**Дисциплина «Надежность программного обеспечения»**

**Практическое занятие № 1**

**Тема:** Оценка характеристик программ на основе лексического анализа с использованием метрики Холстеда.

**Время выполнения практического занятия (аудиторные часы)** –   
4 часа.

**Цель работы:** изучить методику оценки характеристик программ с использованием метрики Холстеда.

**Оборудование и программное обеспечение:** работа выполняется на ПЭВМ типа IBM PC с использованием стандартных функций ОС.

# **1. Теоретические сведения**

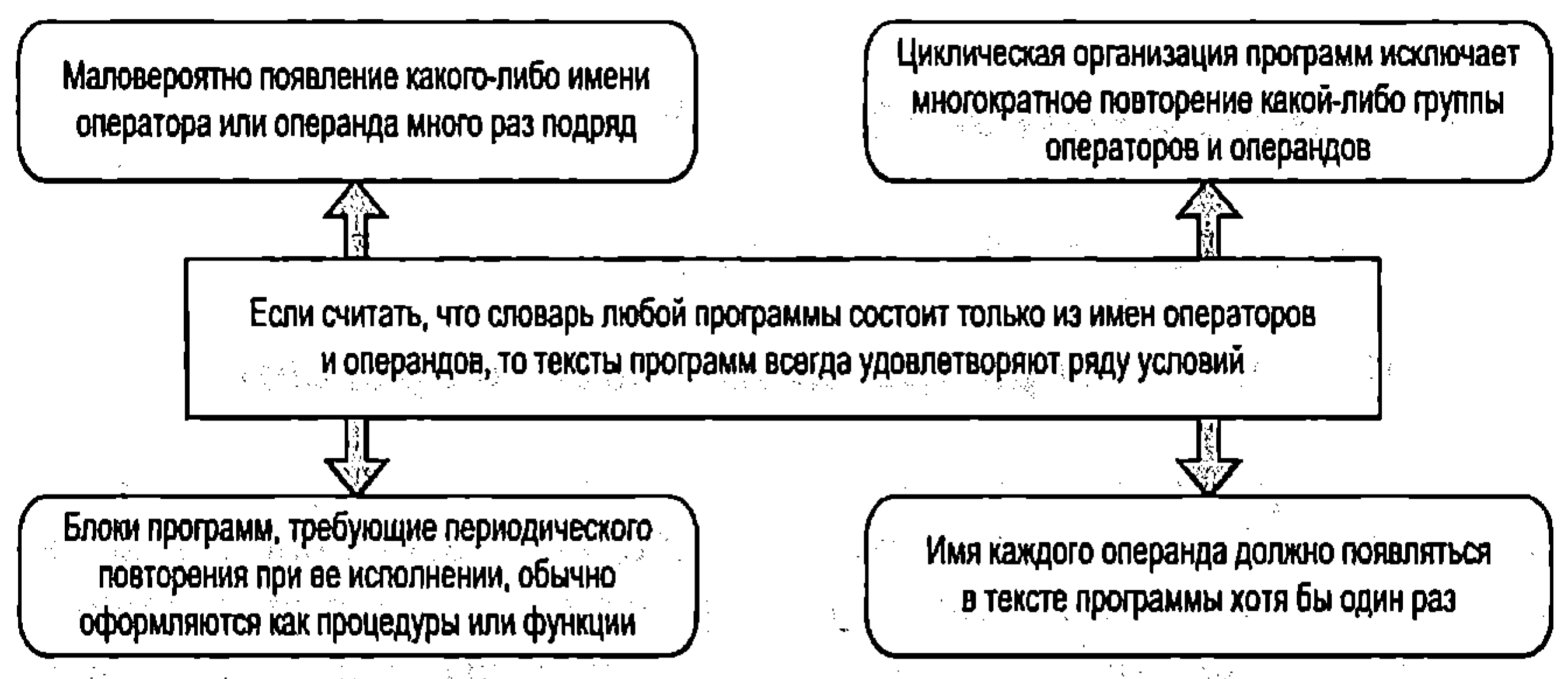
## 1.1 Особенности формирования словаря программы

Любая программа определяет последовательность действий над операндами с помощью операторов. Исходный текст программы, записываемой на том или ином языке программирования представляет собой набор текстовых строк, которые записываются по специальным правилам, и в том числе имеет свои элементы. **Операнд** - это некоторый объект или величина, обрабатываемая в программе, а **оператор** представляет собой обозначение конкретного действия, выполняемого по отношению к операнду.

Если считать, что словарь любой программы состоит только из имен операторов и операндов, то тексты программ всегда удовлетворяют *следующим условиям* [2,4]:

* + - маловероятно появление какого-либо имени оператора или операнда много раз подряд - языки программирования, как правило, позволяют создавать такие конструкции, в которых подобные фрагменты программы имеют минимальную длину;
    - циклическая организация программ исключает многократное повторение какой-либо группы операторов и операндов - более компактные варианты текстов получаются при разумном использовании развитых возможностей языков программирования, причем многообразие языков предоставляет богатую палитру инструментов;
    - блоки программ, требующие периодического повторения при ее исполнении, обычно оформляются как процедуры или функции, поэтому в текстах программ достаточно применения только их имен;
    - имя каждого операнда должно появляться в тексте программы хотя бы один раз - многие среды программирования обращают внимание программистов на неиспользуемые имена, которые следует удалять из текста программ, чтобы сократить объем памяти, используемой при объявлении переменных.

Схематично эти условия, относящиеся к тексту программ, отображены на рис. 1.1.



