ОЦЕНКА ХАРАКТЕРИСТИК ПРОГРАММ НА ОСНОВЕ ЛЕКСИЧЕСКОГО АНАЛИЗА

Метрики Холстеда

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

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:

- 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.

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:

- η_1 – число простых (уникальных) операторов, появляющихся в данной реализации;
- η_2 – число простых (уникальных) операндов, появляющихся в данной реализации;
- N_1 – общее число всех операторов, появляющихся в данной реализации;
- N_2 – общее число всех операндов, появляющихся в данной реализации;
- f_{1j} – число появлений в программе j-го оператора, где $j = 1,2,3, \dots, \eta_1$;
- f_{1j} – число появлений в программе j-го оператора, где $j = 1,2,3, \dots, \eta_2$.

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

- словарь $\eta = \eta_1 + \eta_2$;
- длину реализации программы $N = N_1 + N_2$;
- длину программы $\widetilde{N} = (\eta_1 \cdot log_2\eta_1) + (\eta_2 \cdot log_2\eta_2)$.

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 $A + C - A$. Понятно, что в подобном случае будет выполнено два совершенно ненужных действия, дополняющих переменную C одной и той же величиной, взятой с противоположными знаками.
2. Наличие неоднозначных операндов, например $A = D$ и $A = C$. 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 $A = B$ и $C = B$. Поскольку одно и то же значение должно быть присвоено разным переменным, то для данного примера переменная B вообще может не использоваться. Более лаконичным вариантом является простое приравнивание значений переменных A и C.
4. Наличие общих подвыражений, например:

$$(A + B) \cdot C + D \cdot (A + B).$$

Здесь применено совсем не обязательное повторение суммирования переменных A и B , что приводит к дополнительному времени выполнения программы.

5. Ненужное присваивание, например $\mathrm{C} = \mathrm{A} + \mathrm{B}$, если переменная C 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:

$$X \cdot X + 2 \cdot X \cdot Y + Y \cdot Y$$

Данное преобразование можно представить как

$$(\mathrm{X} + \mathrm{Y}) \cdot (\mathrm{X} + \mathrm{Y}),$$

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:

$$N_1 = \sum_{j=1}^{\eta} f_{1,j};$$

$$N_2 = \sum_{j=1}^{\eta} f_{2,j};$$

$$N = \sum_{i=1}^{2} \sum_{j=1}^{\eta} f_{i,j}.$$

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 η слов, то для задания номера любого из них требуется минимум $log_2\eta$ бит.

Объем программы определяется следующим образом:

$$V = N \cdot log_2\eta = \eta \cdot log_2^2\eta$$

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

$$N \approx \eta \cdot log_2\eta;$$
$$V = \eta \cdot log_2^2\eta.$$

В первом случае зафиксирована взаимная связь между длиной программы N и размером словаря η, во втором – между величиной словаря и объемом программы V. Следовательно, по известному размеру словаря η 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:

$$N = \eta_1 \cdot log_2\eta + \eta_2 \cdot log_2\eta \approx \eta_1 \cdot log_2\eta_1 + \eta_2 \cdot log_2\eta_2 = N_1 + N_2$$

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 η_1 и η_2 независимы и могут принимать произвольные значения. Однако этого нельзя сказать относительно N_1 и N_2, т. е. числа появления всех операторов $N_1 = \eta_1 \cdot log_2\eta_1$, и всех операндов $N_2 = \eta_2 \cdot log_2\eta_2$ 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 $N_1 \approx N_2$, хотя величины словарей η_1 и η_2 могут сильно отличаться друг от друга. Это позволяет прийти к весьма важному практическому выводу относительно объема программы:

$$N \approx 2 \cdot N_2 = 2 \cdot \eta_2 \cdot log_2 \eta_2.$$

Основной исходный параметр, на котором базируются все расчеты метрических характеристик будущего ПС, – количество имен входных и выходных переменных η_2^*, 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 η_2^*, определенный таким образом, рассматривать как размер генеральной совокупности имен входных и выходных переменных, то величина словаря программы η_2 в соответствии с соотношением Холстеда не будет превосходить $\eta_2^* \cdot log_2 \eta_2^*$. Таким образом, можно считать, что

$$\eta_2 \approx \eta_2^* \cdot log_2 \eta_2^*$$

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 η_2^* 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:

$$V^* = (\eta_2^* + 2) \cdot log_2(\eta_2^* + 2)$$

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

$$L = \frac{V^*}{V}$$

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 V^* не изменяется, но действительный объем V может увеличиваться или уменьшаться в зависимости от развитости языков программирования.

