Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана»
(национальный исследовательский университет)

Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Г.С. Иванова, Т.Н. Ничушкина, Е.К. Пугачев

Выбор алгоритмов обработки данных, тестирование и повышение качества программ

Методические указания по выполнению лабораторных работ
по дисциплине «Технология разработки программных систем»

Москва
Кафедра «Компьютерные системы и сети»
2018

УДК 004.021

Рецензент:

Г.С. Иванова, Т.Н. Ничушкина, Е.К. Пугачев

Выбор алгоритмов обработки данных, тестирование и повышение качества программ.
Электронное учебное издание. Методические указания по выполнению лабораторных работ по дисциплине «Технология разработки программных систем» - М.: МГТУ имени Н.Э. Баумана, 2018-48с.

Издание содержит: краткое описание основных структур и методов обработки данных, критерии оценки алгоритмов и структур данных, примеры структур данных, способы оценки и повышения эффективности программ, способы тестирования программ, варианты заданий и порядок выполнения лабораторных работ, предусмотренных учебным планом МГТУ им. Н.Э.Баумана.
Для студентов МГТУ имени Н.Э. Баумана направления «Информатика и вычислительная техника».

Иванова Галина Сергеевна
Ничушкина Татьяна Николаевна
Пугачев Евгений Константинович

Выбор алгоритмов обработки данных, тестирование и повышение качества программ.

© 2018 МГТУ имени Н.Э. Баумана

Оглавление

Предисловие

Методические указания предназначены для получения практических навыков в области разработки программ студентами, обучающихся на кафедре ИУ6.

Методические указания составлены в соответствии с самостоятельно устанавливаемым образовательным стандартом (СУОС), основной образовательной программой по направлению подготовки 09.03.01 «Информатика и вычислительная техника» бакалавров.

Цель выполнения лабораторных работ – приобретение навыков при выборе структур данных и методов их обработки, а также тестирования и оценки качества программ.

После выполнения лабораторных работ студенты овладеют:

- методиками расчета основных параметров, на основании которых осуществляется выбор структур данных и методов их обработки применительно к конкретной задаче;
- методами тестирования программных продуктов;
- методиками оценки качества программ;
- методиками оценки и повышения эффективности программ;
- практическими навыками анализа методов обработки данных;
- практическими навыками составления тестов и проверки работоспособности программы с учетом ее специфики;
- практическими навыками по определению эффективного способа реализации программы.

Планируемые результаты обучения

Студент должен знать:

- основные методы реализации операций, таких как поиск, упорядочение и корректировка;
- основные способы организации данных;
- методы тестирования;

– методы оценки эффективности и качества программ.

Студент должен уметь:

– выделять основные свойства структур данных и проводить сравнительный анализ;

– определять качественные и количественные критерии оценки методов обработки данных;

– проводить тестирование программного продукта на этапах проектирования и реализации;

– определять время работы программы;

– определять способы повышения эффективности и качества программы.

Студент должен иметь навыки:

- выбора структур данных и алгоритмов их обработки применительно к конкретной задаче;
- проведения эффективного тестирования;
- повышения эффективности и качества программ.

Для лабораторных работ необходимо предварительное освоение следующих дисциплин:

1. Основы программирования.
2. Информатика.
4. Объектно-ориентированное программирование.
5. Языки интернет-программирования.

Цели и задачи методических рекомендаций

Основной целью методических указаний является проверка студента или подтверждение студентом соответствия своих знаний и умений квалификационным требованиям, предъявляемым к бакалаврам по направлению 09.03.01 «Информати-

ка и вычислительная техника» . Перечень этих требований приведен в СУОС и основной образовательной программе по данному направлению.

Задачей методических указаний является подтверждение студентами способности разрабатывать компоненты программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования, а также осуществлять постановку и выполнять эксперименты по проверке корректности и эффективности программ.

Методические указания по освоению материала

Основной целью освоения материала методических указаний является самостоятельное изучение студентами важных разделов дисциплины «Технология разработки программных систем».

Для успешного освоения материала необходимо внимательно разбирать примеры, целесообразно синтезировать и прорабатывать аналогичные примеры. Ответы на вопросы позволят обучающемуся самому оценить степень понимания и усвоения теоретических положений, методик и методов. Выполнение заданий обеспечит приобретение умений и навыков, необходимых для постановки и решения задач проектирования, тестирования и оценки программ.

Для оценки степени понимания и усвоения теоретических положений, методик и методов, приобретения умений и навыков служит информативный отчет по каждой лабораторной работе, где полно и точно представлены результаты и их защита.

Введение

Актуальной задачей при разработке программных продуктов является выбор структур данных и методов их обработки. От выполнения данной задачи зависит эффективность работы разрабатываемого программного обеспечения. Грамотное решение указанной задачи может позволить экономно использовать ресурсы вычислительной системы, например, ресурсы оперативной и внешней памяти, а также процессорное время.

Важно отметить, что выбор структуры данных зависит от многих факторов, например, от режима работы разрабатываемой системы, от решаемых задач предметной области и др.

В первой части методических указаний на проработку выносятся способы организации структур данных, способы оценки применимости различных структур применительно к конкретным задачам, а также способы оценки алгоритмов, реализующие различные методы обработки данных для операций поиска, упорядочения и корректировки данных. Студентам предлагается проработать основной вариант по заданию и после получения результатов предложить альтернативный вариант решения, в котором устранены недостатки основного варианта.

Во второй части методических указаний на проработку выносятся способы оценки эффективности и качества программ. После оценки программы студентам необходимо предложить способы повышения эффективности и качества. Предложенные улучшения должны быть подтверждены практическими результатами замеров.

В третьей части методических указаний на проработку выносятся методы тестирования программных продуктов, которые используются на разных стадиях разработки и определять широкий спектр ошибок.

Глава 1. Выбор структур и методов обработки данных

При разработке алгоритмов программ часто возникает задача выбора структур данных и методов их обработки. Исходными составляющими для решения этой задачи являются описание набора и типов хранимых данных, а также перечень операций, выполняемых над ними.

Можно выделить следующие основные вопросы, на которые необходимо ответить при решении поставленной задачи:

- Как логически организовать структуру данных?
- Как реализовать структуру данных?
- Как осуществлять поиск информации?
- Как упорядочить данные?
- Как выполнить функции корректировки данных?

Цель работы – исследование структур данных, методов их обработки и оценки.

Продолжительность работы – 9 часов.

1.1 Краткие теоретические сведения о структурах данных

На этапе логического проектирования программного продукта разработчик определяет абстрактные структуры данных. Классификация по принципу связности элементов данных приведена в таблице 1.1.

Таблица 1.1. – Виды структур абстрактных данных

№	Тип связности элементов структуры данных	Примеры структур	Описание
1.	Элементы не связаны	Множество	Позволяют хранить ограниченное число значений определённого типа без определённого порядка. Доступ путем перебора элементов (счетный).
2.	Элементы связаны неявно	Таблицы	Данные четко структурированы и занесены в специальные таблицы.

			Имеется возможность определить положение одного элемента по отношению к другим. Доступ к элементам может быть последовательным и прямым.
3.	Элементы связаны явно	Списки, деревья и др.	Элементы структуры имеют служебные поля с целью указать непосредственно адрес(а) элемента(ов) с которым(и) он(и) связан(ы). Доступ к элементам косвенный.

