Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

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

Выбор алгоритмов обработки данных, тестирование и повышение качества программ

Учебно-методическое пособие

BAUMANPRESS
Москва
ИЗДАТЕЛЬСТВО
МГТУ им. Н. Э. Баумана
2020

УДК 004.021
ББК 32.973-018.2
И20

Издание доступно в электронном виде по адресу:
https://bmstu.press/catalog/item/6756

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

Рекомендовано
Научно-методическим советом МГТУ им. Н.Э. Баумана
в качестве учебно-методического пособия

Иванова, Г. С.

И20 Выбор алгоритмов обработки данных, тестирование и повышение качества программ : учебно-методическое пособие / Г. С. Иванова, Т. Н. Ничушкина, Е. К. Пугачев. — Москва : Издательство МГТУ им. Н. Э. Баумана, 2020. — 65, [3] с. : ил.

ISBN 978-5-7038-5408-2

Представлены 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, порядок выполнения и требования к защите лабораторных работ, предусмотренных учебным планом МГТУ им. Н.Э. Баумана.

Для студентов МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки «Информатика и вычислительная техника».

УДК 004.021
ББК 32.973-018.2

ISBN 978-5-7038-5408-2

Предисловие

Учебно-методическое пособие предназначено для получения практических навыков в области разработки программ студентами, обучающимися на кафедре «Компьютерные системы и сети».

Издание составлено в соответствии с самостоятельно устанавливаемым образовательным стандартом (СУОС), основной образовательной программой по направлению подготовки 09.03.01 «Информатика и вычислительная техника» бакалавров и предназначено для подготовки к выполнению лабораторных работ по дисциплине «Технология разработки программных систем».

Цель учебно-методического пособия — подготовка студентов к лабораторным работам, связанным с приобретением практических навыков выбора структур данных и методов их обработки, а также навыков тестирования и оценки качества программ.

После выполнения лабораторных работ студенты будут:

знать

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

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;

владеть

- 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.

Кроме того, студент получит следующие 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: информатика, основы программирования, объектно-ориентированное программирование, языки интернет-программирования.

Одной из важных задач учебно-методического пособия является приобретение студентами способности разрабатывать компоненты программных систем, используя современные инструментальные средства и технологии программирования, а также осуществлять постановку и выполнять эксперименты по проверке корректности и эффективности программ.

В лабораторной работе 1 рассмотрены способы организации структур данных, оценки применимости различных структур к конкретным задачам, а также способы оценки алгоритмов, реализующие различные методы обработки данных для операций поиска, упорядочения и корректировки данных. Студентам необходимо проработать основной вариант по заданию и после получения результатов предложить альтернативный вариант решения, в котором устранены недостатки основного варианта.

В лабораторной работе 2 исследованы способы оценки эффективности и качества программ. После оценки программы студентам необходимо предложить способы повышения эффективности и качества. Данные улучшения должны быть подтверждены практическими результатами замеров.

В лабораторной работе 3 приведены методы тестирования программных продуктов, которые используются на разных стадиях разработки и позволяют определять широкий спектр ошибок.

В приложениях представлены примеры оформления лабораторных работ.

Издание позволит студентам самостоятельно изучить важные разделы дисциплины «Технология разработки программных систем».

Для успешного освоения материала необходимо внимательно разбирать примеры, целесообразно синтезировать и прорабатывать аналогичные примеры. Ответы на вопросы позволят обучающемуся самостоятельно 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. Лабораторные работы студенты выполняют и защищают в компьютерном классе. Варианты заданий, требования к отчету и защите приведены отдельно по каждой лабораторной работе.

Введение

При создании программных продуктов необходимо знать подходы разработки отдельных компонентов. Можно выделить следующие три вида компонентов: интерфейсные, обрабатывающие и компоненты данных. Эффективная работа любой программной системы зависит от того, насколько удачно был сделан выбор методов для реализации основных операций обрабатывающих компонент. Во многих программных системах часто используются операции поиска, упорядочения и редактирования. Только для вышеуказанных операций существует множество методов их реализации. Задача выбора метода реализации является непростой, так как в каждом отдельном случае необходимо учитывать структуру данных. Удачно принятые решения, связанные с выбором структур данных и методов их обработки, позволяют экономно использовать ресурсы вычислительной системы, например ресурсы оперативной и внешней памяти, а также процессорное время. Чтобы принять удачные решения, необходимо обладать знаниями, связанными с основными факторами, влияющими на выбор структуры данных, которые, в свою очередь, зависят от режима работы разрабатываемой системы, решаемых задач предметной области и др.

Этап реализации программного продукта включает процесс его тестирования. Тестирование сложной программы представляет собой непростой процесс. Владение навыками составления тестов и разработки алгоритмов тестирования программ является важной задачей. Грамотно проведенное тестирование разрабатываемой программной системы обеспечивает уверенность в том, что все компоненты работают корректно и соответствуют предъявленным требованиям.

Также актуальной задачей является оценка качества уже существующего программного продукта, которая связана как с проверкой соответствия программного продукта функциональным требованиям, указанным в техническом задании, так и с оценкой производительности.

Лабораторная работа 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 с 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: определить основные критерии оценки структуры данных и методов ее обработки применительно к конкретной задаче.

Продолжительность работы: 9 часов.

1.1. Краткие теоретические сведения

Структуры данных

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

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 бывает разным, и от этого зависит, 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.

Таблица 1.1

Виды структур абстрактных данных

Тип связности элементов структуры данных	Примеры структур	Описание
Элементы не связаны	Множество	Позволяют хранить ограниченное число значений определенного типа без определенного порядка. Доступ путем перебора элементов (счетный)
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
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

В просмотровых таблицах доступ к записям последовательный, а основные характеристики следующие:

$$S_i = i; \quad L = (N+1)/2,$$

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-й записи; A_1 — адрес первой записи; l — длина записи.

В таблицах с перемешиванием (кэш-таблицах) $L \cong 1$ 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 первом 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.

Методы обработки данных

Методы упорядочения данных. Частными случаями упорядочения являются сортировки (по возрастанию, по убыванию, по невозрастанию, по неубыванию). Понятие «сортировка» обычно применяют к линейным структурам данных. Понятие «упорядочение» используют применительно к нелинейным структурам данных, например к деревьям.

В методе, базирующемся на попарном сравнении соседних элементов, количество сравнений определяют выражением

$$C = N(N-1),$$

где N — количество элементов.

В методе, базирующемся на поиске минимального (максимального) элемента, количество сравнений определяют по формуле

$$C = N(N-1)/2.$$

В методе вставки количество сравнений определяют по формуле

$$C = N(N-1)/4.$$

В методе Шелла количество операций сравнений оценивают следующим образом:

$$C \le 0{,}5N^{3/2}.$$

В методе квадратичной выборки 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 определяют отдельно.

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 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 вычисляют по формуле

$$C = (N+1)/2,$$

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

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

Метод вычисления адреса (адресный поиск) также является не универсальным. Его применяют к упорядоченным массивам, если значения ключевых признаков не повторяются и их количество 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.

К способам, ускоряющим поиск в списковых структурах, относятся:

- вышеописанный способ с использованием адресной функции;
- метод *K*- и *A*-индексов;
- гнездовой способ организации.

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

Если значения ключей записей образуют арифметическую прогрессию, то их называют *K*-индексами. Если значения ключей адреса записей образуют арифметическую прогрессию, то их называют *A*-индексами.

В гнездовом способе множество элементов разбивают 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 осуществляют в два этапа: во вспомогательной структуре с целью выхода на гнездо и 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: он начинается с вершины дерева, далее осуществляется 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.

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 (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 с данными придется проверять, не удалены ли они.

Операция замены возможна без реорганизации массива.

Корректировку можно выполнять двумя способами. В первом случае изменения вносят в основной массив сразу. Во втором случае, применяемом

при очень больших размерах массивов, записи с измененными данными накапливают в специальном массиве изменений. При достижении определенного объема массив изменений объединяют с основным массивом.

Оценка емкостной и временной сложности

При выборе структуры данных очень важными являются результаты оценки объема памяти (*емкостной сложности*), так как от этого зависит эффективная работа программного продукта.

Кроме того, эффективная работа программы зависит от времени выполнения операций (*временной сложности*), реализующих функционал.

Емкостную сложность оценивают по физической структуре данных с учетом конкретной реализации на выбранном языке программирования.

Для оценки временной сложности можно использовать следующую методику:

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

2) полученный результат преобразовать по таблице соответствий (табл. 1.2) ко времени выполнения фрагмента алгоритма в машинных тактах.

Таблица 1.2

Коэффициенты приведения команд Borland Pascal (Pentium)

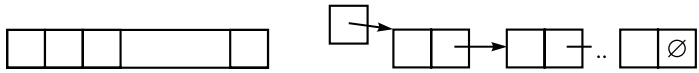
Операция	Обозначение	Время, тактов	Операция	Обозначение	Время, тактов
`a:=b`	$t_{=}$	2	`a/b`	$t/$	28
`c:=a±b, a>b`	t_{+}	2	`a:=b^.num`	$t_{=^{\wedge}}$	2
`A:=a±b`	$t_{=+}$	2^{*}	`b:=b^.p`	$t_{^{\wedge}}$	2^{*}
`a:=a±const`	t_{++}	1	`сдвиги`	$t_{\rightarrow}$	1
`A±const`	t_{+C}	1	`a[i]`	t_{i}	2
`a*b`	t^{*}	20	`a[i,j]`	$t_{i,j}$	26
`If`	t_{if}	$t_{\text{пр}}+1+p_1t_1+$ $+p_2t_2^{**}$	`for`	t_{for}	$t_{\text{уст}}+t_{\text{пр}}+1+$ $+n(t_{\text{тела}}+t_{\text{пр}}+2)^{***}$

* Операции накопления/переадресации (при использовании в цикле коэффициент равен единице).

** $t_{\text{пр}}$ — время проверки условия, p_1, p_2 — вероятности выбора соответствующих ветвей, t_1, t_2 — время выполнения ветвей.

*** $t_{\text{уст}}$ — время установки начального значения индекса, $t_{\text{пр}}$ — время проверки условия, $t_{\text{тела}}$ — время выполнения тела цикла. Для цикла вида `for i:=1 to n do…` получаем $t = 2+2+1+n(t_{\text{тела}}+2+2) = 5+n(t_{\text{тела}}+4)$.

Пример. Оценить память и временную сложность выполнения операций поиска и удаления элемента для двух реализаций хранения последовательности n целых чисел: в виде массива и в виде линейного односвязного списка (рис. 1.1).



