객체지향프로그래밍 실습과제보고서 #5

2019 년 4 월 5 일

김예원
소프트웨어학과

실습과제 #5: 동적메모리를 사용하여 성적처리프로그램 #3 개선

: 구조체 Student와 Subject를 동적할당해서 입력 받는 학생 수와, 학생 별 과목 수를 사용자가 정할 수 있도록 한다.

1. 학생 구조체와 과목 구조체를 포인터 변수로 선언해 동적으로 학생 수와 과목 수를 정하게 한다

*소스코드

```
Student *St = new Student[StudentNum]; //Student 동적할당

pSt[i].Sub = new Subject[pSt[i].SubNum]; //Subject에 대한 동적할당
```

Student 구조체(학생 구조체)는 main에서 선언해, 입력을 담당하는 함수인 InputData()에 인수로 넘겨주었다.

Subject 구조체(과목 구조체)는 InputData() 함수 내에서 선언되었으며, 매개변수로 넘어온 Student 구조체의 Sub에 접근하게 해 동적할당을 했다

***실행시킨 모습**

2. 학생 신상 정보, 성적 정보 입력 부분을 InputData() 함수로 모듈화한다.

***소스코드**

```
void InputData(Student *pSt, int StudentNum) { //학생정보와 과목정보를 받는
함수

	int i = 0;
	while (i < StudentNum) { //학생 n명의 데이터를 입력 받는다
		cout << "₩n";
		cout << "*" << i + 1 << " 번째 학생 이름과 학번을 입력하세요."
```

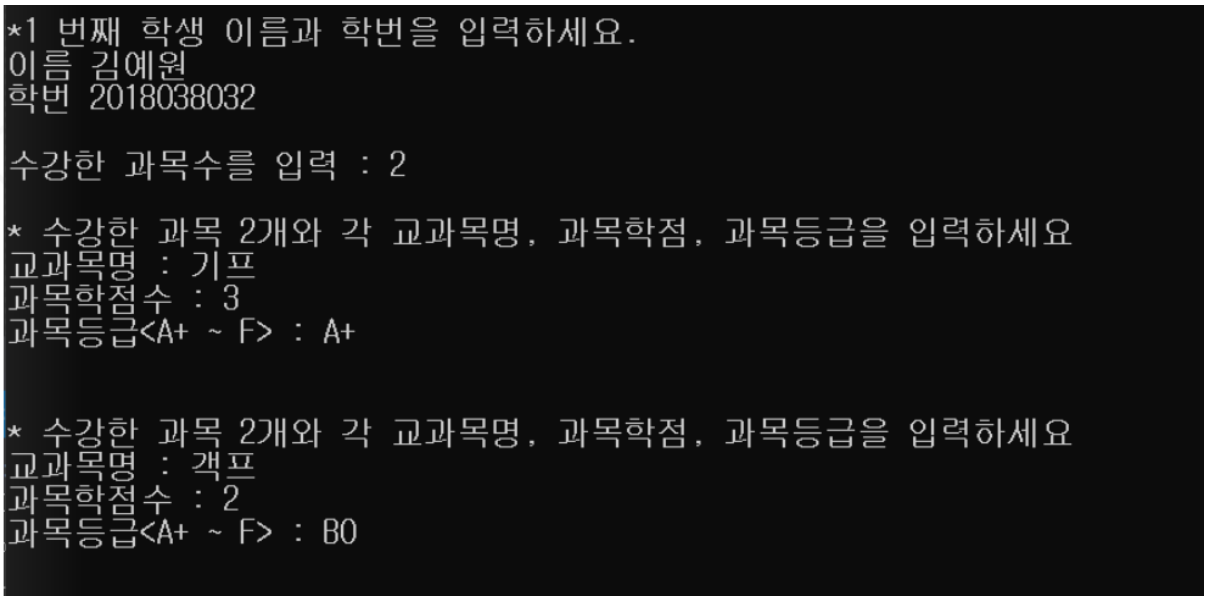
```
<< "₩n";
                cout << "이름 ";
                cin >> pSt[i].StdName;
                cout << "학번 ";
                cin >> pSt[i].Hakbun;
                cout << endl;

                cout << "수강한 과목수를 입력 : ";
                cin >> pSt[i].SubNum;

                pSt[i].Sub = new Subject[pSt[i].SubNum]; //Subject에 대한
동적할당

                int j = 0;
                while (j < pSt[i].SubNum) { //학생의 과목 정보를 동적할당으로
입력받는다

                        cout << "₩n" << "* 수강한 과목 " << pSt[i].SubNum <<
"개와 각 교과목명, 과목학점, 과목등급을 입력하세요₩n";
                        cout << "교과목명 : ";
                        cin >> pSt[i].Sub[j].Subname;

                        cout << "과목학점수 : ";
                        cin >> pSt[i].Sub[j].Hakjum;

                        cout << "과목등급<A+ ~ F> : ";
                        cin >> pSt[i].Sub[j].Grade;
                        cout << endl;

                        pSt[i].Sub[j].GPA = 0;
                        pSt[i].Sub[j].GPA = CalcGPA(pSt[i].Sub[j]); //교과목
평균평점-즉시 초기화
                        j++;
                }
                pSt[i].AveGPA = CalcAveGPA(pSt[i].SubNum, pSt[i].Sub); //즉시
초기화
                i++;

                cout << endl;

        }
}
```

기존에 main에 있던 학생 신상정보와 과목 정보 입력 받는 기능을 InputData()라는 함수로 옮겼다. InputData()에서 Subject에 대해서 동적할당한 것을 알 수 있다.

***함수호출**

```
if (func == 1) {                          // 1. 학생 성적 입력
        InputData(St, StudentNum);
}
```

***실행화면**



수강한 과목 수만큼 과목 정보를 입력하고 있다.