При выборе множеств необходимо учитывать следующие их основные свойства:

- элементы множества не пронумерованы;
- отдельный элемент множества не идентифицируется;
- с элементами нельзя выполнить какие-либо действия;
- действия выполняются только над множеством в целом;
- элементы множества однотипные.

Очень часто при разработке программ используют таблицы. Различают четыре типа таблиц (просмотровые, прямого доступа, двоичного поиска и таблицы с перемешиванием). Доступ к таблицам может отличаться и от этого может зависеть, например, *эффективность* поиска данных. Эффективность поиска в таблицах различных типов оценивают, используя две характеристики:

S – длина поиска – количество записей, которые необходимо просмотреть, чтобы найти запись с заданным ключом;

L – средняя длина поиска при постоянной частоте обращения.

В просмотровых таблицах доступ к записям последовательный, а основные характеристики следующие: $S_i = i$; $L = (N+1)/2$, где N – количество записей.

В таблицах прямого доступа ключ однозначно определяет адрес. Основные характеристики: $S = 1$; $L = 1$.

Частными случаями таблиц прямого доступа являются: *массив, строка, запись*.

Для массива записей фиксированной длины адреса промежуточных записей задаются формулой:

$$A_i = A_1 + (i-1)*l,$$

где A_i – адрес i-ой записи; A_1 – адрес первой записи; l – длина записи.

В таблицах с перемешиванием (кэш-таблицах) L ≅ 1 при нерегулярных ключах. Для построения кэш-таблицы выбирается функция перемешивания (кэш-функция), определенная на множестве значений ключа. Доступ к классу прямой, а внутри класса последовательный.

В таблицах двоичного поиска записи сортируют по ключу поиска, что позволяет использовать более эффективный алгоритм поиска.

Структуры данных с явными связями в целом можно разбить на два класса: линейной структуры и нелинейной.

Линейные структуры с явными связями могут быть реализованы в виде изменяемых векторов или с использованием списков (одно-, двухсвязных или кольцевых). Можно выделить следующие частные случаи: очередь, стек, дек.

В древовидной структуре с явными связями записи располагаются по уровням, где на 1-м уровне только одна запись (корень дерева), а любая запись *i*-го уровня адресуется только с одной записью (*i*-1)-го уровня.

Отдельно можно выделить двоичное дерево, где каждая вершина может адресовать от 0 до 2 вершин следующего уровня.

В общем случае в N-мерном дереве каждая вершина может указывать от 0 до n вершин более низкого уровня.

Сетевые структуры с явными связями представляют граф, в котором вершины связаны между собой по принципу «многие ко многим». Реализуются различными способами: от матриц связности или инцидентности до n-связных списков.

1.2 Краткие теоретические сведения о методах обработки данных

1.2.1 Методы упорядочения данных

Частными случаями упорядочения являются сортировки (по возрастанию, по убыванию, по не возрастанию, по не убыванию). Понятие *сортировка* обычно применяют к линейным структурам данных. Понятие *упорядочение* используют применительно к нелинейным структурам данных, например, к деревьям.

В методе, базирующимся на попарном сравнении соседних элементов, количество сравнений определяется выражением:

$C = N (N-1)$, где N – количество элементов.

В методе, базирующимся на поиске минимального (максимального) элемента, количество сравнений определяется по формуле: $C = N(N-1)/2$.

В методе вставки количество сравнений определяется: $C = N(N-1)/4$

В методе Шелла количество операций сравнений оценивается следующим образом: $C \leq 0{,}5\ N^{3/2}$.

В методе квадратичной выборке количество сравнений в процессе сортировки равно: $C = (N-1)\sqrt{N}/2$.

В методе слияния количество сравнений зависит от характера данных и может быть различным. Метод состоит из двух основных этапов и для каждого этапа среднее количество сравнений определяется отдельно.

1.2.2 Методы поиска

Условия поиска могут быть различными: по совпадению, по попаданию в интервал, по удовлетворению арифметическому условию, по удовлетворению семантическому условию, по нескольким условиям. Если задано только одно значение признака поиска, то такой поиск называется единичным; если же задано множество признаков поиска, то это групповой поиск.

Эффективность различных алгоритмов поиска оценивается количеством сравнений пар признаков, необходимым для выполнения условия поиска.

При реализации последовательного метода поиска необходимо учитывать дополнительные условия: данные не отсортированы и неизвестно количество элементов, удовлетворяющих ключу поиска; данные не отсортированы, но известно число элементов, удовлетворяющих ключу поиска; данные отсортированы по ключевому полю.

Среднее число сравнений для реализации данного метода вычисляется:

$C = (N+1)/2$, где N – число записей в массиве.

При использовании метода двоичного поиска (или дихотомического поиска) необходимо учитывать ряд ограничений. Метод называют двоичным в связи с тем, что после каждого сравнения принимается одно из двух альтернативных решений. Среднее число операций сравнения при поиске в массиве равно

$$C_{ср} = [(N+1) \log_2 (N+1)]/N - 1$$

Метод вычисления адреса (адресный поиск) также является не универсальным. Он применяется к упорядоченным массивам, если значения ключевых признаков не повторяются и их число не превышает числа записей в массиве. При этом записи в массиве должны быть фиксированной длины. Номера записей и числовые значения ключевых признаков связаны между собой адресной функцией: $i = f(p)$, где i – номер записи (или адрес элемента в памяти); p – значение ключевого признака.

Способы, ускоряющие поиск в списковых структурах.

К таким способам относятся:

- вышеописанный способ с использованием адресной функции;
- метод К- и А-индексов;
- гнездовой способ организации.

В методе *К- и А-индексов* поиск можно разбить на два этапа: поиск в массиве индексов и поиск в основной структуре. Другими словами имеется массив *К* или *А* индексов и список записей.

Если значения ключей записей образуют арифметическую прогрессию, то называют К-индексами. Если значения ключей образуют арифметическую прогрессию, то называют А-индексами.

В гнездовом способе множество элементов разбивается на группы по определенному принципу. Затем эти группы объединяют в отдельные гнезда с одинаковой структурой. Далее строят дополнительную структуру, в которой все элементы связываются. Каждый элемент содержит ключевой признак и адрес только одного гнезда. Минимальное количество гнезд равно двум, при этом гнезда должны быть равнозначными. Поиск осуществляется в два этапа: поиск в вспомогательной структуре с целью выхода на гнездо и поиск внутри гнезда.

Отдельно можно выделить способ поиска в древовидных структурах. В частности рассмотрим поиск данных в бинарных упорядоченных деревьях. В упорядоченном бинарном дереве значение ключа каждого элемента больше, чем значение ключа у любого элемента его левой ветви, и не больше, чем значение ключа любого элемента его правой ветви. В связи с вышесказанным, алгоритм поиска достаточно простой: начинается поиск с вершины дерева, далее операция сравнения и в зависимости от ее результатов «отбрасывается» левая или правя ветвь. Бинарные деревья могут быть сбалансированными или несбалансированными. В сбалансированном дереве время выполнения операции поиска может быть существенно меньше.

Ниже приведено описание алгоритмов формирования упорядоченного бинарного дерева.