а *б*

Рис.1.1. Массив (*а*) и односвязный список (*б*)

Решение.

1. Оценка памяти:

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

• для списка: $V = (l_{эл} + l_{ук})n + l_{ук}$, 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:

• для массива: поиск выполняют по индексу $A_i = A_б + (i-1)2 = A_б - 2 + i \cdot 2 = \text{const} + i \cdot 2$, операцию умножения на 2 выполняют посредством сдвига на 1 разряд влево, откуда $t = t_{++} + t_{\to} = 2$ (такта);

• для списка: доступ к элементу выполняют последовательно `f:=first; for j:=1 to i do f:=f^.p;`

Определяем время $t = 2 + 2 + 1 + (i-1)(t_{\wedge} + 2 + 2) = 5 + 5(i-1)$, минимальное время поиска 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:

• для массива: при удалении i-го элемента выполняют 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 $t = t_{пр} + (t_{++} + t_{\max})/2 = 2 + (1 - 2 + 7n)/2 = 1{,}5 + 3{,}5n$ (тактов);

• 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 анализируемых операций.

1.2. Порядок выполнения лабораторной работы 1

Лабораторную работу необходимо выполнять в представленной ниже последовательности.

1. На основе теоретических сведений выделить критерии оценки структур данных, принципы работы и критерии оценки операций поиска, сортировки и корректировки.

2. В соответствии с вариантом задания (см. табл. 1.2) предложить конкретную схему структуры данных (в задании указана абстрактная структура данных) и способ ее реализации на выбранном языке программирования.

3. Определить качественные критерии оценки (универсальность, тип доступа и др.) полученной на шаге 2 структуры данных с учетом специфики задачи по выданному варианту.

4. Определить количественные критерии оценки полученной на шаге 2 структуры данных: требуемый объем памяти на единицу информации, на структуру данных в целом и др.

5. Провести сравнительный анализ структуры данных, предложенной на шаге 2, на основе оценок, полученных на шаге 3 и шаге 4, с другими возможными вариантами реализации с целью поиска лучшей структуры данных к заданию по варианту.

6. Если цель шага 5 достигнута, то необходимо выполнить шаг 2, но для новой абстрактной структуры данных с указанием качественных и количественных критериев.

7. Оценить применимость метода поиска, который указан в варианте задания, с учетом структуры данных.

8. Если метод поиска применим, то необходимо сформулировать его достоинства и недостатки, используя качественные и количественные критерии: универсальность, требуемые ресурсы для реализации, среднее количество сравнений, время выполнения (такты) и др.

9. Предложить альтернативный, более эффективный метод поиска (отличный от задания), если такой существует, с учетом специфики задачи по варианту, а также с учетом структур данных, полученных на предыдущих шагах. Для обоснования выбора альтернативного метода поиска использовать качественные и количественные критерии.

10. Оценить применимость метода упорядочивания, который указан в варианте задания, с учетом структуры данных.

11. Если метод упорядочивания применим, то необходимо сформулировать его достоинства и недостатки, используя качественные

и количественные критерии: универсальность, требуемые ресурсы для реализации, среднее количество сравнений, время выполнения (такты) и др.

12. Предложить альтернативный метод упорядочивания, более эффективный и отличный от задания, если такой существует. При этом должны учитываться задача по варианту и структура данных. Для обоснования выбора альтернативного метода упорядочивания использовать качественные и количественные критерии.

13. Оценить применимость метода корректировки, который указан в задании, к структуре данных.

14. Если метод корректировки применим, то необходимо сформулировать его достоинства и недостатки, используя качественные и количественные критерии: универсальность, требуемые ресурсы для реализации, время выполнения (такты) и др.

15. Предложить альтернативный способ корректировки, более эффективный и отличный от задания, если такой существует. При этом должны учитываться задача по варианту, структура данных. Для обоснования выбора альтернативного способа корректировки использовать качественные и количественные критерии.

16. Определить влияние метода корректировки на выполнение операций поиска и упорядочивания.

17. Определить основной режим работы программы и с учетом этого сделать выводы, а итоговые полученные результаты внести в табл. 1.3. Из данной таблицы должно следовать, что предложенный альтернативный вариант решения задачи лучше. Как минимум должно быть одно улучшение, но могут быть заменены все методы обработки и сама структура данных.

1.3. Требования к отчету

Отчет должен включать:

- титульный лист;
- название работы и ее цель;
- конкретный вариант задания;
- схему структуры данных в соответствии с вариантом задания (см. рис. 1.1);
- 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.

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

Результаты выполнения лабораторной работы необходимо записать в табл. 1.3.

Таблица 1.3

Таблица результатов

Вариант	Структура данных	Метод поиска	Метод упорядочения (сортировки)	Метод корректировки
Основной	Наименование и критерии	Наименование и критерии	Наименование и критерии	Наименование и критерии
Альтернативный	Наименование и критерии	Наименование и критерии	Наименование и критерии	Наименование и критерии

1.4. Требования к защите

К защите лабораторной работы студент готовит по своему отчету краткий доклад, на основе которого должен четко изложить следующие результаты и принятые решения:

- результаты проработки и особенности варианта задания;
- обоснование способа реализации структуры данных по варианту задания, а также его достоинства и недостатки;
- обоснование альтернативной структуры данных в сравнении с исходной структурой (при этом должны использоваться как количественные критерии, так и качественные);
- результаты сравнительной оценки метода поиска по заданию с альтернативным методом, с учетом предметной области задачи и структур данных;
- результаты анализа и оценки исходных кодов реализации метода поиска по заданию и альтернативного способа;
- критерии оценки метода сортировки по заданию и критерии оценки предлагаемого метода;
- результаты анализа кодов реализации: метода сортировки по заданию и альтернативного метода;
- результаты сравнительной оценки операции редактирования. Например, метода удаления данных из заданной структуры в сравнении с предлагаемым методом удаления данных из альтернативной структуры;
- четко сформулированные требования к режиму работы программы, при котором альтернативный вариант однозначно предпочтительнее варианта по заданию.

1.5. Варианты заданий

Варианты заданий для лабораторной работы 1 представлены в табл. 1.4.

Таблица 1.4

Варианты заданий

Номер варианта	Номер задачи	Структура данных	Метод		
			поиска	упорядочения	корректировки
1	1	Таблица	Последовательный	Пузырьком	Удаление сдвигом
2	1	То же	Дихотомический	Вставкой	Удаление маркировкой
3	1	»	Вычисление адреса	Шелла	Замена данных
4	1	»	Последовательный	Квадратичной выборки	Удаление сдвигом
5	2	»	Дихотомический	Слиянием	Вставка записи
6	2	»	Последовательный	Пузырьком	Удаление маркировкой
7	2	»	Дихотомический	Вставкой	Удаление сдвигом
8	2	»	Вычисление адреса	Шелла	Замена данных
9	2	»	Последовательный	Квадратичной выборки	Удаление сдвигом
10	2	»	Дихотомический	Слиянием	Удаление маркировкой
11	2	»	Вычисление адреса	Пузырьком	Удаление сдвигом
12	3	»	Дихотомический	Вставкой	Удаление маркировкой
13	3	»	Вычисление адреса	Шелла	Замена данных
14	3	»	Последовательный	Квадратичной выборки	Удаление сдвигом
15	3	»	Дихотомический	Слиянием	Удаление маркировкой
16	4	Список	Гнездовой	Любой	Удаление записи
17	4	То же	Последовательный	То же	То же
18	4	Кольцевой список	То же	»	»
19	4	Двусвязный список	»	»	Замена данных
20	5	Список	Гнездовой	»	Удаление записи
21	5	То же	Последовательный	»	То же
22	5	Кольцевой список	То же	»	»

Окончание табл. 1.4

Номер варианта	Номер задачи	Структура данных	Метод		
			поиска	упорядочения	корректировки
23	5	Двусвязный список	»	»	»
24	6	Список	Гнездовой	»	»
25	6	То же	Последовательный	»	»
26	6	Кольцевой список	То же	»	»
27	6	Двусвязный список	»	»	»
28	7	Нелинейный двусвязный список	»	»	»
29	7	Дерево	Гнездовой	»	»
30	6	То же	То же	»	»
31	8	Дерево (алгоритм А)	Последовательный	Нет	Исключения неполной вершины
32	8	Дерево (алгоритм Б)	То же	То же	То же
33	9	Дерево (алгоритм А)	»	»	»
34	9	Дерево (алгоритм Б)	»	»	Исключения полной вершины
35	10	Дерево (алгоритм А)	»	»	Исключения неполной и полной вершин

Ниже приведены задачи, применительно к которым в соответствии с вариантом, указанным в табл. 1.4, необходимо выполнить лабораторную работу.

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.

Вопросы для самоконтроля

1. Применим ли метод двоичного поиска к спискам?
2. Как оценить гнездовой метод поиска?
3. Каковы достоинства и недостатки двусвязного списка по отношению к односвязному?
4. Какие количественные оценки применяют к методам сортировки?
5. Какие недостатки у дихотомического метода по сравнению с последовательным?
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?

Лабораторная работа 2.
ОЦЕНКА ЭФФЕКТИВНОСТИ И КАЧЕСТВА ПРОГРАММ

В настоящее время перед разработчиками программного обеспечения стоит задача создания эффективных, технологичных и качественных программ. Эта 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 основные критерии оценки и способы повышения эффективности и качества программных продуктов.

Продолжительность работы: 4 часа.

2.1. Краткие теоретические сведения

Можно выделить следующие основные характеристики, используемые при оценке качества программных продуктов: технологичность, 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 — дефицитный ресурс;
- объем внешней памяти. Оценивают, 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.

1. Следует выбирать алгоритмы обработки, не требующие дублирования исходных данных структурных типов. Например, метод вставки не требует дополнительных массивов.

2. По возможности использовать динамическую память. При необходимости выделять память, а потом освобождать.

3. 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. Выделить из теоретического материала критерии оценки качества программ, способы замеров времени работы выделенного фрагмента кода программы, повышения эффективности использования памяти, уменьшения времени работы процессора, оценки правильности, повышения универсальности программ и повышения проверяемости результатов работы программ.

2. Оценить, функционирует ли исходная программа по выданному варианту в соответствии с заданием.

3. Если исходная программа работает не по заданию, то исправить ошибки и оценить трудоемкость их исправления.

4. Замерить время работы исходной программы.

