МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Е.В. Павлов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3 |
| «МЕТРИКИ ОЦЕНКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИСЛОЖНОСТИ ПОТОКА ДАННЫХ» |
| по дисциплине: «МЕТРОЛОГИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ» |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4631 |  |  |  | С.А. Гришин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург   
2018

1. **Цель работы**Целью данной работы является изучение методик оценки качества и сложности потока данных программы на основе лексического анализа исходного кода.
2. **Задание**  
    Составить в виде таблицы перечень операторов и операндов, которые используются в программе из ЛР №1. Произвести расчет по метрике Холстеда.

Произвести оценку сложности программного обеспечения на основе метрики Джилба и определить информационную прочность программного модуля по метрике Чепина.

Выполнить расчет среднего числа строк для функций и произвести оценку уровня комментируемости кода.

В качестве варианта задания выбрана программа, написанная на языке программирования С++, хеш-таблицу с двойным хешированием.

**3. Метрика Холстеда**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Оператор | Операнд | Общее количество |
| 1 | new |  | 3 |
| 2 | delete |  | 4 |
| 3 | = |  | 45 |
| 4 | != |  | 2 |
| 5 | == |  | 7 |
| 6 | ( ) |  | 88 |
| 7 | ; |  | 122 |
| 8 | if |  | 9 |
| 9 | "" |  | 17 |
| 10 | , |  | 11 |
| 11 | < |  | 13 |
| 12 | <= |  | 1 |
| 13 | && |  | 6 |
| 14 | return |  | 9 |
| 15 | switch |  | 1 |
| 16 | case |  | 5 |
| 17 | break |  | 6 |
| 18 | ++ |  | 13 |
| 19 | for |  | 14 |
| 20 | endl |  | 8 |
| 21 | << |  | 18 |
| 22 | >> |  | 1 |
| 23 | . (оператор доступа) |  | 15 |
| 24 | cin |  | 6 |
| 25 | cout |  | 10 |
| 26 | close |  | 1 |
| 27 | ignore |  | 1 |
| 28 | clear |  | 1 |
| 29 | rdbuf |  | 1 |
| 30 | getline |  | 3 |
| 31 | class |  | 2 |
| 32 | namespace |  | 2 |
| 33 | void |  | 5 |
| 34 | const |  | 6 |
| 35 | menu |  | 8 |
| 36 | main |  | 1 |
| 37 | srand |  | 1 |
| 38 | time |  | 2 |
| 39 | template |  | 2 |
| 40 | typename |  | 2 |
| 41 | print |  | 3 |
| 42 | push\_back |  | 3 |
| 43 | excel |  | 2 |
| 44 | del |  | 5 |
| 45 | seach |  | 1 |
| 46 | put |  | 3 |
| 47 | h2 |  | 16 |
| 48 | h1 |  | 21 |
| 49 | HashMap |  | 12 |
| 50 | getKey |  | 5 |
| 51 | getValue |  | 2 |
| 52 | setValue |  | 2 |
| 53 | HashNode |  | 10 |
| 54 | isdigit |  | 4 |
| 55 | % |  | 6 |
| 56 | length |  | 5 |
|  |  |  | 572 |
| 1 |  | choice | 4 |
| 2 |  | key | 45 |
| 3 |  | val | 16 |
| 4 |  | i | 16 |
| 5 |  | 'A' | 3 |
| 6 |  | '0' | 6 |
| 7 |  | 0 | 29 |
| 8 |  | 1 | 3 |
| 9 |  | 2 | 2 |
| 10 |  | 3 | 3 |
| 11 |  | 4 | 4 |
| 12 |  | 5 | 4 |
| 13 |  | 6 | 3 |
| 14 |  | 9 | 4 |
| 15 |  | N | 33 |
| 16 |  | K | 18 |
| 17 |  | 2500 | 1 |
| 18 |  | MAP\_SIZE | 7 |
| 19 |  | map | 17 |
| 20 |  | exprt | 6 |
| 21 |  | prime | 10 |
| 22 |  | j | 3 |
| 23 |  | h1 | 19 |
| 24 |  | p\_pow | 3 |
| 25 |  | h2 | 6 |
| 26 |  | collision | 4 |
| 27 |  | fout | 3 |
|  |  |  | 272 |

n1 = 70 — число уникальных операторов программы;

n2 = 25 — число уникальных операндов программы (словарь операндов);

N1 = 1930 — общее число операторов в программе;

N2 = 426 — общее число операндов в программе;

