МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Лабораторная работа №2
«Тестирование ПО методом белого и чёрного ящика»
по дисциплине
«Методы тестирования программного обеспечения»

Студент,
группы 5130201/20102 _____ Тищенко А. А.

Преподаватель _____ Курочкин М. А.

«_____»_____ 2025г.

Санкт-Петербург, 2025

Содержание

Введение

Одним из ключевых этапов в современной разработке программного обеспечения является тестирование. Существует несколько видов тестирования: модульное, интеграционное, функциональное, системное и приемочное. В данной работе будет рассмотрено модульное тестирование.

Модульное или unit-тестирование является основным видом тестирования, с которого практически всегда начинается проверка корректности работы программы. Этот вид тестирования включает проверку отдельных блоков ПО: классов, модулей и функций. Цель модульного тестирования – выявить ошибки, то есть несоответствия поведения модуля его спецификации.

Существует два основных подхода к unit-тестированию: методы «чёрного» и «белого ящика».

В данной работе будут рассмотрены оба метода. Также данные приёмы будут применены для тестирования двух программ:

- вычисление факториала числа;
- возведение числа в степень.

Каждая из программ будет протестирована обоими методами с целью сравнения подходов.

1 Постановка задачи

Цель работы: провести тестирование двух программ методами «черного ящика» и «белого ящика». Для достижения цели, были выделены следующие задачи:

- Изучить методы модульного тестирования, в частности методы «белого ящика» и «черного ящика».
- Разработать тесты для двух программ, и провести их тестирование используя оба метода на каждой программе.
- Проанализировать результаты тестирования.

2 Описание методов тестирования

2.1 Метод черного ящика

Метод черного ящика или тестирование управляемое данными, тестирование управляемое входом и выходом – это метод тестирования, в котором программа рассматривается как «черный ящик», внутреннее поведение и структура которого не имеют никакого значения. Вместо этого все внимание фокусируется на выяснении обстоятельств, при которых поведение программы не соответствует спецификации.

Единственной предоставляемой информацией о программе является ее спецификация, полностью описывающая поведение программы при различных входных данных.

При таком подходе тестовые данные выбираются исключительно на основе спецификаций требований (без привлечения каких-либо знаний о внутренней структуре программы).

Чтобы в рамках данного метода обнаружить все ошибки в программе, необходимо выполнить так называемое исчерпывающее входное тестирование (exhaustive input testing), т.е. перебрать все возможные комбинации входных данных

2.1.1 Эквивалентное разбиение

Класс эквивалентности – конечный набор входных данных, позволяющий допускать, что тестирования представительного значения данного класса эквивалентно тестированию любого другого значения принадлежащего тому же классу.

Разбитие области входных данных на конечное число связано с проблемой, того, что для осуществления исчерпывающего тестирования на всех входных наборах данных, неосуществимо, а также преследует следующие цели:

- уменьшение более чем на единицу числа других тестов, которые должны быть разработаны для достижения поставленной цели «обеспечить приемлемое тестирование»;
- покрытие значительной части других возможных тестов, т.е. предоставление некой информации относительно наличия или отсутствия ошибок в ситуациях, не охватываемых данным конкретным набором входных значений.

Определение классов эквивалентности

Определение классов эквивалентности сводится к последовательному рассмотрению каждого из входных условий (обычно это предложение или фраза, приведенные в спецификации) и разбиению его на две или несколько групп.

Определяется два типа классов:

- допустимые классы эквивалентности – допустимые входные данные программы;

- недопустимые классы эквивалентности – все остальные возможные состояния условий (т.е. недопустимые входные значения).

Если есть вероятность, что не для всех значений из одного класса эквивалентности будет одинаковый результат, то следует разбить класс на более мелкие подклассы.

Определенные классы эквивалентности используются для составления тестов.

Разработка тестов

Определение тестов по построенным классам эквивалентности состоит из трех этапов:

1. назначить каждому классу эквивалентности уникальный номер;
2. записать новые тесты, охватывающие как можно большее количество оставшихся неохваченными допустимых классов эквивалентности, пока не будут покрыты все допустимые классы;
3. записать новые тесты, каждый из которых охватывает один и только один из оставшихся неохваченными недопустимых классов эквивалентности, пока не будут покрыты все недопустимые классы.

2.1.2 Анализ граничных значений

Граничные условия – это ситуации, возникающие в области граничных значений входных эквивалентности. Анализ граничных значений отличается от методики разбиения на классы эквивалентности в следующем отношении: вместо того чтобы выбирать любой элемент класса эквивалентности в качестве представителя всего класса, анализ граничных значений требует выбирать такой элемент или элементы, которые обеспечивают тестирование каждой границы класса.

2.1.3 Причинно-следственные диаграммы

Метод причинно-следственных диаграмм или метод диаграмм Исикавы – метод, позволяющий выбирать высокорезультативные тесты. Его дополнительным преимуществом является то, что он позволяет обнаруживать неполноту и неоднозначность исходных спецификаций.

Метод позволяет решить проблему того, что метод анализа граничных значений и разбиения данных на классы эквивалентности не исследует комбинации входных условий.

Причинно-следственная диаграмма представляет собой формальный язык, на который транслируется спецификация, написанная на естественном языке.

Для построения тестов используется процесс, включающий несколько этапов:

1. Спецификация разбивается на части, с которыми легче работать.

2. В спецификации определяются причины и следствия.

   Причина – это отдельное входное условие или класс эквивалентности входных условий. Следствие – это выходное условие или преобразование системы Причины и следствия определяются путем последовательного (слово за словом) чтения спецификации и подчеркивания тех слов или фраз, которые описывают причины и следствия. Каждой причине и каждому следствию присваивается уникальный номер.

3. Семантическое содержание спецификации анализируется и преобразуется в булев граф, связывающий причины и следствия. Полу- ченный граф называется причинно-следственной диаграммой.

4. Диаграмма снабжается примечаниями, задающими ограничения и описывающими комбинации причин и (или) следствий, реализация которых невозможна из-за синтаксических или внешних ограничений. Нотация отображений ограничений представлена на Рис. 1.

5. Путем методичного прослеживания состояний условий диаграмма преобразуется в таблицу решений с ограниченными входами (limited-entry decision table). Каждый столбец таблицы решений соответствует тесту.

6. Столбцы таблицы решений преобразуются в тесты.

Базовая нотация причинно-следственных диаграмм представлена на Рис. 1. Каждый узел диаграммы может находиться в двух состояниях: 0 или 1, где 0 представляет состояние «отсутствует», а 1 – «присутствует».

- Функция тождество устанавливает, что если a равно 1, то и b равно 1; в противном случае b равно 0.
- Функция not устанавливает, что если a равно 1, то b равно 0; в противном случае b равно 1.
- Функция or устанавливает, что если a, или b, или c равно 1, то d равно 1; в противном случае d равно 0.
- Функция and устанавливает, что если и a, и b равны 1, то c равно 1; в противном случае c равно 0.