5. Определить эффективность использования ресурсов оперативной и внешней памяти (получить качественные и количественные оценки ресурсов памяти для исходной программы).

6. Предложить способ повышения эффективности по ресурсам памяти, если такие имеются, исправить исходный код программы, а также получить качественные и количественные оценки ресурсов памяти для улучшенной программы.

7. Замерить время работы улучшенной программы, чтобы выяснить, как повлияло повышение эффективности по ресурсам на время работы.

8. Определить участки кода исходной программы, которые можно модифицировать с целью уменьшения времени ее работы.

9. Предложить способы по дальнейшему улучшению исходного кода, но уже с целью уменьшения времени работы программы.

10. Если код был изменен, то замерить время работы программы.

11. Оценить универсальность исходной программы и предложить способы ее повышения.

12. Оценить проверяемость исходной программы и предложить способы ее повышения.

13. Оценить точность результатов работы исходной программы и предложить способы ее повышения.

14. Составить отчет.

2.3. Требования к отчету

Отчет должен включать:

- титульный лист;
- 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);
- выводы.

Таблица 2.1

Оценка эффективности

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

Таблица 2.2

Оценка качества

Результаты оценки	Критерии оценки			
	Правильность	Универсальность	Проверяемость	Точность результатов
Недостатки				
Оценка, баллы				

2.4. Требования к защите

К защите лабораторной работы студент должен четко сформулировать конкретные результаты по своему варианту задания.

На защите первой части лабораторной работы, касающейся оценки эффективности программы, студент должен:

- показать, как изменить код программы с целью повышения эффективности использования оперативной памяти;

• подтвердить экспериментально повышение эффективности использования оперативной памяти;

• показать, какие участки кода необходимо изменить с целью повышения эффективности по времени работы;

• перечислить критерии, на основе которых можно повысить эффективность по времени;

• перечислить способы замеров времени работы как программы в целом, так и отдельных участков.

При защите второй части лабораторной работы, касающейся оценки качества программы, студент должен:

• изложить результаты оценки правильности работы программы, а при отрицательных результатах оценки определить трудоемкость исправления программы;

• обосновать результаты оценки универсальности и четко сформулировать критерии, позволяющие повысить универсальность программы, а также оценить трудоемкость повышения универсальности;

• изложить результаты оценки проверяемости и предложить способы ее повышения;

• сделать выводы по оценке точности результатов работы программы, а в случае отрицательного заключения предложить способ повышения точности.

2.5. Варианты заданий

Ниже приведены задания, в соответствии с которыми должны работать программы, имена которых для каждого задания указаны в скобках.

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 (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: $1 + n$, $2 + n - 1$, ..., $i + n - i + 1$, $\ldots n/2 + n/2 + 1$ (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. $\int_a^b f(x)dx \approx \frac{h}{3}(f_0 + f_n + 4(f_1 + f_3 + ... + f_{n-1}) + 2(f_2 + f_4 + ... + f_{n-2}))$ (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 генерируется 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 $S = \frac{1}{4} - \frac{1}{16} + \frac{1}{96} - ... ...(-1)^{n+1}\frac{1}{4^n \cdot n!}$ с точностью 0,0001. Ряд сходится и имеет множитель $m = -\frac{1}{4 \cdot n}$ (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 $Ax = b$ с точностью 0,01, где

$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ . & . & . & . \\ a_{n1} & a_{n2} & ... & a_{nn} \end{bmatrix}$ — матрица коэффициентов при неизвестных переменных; $\vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_n \end{bmatrix}$ — вектор-столбец свободных членов; $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix}$ — вектор-столбец неизвестных.

Согласно методу Зейделя, последовательно уточняют компоненты решения. Систему приводят к виду:

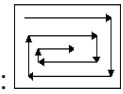
$$X_1' = \left(b_1 - a_{11}X_1^0 - a_{12}X_2^0 - ... - a_{1n}X_n^0\right)/a_{11} + X_1^0$$
$$X_2' = \left(b_2 - a_{12}X_1^0 - a_{22}X_2^0 - ... - a_{2n}X_n^0\right)/a_{22} + X_2^0$$
$$.....$$
$$X_n' = \left(b_n - a_{n1}X_1' - a_{12}X_2' - ... - a_{nn}X_n^0\right)/a_{nn} + X_n^0$$

Выражение в скобках в идеале должно быть равно нулю. Деление на диагональные элементы выполняют, чтобы определить соответствующие неизвестные. В результате, например, вновь уточняемое X_1' должно быть приблизительно равно предыдущему X_1^0 (как следствие большого количества итераций).

По шагам:

• $X_1' = b_1$ (остальные = 0 на данном шаге); $X_2' = b_2 - a_{21}X_1'$ (остальные равны нулю, а X_1' подставляем из первого уравнения) и так далее до X_n';

• на второй итерации все повторяется до тех пор, пока не будет достигнута заданная точность (программа v13.dpr).

14. Написать программу, которая позволяет заполнять двумерный массив целыми двузначными числами по спирали следующим образом:  (программа v14.dpr)

15. Написать программу, которая формирует случайным образом список вещественных чисел и обеспечивает возможность вывода элементов списка в любом диапазоне (программа v15.dpr).

16. Написать программу, которая создает трехмерный массив целых чисел и выводит его на экран с пометкой минимальных и максимальных элементов (программа v16.dpr).

17. Написать программу, которая создает случайным образом одномерный массив вещественных чисел, а затем — одномерный массив целых чисел на основе элементов первого массива путем преобразования вещественных чисел в целые, отбрасывая дробную часть (программа v17.dpr).

18. Написать программу, которая строит график случайных целых чисел. Обеспечить два режима построения графика: случайные числа не отсортированы и отсортированы (программа v18.dpr).

19. Написать программу, которая генерирует двумерный массив целых чисел в диапазоне от –20 до 20 и сортирует его так, чтобы все элементы i+1-й строки были больше элементов i-й строки, а j+1-й элемент i-й строки больше j-го элемента (программа v19. dpr).

20. Написать программу, которая генерирует массив латинских прописных букв и сортирует его по убыванию (программа v20.dpr).

21. Вычислить значение интеграла с точностью 0,01 с помощью метода прямоугольников для функции $Y = x + 5$ (программа v21.dpr).

22. Программа должна генерировать массив целых чисел в диапазоне от –20 до 30 и определять максимальный элемент (программа v22.dpr).

23. Программа должна генерировать случайным образом строку, состоящую из слов разной длины, слова формировать из латинских букв и определять слово максимальной длины (программа v23.dpr).

24. Программа должна генерировать матрицу размером $N \times M$ целых чисел в диапазоне от 10 до 90 и определять сумму элементов главной диагонали (программа v24.dpr).

25. Написать программу, которая создает случайным образом массив записей. Каждая запись имеет поля: год, месяц, день и описание события. Программа должна сортировать по полю «дата» (программа v25.dpr).

Вопросы для самоконтроля

1. Как повысить технологичность программы?
2. Каковы способы повышения эффективности программы?
3. Какие критерии оценки эффективности вы знаете?
4. Каковы 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?

Лабораторная работа 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: приобрести навыки тестирования схем алгоритмов, исходных кодов программ и исполняемых модулей.

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:

- предполагаемые результаты должны быть известны до тестирования;
- следует избегать тестирования программы автором;
- необходимо досконально изучать результаты каждого теста;
- необходимо проверять действия программы на неверных данных;
- необходимо проверять программу на неожиданные побочные эффекты;
- удачным считается тест, который обнаруживает хотя бы одну еще не обнаруженную ошибку;
- вероятность наличия ошибки в части программы пропорциональна количеству ошибок, уже обнаруженных в этой части.

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 осуществляются 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 выдают листинг программы и спецификацию на нее;
- программист рассказывает о логике работы программы и отвечает на вопросы инспекторов;
- программу анализируют 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 выдают листинг программы и спецификацию на нее;
- участникам заседания предлагают несколько тестов, и тестовые данные подвергаются обработке (мысленно) в соответствии с логикой программы;
- программисту задают вопросы о логике проектирования;
- состояние программы (значения переменных) отслеживается на бумаге или доске.

Следующим 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.

1.1. Все ли переменные инициализированы?

1.2. Не превышены ли максимальные (или реальные) размеры массивов и строк?

1.3. Не перепутаны ли строки со столбцами при работе с матрицами?

1.4. Присутствуют ли переменные со сходными именами?

1.5. Используются ли файлы? Если да, то

- При вводе из файла проверяется ли завершение файла?
- Соответствуют ли типы записываемых и читаемых значений?

1.6. 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.

2.1. Правильно ли записаны выражения (порядок следования операторов)?

2.2. Корректно ли производятся вычисления неарифметических переменных?

2.3. Корректно ли выполнены вычисления с переменными различных типов (в том числе с использованием целочисленной арифметики)?

2.4. Возможны ли переполнение разрядной сетки или ситуация машинного нуля?

2.5. Соответствуют ли вычисления заданным требованиям точности?

2.6. Присутствуют ли сравнения переменных различных типов?

3. Передачи управления.

3.1. Будут ли корректно завершены циклы?

3.2. Будет ли завершена программа?

3.3. Существуют ли циклы, которые не будут выполняться из-за нарушения условия входа? Корректно ли продолжатся вычисления?

3.4. Существуют ли поисковые циклы? Корректно ли отрабатываются ситуации «элемент найден» и «элемент не найден»?

4. Интерфейс.

4.1. Соответствуют ли списки параметров и аргументов по порядку, типу, единицам измерения?

4.2. Не изменяет ли подпрограмма аргументов, которые не должны изменяться?

4.3. Не происходит ли нарушения области действия глобальных и локальных переменных с одинаковыми именами?

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 1 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 1 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 1 раз. Это покрытие не всегда приводит к выполнению каждого оператора, так как при данном критерии требуется, чтобы каждой точке входа управление было передано по крайней мере 1 раз.

Покрытие решений/условий требует составить тесты так, чтобы все возможные результаты каждого условия выполнились по крайней мере 1 раз, все результаты каждого решения выполнились по крайней мере 1 раз и каждой точке входа управление передается по крайней мере 1 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 1 раз.

Тестирование по принципу «черного ящика»

Этот вид тестирования еще называют тестированием 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 программ, где исполнение команды зависит от предшествующих ей событий, необходимо проверить и все возможные последовательности.