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

Оценка уровня языков программирования

Если N - длина программы, а η— словарь программы, то общее количество выборок необходимых элементов словаря (т. е.

фактически – *работа программирования*) в соответствии с законом Хика будет равна объему программы $N \cdot log\eta$.

В то же время необходимо еще учесть уровень реализации программы L: количество выборок при этом возрастет в $^1/_L$ раз. Обозначив работу программирования символом E и учитывая формулу для вычисления уровня реализации программы, окончательно получим:

$$E = \frac{N \cdot log_2\eta}{L} = \frac{V}{L} = \frac{V^2}{V^*}$$

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

$$\lambda = L \cdot V^*,$$

где L - уровень реализации программы, V^* - 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 V^* 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 V^* уровень программы L уменьшится в том же отношении. Следовательно, произведение L на V^* 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:

$$V^* = (\eta_2^* + 2)\log(\eta_2^* + 2)$$

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

$$2^1 + 2^2 + 2^3 + \cdots + 2^{V-1} + 2^V \approx 2^{V+1},$$

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 $\log 2^{V+1} \approx V$, и полная работа программирования при использовании потенциального языка будет определяться следующим образом:

$$E_{\text{пот}} = V^* \cdot V.$$

В таком случае мы получим следующее соотношение:

$$\lambda = \frac{E_{\text{пот}}}{E} = \frac{V^{*2}}{V}$$

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

Из последнего выражения видно, что для постоянства λ, 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 λ, чем программы вычисления большого числа переменных по элементарным выражениям.

В связи с этим метрику уровня языка программирования λ 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).

Таблица 1

Уровни языков программирования

Язык	λ	**Отклонения**
Естественный 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 E и уровня языка λ

$$E = \frac{V^2}{V^*} \text{ и } \lambda = \frac{V^{*2}}{V},$$

то, исключая из выражений величину V, получим:

$$E = \frac{V^{*3}}{\lambda^2}.$$

Тогда квалификационное время программирования будет определяться следующим образом:

$$T = \frac{E}{S} = \frac{V^{*3}}{S\lambda^2},$$

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

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 E. Следовательно, число моментов, в которые можно сделать ошибочное различение, также определяется значением E 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. $\eta_2^* = 6$.

С помощью равенств

$$\eta_1^* = 2 \text{ и } V^* = (\eta_1^* + \eta_2^*) log_2(\eta_1^* + \eta_2^*)$$

получаем предельное значение потенциального объема:

$$V_{\text{крит}}^* = (2 + 6) \cdot log_2(2 + 6) = 24$$

Далее из уравнения уровня языка программирования имеем:

$$E = \frac{V^{*3}}{\lambda^2}$$

а из таблицы уровней языка известно, что для естественного (английского) языка $\lambda = 2{,}16$.

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

$$E_{\text{крит}} = \frac{24^3}{2.16^2} = 3000.$$

Определим теперь E_0 как среднее число элементарных различений между возможными ошибками в программировании, а B - как число переданных ошибок в программе. Можно ожидать, что

$$B = \frac{E}{E_0},$$

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 последнего полученного уравнения реальнее ожидать, что на количество ошибок в программе будет влиять и сам язык программирования, который мы можем учесть при помощи уровня реализации программы:

$$B = L \cdot \frac{E}{E_0}$$

Учитывая, что произведение $L \cdot E$ можно заменить на значение объема программы V, получим:

$$B = \frac{V}{E_0}$$

Если теперь приравнять E_0 значению $E_{\text{крит}}$, получим соотношение

$$B = \frac{V}{3000}$$

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

$$B = \frac{V^{*2}}{3000 \cdot \lambda}$$

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:

$$B_0 = \frac{E}{E_0} = \frac{V}{3000}$$

где V- расчетный объем программного средства.

ПРИМЕР ОЦЕНКИ ХАРАКТЕРИСТИК

Задача «Расчет значений функции»

Разработать программу для вычисления значений заданной функции F:

$$F = \begin{cases} \sin(x) + cos^2(y), \text{при } x < y; \\ \ln(x), \text{при } x = y; \\ sin^2(x) + \cos(y), \text{при } x > y. \end{cases}$$

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

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 C#, приведен на рис. 1.