1.2.3 Способы корректировки данных

Корректировка представляет собой процедуру внесения изменений в структуру данных. Различают следующие виды корректировки: вставка (добавление), аннулирование (удаление), замена элементов.

Опишем некоторые особенности корректировки последовательных структур данных, реализованных в виде сортированных массивов. При необходимости сохранения сортировки для вставки требуется наличие свободных участков памяти между записями массива для размещения вставляемых записей. Заранее спланировать необходимый объем памяти и зарезервировать место (адреса) невозможно, поэтому реализация вставки невозможна без реорганизации массива. Для вставки новой записи требуется сдвиг всех последующих записей (в среднем – половины элементов массива).

При удалении записей также необходим сдвиг всех последующих записей (также в среднем половины элементов массива). Аннулирование записей возможно без реорганизации массива. Появляющиеся при этом свободные участки памяти могут быть отмечены как свободные в специальном массиве, тогда каждый раз при выполнении операции над данными придется проверять, не удалены ли данные.

Операция замены возможна без реорганизации массива.

Корректировка может выполняться двумя способами. В первом случае изменения вносятся в основной массив сразу. Во втором случае, применяемом при очень больших размерах массивов, записи с измененными данными накапливаются в специальном массиве изменений. При достижении определенного объема массив изменений объединяется с основным массивом.

1.3 Порядок выполнения лабораторной работы №1

В лабораторной работе необходимо выполнить следующие шаги:

1. Ознакомиться с теоретическими сведениями по абстрактным структурам данных и методам их обработки.
2. Определить достоинства и недостатки структуры данных применительно к поставленной задаче в соответствии с вариантом задания. Для указанной задачи и типа данных (см. таблицу 1.2) предложить способ реализации и определить требуемый объем памяти
3. Провести сравнительный анализ основных структур данных с целью определения наилучшей структуры применительно к поставленной задаче. Для этого необходимо определить требуемый объем памяти, тип доступа и др.
4. Оценить применимость метода поиска с учетом заданной структуры данных.
5. Определить возможные альтернативные методы поиска применительно к поставленной задаче и выбранного способа представления данных. Для выбора метода поиска использовать критерии: среднее количество сравнений, время выполнения (такты), универсальность и др.
6. Оценить применимость метода упорядочивания с учетом заданной структуры данных.
7. Определить возможные альтернативные методы упорядочивания применительно к поставленной задаче и выбранного способа представления данных. При выборе метода использовать критерии: среднее количество сравнений, время выполнения, универсальность и др.
8. Оценить применимость метода корректировки к заданной структуре данных.
9. Определить альтернативные способы корректировки применительно к выбранному способу представления данных. При выборе способа кор-

ректировки использовать критерии: время выполнения, ограничения на применимость и др.

10. Определить степень влияния метода корректировки на выполнение операций поиска и упорядочивания.

11. Обобщить полученные результаты и предложить улучшенный альтернативный вариант решения задачи, в котором должно быть минимум одно улучшение. В частности, могут быть заменены все методы обработки и структура данных. Также альтернативный вариант может быть определен с учетом выбранного режима работы программы.

При выполнении работы необходимо учитывать следующее.

Улучшения могут касаться как структуры данных, так и основных операций. При обосновании решений необходимо использовать количественные и качественные критерии. Количественными критериями являются: объем памяти, среднее количество сравнений и количество тактов. Качественные критерии определяют возможность использования того или иного метода применительно к разработанной структуре. К ним можно отнести: применимость операции только к упорядоченным данным; необходимость знать количество элементов; наличие признака разбивки на гнезда; необходимость в прямом доступе к элементам; знание граничных значений; невозможность создать структуру в соответствии с арифметической прогрессией и др.

В целом сравнение вариантов выполняется по требуемым объемам памяти (*емкостной сложности*) и времени выполнения операций (*временной сложности*).

Емкостная сложность оценивается по физической структуре с учетом используемых типов данных.

Для оценки временной сложности можно использовать следующую методику:

1) разрабатывается фрагмент алгоритма и по нему примерно определяется последовательность выполнения команд, реализующих ту или иную операцию над данными;

2) полученный результат преобразуется по таблице соответствий (см. таблицу 1.2) к времени выполнения фрагмента алгоритма в машинных тактах.

Таблица 1.2 - Коэффициенты приведения команд Borland Pascal (Pentium)

Операция	Обозначение	Время, тактов	Операции	Обозначение	Время, тактов
a:=b	$t_{=}$	2	a/b	$t_{/}$	28
c:=a±b, a>b	t_{+}	2	a:=b^.num	$t_{=\wedge}$	2
A:=a±b	$t_{=+}$	2*	b:=b^.p	$t_{\wedge}$	2*
a:=a±const	t_{++}	1	сдвиги	$t_{\rightarrow}$	1
A±const	t_{+c}	1	a[i]	t_i	2
a*b	t_{*}	20	a[i,j]	$t_{i,j}$	26
If	t_{if}	$t_{пр}$+1+p1×t 1+ +p2×t2 **	for	t_{for}	$t_{уст}+t_{пр}+1+$ $+n(t_{тела}+t_{пр}+2)$ ***

* – операции накопления/переадресации (при использовании в цикле коэффициент равен 1);

**- $t_{пр}$ – время проверки условия, p1,p2 – вероятности выбора соответствующих ветвей, t1,t2 – времена выполнения ветвей;

***- $t_{уст}$ – время установки начального значения индекса, $t_{пр}$ – время проверки условия, $t_{тела}$ – время выполнения тела цикла; для цикла вида

for i:=1 to n do … получаем: $t=2+2+1+n(t_{тела}+2+2)= 5+n(t_{тела}+4)$.

Пример. Выполнить оценки памяти и вычислительной сложности выполнения операций поиска и удаления элемента для двух реализаций хранения последовательности n целых чисел: в виде массива и в виде линейного односвязного списка (см. рис. 1.1).

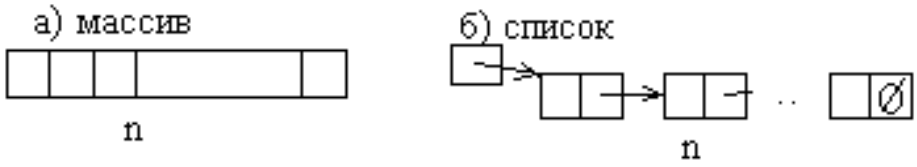


Рисунок 1.1 – Массив (а) и односвязный список (б)

Решение.

1. *Оценка памяти*:

а) для массива: $V= l_{эл}*n$, если в массиве хранятся целые числа, то $V=2n$ (байт)

б) для списка: $V=(lэл+lук)*n+lук,$

если в списке хранятся целые числа, а указатели имеют длину 4 байта, то $V=6n+4$ (байт)

2. *Оценка времени поиска i-го элемента данных:*

а) для массива:

поиск выполняется по индексу $A_i =A_б+(i-1)*2=A_б-2+i*2=const+i*2,$

операция умножения на 2 выполняется посредством сдвига на 1 разряд влево,

откуда $t= t_{++} + t_{\rightarrow} = 2$ (такта)

б) для списка:

доступ к элементу выполняется последовательно

f:=first; for j:=1 to i do f:=f^.p;

определяем $t =2+2+1+(i-1)*(t_{\wedge}+2+2)=5+5(i-1),$

минимальное время – первого элемента $t_{min}=5,$

максимальное время поиска – последнего элемента $t_{max} =5+(n-1)(t_{\wedge}+4)=5+5n-5=5n,$

в среднем $t= (t_{max}+t_{min})/2= 2.5+2.5n$ (тактов).

