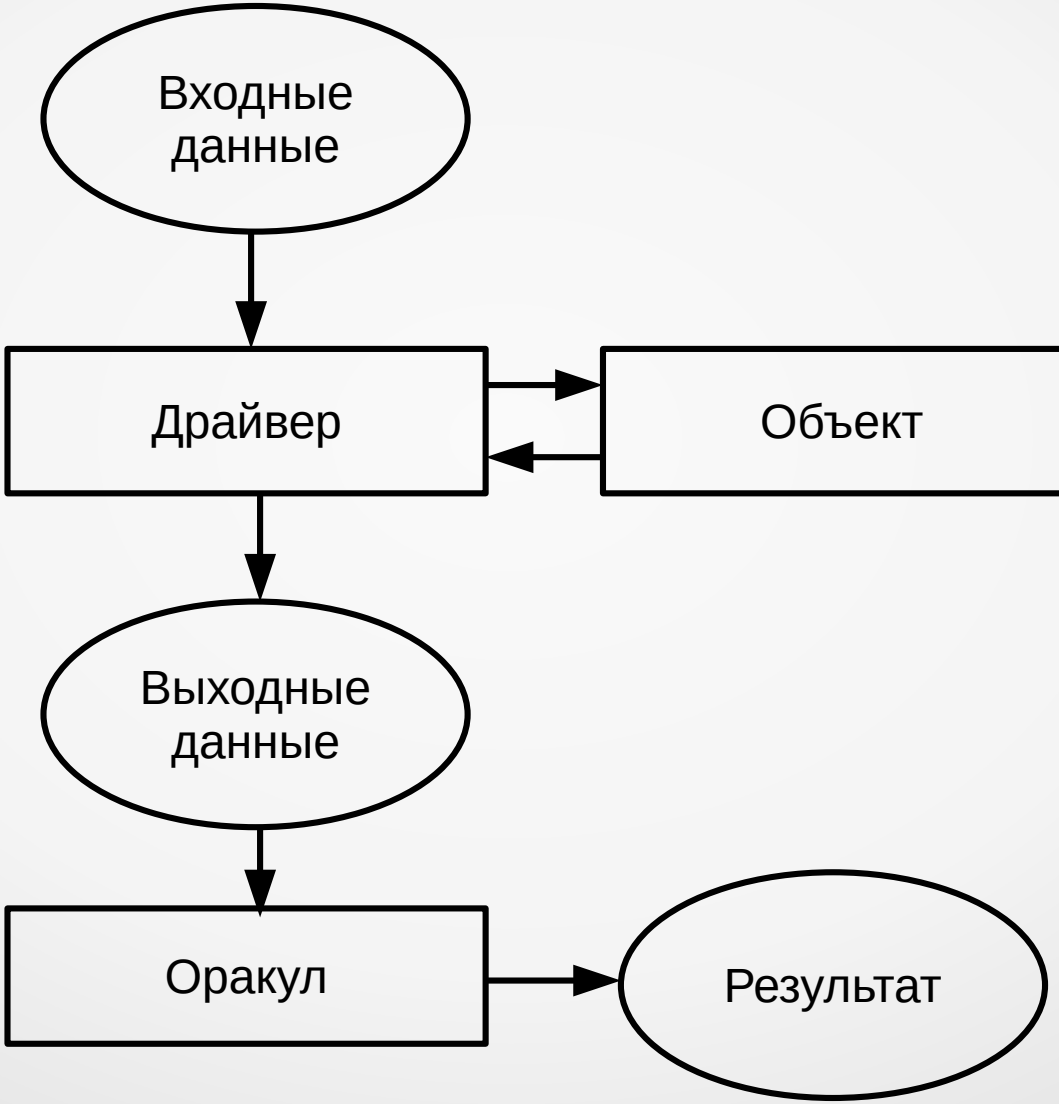
Основы разработки ПО

Методы разработки тестов

Кулаков Кирилл Александрович

Схема тестирования



Объект тестирования

- Функция/процедура, выполняющая полезную работу
  - входные данные (параметры вызова)
  - косвенные данные
    - глобальные переменные
    - структуры
    - экземпляры классов
  - другие функции
  - ресурсы
    - файлы
    - потоки
    - сокеты
  - результат (возвращаемое значение)

Примеры объектов

- int getRandomNumber() { … }
- char * convertString(char *) {…}
- {

  …

  totalCount++;

  …

  }
- {

  …

  int fd = open("output.txt", O_WRONLY|O_CREATE);

  …

  }

Драйвер

- Цели драйвера
  - Запустить объект тестирования
  - Настроить окружение для запуска
  - Задать параметры объекта
  - Получить результат работы объекта
- Драйвер пишется под конкретный объект и тест!
- Драйвер может включать вызов оракула!