При «разумном» тестировании выбирают наиболее подходящее подмножество данных, которое обеспечит 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, являющийся 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;

• разработку 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:

1) если входной параметр описывает область значений, то необходимо написать тест, проверяющий на границе, а также тесты с неправильными входными данными вблизи этой границы;

2) если входной параметр принадлежит дискретному ряду значений, то следует построить тесты для минимального и максимального значений ряда, а также для ближайших значений, выходящих за границы ряда;

3) использовать правило 1 для каждого выходного условия. Использовать правило 2 для каждого выходного условия;

4) если вход или выход программы есть упорядоченное множество (например, последовательный файл, линейный список, таблица), то нужно сосредоточить внимание на первом и последнем элементах этого множества.

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 спецификации 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. Главная 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 применением всех методов.

3.2. Порядок выполнения лабораторной работы 3

1. Ознакомиться с теоретическими сведениями по стратегиям тестирования.

2. Для заданного варианта выполнить структурный контроль, используя перечень вопросов теоретической части. Заполнить таблицу, пример которой с результатами тестирования представлен в табл. 3.1.

Сделать общий вывод о роли структурного контроля в процессе создания программы. Сформулировать его достоинства и недостатки.

3. Для заданного фрагмента схемы алгоритма подготовить тесты, используя методы стратегии «белого ящика». Предлагаемые тесты записать в табл. 3.2.

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

Таблица 3.2

Таблица тестов стратегии «белого ящика»

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат

Сравнить тесты, предлагаемые различными методами. Сделать вывод о роли тестирования с использованием стратегии «белого ящика» и возможностях его применения. Сформулировать его достоинства и недостатки.

4. Внимательно изучить формулировку своего варианта задачи, подготовить тесты по методикам стратегии «черного ящика». Предлагаемые тесты записать в табл. 3.3.

Таблица 3.3

Таблица тестов стратегии «черного ящика»

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

Получить у преподавателя выполняемый модуль программы. Выполнить тестирование. Занести в таблицу результаты.

Сделать вывод о роли тестирования с использованием стратегии «черного ящика» и возможностях его применения. Сформулировать его достоинства и недостатки.

3.3. Требования к отчету

Отчет должен включать:

- титульный лист;
- название работы и ее цель;
- описания задач и схемы алгоритмов, для которых разрабатываются тесты;
- наборы тестов в виде таблиц для каждой из заданных стратегий с пояснениями;
- выводы о возможностях как каждого метода тестирования отдельно, так и группы методов в целом.

3.4. Требования к защите

Студент должен четко понимать возможности каждого метода тестирования. Это касается их использования на различных этапах жизненного цикла, а также возможностей нахождения различных видов ошибок каждым методом в отдельности. Кроме вышесказанного студенты должны оценивать трудоемкость использования методов тестирования, а также знать требования к уровню квалификации разработчиков, способных проводить тестирование.

При защите первой части лабораторной работы, связанной с освоением метода структурного контроля, студент должен:

- знать основной перечень вопросов, используемых при проверке исходного кода программы;
- дать формулировки по всем найденным ошибкам;
- предложить способы исправления ошибок.

При защите второй части лабораторной работы, связанной с освоением методов стратегии «белого ящика», студент должен:

- показать на тестах умение проверять выполнение программы по всем возможным маршрутам передач управления;
- продемонстрировать навыки составления тестов по всем методам тестирования: покрытие операторов, покрытие решений, покрытие условий и др.

При защите третьей части лабораторной работы, связанной с освоением методов стратегии «черного ящика», студент должен:

- знать методы формирования тестовых наборов: эквивалентное разбиение; анализ граничных значений и др.;
- показать умение составлять тесты, находить ошибки и давать формулировки по обнаруженным ошибкам.

3.5. Варианты заданий

Варианты заданий для структурного контроля

Ниже приведены задания, в соответствии с которыми должны работать программы, исходные коды которых предоставляются студентам.

1. Программа должна читать текстовый файл, выбирать слова, в которых более четырех букв, и записывать их в другой текстовый файл (файл исходного кода v1.doc).

2. Программа должна создавать динамический односвязный список вещественных чисел и проверять на совпадение первой половины списка со второй (файл исходного кода v2.doc).

3. Программа должна формировать массив чисел от 3 до 25 , а затем сортировать элементы массива по возрастанию и исключать повторяющиеся элементы (файл исходного кода v3.doc).

4. Программа должна генерировать массив чисел и сортировать данные методом вставки (файл исходного кода v4.doc).

5. Написать программу, которая выводит в диапазоне от 10 до 100 все простые числа и их количество. Затем проверяет, является ли результат произведения двух простых чисел натуральным числом, путем случайного выбора (файл исходного кода v5.doc).

6. Составить программу построения таблицы arcsin x: arcsin $x = x + \frac{x^3}{2\cdot 3} + \frac{3x^5}{2\cdot 4\cdot 5} + \frac{3\cdot 5x^7}{2\cdot 4\cdot 6\cdot 7} + \ldots$ для $0 \le x \le 1$, $h = 0{,}05$, $\varepsilon = 0{,}00001$ (файл исходного кода v6.doc).

7. Размер ренты в расчете на 1$ определяется следующим образом:

$$S = \frac{(1+r)^n - 1}{r},$$

где r — норма процента; n — количество периодов времени.

Определить r при заданных S и n (файл исходного кода v7.doc).

8. Программа должна читать текстовый файл, в котором на каждой строке находится фамилия, и строить динамический сортированный список (файл исходного кода v8.doc).

9. Составить программу преобразования строки путем добавления пробела после каждого пятого символа строки, используя процедуру (файл исходного кода v9.doc).

10. Программа должна читать типизированный файл и выводить данные на экран (файл исходного кода v10.doc).

11. Программа должна формировать типизированный файл с информацией о фамилии человека, дне рождения, а также осуществлять поиск в файле информации о дне рождения (файл исходного кода v11.doc).

12. Написать программу, которая генерирует числа в диапазоне от −25 до 25 и сортирует их по убыванию методом Шелла (файл исходного кода v12.doc).

13. Программа должна решать систему линейных уравнений прямым методом Гаусса (файл исходного кода v13.doc).

14. Программа должна проверять строку на предмет чередования в ней цифр и букв (файл исходного кода v14.doc).

15. Программа должна определять корни уравнений с помощью метода хорд (файл исходного кода v15.doc).

16. Программа должна генерировать двумерный массив вещественных чисел в диапазоне от −100 до 200 и определять позицию слова, если оно входит в созданный массив (файл исходного кода v16.doc).

17. Программа должна генерировать случайным образом два одномерных массива вещественных чисел. Размеры массивов пользователь вводит с клавиатуры. Диапазон генерируемых чисел от −10 до 10. Далее программа должна выводить и подсчитывать элементы, которые являются общими (файл исходного кода v17.doc).

18. Программа должна давать пользователю возможность работать с текстовым файлом и выполнять следующие функции: прочитать файл, добавить информацию и создать файл заново (файл исходного кода v18.doc).

19. Программа должна генерировать двумерный массив, размер которого вводится с клавиатуры. Диапазон генерируемых чисел от −10 до 10. Также программа должна удалять отрицательные числа, но только в нечетных столбцах (файл исходного кода v19.doc).

20. Программа должна исправлять строку, если пользователь забыл переключить с английского алфавита на русский (файл исходного кода v20.doc).

21. Программа должна анализировать строку на основе рекурсивного алгоритма. При анализе программа должна проверять, чередуются цифры с буквами или нет (файл исходного кода v21.doc).

22. Программа должна сортировать одномерный массив по убыванию методом слияния (файл исходного кода v22.doc).

23. Программа должна генерировать массив неповторяющихся чисел, сортировать по возрастанию и определять количество вхождений элемента, который пользователь ввел с клавиатуры (файл исходного кода v23.doc).

24. Программа должна генерировать массив двузначных целых чисел, сортировать элементы по неубыванию, удалять элемент массива, если он существует (если элементов несколько, то удалять все). Элемент массива пользователь вводит с клавиатуры (файл исходного кода v24.doc).

25. Программа должна генерировать случайным образом слова в количестве N и удалять слово из строки, если оно существует и начинается на гласную букву (файл исходного кода v25.doc).

Варианты заданий для метода «черного ящика»

Ниже приведены задания, в соответствии с которыми должны работать исполняемые модули, имена которых указаны в скобках.

1. Реализовать калькулятор, который выполняет два действия «+» и «–» с действительными числами (исполняемый модуль v1.exe).

2. Реализовать калькулятор, который выполняет два действия «+» и «–» с целыми числами (исполняемый модуль v2.exe).

3. Разработать программу решения уравнения $ax^2 + bx + c = 0$, где a, b, c — любые вещественные числа (исполняемый модуль v3.exe).

4. Программа должна определять корни уравнения $y = x^2 - 2$ на заданном интервале и с заданной точностью на основе «Метода хорд» (исполняемый модуль v4.exe).

5. Создать программу, которая позволяет в заданном диапазоне и количестве с помощью генератора случайных чисел создавать массив, а затем сортировать его (исполняемый модуль v5.exe).

6. Создать программу, которая решает систему из трех 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 (исполняемый модуль v6.exe).

7. Программа должна вычислять с заданной точностью значение интеграла функции $F(x) = x^2$ на введенном с клавиатуры интервале от a до b (исполняемый модуль v7.exe).

8. Программа должна вычислять значение интеграла функции $F(x) = \frac{1}{x}$. Исходными данными являются интервал и количество шагов (исполняемый модуль v8.exe).

9. Написать программу, которая выводит на экран гистограмму. Параметры гистограммы должны вводиться с помощью таблицы (исполняемый модуль v9.exe).

10. Написать программу, которая реализует секундомер. Пользователю должны выводиться сотые доли секунды, секунды и минуты (исполняемый модуль v10.exe).

11. Программа должна строить график функции по заданным в таблице значениям. Обеспечить возможность выбора вида графика: точки отдельно или точки соединены (исполняемый модуль v11.exe).

12. Программа должна определять, во сколько рублей обойдется поездка на дачу (туда и обратно). Исходными данными являются: расстояние, цена бензина, потребление бензина (исполняемый модуль v12.exe).

13. Написать программу, которая определяет доход по вкладу и сумму в конце срока хранения. Исходными данными являются: сумма, срок и процентная ставка (исполняемый модуль v13.exe).

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

и напряжение, сопротивление и ток, ток и напряжение (исполняемый модуль v14.exe).

15. Написать программу, которая позволяет управлять перемещением шара. Шар должен перемещаться в пределах выделенной зоны. Обеспечить работу программы в двух режимах: с отражением шара от границ зоны и без отражения. Параметрами управления являются: смещение по оси *X*, по оси *Y* (исполняемый модуль v15.exe).