3. *Оценка времени удаления i-го элемента:*

а) для массива:

при удалении i-го элемента выполняется сдвиг остальной части массива:

for j:=i to n-1 do a[j] := a[j+1];

откуда $t=t_{+c}+2+2+(n-i)\ (t_{+c}+t_i+t_i+t_{=})=5+7(n-i),$

минимальное время работы цикла (при $i=n-1$) $t_{min}=12$,

максимальное время работы цикла (при $i=1$) $t_{max}=5+7(n-1)=-2+7n$,

соответственно среднее время работы цикла $t=5+3,5n$ (тактов).

При $i=n$ сдвиг не выполняется, а только значение n уменьшается на единицу ($t_{++}=1$). Тогда среднее время удаления равно: $t = t_{пр} + (t_{++} + t_{max})/2 = 2+(1-2+7n)/2 =1,5 + 3,5n$ (таков)

б) для списка:

при удалении элемента достаточно перезаписать адресное поле: *f^.p:=r^.p* (без учета времени освобождения памяти и времени поиска элемента), следовательно, $t=t_{\wedge}+t_{=\wedge}=3$ (такта).

При выборе структуры учитывают удельный вес операций анализируемых операций.

Отчет должен включать:

- название работы и ее цель;
- конкретный вариант задания;
- рисунок структуры данных в соответствии с вариантом задания;
- оценку требуемого объема памяти;
- описания или схемы алгоритмов оцениваемых операций;
- расчеты оценок вычислительной сложности;
- описание альтернативного варианта и его оценки;
- выводы о достоинствах и недостатках основного варианта в сравнении с альтернативным вариантом реализации.

Примечания:

1. Если структура данных в альтернативном варианте заменена, а метод реализующий операцию остался прежним, то для обоснования необходимо определить количество тактов.

2. Результаты выполнения лабораторной работы свести в таблицу 1.3.

Таблица 1.3 – Таблица результатов

	Структура данных	**Метод поиска**	**Метод упорядочения (сортировки)**	**Метод корректировки**
Основной вариант	*Наименование, количественные и качественные критерии*	"-"	"-"	"-"
Альтернативный вариант	"-"	"-"	"-"	"-"

1.4 Варианты заданий

Таблица 1.4 - Варианты заданий.

Вари-ант	За-дача	Структура	Поиск	Упорядочение	Корректировка
1	1	Таблица	последоват.	пузырьком	удаление сдвигом
2	1	Таблица	дихотомич.	вставкой	удаление маркир.
3	1	Таблица	вычисл. адр.	Шелла	замена данных
4	1	Таблица	последоват.	кв. выборки	удаление сдвигом
5	2	Таблица	дихотомич.	слиянием	вставка записи
6	2	Таблица	последоват.	пузырьком	удаление маркир.
7	2	Таблица	дихотомич.	вставкой	удаление сдвигом
8	2	Таблица	вычисл. адр.	Шелла	замена данных
9	2	Таблица	последоват.	кв. выборки	удаление сдвигом
10	2	Таблица	дихотомич.	слиянием	удаление маркир.
11	2	Таблица	вычисл. адр.	пузырьком	удаление сдвигом
12	3	Таблица	дихотомич.	вставкой	удаление маркир.
13	3	Таблица	вычисл. адр.	Шелла	замена данных
14	3	Таблица	последоват.	кв. выборки	удаление сдвигом
15	3	Таблица	дихотомич.	слиянием	удаление маркир.
16	4	Список	гнездовой	любой	удаление записи
17	4	Список	последоват.	любой	удаление записи
18	4	кольц.сп.	последоват.	любой	удаление записи
19	4	$2^{св}$ список	последоват.	любой	замена данных
20	5	Список	гнездовой	любой	удаление записи
21	5	Список	последоват.	любой	удаление записи
22	5	кольц.сп.	последоват.	любой	удаление записи
23	5	$2^{св}$ список	последоват.	любой	удаление записи
24	6	Список	гнездовой	любой	удаление записи
25	6	Список	последоват.	любой	удаление записи
26	6	кольц.сп.	последоват.	любой	удаление записи

27	6	$2^{св}$ список	последоват.	любой	удаление записи
28	7	нелин.$2^{св}$ сп.	последоват.	любой	удаление записи
29	7	Дерево	гнездовой	любой	удаление записи
30	6	Дерево	гнездовой	любой	удаление записи
31	8	бин. упор. дерево (алгоритм А)	последоват.	нет	исключения неполной вершины
32	8	"-" (Б)	последоват.	нет	исключения неполной вершины
33	9	"-" (А)	последоват.	нет	исключения неполной вершины
34	9	"-" (Б)	последоват.	нет	исключения полной вершины
35	10	"-" (А)	последоват.	нет	исключения неполной и полной вершин

Ниже приведены задачи, которые необходимо проработать в рамках различных вариантов выполнения лабораторной работы.

Задачи:

Задача 1. Даны N записей вида: код материала; дата поступления; номер склада; количество; сумма.

Задача 2. Дана таблица материальных нормативов, состоящая из K записей фиксированной длины вида: код детали; код материала; единица измерения; номер цеха; норма расхода.

Задача 3. Даны M записей вида: код группы; ФИО; дата рождения.

Задача 4. Дано N элементов, где каждый элемент является предложением на естественном языке.

Задача 5. Даны элементы: 230, 150, 100, 85, 77, 33, 93, 200, 57, 137, 205.

Задача 6. Даны элементы: 130, 50, 120, 185, 27, 43, 913, 210, 5, 17, 245.

Задача 7. «Поток ИУ-6». В потоке имеется 6 групп, в каждой группе не более 25 человек. Необходимо хранить следующую информацию о каждом студенте: ФИО, дата рождения, пол.

Задача 8. Даны ключи: 61, 36, 50, 42, 17, 75, 54, 14, 90, 254, 134, 248, 123.

Задача 9. Даны ключи: 81, 122, 51, 3, 52, 116, 79, 46,178, 44, 124, 45, 165, 38, 198, 9, 7.

*Задача 10.*Даны ключи: 88, 36, 57, 47, 57, 85, 343, 64,72, 638, 596, 94, 7, 24, 514, 36, 173, 89, 673,524.

1.5 Вопросы для самопроверки

1. Применим метод двоичного поиска к спискам?
2. Как оценить гнездовой метод поиска?
3. Перечислите достоинства и недостатки двухсвязного списка по отношению к односвязному?
4. Какие количественные оценки применяют к методам сортировки?
5. Какие недостатки у дихотомического метода в сравнении с последовательным методом?
6. Как оценить метод Шелла?
7. Какой метод сортировки лучше?
8. Какие недостатки у метода удаления сдвигом?
9. Как построить бинарное дерево?
10. Какое бинарное дерево лучше?
11. Когда используют структуры данных с явными связями?
12. Зачем нужна кэш-таблица?
13. Какие недостатки у метода упорядочения обменом?
14. Какие недостатки у метода вычисления адреса?
15. Какие ограничения на количество гнезд в гнездовом методе?
16. В каком дереве удаление элемента выполняется быстрее?

