*Федеральное государственное бюджетное образовательное учреждение* *высшего образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана***  ***(национальный исследовательский университет)»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ \_\_\_\_\_ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА \_\_\_\_\_\_\_ Компьютерные Системы и сети (ИУ6)\_\_\_\_\_\_\_\_\_\_\_

**Отчет**

**по лабораторной работе № 2**

**Вариант №3, 3, 8**

**Дисциплина: технология разработки программных систем**

**Название лабораторной работы:** **Тестирование программного обеспечения**

Студент гр. ИУ6-44  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шумаков А.А.**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Хорунжина К.С.**

(Подпись, дата) (И.О. Фамилия)

Москва, 2018

**Цель работы** - знакомство с существующими стратегиями тестирования, приобретение навыков выбора стратегии и разработки тестов для отдельных задач, сравнение и оценка различных методов тестирования и их возможностей.

**Задание №1. Структурный анализ**

**Вариант 3**

для своего варианта задания выполните структурный контроль, используя перечень вопросов ниже. В процессе выполнения заполните таблицу. Вопросы, которые не актуальны для данной программы, можно в таблице не фиксировать. Сделайте общий вывод о роли структурного контроля в процессе создания программы. Сформулируйте его достоинства и недостатки. Ниже приведён текст программы.

{ Вар.3. Программа должна формировать массив чисел от 3 до 25 , а затем сортировать элементы массива по возрастанию и исключать повторяющиеся элементы}

1. *program v3;*
2. *{$APPTYPE CONSOLE}*
3. *uses*
4. *SysUtils;*
5. *const n=10;*
6. *var m:array [1..N] of integer;*
7. *i,j,k,L,b:integer;*
8. *begin*
9. *{ TODO -oUser -cConsole Main : Insert code here }*
10. *randomize; L:=N;*
11. *for i:=1 to L do begin m[i]:=random(25)+3; write(m[i],' ') end; writeln;*
12. *k:=1;*
13. *repeat i:=1;*
14. *repeat if m[i] = m[i+1] then*
15. *begin for j:=i+1 to L-1 do m[j]:=m[j+1]; dec(L); dec(i) end*
16. *else if m[i] > m[i+1] then begin*
17. *b:=m[i]; m[i]:=m[i+1]; m[i+1]:=b end;*
18. *inc(i);*
19. *until I<L-1;*
20. *inc(k);*
21. *until K>L-1;*
22. *for i:=1 to L do begin write(m[i],' ') end;*
23. *readln;*
24. *end.*

**Перечень вопросов для структурного контроля текста.**

1. *Обращения к данным.*
2. Все ли переменные инициализированы?
3. Не превышены ли максимальные (или реальные) размеры массивов и строк?
4. Не перепутаны ли строки со столбцами при работе с матрицами?
5. Присутствуют ли переменные со сходными именами?
6. Используются ли файлы? Если да, то
7. При вводе из файла проверяется ли завершение файла?
8. Соответствуют ли типы записываемых и читаемых значений?
9. Использованы ли нетипизированные переменные, открытые массивы, динамическая память? Если да, то
10. Соответствуют ли типы переменных при "наложении" формата?
11. Не выходят ли индексы за границы массивов?
12. *Вычисления.*
13. Правильно ли записаны выражения (порядок следования операторов)?
14. Корректно ли производятся вычисления неарифметических переменных?
15. Корректно ли выполнены вычисления с переменными различных типов (в том числе с использованием целочисленной арифметики)?
16. Возможно ли переполнение разрядной сетки или ситуация машинного нуля?
17. Соответствуют ли вычисления заданным требованиям точности?
18. Присутствуют ли сравнения переменных различных типов?
19. *Передачи управления.*
20. Будут ли корректно завершены циклы?
21. Будет ли завершена программа?
22. Существуют ли циклы, которые не будут выполняться из-за нарушения условия входа? Корректно ли продолжатся вычисления?
23. Существуют ли поисковые циклы? Корректно ли отрабатываются ситуации "элемент найден" и "элемент не найден"?
24. *Интерфейс.*
25. Соответствуют ли списки параметров и аргументов по порядку, типу, единицам измерения?
26. Не изменяет ли подпрограмма аргументов, которые не должны изменяться?
27. Не происходит ли нарушения области действия глобальных и локальных переменных с одинаковыми именами?