*Рис. 1.1. Условия образования словаря программы*

В языках программирования С, C++ и С# добавление к набору символов, являющихся выражением, знака «;» превращает его в оператор. Так, набор символов «***i++***» является выражением, а конструкция «***i++;***» уже является оператором (или statement по терминологии Кернигана и Ритчи [1]).

При разработке программ существует ряд особенностей, связанных со скобками: круглыми, фигурными и квадратными.

Одним из действий, которые могут быть записаны в исходном тексте программ, является вызов функций, записываемый в формате

*Имя\_функции().*

В примере применения метрики Холстеда (см. пункт 1.6), в словаре операторов и операций различают вызов функции по появлению пары круглых скобок и следующим перед ними именем. Следует обратить внимание, что при этом нужно отличать пару круглых скобок, используемых с другими языковыми конструкциями (например, с операторами ***if*** или **for**).

При формировании словаря операторов и операций учитываются еще и фигурные скобки. Они, казалось бы, не вызывают никаких действий, но связаны с ограничением области видимости переменных.

Операции, которые связаны с квадратными скобками, это обращение к элементам массива. Действия, связанные с их интерпретацией, обеспечивают доступ к элементам массивов.

Операторы и операции связаны с объектами, над которыми они выполняются. Таким объектами являются, прежде всего, константы, простые переменные, массивы и структуры, однако подобными объектами (операндами) являются и функции, а также классы и методы.

*Примечание.*

*При формировании словарей учитываются элементы, используемые в разделах описаний, поскольку эти элементы также можно рассматривать как операнды или операторы.*

## 1.2 Измеряемые свойства программ

При разработке программы формируется ее текст на каком-либо языке программирования, реализующий алгоритм получения искомого результата на основании обработки заданной совокупности данных. В тексте программы можно идентифицировать все операнды, определенные как переменные или константы, используемые в данной реализации. Аналогичным образом идентифицируются все операторы, определенные как символы или комбинации символов, влияющие на способ обработки, изменение значения или порядок следования и преобразования операндов. Исходя из идентификации операторов и операндов, можно определить ряд измеримых категорий, обязательно присутствующих в версиях любого алгоритма. Они определяются метриками, с помощью которых могут быть получены основные характеристики качества программ.

В состав измеримых свойств любого представления алгоритма (или программы) могут быть включены следующие **метрические характеристики**:

* – число простых (уникальных) *операторов,* появляющихся в данной реализации;
  + - – число простых (уникальных) операндов, появляющихся в данной реализации;
    - – общее число всех операторов, появляющихся в данной реализации;
    - – общее число всех операндов, появляющихся в данной реализации;
    - – число появлений в программе j-го оператора, где j = 1, 2, 3,…,
    - – число появлений в программе j-го операнда, где j = 1, 2, 3,…, .

Учитывая эти основные метрические характеристики для программы, в конкретной реализации текста программы можно определить:

* + - словарь ;
    - длину реализации программы ;
    - длину программы .

Следует отметить, что помимо своего прямого назначения метрики длины программы и длины реализации можно использовать для выявления несовершенств программирования, которые являются следствием применения не самых удачных приемов программирования. Если расчетные значения длины программы и длины реализации отличаются более чем на 10 %, то это свидетельствует о возможном наличии в программе следующих шести классов несовершенств:

**1.** Наличие последовательности дополняющих друг друга операторов к одному и тому же операнду, например А + С - А. *В подобном случае будет выполнено два совершенно ненужных действия, дополняющих переменную С одной и той же величиной, взятой с противоположными знаками.*

**2.** Наличие неоднозначных операндов, например А = D и А = С. *При выполнении таких действий программа будет поставлена в затруднительное положение, поскольку присвоение осуществляется путем приравнивания значения операнда, указанного в левой части, значению, приведенному в правой части выражения. В лучшем случае произойдет ненужное присвоение нового значения уже имеющемуся.*

3. Наличие синонимичных операндов, например А = В и С = В. *Поскольку одно и то же значение должно быть присвоено разным переменным, то для данного примера переменная* **В** *вообще может не использоваться. Более лаконичным вариантом является простое приравнивание значений переменных* **А** *и* ***С****.*

4. Наличие общих подвыражений, например:

***(А + В) • С + D • (А* +В).**

*Здесь применено совсем не обязательное повторение суммирования переменных* **А** *и* **В**, *что приводит к дополнительному времени выполнения программы.*

5. Ненужное присваивание, например **С=А+В**, если переменная С используется в программе только один раз. *При однократном выполнении каких-либо операций над переменной нецелесообразно вводить дополнительный операнд, это ведет к увеличению объема памяти, резервируемой под переменные программы, и увеличивает размер словаря.*

**6.** Наличие выражений, которые не представлены в свернутом виде как произведение множителей, например:

.

Данное преобразование можно представить как

,

т. е. свернуть выражение до квадрата суммы переменных ***X*** и **Y**.

*Такое представление окажется более лаконичным и сократит время, необходимое для выполнения программы.*

В соответствии с приведенными определениями применяются следующие соотношения:

Таким образом, длина реализации и объем программы определяются исключительно на основе анализа текста программы путем подсчета количества операндов и операторов, а также числа их вхождений в текст программы, т. е. на основе лексического анализа текста программы. Длина программы представляет собой математическое ожидание количества слов в тексте программы при фиксированном словаре.

Другой важной характеристикой программы является **ее объем V**. В отличие от длины программы N объем измеряется не количеством слов, а числом двоичных разрядов. Если в словаре имеется слов, то для задания номера любого из них требуется минимум бит.