Глава 2. Оценка эффективности и качества программы

В настоящее время перед разработчиками программного обеспечения стоит задача создания эффективных, технологичных и качественных программ. Данная задача усложняется тем, что четких и универсальных рекомендаций для оценки указанных свойств не существует. Однако, на данный момент накоплен некоторый опыт, который может быть использован разработчиками.

Цель работы - изучить известные критерии оценки и способы повышения эффективности и качества программных продуктов.

Продолжительность работы - 4 часа.

2.1 Основные теоретические сведения

Можно выделить следующие основные характеристики, используемые при оценке качества программных продуктов: т*ехнологичность, эффективность, универсальность, надежность и др.*

От технологичности зависят трудовые и материальные затраты на реализацию и последующую модификацию программного продукта. В данной лабораторной работе из-за ограниченного объема времени оценка этого качества не выполняется.

Эффективность программы можно оценить отдельно по каждому ресурсу вычислительной машины. Критериями оценки являются:

- *Время ответа или выполнения.* Оценивается для программ, работающих в интерактивном режиме или в режиме реального времени.
- *Объем оперативной памяти.* Является важным для вычислительных систем, где оперативная память является дефицитным ресурсом.
- *Объем внешней памяти.* Оценивается, если внешняя память дефицитный ресурс или при интенсивной работе с данными.
- *Количество обслуживаемых терминалов* и др.

Время выполнения программы в первую очередь зависит от используемых в ней методов. Однако, важную роль играют циклические фрагменты с большим количеством повторений. Поэтому по возможности необходимо минимизировать тело цикла.

При написании циклов рекомендуется:

- выносить из тела цикла выражения, не зависящие от параметров цикла;
- заменять «длинные» операции умножения и деления вычитанием и сдвигами;
- уменьшить количество преобразования типов в выражениях;
- исключать лишние проверки;
- исключать многократные обращения к элементам массивов по индексам (особенно многомерных, так как при вычислении адреса элемента используются операции умножения на значение индексов).

Ниже приведен перечень рекомендаций, которые применяются для экономии памяти.

- Следует выбирать алгоритмы обработки, *не требующие дублирования исходных данных структурных типов*. Например, метод «вставки» - не требует дополнительных массивов.
- По возможности использовать динамическую память. При необходимости выделять память, а потом освобождать.
- При передаче структурных данных в подпрограмму по значению, копии этих данных размещаются в стеке. Избежать копирования можно, если передавать данные не по значению, а как неизменяемые (описанные const). В последнем случае в стеке размещается только адрес данных, например:

Type Massiv=array[1..100] of real;

function Summa(Const a:Massiv;

Качество программных продуктов может оцениваться следующими критериями:

- правильность – функционирование в соответствии с заданием;

- универсальность – обеспечение правильной работы при любых допустимых данных и содержание средств защиты от неправильных данных;
- надежность (помехозащищенность) – обеспечение полной повторяемости результатов, т.е. обеспечение их правильности при наличии различного рода сбоев;
- проверяемость – возможность проверки получаемых результатов;
- точность результатов – обеспечение погрешности результатов не выше заданной;
- защищенность – сохранение конфиденциальности информации;
- эффективность – использование минимально возможного объема ресурсов технических средств;
- адаптируемость – возможность быстрой модификации с целью приспособления к изменяющимся условиям функционирования;
- повторная входимость – возможность повторного выполнения без перезагрузки с диска (для резидентных программ);
- реентерабельность – возможность «параллельного» использования несколькими процессами.

Экспериментально подтвердить сделанные выводы можно следующим образом:

1. Ввести в программу контрольные точки, например, фиксировать время начала выполнения цикла и время окончания с помощью процедуры Gettime (или других команд), а затем определить время выполнения.

2. Чтобы время выполнения было легче зарегистрировать или увидеть, можно воспользоваться, образно говоря, «увеличительной линзой». Другими словами, можно ввести дополнительный цикл, телом которого является исследуемый фрагмент.

*Примечание. При внесении в программу контрольных точек, необходимо учитывать их влияние.

2.2 Порядок выполнения лабораторной работы №2

1. Ознакомиться с теоретическими сведениями.
2. Исследовать заданный пример программы с целью определения ее эффективности и качества.
3. Определить основные критерии оценки и количественные характеристики для заданной программы.
4. Предложить варианты повышения эффективности и улучшения качества для заданного примера программы.
5. Составить отчет.

Отчет должен включать:

- название работы и ее цель;
- конкретный вариант задания и исходную программу;
- улучшенный вариант программы;
- таблицу оценки эффективности (табл.2.1);
- таблицу оценки качества (табл.2.2);
- описание альтернативного варианта и его оценки;
- выводы о достоинствах и недостатках основного варианта в сравнении с альтернативным вариантом реализации.

Таблица 2.1- Оценка эффективности.

	Исходная программа		Улучшенная программа	
Критерий оценки	Недостатки	Количественная оценка	Улучшения	Колич-я оценка
Время выполнения				
Оперативная память				
Внешняя память				
. . . . .				

Таблица 2.2 - Оценка качества.

	Правиль-ность	Универсаль-ность	Проверяе-мость	Точность результатов
Недостатки				
Оценка				

2.3 Варианты заданий

1. Написать программу, которая строит отсортированный список вещественных чисел в динамической памяти. Реализовать сортировку по убыванию (программа - v1.dpr).

2. Написать программу, которая позволяет строить графики функций y=| cos(x)*x| и y=| sin(x)| . Обеспечить возможность вывода на экране графиков одновременно для двух функций с наложением фиксированных осей координат. (в папке V2 программа - v2. dpr).

3. Написать программу, в которой создается динамический список неповторяющихся целых чисел в диапазоне от -50 до 50. Обеспечить прямой и обратный вывод элементов списка. Программа должна посчитать сумму: 1+n,2+n-1, …i+n-i+1, …n/2+n/2+1. (программа - v3. dpr).

4. Написать программу, в которой реализуется метод «хорд» для нахождения корней функции с точностью 0.001 (программа - v4. dpr).

5. Вычислить значение интеграла с точностью 0.0001 с помощью метода прямоугольников (программа - v5. dpr).

6. Вычислить значение интеграла с точностью 0.001 используя метод Симпсона, т.е. $\int_a^b f(x)dx \approx \frac{h}{3}(f_0 + f_n + 4\bullet(f_1 + f_3 + \ldots + f_{n-1}) + 2\bullet(f_2 + f_4 + \ldots + f_{n-2}))$ (программа - v6. dpr).

7. Программа должна генерировать массив вещественных чисел в диапазоне от –10 до 10 и определять все минимальные положительные элементы, если их много (программа - v7. dpr).

8. Создать программу, в которой создается массив целых чисел в диапазоне от -20 до 5 и сортируется по убыванию (программа - v8. dpr).

9. Создать программу, в которой генерируется массив целых чисел в диапазоне от –20 до –10, а затем сортируется по убыванию с исключением повторяющихся элементов (программа - v9. dpr).