Также для установления связи между входными условиями могут использоваться следующие ограничения: Существуют следующие ограничения.

- Ограничение требует, чтобы всегда выполнялось условие, в соответствии с которым только или только b может быть равно 1 (и b не могут быть равны 1 одновременно, но обе величины могут быть равны 0).
- Ограничение I требует, чтобы по крайней мере одна из величин, a, b или , была равна 1 (, b и не могут быть равны 0 одновременно).
- Ограничение требует, чтобы одна и только одна из величин, a или b,была равна 1.

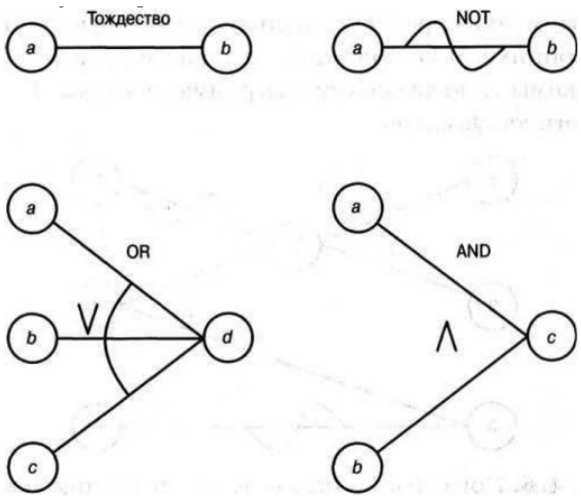

Рис. 1. Базовая нотация, используемая в причинно-следственных диаграммах.

- Ограничение R требует, чтобы a было равно 1, только если b равно 1 (т.е. не может быть равно 1, если b равно 0). На Рис. 2 представлены символы ограничений.

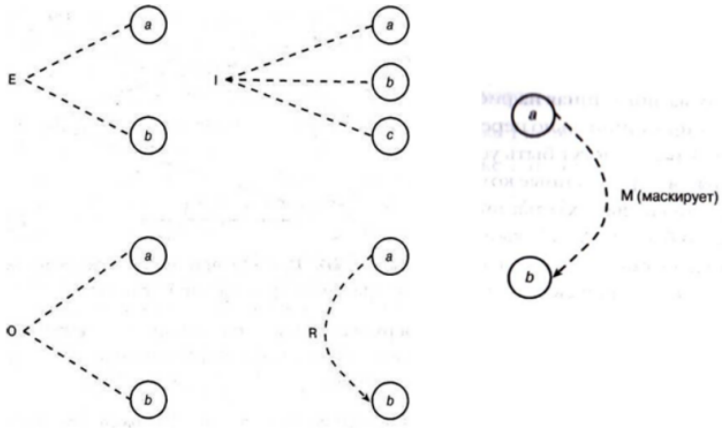

Рис. 2. Базовая нотация ограничений, используемая в причинно-следственных диаграммах.

При составлении причинно-следственной диаграммы возможно введение ограничения не использующегося в базовой нотации и характерного только для рассматриваемого случая. В таком случае, вводимое ограничение всегда сопровождается пояснительной подписью.

2.2 Метод белого ящика

Метод белого ящика – метод тестирования основанный на анализе внутренней структуры кода. Для данного тестирования используется спецификация программы, описывающая её поведение и блоксхема, по которой тестировщик может осуществлять тестирование на уровне отдельных модулей, функций и логики программы. Тестирование методом белого ящика имеет и другое название, отражающее суть

данной методологии: тестирование с управляемой логикой программы (logic-driving testing). При использовании данной методологии, тестировщик подбирает тестовые данные путём анализа логики программы с учетом её спецификации. Исчерпывающее тестирование методом «белого ящика» практически невозможно в следствии необходимости перебирать огромное количество данных. Поэтому для упрощения тестирования применяются следующие эвристические методы:

2.2.1 Покрытие операторов

Критерий покрытия операторов/инструкций заключается в построении тестов, в которых каждая инструкция выполнялась как минимум один раз. Является необходимым но недостаточным критерием для тестирования программы методом «белого ящика»

2.2.2 Покрытие решений

Критерий покрытие решений (decision coverage) или покрытие ветвлений заключается в построении тестов таким образом, чтобы каждое условие в программе хотя бы раз принимало как значение true (истина), так и false (ложь). Иными словами, каждая логическая ветвь каждой инструкции ветвления в программе должна быть выполнена хотя бы один раз. Обычно покрытие решений удовлетворяет критерию покрытия операторов/инструкций. Однако это правило не действует по крайней мере в трех перечисленных ниже случаях.

1. Программы, не содержащие точек ветвления.
2. Программы или подпрограммы (методы), имеющие несколько точек входа. В таком случае некоторые инструкции будут выполняться лишь тогда, когда выполнение программы начинается с определенной точки входа.
3. Инструкции в блоках ON. Выполнение всех ветвей не обязательно приведет к выполнению всех блоков ON.

2.2.3 Покрытие решений и условий

Согласно этому критерию набор тестов является достаточно полным, если выполняются следующие требования:

- каждое условие в решении принимает каждое возможное значение по крайней мере один раз;
- каждый возможный исход решения проверяется по крайней мере один раз;
- каждой точке входа управление передается по крайней мере один раз.

Однако, несмотря на кажущуюся возможность охвата им всех возможных исходов решений для всех условий, это зачастую не обеспечивается из-за маскирования одних условий

2.2.4 Комбинаторное покрытие условий

Комбинаторное покрытие условий (multiple-condition covering) – критерий, требующий создания такого количества тестов, при котором каждая возможная комбинация результатов вычисления условий в каждом решении и каждая точка входа проверяются по крайней мере один раз.

3 Тестирование программы №1

3.1 Формальное описание программы

Название: «Вычисление факториала числа».

Дано:

- N — целое положительное число.

Требуется: Вычислить факториал числа N и вывести результат на экран.

Ограничения:

- $1 \leq N \leq 65$;
- N — целое.

Спецификация

Входные данные	**Выходные данные**	**Реакция программы**
$N = -5$	"Ошибка! Введите положительное целое число:"	Вывод на экран сообщения: "Ошибка! Введите положительное целое число:". Ожидание корректного ввода N.
$N = 5.7$	"Ошибка! Введите целое число, а не дробное:"	Вывод на экран сообщения: "Ошибка! Введите целое число, а не дробное:". Ожидание корректного ввода N.
$N =$ "пять"	"Ошибка! Введите целое число, а не строку:"	Вывод на экран сообщения: "Ошибка! Введите целое число, а не строку:". Ожидание корректного ввода N.
$N = 70$	"Ошибка! Введите целое положительное число, не более 65:"	Вывод на экран сообщения: "Ошибка! Введите целое положительное число, не более 65:". Ожидание корректного ввода N.
$N = 5$	120	Вывод на экран значения факториала для заданного числа N. Завершение программы.
$N = 0$	1	Вывод на экран значения факториала для заданного числа N. Завершение программы.
$N = 1$	1	Вывод на экран значения факториала для заданного числа N. Завершение программы.