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 y");
16	Console.Write("y=");
17	y = 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	Console.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	}

Рис. 1. Текст программы

Словарь программы

В табл. 2 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.

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. Отметим, что 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.

Таблица 2.

Словарь операторов и операций программы

№ 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...while()	10 – 28	1
8	Console.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**

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. 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 (табл. 3). Имя $System$ встречается в программе один раз.

Следующая строчка программы $namespace\ Ex$ состоит из оператора $namespace$ и операнда Ex , которые также присутствуют в тексте программы в единственном экземпляре. Оператор занесен в таблицу операторов, а операнд Ex - в таблицу.

Строки

- $class\ Program;$
- $static\ void\ MainQ;$
- $double\ x, y, F;$
- $char\ check;$

также представляют собой сочетание операторов и операндов, которые встречаются в тексте один раз, где ключевые слова $class$, $static\ \ void$, $double$ и $char$ представляют соответственно операции, а $Program$, $Main$, x, y, F и $check$ – имена (операнды). Все операции попадают в словарь операторов, а имена – в словарь операндов.

Следующий оператор $do \ldots while()$ представляет собой инструкцию реализации циклического алгоритма, которая

используется в тексте программы один раз. Рассмотрим тело цикла (блок операторов, заключенных между ключевыми словами $do \ldots while$). Первой строчкой цикла является операция вызова функции вывода строк на экран монитора $Console.WriteLine()$.

Данная операция повторяется в тексте программы 4 раза. 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 x ";
- "Введите значение переменной y ";
- "$F =$ ";
- "Хотите запустить программу вновь? Y/N".

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

Следующим по ходу выполнения программы выполняется оператор $Console.Write()$; вызова функции вывода символов на экран.

Оператор используется 2 раза с разными операндами:

- "x = ";
- "y = ",

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

Следующая операция

$$x = double.Parse(Console.ReadLine());$$

включает три оператора:

- $=$ – оператор присваивания используется в программе 6 раз;
- $\ldots Parse()$ – оператор вызова функции преобразования строки в заданный тип используется в программе 3 раза;
- $Console.ReadLine()$ – оператор вызова функции считывания строки с клавиатуры используется в программе 3 раза.

Следующий оператор $if() \ldots else \ldots$ используется дважды в тексте программы для ветвления алгоритма.

Операции

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

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, x и y являются операндами: F используется 5 раз, x - 8 раз, y -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 табл. 2. 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. 2 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. 3 приведены операнды рассматриваемой программы.

Таблица 3

Словарь операндов программы

№ п/п	Операнды	Номера строк	Количество повторений
1	System	1	1
2	Ex	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 x"	12	1
9	"x="	13	1
10	"Введите значение переменной y"	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. 3. 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. 3):

- "Введите значение переменной x ";
- "x=";
- Введите значение переменной y ;
- "y = ";
- "Хотите запустить программу вновь? Y/N";

- "F = ";

используются в тексте программы однократно.

Символьные константы $'Y'$ и $'$у$'$ применяются также по одному разу. Имена переменных x, у, $check$ и F повторяются в программе соответственно 8,7,4 и 5 раз.

Для рассматриваемой программы список входных и выходных параметров (табл. 4) 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 4.

Входные и выходные переменные программы

Входные переменные	**Выходные переменные**
X	"Введите значение переменной х"
у	"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 = ");
- $Console.WriteLine$("Введите значение переменной у");
- $Console.Write$("y = ");
- $Console.WriteLine$("F = " + F) – в этом случае два выходных параметра: строковая константа "F = " и переменная F;
- $Console.WriteLine$("Хотите 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:
$$n = n_1 + n_2 = 28 + 16 = 44;$$
- длина реализации:
$$N = N_1 + N_2 = 116 + 36 = 152;$$
- длина программы:
$$\widetilde{N} = n_1 \cdot \log_2 n_1 + n_2 \cdot \log_2 n_2 = 28 \cdot log_2 28 + 16 \cdot log_2 16 = 28 \cdot 4.81 + 16 \cdot 4 = 134.68 + 64 = 198.68;$$
- объем программы в битах:
$$V = (N_1 + N_2) \cdot log_2(n_1 + n_2) = (116 + 36) \cdot log_2(28 + 16) = 152 \cdot log_2 44 = 152 \cdot 5.46 = 829.92;$$
- потенциальный объем программы:
$$V^* = (n_2^* + 2) \cdot log_2(n_2^* + 2) = (10 + 2) \cdot log_2(10 + 2) = 12 \cdot 3.58 = 42.96;$$
- уровень программы:
$$L = \frac{V^*}{V} = \frac{42.96}{829.92} 0.052;$$
- уровень языка:
$$\lambda = L \cdot V^* = 0.052 \cdot 42.96 = 2.23;$$
- интеллектуальное содержание программы:
$$I = L \cdot V = 0.052 \cdot 829.92 = 43.16;$$
- работа по программированию:
$$E = \frac{V}{L} = \frac{829.92}{0.052} = 15960.$$

Сведем все результаты расчетов метрик Холстеда в табл. 5.

