Key

```
1   #ifndef COURSE_H
2   #define COURSE_H
3
4   #include <string>
5
6   using namespace std;
7
8   struct course {
9       string course_name;
10      int num_credits;
11      string term;
12      string grade;
13  };
14
15  #endif
```

Header gaurds will prevent multiple inclusions

Not allowed

```
1   #ifndef STUDENT_H
2   #define STUDENT_H
3
4   #include "course.h"
5
6   using namespace std;
7
8   class Student {
9       private:
10          int id_num;
11          string first_name;
12          string last_name;
13          string major;
14          float gpa;
15          int num_credits;
16          int num_courses;
17          course* courses;
18          static int count;
19      public:
20          //constructors
21          Student();
22          Student(int, string, string, string, float, int, int);
23          //accessors
24          int get_id_num() const;
25          string get_first_name() const;
26          string get_last_name() const;
27          string get_major() const;
28          float get_gpa() const;
29          int get_num_credits() const;
30          int get_num_courses() const;
31          string get_course_name(int) const;
32          int get_course_credit(int) const;
33          string get_course_term(int) const;
34          string get_course_grade(int) const;
35          //mutators
36          void assign_id_num(int);
37          void set_first_name(string);
38          void set_last_name(string);
39          void set_major(string);
40          void calculate_gpa();
41          void calc_num_credits();
42          void set_num_courses(int);
43          void set_course(string, int, string, string, int);
44          void add_course(string, int, string, string);
45          static int get_count();
46          //copy constructor
47          Student(const Student&);
48          //assignment operator overload
49          const Student& operator=(const Student &);
50          //destructor
51          ~Student();
52
53  };
54
55  #endif
```

Header gaurds

```
1   #include <iostream>
2   #include "student.h"
3
4   using namespace std;
5
6   //constructors
7
8   Student::Student() {
9       id_num = 0;
10      first_name = "Test";
11      last_name = "Student";
12      major = "n/a";
13      gpa = 0.0;
14      num_credits = 0;
15      num_courses = 0;
16      courses = NULL;
17  }
18
19  Student::Student(int idn, string fn, string ln, string m, float gpa, int
     ncr, int nc) {
20      id_num = idn;
21      first_name = fn;
22      last_name = ln;
23      major = m;
24      this->gpa = gpa;
25      num_credits = ncr;
26      num_courses = nc;
27      courses = new course[num_courses];
28      for(int i=0; i<num_courses; i++) {
29          courses[i].course_name = "Course";
30          courses[i].num_credits = 0;
31          courses[i].term = "n/a";
32          courses[i].grade = "n/a";
33      }
34  }
35  //accessors
36  int Student::get_id_num() const { return id_num; }
37  string Student::get_first_name() const { return first_name; }
38  string Student::get_last_name() const { return last_name; }
39  string Student::get_major() const { return major; }
40  float Student::get_gpa() const { return gpa; }
41  int Student::get_num_credits() const { return num_credits; }
42  int Student::get_num_courses() const { return num_courses; }
43  string Student::get_course_name(int index) const { return courses[index
    ].course_name; }
44  int Student::get_course_credit(int index) const { return courses[index].
    num_credits; }
45  string Student::get_course_term(int index) const { return courses[index
    ].term; }
46  string Student::get_course_grade(int index) const { return courses[index
    ].grade; }
47
48  //mutators
49  void Student::assign_id_num(int id) { id_num = id; }
50  void Student::set_first_name(string name) { first_name = name; }
```

```
51  void Student::set_last_name(string name) { last_name = name; }
52  void Student::set_major(string m) { major = m; }
53
54  void Student::calculate_gpa() {
55      float total = 0;
56      for(int i=0; i<num_courses; i++) {
57          if(courses[i].grade == "A")
58              total += (4.0 * courses[i].num_credits);
59          else if(courses[i].grade == "A-")
60              total += (3.7 * courses[i].num_credits);
61          else if(courses[i].grade == "B+")
62              total += (3.3 * courses[i].num_credits);
63          else if(courses[i].grade == "B")
64              total += (3.0 * courses[i].num_credits);
65          else if(courses[i].grade == "B-")
66              total += (2.7 * courses[i].num_credits);
67          else if(courses[i].grade == "C+")
68              total += (2.3 * courses[i].num_credits);
69          else if(courses[i].grade == "C")
70              total += (2.0 * courses[i].num_credits);
71          else if(courses[i].grade == "C-")
72              total += (1.7 * courses[i].num_credits);
73          else if(courses[i].grade == "D+")
74              total += (1.3 * courses[i].num_credits);
75          else if(courses[i].grade == "D")
76              total += (1.0 * courses[i].num_credits);
77          else
78              total += 0.0;
79      }
80      gpa = total/float(num_credits);
81  }
82
83  void Student::calc_num_credits() {
84      int total = 0;
85      for(int i=0; i<num_courses; i++) {
86          total += courses[i].num_credits;
87      }
88      num_credits = total;
89  }
90
91  void Student::set_num_courses(int nc) { num_courses = nc; }
92
93  void Student::set_course(string name, int credits, string term, string
    grade, int index) {
94      if(index < 0 || index > num_courses) {
95          cout << "Index error" << endl;
96      }
97      else {
98          courses[index].course_name = name;
99          courses[index].num_credits = credits;
100         courses[index].term = term;
101         courses[index].grade = grade;
102     }
103 }
104 /*****************************************************************
```

```
105  ** Function: add_course
106  ** Description: appends a new course to the old
107  ** Parameters: string name of course, int number of credits, string
     term that the
108  ** course was taken, string grade received in the course
109  ** Pre-Conditions: all parameters are initilized to safe values
110  ** Post-Conditions: num_courses updated by one, new course is appended
111  ********************************************************************/
112  void Student::add_course(string name, int credits, string term, string
     grade) {
113      num_courses++;
114      course* temp = new course[num_courses];
115      for(int i=0; i<num_courses-1; i++) {
116          temp[i] = courses[i];
117      }
118      delete [] courses;
119      courses = temp;
120      set_course(name, credits, term, grade, num_courses-1);
121  }
122  int Student::count = 0;
123
124  int Student::get_count() {
125      count++;
126      return count;
127  }
128  //copy constructor
129  Student::Student(const Student& copy) {
130      id_num = copy.id_num;
131      first_name = copy.first_name;
132      last_name = copy.last_name;
133      major = copy.major;
134      gpa = copy.gpa;
135      num_credits = copy.num_credits;
136      num_courses = copy.num_courses;
137      if(num_courses == 0)
138          courses = NULL;
139      else {
140          courses = new course[num_courses];
141          for(int i=0; i<num_courses; i++) {
142              courses[i] = copy.courses[i];
143          }
144      }
145  }
146
147
148
149
150
151
152
153
154
155
156
157
```