Блок-схема

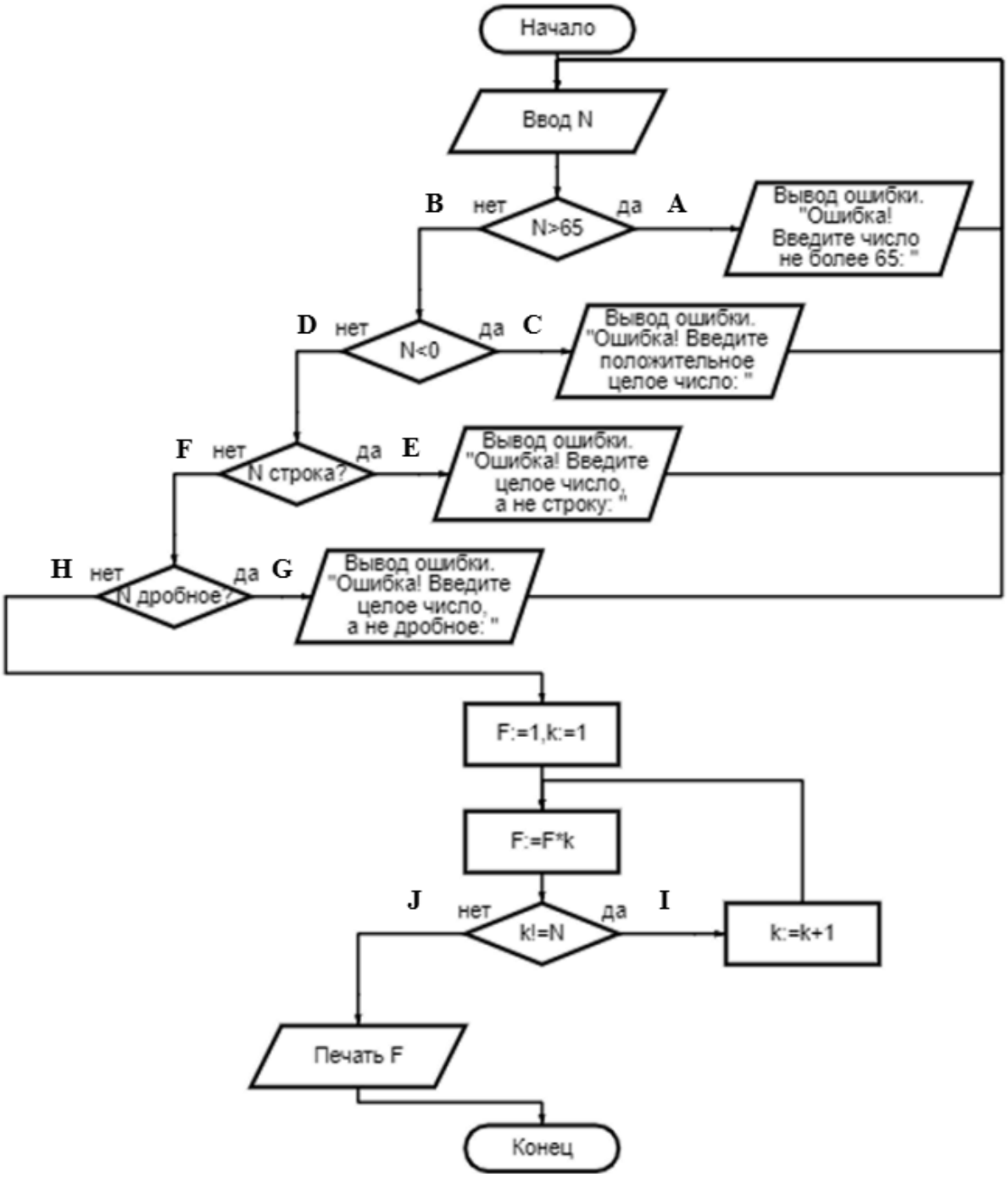

Рис. 3. Блок схема программы №1.

3.2 Тестирование методом «белого ящика»

Алгоритм составления тестов методом «белого» ящика предполагает обход всех возможных путей в теле программы и проверку выполнения каждого оператора не менее одного раза. Для этого на блок-схеме программы, которая изображена на Рис. 3, все возможные пути обозначены символами латинского алфавита от A до I.

Условия в ветвлениях программы:

1. $N > 65$;
2. $N < 0$;
3. N строка?
4. N дробное?
5. $k \neq N$

3.2.1 Покрытие операторов

Критерием покрытия является выполнение каждого оператора программы хотя бы один раз. Это необходимое, но не достаточное условие для приемлемого тестирования по принципу белого ящика.

Для покрытия всех операторов был составлен набор тестов из 6 тестов (Рис. 4).

№	Входные данные	Ожидаемый результат	Фактический результат	Путь	Условия					Статус
					1	2	3	4	5	
1	70	Ошибка! Введите целое положительное число, не более 65	Ошибка! Введите целое положительное число, не более 65	A	И	-	-	-	-	Пройден
2	-5	Ошибка! Введите целое положительное число	Ошибка! Введите целое положительное число	B->C	Л	И	-	-	-	Пройден
3	строка	Ошибка! Введите целое число, а не строку	Программа завершилась с кодом 1	-	-	-	-	-	-	Не пройден
4	5.7	Ошибка! Введите целое число, а не дробное	Ошибка! Введите целое число, а не дробное	B->D->F ->G	Л	Л	Л	И	-	Пройден
5	5	120	120	B->D->F ->G->H->I	Л	Л	Л	Л	И	Пройден
6	1	1	1	B->D->F ->G->H->J	Л	Л	Л	Л	Л	Пройден

Рис. 4. Набор тестов для покрытия операторов программы.

При тестировании покрытия операторов был составлен тест (№3), который программа не проходит. Программа не может пройти тест №3, так как была допущена ошибка при составлении блок-схемы программы. При вводе строки программа завершается при попытке сравнить строку с числом 65, хотя по спецификации должна выводить строку «Ошибка! Введите целое число, а не строку».

3.2.2 Покрытие решений

В соответствии с этим критерием необходимо составить такой набор тестов, при котором каждое условие в программе примет как истинное, так и ложное значения. Таким образом, к тестам, составленным для метода покрытия операторов, необходимо добавить тесты, которые будут проверять все возможные переходы.

Тесты, покрывающие все решения программы представлены на Рис. 6.

