Вначале создадим соответствующий класс для хранения информации о факультативе и определим в нем операции для сравнения двух факультативов по дате проведения и сравнения факультатива с конкретной датой.

После того, как мы считали все данные, отсортируем все факультативы по дате начала. Далее будем идти с конца этого массива и набирать себе факультативы.

Создадим массив в котором на i-ом месте будет хранится наибольшее количество кредитов, которое мы бы получили рассмотрев все факультативы после i-ого.

Первый (с конца) мы всегда может взять — берем его (и записываем его кредиты в наш специальный массив). Для всех следующих факультативов нам необходимо решить, брать их или нет:

Для i-го факультатива найдем первой после него, который не конфликтует с ним (для быстроты используем бинарный поиск).

Если такого не нашлось, тогда мы или берем только этот факультатив или берем все те, что уже выбрали. Таким образом в наш спец. массив запишем максимум из количества кредитов текущего факультатива и максимального количества из тех факультативов, что мы уже рассмотрели (а оно по определению лежит в i+1 ячейке массива).

Если нашелся не конфликтующий факультатив C, тогда рассуждаем таким же образом: мы или выбираем этот факультатив и тот набор факультативов, который дает наибольшее кол-во кредитов при рассмотрении всех факультативов после C или оставляем текущий. В массив опять же запишем максимум из этих двух вариантов.

Из свойства этого массива понятно, что ответом будет значение в 0 ячейке массива.

Выведем его.

Решение на C++:

```cpp
#include <algorithm>
#include <iostream>
#include <vector>

class Elective {
public:
    Elective () {}
    bool operator<(Elective b) {
        return start < b.start;
    }
    bool operator<(int b) {
        return start < b;
    }
    int start;
    int finish;
    int credits;
};
int main(int argc, const char *argv[])
{
    int n;
    std::cin >> n;
    if (n == 0) {
        std::cout << 0 << std::endl;
        return 0;
    }

    std::vector<Elective> intervals(n);
    for (size_t i = 0; i != n; ++i)
        std::cin >> intervals[i].start >> intervals[i].finish >>
intervals[i].credits;

    std::sort(intervals.begin(), intervals.end());

    std::vector<int> maxCredits(n);
    maxCredits[n-1] = intervals[n-1].credits;

    for (int i = n - 2; i >= 0; --i) {
        int current_finish = intervals[i].finish;
        auto next = std::lower_bound(intervals.begin(), intervals.end(),
current_finish); // находим подходящий отрезок

        if (next == intervals.end()) { // Не нашлось
            maxCredits[i] = std::max(maxCredits[i + 1], intervals[i].credits);
        } else {
            maxCredits[i] = std::max(maxCredits[i + 1], intervals[i].credits +
maxCredits[std::distance(intervals.begin(), next)]);
        }
    }
    std::cout << maxCredits[0] << std::endl;
    return 0;
}
```