Объем программы определяется следующим образом:

Тогда с точностью до обозначений полученные соотношения окажутся совершенно идентичными, хотя смысл их будет различным:

В первом случае зафиксирована взаимная связь между длиной программы **N** и размером словаря , во втором – между величиной словаря и объемом программы **V**. Следовательно, по известному размеру словаря можно найти значения **N** и **V**. Идентичность этих выражений говорит о том, что соотношение между величиной словаря и длиной текста единственно и взаимно однозначно.

Выше было отмечено, что словарь программы состоит только из операторов и операндов. Учитывая принятые обозначения, соотношение Холстеда примет следующий вид:

Как правило, при проведении статистических исследований текстов программ к словарю операторов относят следующие элементы:

* + - имена арифметических и логических операций;
    - присваивания;
    - условные и безусловные переходы;
    - разделители;
    - скобки (парные);
    - имена процедур и функций;
    - выражения типа BEGIN...END, IF...THEN...ELSE, DO...WHILE.

Выражения типа BEGIN...END, IF...THEN...ELSE, DO...WHILE и им подобные, осуществляющие блочную группировку операторов, при этом рассматриваются как единые операторы (то же относится и к парам скобок).

Величины количества операторов и операндов и независимы и могут принимать произвольные значения. Однако этого нельзя сказать относительно и , т. е. числа появления всех операторов и всех операндов в тексте программы: между ними можно установить приблизительное соответствие, причем оно будет взаимно однозначным. В каждом конкретном случае каждый операнд не может позиционироваться в программе обособленно, он входит в текст, по крайней мере, хотя бы с одним оператором: например, с разделителем (точка с запятой), отделяющим его от других операторов, или другим набором символов, определяющим способ действия над этим операндом. В то же время применение нескольких операторов к одному операнду маловероятно. Поэтому можно утверждать, что , хотя величины словарей и могут сильно отличаться друг от друга. Это позволяет прийти к весьма важному практическому выводу относительно объема программы:

Основной исходный параметр, на котором базируются все расчеты метрических характеристик будущего ПС, - количество имен входных и выходных переменных , представленных в предельно краткой записи (с точки зрения алгоритмической сложности - сжатой). Например, для задания одномерного массива (т. е. строки), каково бы ни было число его элементов, требуется всего два имени:

* + - указатель адреса начала массива;
    - количество элементов в нем.

Точно так же для задания двумерного массива достаточно иметь три параметра:

* + - указатель адреса первого элемента;
    - число столбцов;
    - число строк.

Если параметр , определенный таким образом, рассматривать как размер генеральной совокупности имен входных и выходных переменных, то величина словаря программы в соответствии с соотношением Холстеда не будет превосходить . Таким образом, можно считать, что

Предположим, что существует некоторый язык программирования, называемый потенциальным, в котором все программы (по крайней мере - для некоторой предметной области) уже написаны и представлены в виде заранее подготовленных процедур или функций. Тогда для реализации любого алгоритма на таком языке потребуется всего два оператора (функция и присваивание) и имен входных и выходных переменных. Это объясняется тем, что в таком потенциальном языке программист должен будет только выбрать нужную процедуру или функцию и применить ее к нужной переменной. Поскольку в такой записи никакие слова не повторяются, то длина программы совпадает с ее объемом и равна:

. (1.10)

Эта величина называется **потенциальным объемом** (минимально возможным), соответствующим максимально компактному тексту программы, реализующей данный алгоритм. Это объясняется тем, что в потенциальном языке минимизировано число операторов, а все операнды сведены к перечню процедур или функций и списку входных и выходных переменных.

В таком случае можно определить **уровень реализации** программы, который рассчитывается с помощью отношения:

Уровень реализации представляет собой метрический показатель, который *характеризует степень компактности программы, экономичность использования средств алгоритмического языка.* Чем ближе значение **L** к единице, тем более совершенна программа.

При переводе алгоритма с одного языка на другой его потенциальный объем не изменяется, но действительный объем **V** может увеличиваться или уменьшаться в зависимости от развитости языков программирования.

Для потенциального языка справедливо равенство **V =** , для любого менее развитого языка следует учитывать соотношение **V >** . Это обусловлено тем, что для потенциального языка , в то время как для всех других языков применяется уравнение длины и учет соотношения .

## 1.3 Оптимизация количества и длины модулей в программе

Соотношение Холстеда позволяет контролировать процесс разработки программных средств, благодаря тому, что если заранее определить их длину, то можно выходить на эту заданную длину модулей при проектировании программ. Действительно, длина модулей (в пределах точности этого соотношения) определяется только их словарями. В свою очередь, величина словарей зависит только от числа групп, на которые разбивается входных и выходных переменных программируемой задачи. Таким образом, величина словаря модуля - контролируемый параметр, а если это так, то, следовательно, и его длина может контролироваться.

Пусть **k** - число модулей. Тогда словарь операндов одного модуля будет определяться следующим выражением:

а для всей программы (с учетом того, что в программе содержится **k** модулей) данное выражение будет иметь вид:

Понятно, что каждой группе присваивается имя, которое замещает эту группу в тексте программы. С учетом этого обстоятельства окончательно имеем:

Наилучшее количество модулей, которое будет обеспечивать минимальную длину программы, можно найти следующим образом:

При этом число входных имен каждого модуля будет равно:

Исследуя надежность программных средств, Морис Холстед в своих работах показал, что наименьшее количество ошибок обнаруживается в модулях, число входных переменных которых не превосходит восьми, т. е. при .