№	Входные данные	Ожидаемый результат	Фактический результат	Путь	Условия					Статус
					1	2	3	4	5	
1	70	Ошибка! Введите целое положительное число, не более 65	Ошибка! Введите целое положительное число, не более 65	A	И	-	-	-	-	Пройден
2	-5	Ошибка! Введите целое положительное число	Ошибка! Введите целое положительное число	B->C	Л	И	-	-	-	Пройден
3	строка	Ошибка! Введите целое число, а не строку	Программа завершилась с кодом 1	-	-	-	-	-	-	Не пройден
4	5.7	Ошибка! Введите целое число, а не дробное	Ошибка! Введите целое число, а не дробное	B->D->F ->G	Л	Л	Л	И	-	Пройден
5	5	120	120	B->D->F ->G->H->I	Л	Л	Л	Л	И	Пройден
6	1	1	1	B->D->F ->G->H->J	Л	Л	Л	Л	Л	Пройден

Рис. 5. Набор тестов для покрытия решений программы.

Покрыть истинную ветку условия 3 (путь B->D->E) невозможно из-за экстренного завершения программы при вводе строкового значения N.

3.2.3 Покрытие условий

В соответствии с этим критерием количество тестов должно быть таким, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз.

Тесты, покрывающие все условия программы представлены на Рис. 6.

№	Входные данные	Ожидаемый результат	Фактический результат	Путь	Условия					Статус
					1	2	3	4	5	
1	70	Ошибка! Введите целое положительное число, не более 65	Ошибка! Введите целое положительное число, не более 65	A	И	-	-	-	-	Пройден
2	-5	Ошибка! Введите целое положительное число	Ошибка! Введите целое положительное число	B->C	Л	И	-	-	-	Пройден
3	строка	Ошибка! Введите целое число, а не строку	Программа завершилась с кодом 1	-	-	-	-	-	-	Не пройден
4	5.7	Ошибка! Введите целое число, а не дробное	Ошибка! Введите целое число, а не дробное	B->D->F ->G	Л	Л	Л	И	-	Пройден
5	5	120	120	B->D->F ->G->H->I	Л	Л	Л	Л	И	Пройден
6	1	1	1	B->D->F ->G->H->J	Л	Л	Л	Л	Л	Пройден

Рис. 6. Набор тестов для покрытия решений программы.

Покрыть истинную ветку условия 3 (путь B->D->E) невозможно из-за экстренного завершения программы при вводе строкового значения N.

3.2.4 Покрытие решений и условий

Согласно этому критерию набор тестов является достаточно полным, если удовлетворяются следующие требования: каждое условие в решении принимает каждое возможное значение по крайней мере один раз, каждый возможный исход решения проверяется по крайней мере один раз и каждой точке входа управление передается по крайней мере один раз.

Тесты, написанные ранее, обеспечивают покрытие решений и условий (Рис. 7).

№	Входные данные	Ожидаемый результат	Фактический результат	Путь	Условия					Статус
					1	2	3	4	5	
1	70	Ошибка! Введите целое положительное число, не более 65	Ошибка! Введите целое положительное число, не более 65	A	И	-	-	-	-	Пройден
2	-5	Ошибка! Введите целое положительное число	Ошибка! Введите целое положительное число	B->C	Л	И	-	-	-	Пройден
3	строка	Ошибка! Введите целое число, а не строку	Программа завершилась с кодом 1	-	-	-	-	-	-	Не пройден
4	5.7	Ошибка! Введите целое число, а не дробное	Ошибка! Введите целое число, а не дробное	B->D->F ->G	Л	Л	Л	И	-	Пройден
5	5	120	120	B->D->F ->G->H->I	Л	Л	Л	Л	И	Пройден
6	1	1	1	B->D->F ->G->H->J	Л	Л	Л	Л	Л	Пройден

Рис. 7. Набор тестов для покрытия решений программы.

Покрыть истинную ветку условия 3 (путь B->D->E) невозможно из-за экстренного завершения программы при вводе строкового значения N.

3.2.5 Комбинаторное покрытие условий

Этот критерий требует создания такого набора тестов, при котором каждая возможная комбинация результатов вычисления условий в каждом решении и каждая точка входа проверяются по крайней мере один раз.

Совокупность всех ранее написанных тестов дает покрытие условий и решений (Рис. 8).

№	Входные данные	Ожидаемый результат	Фактический результат	Путь	Условия					Статус
					1	2	3	4	5	
1	70	Ошибка! Введите целое положительное число, не более 65	Ошибка! Введите целое положительное число, не более 65	A	И	-	-	-	-	Пройден
2	-5	Ошибка! Введите целое положительное число	Ошибка! Введите целое положительное число	B->C	Л	И	-	-	-	Пройден
3	строка	Ошибка! Введите целое число, а не строку	Программа завершилась с кодом 1	-	-	-	-	-	-	Не пройден
4	5.7	Ошибка! Введите целое число, а не дробное	Ошибка! Введите целое число, а не дробное	B->D->F ->G	Л	Л	Л	И	-	Пройден
5	5	120	120	B->D->F ->G->H->I	Л	Л	Л	Л	И	Пройден
6	1	1	1	B->D->F ->G->H->J	Л	Л	Л	Л	Л	Пройден

Рис. 8. Набор тестов для покрытия решений программы.

Покрыть истинную ветку условия 3 (путь B->D->E) невозможно из-за экстренного завершения программы при вводе строкового значения N.

3.2.6 Результаты тестирования методом «белого ящика»

В результате тестирования методом «белого ящика» было составлено 6 тестов, из которых были пройдены 5 и не пройден 1. При строковом значении N программа должна вывести сообщение "Ошибка! Введите целое число, а не строку:" и ожидать повторного ввода значения N, однако она экстренно завершается с кодом -1 на первом блоке ветвления ($N > 65$) при попытке сравнить строку с числом, что означает некорректно составленную программу и спецификацию.

3.3 Тестирование методом «чёрного ящика»

3.3.1 Разбиение на классы эквивалентности

Составлено разбиение на классы эквивалентности исходя из ограничений для программы, представленное на Рис. 9. Тесты для допустимых классов эквивалентности представлены на Рис. 10 и тесты для недопустимых классов эквивалентности — на Рис. 11.

Входное условие	Допустимые классы эквивалентности	Недопустимые классы эквивалентности
N	(1) N - целое положительное число	(2) N - не целое
		(3) N - отрицательное
		(4) N - не число
	(5) 1 <= N <= 65	(6) N < 1
		(7) N > 65

Рис. 9. Разбиение на классы эквивалентности.

№	Класс эквивалентности	Входные данные	Ожидаемый результат	Фактический результат	Статус
1	1, 5	N = 5	120	120	Пройден

Рис. 10. Тесты для допустимых классов эквивалентности.