Оракул

- Вызывается после завершения работы объекта тестирования
- Цель:
  - сравнить ожидаемый результат с фактическим
  - сообщить системе тестирования результат теста
- Оракулов (проверок) может быть несколько в 1 тесте

- В нашем случае оракулом выступает вызов методов ASSERT_*(), EXPECT_*(), SUCCEED(), FAIL(), ….

Запуск объекта тестирования

- Простой вариант

```
TEST(group_func1, simple) {
    // запуск функции
    func1();

    // здесь вызывается оракул
    SUCCEED();
}
```

Запуск объекта тестирования

- Функция с return

```
TEST(group_func1, return) {
    // запуск функции
    ret = func1();

    // здесь вызывается оракул, например сравнение
    ASSERT_EQ(ret, val);
}
```

Запуск объекта тестирования

- Функция с параметром(-ами)

```
TEST(group_func1, params) {
    // устанавливаем параметры
    arg1 = 10; arg2 = 20;

    // запуск функции
    ret = func1(arg1, arg2);

    // здесь вызывается оракул, например сравнение
    ASSERT_EQ(ret, val);
}
```

Запуск объекта тестирования

- Функция с параметром(-ами)
  - параметры могут загружаться из файла

```
char *filename = (char *)malloc(sizeof(char) * 1024);  // ОТКРЫВАЕМ ФАЙЛ
sprintf(filename, "%s/input321.txt", INPUTDIR);
int fd = open(filename, O_RDONLY);
free(filename);
if (fd < 0)
    ASSERT_EQ(errno, 0);
char *buf = (char *)malloc(sizeof(char) * 512);              // ЧИТАЕМ АРГУМЕНТЫ
read(fd, buf, 512);
close(fd);
int input = 0, output =0;
int ret = sscanf(buf, "%d %d", &input, &output);
free(buf);
ASSERT_EQ(ret, 2);
ret = fibonachi(input);                                      // ЗАПУСКАЕМ ОБЪЕКТ
ASSERT_EQ(ret, output);                                      // ПРОВЕРЯЕМ РЕЗУЛЬТАТ ОРАКУЛОМ
```

Запуск объекта тестирования

- Установка косвенных данных
  - выполняется для каждого теста

```
extern int globalVal;

TEST(group1, test1) {
    globalVal = 5;
    myFunc();
    SUCCEED();
}
```

Запуск объекта тестирования

- Чтение данных из входного потока

```
int inputData = open(file, O_RDONLY);
int oldStdin = dup(STDIN);
dup2(inputData, STDIN);
// запуск функции
close(inputData);
dup2(oldStdin, STDIN);
```

Запуск объекта тестирования

- Работа с файлами
  - готовим копии тестовых файлов
  - запускаем функцию
  - проверяем результат
  - удаляем копии (при необходимости)
- Работа с БД
  - создаем тестовую БД
  - заполняем данными
  - запускаем функцию
  - проверяем результат
  - удаляем БД (при необходимости)

Запуск объекта тестирования

- Взаимодействие с классами
  - использование реальных классов
  - использование классов-заглушек
- Взаимодействие с функциями
  - использование реальных функций
  - использование функций-заглушек
- Взаимодействие с библиотеками
  - использование реальных библиотек
  - использование заглушек
- Возможность подмены объекта в коде

Получение результата

- Чтение выходного потока

```
int outFd = open(testOutputFile, O_RWONLY|O_CREAT);
int oldOutput = dup(OUTPUT);
dup2(outFd, OUTPUT);
// запуск функции
close(outFd);
dup2(oldOutput, OUTPUT);
```

Получение результата

- Чтение созданных файлов

```
int testFd = open(testOutput, O_RDONLY);
int originFd = open(originalOutput, O_RDONLY);
int outputCount;
do {
    outputCount = read(testFd, outBuffer, outBufferSize);
    originCount = read(originFd, originBuffer, outBufferSize);
    ASSERT_EQ(outputCount, originCount);
    for (int i = 0; i < outputCount; i++) {
        ASSERT_EQ(outBuffer[i], originBuffer[i]);
} while (outputCount > 0);
```

Характеристики хорошего теста

- Цель тестирования выявление ошибок
- Ошибка — отклонение от эталона
- Варианты эталонов:
  - неформальное представление того, «как ПО должно работать»;
  - формальная техническая спецификация;
  - набор тестовых примеров;
  - корректные результаты работы программы;
  - другая (априори корректная) реализация той же исходной спецификации.