Программные средства реальных информационных систем имеют сложную иерархическую структуру. Как правило, самый нижний уровень, на котором располагаются «исполнительные» модули, является наиболее многочисленным, в то время как в верхней части управляющей «пирамиды» размещается один модуль, являющийся головным. Изучая структуры больших программных систем, В.В. Липаев установил, что наиболее жизнеспособными являются такие программные средства, число уровней которых не превосходит   
7-8.

Проблема структуризации проектируемых программных средств, безусловно, имеет сугубо содержательный характер и принципиально не может быть формализована. Однако расчетные метрические характеристики (длина модулей, их число, количество иерархических уровней) задают оптимальные параметры структуры программных средств, наиболее рациональные в аспекте обеспечения качества реализации проекта.

## 1.4 Оценка уровня языков программирования

Если **N** - длина программы, а - словарь программы, то общее количество выборок необходимых элементов словаря (т. е. фактически - работа программирования) в соответствии с законом Хика будет равна объему программы .

В то же время необходимо еще учесть уровень реализации программы **L**: количество выборок при этом возрастет в ***1/L*** раз. Обозначив работу программирования символом **Е** и учитывая формулу для вычисления уровня реализации программы (1.11), окончательно получим:

В начале 1980-х гг. Холстед ввел формальное определение уровня языка программирования [3], определив этот уровень языка следующим образом:

, (1.18)

где **L** - уровень реализации программы, - ее потенциальный объем.

Для любого алгоритма, который программируется с использованием разных языков, с увеличением объема уровень реализации уменьшается в той же пропорции. В результате произведение уровня **L** на объем ***V*** равняется потенциальному объему данного алгоритма. С другой стороны, если язык реализации остается одним и тем же, а разрешено менять сам алгоритм, имеется другое, но похожее соотношение. В этом случае с увеличением потенциального объема уровень программы **L** уменьшится в том же отношении. Следовательно, произведение **L** на остается неизменным для любого языка.

Определим работу программирования при использовании потенциального языка. В записи на потенциальном языке программа имеет минимально возможную длину, и так как слова (операнды и операторы) в ней не повторяются, то она совпадает с объемом:

Работа программирования в потенциальном языке сводится к выбору из конечного, но огромного по масштабам числа имен функций и процедур:

где **V** - объем программы.

Существует закон Хика, в соответствии с которым время реакции при выборе из некоего числа альтернативных сигналов зависит от их количества. Тогда по закону Хика работа выбора из библиотеки функций составит   
, и полная работа программирования при использовании потенциального языка будет определяться следующим образом:

В таком случае мы получим следующее соотношение:

Полученное значение как раз можно считать количественной мерой уровня любого алгоритмического языка.

Из выражения (1.22) видно, что для постоянства увеличение объема программы должно квадратично зависеть от увеличения объема информации по внешним связям. Поэтому алгоритмически сложные программы вычисления малого числа переменных будут давать значительно более низкое значение , чем программы вычисления большого числа переменных по элементарным выражениям.

В связи с этим метрику уровня языка программирования для сравнения языков следует применять только для конкретной предметной области и близких типов задач. Ниже приведены данные об уровнях некоторых известных языков программирования (табл. 1.1).

*Таблица 1.1. Уровни языков программирования*

|  |  |  |
| --- | --- | --- |
| **Язык** |  | **Отклонения** |
| ***Естественный (английский) язык*** | 2,16 | 0,74 |
| ***PL/1*** | 1,53 | 0,92 |
| ***Алгол*** | 1,21 | 0,74 |
| ***Паскаль*** | 1,25 | 0,76 |
| ***Бейсик*** | 1,22 | 0,72 |
| ***Фортран*** | 1,14 | 0,81 |
| ***Ассемблер*** | 0,88 | 0,42 |

Поскольку известны соотношения для работы программирования **Е** и уровня языка

то, исключая из выражений величину **V**, получим:

Тогда квалификационное время программирования будет определяться следующим образом:

где **S** - число Страуда (**5 < S < 20**, среднее его значение принято считать равным **18**).

## 1.5 Метрика числа ошибок в программе

Значительная часть усилий и времени, которые затрачиваются на создание большинства программных продуктов, приходится на их отладку, т. е. выявление и устранение ошибок типа «лишних и недостающих элементов», внесенных в начальный период написания программы. Следовательно, любое обоснованное представление о количестве первоначальных ошибок, ожидаемых в данной программе, даст важную оценку для практики. Рассматриваемая метрика позволяет предсказать число первоначальных ошибок (до отладки и тестирования), однако не может служить свидетельством правильности (корректности) программы, даже если ее значение равно нулю.

Время, требуемое на разработку программы, характеризуется числом элементарных мысленных различений **Е**. Следовательно, число моментов, в которые можно сделать ошибочное различение, также определяется значением **Е** или связанным с ним значением объема программы **V.**

Пусть каждый объект так же, как и результат, соответствует единице уникальных операндов в потенциальном языке, т. е. .

С помощью равенств

получаем предельное значение потенциального объема:

Далее из уравнения уровня языка программирования имеем:

а из таблицы уровней языка известно, что для естественного (английского) языка = 2,16.

Тогда для описания программы на уровне английского языка приходим к выводу:

Определим теперь как среднее число элементарных различений между возможными ошибками в программировании, а **В** - как число переданных ошибок в программе. Можно ожидать, что

но при этом не будет учтено наличие какой-либо избыточности в создаваемой программе, т. е. если в программе применены какие-либо фрагменты неоптимальной структуры, то этот аспект в данном расчете во внимание не принимается.

