МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
"Омский государственный технический университет"

Кафедра «Автоматизированные системы обработки информации и управления»

ПРАКТИЧЕСКАЯ РАБОТА №6

по дисциплине

«Измерительные средства аналитики программных систем и технологий»

по теме

«Метрика Кокола.»

Выполнил: Шмидт А.В.
гр. ИВТ - 244
Проверил: Зубарев А.А.
к.т.н., доцент

Омск 2025

Задание

В рамках практического занятия необходимо выполнить комплексный анализ качества программного кода с использованием гибридной метрики Кокола. Для этого требуется выбрать программу, содержащую ветвления и/или циклы, и провести её оценку по четырём ключевым аспектам: объёмным характеристикам (на основе модели Холстеда), сложности управления (по методу МакКейба), структурированности и общему объёму исходного кода (метрика SLOC). Анализ включает построение схемы алгоритма, расчёт базовых и нормированных значений метрик, определение весовых коэффициентов и вычисление итогового значения гибридной метрики Кокола. Исходный код исследуемой программы, на основе которого выполняется весь расчёт, приведён в Приложении А.

Схема алгоритма показана на рис. 1.

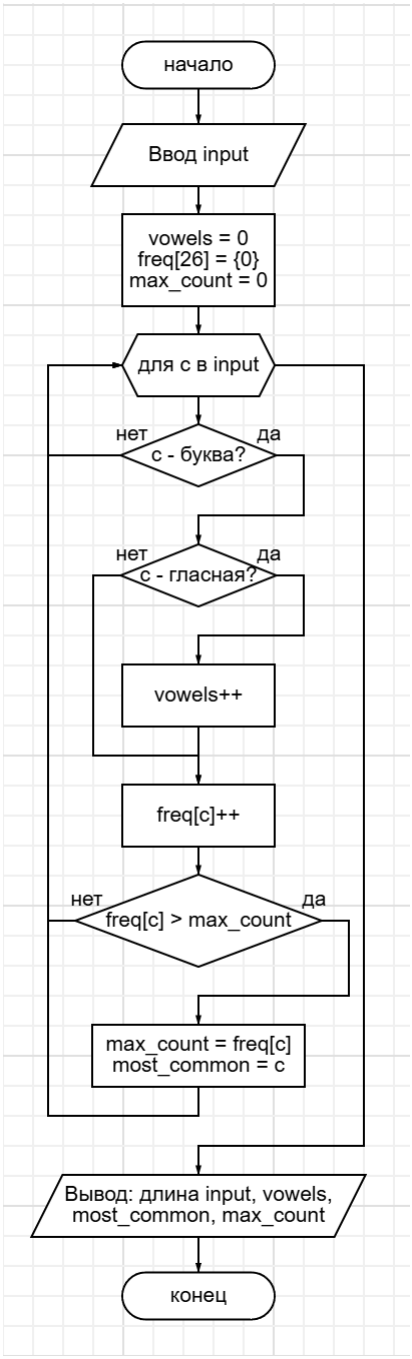

Рисунок 1 – Схема алгоритма программы.

Расчёт метрик по модели Холстеда (Базовая мера):

Для расчёта параметров по модели Холстеда проанализируем исходный код программы.

Таблица 1 – Подсчёт числа типов операторов.

Оператор	Уникальный n1	N2
;	1	26
,	2	1
{}	3	9
()	4	20
=	5	13
++	6	2
-	7	1
[]	8	4
for	9	1
int	10	6
<	11	3
>	12	3
if	13	6
else	14	1
return	15	1
include	16	3
<<	17	18
&&	18	1
!=	19	1
.	20	6
::	21	1
!	22	1
string	23	3

Продолжение таблицы 1 – Подсчёт числа типов операторов.

Оператор	Уникальный n1	N1
char	24	3
cout	25	8
cin	26	1
getline	27	1
	27	144
	n1	N1

Таблица 2 – Подсчёт числа типов операндов.

Операнд	Уникальный n2	N2
input	1	7
length	2	3
vowels	3	3
freq	4	4
most_common	5	3
max_count	6	6
vowel_chars	7	2
c	8	4
lower	9	3
index	10	4
	10	39
	n2	N2

Результаты подсчёта числа операторов и операндов приведены в таблице 3.

Таблица 3 – Результат подсчёта числа операторов и операндов.

