

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)**

Факультет Информатика и вычислительная техника

(наименование факультета)

Кафедра Кибербезопасность информационных систем

(наименование кафедры)

**Практическое занятие № 2**

на тему «Оценка характеристик программ на   
основе лексического анализа с использованием метрики Джилба»

Выполнил обучающийся гр. ВКБ42

Михайлов Александр Сергеевич

Проверил:

Куликова Ольга Витальевна

Ростов-на-Дону

2024

**Цель работы:**

Изучить методику оценки характеристик программ с использованием метрики Джилба.

.

**Контрольные вопросы:**

1. Что в метрике Джилба является мерой логической трудности, какое практическое значение эта мера имеет?

Логическая сложность программы Джилб определяет как насыщенность программы условными операторами типа ***IF-THEN-ELSE*** и операторами цикла (при этом следует учитывать, что фактическая запись условий и циклов в разных языках программирования может быть представлена в разной форме при сохранении указанного смысла операторов). При этом вводятся следующие характеристики программного средства:

• ***CL*** - абсолютная сложность программы, характеризуемая количеством операторов условий;

• ***cl*** - относительная сложность программы, определяющая насыщенность программы операторами условия (т. е. относительная сложность программы ***cl*** вычисляется как отношение абсолютной сложности ***CL*** к общему числу операторов L).

2. Какие характеристики программного средства предусмотрены метрикой Джилба?

• количество операторов цикла ***Lloop***;

• количество операторов условия ***LIF***;

• число модулей или подсистем ***Lmod***;

• отношение числа связей между модулями к числу модулей

**;**

• отношение числа ненормальных выходов из множества операторов к общему числу операторов

3. Как автор метрики определяет надежность программы?

Среди всех показателей качества программ Джилб указывает надежность программы, которую он характеризует как возможность того, что данная программа проработает определенный период времени без логических сбоев. В качестве практической оценки программной надежности автор метрик предлагает рассчитывать как единицу минус отношение числа логических сбоев к общему числу запусков.

4. Что такое прецизионность в метриках Джилба?

Отношение количества правильных данных ко всей совокупности данных приводится Джилбом в качестве меры точности (свободы от ошибок), поскольку он считает, что точность нужна как средство обеспечения надежности программы. Прецизионность определяется как мера того, насколько часто появляются ошибки, вызванные одинаковыми причинами. Джилб оценивает этот показатель дробью, в числителе которой указывается число фактических ошибок на входе, а в знаменателе - общее число наблюдаемых ошибок, причинами которых явились ошибки на входе. Так, например, если одна ошибка вызывает в течение определенного периода времени появление 50 сообщений об ошибках, то прецизионность равна 0,02.

**Задания:**

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

1. разработать программу, реализующую заданный, в соответствии с вариантом, алгоритм (рекомендуется использовать язык программирования С#);

2. оценить характеристики разработанной программы на основе лексического анализа текста и применения метрик Джилба.

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

Вариант 2:

Функция :

Вычисления проводить без использования метода *Math.Pow()* для значений аргументов:

• *x* от -1,1 до 0,3 с шагом 0,2;

• *m* от 1 до 5 с шагом 1.

1. Реализация программы:

|  |  |
| --- | --- |
| **Номера**  **строк** | **Строки программы** |
| 1 | using System; |
| 2 |  |
| 3 | public class Program |
| 4 | { |
| 5 | public static void Main(string[] args) |
| 6 | { |
| 7 | double[] xValues = { -1.1, 0.3 }; |
| 8 | double[] mValues = { 1, 5 }; |
| 9 |  |
| 10 | foreach (double x in xValues) |
| 11 | { |
| 12 | foreach (double m in mValues) |
| 13 | { |
| 14 | (double resultX, double resultSin) = CalculateZ(x, m); |
| 15 | Console.WriteLine($"z({x}, {m}) = {resultX} {resultSin}"); |
| 16 | } |
| 17 | } |
| 18 | } |
| 19 |  |
| 20 | public static (double, double) CalculateZ(double x, double m) |
| 21 | { |
| 22 | double resultX = 1; |
| 23 | double resultSin = 1; |
| 24 |  |
| 25 | for (int i = 0; i < m; i++) |
| 26 | { |
| 27 | resultX = x; |
| 28 | } |
| 29 |  |
| 30 | for (int i = 0; i < m; i++) |
| 31 | { |