Таблица структурного контроля

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вопроса | Строки, подлежащие проверке | Результат проверки | Вывод |
| 1.1 | 6,7,10,11,12,15,17 | *L:=N*  *i:=1 (инициализация в цикле)*  *m[i]:=random(25)+3*  *k:=1*  *j:=i+1 (инициализация в цикле)*  *b:=m[i] ()* | Обращения к неинициализированным переменным не происходит, но массив m заполнится в диапазоне [3,28), что не соответствует условию задачи |
| 1.2 | ­5,6,11 | Размер массива задается константой n, массив заполняется в цикле случайными числами | Максимальный размер массива не превышен, строки не используются |
| 1.3 | Вся программа | **—** | С матрицами работы нет |
| 1.4 | Вся программа | **—** | Переменные со сходными именами не присутствуют |
| 1.5 | Вся программа | **—** | Файлы не используются |
| 1.6 | Вся программа | **—** | Нетипизированные переменные, открытые массивы, динамическая память не используются |
| 2.1 | Вся программа | **—** | Порядок записи операторов верен |
| 2.2 | Вся программа | **—** | Вычисления неарифметических переменных производятся корректно |
| 2.3 | Вся программа | **—** | Все переменные одного типа, все вычисления выполнены корректно |
| 2.4 | 11  15  18  20 | *m[i]:=random(25)+3*  *dec(L); dec(i);*  *inc(i);*  *inc(k);* | Переполнение разрядной сетки и ситуация машинного нуля не возможны |
| 2.5 | Вся программа | **—** | Все вычисления соответствуют заданным требованиям точности |
| 2.6 | Вся программа | **—** | Сравнения переменных различных типов отсутствуют |
| 3.1 | 11  13-21  14-19  15  22 | *for i:=1 to L do begin*  *repeat… until K>L-1;*  *repeat…until I<L-1;*  *for j:=i+1 to L-1 do*  *for i:=1 to L do begin* | Все циклы завершаются, зацикливание программы не происходит , последний цикл имеет лишнюю операторную скобку |
| 3.2 | Вся программа | **—** | Программа завершится, но результат ее работы будет некорректным |
| 3.3 | 13-21  14-19 | *repeat…until I<L-1;*  Данный цикл имеет неправильное постусловие | Из-за неправильного цикла с постусловием программа выдаст неверный результат |
| 3.4 | 14-17 | *if m[i] = m[i+1] then*  *begin for j:=i+1 to L-1 do m[j]:=m[j+1]; dec(L); dec(i) end*  *else if m[i] > m[i+1] then begin b:=m[i]; m[i]:=m[i+1]; m[i+1]:=b end;* | Поисковых цикл работает правильно |
| 4.1-4.3 | **—** | **—** | Не используются функции и процедуры |

**Вывод**: Структурный контроль полезен при осуществлении анализа маленьких программ или определенных фрагментов больших программ. Маленькие программы данным видом контроля легче анализируются, но результаты контроля, в первую очередь, будут зависеть от качества составленных вопросов. При контроле больших программ многие вопросы могут быть опущены или в программе будет большое количество кода, подвергающегося анализу. Однако, данный вид контроля позволяет проанализировать программу по конкретным интересующим нас пунктам, что, несомненно, является преимуществом данного вида контроля. К недостаткам метода можно отнести необходимость выполнять алгоритм в уме, то есть проверка работоспособности программы без ее запуска.

**Задание №2. Тестирование по принципу «белого ящика»**

**Вариант 3**

Для заданного фрагмента схемы алгоритма подготовьте тесты, используя методы стратегии "белого ящика". Предлагаемые тесты сведите в таблицу. Сравните тесты, предлагаемые различными методами. Сделайте вывод о роли тестирования с использованием стратегии "белого ящика" и возможностях его применения. Сформулируйте его достоинства и недостатки.