Таблица 5.

Значения метрик Холстеда для программы

Наименование характеристики	Обозначение и формула для вычисления	Значение
Число простых (уникальных) операторов и операций	n_1	28
Число простых (уникальных) операндов	n_2	16
Общее число всех операторов и операций	N_1	116
Общее число всех операндов	N_2	36

Наименование характеристики	Обозначение и формула для вычисления	Значение
Число входных и выходных переменных (параметров)	n_2^*	10
Словарь программы	$n = n_1 + n_2$	44
Длина реализации программы	$N = N_1 + N_2$	152
Объем программы (в битах)	$V = (N_1 + N_2) \cdot \log_2(n_1 + n_2)$	830
Потенциальный объем программы	$V^* = (n_2^* + 2) \cdot \log_2(n_2^* + 2)$	43
Уровень реализации программы	$L = V^*/V$	0,052
Уровень реализации языка	$\lambda = L \cdot V^*$	2,23
Работа программирования	$E = V/L$	15960

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

Задача «Зеркальное число»

Ввести с клавиатуры трехзначное натуральное число. Вычислить и вывести на экран зеркальное число, полученное путем изменения цифр числа на обратное по отношению к исходной позиции (например, для числа 253 должно быть получено 352). Разработать программу и 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 C#, приведен на рис. 3.

Номера строк	Строки программы
1	using System;
2	class Revers
3	{
4	public static void Main()
5	{
6	uint ch, copia, cifra, newch;
7	string str;
8	Console.Write("Введите трехзначное натуральное число: ");
9	str = Console.ReadLine();
10	ch = uint.Parse(str);
11	copia = ch;
12	newch = 0;
13	cifra = copia % 10;
14	newch = newch*10 + cifra;
15	copia /= 10;
16	cifra = copia % 10;
17	newch = newch*10 + cifra;
18	copia /= 10;
19	cifra = copia % 10;
20	newch = newch*10 + cifra;
21	copia /= 10;
22	str = "Если перевернуть " + ch + " будет " + newch;
23	Console.WriteLine(str);
24	Console.ReadLine();
25	}
26	}

Рис. 3. Текст программы задачи «Зеркальное число»

Словарь программы

В табл. 6 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 6

Словарь операторов и операций программы

№ п/п	Операторы, операции	Номера строк	Количество повторений
1	using ...;	1	1
2	class …	2	1
3	static void…	4	1
4	uint…	6	1
5	string …	7	1
6	Console.Write()	8	1
7	Console.WriteLine()	23	1
8	Console.ReadLine()	9, 24	2
9	….Parse()	10	1
10	;	1, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24	20
11	,	6, 6, 6	3
12	*	14, 17, 20	3
13	=	9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22	14
14	+	14, 17, 20, 22, 22, 22	6
15	{}	3(26), 5(25)	2
16	()	4, 8, 9, 10, 23, 24	6
17	“ ”	8, 22, 22	3
18	/	15, 18, 21	3
19	.	8, 9, 10, 23, 24	5
20	%	13, 16	2
		Всего	**77**

В таб. 7 приведены операнды рассматриваемой программы.

Таблица 7.

Словарь операндов программы

№ п/п	Операнды	Номера строк	Количество повторений
1	System	1	1
2	Revers	2	1
3	Main	4	1
4	ch	6, 10, 11, 22	4
5	copia	6, 11, 13, 15, 16, 18, 21	7
6	cifra	6, 13, 14, 16, 17, 19, 20	7
7	newch	6, 12, 14, 14, 17, 17, 20, 20, 22	9
8	str	7, 9, 10, 22, 23	5
9	"Введите трехзначное натуральное число: "	8	1
10	"Если перевернуть "	22	1
11	" будет "	22	1
12	10	13, 14, 15, 16, 17, 18, 19, 20, 21	9
		Всего	**47**

Для рассматриваемой программы список входных и выходных параметров (табл. 8) включает две переменные и одну константу.