16. Написать программу, которая генерирует массив целых двузначных чисел в заданном количестве и подсчитывает количество чисел, кратных величине, которую задает пользователь (исполняемый модуль v16.exe).

17. Написать программу, которая генерирует массив целых чисел в интервале от −500 до 500 в количестве до 200 элементов. Программа должна подсчитывать количество чисел в заданном пользователем диапазоне (исполняемый модуль v17.exe).

18. Написать программу, которая генерирует два массива целых чисел в заданном количестве и подсчитывает количество совпадающих чисел (исполняемый модуль v18.exe).

19. Написать программу, которая позволяет управлять перемещением двух шаров. Шары должны перемещаться в пределах выделенной зоны, отражаться от стенок зоны и друг друга (исполняемый модуль v19.exe).

20. Программа должна вычислять с заданной точностью значение интеграла функций $F(x)=x$, $F(x)=x^2/2$, $F(x)=x^2/(1-x)$ на введенном с клавиатуры интервале от *a* до *b* (исполняемый модуль v20.exe).

21. Написать программу, которая реализует секундомер с возможностью задания конечных значений. Должны выводиться секунды и сотые доли секунды (исполняемый модуль v21.exe).

22. Написать программу, которая визуально демонстрирует работу метода сортировки вставкой, сортирует по неубыванию, выдает комментарии о каждом шаге и имеет возможность приостанавливать процесс сортировки и начать заново (исполняемый модуль v22.exe).

23. Написать программу, которая визуально демонстрирует работу метода сортировки пузырьком, сортирует по возрастанию, выдает комментарии о каждом шаге, имеет возможность приостанавливать процесс сортировки, начать процесс заново, по желанию пользователя выполнять следующий шаг (исполняемый модуль v23.exe).

24. Написать программу, которая случайным образом генерирует строку, состоящую из прописных букв латинского алфавита, и подсчитывает количество гласных и согласных букв. Входными параметрами являются: минимальная, максимальная длина слов и количество слов в строке (исполняемый модуль v24.exe).

25. Написать программу, которая генерирует строку случайным образом и определяет самое короткое и самое длинное слово. Входным параметром является количество слов в строке (исполняемый модуль — v25.exe).

Вопросы для самоконтроля

1. Какие методы ручного тестирования вы знаете?
2. Что необходимо для выполнения структурного контроля?
3. Какими возможностями обладают методы ручного тестирования?
4. Каковы методы «белого ящика»?
5. Какие методы «белого ящика» дают наибольшее количество тестов?
6. Какие примеры перекрытий методов «белого ящика» вы можете привести?
7. Какие методы «черного ящика» существуют?
8. Какие методы «черного ящика» применимы к обычным данным, а какие к управляющим данным?
9. Какой пример возможного плана тестирования применительно к большим программным системам вы можете привести?
10. Какие существуют общие принципы тестирования программ?
11. Какие существуют виды ошибок?
12. Как провести комплексное тестирование большой системы?

Литература

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

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

Иванова Г.С., *Пугачев Е.К.* Оценка методов обработки данных и качества программ. М.: Издательство МГТУ им. Н.Э. Баумана, 2015. 38 с.

Структуры данных, алгоритмы сортировки, поиска и др. URL: https://prog-cpp.ru (дата обращения 10.02.2020).

Тестирование программных продуктов. URL: http://lib.kstu.kz:8300/tb/books/Razrabotka_PO(VT)/тестирование.htm (дата обращения 10.02.2020).

Приложение 1.
ПРИМЕР ОФОРМЛЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ 1

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«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)

О т ч е т
по лабораторной работе № 1

Вариант № 1

Дисциплина: Технологии разработки программных систем

Название лабораторной работы: Выбор структур и методов обработки данных

Студент гр. ИУ6-41Б _____ Иванов И.И.
(Подпись, дата) (Фамилия И.О.)

Преподаватель _____ Сергеев С.С.
(Подпись, дата) (Фамилия И.О.)

2020 г.

1 Цель лабораторной работы

Целью данной работы является определение основных критериев оценки структуры данных и методов ее обработки применительно к конкретной задаче.

2 Описание задания

2.1 Задание

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

2.2 Основные требования

Основной вариант задания включает в себя следующие требования:

- структура данных — таблица;
- поиск — двоичный;
- упорядочение — слияние;
- корректировка — удаление маркировкой.

3 Описание основного варианта задания

3.1 Структура данных

Структуру данных по заданию (таблицу) предлагается реализовать с помощью одномерного массива записей, структура которого представлена на рисунке 1.

1	Запись 1
	. . . . .
i	Запись *i*
	. . . . .
k	Запись *k*

Рисунок 1 — Структура массива записей

Каждый элемент массива (запись) содержит поля:

- код детали (целое число);
- код материала (длина: 6 символов);
- единица измерения (длина: 15 символов);
- номер цеха (целое число);
- норма расхода (целое число).

3.1.1 Реализация структуры на языке C++

Предлагается структуру данных реализовать следующим образом:

```
struct shop {
int detail_code; //код детали
char mat_code[6]; //код материала
char measure[15]; //единица измерения
int num_fact; //номер цеха
int norm_consumption; //норма расходов
bool delete_flag; //маркер удаления
};
int k = 5; //задаем k — число элементов в таблице
shop uptown[k]; //создаем таблицу uptown с k элементами
…
detail_code int
mat_code char[6]
measure char[15]
num_fact int
norm_consumption int
delete_flag bool
for (int i = 0; i<k; i++){
cin >> uptown[i].detail_code >> uptown[i].mat_code >>
  uptown[i].measure >> uptown[i].num_fact >>
uptown[i].norm_consumption;
uptown[i].delete_flag = false;
}
```

3.1.2 Расчет памяти, занимаемой массивом

Объем занимаемой памяти массивом $V = kV_э$, где k — количество элементов, а $V_э$ — размер одного элемента. Множитель k определяется пользователем, но не может быть динамически изменен. Размер элемента является суммой размера полей элемента. Рассчитаем эти размеры, учитывая, что 1 символ занимает 1 байт, а булева переменная — 1 байт:

$$V_э = l_{det} + l_{mat} + l_{meas} + l_{fact} + l_{cons} + l_{del} =$$
$$= 2 + 6 \cdot 1 + 15 \cdot 1 + 2 + 2 + 1 = 28 \text{ байт.}$$

Получаем $V = 28k$ байт.

Если массив статический, а k неизвестно, то это приведет к неэффективному использованию оперативной памяти.

3.1.3 Оценка времени доступа к *i*-му элементу

В массиве доступ выполняется по индексу. Соответственно, количество тактов, необходимых для доступа к i-му элементу, складывается из количества тактов, необходимых для умножения и для сложения с постоянной:

$$T_д = t_{++} + t_{\to} = 1 + 1 = 2 \text{ такта.}$$

3.1.4 Оценка времени удаления *i*-го элемента

При удалении элемента происходит сдвиг следующей за ним части массива, т. е. проводится циклическое смещение элементов, подразумевающее копирование очередного элемента в ячейку выше. Далее, после выхода из цикла, необходимо уменьшить значение переменной, хранящей текущее количество элементов. Количество тактов при удалении элемента массива равно:

$$DT_i = t_{for} + t_{++} = t_{\text{уст}} + t_{\text{пр}} + 1 + (k-i)(t\_l_{\text{эл}} + t_{\text{пр}} + 2) + t_{++} =$$
$$= 21 + 2 + 1 + (k-i)(2 \cdot 28 + 2 + 2) + 1 =$$
$$= 25 + 60(k-i) \text{ тактов.}$$

3.2 Анализ применимости метода поиска

По условию задания необходимо использовать дихотомический поиск, который применим к заданной структуре, так как массив обеспечивает возможность применения прямого метода доступа к элементу данных. У рассматриваемого метода поиска имеется ряд недостатков, например, требуется упорядочивать массив и др.

3.2.1 Реализация метода поиска

Ниже приведена реализация метода поиска:

```
shop *find_1(int x, shop s[x], int det_to_find){
shop *det_out = NULL;
int left = 0, right = x-1, middle;
while (left <= right) {
middle = left + (right − left) / 2;
if (s[middle].detail_code < det_to_find)
right = middle − 1;
else if (s[middle].detail_code > det_to_find)
left = middle + 1;
else {
det_out=&s[middle];
left = right;
}
}
return det_out;
}
```

3.2.2 Среднее количество сравнений

Среднее количество сравнений для операции поиска равно:

$$C = \left[(k+1)\log 2(k+1)\right] / k - 1.$$

3.2.3 Оценка времени поиска

Ниже приведена оценка времени поиска.

$$T = t_{=} + t_{=} + C\left(t_{+} + t_{=} + t_{+} + t_{+} + t_{/} + t_{if}\right) =$$

$$= 2 + 2 + C\left(2 + 2 + 2 + 2 + 28 + 12\right) = 4 + 48C \text{ тактов.}$$

$$t_{if} = t_{\text{пр}} + 1 + p_1 t_1 + p_2 t_2 = t_{\text{пр}} + 1 + p_1(t_{=} + t + c) + p_2 t_{if} =$$

$$= t_{+} + t_{\wedge} + 1 + p_1(t_{=} + t + c) + p_2(t_{\text{пр}} + 1 + p_1 t_1 + p_2 t_2) =$$

$$= t_{+} + t_{\wedge} + 1 + p_1(t_{=} + t + c) + p_2(t_{+} + t_{\wedge} + 1 + p_1(t_{=} + t + c) + p_2 * (t_{=} + t_{=} + t_i + t)) =$$

$$= 2 + 2 + 1 + 0{,}5 \cdot \left(2 + 1\right) + 0{,}5 \cdot \left(2 + 2 + 1 + 0{,}5 \cdot \left(2 + 1\right) + 0{,}5 \cdot \left(2 + 2 + 2 + 2\right)\right) =$$

$$= 5 + 1{,}5 + 0{,}5 \cdot \left(5 + 1{,}5 + 4\right) = 6{,}5 + 0{,}5 \cdot 11 = 12.$$

В итоге $T = 4 + 48([(k+1)\log 2(k+1)]\ /\ k - 1)$.

3.3 Анализ метода упорядочивания

Метод слияния предполагает деление исходной структуры пополам, далее каждая часть сортируется и проводится слияние этих частей. Анализ показал, что ускорение сортировки происходит за счет деления на части. На этапе слияния количество сравнений зависит от характера данных и может быть различным.