*Таблица 1.3 Операторы и операции, используемые в программе*

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Операторы,  операции | Номера строк | Количество  повторений |
| 1 | Using | 1 | 1 |
| 2 | Public | 2, 12 | 2 |
| 3 | Class | 2 | 1 |
| 4 | Static | 3, 12 | 2 |
| 5 | Void | 3 | 1 |
| 6 | Main | 3 | 1 |
| 7 | String[] | 3 | 1 |
| 8 | Args | 3 | 1 |
| 9 | = | 5, 6, 9, 9, 10, 10, 10, 21, 21, 23 | 10 |
| 10 | [] | 5, 5, 6, 6 | 4 |
| 11 | Foreach | 7, 8 | 2 |
| 12 | In | 7, 8 | 2 |
| 13 | () | 9, 10, 10, 10, 10, 21, 21, 23 | 8 |
| 14 | Console.WriteLine | 10 | 1 |
| 15 | Calculatez | 9 | 1 |
| 16 | , | 5, 6, 9, 9, 9, 10, 10, 23 | 8 |
| 17 | Double | 5, 6, 9, 9, 14, 15 | 7 |
| 18 | For | 17, 20 | 7 |
| 19 | < | 17, 20 | 2 |
| 20 | Math.Sin | 21 | 2 |
| 21 | Math.Pi | 21 | 1 |
| 22 | Return | 23 | 1 |
| 23 | Int | 17, 20 | 1 |
| 24 | i | 17, 17, 18, 20 | 4 |
| **Всего** | | | 64 |

*Таблица 1.4. Словарь операндов программы*

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Операнды | Номера строк | Количество  повторений |
| 1 | System | 1 | 1 |
| 2 | Program | 2, 3 | 2 |
| 3 | Main | 3 | 1 |
| 4 | String[] | 3 | 1 |
| 5 | Args | 3 | 1 |
| 6 | xValues | 5, 7, 7, 7 | 4 |
| 7 | -1.1 | 5 | 1 |
| 8 | 0.3 | 5 | 1 |
| 9 | m.Values | 6, 8, 8, 8 | 4 |
| 10 | 1 | 6, 14, 15 | 3 |
| 11 | 5 | 6 | 1 |
| 12 | X | 7, 7, 8, 9, 10, 18, 21 | 7 |
| 13 | M | 8. 8, 9, 10, 17, 20 | 6 |
| 14 | Calculatez | 9 | 1 |
| 15 | resultX | 9, 10, 14, 18, 23 | 5 |
| 16 | ResultSin | 9, 10, 15, 21, 23 | 5 |
| 17 | Console | 10 | 1 |
| 18 | “z({x}…” | 10 | 1 |
| 19 | 0 | 17, 20 | 2 |
| 20 | Math | 21, 21 | 2 |
| 21 | PI | 21 | 1 |
|  |  | **Всего** | 52 |

3. Количество операторов цикла ***Lloop***;

• количество операторов условия ***LIF***;

• число модулей или подсистем ***Lmod***;

Определим значения характеристик ***LIF, Lloop, Lmod, f, L, CL*** и ***cl***.

Значение характеристики ***LIF*** определяется количеством используемых в программе операторов *if*. В представленном решении их один.

***LIF* = 1;**

Значение характеристики ***Lloop*** определяется количеством используемых в программе циклов. В исходном тексте данной программы содержится 2 цикла с использованием оператора *for*.

***Lloop* = 2;**

Значение характеристики ***Lmod*** определяется количеством используемых программных модулей в решении. В представленном решении используется два программных модуля, каждый из которых определяется следующими строками:

• *double F(double x, double y)*;

• *int main()*;

Общее количество операторов условия 3, из них 1 оператор *if* и два оператора цикла *for*. Общее число всех используемых операторов ***L = 64***.   
Таким образом:

• ***CL* = 3** - абсолютная сложность программы;

• ***cl = CL/L*** ***= 3/64 = 0,046875***.

Количество связей между модулями равно одной - между основным и дополнительным модулем. Отношение числа связей к числу модулей определяется следующим образом:

**Вывод:** Из полученных результатов анализа текста программы следует, что исходный код имеет невысокую сложность, так как на 64 оператора текста приходится всего лишь 3 оператора условий. Общее число программных модулей решения также невелико (2 модуля), что подтверждает низкий уровень сложности программы.