3. 전체 학생의 학번, 이름 목록을 출력하는 PrintAllStdList() 함수를 구현한다.

***소스코드**

```
void PrintAllStdList(Student *pSt, int StudentNum) {
        cout << "****************************" << endl;
        cout << "      학번" << "      이름" << endl;
        cout << "****************************" << endl;

        for (int i = 0; i < StudentNum; i++) {
                cout.width(10);
                cout << pSt[i].Hakbun;
                cout.width(10);
                cout << pSt[i].StdName << endl;
```

```
        }
        cout << endl;
        cout << "*****************************" << endl;

}
```

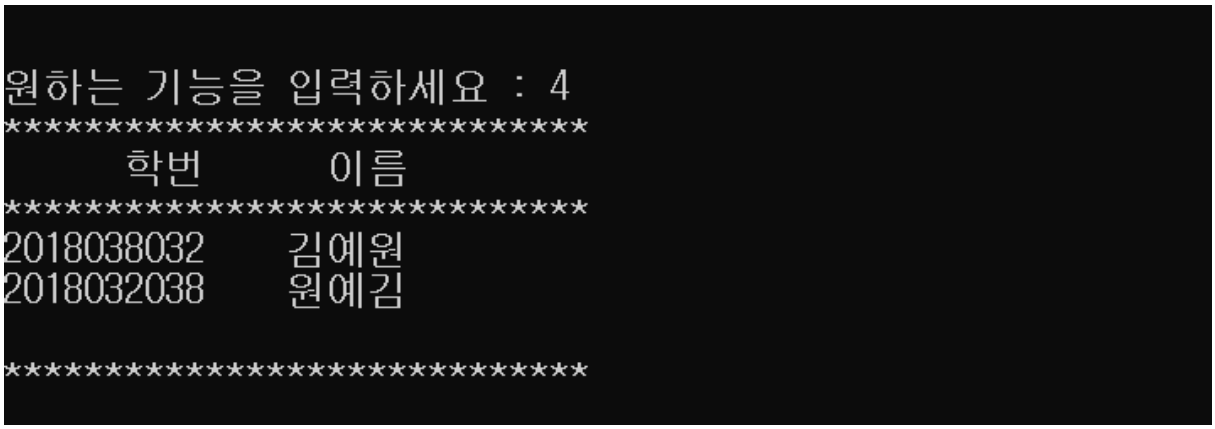
1번에서 동적할당 했던 Student와 StudentNum의 값을 매개변수로 넘겨받았다. 반복문을 이용해 할당 되어있는 데이터를 출력하게 했다.

***함수 호출**

```
else if (func == 4) {                  //4. 전체 학생 목록 보기
        PrintAllStdList(St, StudentNum);

}
```

4번 전체학생 목록 보기 기능을 선택하면 위의 함수가 실행되어 학생의 목록을 보여준다.

***실행화면**



4. remove() 함수를 이용해 동적할당을 해제했다

***소스코드**

```
void remove(Student* pSt,int StudentNum) { //동적배열을 해제한다
	for (int i = 0; i < StudentNum; i++) {
			delete[] pSt[i].Sub; //Subject에 대한 동적배열 해제
	}
	delete[] pSt; //Student에 대한 동적배열 해제
	pSt = NULL;
}
```

***함수 호출**

```
else {                                                  //5. 프로그램 종료
	cout << "종료합니다." << endl;

	remove(St, StudentNum);
	return 0;
}
```

5번 프로그램 종료 메뉴를 선택하면 프로그램이 종료되는데, 그 전에 remove()함수를 호출해 동적할당 해제가 완료되게 했다. Student를 넘기면 함수 내부에서 Subject에 접근해 반복문을 이용해 각 Subject의 동적배열을 해제한다. 반복문이 종료되면 Student의 동적할당이 해제된다.

동적할당에 대한 미션은 전반적으로 어렵지 않았다. 함수를 만들 때 까지는 수월하게 진행되는 듯했다. 하지만 동적할당 해제 부분에서 많은 고민을 해야했었다. Student의 경우, return 0; 위에 delete[] St;를 선언하면 됐었지만 문제는 InputData() 안에서 선언한 Subject 동적할당 이었다. InputData() 내부에서 이후에 함수에서 호출될 때에 메모리가 해제되어 불러올 것이 없어 쓰레기 값을 출력할 것이었다. 어떻게 해결해야 할지 주말 내내 고민했었다. 마침내 생각해낸 것은 함수로 만들어 필요할 때에 할당해제를 시킬 수 있게 하는 방법이었다. Student를 동적할당한 St와 StudentNum을 넘겨받아서 학생의 수 만큼의 과목 정보를 해제해 Subject의 동적할당을 해제했다. 이후 St 자체도 동적할당을 해제하게끔 만들었는데, 이 방법은 동적할당 과정을 함수로 모듈화 할 수 있을 뿐만 아니라 두 개의 동적할당 해제를 동시에 할 수 있어서 소스코드가 약간 더 깔끔해 질 수 있었다.

동적할당 해제하는 함수를 생각하는 것이 오래 걸렸던 이유는 함수로 모듈화 하는 것이 아직까지는 적응되지 않았기 때문이었던 것 같다. 그래도 결국엔 내 힘으로 그런 함수를 생각해내고 적용할 수 있었다는 것이 크나큰 뿌듯함으로 다가왔다. 그동안 과제를 하면서 선배들의 도움을 구해야 할 때가 있었는데, 사실 그럴 때마다 내가 잘하지 못하는 것 같아서 약간의 자괴감이 들었었다. 하지만 이번 일을 통해 나도 언젠간 잘 할 수 있을 것이라는 자신감을 얻게 되었다.