38. Неопределенное поведение.
План ответа:
	1. особенности вычисления выражений с побочным эффектом;
	2. понятие «точка следования»;
	3. расположение «точек следования».
	4. More (этого в плане не было, но упомянуть явно нужно)

1. особенности вычисления выражений с побочным эффектом;

2. понятие «точка следования»;
	Компилятор вычисляет выражения. Выражения будут вычисляться почти в том же порядке, в котором они указаны в исходном коде: сверху вниз и слева направо.

	Точка следования — это точка в программе, в которой программист знает какие выражения (или подвыражения) уже вычислены, а какие выражения (или подвыражения) еще нет.

3. расположение «точек следования».
	Определены следующие точки следования:
		3.1 Между вычислением левого и правого операндов в операциях &&, || и "," .
		*p++ != 0 && *q++ != 0

		3.2 Между вычислением первого и второго или третьего операндов в тернарной операции.
		a = (*p++) ? (*p++) : 0;

		3.3 В конце полного выражения.
			a = b;
			if ()
			switch ()
			while ()
			do{} while()
			for ( x; y; z)
			return x

		3.4 Перед входом в вызываемую функцию.
		Порядок, в котором вычисляются аргументы не определен, но эта точка следования гарантирует, что все ее побочные эффекты проявятся на момент входа в функцию.

		3.5 В объявлении с инициализацией на момент завершения вычисления инициализирующего значения.
		int a = (1 + i++);


		Почему результата выражения «x[i] = i++ + 1;» не определен?
			Порядок вычисления выражений и подвыражений между точками следования не определен.
			В выражении «x[i] = i++ + 1;» есть единственная точка следования, которая находится в конце полного выражения.
			В выражении «x[i] = i++ + 1;» есть два обращения к переменной i. Множественный доступ и является источником проблемы.

		Почему спецификация языка оставляет открытым вопрос, в каком порядке компиляторы должны вычислять выражения между точками следования?
			Причина неопределенности порядка вычислений — простор для оптимизации.
4. MORE
	Примеры неопределенного поведения
		Использование неинициализированных переменных.
		Переполнение знаковых целых типов.
		Выход за границы массива.
		Использование «диких» указателей.

	Зачем нужно неопределенное поведение
		освободить разработчиков компиляторов от необходимости обнаруживать ошибки, которые трудно диагностировать.
		избежать предпочтения одной стратегии реализации другой.

отметить области языка для расширения языка (language extension).

Виды

Undefined behavior

Такое поведение возникает как следствие неправильно написанной программы.

Стандарт ничего не требует, может случиться все что угодно.

Unspecified behavior.

Стандарт предлагает несколько вариантов на выбор.

Компилятор может реализовать любой вариант.

При этом на вход компилятора подается корректная программа.

Например: все аргументы функции должны быть вычислены до вызова функции, но они могут быть вычислены в любом порядке.