Однако уровень реализации программы **L**, собственно, и является мерой такой избыточности. Заметим, что только в потенциальном языке, на котором любая программа может быть выражена в виде вызова процедуры, не повторяются ни операторы, ни операнды. Для потенциального языка уровень реализации **L= 1**, для всех остальных языков **L** уменьшается с увеличением избыточности, неминуемо присущей любому языку в связи с введением дополнительных условий записи функций и процедур.

Следовательно, вместо уравнения (1.30) реальнее ожидать, что на количество ошибок в программе будет влиять и сам язык программирования, который мы можем учесть при помощи уровня реализации программы:

Учитывая, что произведение можно заменить на значение объема программы **V**, получим:

Если теперь приравнять из уравнения (1.30) значению , найденному по формуле (1.29), получим соотношение

С другой стороны, подставив в это выражение значение для **V** из формулы для определения уровня языка, получим:

Из этого выражения следует, что поскольку для определения потенциального объема необходимо только знание числа независимых входных и выходных параметров программы, задаваемого в техническом задании на разработку программы, то после выбора языка программирования потенциальное количество ошибок можно оценить до начала написания программы, т. е. до начала проектирования.

Опытно установлено, что количество ошибок в текстах программ пропорционально работе программирования, которая, как показано выше, может быть вычислена. Кроме того, формирование текста программы человеком происходит не в виде цельного готового продукта, а некоторыми фрагментами ограниченного объема, причем размеры этих фрагментов зависят от очень многих факторов, включая настроение программиста. Предположив, что с каждым из таких фрагментов связана, по крайней мере, одна ошибка (это доказано на основе многочисленных статистических исследований), получим:

где **V** - расчетный объем программного средства.

## 1.6 Практическая реализация оценки характеристик программы на основе лексического анализа с использованием метрики Холстеда

**Задача:** Разработать программу для вычисления значений функции F:

Значения аргументов функции ввести с клавиатуры. На экран монитора вывести значение функции.

**Требуется:** Определить значения метрик Холстеда, на основе которых дать оценку качества разработанного исходного текста программы.

**Реализация программы**

Текст программы для реализации возможного решения поставленной задачи, разработанной с использованием языка программирования С#, приведен на Таблице 1.2.

*Таблица 1.2. Текст программы задачи «Расчет значений функции»*

|  |  |
| --- | --- |
| **Номера**  **строк** | **Строки программы** |
| 1 | using System; |
| 2 | namespace Ex |
| 3 | { |
| 4 | class Program |
| 5 | { |
| 6 | static void Main() |
| 7 | { |
| 8 | double x, y, F; |
| 9 | char check; |
| 10 | do |
| 11 | { |
| 12 | Console.WriteLine("Введите значение переменной x"); |
| 13 | Console.Write("x="); |
| 14 | x = double.Parse(Console.ReadLine()); |
| 15 | Console.WriteLine("Введите значение переменной у"); |
| 16 | Console.Write("y="); |
| 17 | у = double.Parse(Console.ReadLine()); |
| 18 | if (x < y) |
| 19 | F = Math.Sin(x) + Math.Cos(y) \* Math.Cos(y); |
| 20 | else |
| 21 | if (x= =y) |
| 22 | F = Math.Log(x); |
| 23 | else |
| 24 | F = Math.Sin(x) \* Math.Sin(x) + Math.Cos(y); |
| 25 | ConsoIe.WriteLine("F = "+F); |
| 26 | Console.WriteLine("Хотите запустить программу вновь? Y/N"); |
| 27 | check=char.Parse(Console.ReadLine()); |
| 28 | }while (check= ='Y' || check= ='y'); |
| 29 | } |
| 30 | } |
| 31 | } |

**Словарь программы**

В таблице 1.3 приведены операторы и операции, используемые в программе (столбец 2). Номера строк исходной программы, где встречается каждый оператор или операция, указаны в третьем столбце. В четвертом столбце указано число повторений каждого оператора или операции в исходном тексте программы.

*Таблица 1.3 Операторы и операции, используемые в программе*

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Операторы,  операции | Номера строк | Количество  повторений |
| 1 | using …; | 1 | 1 |
| 2 | namespace ... | 2 | 1 |
| 3 | class ... | 4 | 1 |
| 4 | static void... | 6 | 1 |
| 5 | double... | 8 | 1 |
| 6 | char... | 9 | 1 |
| 7 | do...whiIe() | 10-28 | 1 |
| 8 | ConsoIe.WriteLine() | 12,15,25,26 | 4 |
| 9 | Console.Write() | 13,16 | 2 |
| 10 | ....Parse() | 14,17,27 | 3 |
| 11 | Console.ReadLine() | 14,17,27 | 3 |
| 12 | if()...else... | 18,21 | 2 |
| 13 | Math.Sin() | 19,24,24 | 3 |
| 14 | Math.Cos() | 19, 19, 24 | 3 |
| 15 | Math.Log() | 22 | 1 |
| 16 | ; | 1, 8, 9, 12, 13, 14, 15, 16, 17, 19, 22, 24, 25, 26, 27, 28 | 16 |
| 17 | , | 8,8 | 2 |
| 18 | \* | 19,24 | 2 |
| 19 | = | 14,17,19,22,24,27 | 6 |
| 20 | + | 19,24,25 | 3 |
| 21 | < | 18 | 1 |
| 22 | = = | 21,28,28 | 3 |
| 23 | {} | 3(31), 5(30), 7(29), 11(28) | 4 |
| 24 | () | 6, 12, 13, 14, 14, 15, 16, 17, 17, 18, 19,  19, 19, 21, 22, 24, 24, 24, 25, 26, 27, 27,  28 | 23 |
| 25 | || | 28 | 1 |
| 26 | " " | 12,13,15,16,25,26 | 6 |
| 27 | ' ' | 28,28 | 2 |
| 28 | . | 12, 13, 14, 14, 15, 16, 17, 17, 19, 19, 19, 22, 24, 24, 24, 25, 26, 27, 27 | 19 |
| **Всего** | | | **116** |

Таким образом, количество строк этой таблицы есть число уникальных операторов и операций, появляющихся в данном тексте. Если вычислить сумму значений из четвертого столбца, то получим общее число всех операторов и операций, используемых в исходном тексте программы.