10. Программа должна сортировать четные строки массива вещественных чисел (программа - v10. dpr).

11. Написать программу вычисления суммы ряда с заданной точностью. Например, дан ряд: $S=\frac{1}{4}-\frac{1}{16}+\frac{1}{96}-...(-1)^{n+1}\frac{1}{4^n\cdot n!}$, ряд сходится и имеет множитель $m=-\frac{1}{4\cdot n}$. Определить сумму ряда с точностью 0.0001 (программа - v11. dpr).

12. Создать массив целых чисел. Определить все максимальные элементы в каждом столбце (программа - v12. dpr).

13. Реализовать итерационный метод Зейделя для решения систем линейных уравнений вида $A\cdot x=b$ с точность 0.01, где

$A=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ . & . & . & . \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$ - матрица коэффициентов при неизвестных переменных;

$b=\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$ - вектор столбец свободных членов; $x=\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$ - вектор столбец неизвестных.

В данном методе последовательно уточняются компоненты решения. Систему приводят к виду:

$$X_1^{'}=(b_1-a_{11}\cdot X_1^0-a_{12}\cdot X_2^0-\dots-a_{1n}\cdot X_n^0)/a_{11}+X_1^0$$
$$X_2^{'}=(b_2-a_{12}\cdot X_1^{'}-a_{22}\cdot X_2^0-\dots-a_{2n}\cdot X_n^0)/a_{22}+X_2^0$$
. . . . . .
$$X_n^{'}=(b_n-a_{n1}\cdot X_1^{'}-a_{12}\cdot X_2^{'}-\dots-a_{nn}\cdot X_n^0)/a_{nn}+X_n^0$$

Выражение в скобках в идеале должно быть равно нулю. На диагональные элементы делим, т.к. определяем соответствующие неизвестные. В результате, например, вновь уточняемое $X_1^{'}$ должно быть приблизительно равно предыдущему X_1^0 (как следствие большого количества итераций).

По шагам:

а) $X_1^{'} = b_1$ (остальные = 0 на данном шаге);

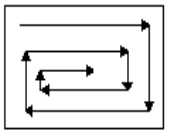
$X_2^{'} = b_2 - a_{21} \cdot X_1^{'}$ (остальные =0, а $X_1^{'}$ подставляем из 1-го уравнения);

и т.д. до $X_n^{'}$.

б) На второй итерации все повторяется.

и т.д. до тех пор , пока не достигнем заданной точности. (программа - v13. dpr)

14. Написать программу, которая позволяет заполнять двумерный массив целыми двузначными числами по спирали следующим образом:



(программа - v14. dpr)

2.4 Вопросы для самопроверки

1. Как повысить технологичность программы?
2. Перечислите способы повышения эффективности программы?
3. Перечислите критерии оценки эффективности?
4. Перечислите способы уменьшения времени выполнения?
5. Как замерить время выполнения программы?
6. Какие факторы искажают результаты замеров?
7. Какой способ наиболее существенно может повысить эффективность?
8. Что значит проверить на правильность?
9. Как оценить универсальность программы?
10. Что значит проверить на точность результатов?
11. Как оценить проверяемость?
12. Какие имеются способы экономии памяти?
13. Что такое контрольные точки?
14. Как влияет конфигурация оборудования на время выполнения?

Глава 3. Тестирование программного обеспечения

Одним из наиболее трудоемких этапов (от 30 до 60% общей трудоемкости) создания программного продукта является тестирование. Причем доля стоимости тестирования в общей стоимости разработки имеет тенденцию возрастать при увеличении сложности комплексов программ и повышении требований к их качеству. В связи с этим большое внимание уделяется выбору стратегии и методов тестирования, что не является тривиальной задачей.

Таким образом, при подготовке к тестированию необходимо ответить на следующие вопросы:

- Какую стратегию тестирования выбрать и почему? Как ее реализовать?
- Какой из методов выбранной стратегии тестирования выбрать и почему?
- Как грамотно подготовить тестовый набор данных и сколько тестов необходимо разработать?

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.

Продолжительность работы - 4 часа.

3.1 Краткие теоретические сведения

Исходными данными для тестирования являются техническое задание, спецификации, а для некоторых методов тестирования - алгоритм тестирования.

При тестировании рекомендуется соблюдать следующие *основные принципы:*

1. Предполагаемые результаты должны быть известны до тестирования.
2. Следует избегать тестирования программы автором.
3. Необходимо досконально изучать результаты каждого теста.
4. Необходимо проверять действия программы на неверных данных.
5. Необходимо проверять программу на неожиданные побочные эффекты.

6. Удачным считается тест, который обнаруживает хотя бы одну еще не обнаруженную ошибку.

7. Вероятность наличия ошибки в части программы пропорциональна количеству ошибок, уже обнаруженных в этой части.

3.1.1 Ручное тестирование программных продуктов

Методы ручного контроля используются, когда получены исходные коды, но к тестированию на машине еще не приступили. Основными методами ручного тестирования являются: инспекции исходного текста; сквозные просмотры; просмотры за столом; обзоры программ.

Инспекции исходного текста (структурный контроль) с целью обнаружения ошибок осуществляется группой специалистов, в которую входят автор программы, проектировщик, специалист по тестированию и координатор (компетентный программист, но не автор программы). Общая процедура инспекции состоит из следующих этапов: участникам заранее выдается листинг программы и спецификация на нее; программист рассказывает о логике работы программы и отвечает на вопросы инспекторов; программа анализируется по заранее сформированному списку вопросов.

Сквозной просмотр также осуществляется группой лиц, но отличается процедурой и методами обнаружения ошибок. Здесь группа состоит из 3-5 человек (председатель или координатор, секретарь, фиксирующий все ошибки, специалист по тестированию, программист и независимый эксперт). Этапы процедуры сквозного контроля: участникам заранее выдается листинг программы и спецификация на нее; участникам заседания предлагается несколько тестов и тестовые данные подвергаются обработке (мысленно) в соответствии с логикой программы; программисту задаются вопросы о логике проектирования; состояние программы (значения переменных) отслеживается на бумаге или доске.

Третьим методом ручного обнаружения ошибок является проверка исходного текста или сквозные просмотры, выполняемые одним человеком, который

читает текст программы, проверяет его по списку и пропускает через программу тестовые данные. При этом тестирование проводит не автор программы.

Оценка посредством просмотра явно не связана с тестированием. Это метод оценки анонимной программы в терминах ее общего качества, простоты эксплуатации и ясности. Цель этого метода - обеспечить сравнительно объективную оценку и самооценку программистов.

Ниже приведен перечень вопросов для структурного контроля текста.

