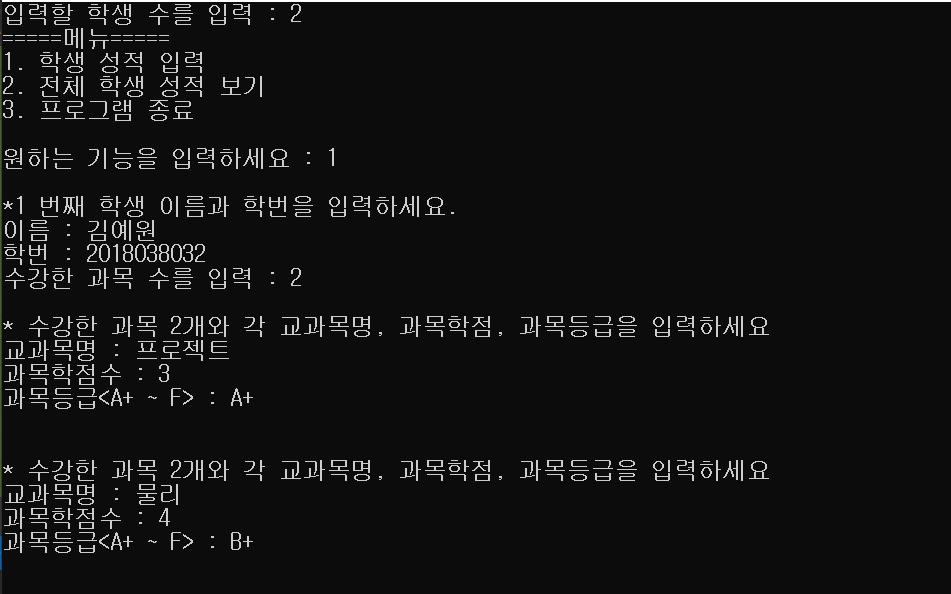
main이 위치한 파일에 ShowData()의 정의를 작성했다. Student 클래스에서는

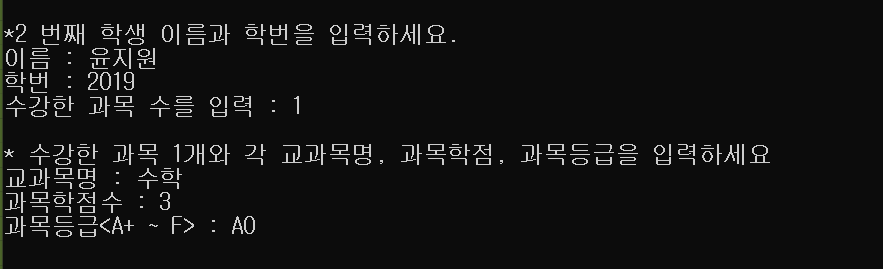
함수 선언 앞에 friend를 입력해주었다. 메뉴 2번을 선택하면 자연스럽게 출력되

도록 작성했다. ShowData() 함수를 이용하니 기존에 불가능했던 protected 내의

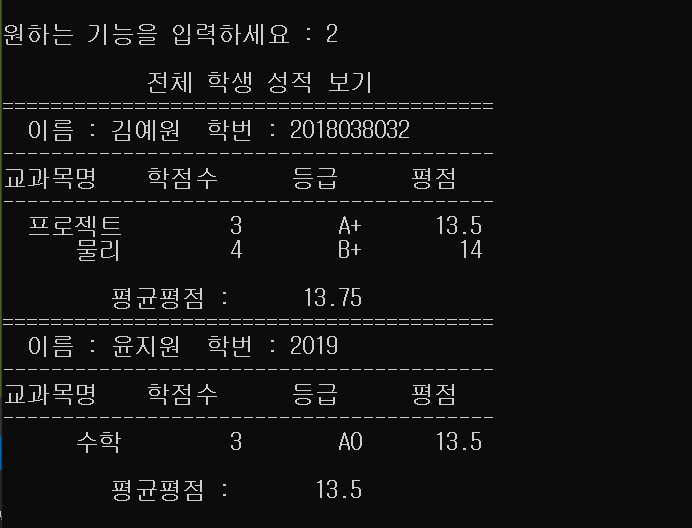
멤버변수 접근이 가능했다.

**이번 과제 실행화면**

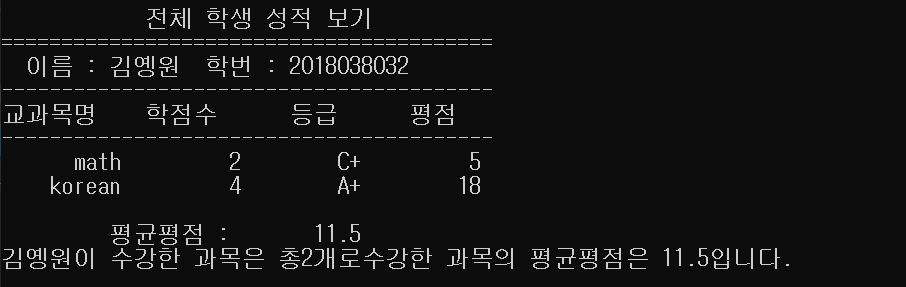


****

학생 수와 과목 수를 입력 받을 때 동적할당을 이용했다.



정상적으로 출력된 모습



ShowData함수를 이용해 마지막 문장을 출력했다.

**느낀점**

객체지향 답게 프로그래밍 해본 것이 처음이기도 했고 전반적으로 멤버 함수가 많아서 쉽게 느껴지지 않았다. Initialize 함수의 경우, m\_name 등에 저장되기 전, 초기화의 과정을 거쳐야 했는데, 처음에는 정확히 무슨 용도인지 파악하지 못했었다. 마지막에 함수를 점검하면서 사용하지 않은 함수를 확인하다가 용도를 파악할 수 있었다.

Student 클래스의 PrintData()를 이용할 때 어려움을 겪었다. 포인터를 사용하지 않는 점에서 절차지향 프로그래밍 보다 안전함을 느꼈지만, 아직 익숙하지 않았던 터라 m\_sub[i].PrintData();를 생각하기까지 많은 고민의 과정을 거쳐야했다. ‘포인터를 이용했다면 cout << s.Sub[i].Subname;와 같이 표현할 수 있었을텐데, 함수 인자를 포함할 순 없을까’ 등등 과제 하는 날 내내 머릿속에서 그런 고민들이 맴돌았던 것 같다.

분할컴파일 과정도 처음에는 익숙하지가 않았지만 사용하다 보니 너무 편리했다. 항상 긴 소스파일들은 함수 정의나 선언 부분을 확인해야 할 때 등 스크롤을 많이 내리게 되면 불편한 일이 많았다. 그래서 비주얼 스튜디오의 창 분할 기능을 많이 사용했었는데, 분할 하는 방법이 창 분할 보다 더 가독력이 높은 것 같다. 분할컴파일을 하고 난 뒤에는 좀 더 편하게 코딩에 집중할 수 있었던 것 같다.