*Примечание:*

*Для фигурных скобок, определяющих блок, приведены два номера строки. Первый определяет левую фигурную скобку, открывающую блок, а второй - закрывающую. Отметим, что такая пара в словаре учитывается только один раз.*

Словарь всех операторов в исходном тексте этой реализации программы сведен в столбце 2. В третьем столбце этой таблицы приведены номера строк исходной программы, где встречаются операторы. В последнем столбце приводится количество повторений (число вхождений) операторов в тексте программы.

Таким образом, количество строк этой таблицы есть число уникальных операторов программы. Сложив значения четвертого столбца, получим общее число вхождений всех операторов.

Проведем подробный анализ исходного текста программы в соответствии с полученной таблицей, начиная с первой позиции (первая строка программы (using System;).

Ключевое слово using представляет собой команду (инструкцию), обеспечивающую доступ к именам пространства имен System. Следовательно, команду using можно отнести к выполняемым операторам (см. табл. 1.3,   
позиция 1). Оператор using встречается в программе всего один раз. Слово System представляет собой имя, над которым осуществляется операция using. Таким образом, имя System заносится в таблицу словаря операндов (см. табл. 1.4, позиция 1). Имя System встречается в программе один раз.

Следующая строчка программы namespace Ex (см. табл. 1.2, строка 2) состоит из оператора namespace и операнда Ех, которые также присутствуют в тексте программы в единственном экземпляре. Оператор занесен в таблицу операторов (см. табл. 1.3, позиция 2), а операнд Ех - в таблицу операндов (см. табл. 1.4, позиция 2).

Строки

* + - *class Program;*
    - *static void Main();*
    - *double x, y, F;*
    - *char check;*

также представляют собой сочетание операторов и операндов, которые встречаются в тексте один раз (см. табл. 1.2, строки 4, 6 и 8), где ключевые слова class, static void, double и char представляют соответственно операции, в Program, Main, х, у, F и check — имена (операнды). Все операции попадают в словарь операторов (см. табл. 1.3), а имена - в словарь операндов (см. табл. 1.4).

Следующий оператор do ... while*()* (см. табл. 1.2, строки 10-28) представляет собой инструкцию реализации циклического алгоритма, которая используется в тексте программы один раз. Рассмотрим тело цикла (блок операторов, заключенных между ключевыми словами do ... while). Первой строчкой цикла является операция вызова функции вывода строк на экран монитора Console. WriteLine().

Данная операция повторяется в тексте программы 4 раза (см. табл. 1.2, строки 12, 15, 25, 26). В каждом из этих случаев применения оператора вызова функции (метода) Console.WriteLine() входным параметром функции является строка (строковая константа). Значение строковой константы в каждом случае применения оператора разное:

* *"Введите значение переменной х";*
* *"Введите значение переменной у";*
* *"F= ";*
* *"Хотите запустить программу вновь? Y/N".*

Все перечисленные константы являются операндами и заносятся в   
табл. 1.4. Каждый из перечисленных операндов используется один раз.

Следующим по ходу выполнения программы выполняется оператор Console. Write(); вызова функции вывода символов на экран (см. табл. 1.2, строки 13, 16).

Оператор используется 2 раза с разными операндами:

* *"x=";*
* *"у=",*

каждый из которых используется в программе однократно. Оператор включается в таблицу операторов, операнды - в таблицу операндов.

Следующая операция

*х* = *double.Parse(Console.ReadLine());*

включает три оператора:

* = - оператор присваивания (см. табл. 1.2, строки 14, 17, 19, 22, 24, 27) используется в программе 6 раз;
* ... Parse() - оператор вызова функции преобразования строки в заданный тип (см. табл. 1.2, строки 14, 17, 27) используется в программе 3 раза;
* Console.ReadLine() - оператор вызова функции считывания строки с клавиатуры (см. табл. 1.2, строки 14, 17, 27) используется в программе 3 раза.

Следующий оператор if()...else... (см. табл. 1.2, строки 18, 21) используется дважды в тексте программы для ветвления алгоритма.

Операции (см. табл. 1.2, строки 19, 22,24)

* *F* = *Math.Sin(x) + Math. Cos(y)* \* *Math. Cos(y);*
* *F* = *Math.Log(x);*
* F = Math.Sin(x) \* Math.Sin(x) + Math.Cos(y);

включают следующие ранее не рассмотренные операторы:

* Math.Sin(x) - оператор вызова функции вычисления синуса, используется 3 раза;
* Math.Cos(y) - оператор вызова функции вычисления косинуса, применяется 3 раза;
* Math.Log(x) - оператор вызова функции вычисления логарифма, используется один раз.

Имена F, х и у являются операндами: F используется 5 раз, х - 8 раз, у - 7 раз.

Символы «;», «,», «\*» и «+», используемые в программе, обозначают следующие операции:

* ; - операция определения завершения оператора, используется 16 раз;
* , - операция отделения элементов списка, используется 2 раза;
* - операция умножения, используется 2 раза;
* + - операция сложения (сцепления строк), используется 3 раза.