№	Класс эквивалентности	Входные данные	Ожидаемый результат	Фактический результат	Статус
1	2	N = 5.7	Ошибка! Введите целое число, а не дробное	Ошибка! Введите целое число, а не дробное	Пройден
2	3, 6	N = -5	Ошибка! Введите целое положительное число	Ошибка! Введите целое положительное число	Пройден
3	4	N = строка	Ошибка! Введите целое число, а не строку	Программа завершилась с кодом 1	Не пройден
4	7	N = 70	Ошибка! Введите целое положительное число, не более 65	Ошибка! Введите целое положительное число, не более 65	Пройден

Рис. 11. Тесты для недопустимых классов эквивалентности.

При тестировании недопустимых классов эквивалентности был составлен тест, который программа не проходит. Программа не может пройти тест №3, так как была допущена ошибка при составлении блок-схемы программы. При вводе строки программа завершается при попытке сравнить строку с числом 65, хотя по спецификации должна выводить строку «Ошибка! Введите целое число, а не строку».

3.3.2 Анализ граничных условий

В программе можно выделить следующие граничные условия:

- $N \geq 0$;
- $N \leq 65$.

Для каждой из границ определим тесты, соответствующие:

- граничному целому числу (верхнему/нижнему);
- целому числу, выходящему за границу (верхнюю/нижнюю) на единицу;
- дробному числу, на 0.001 выходящему за границу (верхнюю/нижнюю).

Составленные тесты представлены на Рис. 12.

№	Входные данные	Ожидаемый результат	Фактический результат	Статус
1	N = 0	1	Ошибка! Программа зависла в бесконечном цикле	Не пройден
2	N = 65	65!	65!	Пройден
3	N = -1	Ошибка! Введите целое положительное число	Ошибка! Введите целое положительное число	Пройден
4	N = 66	Ошибка! Введите целое положительное число, не более 65	Ошибка! Введите целое положительное число, не более 65	Пройден
5	N = -0.001	Ошибка! Введите целое положительное число	Ошибка! Введите целое положительное число	Пройден
6	N = 65,001	Ошибка! Введите целое положительное число	Ошибка! Введите целое положительное число	Пройден

Рис. 12. Тесты граничных условий.

При тестировании граничных условий был составлен тест, который программа не проходит. Программа не может пройти тест №1 (Рис. 12), так как была допущена

ошибка при составлении блок-схемы программы. При вводе числа 0 программа зависает в бесконечном цикле (из-за условия «k != N»), хотя по спецификации должна выводить число 1.

3.3.3 Причинно-следственная диаграмма

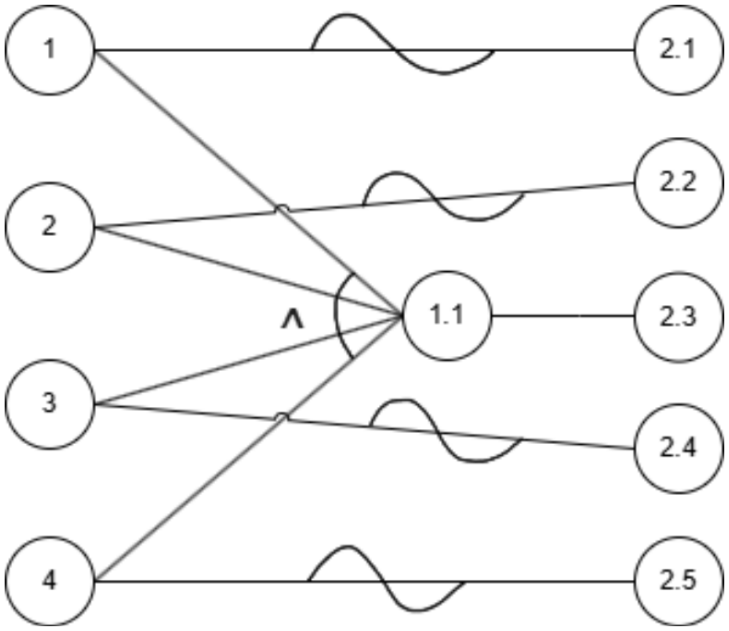

Рис. 13. Причинно следственная диаграмма.

Причины:

1. N – число;
2. N – целое;
3. $N \geq 0$;
4. $N \leq 65$.

Промежуточные причины:

1.1 N – целое и $0 \leq N \leq 65$.

Следствия:

2.1 Программа выводит сообщение об ошибке (N – строка) и заново запрашивает число;

2.2 Программа выводит сообщение об ошибке (N – дробное) и заново запрашивает число;

2.3 Программа выводит значение факториала для числа N;

2.4 Программа выводит сообщение об (ошибке $N < 0$) и заново запрашивает число;

2.5 Программа выводит сообщение об (ошибке $N > 65$) и заново запрашивает число.

На Рис. 13 представлена причинно-следственная диаграмма.

Таблица решений для диаграммы представлена на Рис. 14.

№	N - число	N - целое	N >= 0	N <= 65	Ошибка: N - строка	Ошибка: N - дробное	Ошибка: N < 0	Ошибка: N > 65	Вывод факториала
1	0	-	-	-	1	0	0	0	0
2	1	0	-	-	0	1	0	0	0
3	1	1	0	-	0	0	1	0	0
4	1	1	1	0	0	0	0	1	0
5	1	1	1	1	0	0	0	0	1

Рис. 14. Таблица решений.

Все тесты для таблицы решений уже были покрыты ранее при рассмотрении классов эквивалентности и граничных условий.

3.3.4 Результаты тестирования методом «чёрного ящика»

В результате тестирования методом чёрного ящика было составлено 11 тестов, из которых не пройдено 2. Ошибки, из-за которых тесты не были пройдены, связаны с некорректной проверкой входных значений и неверно составленной спецификацией: программа экстренно завершается при вводе вместо числа N строки, а также входит в бесконечный цикл при $N = 0$, что не соответствует поведению, описанному в спецификакции.

4 Тестирование программы №2

4.1 Формальное описание программы

Название: «Алгоритм быстрого возведения в степень».

Дано:

- n — целое положительное число.
- k — целое положительное число.

Требуется: Возвести число n в степень k и вывести результат на экран.

Ограничения:

- n — целое.
- k — целое.
- $1 \leq n \leq 15$;
- $1 \leq k \leq 15$;

Спецификация

Входные данные	**Выходные данные**	**Реакция программы**
$n = -5, k = 10$	”Ошибка! Введите числа от 1 до 15.”	Вывод на экран сообщения: ”Ошибка! Введите числа от 1 до 15.”. Ожидание корректного ввода n и k.
$n = 5, k = -10$	”Ошибка! Введите числа от 1 до 15.”	Вывод на экран сообщения: ”Ошибка! Введите числа от 1 до 15.”. Ожидание корректного ввода n и k.
$n = 25, k = 10$	”Ошибка! Введите числа от 1 до 15.”	Вывод на экран сообщения: ”Ошибка! Введите числа от 1 до 15.”. Ожидание корректного ввода n и k.
$n = 5.7, k = 10$	”Ошибка! n должно быть целым числом.”	Вывод на экран сообщения: ”Ошибка! k должно быть целым числом.”. Ожидание корректного ввода n и k.
$n =$ ”строка” $, k = 10$	”Ошибка! n должно быть целым числом.”	Вывод на экран сообщения: ”Ошибка! k должно быть целым числом.”. Ожидание корректного ввода n и k.
$n = 2, k = 10$	1024	Вывод на экран значения 2^10. Завершение программы.