Словарь программы: n = n1+n2 = 69 + 25 = 94

Длина программы: N = N1 + N2 = 1928 + 426 = 235

Объем текста программы: V= N\* log2(n) = 2354 \* log2(94) ≈ 15429.5

**4. Метрики сложности потока данных**

**4.1 Метрика Джилба:**

CL = 126

cl = CL / n = 126 / 1928 = 0,06535 ≈ 0.06

**4.2 Метрика Чепина:**

Q = a1×P + a2×M + a3×C + a4×T = 1 × 6 + 2 × 7 + 3 × 16 + 0.5 × 0 = 68

**5. Количественные оценки.**

**5.1 Оценка среднего числа строк для функций (методов):**

S = Sсум / Sкол = (~907) / 35 = 26

**5.2 Оценка уровня комментируемости (учитывая бесполезные комментарии):**

F = (Nком / Nстр) × 100% = (109 / 992) × 100% = 11%

**6. Выводы**

В результате выполнения данной лабораторной работы были изучены методы оценки качества и сложности потока данных программы. Выполнены расчеты соответствующих оценок на основе лексического анализа исходного кода.

Результатами работы являются следующие оценки:

- объем программы по метрике Холстеда: 15429.5

- сложность потока данных по метрике Джилба: 0,06

- сложность потока данных по метрике Чепина: 68

- среднее число строк для функций (методов): 26

- уровень комментируемости кода: 11%

Оценки, полученные по метрикам Холстеда, Джилба и Чепина несут скорее исследовательский характер, чем практический, и должны рассматриваться в сравнении с аналогичными значениями.

1. Метрики — это и контроль качества кода (не пишем большие и сложные функции), и «производительность» (в кавычках) программистов, и скорость развития проекта, но до сих пор в метриках не придумано главного — что с ними делать. @Habrahabr. Т.е. мы можем с помощью различных метрик оценивать качество ПО, получить даже численные значения, но без сравнения – они бесполезны (ЛР №4).

Так же метрики позволяют оценивать время на разработку и затраты. Позволяет оценивать многократность использования кода (классы, наследование…).

2. Управляющие переменные - это переменные, значения которых могут изменяться в процессе поиска решения до тех пор, пока не будут выполнены ограничения, наложенные на управляющие переменные и целевую функцию, и не будет выполнено условие оптимизации целевой функции. (копипаст, устал)

Самый очевидный пример – for(int i = 0, i<100, i++), где переменная i – управляющая.

3. Паразитные переменные – это переменные которые не используются в коде -\_-

Примеры:

int main() {

int i;

return 0;

}

Переменная i – не используется

struct MyStruct {

static int test()

{

return 1;

}

};

int main() {

MyStruct a;

int temp = a.test();

return 0;

}

<a> не используется (из-за static)

class MyClass

{

public:

void metod() {

int a;

}

};

int main() {

MyClass test;

test.metod();

return 0;

}

Переменная <a> не используется.

4. Последний вопрос, 4 часа ночи… Простым языком – комментарии должны дополнять код, а не дублировать. В большинстве случаев, при хорошо написанном коде – комментарии излишне (могут раскрывать общую суть работу, а не раскладывать по полочкам код). Код должен сам за себя говорить, следовательно, дублирование комментариями кода – это плохие комментарии (например, i++ //увеличиваем счетчик на единицу). Также чрезмерное злоупотребление комментариями несет негативный характер, так как затрудняет процесс чтения кода.

**7. Использованные источники**1. Макконели С. Совершенный код. Мастер-класс / Пер. с англ. – СПб. : БХВ, 2017. – 896 стр. : ил

2. Метрики кода и их практическая реализация [Электронный ресурс]. – Электронные данные – ООО СМ-Консалт (СМК), 2004-2016. – URL: <http://cmcons.com/articles/CC_CQ/dev_metrics/mertics_part_1/>  
  
3. Значения метрик кода [Электронный ресурс]: документация для Visual Studio 2017. – Электронные данные – Microsoft, 2017. – URL: [https://msdn.microsoft.com/ru- ru/library/bb385914.aspx](https://msdn.microsoft.com/ru-ru/library/bb385914.aspx)

**Приложение**

MAIN.CPP

#include "HashMap.h"

#include <iostream>

#include <string>

#include <time.h>

using namespace std;

const int N = 2500;

const int K = 3;

void menu(HashMap<string> &HMap) {

cout << endl

<< "1 - Print\n"

<< "2 - Add element\n"

<< "3 - Delete elem\n"

<< "4 - Export\n"

<< "5 - Exit\n"

<< "Select the menu item: ";

int choice;

cin >> choice;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

switch (choice) {

case 1: {

HMap.print();

menu(HMap);

break;

}

case 2: {

string key;

string val;

cout << "Key: ";

getline(cin, key);

if (key.length() == 6 && isupper(key[0]) && isupper(key[5]) &&

isdigit(key[1]) && isdigit(key[2]) && isdigit(key[3]) &&

isdigit(key[4])) {

cout << "Value: ";

getline(cin, val);

HMap.put(key, val);

}

menu(HMap);

break;