Характеристики хорошего теста

- **достижение** (Reachibility) — тест должен выполнить место в исходном коде, где присутствует программная ошибка;
- **повреждение** (Corruption) — при выполнении ошибки состояние программы должно испортиться с появлением сбоя;
- **распространение** (Propagation) — сбой должен распространиться дальше и вызвать неудачу в работе программы.

Позитивные и негативные тесты

- **Позитивные тесты**:
  - тесты, предназначенные для проверки, что программа выполняет свое основное предназначение;
  - тесты на основании «правильных» входных данных;
  - тестирование с целью проверки соответствий требованиям.

Позитивные и негативные тесты

- **Негативные тесты**:
  - тесты для проверки устойчивости ПО к негативным входным данным;
  - тесты на проверку устойчивости ПО к ошибкам пользователя;
  - тесты на то, что у программы нет неожиданных побочных эффектов;
  - тестирование с целью «сломаем это!».

Методы разработки тестов

Критерий	Черный ящик	Белый ящик
Основной уровень применимости	Приемочное тестирование	Модульное тестирование
Ответственный	Независимый тестировщик	Разработчик
Знание программирования	Необязательно	Необходимо
Знание реализации	Необязательно	Необходимо
Знание сценариев использования	Необходимо	Необязательно
Основа тестовых сценариев	Спецификации	Код

Классы эквивалентности и граничные значения

- Если от выполнения двух тестов ожидается один и тот же результат, они считаются эквивалентными.
- Группа тестов представляет собой класс эквивалентности, если выполняются следующие условия
  - Все тесты предназначены для выявления одной и той же ошибки.
  - Если один из тестов выявит ошибку, остальные, скорее всего, тоже это сделают.
  - Если один из тестов не выявит ошибки, остальные, скорее всего, тоже этого не сделают.

Классы эквивалентности и граничные значения

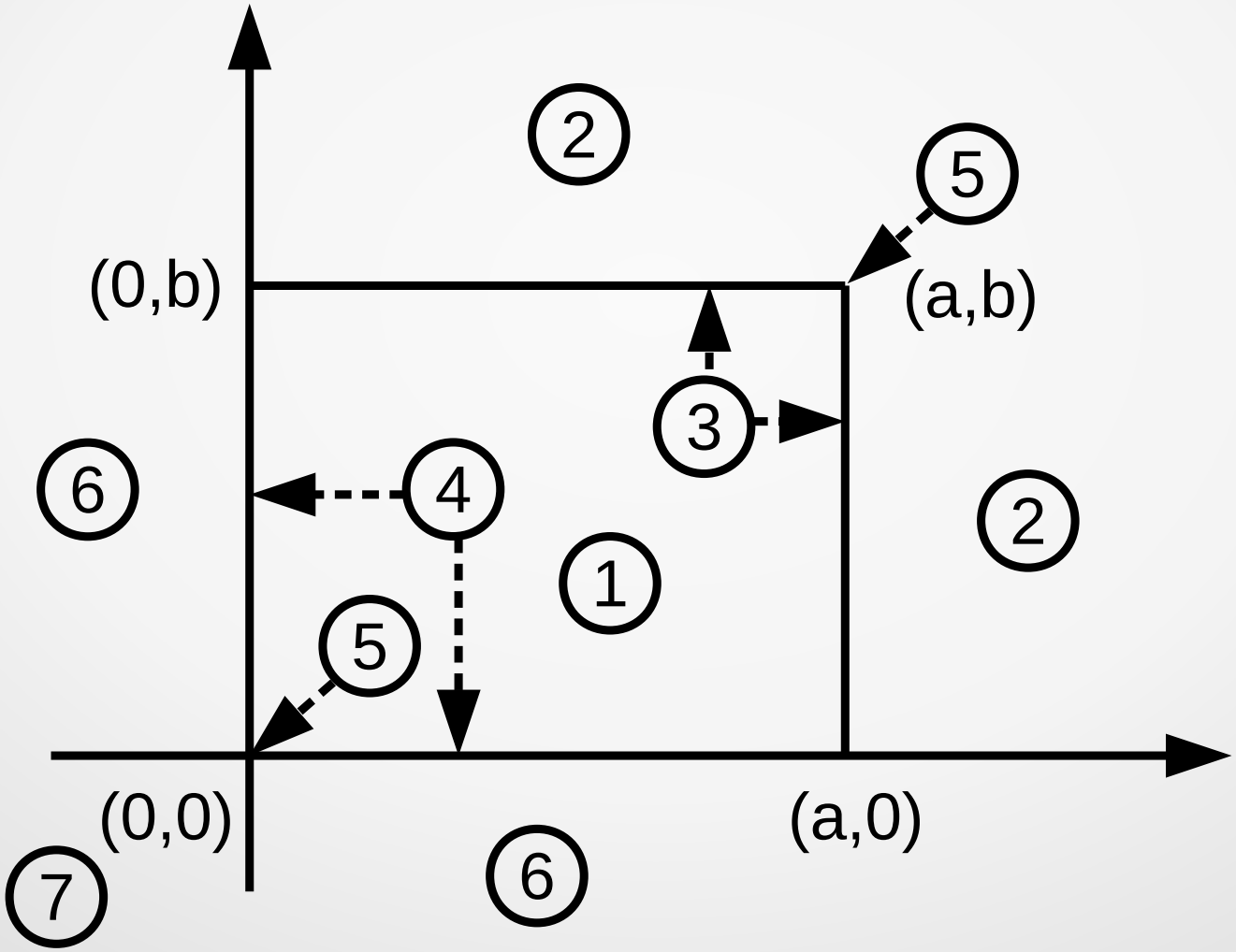
- Практические критерии классов эквивалентности:
  - Тесты включают значения одних и тех же входных данных.
  - Для их проведения выполняются одни и те же операции программы.
  - В результате всех тестов формируются значения одних и тех же выходных данных.
  - Либо ни один из тестов не вызывает выполнения блока обработки ошибок программы, либо выполнение этого блока вызывается всеми тестами группы.