Символы «<» и «= =» используются для определения логических операций сравнения:

* < - операция сравнения «меньше», используется один раз;
* = = - операция сравнения «равно», используется 3 раза.

В позициях 23 и 24 таб. 1.3. представлены символы, определяющие следующие операции:

* {} - операция начала и завершения блока инструкций, используется 4 раза;
* () - операция начала и завершения списка параметров или условия, используется 22 раза.

Оставшиеся четыре позиции табл. 1.3 содержат символы:

* || - операция логического сложения (дизъюнкция), используется один раз;
* " " - операция определения строковых констант, используется 6 раз;
* ' ' - операция определения символьных констант, используется 2 раза;
* . - операция связывания имен, используется 19 раз.

В табл. 1.4 приведены операнды рассматриваемой программы.

*Таблица 1.4. Словарь операндов программы*

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Операнды | Номера строк | Количество  повторений |
| 1 | System | 1 | 1 |
| 2 | Ех | 2 | 1 |
| 3 | Program | 4 | 1 |
| 4 | Main | 6 | 1 |
| 5 | x | 8, 14, 18, 19, 21, 22, 24, 24 | 8 |
| 6 | y | 8, 17, 18, 19, 19, 21, 24 | 7 |
| 7 | check | 9, 27, 28, 28 | 4 |
| 8 | "Введите значение переменной х" | 12 | 1 |
| 9 | "х=" | 13 | 1 |
| 10 | "Введите значение переменной у" | 15 | 1 |
| 11 | "y=" | 16 | 1 |
| 12 | F | 8, 19, 22, 24, 25 | 5 |
| 13 | "Хотите запустить программу вновь? Y/N" | 26 | 1 |
| 14 | 'Y' | 28 | 1 |
| 15 | 'y' | 28 | 1 |
| 16 | "F =" | 25 | 1 |
|  |  | **Всего** | **36** |

Проанализируем содержание табл. 1.4. Позиции 1, 2, 3 и 4 содержат имена операндов System, Ex, Program, Main() которые используются в программе по одному разу.

Строковые константы (позиции 8, 9,10, 11,13 и 16 табл. 1.4):

* *"Введите значение переменной х";*
* *"х=";*
* *"Введите значение переменной у";*
* *"y=";*
* *"Хотите запустить программу вновь? Y/N";*
* "F *= "*

используются в тексте программы однократно (см. табл. 1.2).

Символьные константы *'Y'* и 'у' применяются также по одному разу (см. табл. 1.2). Имена переменных х, у, check и F повторяются в программе соответственно 8, 7, 4 и 5 раз.

Для рассматриваемой программы список входных и выходных параметров (табл. 1.5) не обладает большим разнообразием. Входными параметрами являются значения переменных:

* *х* = *double. Parse(Console.ReadLine());*
* *у = double.Parse(Console.ReadLine());*
* *check=char.Parse(Console.ReadLine()).*

*Таблица 1.5. Входные и выходные переменные программы*

|  |  |
| --- | --- |
| **Входные переменные** | **Выходные переменные** |
| x | "Введите значение переменной х" |
| y | "х=" |
| check | "Введите значение переменной у" |
|  | "у=" |
|  | "F = " |
|  | "Хотите запустить программу вновь? Y/N" |
|  | F |

Выходными значениями являются шесть констант, для которых имена совпадают со значениями и одна переменная F:

* Console.WriteLine("Введите значение переменной х");
* Console.Write("x=");
* Console.WriteLine("BBeflHTe значение переменной у");
* Console.Write("y=");
* Console.WriteLine("F = "+F) - в этом случае два выходных параметра: строковая константа "F *=*" и переменная F;
* Console.WriteLine("Xoтитe запустить программу вновь? Y/N").

**Оценка характеристик программы**

Используя сформированные таблицы с необходимыми параметрами для расчета и применяя соотношения Холстеда, вычислим характеристики рассматриваемой программы:

* словарь программы:

**;**

* длина реализации:

;

* длина программы:

;

* объем программы в битах:

;

* потенциальный объем программы:

;

* уровень реализации программы:

;

* уровень реализации языка:

**;**

* интеллектуальное содержание программы:

**;**

* работа по программированию:

**.**

Сведем все результаты расчетов метрик Холстеда в табл. 1.6.

*Таблица 1.6 Значения метрик Холстеда для программы*

|  |  |  |
| --- | --- | --- |
| **Наименование**  **характеристики** | **Обозначение и формула для вычисления** | **Значение** |
| Число простых (уникальных) операторов и операций |  | 28 |
| Число простых (уникальных) операндов |  | 16 |
| Общее число всех операторов и операций |  | 116 |
| Общее число всех операндов |  | 36 |
| Число входных и выходных переменных (параметров) |  | 10 |
| Словарь программы |  | 44 |
| Длина реализации программы |  | 152 |
| Объем программы (в битах) |  | 830 |
| Потенциальный объем программы |  | 43 |
| Уровень реализации программы |  | 0,052 |
| Уровень реализации языка |  | 2,23 |
| Работа программирования |  | 15960 |

**Вывод:** Уровень исследуемой программы весьма низкий, так как потенциальный объем программы в значительной степени меньше ее реального объема.

# 2. Задание

При оценке характеристик программ на основе лексического анализа с использованием метрики Холстеда необходимо выполнить следующее:

* 1. разработать программу, реализующую заданный, в соответствии с вариантом, алгоритм (рекомендуется использовать язык программирования С#);
  2. сформировать словарь программы, охватывающий операнды, а также операторы и операции;
  3. словари оформить в виде таких же таблиц, как в рассмотренном примере;
  4. рассчитать метрики Холстеда, оформив результат в виде итоговой таблицы;
  5. провести анализ полученных результатов, сформировав содержательные выводы.