3.3.1 Реализация метода

Код реализации метода сортировки слиянием приведен ниже:

```
void *sort_1( int x, shop *s) {
if (x > 1) {
shop tmp;
shop alt[x]; //создаем альтернативный массив для буфера и
  заполняем его
for (int i = 0; i < x; i++)
alt[i] = s[i];
//сортировка двух половин массива методом вставки
for (int i = 0; i < x/2+1; i += x/2) //обеспечаиваем раз-
  деление массива на два
for (int j = 1+i; j < x/2+i+(i ? x%2 : 0); j++)
for (int z = i; z < x/2-1+i+(i ? x%2 : 0); z++)
if (alt[z].detail_code > alt[z+1].detail_code) {
tmp = alt[z];
alt[z] = alt[z+1];
alt[z+1] = tmp;
}
//слияние массивов
int a = 0, b = x/2;
```

```
bool e_switch;
for (int i = 0; i < x; i++) {
e_switch = (b < x) && (alt[a].detail_code > alt[b].detail_
  code) || (a == x/2);
if (e_switch) {
s[i] = alt[b];
b++;
} else {
s[i] = alt[a];
a++;
}
}
}
return nullptr;
}
```

3.3.2 Оценка количества сравнений

Учитывая, что при данном методе упорядочивания после разбиения массива на две половины происходит их упорядочивание методом вставок, а затем их слияние путем попарного сравнения элементов подмассива, получаем:

$$\mathbf{C} = 2 \cdot 1/8 \cdot (k-2)k \; + \; k/2 = 1/4 \cdot k(k-2) + k/2.$$

3.3.3 Оценка объема буферных данных

Для проведения слияния создается буферный массив, размер которого равен исходному массиву, и буферный элемент для сортировки вставками.

$$V_{\text{В}} = V_{\text{эл}}k + V_{\text{эл}} = 28(k+1) \text{ байт.}$$

3.4 Метод корректировки

Удаление маркировкой. Удаляемый элемент из массива физически сразу не удаляется, а лишь помечается в специальном поле delete_flag. При этом реализуется возможность замещения помеченного элемента новым. Во время работы возникает необходимость упаковки (сжатия) массива, т. е. физически удалять помеченные элементы.

3.4.1 Реализация метода

Код реализации метода удаления маркировкой следующий:

```
void del_mark(shop *s, int i) {
s[i].delete_flag = true;
}
//удаление маркировкой
void *del_mark1(int x, shop *s){
```

```
for (int i=0; i<x; i++){
if (s[i].delete_flag == true) {
for (int j = i; j < x; j++)
s[j] = s[j + 1];
}
}
return nullptr;
}
```

3.4.2 Расчет времени удаления элемента

Для физического удаления i-го элемента необходимо сначала его пометить, а затем выполнить сжатие массива. Для сжатия необходимо выполнить поиск этого элемента с начала массива. Итого, получим следующее время физического удаления элемента:

время для маркировки: $MT = t_{\wedge} + t_i + t = 2 + 2 + 2 = 6$;

время удаления i-го элемента из массива посчитано ранее:

$$DT_i = t_{for} + t_{++} = t_{\text{уст}} + t_{\text{пр}} + 1 + (k-i)\left(t_{=}l_{\text{эл}} + t_{\text{пр}} + 2\right) + t_{++} =$$

$$= 21 + 2 + 1 + (k-i)(\ 2 \cdot 28 + 2 + 2) + 1 = 25 + 60(k-i);$$

$$T = t_{\wedge} t_i + t_{=} + MT + DT_i = 2 + 2 + 2 + 6 + 25 + 60(k-i) = 37 + 60(k-i).$$

4 Альтернативный вариант

4.1 Выбор альтернативной структуры данных

В качестве альтернативной структуры данных предлагается гнездовая организация на основе односвязного списка (рисунок 2).

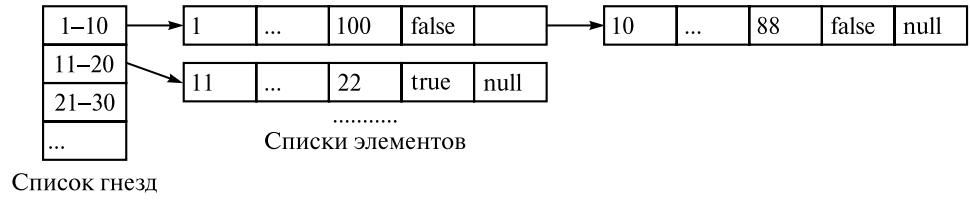


Рисунок 2 — Альтернативная структура данных

Структура разбита на гнезда по коду детали, т. е. имеем гнезда — группы деталей. Другими словами, организуется быстрый поиск по коду детали.

Каждый элемент в гнезде имеет следующую структуру:

- код детали (целое число);
- код материала (длина 6 символов);
- единица измерения (длина 15 символов);
- номер цеха (целое число);

- норма расхода (целое число);
- указатель на следующий элемент.

Ниже представлен код реализации структуры элемента и код инициализации массива:

```
struct shop {
int detail_code; //код детали
char mat_code[6]; //код материала
char measure[15]; //единица измерения
int num_fact; //номер цеха
int norm_consumption; //норма расходов
bool delete_flag; //маркер удаления
shop *next;
};
struct gnezdo {
int max_code;
shop *first_detail;
};
int k = 289;
int x = k/10 + (k%100 ? 1:0);
gnezdo g[x];
cout << x << endl;
shop *prev = NULL;
//создания x гнезд и списков
for (int i = 0; i<x; i++){
g[i].max_code=(i+1)*10;
//cout << «введите все элементы гнезда 1
// (код детали не больше» << g[i].max_code << «)»<<endl;
for (int j = 0; j < 10; ++j) {
if (i * 10 + j + 1 < k) {
shop *cur = new shop();
cin >> cur->detail_code >> cur->num_fact;
cur->next=NULL;
if (j == 0)
g[i].first_detail = cur;
else
prev->next=cur;
prev = cur;
}
}
```

4.1.2 Расчет занимаемой памяти

Для элементов в группах $V = kV_э$, где k — количество элементов; $V_э$ — размер одного элемента. Считая, что размер указателя равен 4 байтам, получаем:

$$V_э = l_{det} + l_{mat} + l_{meas} + l_{fact} + l_{cons} + l_{del} + l_{next} =$$
$$= 2 + 6 \cdot 1 + 15 \cdot 1 + 2 + 2 + 1 + 4 = 32 \text{ байт.}$$

В результате $V = 32k$ байт.

Помимо этого, мы создаем отдельный массив указателей, который хранит указатели на гнезда:

$$V_{мас} = x \cdot 4,$$

где x — количество гнезд, $x = k/10$.

Получается, что общая память равна:

$$V_{гн} = 4 + 32k + 0{,}4k.$$

4.1.3 Оценка времени доступа к *i*-му элементу

$$T_д = t_{for} = 2 + 2 + 1 + (i-1)(t_^ + 2 + 2) = 5 + i(2+4) = 5 + (i-1) \cdot 6.$$

4.1.4 Оценка времени удаления элемента

Для удаления элемента нужно перенаправить указатель предыдущего элемента на следующий:

$$DT = t_^ + t_^ = 3 \text{ такта.}$$

4.2 Метод поиска

Поиск в первой структуре — дихотомический, а внутри гнезда — последовательный.

4.2.1 Реализация метода на языке C++

```
shop *find_1(int x, gnezdo s[x], int det_to_find){
shop *det_out = NULL;
int left = 0, right = x-1, middle;
while (left <= right) {
middle = (right + left) / 2;
if (s[middle].max_code -99 > det_to_find)
right = middle – 1;
else if (s[middle].max_code < det_to_find)
left = middle + 1;
else {
```

```
det_out = s[middle].first_detail;
left = right+1;
}
}
//поиск детали внутри гнезда последовательным методом
shop *cur = det_out;
while (cur) {
if (cur->detail_code == det_to_find) {
det_out = cur;
cur = NULL;
} else
cur = cur->next;
}
return det_out;
}
```

4.2.2 Среднее количество сравнений при поиске

Для выбора нужного гнезда:

$$C_{\text{мас}} = \left[(x+1)\log 2(x+1)\right]/x - 1 = \left[(k/10+1)\log 2(k/10+1)\right]/(k/10) - 1.$$

Для выборки детали из списка внутри гнезда:

$$C_{\text{гн}} = (k/10+1)/2.$$

Общее число сравнений:

$$C = C_{\text{мас}} + C_{\text{гн}} = \left[(k/10+1)\log 2(k/10+1)\right]/(k/10) - 1 + (k/10+1)/2.$$

4.2.3 Оценка времени поиска

Поиск элемента в структуре определяется суммой времени поиска и времени поиска элемента внутри гнезда.

Поиск гнезда:

$$GT = C_{\text{мас}}\left(t_+ + t_+ + t_/ + t_= + t_{if}\right) = C_{\text{мас}}\left(2+2+28+2+t_{\text{пр}}+1+p_1t_1+p_2t_2\right) =$$
$$= C_{\text{мас}}(35 + t_i + t_\wedge + t_+ + t_+ + 0{,}5(t_= + t + c) + 0{,}5t_{if}) =$$
$$= C_{\text{мас}}(43 + 2 + 0{,}5(t_{\text{пр}} + 1 + p_1t_1 + p_2t_2)) = 52C_{\text{мас}}.$$

Поиск элемента в списке:

$$ST = C_{\text{гн}}\left(t_+ + t_{if}\right) = C_{\text{гн}}(2 + t_{\text{пр}} + 1 + p_1t_1 + p_2t_2) =$$
$$= C_{\text{гн}}(2 + t_\wedge + t_+ + 0{,}5(t_= + t_=) + 0{,}5(t_= + t_\wedge)) = C_{\text{гн}}(6+2+2) = 10C_{\text{гн}}.$$

Общий поиск элемента:

$$T = GT + ST = 52C_{\text{мас}} + 10C_{\text{гн}}.$$

4.3 Выбор метода упорядочивания гнезд

В работе было отдано предпочтение методу сортировки, который просто реализуется и базируется на сравнении соседних элементов. Вначале, при сортировке по возрастанию, сравнивается первый элемент со вторым, и если он больше, то элементы меняются местами. Далее второй элемент сравнивается с третьим и т. д. После анализа последнего элемента процесс повторяется сначала. Количество сравнений определяется выражением: $C = N(N-1)$, где N — количество элементов.