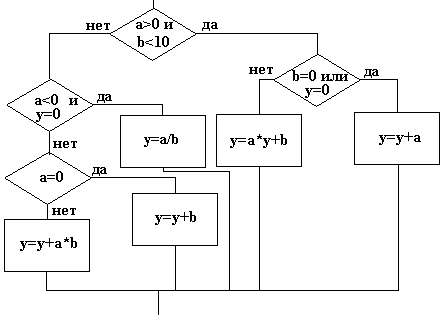


Таблица тестирования по методу «Белого ящика»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер теста | Метод  тестирования | Назначение теста | Значения исходных данных | Маршрут | Ожидаемый результат |
| 1  2  3  4  5 | Покрытие операторов | проверить y=y+a\*b  проверить y=y+b  проверить y=a/b  проверить y=a\*y+b  проверить y=y+a | a=-1 b=1 y=1  a=0 b=1 y=2  a=-1 b=0 y=0  a=1 b=1 y=1  a=2 b=0 y=-1 | нет, нет, нет  нет, нет, да  нет, да  да, нет  да, да | y=0  y=3  Деление на ноль  y=2  y=1 |
| 6  7  8  9  10 | Покрытие  решений  (переходов) | решение нет-нет-нет  решение нет-нет-да  решение нет-да  решение да-нет  решение да-да | a=-1 b=1 y=1  a=0 b=1 y=2  a=-1 b=0 y=0  a=1 b=1 y=1  a=2 b=0 y=-1 | нет, нет, нет  нет, нет, да  нет, да  да, нет  да, да | y=0  y=3  Деление на ноль  y=2  y=1 |
| 11  12  13  14  15  16 | Покрытие решений/  условий | a!=0 на ветви нет-нет-нет  a=0 на ветви нет-нет-да  a<0 и y=0 на ветви нет-да  b=0 или y=0 на ветви да-нет  b=0 или y=0 на ветви да-да  a>0 и b<10 на ветви да-да | a=-2 b=11 y=1  a=0 b=11 y=1  a=-6 b=0 y=0  a=1 b=5 y=1  a=1 b=0 y=1  a=1 b=5 y=0 | нет, нет, нет  нет, нет, да  нет, да  да, нет  да, да  да, да | y=-21  y=12  Деление на ноль  y=6  y=2  y=1 |
| 17  18  19  20  21  22  23  24  25 | Комбинаторное покрытие  условий | Комбинация a<0,b<10,y!=0  Комбинация a<0,b<10,y=0  Комбинация a=0,b<10,y=0  Комбинация a>0,b>10,y!=0  Комбинация a>0,b>10, y=0  Комбинация a>0,b=0, y!=0  Комбинация a>0,b<10, y=0  Комбинация a>0,b=0, y=0  Комбинация a>0,b<10,b!=0 y!=0 | a=-5 b=5 y=1  a=-5 b=0 y=0  a=0 b=5 y=0  a=5 b=15 y=1  a=5 b=15 y=0  a=5 b=0 y=1  a=5 b=2 y=0  a=5 b=0 y=0  a=5 b=5 y=-1 | нет, нет, нет  нет, да  нет, нет, да  нет, нет, нет  нет, нет, нет  да, да  да, да  да, да  Да, нет | y=-24  Деление на ноль  y=5  y=76  y=75  y=6  y=5  y=5  y=0 |

**Недостатки белого ящика**: даже в столь простой схеме алгоритма метод «белого ящика» занял длительное время и потребовал значительных усилий.

**Достоинства:** теоретически этот метод позволяет проверить структуру программы полностью, найти в каком месте в программе допущена ошибка в операторах и условиях.

**Вывод**: Метод покрытия операторов самый простой, но для данной задачи даже он был полезен, так как с помощью него легче всего находить ошибки. Однако данный метод не всегда позволяет найти все ошибки.

Метод покрытия решения более сложен, чем метод покрытия операторов, зато выявляет большее количество ошибок в условиях и проверяет пути без операторов. Для данной задачи тесты по данному методу совпали с тестами, составленными по методу покрытия операторов.

Метод покрытия решений/условий объединил в себе два предыдущих метода.