Таблица 8

Входные и выходные переменные программы

Входные переменные	Выходные переменные
str	*str*
	"Введите трехзначное натуральное число: "

Оценка характеристик программы

Используя сформированные таблицы с необходимыми параметрами и для расчета и применяя соотношения Холстеда, вычислим характеристики рассматриваемой программы (табл. 9).

Таблица 9.

Значения метрик Холстеда для программы

Наименование характеристики	Обозначение и формула для вычисления	Значение
Число простых (уникальных) операторов и операций	n_1	20
Число простых (уникальных) операндов	n_2	12
Общее число всех операторов и операций	N_1	77
Общее число всех операндов	N_2	47
Общее число входных и выходных переменных (параметров)	n_2^*	3
Словарь программы	$n = n_1 + n_2$	32
Длина реализации	$N = N_1 + N_2$	124
Длина программы	$\tilde{N} = n_1 \cdot \log_2 n_1 + n_2 \cdot \log_2 n_2$	129,45
Объем программы (в битах)	$V = (N_2 + N_1) \cdot \log_2(n_1 + n_2)$	620
Потенциальный объем программы	$V^* = (n_2^* + 2) \cdot \log_2(n_2^* + 2)$	11,6
Уровень реализации программы	$L = V^* / V$	0,019
Уровень языка	$\lambda = L \cdot V^*$	0,22
Работа по программированию	$E = V / L$	32631,58

Анализируя полученные показатели, можно сказать, что уровень исследуемой программы весьма низкий, так как потенциальный объем программы в значительной степени меньше ее реального объема, данном решении возможности языка программирования C# использованы на низком уровне. Длина реализации значительно меньше расчетной длины программы. Следовательно, несовершенства программирования в представленном решении задачи отсутствуют.

Задача «Заправка бака топливом

Определить класс «Бак», описывающий понятие «Топливный бак». Данный класс должен иметь следующие поля:

- ширина, длина и высота в сантиметрах;
- вид топлива.

В классе должны присутствовать операции (методы):

- полная заправка бака заданным видом топлива;
- вычисление стоимости полной заправки бака.

Бак имеет вид параллелепипеда. Стандартным считается бак, имеющий вид куба (ширина, длина и высота совпадают).

Возможные разновидности баков:

- бак с одинаковой шириной и длиной и индивидуальной высотой;
- бак с индивидуальными размерами по ширине, длине и высоте.

Стоимость одного кубического сантиметра топлива определяется полями класса «Топливо». Стоимость топлива в рамках решения задачи неизменна. Выдача стоимости топлива должна быть реализована специальной операцией этого класса.

Заполнить и вывести на экран стоимость заправки четырех баков:

- 10x20x30 см, топливо - газ;
- 10x 10x20 см, топливо - керосин;
- 10x10x10 см, топливо - бензин;
- 20x20x20 см, топливо - бензин.

Разработать программу и определить значения метрик Холстеда, на основе которых дать оценку качества разработанного исходного текста программы.

Реализация программы

Текст программы приведен на рис. 4

Номера строк	Строки программы
1	using System;
2	namespace EX2
3	{
4	class Топливо
5	{
6	private static double ценаГаз = 0.05;
7	private static double ценаКеросин = 0.08;
8	private static double ценаБензин = 0.1;
9	public static double Цена(string вид)
10	{
11	switch (вид)
12	{
13	case “газ”: return ценаГаз;
14	case “керосин”: return ценаКеросин;
15	case “бензин”: return ценаБензин;
16	default: return 0.0;
17	}
18	}
19	}
20	class Бак

```
21  {
22  private double x, y, h;
23  private string видТоплива;
24  public void Заполнить(double xb, string вид)
25  {
26  x = xb; y = xb; h = xb; видТоплива = вид;
27  }
28  public void Заполнить(ref double xb, string вид)
29  {
30  x = xb; y = xb; h = xb; видТоплива = вид;
31  }
32  public void Заполнить(double xb, double yb, string вид)
33  {
34  x = xb; y = yb; h = xb; видТоплива = вид;
35  }
36  public void Заполнить(double xb, double yb, double hb, string вид)
37  {
38  x = xb; y = yb; h = hb; видТоплива = вид;
39  }
40  public double Оплата()
41  {
42  return x * y * h * Топливо.Цена(видТоплива);
43  }
44  }
45  class Program
46  {
47  static void Main()
48  {
49  double x = 20.0;
50  Бак b;
51  b = new Бак();
52  b.Заполнить(10.0,20.0,30.0, “газ”);
53  Console.WriteLine(«Стоимость заправки: « + b.Оплата());
54  b.Заполнить(10.0, 20.0,»керосин»);
55  Console.WriteLine(«Стоимость заправки: « + b.Оплата());
56  b.Заполнить(10.0,»бензин»);
57  Console.WriteLine(«Стоимость заправки: « + b.Оплата());
58  b.Заполнить(ref x, «бензин»);
59  Console.WriteLine(«Стоимость заправки: « + b.Оплата());
60  Console.ReadKey();
61  }
62  }
63  }
```