Классы эквивалентности и граничные значения

- Рекомендации для поиска классов эквивалентности:
  - Не забывайте о классах, охватывающих заведомо неверные или недопустимые входные данные.
  - Организуйте формируемый перечень классов в виде таблицы или плана.
  - Определите диапазоны числовых значений.
  - Для полей или параметров, принимающих фиксированные перечни значений, выясните, какие из значений входят в перечень.
  - Проанализируйте возможные результаты выбора из списков и меню.
  - Поищите переменные, значения которых должны быть равными.
  - Поищите классы значений, зависящих от времени.
  - Выявите группы переменных, совместно участвующих в определенных вычислениях, результат которых ограничивается конкретным набором или диапазоном значений.
  - Посмотрите, на какие действия программа отвечает эквивалентными событиями.
  - Продумайте варианты операционного окружения.

Классы эквивалентности и граничные значения

- Пример: функция от двух аргументов



Белый ящик

- Техника Белого ящика включает в себя следующие методы тестирования:
  - покрытие решений
  - покрытие условий
  - покрытие решений и условий
  - комбинаторное покрытие условий

Белый ящик

- Покрытие операторов
  - выполнение каждого оператора программы по крайней мере один раз
  - Пример:

```
void func(int a, int b, float x) {
    if ((a > 1) && (b == 0)) x = x/a;
    if (a == 2 || x > 1) x++;
}
```

  - единственный тест со следующими значениями входных данных (a = 2, b = 0, x = 3)

Белый ящик

- Покрытие решений (покрытие переходов)
  - каждое условие в программе примет как истинное значение, так и ложное значение (проверка каждой ветви)
    - более сильный метод, т. к. операторы лежат на пути ветвей
  - Пример:

```
void func(int a, int b, float x) {
    if ((a > 1) && (b == 0)) x = x/a;
    if (a == 2 || x > 1) x++;
}
```

  - Тесты: (a=0, b=0, x=0), (a=2, b=0, x=2).