Блок-схема

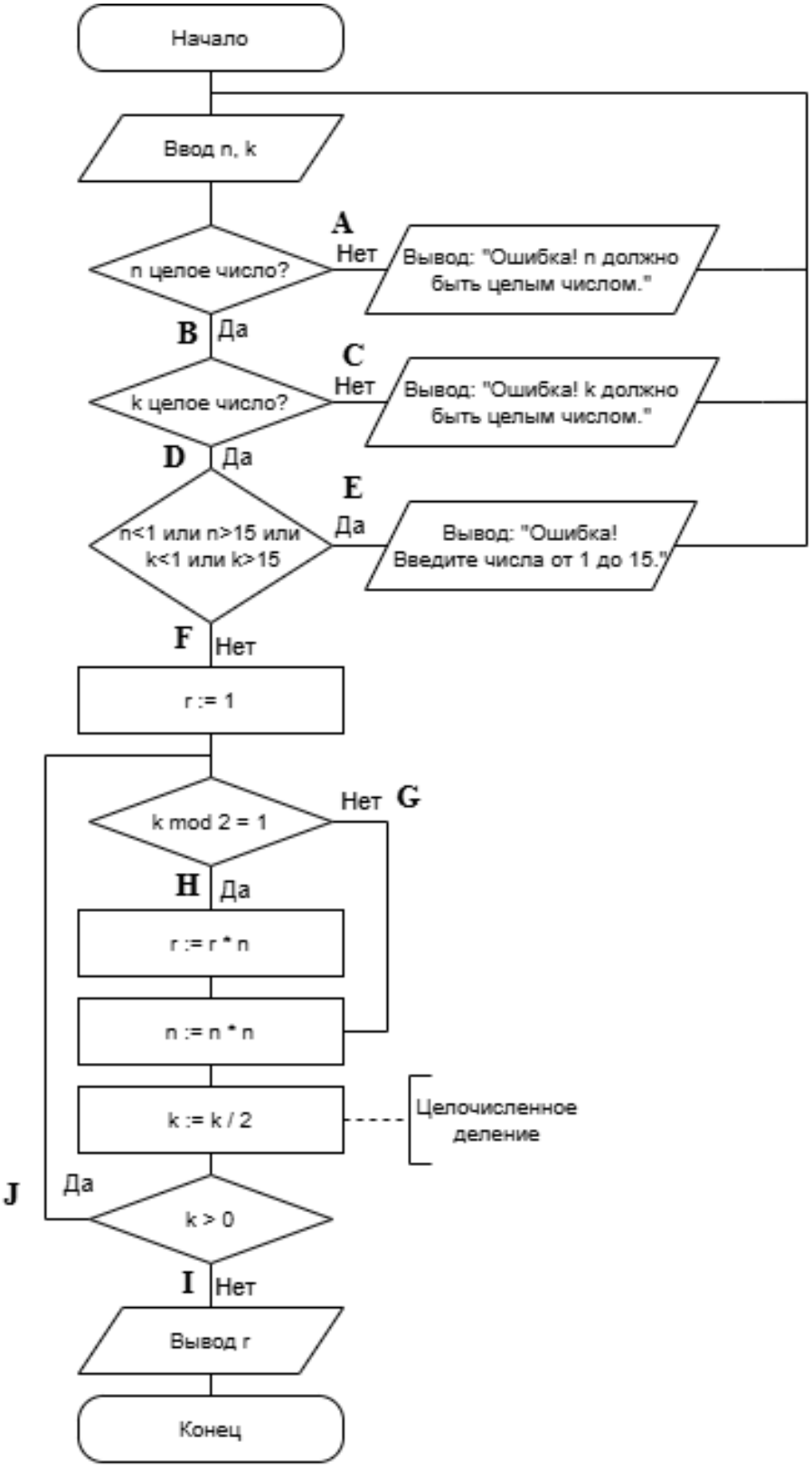

Рис. 15. Блок схема программы №2.

4.2 Тестирование методом «белого ящика»

Алгоритм составления тестов методом «белого» ящика предполагает обход всех возможных путей в теле программы и проверку выполнения каждого оператора не менее одного раза. Для этого на блок-схеме программы, которая изображена на Рис. 15, все возможные пути обозначены символами латинского алфавита от A до I.

Условия в ветвлениях программы:

1. n целое число?
2. k целое число?
3. $n < 1$;
4. $n > 15$;
5. $k < 1$;
6. $k > 15$;
7. $k \mod 2 = 1$;
8. $k > 0$.

4.2.1 Покрытие операторов

Критерием покрытия является выполнение каждого оператора программы хотя бы один раз. Это необходимое, но не достаточное условие для приемлемого тестирования по принципу белого ящика.

Для покрытия всех операторов был составлен набор тестов из 4 тестов (Рис. 16).

№	Входные данные	Ожидаемый результат	Фактический результат	Путь	Условия								Статус
					1	2	3	4	5	6	7	8	
1	n = 5.7 k = 10	Ошибка! n должно быть целым числом	Ошибка! n должно быть целым числом	A	Л	-	-	-	-	-	-	-	Пройден
2	n = 5 k = 10.5	Ошибка! k должно быть целым числом	Ошибка! k должно быть целым числом	B->C	И	Л	-	-	-	-	-	-	Пройден
3	n = 25 k = 10	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	B->D->E	И	И	Л	И	Л	Л	-	-	Пройден
4	n = 2 k = 1	2	2	B->D->F ->H->I	И	И	Л	Л	Л	Л	И	Л	Пройден

Рис. 16. Набор тестов для покрытия операторов программы.

Для покрытия всех операторов программы достаточно было составить тесты для четырёх путей: A, BC, BDE, BDFHI.

4.2.2 Покрытие решений

В соответствии с этим критерием необходимо составить такой набор тестов, при котором каждое условие в программе примет как истинное, так и ложное значения. Таким образом, к тестам, составленным для метода покрытия операторов, необходимо добавить тесты, которые будут проверять все возможные переходы.

Тесты, покрывающие все решения программы представлены на Рис. 17.

№	Входные данные	Ожидаемый результат	Фактический результат	Путь	Условия								Статус
					1	2	3	4	5	6	7	8	
1	n = 5.7 k = 10	Ошибка! n должно быть целым числом	Ошибка! n должно быть целым числом	A	Л	-	-	-	-	-	-	-	Пройден
2	n = 5 k = 10.5	Ошибка! k должно быть целым числом	Ошибка! k должно быть целым числом	B->C	И	Л	-	-	-	-	-	-	Пройден
3	n = 25 k = 10	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	B->D->E	И	И	Л	И	Л	Л	-	-	Пройден
4	n = 2 k = 2	4	4	B->D->F ->G->J ->H->I	И	И	Л	Л	Л	Л	Л;И	И;Л	Пройден

Рис. 17. Набор тестов для покрытия решений программы.

