3)Report

Note: I used XXXXXX as a wild card in my report to not bother with repetition for similar functions

1)In main: university_array[3].evaluate_student(student3);

calls the function: void University::evaluate_student(const Student& stud); with the argument: student3.

The object here is student3 and is given the alias stud because of the refrence operator.

The function parameter is: const Student& stud

Evaluate_student function calls the following functions with no arguments as they are getter methods:

stud.get_GPA(), get_GPA(), stud.get_GRE(), get_GRE(), stud.get_TOEFL(), get_TOEFL(), get_bias().

Any function with stud.get_XXXXXX () is a getter method returning the XXXXXX attribute of stud. Here stud is student 3

The other getter methods return the weight values of the attributes of the University class. Here the object of the University is the last one created in the array since it is university_array[3] so it has the name McLaren's University. Those getters also don't have any arguments.

The above getters are used to calculate z which is controlled in an if statement in the evaluate_student function.

For student3 due to the value of the calculated z being less than 0 we go to the else part where std::cout ,stud.get_name(), get_name(), std::endl are called with no arguments stud.get_name utilizes student 3 as the object and get_name utilizes university_array[3] as its object.

after that stud.inc_app() is called with no arguments, that function adds 1 to the No_of_application of student3 since it is the object

after that, due to delete student4 in main, its destructor is called Student::~Student() the destructor calls std::cout, get_name, get_app,

std::endl. The get_XXXXXX functions work on student 4 object.get_name returns the name of student 4 in this case lily. And the get_app returns the number of applications she made which are 0 since she was never evaluated by a university.

After that due to the end of the file all objects get their default destructors because they went out of scope. But the objects are destructed in inverse order of when they were created. Thus;

student3 "Amy" goes out of scope so its destructor Student::~Student() is also called which in turn calls std::cout, get_name, get_app, std::endl. The get_XXXXXX functions work on student 3 object.get_name returns the name of student 3 in this case Amy. And the get_app returns the number of applications she made.

after that student2 "Ross" goes out of scope so its destructor Student::~Student() is also called which in turn calls std::cout, get_name, get_app, std::endl. The get_XXXXXX functions work on student 2 object.get_name returns the name of student 2 in this case Ross. And the get_app returns the number of applications he made.

after that student1 "Michael" goes out of scope so its destructor Student::~Student() is also called which in turn calls std::cout, get_name, get_app, std::endl. The get_XXXXXX functions work on student 1 object.get_name returns the name of student 1 in this case Michael. And the get_app returns the number of applications he made.

After that the university's array calls on the compilers default destructors to destruct the objects of university created since it also went out of scope. For the university class I didn't create a specific destructor, but the compiler will use its default one and the calls will be for the objects in inverse order, so the objects would be destroyed from university_array[3] to university_array[2] to university_array[1] to university_array[0]

Then the program ends.