1. *Обращения к данным.*
   1) *Все ли переменные инициализированы?*
   2) *Не превышены ли максимальные (или реальные) размеры массивов и строк?*
   3) *Не перепутаны ли строки со столбцами при работе с матрицами?*
   4) *Присутствуют ли переменные со сходными именами?*
   5) *Используются ли файлы? Если да, то*
      - *При вводе из файла проверяется ли завершение файла?*
      - *Соответствуют ли типы записываемых и читаемых значений?*
   6) *Использованы ли нетипизированные переменные, открытые массивы, динамическая память? Если да, то*
      - *Соответствуют ли типы переменных при "наложении" формата?*
      - *Не выходят ли индексы за границы массивов?*
2. *Вычисления.*
   1) *Правильно ли записаны выражения (порядок следования операторов)?*
   2) *Корректно ли производятся вычисления неарифметических переменных?*
   3) *Корректно ли выполнены вычисления с переменными различных типов (в том числе с использованием целочисленной арифметики)?*
   4) *Возможно ли переполнение разрядной сетки или ситуация машинного нуля?*
   5) *Соответствуют ли вычисления заданным требованиям точности?*
   6) *Присутствуют ли сравнения переменных различных типов?*
3. *Передачи управления.*
   1) *Будут ли корректно завершены циклы?*
   2) *Будет ли завершена программа?*
   3) *Существуют ли циклы, которые не будут выполняться из-за нарушения условия входа? Корректно ли продолжатся вычисления?*
   4) *Существуют ли поисковые циклы? Корректно ли отрабатываются ситуации "элемент найден" и "элемент не найден"?*

4. *Интерфейс.*
   1) *Соответствуют ли списки параметров и аргументов по порядку, типу, единицам измерения?*
   2) *Не изменяет ли подпрограмма аргументов, которые не должны изменяться?*
   3) *Не происходит ли нарушения области действия глобальных и локальных переменных с одинаковыми именами?*

3.1.2 Тестирование по принципу «белого ящика»

Стратегия тестирования по принципу «белого ящика» (или стратегия тестирования, управляемая логикой программы, т.е. с учетом алгоритма), позволяет проверить внутреннюю структуру программы. В этом случае тестирующий получает тестовые данные путем анализа логики программы.

Считается, что программа проверена полностью, если с помощью тестов удается осуществить выполнение программы по всем возможным маршрутам передач управления. Для оценки сложных программ провести исчерпывающее тестирование практически невозможно.

Стратегия «белого ящика » включает в себя следующие методы тестирования: *покрытие операторов, покрытие решений, покрытие условий, покрытие решений/условий, комбинаторное покрытие условий.*

Критерий покрытия операторов подразумевает выполнение каждого оператора программы, по крайней мере, один раз. Это необходимое, но недостаточное условие для приемлемого тестирования.

Покрытие решений (переходов) предусматривает достаточное число тестов, такое, что каждое решение на этих тестах принимает значение «истина» или «ложь», по крайней мере, один раз. Данный критерий обеспечивает большее количество тестов по сравнению с критерием покрытия операторов.

Критерий покрытия условий более сильный по сравнению с предыдущим. В этом случае записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены, по крайней

мере, один раз. Это покрытие не всегда приводит к выполнению каждого оператора, т.к. при данном критерии требуется, чтобы каждой точке входа управление должно быть передано, по крайней мере, один раз.

Покрытие решений/условий требует составить тесты так, чтобы все возможные результаты каждого условия выполнились, по крайней мере, один раз, все результаты каждого решения выполнились, по крайней мере, один раз и каждой точке входа управление передается, по крайней мере, один раз.

Комбинаторное покрытие условий требует создания такого числа тестов, чтобы все возможные комбинации результатов условий в каждом решении и все точки входа выполнялись, по крайней мере, один раз.

3.1.3 Тестирование по принципу «черного ящика»

Данный вид тестирования еще называют тестирование с управлением по данным. Здесь программа рассматривается как "черный ящик" и тестирование выявляет несоответствие программы спецификации. *Исчерпывающее тестирование* (проверка на всех возможных наборах данных) в больших системах невозможно, поэтому выполняют *"разумное" тестирование*. Для тех же программ, где исполнение команды зависит от предшествующих ей событий, необходимо проверить и все возможные последовательности.

При "разумном" тестировании выбирают наиболее подходящее подмножество данных, которое обеспечить наивысшую вероятность обнаружения ошибок.

Стратегия "черного ящика" включает в себя следующие методы формирования тестовых наборов: *эквивалентное разбиение; анализ граничных значений; анализ причинно-следственных связей; предположение об ошибке*.

Основу эквивалентного разбиения составляют два положения:

- Исходные данные программы необходимо разбить на конечное число классов эквивалентности, так чтобы можно было предположить, что каждый тест, яв-

ляющийся представителем некоторого класса, эквивалентен любому другому тесту этого класса.

- Каждый тест должен включать по возможности максимальное количество различных входных условий, что позволяет минимизировать общее число необходимых тестов.

Разработка тестов методом эквивалентного разбиения осуществляется в два этапа: выделение классов эквивалентности и построение тестов.

Классы эквивалентности выделяются путем выбора каждого входного условия и разбиением его на две или более групп. Для этого используется таблица, состоящая из трех столбцов: *входное условие, правильные классы эквивалентности, неправильные классы эквивалентности.*

Правильные классы включают правильные данные, неправильные классы - неправильные данные. Выделение классов эквивалентности является эвристическим процессом.

Построение тестов включает в себя:

- Назначение каждому классу эквивалентности уникального номера.
- Разработка тестов, каждый из которых покрывает как можно большее число непокрытых классов эквивалентности, до тех пор, пока все правильные классы не будут покрыты (только не общими) тестами.
- Определение тестов, каждый из которых покрывает один и только один из непокрытых неправильных классов эквивалентности, до тех пор, пока все неправильные классы не будут покрыты тестами.

При анализе граничных значений определяются ситуации, возникающие непосредственно на границе, а также выше или ниже границ входных классов эквивалентности. Анализ граничных значений отличается от эквивалентного разбиения следующим:

- Выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных условий осуществляется таким образом, чтобы проверить тестом каждую границу этого класса.

- При разработке тестов рассматриваются не только входные условия (*пространство входов*), но и *пространство результатов*.

Применение метода анализа граничных условий требует наличие знаний предметной области задачи. Можно выделить несколько правил для этого метода:

- Если входной параметр описывает область значений, но необходимо написать тест, проверяющий на границе, а также тесты с неправильными входными данными вблизи этой границы.
- Если входной параметр принадлежит дискретному ряду значений, то нужно построить тесты для минимального и максимального значений ряда, а также для ближайших значений, выходящих за границы ряда.
- Использовать правило 1 для каждого выходного условия. Использовать правило 2 для каждого выходного условия.
- Если вход или выход программы есть упорядоченное множество (например, последовательный файл, линейный список, таблица), то сосредоточить внимание на первом и последнем элементах этого множества и др.

Метод анализа причинно-следственных связей помогает системно выбирать тесты с высокой результативностью. Он дает полезный побочный эффект, позволяя обнаруживать неполноту и неоднозначность исходных спецификаций.

Для использования метода необходимо понимание булевской логики (логических операторов - и, или, не). Построение тестов осуществляется в несколько этапов.

- Спецификация разбивается на «рабочие » участки, для которых создаются таблицы причинно-следственных связей.
- В спецификации определяются множество причин и множество следствий. *Причина* есть отдельное входное условие или класс эквивалентности входных условий. *Следствие* есть выходное условие или преобразование системы. Каждым причине и следствию приписывается отдельный номер.
- На основе анализа семантического (смыслового) содержания специ-

фикации строится таблица истинности, в которой последовательно перебираются все возможные комбинации причин и определяются следствия каждой комбинации причин. Таблица снабжается примечаниями, задающими ограничения и описывающими комбинации причин и/или следствий, которые являются невозможными из-за синтаксических или внешних ограничений. Аналогично, при необходимости строится таблица истинности для класса эквивалентности.