4.2.3 Покрытие условий

В соответствии с этим критерием количество тестов должно быть таким, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз.

Тесты, покрывающие все условия программы представлены на Рис. 18.

№	Входные данные	Ожидаемый результат	Фактический результат	Путь	Условия								Статус
					1	2	3	4	5	6	7	8	
1	n = 5.7 k = 10	Ошибка! n должно быть целым числом	Ошибка! n должно быть целым числом	A	Л	-	-	-	-	-	-	-	Пройден
2	n = 5 k = 10.5	Ошибка! k должно быть целым числом	Ошибка! k должно быть целым числом	B->C	И	Л	-	-	-	-	-	-	Пройден
3	n = 25 k = 25	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	B->D->E	И	И	Л	И	Л	И	-	-	Пройден
4	n = -1 k = -1	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	B->D->E	И	И	И	Л	И	Л	-	-	Пройден
5	n = 2 k = 2	4	4	B->D->F ->G->J ->H->I	И	И	Л	Л	Л	Л	Л;И	И;Л	Пройден

Рис. 18. Набор тестов для покрытия условий программы.

4.2.4 Покрытие решений и условий

Согласно этому критерию набор тестов является достаточно полным, если удовлетворяются следующие требования: каждое условие в решении принимает каждое возможное значение по крайней мере один раз, каждый возможный исход решения проверяется по крайней мере один раз и каждой точке входа управление передается по крайней мере один раз.

Тесты, написанные ранее, обеспечивают покрытие решений и условий (Рис. 19).

№	Входные данные	Ожидаемый результат	Фактический результат	Путь	Условия								Статус
					1	2	3	4	5	6	7	8	
1	n = 5.7 k = 10	Ошибка! n должно быть целым числом	Ошибка! n должно быть целым числом	A	Л	-	-	-	-	-	-	-	Пройден
2	n = 5 k = 10.5	Ошибка! k должно быть целым числом	Ошибка! k должно быть целым числом	B->C	И	Л	-	-	-	-	-	-	Пройден
3	n = 25 k = 25	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	B->D->E	И	И	Л	И	Л	И	-	-	Пройден
4	n = -1 k = -1	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	B->D->E	И	И	И	Л	И	Л	-	-	Пройден
5	n = 2 k = 2	4	4	B->D->F ->G->J ->H->I	И	И	Л	Л	Л	Л	Л;И	И;Л	Пройден

Рис. 19. Набор тестов для покрытия решений и условий программы.

4.2.5 Комбинаторное покрытие условий

Этот критерий требует создания такого набора тестов, при котором каждая возможная комбинация результатов вычисления условий в каждом решении и каждая точка входа проверяются по крайней мере один раз.

Тесты, обеспечивающие комбинаторное покрытие условий, представлены на Рис. 20.

№	Входные данные	Ожидаемый результат	Фактический результат	Условия								Статус
				1	2	3	4	5	6	7	8	
1	n = 5.7 k = 10	Ошибка! n должно быть целым числом	Ошибка! n должно быть целым числом	Л	-	-	-	-	-	-	-	Пройден
2	n = 5 k = 10.5	Ошибка! k должно быть целым числом	Ошибка! k должно быть целым числом	И	Л	-	-	-	-	-	-	Пройден
3	n = 10 k = 25	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	И	И	Л	Л	Л	И	-	-	Пройден
4	n = 10 k = -1	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	И	И	Л	Л	И	Л	-	-	Пройден
5	n = 25 k = 10	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	И	И	Л	И	Л	Л	-	-	Пройден
6	n = 25 k = 25	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	И	И	Л	И	Л	И	-	-	Пройден
7	n = 25 k = -1	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	И	И	Л	И	И	Л	-	-	Пройден
8	n = -1 k = 10	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	И	И	И	Л	Л	Л	-	-	Пройден
9	n = -1 k = 25	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	И	И	И	Л	Л	И	-	-	Пройден
10	n = -1 k = -1	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	И	И	И	Л	И	Л	-	-	Пройден
11	n = 2 k = 2	4	4	И	И	Л	Л	Л	Л	Л;И	И;Л	Пройден

Рис. 20. Набор тестов для комбинаторного покрытия условий программы.

Покрыть истинную ветку условия 3 (путь B->D->E) невозможно из-за экстренного завершения программы при вводе строкового значения N.

4.2.6 Результаты тестирования методом «белого ящика»

В результате тестирования методом «белого ящика» было составлено 11 тестов. Программа прошла все 11 тестов. Это может говорить как о корректности программы, так и о том, что данный метод просто не подходит для тестирования данной программы, так как не рассматривает какую-либо ситуацию, в которой бы возникла ошибка.

4.3 Тестирование методом «чёрного ящика»

4.3.1 Разбиение на классы эквивалентности

Составлено разбиение на классы эквивалентности исходя из ограничений для программы, представленное на Рис. 21. Тесты для допустимых классов эквивалентности представлены на Рис. 22 и тесты для недопустимых классов эквивалентности — на Рис. 23.

Входное условие	Допустимые классы эквивалентности	Недопустимые классы эквивалентности
n	(1) n - целое число	(2) n - не целое
		(3) n - не число
	(4) 1 <= n <= 15	(5) n < 1
		(6) n > 15
k	(7) k - целое число	(8) k - не целое
		(9) k - не число
	(10) 1 <= k <= 15	(11) k < 1
		(12) k > 15

Рис. 21. Разбиение на классы эквивалентности.

№	Класс эквивалентности	Входные данные	Ожидаемый результат	Фактический результат	Статус
1	1, 5, 7, 10	n = 2 k = 2	4	4	Пройден

Рис. 22. Тесты для допустимых классов эквивалентности.

№	Класс эквивалентности	Входные данные	Ожидаемый результат	Фактический результат	Статус
1	2	n = 2.5 k = 2	Ошибка! n должно быть целым числом	Ошибка! n должно быть целым числом	Пройден
2	8	n = 2 k = 2.5	Ошибка! k должно быть целым числом	Ошибка! k должно быть целым числом	Пройден
3	3	n = "строка" k = 2	Ошибка! n должно быть целым числом	Ошибка! n должно быть целым числом	Пройден
4	9	n = 2 k = "строка"	Ошибка! k должно быть целым числом	Ошибка! k должно быть целым числом	Пройден
5	5	n = -1 k = 2	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 16	Пройден
6	11	n = 2 k = -1	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 16	Пройден
7	6	n = 25 k = 2	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 16	Пройден
8	12	n = 2 k = 25	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 16	Пройден

Рис. 23. Тесты для недопустимых классов эквивалентности.