В альтернативном варианте была выбрана гнездовая структура данных, поэтому необходимо учесть количество гнезд.

При сортировке одного гнезда количество сравнений равно $C_{\text{г}} = k/10(k/10-1)$.

Количество сравнений при полной сортировке, при условии, что гнезд десять и они равнозначные, равно $C = C_{\text{г}} \cdot 10 = k(k/10-1)$.

4.4 Метод корректировки

Непосредственное удаление. При удалении элемента из списка достаточно перезаписать адресное поле (без учета времени освобождения памяти и времени поиска элемента).

Код реализация метода удаления: `prev->next=cur->next`;

Количество тактов указано выше в данном разделе.

Так как количество гнезд фиксировано, то нет необходимости удалять или добавлять элементы вспомогательной структуры.

Заключение

В результате выполнения лабораторной работы были проведены качественные и количественные оценки структур данных и методов их обработки в соответствии с вариантом задания. В альтернативном варианте предложены решения, которые обеспечат более эффективные поиск, сортировку и удаление данных.

Таблица 1 — Таблица результатов

Варианты	Структура данных	Метод поиска	Метод упорядочения (сортировки)	Метод корректировки
Основной	Массив записей $V = 28k;$ $T_д = 2$	Дихотомический $C = \frac{(k+1)\log 2(k+1)}{k-1};$ $T = 4 +$ $+ 48\frac{(k+1)\log 2(k+1)}{k-1}$	Метод слияния $C = 1/4 \cdot k(k-2) +$ $+ k/2$	Удаление маркировкой $T = 37 +$ $+ 60(k-i)$
Альтернативный	Гнездовая организация односвязных списков $V_{гн} = 4 + 32k +$ $+ 0,4k;$ $T_д = 5 + (i-1) \cdot 6$	Дихотомический и последовательный $C =$ $= \frac{(k/10+1)\log 2(k/10+1)}{(k/10)} -$ $-1 + (k/10+1)/2;$ $T = GT + ST =$ $= 52\frac{(k/10+1)\log 2(k/10+1)}{(k/10)} -$ $-1 + 10(k/10+1)/2$	Метод пузырька $C = C_г \cdot 10 =$ $= k(k/10-1)$	Непосредственное удаление из списка $T = 3$

К недостаткам альтернативного варианта можно отнести то, что гнездовая структура требует больше оперативной памяти.

Приложение 2.
ПРИМЕР ОФОРМЛЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ 2

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«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)

О т ч е т
по лабораторной работе № 2

Вариант № 1

Дисциплина: Технологии разработки программных систем

Название лабораторной работы: Оценка эффективности и качества программ

Студент гр. ИУ6-41Б _____ Иванов И.И.
(Подпись, дата) (Фамилия И.О.)

Преподаватель _____ Сергеев С.С.
(Подпись, дата) (Фамилия И.О.)

2020 г.

1 Цель лабораторной работы

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

2 Описание задания

Программа должна сортировать четные строки массива вещественных чисел.

3 Исходный код программы с точками фиксации времени

```
program V1;
{$APPTYPE CONSOLE}
uses
SysUtils;
const N=5;
var m:array [1..N,1..N] of integer;
i,j,b,d,x,y:integer;
P1,P2: TDateTime; // вспомогательные переменные
H1,M1,S1,Ms1,H2,M2,S2,Ms2:word;
Begin
P1:= Now; // фиксация времени начала
DecodeTime(P1, H1, M1, S1, Ms1);
randomize;
x:=5;
for i:=1 to N do begin
writeln;
for j:=1 to N do begin
m[i,j]:=random(10);
write(m[i,j]:x) end;
end;
writeln; writeln;
for i:=1 to N do
for j:=1 to N do
for d:=1 to N-1 do
if odd(i) then
if m[i,d]>m[i,d+1] then begin
b:=m[i,d]; m[i,d]:=m[i,d+1]; m[i,d+1]:=b;
end;
x:=5;
for i:=1 to N do begin
for j:=1 to N do write(m[i,j]:x);
writeln;
end;
P2:= Now; // фиксация времени окончания
```

```
DecodeTime(P2, H2, M2, S2, Ms2);
Writeln(H1,':',M1,':',S1,':',Ms1); // время начала выпол-
   нения
Writeln(H2,':',M2,':',S2,':',Ms2); //время окончания
readln;
readln;
end.
```

4 Улучшенный вариант программы

```
{Сортировка четных строк массива}
program V1;
{$APPTYPE CONSOLE}
uses
SysUtils;
var m: array of array of real;
i,j,d, n:integer;
b: real;
P1,P2: TDateTime; // вспомогательные переменные
H1,M1,S1,Ms1,H2,M2,S2,Ms2:word;
Begin
read(n);
SetLength(m, n, n);
for i:=0 to n-1 do begin
for j:=0 to n-1 do read(m[i, j]);
end;
writeln;
P1:= Now; // фиксация времени начала
DecodeTime(P1, H1, M1, S1, Ms1);
for i:=1 to N do begin
if i mod 2 = 0 then
for j:=1 to N do
for d:=1 to N-1 do
if m[i,d]>m[i,d+1] then begin
b:=m[i,d];
m[i,d]:=m[i,d+1];
m[i,d+1]:=b;
end;
for j:=1 to N do write(m[i,j]:5);
writeln;
end;
m:=nil;
P2:= Now; // фиксация времени окончания
DecodeTime(P2, H2, M2, S2, Ms2);
```

```
Writeln(H1,':',M1,':',S1,':',Ms1); // время начала выпол-
   нения
Writeln(H2,':',M2,':',S2,':',Ms2); //время окончания
readln;
readln;
end.
```

5 Оценка эффективности

В таблице 1 отражены результаты замеров времени и оценки памяти для исходной программы и улучшенной, а также указаны недостатки и способы улучшения.

Таблица 1 — Оценка эффективности

Критерий оценки	Исходная программа		Улучшенная программа	
	Недостатки	Количественная оценка	Улучшения	Количественная оценка
Время выполнения	Лишний цикл. Лишние проверки (условие проверяется для каждого элемента массива, а можно 1 раз для всей строки)	100 мс	Удален лишний цикл	10 мс
Оперативная память	Лишние переменные *x* и *y*. Неверный тип элементов массива, массив статический	5 переменных типа integer: $V_1 = 5 \cdot 2 = 10$; 25 элементов в массиве: $V_2 = 5 \cdot 5 \cdot 2 = 50$. Итого 60 байт	Удалены переменные *x* и *y*. Изменен тип элементов в массиве и переменной *b*. Создан динамический массив	160 байт

6 Оценка качества

В таблице 2 отражены результаты оценки качества исходной программы.

Таблица 2 — Оценка качества

Результаты оценки	Критерии оценки			
	Правильность	Универсальность	Проверяемость	Точность результатов
Недостатки	Неверный тип элементов у массива (по условию вещественные числа). Сортируются нечетные строки, а нужно четные	Используется статический массив, но лучше динамический. Можно обеспечить работу с вещественными числами. Числа только генерируются, но можно обеспечить ввод данных пользователем	Выводится исходный массив и результирующий. Можно добавить номера сортируемых строк	—
Оценка, баллы	3	2	4	—

Заключение

В результате проведенных экспериментов были выполнены замеры времени работы программы, оценки памяти, а также предложены способы повышения эффективности и качества программы.

Приложение 3.
ПРИМЕР ОФОРМЛЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ 3

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«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)

О т ч е т
по лабораторной работе № 3

Вариант № 1

Дисциплина: Технологии разработки программных систем

Название лабораторной работы: Тестирование программного обеспечения

Студент гр. ИУ6-41Б _____ Иванов И.И.
(Подпись, дата) (Фамилия И.О.)

Преподаватель _____ Сергеев С.С.
(Подпись, дата) (Фамилия И.О.)

2020 г.

1 Цель лабораторной работы

Цель работы — приобрести навыки тестирования схем алгоритмов, исходных кодов программ и исполняемых модулей.

2 Структурный контроль

2.1 Описание задания

Программа должна читать типизированный файл и выводить данные на экран.

2.2 Исходный код программы для тестирования

```
Program v1;
{$APPTYPE CONSOLE}
uses SysUtils;
Type
 Rec=record
 Fio:string[25];
 Year:word;
 Adr:string[30];
 Tel:string[9];
 end;
Var Ftp:file of rec;
 Mark:longint;
 V:Rec;
 s,Fn:string;
 c:char;
begin
 { TODO -oUser -cConsole Main : Insert code here }
 readln(Fn);
 s:=filesearch(Fn,'');
 If (s<>'') and (filesize(Ftp)>0) then begin
 assign(Ftp,Fn); reset(Ftp); Mark:=0;
 While (not Eof(Ftp)) do
 begin
 seek(Ftp,Mark);
 readln(Ftp,V);
 writeln(V.Fio); writeln(V.Year);
 writeln(V.Adr); writeln(V.Tel);
 end
 end;
 end.
```

2.3 Результаты тестирования

Результаты тестирования структурным контролем представлены в таблице 1.

Таблица 1 — Результаты тестирования структурным контролем

Номер вопроса	Строки, подлежащие проверке	Результат проверки	Вывод
1.1	Все строки программы	*Mark* = 0. F_n — вводится. *S* — пустая строка/название файла. *V* — поочередные записи из файла. F_{tp} — ? *C* — ?	Не все переменные инициализированы. Переменная нигде не используется, т. е. не вызовет ошибки. Переменная F_{tp} приведет к ошибке при вызове функции `filsize(Ftp)` (23-я строка)
1.2	20, 27	F_n — не задана длина. *V* — для считывания из типизированного файла	Максимальные размеры строк не превышены. Массивы не использованы
1.4	6–17	`Ftp, Fn, s, V c, Fio, Year,Adr, Tel`	Схожие имена: `Ftp, Fn`
1.5	25	`Readln (ftp, v)`	В типизированном файле типа record недопустимо использование функции Readln
3.1	24–30	`Seek(Ftp, 0)`	Указатель всегда будет указывать на первый элемент в файле, поэтому не будет осуществлен выход из цикла

Выводы: структурный контроль позволяет обнаружить общие ошибки кодирования.

Достоинства:

- не требует выполнения программы;
- позволяет обнаружить общие ошибки программирования.

Недостатки:

- ошибки, на обнаружение которых направлен структурный контроль, автоматически выявляются средствами разработки;
- по списку вопросов трудно обнаружить ошибки в логике программы;
- большие программы трудно инспектировать.