Метод комбинаторного покрытия решений выявляет наибольшее число возможных ошибок, как бы объединяя все остальные методы. Данный метод позволяет полностью проверить все условия всех решений в общем случае, но цена за это перебор всех возможных комбинаций, что приведет к увеличению времени тестирования.

Роль тестирования методом белого ящика – выявление структурных ошибок в программе, простейшие из методов тестирования ограниченно применимы к сложным программам, сложнейшие методы вроде комбинаторного – только к достаточно простым программам.

**Задание №3. Тестирование по принципу «черного ящика»**

**Вариант 8**

Внимательно изучите формулировку своего варианта задачи, подготовьте тесты по методикам стратегии "черного ящика". Выполните тестирование. Занесите в таблицу результаты. Сделайте вывод о роли тестирования с использованием стратегии "черного ящика" и возможностях его применения. Сформулируйте его достоинства и недостатки.

Программа должна вычислять значение интеграла функции  . Исходными данными являются: интервал и количество шагов.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер теста | Назначение теста | Значение исходных данных | Ожидаемый результат | Реакция программы | Вывод |
| 1 | Проверка на пустой ввод | **—** | Выдача сообщения об ошибке ввода данных | Выдача сообщения об ошибке | Есть проверка на пустой ввод |
| 2 | Проверка на частичный ввод и ввод строки | a= “Error”  b= “5”  n= “” | Выдача сообщения об ошибке ввода данных | Выдача сообщения об ошибке | Есть проверка на некорректный ввод для полей a и b |
| 3 | Проверка на ввод | a=1  b=2  n= “error100” | Выдача сообщения об ошибке  ввода данных | Ошибка программы | Нет проверки на неправильный ввод для поля n |
| 4 | Проверка на не берущийся интеграл | a=0  b=2  n=100 | Выдача сообщения о том что интеграл расходится | Ошибка программы | Нет проверки на расходимость интеграла |
| 5 | Проверка на не берущийся интеграл | a=-1  b=1  n=100 | Выдача сообщения о том что интеграл расходится | Ошибка программы | Нет проверки на расходимость интеграла |
| 6 | Проверка на ввод количества шагов n<0 | a=1  b=2  n=-100 | Выдача сообщения об ошибке ввода данных | -0.02527 | Нет проверки на отрицательный ввод количества шагов |
| 7 | Проверка на ввод количества шагов n=0 | a=1  b=2  n=0 | Выдача сообщения об ошибке ввода данных | Ошибка программы:  “обнаружено деление на ноль” | Нет проверки на ввод n=0 |
| 8 | Поверка на вычисления с вещественным интервалом | a=1.5  b=2.1  n=100 | 0.33647 | 0.33647 | Вычисления произведены корректно |
| 9 | Поверка на вычисления с отрицательным вещественным интервалом | a=-2.1  b=-1.5  n=100 | -0.33647 | 0.33647 | Вычисления произведены корректно |
| 10 | Проверка на граничное условие для типа real | a=1  b=2^32  n=100 | 22.181 | Ошибка программы | Ошибка из-за переполнения разрядной сетки |

**Недостатки черного ящика**: Возможность тестирования только готовой программы, невозможность проверить сам алгоритм программы, тестирование производится только исходя из входных данных.

**Достоинства**: Простота применения, ошибки довольно просто находить, занимает меньше времени по сравнению с методом «белого ящика».

**Вывод**: «Черный ящик» позволяет выявить обстоятельства, при которых поведение программы не соответствует спецификациям. «Черный ящик» применим только к готовой программе (или модуля программы), т.е. для тестирования этим методом необходимо сформировать исполняемый файл программы.

**Общий вывод**: В общем случае для тестирования программных продуктов лучшего всего подходит тестирование по методу «Черного ящика». Т.к. оно позволяет автоматически тестировать модули программы, при наименьшем количестве затрат на написание тестов и траты времени. Тестирование по методу «Белого ящика» хорошо применимо, если требуется проверить логику, внутреннюю структуры программы, но данный метод занимает большое количество времени (особенно для больших программ). Структурный контроль применим к небольшим программам или к определенным фрагментам больших программ. Осуществление структурного контроля, как правило, выявляет самые распространенные ошибки в программах.