Белый ящик

- Покрытие условий
  - все возможные результаты каждого условия в решении были выполнены по крайней мере один раз (проверка всех компонент условий)
    - выполнение каждого оператора по крайней мере один раз
  - Пример:

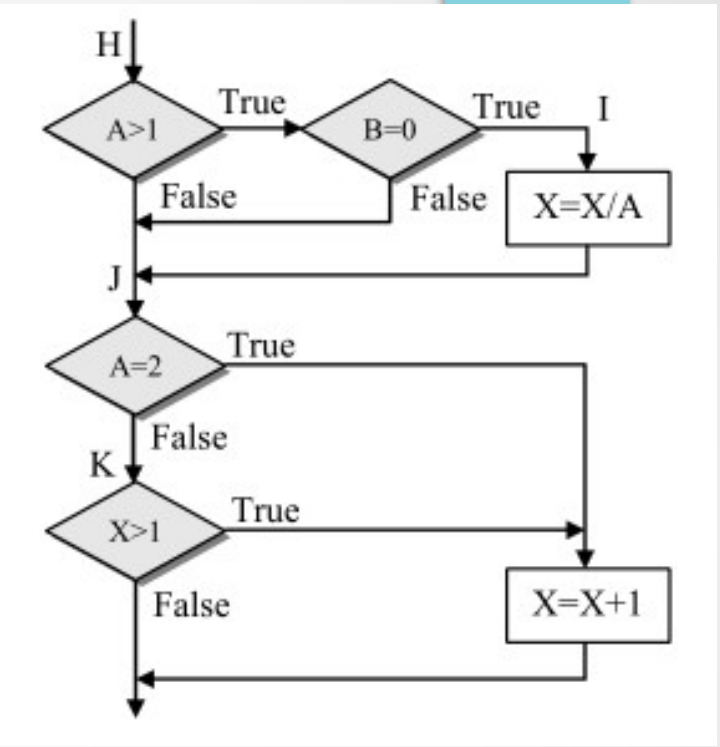
```
void func(int a, int b, float x) {
    if ((a > 1) && (b == 0)) x = x/a;
    if (a == 2 || x > 1) x++;
}
```

  - Тесты: (a=2, b=0, x=4), (a=1, b=1, x=0).

Белый ящик

- Покрытие условий и решений
  - результаты каждого условия выполнялись хотя бы один раз
  - результаты каждого решения (ветви) так же выполнялись хотя бы один раз
  - каждый оператор должен быть выполнен хотя бы один раз

- Недостатки:
  - не всегда можно проверить все условия
  - невозможно проверить условия, которые скрыты другими условиям
  - метод обладает недостаточной чувствительностью к ошибкам в логических выражениях



- Тесты:
  (A=2, B=0, X=4),
  (A=0, B=0, X=0) и
  (A=3, B=1, X=2)

