#ifndef COURSE_H
#define COURSE_H

#include <string>

using namespace std;

struct course {
    string course_name;
    int num_credits;
    string term;
    string grade;
};

#endif
#ifndef STUDENT_H
#define STUDENT_H

#include "course.h"

using namespace std;

class Student {
private:
  int id_num;
  string first_name;
  string last_name;
  string major;
  float gpa;
  int num_credits;
  int num_courses;
  course* courses;
  static int count;

public:
  // constructors
  Student();
  Student(int, string, string, string, float, int, int);
  // accessors
  int get_id_num() const;
  string get_first_name() const;
  string get_last_name() const;
  string get_major() const;
  float get_gpa() const;
  int get_num_credits() const;
  int get_num_courses() const;
  string get_course_name(int) const;
  int get_course_credit(int) const;
  string get_course_term(int) const;
  string get_course_grade(int) const;
  // mutators
  void assign_id_num(int);
  void set_first_name(string);
  void set_last_name(string);
  void set_major(string);
  void calculate_gpa();
  void calc_num_credits();
  void set_num_courses(int);
  void set_course(string, int, string, string, string, int);
  void add_course(string, int, string, string);
  static int get_count();
  // copy constructor
  Student(const Student&);
  // assignment operator overload
  const Student& operator=(const Student &);
  // destructor
  ~Student();
};
#endif
```cpp
#include <iostream>
#include "student.h"

using namespace std;

//constructors

Student::Student() {
    id_num = 0;
    first_name = "Test";
    last_name = "Student";
    major = "n/a";
    gpa = 0.0;
    num_credits = 0;
    num_courses = 0;
    courses = NULL;
}

Student::Student(int idn, string fn, string ln, string m, float gpa, int ncr, int nc) {
    id_num = idn;
    first_name = fn;
    last_name = ln;
    major = m;
    this->gpa = gpa;
    num_credits = ncr;
    num_courses = nc;
    courses = new course[num_courses];
    for(int i=0; i<num_courses; i++) {
        courses[i].course_name = "Course";
        courses[i].num_credits = 0;
        courses[i].term = "n/a";
        courses[i].grade = "n/a";
    }
}

//accessors
int Student::get_id_num() const { return id_num; }
string Student::get_first_name() const { return first_name; }
string Student::get_last_name() const { return last_name; }
string Student::get_major() const { return major; }
float Student::get_gpa() const { return gpa; }
int Student::get_num_credits() const { return num_credits; }
int Student::get_num_courses() const { return num_courses; }
string Student::get_course_name(int index) const { return courses[index].course_name; }
int Student::get_course_credit(int index) const { return courses[index].num_credits; }
string Student::get_course_term(int index) const { return courses[index].term; }
string Student::get_course_grade(int index) const { return courses[index].grade; }

//mutators
void Student::assign_id_num(int id) { id_num = id; }
void Student::set_first_name(string name) { first_name = name; }
```
void Student::set_last_name(string name) { last_name = name; }
void Student::set_major(string m) { major = m; }

void Student::calculate_gpa() {
  float total = 0;
  for (int i=0; i<num_courses; i++) {
    if (courses[i].grade == "A")
      total += (4.0 * courses[i].num_credits);
    else if (courses[i].grade == "A-")
      total += (3.7 * courses[i].num_credits);
    else if (courses[i].grade == "B+")
      total += (3.3 * courses[i].num_credits);
    else if (courses[i].grade == "B")
      total += (3.0 * courses[i].num_credits);
    else if (courses[i].grade == "B-")
      total += (2.7 * courses[i].num_credits);
    else if (courses[i].grade == "C+")
      total += (2.3 * courses[i].num_credits);
    else if (courses[i].grade == "C")
      total += (2.0 * courses[i].num_credits);
    else if (courses[i].grade == "C-")
      total += (1.7 * courses[i].num_credits);
    else if (courses[i].grade == "D+")
      total += (1.3 * courses[i].num_credits);
    else if (courses[i].grade == "D")
      total += (1.0 * courses[i].num_credits);
    else
      total += 0.0;
  }
  gpa = total/float(num_credits);
}

void Student::calc_num_credits() {
  int total = 0;
  for (int i=0; i<num_courses; i++) {
    total += courses[i].num_credits;
  }
  num_credits = total;
}

void Student::set_num_courses(int nc) { num_courses = nc; }

void Student::set_course(string name, int credits, string term, string grade, int index) {
  if (index < 0 || index > num_courses) {
    cout << "Index error" << endl;
  } else {
    courses[index].course_name = name;
    courses[index].num_credits = credits;
    courses[index].term = term;
    courses[index].grade = grade;
  }
}