Практическая работа выполняется в соответствии с вариантом (номер варианта соответствует номеру студента в журнале группы).

**Варианты заданий**

**Вариант 1.** Написать и протестировать функцию, которая «переворачивает» строку, передаваемую ей в качестве параметра, в зеркальное состояние.

**Вариант 2.** Дано натуральное число N. Вывести на экран число, которое получится после выписывания цифр числа N в обратном порядке. Для получения нового числа составить функцию.

**Вариант 3.** Написать и протестировать функцию, подсчитывающую количество минимальных элементов в каждой строке целочисленной матрицы.

**Вариант 4.** Написать и протестировать функцию для вычисления числа сочетаний по формуле

**Вариант 5**. По заданным значениям *X[20], Y[20]* вычислить

**Вариант 6**. Написать и протестировать функцию, преобразующую строку восьмеричных цифр в эквивалентное ей целое десятичное число.

**Вариант 7**. Описать функцию *minmax(x,y)*, которая присваивает первому параметру большее, а второму - меньшее из значений *х* и *у*. Используя эту функцию, перераспределить введенные значения переменных А, В, С так, чтобы стало А < В < С.

**Вариант 8**. Даны две квадратные матрицы. Напечатать ту из них, которая имеет минимальный «след», т. е. сумму элементов главной диагонали. Использовать функцию для нахождения следа матрицы и функцию печати матрицы.

**Вариант 9**. Составить и протестировать функцию для вычисления

**Вариант 10**. Написать и протестировать функцию *compress()*, которая «сжимает» строку, удаляя из нее все пробелы.

**Вариант 11**. Написать и протестировать функцию, которая подсчитывает, сколько раз в заданной строке встретился указанный символ.

**Вариант 12**. Написать и протестировать функцию, которая находит в массиве минимальный по модулю элемент и заменяет им все элементы с нечетными номерами.

**Вариант 13**. Написать и протестировать функцию, которая в прямоугольной матрице находит сумму элементов j-й строки.

**Вариант 14**. Написать и протестировать функцию, которая по заданному натуральному числу определяет количество цифр в нем и их сумму.

**Вариант 15**. Написать и протестировать функцию, которая по заданной строке Str, содержащей буквы и цифры, формирует новую, состоящую только из цифр, входящих в Str.

**Вариант 16**. Написать и протестировать функцию, подсчитывающую количество положительных элементов в массиве.

**Вариант 17**. Написать и протестировать функцию, вычисляющую используя итерационную формулу

Начальное приближение . Итерации прекратить при достижении условия .

**Вариант 18**. Составить и протестировать функцию для замены символов «:» на «.» в заданной строке, начиная с указанной позиции.

**Вариант 19**. Выяснить, сколько простых чисел находится в интервале *[n,m]*, и распечатать их. Для определения, является ли очередное число простым, составить функцию.

**Вариант 20**. Написать и протестировать функцию для вычисления площади треугольника, заданного координатами вершин.

**Вариант 21**. Написать и протестировать функцию для нахождения в прямоугольной матрице номера строки, имеющей максимальную сумму элементов.

**Вариант 22**. Написать и протестировать функцию, которая преобразует строку двоичных цифр в эквивалентное ей целое десятичное число.

**Вариант 23**. Написать и протестировать функцию, которая в строке, передаваемой ей в качестве параметра, заменяет каждый второй элемент на заданный символ.

**Вариант 24**. Написать и протестировать функцию для сложения и вычитания вещественных матриц. Одним из формальных параметров должен быть признак вида операции.

**Вариант 25**. Написать и протестировать функцию, которая преобразует строку шестнадцатеричных элементов числа в эквивалентное ей целое десятичное число.

# 3. Требования к отчету

Отчёт выполняется каждым студентом индивидуально. Работа должна быть оформлена в электронном виде в формате .doc и распечатана на листах формата А4. На титульном листе указываются: наименование учебного учреждения, наименование дисциплины, название и номер работы, вариант, выполнил: фамилия, имя, отчество, группа, проверил: преподаватель ФИО (образец титульного листа представлен в приложении 1).

Отчет должен содержать:

* титульный лист;
* цель работы;
* краткие теоретические сведения, ответы на контрольные вопросы;
* задания в соответствии с вариантом;
* выводы.

# 4. Контрольные вопросы

1. Какие метрические характеристики включены в состав измеримых свойств алгоритма (или программы)?
2. Каким условиям удовлетворяют тексты программ?
3. Каков смысл величины - длина реализации программы?
4. В чем измеряется длина программы и объем программы?
5. Как найти объем программы?
6. Какие элементы программы относят к словарю операторов?
7. Что такое потенциальный объем программы? В чем заключается смысл этого понятия?

# 5. Литература

1. Керниган Б., Ритчи Д. Язык программирования С: Пер. с англ. М.: Вильямс, 2009.

2. Майерс Г. Надежность программного обеспечения. М.: Мир, 1980.

3. Холстед М.Х. Начала науки о программах. М.: Финансы и статистика, 1981.

4. Черников Б.В. Управление качеством программного обеспечения: учебник. М.: ИД «ФОРУМ»: ИНФРА-М, 2012.

**Приложение 1**



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет Информатика и вычислительная техника

Кафедра Кибербезопасность информационных систем

**Практическое занятие № \_\_\_\_**

на тему «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

Выполнил обучающийся гр.\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, Имя, Отчество)

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, Фамилия, Имя, Отчество)

Ростов-на-Дону

201\_