Название данных	Частота данных
Число уникальных операторов (n1)	27
Число уникальных операндов (n2)	10
Общее число операторов (N1)	144
Общее число операндов (N2)	39
Словарь программы (n)	37
Количество входных, выходных переменных (n2*)	10

Учитывая данные характеристики, Холстедом были введены следующие оценки:

словарь программы: n = n1 + n2 = 27 + 10 = 37;

длина программы: N = N1 + N2 = 144 + 39 = 183;

объём программы: V = Nlog2(n) = 183 * log2(37) = 954 бит;

теоретическая длина программы: NT = n1log2(n1) + n2log2(n2) = 27*log2(27) + 10*log2(10) = 161,10;

экспериментальная длина программы: Nэ = N1 + N2 = 144 + 39 = 183;

потенциальный объем программы: V* = (n2* + 2)*log2(n2* + 2) = (10 + 2)*log2(10 + 2) = 43,02 бит;

теоретический объём программы: Vt= n1 * log2(n1) + n2 * log2(n2) = 27 * log2(27) + 10 * log2(10) = 161,7 бит;

уровень качества программирования: L = V* / V = 43,02 / 954 = 0,05;

сложность программы: S = 1 / L = 1 / 0,05 = 20;

уровень программы: L^ = (2n2) / (n1N2) = (2 * 10) / (27 * 39) = 0,019;

интеллект программы: I = L^V = 0,019 * 954 = 18,126;

работа по программированию: E = VS = 954 * 20 = 19080 [количество элементарных операций по осмыслению];

время кодирования: T = E / Str = 19080 / 10 = 1908, где Str = 10 — число Страуда.

ожидаемое время кодирования: T = n1N2(n1log2n1 + n2log2n2)log2n / 2n2Str = 27*39(27*log2(27) + 10*log2(10)) / 2*10*18 = 472,68;

уровень языка программирования: λ = LV* = 0,05 * 43,02 = 2,15;

ожидаемый уровень ошибок: B = V / 3000 = 954 / 3000 = 0,318, где 3000 — среднее число элементарных различений между возможными ошибками при программировании;

оптимальная модульность: M = n2* / 6 = 10 / 6 = 1,67;

прогноз числа ошибок: B = Nlog2(n / 3000) = 183*log2(37 / 3000) = -1160,2;

Выбор базовой меры и нормализация:

В качестве базовой меры для метрики выберем объём программы V (954 бит), так как он наиболее полно отражает "объёмную характеристику" кода. Для нормализации используем теоретический объём Vt (161,7 бит) в качестве нормы.

Нормированное значение базовой меры:

M_б_норм = V / V_норм = 954 / 161,7 ≈ 5,9

Граф потока управления и цикломатическая сложность (Мера M1):

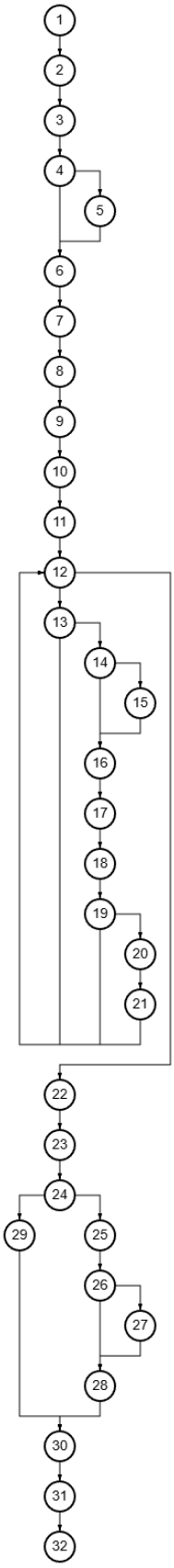

Рисунок 2 – Граф потока управления.

Вершины (V): 32

Дуги (e): 39

Компоненты связности (p): 1

Расчёт цикломатической сложности:

Используем формулу МакКейба,

Z(G) = 39 - 32 + 2 × 1 = 9

Цикломатическая сложность Z(G) = 9

Как указано в методическом пособии, принимаем норму M1 НОРМ = 9. Нормированное значение:

МН(M1) = Z(G) / M1 НОРМ = 9 / 5,9 = 1,5

В программе количество разветвлений (ветвлений if) равно 3. Это попадает в категорию "не более 3". Следовательно, весовой коэффициент R1 = 1.0

Расчёт метрики SLOC (Мера M2)

Подсчитаем физические строки кода (SLOC), исключая пустые строки и комментарии, код в приложении А.