/***************************************************************************/
** Function: add_course  
** Description: appends a new course to the old  
** Parameters: string name of course, int number of credits, string term that the  
** course was taken, string grade received in the course  
** Pre-Conditions: all parameters are initialized to safe values  
** Post-Conditions: num_courses updated by one, new course is appended  
*********************************************************************************/

```cpp
void Student::add_course(string name, int credits, string term, string grade) {
    num_courses++;
    course* temp = new course[num_courses];
    for(int i=0; i<num_courses-1; i++) {
        temp[i] = courses[i];
    }
    delete [] courses;
    courses = temp;
    set_course(name, credits, term, grade, num_courses-1);
}
```

```cpp
int Student::count = 0;
```

```cpp
int Student::get_count() {
    count++;
    return count;
}
```

```cpp
//copy constructor
Student::Student(const Student& copy) {
    id_num = copy.id_num;
    first_name = copy.first_name;
    last_name = copy.last_name;
    major = copy.major;
    gpa = copy.gpa;
    num_credits = copy.num_credits;
    num_courses = copy.num_courses;
    if(num_courses == 0)
        courses = NULL;
    else {
        courses = new course[num_courses];
        for(int i=0; i<num_courses; i++) {
            courses[i] = copy.courses[i];
        }
    }
}
```

```cpp
1. Pass by value (106)  
2. Return value (106)  
3. Initialize obj (similar to 16)
```
const Student& Student::operator=(const Student& copy) {
    id_num = copy.id_num;
    first_name = copy.first_name;
    last_name = copy.last_name;
    major = copy.major;
    gpa = copy.gpa;
    num_credits = copy.num_credits;
    num_courses = copy.num_courses;
    if(courses != NULL)
        delete [] courses;
    if(num_courses == 0)
        courses = NULL;
    else {
        courses = new course[num_courses];
        for(int i=0; i<num_courses; i++) {
            courses[i] = copy.courses[i];
        }
    }
    return *this;
}

//destructor
Student::~Student() {
    delete [] courses;
}

//assignment operator overload
Called when both the left and right operand are of the class type
Can't assume the left is empty

Called when a variable of the class type goes out of scope
1. When a function ends
2. When a program ends
3. When a local block ends
4. When delete gets called
C.J. Cregg Political_Science 4
PS201 4 F16 A
PS202 4 W17 A
PS300 4 SP17 B
COMM400 3 F16 A
Olivia Dunham Criminal_Justice 2
COMM100 3 F16 A-
CH200 4 W17 B+
Josh Lyman Law 4
PS401 4 F16 A
PS402 4 W17 A
PS403 4 SP17 A
HST501 3 F16 B+
Toby Ziegler Communications 3
COMM315 4 F16 A
ECON200 4 W17 B-
COMM400 3 SP17 A
Leslie Knope Public_and_Environmental_Affairs 3
HST350 3 W17 A
BIO111 4 W17 C+
PS201 4 W17 B+
Sam Seaborn Law 4
PS401 4 F16 A
PS402 4 W17 A
PS403 4 SP17 A
HST501 3 F16 A
Walter Bishop General_Sciences 3
PH500 4 W16 A
BIO500 4 W16 A
CH500 4 W16 A
```cpp
#include <iostream>
#include <fstream>
#include "student.h"

void pop_from_file(Student* s, int num_students) {
    ifstream rf;
    string f_name, l_name, m, cn, term, grade;
    int nc = 0, id, ncr = 0;
    rf.open("input.txt");
    for(int i=0; i<num_students; i++) {
        rf >> f_name;
        rf >> l_name;
        rf >> m;
        rf >> nc;
        id = 100 + Student::get_count();
        s[i] = Student(id, f_name, l_name, m, 0.0, 0, nc); //What constructor is called? How many times?
        for(int j=0; j<s[i].get_num_courses(); j++) {
            rf >> cn;
            rf >> ncr;
            rf >> term;
            rf >> grade;
            s[i].set_course(cn, ncr, term, grade, j);
        }
        s[i].calc_num_credits();
        s[i].calculate_gpa();
    }
    rf.close();
}

void print_students(const Student& a) {
    //Are any of the Big 3 used in this function? Why or why not?
    // No because we are passing by reference
    cout << "Name: " << a.get_first_name() << " " << a.get_last_name() << endl;
    cout << "ID: " << a.get_id_num() << endl;
    cout << "Major: " << a.get_major() << endl;
    cout << "GPA: " << a.get_gpa() << " Credits: " << a.get_num_credits() << endl;
    cout << "Courses: " << endl;
    for(int i=0; i<a.get_num_courses(); i++) {
        cout << a.get_course_name(i) << " Credits: " << a.get_course_credit(i) << endl;
        cout << " Term: " << a.get_course_term(i) << " Grade: " << a.get_course_grade(i) << endl;
    }
    cout << endl;
}
```