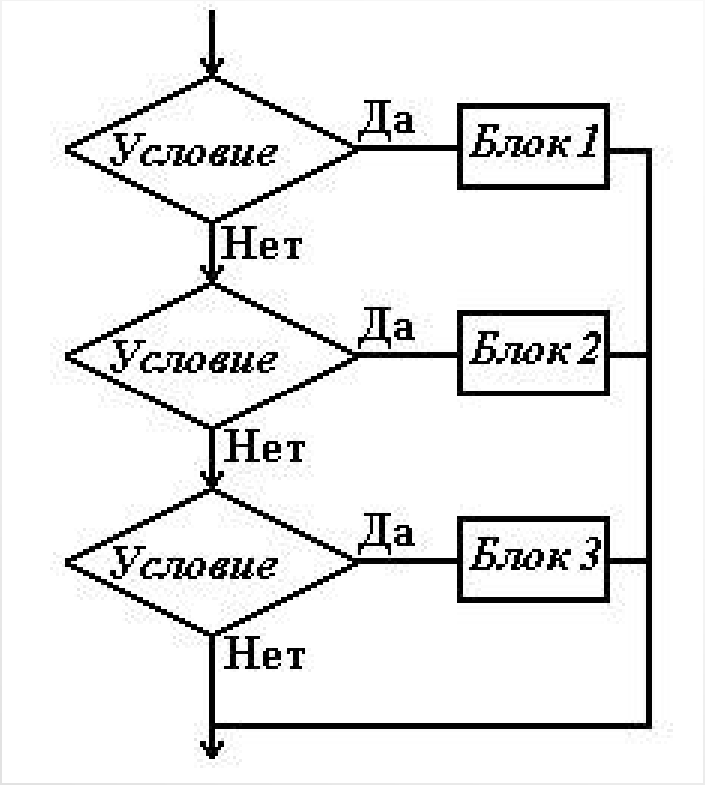
Белый ящик

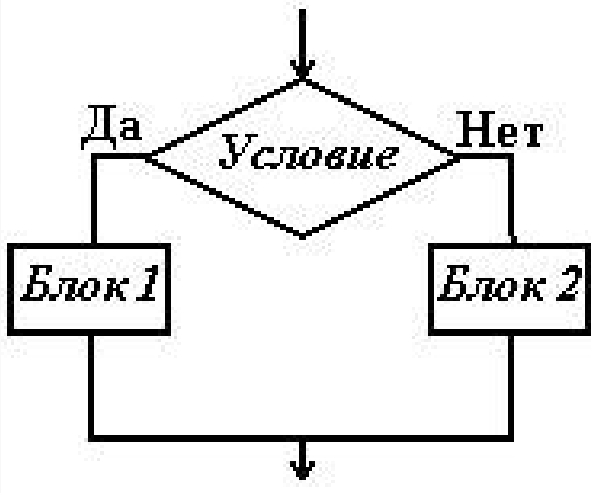
- Комбинаторное покрытие условий
  - все возможные комбинации результатов условий в каждом решении выполнялись хотя бы один раз
  - каждый оператор должен быть выполнен хотя бы один раз
  - Пример:

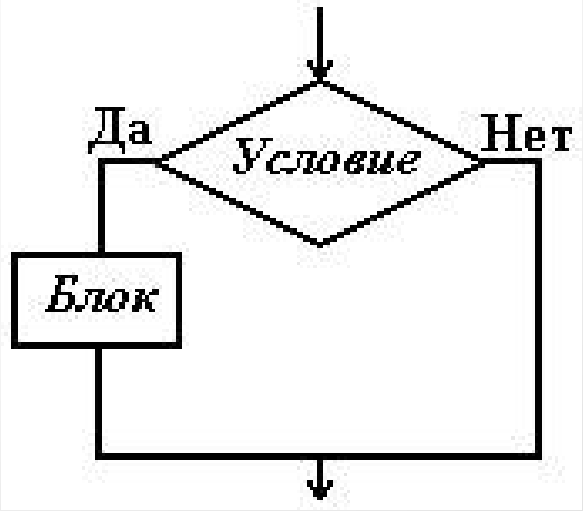
void func(int a, int b, float x) {
  if ((a > 1) && (b == 0)) x = x/a;
  if (a == 2 || x > 1) x++;
}
	a > 1, b = 0.
	a > 1, b ≠ 0.
	a ≤ 1, b = 0.
	a ≤ l, b ≠ 0.
	a = 2, x > 1.
	a = 2, x ≤ l.
	a ≠ 2, x > 1.
	a ≠ 2, x ≤ l.
		a = 2; b = 0; x = 4
		a = 0; b = 0; x = 0
		a = 2; b = 1; x = 0
		a = 0; b = 1; x = 2

Тестирование условий

- Варианты тестов:
  - условие истинно
  - условие ложно

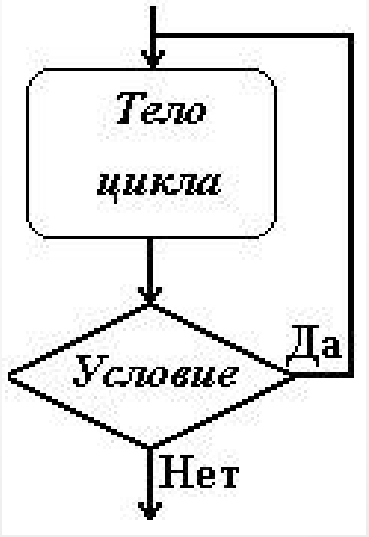


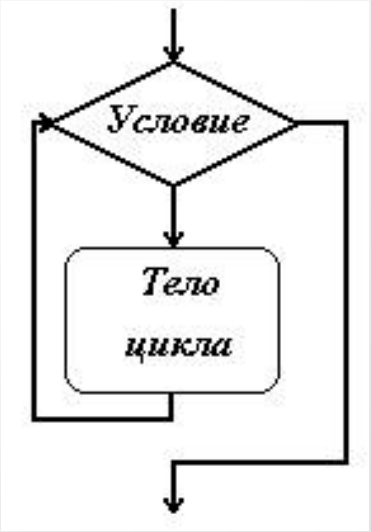




Тестирование циклов

- Варианты тестов
  - тело не выполняется
  - тело выполняется 1 раз
  - тело выполняется m раз (m<n)
  - тело выполняется n раз





Тестирование циклов

- Циклы могут быть разные