SLOC = 45

Программа простая, выполняет одну задачу. Количество строк 45 находится в диапазоне "Простые" (20–150 строк). M2 НОРМ = 55 строк.

Нормированное значение:

МН(M2) = SLOC / M2 НОРМ = 45 / 55 ≈ 0.82

Весовой коэффициент R2 = 0.00055.

Расчёт гибридной метрики Кокола (НМ):

НМ = [МБ НОРМ + R1 * МН(M1) + R2 * МН(M2)] / (1 + R1 + R2)

НМ = [5,9 + 1.0 * 0.6 + 0.00055 * 0.82] / (1 + 1.0 + 0.00055)

НМ = [5,9 + 0.6 + 0.000451] / 2.00055

НМ = 3,25

Анализ составляющих:

Вклад = (МБ НОРМ) / (1 + R1 + R2) = 5,9 / 2.00055 ≈ 2,95

Доля от НМ = 2,95 / 3,25 ≈ 91%

Оценка: Объём программы (161,7 бит) находится в пределах рекомендуемого диапазона (100–8000 бит), что говорит об адекватном размере модуля. Качество кода по-прежнему в основном определяется его объёмом.

Вклад меры сложности (МакКейб):

Вклад = (R1 * MH(M1)) / (1 + R1 + R2) = (1.0 * 0.6) / 2.00055 ≈ 0.6 / 2.00055 ≈ 0.300

Доля от HM = 0.300 / 3.25 ≈ 9.2%

Оценка: Цикломатическая сложность (Z=9) выше порогового значения (Z=7), что указывает на повышенную сложность потока управления. Код не очень легко читается и тестируется. Влияние сложности на качество заметно, но вторично по отношению к объёму.

Вклад меры объёма (SLOC):

Вклад = (R2 * MH(M2)) / (1 + R1 + R2) = (0.00055 * 0.82) / 2.00055 ≈ 0.00022

Доля от HM = 0.00022 / 3.25 ≈ 0.00007%

Оценка: Влияние количества строк кода на итоговую метрику пренебрежимо мало. Это логично, так как программа имеет небольшой объём, и данный показатель не является определяющим для качества данной программы.

Вывод

В рамках выполнения практической работы был осуществлён всесторонний анализ качества программного кода с применением гибридной метрики Кокола. Для исследуемой программы, включающей конструкции циклов и ветвлений, были определены основные показатели: метрика Холстеда (объём программы около 161,7 бит), цикломатическая сложность по МакКейбу (Z = 9), а также размер исходного кода в строках (SLOC = 45). После процедуры нормализации и использования заданных весовых коэффициентов (R1 = 1,0, R2 = 0,00055) итоговое значение гибридной метрики составило HM ≈ 3,25.

Полученный результат свидетельствует о высоком уровне качества программы: её объём соответствует рекомендуемым значениям, логическая сложность управления невелика, а размер кода не создаёт существенных затруднений при сопровождении. В целом можно сделать вывод, что анализируемая программа удовлетворяет современным требованиям к качественному программному обеспечению, характеризуясь хорошей структурой и низким уровнем сложности.

Приложение А

```cpp
#include <iostream>
#include <string>
#include <cctype>
using namespace std;
int main() {
    string input;
    cout << "Enter a string (up to 100 characters): ";
    getline(cin, input);
    if (!input.empty() && input.back() == '\n') {
        input.pop_back();
    }
    int length = input.length();
    int vowels = 0;
    int freq[26] = {0};
    char most_common = '\0';
    int max_count = 0;
    string vowel_chars = "aeiouyAEIOUY";
    for (char c : input) {
        if (isalpha(c)) {
            if (vowel_chars.find(c) != string::npos) {
                vowels++;
            }
            char lower = tolower(c);
            int index = lower - 'a';
            freq[index]++;
            if (freq[index] > max_count) {
                max_count = freq[index];
                most_common = lower;
            }
        }
    }
    cout << "String length: " << length << '\n';
    cout << "Number of vowels: " << vowels << '\n';
    if (max_count > 0) {
        cout << "Most frequent letter: '" << most_common
             << "' (appears " << max_count << " time";
        if (max_count > 1) cout << "s";
        cout << ")" << '\n';
    } else {
        cout << "Most frequent letter: (no letters)" << '\n';
    }
    cout << "\nPress Enter to exit...";
    cin.get();
    return 0;
}
```