Рис. 4. Текст программы задачи «заправка бака топливом»

Словарь программы

В табл. 10 приведены операторы и операции, используемые в программе.

Таблица 10

Словарь операторов и операций программы

№ п/п	Операторы, операции	Номера строк	Количество повторений
1	using…	1	1
2	namespace	2	1
3	class …	4	1
4	double…	24, 28, 32, 36, 40	5
5	void…	9, 16, 23, 34, 47	5
6	double	6, 7, 8, 22, 24, 28, 32, 32, 36, 36, 36, 49	12
7	string …	9, 23, 24, 28, 32, 36	6
8	Бак…	50	1
9	Console.WriteLine()	53, 55, 57, 59	4
10	Console.ReadKey()	60	1
11	new Бак()	51	1
12	Заполнить()	52, 54, 56, 58	4
13	Оплата()	53, 55, 57, 59	4
14	return	13, 14, 15, 16, 42	5
15	Топливо.Цена()	42	1
16	;	6, 7, 8, 13, 14, 15, 16, 22, 23, 26, 26, 26, 26, 30, 30, 30, 30, 34, 34, 34, 34, 38, 38, 38, 38, 42, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60	38
17	,	22, 22, 24, 28, 32, 32, 36, 36, 36, 52, 52, 52, 54, 54, 56, 58	16
18	*	42, 42, 42	3
19	=	6, 7, 8, 26, 26, 26, 26, 30, 30, 30, 30, 34, 34, 34, 34, 38, 38, 38, 38, 49, 51	21
20	:	13, 14, 15, 16	4
21	+	53, 55, 57, 59	4
22	{}	3(63), 5(19), 10(18), 12(17), 21(44), 25(27), 29(31), 33(35), 37(39), 41(43), 46(62), 48(61)	12
23	()	9, 24, 28, 32, 36, 40, 42, 47, 51, 52, 53, 53, 54, 55, 55, 56, 57, 57, 58, 59, 59, 60	22
24	“ ”	13, 14, 15, 52, 53, 54, 55, 56, 57, 58	10
25	switch (вид)	11	1

№ п/п	Операторы, операции	Номера строк	Количество повторений
26	.	6, 7, 8, 16, 42, 49, 52, 52, 52, 53, 54, 54, 54, 55, 56, 56, 57, 57, 58, 59, 59, 60	21
27	case	13, 14, 15	3
28	default	16	1
		Всего	**208**

В табл. 11 приведены операнды рассматриваемой программы.

Таблица 11

Словарь операндов программы

№ п/п	Операнды	Номера строк	Количество повторений
1	System	1	1
2	EX2	2	1
3	Топливо	4	1
4	ценаГаз	6, 13	2
5	0.05	6	1
6	ценаКеросин	7, 14	2
7	ценаБензин	8, 15	2
8	0.08	7	1
9	0.1	8	1
10	Цена	9	1
11	вид	9, 11, 24, 26, 28, 30, 32, 34, 36, 38	10
12	"газ"	13, 52	2
13	"керосин"	14, 54	2
14	"бензин"	15, 56, 58	3
15	0.0	16	1
16	x	22, 26, 30, 34, 38, 42, 49, 58	8
17	y	22, 26, 30, 34, 38, 42	6
18	h	22, 26, 30, 34, 38, 42	6
19	видТоплива	23, 26, 30, 34, 38, 42	6
20	Заполнить	24	1
21	xb	24, 26, 26, 28, 30, 30, 30, 32, 34, 34, 36, 38	12
22	yb	32, 34, 36, 38	4
23	hb	36, 38	2
24	20.0	49, 52, 54	3
25	10.0	52, 54, 56	3
26	b	50, 51, 52, 53, 54, 55, 56, 57, 58, 59	10
27	Main	47	1
		Всего	**93**

Для рассматриваемой программы список входных и выходных параметров (табл. 12) включает одну строковую константу и возвращаемое значение метода Оплата().

