Задания для домашней контрольной работы по дисциплине «Основы программной инженерии».

Задание 1.

Прикладные инструменты доказательства корректности программ.

Метода проверки корректности:

1) Верификация – установление соответствия между программой и ее спецификацией. Верификация проверяет, что программа разработана правильно

2) Валидация – установление соответствия между функциями программы и ее целевым назначением. Валидация проверяет, что разработана правильная программа.

Критерии корректности:

1) Функциональная оценка – определяется предметной областью

2) Конструктивная оценка – определяется общими свойствами программы.

Корректность программ:

- корректность текстов

- корректность модулей

- корректность данных

- корректность комплексов программ

Корректность текстов учитывает только структурную проверку корректности

Корректность модулей учитывает обе:

1) Структурная – правильность организации модуля с точки зрения технологий, применяемых в организации или языке

2) Функциональная – может проверяться тремя способами:

- детерминированная проверка – фиксированный набор данных, фиксированный набор результатов

- стохастическая – входные данные заданы случайными величинами с заданными законами распределения, проверяется соответствие результатов с требованиями (математическое ожидание, дисперсия) и корреляция между результатами

- динамическая – для заданного порядка входных данных проверяется заданный порядок формирования выходных данных

Корректность данных

1) Структурная – правильность организации структур данных в программе

2) Функциональная – проверяет диапазоны изменения значений данных, соответствие полей структур данных типам задаваемых значений

Корректность комплексов программ

1) Конструктивная – корректность взаимоотношения компонент комплекса

2) Функциональная – соответствие функций комплекса функциям спецификации.

Понятие верификации программы

Верификация – аналитическое исследование корректности программы по ее тексту на основе формально-логической системы установления соответствия между программой и ее спецификацией с помощью аппарата исчисления предикатов первого порядка. В конечном счете это вычисление истинности предиката от двух аргументов: программы и ее спецификации.

Для упрощения верификации разделяют две составляющих выполнения программы:

1) частичную корректность – удовлетворение входной и выходной спецификациям программы при условии ее успешно завершения

2) проверка завершенности программы – достижение выхода при заданных спецификацией входных данных

Основные задачи анализа корректности при верификации

1) частичная корректность при условии завершенности

2) частичная некорректность при условии завершенности

3) завершенность

4) незавершенность

5) тотальная корректность – частичная корректность и завершенность

6) некорректность – частичная некорректность или незавершенность

Задание 2.

Статическая и динамическая типизация.

Статическая и динамическая типизация¶

При таких операциях, как сложение целых чисел, машина не способна проверить принадлежность к типу переданных ей операндов. Для неё это просто последовательность битов. Отсутствие механизма проверки типов делает низкоуровневые программы ненадежными.

Контроль типов устанавливает принадлежность каждого операнда к конкретному типу данных, а также проверяет его на совместимость с оператором, участвующим в текущей операции. Любое несоответствие приводит к возникновению ошибки.

Этап на котором происходит контроль типов, определяет вид типизации. Контроль типов который происходит на этапе компиляции, называется статическим контролем типов или статической типизацией. Контроль типов, который происходит на этапе выполнения программы, называется динамическим контролем типов или динамической типизацией.

Статическая типизация¶

Статическая типизация (Static Typing) обуславливается тем, что нуждающаяся в аннотации типа конструкция связывается с типом данных в момент объявления на этапе компиляции. При этом связь с типом данных остается неизменна.

Другими словами, до того момента, как программа будет запущена, компилятор осуществляет проверку совместимости типов участвующих в различных операциях. Сущность, представляющая любую конструкцию нуждающуюся в аннотации типа, связывается с типом данных при объявлении. При этом связь с типом данных в дальнейшем не может быть изменена.

За счет того, что большая часть проверок происходит на этапе компиляции, программа обладает большей производительностью из-за отсутствия проверок во время выполнения.

Языки со статической проверкой типов значительно повышают семантическую привлекательность кода делая его более читабельным.

Благодаря статическому контролю типов, редактор кода способен на основе синтаксического анализа выводить вспомогательную информацию, ускоряющую разработку.

К статически типизированным относятся такие языки, как C#, Java.

Например в Java: float f = 0.5;

Динамическая Типизация¶

Динамическая типизация (Dynamic Typing) обусловлена тем, что конструкция нуждающаяся в аннотации типа, связывается с типом данных на этапе выполнения программы в момент присвоения значения. При этом связь с типом данных может быть изменена.

В динамической среде выполнения тип данных может быть определен только на основании вычисленного результата выражения. При этом операция присвоения значения может изменить тип данных сущности с которым она была связана ранее.

Динамический контроль типов проверяет операнды на совместимость типов и совместимость с оператором в каждой отдельной операции. Проверка типов данных во время выполнения программы отрицательно сказывается на её производительности. Поэтому языки с динамическим контролем типов проигрывают в производительности языкам со статическим контролем типов у которых совместимость проверяется на этапе компиляции.

По причине выявления несовместимости типов только во время выполнения программы ошибки выявляются менее эффективно. Программа не узнает о скрывающейся в ней ошибке пока не дойдет очередь выполнения проблемного участка кода.

К тому же, редакторы кода, даже современные, испытывают трудности при синтаксическом анализе кода необходимого для вывода информации облегчающей разработку.

К динамически типизированным языкам относятся такие языки как PHP, Python, JavaScript.

Например в Python: f = 0.5