Called

1. Pass by value (106)
2. Return value (106)
3. Initialize obj (similar to 16)

```
158  //assignment operator overload
159  const Student& Student::operator=(const Student& copy) {
160      id_num = copy.id_num;
161      first_name = copy.first_name;
162      last_name = copy.last_name;
163      major = copy.major;
164      gpa = copy.gpa;
165      num_credits = copy.num_credits;
166      num_courses = copy.num_courses;
167      if(courses != NULL)
168          delete [] courses;
169      if(num_courses == 0)
170          courses = NULL;
171      else {
172          courses = new course[num_courses];
173          for(int i=0; i<num_courses; i++) {
174              courses[i] = copy.courses[i];
175          }
176      }
177      return *this;
178  }
179  //destructor
180  Student::~Student() {
181      delete [] courses;
182  }
183
```

Called when both the left and right operand are of the class type

Can't assume the left is ~~empty~~ empty

Called when a ~~variable~~ object 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);
        cout << " Term: " << a.get_course_term(i) << " Grade: " << a.
        get_course_grade(i) << endl;
    }
    cout << endl;
}
```

Nondefault, numstudent times
400 also called
Destructor called

```
51  void print(Student* s, int num_students) {
52      //Are any of the Big 3 used in this function? Why or why not?
53      // No because we are passing addresses
54      //
55      for(int i=0; i<num_students; i++) {
56          print_students(s[i]);
57      }
58  }
59
60  int editing_options() {
61      int choice = 0;
62      cout << "What would you like to change about the student?" << endl;
63      cout << "0. Nothing, this was a mistake" << endl;
64      cout << "1. Major" << endl;
65      cout << "2. Add a course" << endl;
66      cout << "Selection: ";
67      cin >> choice;
68      cin.ignore();
69      cin.clear();
70      return choice;
71  }
72
73  void change_major(Student& s) {
74      //Are any of the Big 3 used in this function? Why or why not?
75      // No because we are passing by reference
76      //
77       string m;
78       cout << "What major do you want to change to?";
79       getline(cin, m);
80       s.set_major(m);
81  }
82
83  void add_course(Student& s) {
84      //Are any of the Big 3 used in this function? Why or why not?
85      // No because we are passing by reference
86      //
87      string course_name, term, grade;
88      int credits = 0;
89      cout << "What is the name of the course? ";
90      getline(cin, course_name);
91      cout << "What term was it taken? ";
92      getline(cin, term);
93      cout << "What was the letter grade earned? ";
94      getline(cin, grade);
95      cout << "How many credits was the course? ";
96      cin >> credits;
97      s.add_course(course_name, credits, term, grade);
98  }
99
100
101
102
103
104
105
```

Student x = s; Student x; x = s;

```
106  Student edit_student(Student s) {
107      //Are any of the Big 3 used in this function? Why or why not?
108      // Yes copy constructor for pass by value, copy constructor for returning
109      //  the value, destructor when the function ends x2
110      int choice = editing_options();
111      if(choice == 1) {
112          change_major(s);
113      }
114      else if(choice == 2) {
115          add_course(s);
116      }
117      return s;
118  }
119
120  void remove_student(Student** s, int num_s, int index) {
121      //Are any of the Big 3 used in this function? Why or why not?
122      // Yes ~~dest~~ assignment operator for copying to temp, destructor when
123      //  delete is called
124      Student* temp = new Student[num_s-1];
125      for(int i=0; i<index; i++) {
126          temp[i] = (*s)[i];
127      }
128      for(int i=index; i<(num_s-1); i++) {
129          temp[i] = (*s)[i+1];
130      }
131      delete [] (*s);
132      (*s) = temp;
133  }
134
135  int main() {
136      //CREATE STUDENT ARRAY
137      Student* students = new Student[7]; //Which constructor is called?
         How many times? Default 7 times
138      //POP FROM FILE
139      pop_from_file(students, 7);
140      print(students, 7);
141
142      print_students(students[0]);
143      //EDIT STUDENT
144      students[0] = edit_student(students[0]); //Does one of the Big 3
         get called here? Which one? How many times? Assignment operator once
145      print_students(students[0]);
146      //REMOVE STUDENT
147      remove_student(&students, 7, 3);
148      print(students, 6);
149
150      delete [] students; //Is the destructor called? How many times?
151                          Yes 6 times
152      return 0;
153  }
154
```

instantiates temp obj

1. CC S PV
2. CC S RV
3. D S formal param
4. AOO
5. Return obj destructor

Total Calls

Default Constructor: 7+6 = 13

Non default Constructor: 7

Copy Constructor: 2

Assignment Operator Overload: 14

Destructor: 20