Таблица 20

Входные и выходные переменные программы

Входные переменные	Выходные переменные
	"Стоимость заправки: "
	b.Оплата()

Оценка характеристик программы

Используя сформированные таблицы с необходимыми параметрами для расчета и применяя соотношения Холстеда, вычислим характеристики рассматриваемой программы (табл. 21).

Таблица 21

Значение метрик Холстеда для программы

Наименование характеристики	Обозначение и формула для вычисления	Значение
Число простых (уникальных) операторов и операций	n_1	28
Число простых (уникальных) операндов	n_2	27
Общее число всех операторов и операций	N_1	208
Общее число всех операндов	N_2	93
Общее число входных и выходных переменных (параметров)	n_2^*	2
Словарь программы	$n = n_1 + n_2$	55
Длина реализации	$N = N_1 + N_2$	301
Длина программы	$\tilde{N} = n_1 \cdot \log_2 n_1 + n_2 \cdot \log_2 n_2$	263
Объем программы (в битах)	$V = (N_2 + N_1) \cdot \log_2(n_1 + n_2)$	1740
Потенциальный объем программы	$V^* = (n_2^* + 2) \cdot \log_2(n_2^* + 2)$	4
Уровень реализации программы	$L = V^* / V$	0,002
Уровень языка	$\lambda = L \cdot V^*$	0,008
Работа по программированию	$E = V / L$	757065

Уровень исследуемой программы весьма низкий, так как потенциальный объем программы в значительной степени меньше ее реального объема. В данном решении возможности языка программирования C# использованы на низком уровне. Длина реализации превышает расчетную длину программы больше чем

на 10%. В программе присутствуют несовершенства. Проявлением несовершенства программы являются строки:

x = xb; y = xb; h = xb;

x = xb; h = xb;.

Представленная последовательность присваиваний представляет собой набор операций с синонимичными операндами, что в значительной степени снижает качественные характеристики программы.

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

В задачах, предлагаемых для самостоятельного решения, необходимо выполнить следующее:

- разработать программу, реализующую заданный алгоритм (рекомендуется использовать язык программирования C#);
- сформировать словарь программы, охватывающий операнды, а также операторы и операции;
- словари оформить в виде таких же таблиц, как в рассмотренных примерах;
- рассчитать метрики Холстеда, оформив результат в виде итоговой таблицы;
- провести анализ полученных результатов, сформировав содержательные выводы.

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

Задача 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.

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

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

$$C_n^k = \frac{n!}{k!\,(n-k)!} = \frac{n(n-1)\ldots(n-k+1)}{k!}.$$

Задача 5. По заданным значениям $X[20], Y[20]$ вычислить

$$u = \begin{cases} \sum_{i=0}^{19} x_i^2 \text{ ; при } \sum_{i=1}^{15} x_i \cdot y_i > 0 \text{ ;} \\ \sum_{i=10}^{19} y_i^2 \text{ ; при } \sum_{i=1}^{15} x_i \cdot y_i \le 0. \end{cases}$$

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

Задача 7. Описать функцию $minmax(x, y)$, которая присваивает первому параметру большее, а второму – меньшее из значений x и y . Используя эту функцию, перераспределить введенные значения переменных A, B, C так, чтобы стало $A \le B \le C$.

Задача 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.

Задача 9. Составить и протестировать функцию для вычисления

$$f(x, n, m) = \sum_{i=m}^{n} \frac{x^{2i}}{(2 \cdot i)!}$$

Задача 10. Написать и протестировать функцию $compres()$, которая «сжимает» строку, удаляя из нее все пробелы.

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

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

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

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

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

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

Задача 17. Написать и протестировать функцию, вычисляющую $y = \sqrt[3]{x}\ (0 < |x| < 2)$, используя итерационную формулу

$$y_{i+1} = y_i + \frac{1}{3}(y_i - \frac{y_i^4}{x})$$

Начальное приближение $y_0 = x$. Итерации прекратить при достижении условия

$$|y_{i+1} - y_i| < 2 \cdot 10^{-6}$$

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

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

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

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

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

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

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

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