Методом предположения об ошибке может пользоваться программист с большим опытом. Процедура метода предположения об ошибке в значительной степени основана на интуиции и опыте. Основная идея метода состоит в том, чтобы перечислить в некотором списке возможные ошибки или ситуации, в которых они могут появиться, а затем на основе этого списка составить тесты.

При тестировании больших систем могут быть использованы все стратегии тестирования. При этом может быть построен общий алгоритм тестирования с использованием всех методов.

3.2 Порядок выполнения лабораторной работы №3

1. Ознакомьтесь с теоретическими сведениями по стратегиям тестирования.

2. Для своего варианта задания выполните **структурный контроль**, используя перечень вопросов теоретической части. В процессе выполнения заполните таблицу 3.1:

Таблица 3.1 – Таблица тестов для структурного контроля

Номер вопроса	Строки, подлежащие проверке	Результат проверки	Вывод
1.1	4, 8,9	a=0 b=67 c - вводится	Все переменные инициализированы
1.2	-		
и т.д.			

Примечание. Вопросы, которые не актуальны для данной программы можно не фиксировать в таблице.

Сделайте общий вывод о роли структурного контроля в процессе создания программы. Сформулируйте его достоинства и недостатки.

3. Для заданного фрагмента схемы алгоритма подготовьте тесты, используя методы **стратегии "белого ящика"**. Предлагаемые тесты сведите в таблицу 3.2.

Таблица 3.2 – Таблица тестов стратегии «белого ящика»

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат

Сравните тесты, предлагаемые различными методами. Сделайте вывод о роли тестирования с использованием стратегии "белого ящика" и возможностях его применения. Сформулируйте его достоинства и недостатки.

4. Внимательно изучите формулировку своего варианта задачи, подготовьте тесты по методикам **стратегии "черного ящика"**. Предлагаемые тесты сведите в таблицу 3.3.

Таблица 3.3 – Таблица тестов стратегии «черного ящика»

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат	Реакция программы	Вывод

Получите у преподавателя выполняемый модуль программы. 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.

Отчет должен включать:

- название работы и ее цель;

- описания задач или схемы алгоритмов, для которых разрабатываются тесты;
- наборы тестов для каждой из заданных стратегий с пояснениями;
- выводы о том, в каких случаях должен использоваться тот или иной метод и стратегия в целом.

3.3 Варианты заданий

Задача 1.

Реализовать калькулятор, который выполняет два действия «+» и «-» с действительными числами.

Задача 2.

Реализовать калькулятор, который выполняет два действия «+» и «-» с целыми числами.

Задача 3.

Разработать программу решения уравнения $ax^2 + bx + c = 0$, где a, b, c - любые вещественные числа.

Задача 4.

Программа должна определять корни уравнения $y = x^2 - 2$ на заданном интервале и с заданной точностью на основе «Метода хорд».

Задача 5.

Создать программу , которая позволяет в заданном диапазоне и количестве с помощью генератора случайных чисел создавать массив, а затем сортировать его.

Задача 6.

Создать программу , которая решает систему из 3-х линейных алгебраических уравнений с заданной точностью с помощью итерационного метода. Если количество итераций превысит максимально допустимое, то программа должна выдавать сообщение о том, что сходимости нет.

Задача 7.

Программа должна вычислять с заданной точностью значение интеграла функции $F(x) = x^2$ на введенном с клавиатуры интервале от a до b.

Задача 8.

Программа должна вычислять значение интеграла функции $F(x)=\frac{1}{x}$. Исходными данными являются : интервал и количество шагов.

Задача 9.

Написать программу, которая выводит на экран гистограмму. Параметры гистограммы должны вводится с помощью таблицы.

Задача10.

Написать программу, которая реализует секундомер. Пользователю должны выводиться сотые доли секунды, секунды и минуты.

Задача11.

Программа должна строить график функции по заданным в таблице значениям. Обеспечить возможность выбора вида графика : точки отдельно или точки соединены.

Задача12.

Программа должна определять во сколько рублей обойдется поездка на дачу (туда и обратно). Исходными данными являются: расстояние, цена бензина, потребление бензина.

Задача13.

Написать программу, которая определяет доход по вкладу и сумму в конце срока хранения. Исходными данными являются: сумма, срок и процентная ставка.

Задача14.

Написать программу, которая вычисляет по закону Ома ток, напряжение и сопротивление. Исходными данными являются: сопротивление и напряжение, сопротивление и ток , ток и напряжение.

3.4 Вопросы для самопроверки

1. Перечислите методы ручного тестирования?
2. Что необходимо для выполнения структурного контроля?
3. Какими возможностями обладают методы ручного тестирования?
4. Перечислите методы «белого ящика»?
5. Какие методы «белого ящика» дают наибольшее количество тестов?
6. Приведите примеры перекрытий методов «белого ящика»?
7. Какие методы «черного ящика» существуют?
8. Дайте анализ методов «черного ящика» применительно к различным видам данных?
9. Приведите пример возможного плана тестирования применительно к большим программным системам?
10. Какие существуют общие принципы тестирования программ?
11. Какие существуют виды ошибок?
12. Как провести комплексное тестирование большой системы.

Заключение

После выполнения лабораторных работ студенты должны приобрести практические навыки:

- оценки структур данных;
- разработки структур данных;
- оценки методов обработки данных;
- оценки эффективности программ;
- оценки качества программ;
- тестирования программ.

Полученные результаты оценки основных используемых структур данных, алгоритмов работы, а также фрагменты кодов программ, оценочных таблиц и таблиц результатов тестирования должны быть представлены в отчете.

Уровень освоения теоретического материала и правильность предлагаемых решений определяется во время защиты лабораторных работ при наличии отчета.

Список литературы

1. Гагарина Л. Г., Колдаев В. Д., Алгоритмы и структуры данных. - М:Финансы и статистика, 2009 -: 303с.

2. Ахо А.В. , Хопкрофт Д.Э., Ульман Д.Д., Структуры данных и алгоритмы. – М: Вильямс, 2000 - 384 с.

3. Кинг Д., Создание эффективного программного обеспечения. - М:Мир, 1991 – 288с.

3. Вирт Н. Алгоритмы и структуры данных. -М: ДМК Пресс, 2010 -274с.

4. Иванова Г.С. Технология программирования: Учебник. - М: Кнорус, 2011 – 333 с.

5. Иванова Г.С. Программирование. -М.: Кнорус, 2013 - 426с.

6. Майерс Г. Искусство тестирования программ / Пер. с англ. под ред. Б. А. Позина. - М.: Финансы и статистика,1982-176с.,

7. Иванова Г.С. Технология программирования. -М.:КноРус – 2011- 333с.

8. Иванова Г.С., Ничушкина Т.Н., Пугачев Е.К. Методы обработки данных и оценки программ. Методическое пособие по выполнению лабораторных работ по дисциплине «Технология разработки программных систем». Библиотека каф. ИУ6., 2018 – 63с.