3 Методы белого ящика

3.1 Схема алгоритма для тестирования

В соответствии с вариантом задания № 10 была протестирована схема алгоритма, представленная на рисунке 1.

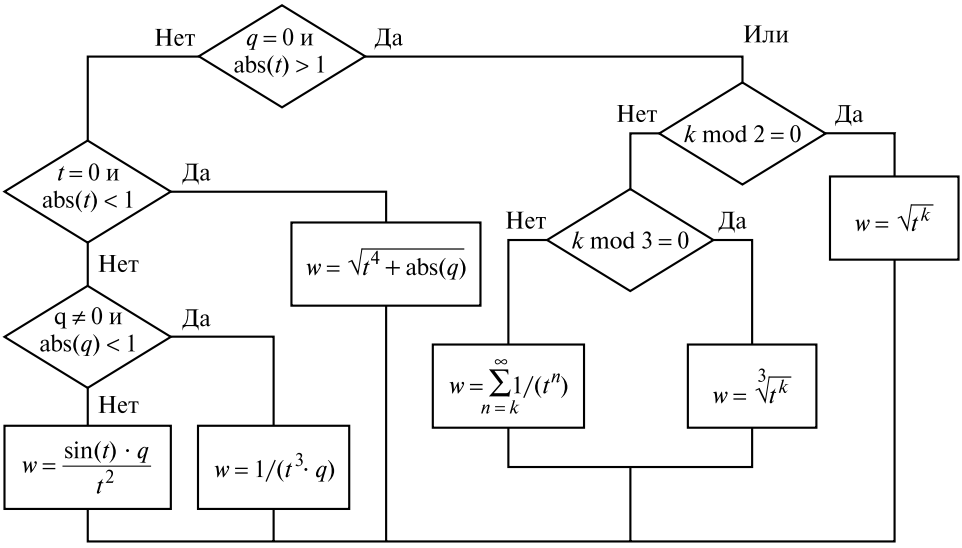


Рисунок 1 — Схема алгоритма

3.2 Метод покрытия операторов

Результаты тестирования по методу покрытия операторов представлены в таблице 2.

Таблица 2 — Результаты тестирования по методу покрытия операторов

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат
1	Проверить оператор `w=sint(t)*q/t^2`	$q = 0$, $t = 0{,}2$, $k = 1$	$w = 0$
2	Проверить оператор `w=1/(t^3*q)`	$q = 0{,}5$, $t = 1$, $k = 2$	$w = 2$
3	Проверить оператор `w=sqrt(t^4+abs(q)`	$q = -4$, $t = 0$, $k = 5$	$w = 2$
4	Проверить оператор `w=sqrt(t^k)`	$q = 0$, $k = 4$, $t = -4$	$w = 16$
5	Проверить оператор `w=(t^k)^(1/3))`	$q = 0$, $k = 3$, $t = -3$	$w = -3$
6	Проверить оператор `w=S(1/t^n)`	$q = 0$, $k = 5$, $t = 2$	Error, ошибка времени исполнения

3.3 Метод покрытия решений

Результаты тестирования по методу покрытия решений представлены в таблице 3.

Таблица 3 — Результаты тестирования по методу покрытия решений

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат
1	Нет, нет, нет	$q = 0$, $t = 0{,}2$, $k = r$	$w = 0$
2	Нет, нет, да	$q = 0{,}5$, $t = 1$, $k = r$	$w = 2$
3	Нет, да	$q = -4$, $t = 0$, $k = r$	$w = 2$
4	Да, да	$q = 0$, $k = 4$, $t = -4$	$w = 16$
5	Да, нет, да	$q = 0$, $k = 3$, $t = -3$	$w = -3$
6	Да, нет, нет	$q = 0$, $k = 5$, $t = 2$	Error, ошибка времени исполнения

3.4 Метод комбинаторного покрытия условий

По схеме алгоритма можно выделить 16 комбинаций условий:

1) q = 0,abs(t) > 1;
2) q = 0,abs(t) ≤ 1;
3) q ≠ 0,abs(t) > 1;
4) q ≠ 0,abs(t) ≤ 1;
5) t = 0,abs(t) < 1;
6) t = 0,abs(t) ≥ 1;
7) t ≠ 0,abs(t) < 1;
8) t ≠ 0,abs(t) ≥ 1;
9) q ≠ 0,abs(q) < 1;
10) q ≠ 0,abs(q) ≥ 1;
11) q ≠ 0,abs(q) < 1;
12) q ≠ 0,abs(q) ≥ 1;
13) Kmod2 = 0;
14) kmod3 = 0;
15) kmod2 ≠ 0;
16) kmod3 ≠ 0.

Вышеперечисленные комбинации можно покрыть семью тестами. Результаты тестирования представлены в таблице 4.

Таблица 4 — Таблица результатов тестирования для комбинаторного покрытия условий

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат
1	$q = 0$, $t = 5$, $k = 2$	1, 13	`w=sqrt(t^k)` `w=5`
2	$q = 0$, $t = 4$, $k = 3$	1, 14, 15	`w=(t^k)^(1/3))` `w=4`
3	$q = 0$, $t = 6$, $k = 5$	1, 15, 16	Error, ошибка времени исполнения
4	$q = 0$, $t = 0$, $k = 1$	2, 5	`w=sqrt(t^4+abs(q))` `w=0`
5	$q = 0{,}5$, $t = 0{,}1$, $k = 2$	4, 7, 9	`w=1/(t^3*q)` `w=2000`
6	$q = 3$, $t = 3$, $k = 4$	3, 8, 10	`w=sint(t)*q/t^2` `w=3.339`
7	$q = 0$, $t = 0{,}1$, $k = 2$	2, 7, 11	`w=sint(t)*q/t^2` `w=0`

Из таблицы 4 видно, что метод комбинаторного покрытия условий позволил обнаружить ошибку. Также при тестировании была выявлена невозможность задать комбинации, показанные в условиях 6 и 12.

4 Метод черного ящика

4.1 Требования к тестируемой программе

Написать программу, которая реализует секундомер. Программа должна выдавать пользователю сотые доли секунды, секунды и минуты.

4.2 Метод эквивалентного разбиения

Данный метод неприменим к программе по заданию, так как нет возможности задавать исходные данные, как обычные, так и управляющие.

4.3 Метод граничных условий

Метод граничных условий позволил найти ряд ошибок.

Таблица 5 — Таблица результатов тестирования для метода граничных условий

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат	Реакция программы	Вывод
1	Проверка границ секунд	Начало отсчета 00:00:00	Границы для значений секунд 0–59	Границы для значений секунд 0–49	Программа работает неверно
2	Проверка границ долей секунд	Начало отсчета 00:00:00	Границы для значений долей секунд 0–99	Границы для значений долей секунд 0–100	Программа работает неверно
3	Проверка границ минут	Начало отсчета 00:00:00	Границы для значений минут 0–59. Сообщение об ошибке	Границы для значений минут 0–58. Сообщение об ошибке	Программа работает неверно

4.4 Анализ причинно-следственных связей

В программе по заданию пользователь имеет возможность задавать управляющие данные путем нажатия кнопок. В таблице 6 представлены результаты тестирования, полученные на основе анализа причинно-следственных связей.

Таблица 6 — Таблица результатов тестирования на основе анализа причинно-следственных связей.

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат	Реакция программы	Вывод
1	Запуск	Нажатие кнопки Пуск	Запуск секундомера	Изменение названия кнопки (запуск на стоп)	Программа работает верно
2	Стоп	Нажатие кнопки Стоп	Остановка отсчета при сохранении значений секундомера	Остановка отсчета при сохранении значений секундомера	Программа работает верно
3	Сброс	Нажатие кнопки Сброс	Обнуление отсчета	Обнуление отсчета	Программа работает верно
4	Увеличение минут	Достижение секунд 59	Минуты увеличиваются на 1	При достижении 58-й секунды минуты увеличиваются на 1	Программа работает неверно
5	Увеличение секунд	При достижении долей секунд 99	Секунды увеличиваются на 1	При достижении 99-й доли секунд секунды увеличиваются на 1	Программа работает верно

Заключение

В результате исследования методов тестирования были получены следующие результаты:

- выявлены ошибки различных видов;
- оценена специфика каждого метода тестирования;
- оценена трудоемкость тестирования для каждого метода.

Оглавление

Предисловие 3

Введение 5

Лабораторная работа 1. Выбор структур и методов обработки данных 6
1.1. Краткие теоретические сведения 6
1.2. Порядок выполнения лабораторной работы 1 13
1.3. Требования к отчету 14
1.4. Требования к защите 15
1.5. Варианты заданий 16
Вопросы для самоконтроля 18

Лабораторная работа 2. Оценка эффективности и качества программ 19
2.1. Краткие теоретические сведения 19
2.2. Порядок выполнения лабораторной работы 2 21
2.3. Требования к отчету 22
2.4. Требования к защите 22
2.5. Варианты заданий 23
Вопросы для самоконтроля 26

Лабораторная работа 3. Тестирование программного обеспечения 27
3.1. Краткие теоретические сведения 27
3.2. Порядок выполнения лабораторной работы 3 32
3.3. Требования к отчету 34
3.4. Требования к защите 34
3.5. Варианты заданий 35
Вопросы для самоконтроля 39

Литература 40

Приложение 1. Пример оформления лабораторной работы 1 41

Приложение 2. Пример оформления лабораторной работы 2 53

Приложение 3. Пример оформления лабораторной работы 3 58

Учебное издание

Иванова Галина Сергеевна
Ничушкина Татьяна Николаевна
Пугачев Евгений Константинович

ВЫБОР АЛГОРИТМОВ ОБРАБОТКИ ДАННЫХ, ТЕСТИРОВАНИЕ И ПОВЫШЕНИЕ КАЧЕСТВА ПРОГРАММ

Редактор *Е.Д. Нефедова*
Художник
Корректор *Ю.Н. Морозова*
Компьютерная верстка *И.Д. Звягинцевой*

Оригинал-макет подготовлен в Издательстве
МГТУ им. Н.Э. Баумана.

В оформлении использованы шрифты
Студии Артемия Лебедева.

Подписано в печать 20.07.2020. Формат 70×100/16.
Усл. печ. л. 5,525. Тираж 102 экз. Изд. № 729-2019. Заказ №

Издательство МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
press@bmstu.ru
www.baumanpress.ru

Отпечатано в типографии МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
baumanprint@gmail.com