4.1 Параметры компилятора и объяснение их действия. Примеры предупреждений компилятора, не являющихся ошибками с точки зрения языка. Пример неожиданного поведения компилятора при UB с переполнением int и включенном параметре -O. Объяснение этого поведения.

Параметры компилятора:

- **-Wall** Включает базовые предупреждения, в том числе предупреждения, отключенные по умолчанию. Например:
  * **-Wreorder** Предупреждает о том, что порядок инициализации членов класса не соответствует порядку их объявления. Поскольку компилятор может переупорядочить инициализацию этих членов, результат может быть неочевидным.
  * **-Wreturn-type** Предупреждает о том, что из функции не вернули заявленный результат
  * **-Wunused-variable** Предупреждает о том, что переменная не используется.
- **-Wextra** Включает дополнительные предупреждения, которых нет в -Wall
  * **-Wempty-body** Предупреждает о пустом теле условных выражений
  * **-Wunused-parameter** Предупреждает о неиспользуемом параметре функции. Возможно, про него забыли, и в этом случае функция может работать некорректно
  * **-Wmissing-field-initializers** Предупреждает о том, что отдельные члены структуры не были проинициализированы. Скорее всего это просто забыли сделать
- **-Werror** Сообщает компилятору, чтобы все предупреждения были превращены в ошибки, и при их наличии компиляция прерывалась.
- **-O1, -O2, -O3** Различные уровни оптимизации.
- **-O0** Отключение оптимизации.
- **-std=c++11, -std=c++14, -std=c++17, -std=c++2a** Подключение функционала C++11/14/17/20 соответственно.

Примеры предупреждений компилятора, не являющихся ошибками с точки зрения языка:

```
    // assignment in a conditional statement
    int x=3, y=4;
    if(x=y) {}

    // implicit type conversion
    int n = 10;
    for(size_t i = 0; i < n; ++i){}
```

Подробнее про все предупреждения можно почитать тут: ссылочка на хабр

Интересное UB с параметром -O2

Рассмотрим пример того, как неопределенное поведение в программе может приводить к неожиданным последствиям. Обратимся к коду ниже:

```
    for(int i = 0; i < 300; i++) {
        cout << i << " " << i * 12345678 << endl;
    }
```

Если скомпилировать этот код без параметра оптимизации, то мы получим, просто 300 чисел (при этом на 174 шаге происходит переполнение и выводятся отрицательные числа). Однако, если скомпилировать данный код с оптимизатором -O2, то цикл станет бесконечным. Почему? Компилятор считает, что ввод корректен (прогер не дурак), значит i не превосходит 173 (так как иначе происходит переполнение), поэтому оптимизатор заменяет условие `i < 300` на true и бинго, у нас бесконечный цикл.