void print(Student* s, int num_students) {
    // Are any of the Big 3 used in this function? Why or why not?
    // No because we are passing addresses
    for(int i=0; i<num_students; i++) {
        print_students(s[i]);
    }
}

int editing_options() {
    int choice = 0;
    cout << "What would you like to change about the student?" " endl;
    cout << "0. Nothing, this was a mistake" " endl;
    cout << "1. Major" " endl;
    cout << "2. Add a course" " endl;
    cout << "Selection: " ;
    cin >> choice;
    cin.ignore();
    cin.clear();
    return choice;
}

void change_major(Student& s) {
    // Are any of the Big 3 used in this function? Why or why not?
    // No because we are passing by reference
    string m;
    getline(cin, m);
    s.set_major(m);
}

void add_course(Student& s) {
    // Are any of the Big 3 used in this function? Why or why not?
    // No because we are passing by reference
    string course_name, term, grade;
    int credits = 0;
    cout << "What is the name of the course? ";
    getline(cin, course_name);
    cout << "What term was it taken? ";
    getline(cin, term);
    cout << "What was the letter grade earned? ";
    getline(cin, grade);
    cout << "How many credits was the course? ";
    cin >> credits;
    s.add_course(course_name, credits, term, grade);
}
Student x = s;

Student edit_student(Student s) {
    // Are any of the Big 3 used in this function? Why or why not?
    // Yes, copy constructor for pass by value, copy constructor for returning
    // the value, destructor when the function ends \%
    int choice = editing_options();
    if(choice == 1) {
        change_major(s);
    }
    else if(choice == 2) {
        add_course(s);
    }
    return s;
}

void remove_student(Student** s, int num_s, int index) {
    // Are any of the Big 3 used in this function? Why or why not?
    // Yes, destructor assignment operator for copying to temp, destructor when
delete is called
    Student* temp = new Student[num_s-1];
    for(int i=0; i<index; i++) {
        temp[i] = (*s)[i];
    }
    for(int i=index; i<(num_s-1); i++) {
        temp[i] = (*s)[i+1];
    }
    delete [] (*s);
    (*s) = temp;
}

int main() {
    // CREATE STUDENT ARRAY
    Student* students = new Student[7]; // Which constructor is called?
    How many times? Default 7 times
    // POP FROM FILE
    pop_from_file(students, 7);
    print(students, 7);

    print_students(students[0]);
    // EDIT STUDENT
    students[0] = edit_student(students[0]); // Does one of the Big 3 get called here? Which one? How many times?
    print_students(students[0]);
    // REMOVE STUDENT
    remove_student(&students, 7, 3);
    print(students, 6);

    delete [] students; // Is the destructor called? How many times?
    Yes 6 times
    return 0;
}

Total Calls
Default Constructor: 7+6 = 13
Non default Constructor: 7
Copy Constructor: 2

Assignment Operator Overload: 1+
Destructor: 20