Программа прошла все 9 тестов, составленных при рассмотрении классов эквивалентности входных данных.

4.3.2 Анализ граничных условий

В программе можно выделить следующие граничные условия:

- $n \geq 1$;
- $k \leq 15$;
- $n \geq 1$;
- $k \leq 15$.

Для каждой из границ определим тесты, соответствующие:

- граничному целому числу (верхнему/нижнему);
- целому числу, выходящему за границу (верхнюю/нижнюю) на единицу;

Составленные тесты представлены на Рис. 24.

№	Входные данные	Ожидаемый результат	Фактический результат	Статус
1	n = 1 k = 2	1	1	Пройден
2	n = 2 k = 1	2	2	Пройден
3	n = 15 k = 2	225	225	Пройден
4	n = 2 k = 15	32 768	32 768	Пройден
5	n = 0 k = 2	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	Пройден
6	n = 2 k = 0	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	Пройден
7	n = 16 k = 2	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	Пройден
8	n = 2 k = 16	Ошибка! Введите числа от 1 до 15	Ошибка! Введите числа от 1 до 15	Пройден

Рис. 24. Тесты граничных условий.

При анализе граничных условий был составлен набор из 8 тестов. Программа прошла все 8 тестов.

4.3.3 Причинно-следственная диаграмма

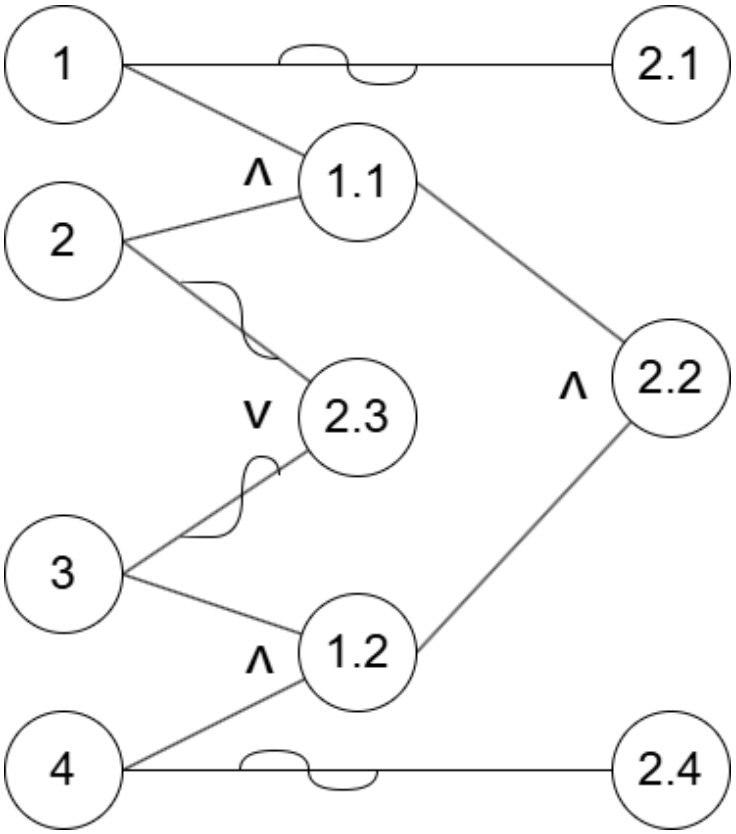

Рис. 25. Причинно следственная диаграмма.

Причины:

1. n – целое число;
2. $1 \leq n \leq 15$;
3. $1 \leq k \leq 15$.
4. k – целое число;

Промежуточные причины:

1.1 n – целое и $1 \leq n \leq 15$.

1.2 k – целое и $1 \leq k \leq 15$.

Следствия:

2.1 Программа выводит сообщение об ошибке "Ошибка! n должно быть целым числом" и заново запрашивает числа n и k;

2.2 Программа выводит значение n^k и завершает работу;

2.3 Программа выводит сообщение об ошибке "Ошибка! Введите числа от 1 до 15" и заново запрашивает числа n и k;

2.4 Программа выводит сообщение об ошибке "Ошибка! k должно быть целым числом" и заново запрашивает числа n и k;

На Рис. 25 представлена причинно-следственная диаграмма.

Таблица решений для диаграммы представлена на Рис. 26.

№	1	2	3	4	1.1	1.2	2.1	2.2	2.3	2.4
1	0	1	1	0	0	0	1	0	0	0
2	1	1	1	1	1	1	0	1	0	0
3	1	1	0	1	1	0	0	0	1	0
4	1	0	1	0	1	0	0	0	1	0
5	1	1	1	0	1	0	0	0	0	1

Рис. 26. Таблица решений.

Все тесты для таблицы решений уже были покрыты ранее при рассмотрении классов эквивалентности и граничных условий.

4.3.4 Результаты тестирования методом «чёрного ящика»

В результате тестирования методом чёрного ящика было составлено 17 тестов. Программа прошла все тесты. Это может свидетельствовать как о корректности программы, так и о том, что данный метод просто не подходит для данной программы, так как не рассматривает какую-либо ситуацию, в которой бы возникла ошибка.

Заключение

В ходе выполнения данной лабораторной работы были изучены методологии модульного тестирования: метод «белого ящика» и метод «чёрного ящика».

При помощи изученных методологий были спроектированы тесты для программы вычисления факториала числа и возведения числа в степень. При составлении тестов использовались методы:3

- Разбиения на классы эквивалентности;
- Анализа граничных значений;
- Причинно-следственной диаграммы;
- Критерия покрытия операторов;
- Критерия покрытия решений;
- Критерия покрытия условий;
- Критерия покрытия решений и условий;
- Критерия комбинаторного покрытия условий.

Тестирование проводилось с помощью комбинированного подхода: сначала проводилось тестирование методом «белого ящика», затем добавлялись тесты, основанные на методе «черного ящика».

При тестировании первой программы методом чёрного ящика не было пройдено 2 теста, методом белого ящика — 1 тест. Ошибки, из-за которых тесты не были пройдены, связаны с некорректной проверкой входных значений и неверно составленной спецификацией.

При тестировании второй программы методом чёрного ящика было составлено 17 тестов, методом белого ящика — 11 тестов. В результате тестирования не удалось составить тест, который программа бы не прошла. Это может свидетельствовать как о корректности программы, так и о том, что данные методы просто не подходят для тестирования данной программы, так как не рассматривают какую-либо ситуацию, в которой бы возникла ошибка.

Более эффективным методом для проверки программ оказался метод «чёрного ящика». С его помощью удалось составить два теста, которые программа не смогла пройти. С помощью метода «белого ящика» удалось составить только один такой тест.

Список литературы

[1] Майерс, Г. Искусство тестирования программ. – Санкт-Петербург: Диалектика, 2012 г.