}

case 3: {

string key;

cout << "Key: ";

getline(cin, key);

if (key.length() == 6 && isupper(key[0]) && isupper(key[5]) &&

isdigit(key[1]) && isdigit(key[2]) && isdigit(key[3]) &&

isdigit(key[4])) {

cout << "Elements with collisions: " << HMap.del(key).size() << endl;

}

menu(HMap);

break;

}

case 4: {

HMap.excel(string("excel.txt"));

cout << "OK" << endl;

menu(HMap);

break;

}

case 5: {

break;

}

default:

cout << "Incorrect choice." << endl;

menu(HMap);

}

}

int main() {

srand(time(0));

HashMap<string> HMap(N);

for (int i = 0; i < K; i++) {

string strKey(6, '0');

string strVal(10 + rand() % 70, '\0');

strKey[0] = 'A' + rand() % 26;

strKey[1] = '0' + rand() % 9;

strKey[2] = '0' + rand() % 9;

strKey[3] = '0' + rand() % 9;

strKey[4] = '0' + rand() % 9;

strKey[5] = 'A' + rand() % 26;

for (int i = 0; i < strVal.length(); i++)

strVal[i] = (char)'A' + rand() % 26;

HMap.put(strKey, strVal);

}

menu(HMap);

return 0;

}

HASHNODE.H

#pragma once

#include <string>

template <typename V>

class HashNode {

public:

HashNode(const std::string& key, const V& value) : key(key), value(value) {}

std::string getKey() const { return key; }

V getValue() const { return value; }

void setValue(V val) { HashNode::value = val; }

private:

std::string key;

V value;

};

HASHMAP.H

#pragma once

#include "HashNode.h"

#include <fstream>

#include <iostream>

#include <vector>

using namespace std;

template <typename V> class HashMap {

public:

HashMap(int N) {

MAP\_SIZE = N;

map = new HashNode<V> \*[N];

exprt = new int[N];

for (int i = 0; i < N; i++) {

map[i] = nullptr;

exprt[i] = 0;

}

// Generate prime numbers for h2

// 2, 3, 5, 7, 11, 13, 17, 19...N

primes\_h2.push\_back(2);

for (int i = 3; i < N; i++) {

bool prime = true;

for (int j = 0; j < primes\_h2.size() && primes\_h2[j] \* primes\_h2[j] <= i;

j++) {

if (i % primes\_h2[j] == 0) {

prime = false;

break;

}

}

if (prime)

primes\_h2.push\_back(i);

}

};

~HashMap() {

for (int i = 0; i < N; i++)

delete map[i];

delete map;

delete exprt;

};

unsigned int h1(string &key) {

unsigned int h1 = 0;

const int p = 37;

unsigned int p\_pow = 1;

// h(S) = S[0] + S[1] \* P + S[2] \* P ^ 2 + S[3] \* P ^ 3 + ... + S[N] \* P^N

for (size\_t i = 0; i < key.length(); i++) {

h1 += (key[i] - '0' + 1) \* p\_pow;

p\_pow \*= p;

}

return h1;

}

unsigned int h2(string &key) {

unsigned int h2 = 0;

for (size\_t i = 0; i < key.length(); i++)

h2 += key[i] \* key[i];

// Take a random item

return primes\_h2[h2 % primes\_h2.size()];

}

void put(string key, V value) {

unsigned int h1 = HashMap::h1(key);

unsigned int h2 = HashMap::h2(key);

for (size\_t i = 0; i < MAP\_SIZE; i++) {

exprt[h1 % N] += 1;

if (map[h1 % N] == nullptr) {

map[h1 % N] = new HashNode<V>(key, value);

return;

}

else {

if (map[h1 % N]->getKey() == key)

map[h1 % N]->setValue(value);

h1 = (h1 + h2) % N;

}

}

cout << "OVERFLOW" << endl;

}

HashNode<V> \*seach(string &key) {

unsigned int h1 = HashMap::h1(key);

unsigned int h2 = HashMap::h2(key);

for (size\_t i = 0; i < MAP\_SIZE; i++) {

if (map[h1 % N] != nullptr && map[h1 % N]->getKey() == key)

return map[h1 % N];

else

h1 = (h1 + h2) % N;

}

return nullptr;

}

vector<HashNode<V> \*> del(string &key) {

unsigned int h1 = HashMap::h1(key);

unsigned int h2 = HashMap::h2(key);

vector<HashNode<V> \*> collision;

for (size\_t i = 0; i < MAP\_SIZE; i++) {

if (map[h1 % N]->getKey() == key) {

delete map[h1 % N];

map[h1 % N] = nullptr;

return collision;

}

else {

collision.push\_back(map[h1 % N]);

h1 = (h1 + h2) % N;

}

}

}

void excel(string &name) {

ofstream fout(name);

for (size\_t i = 0; i < MAP\_SIZE; i++)

fout << exprt[i] << endl;

fout.close();

}

void print() {

cout << endl;

for (size\_t i = 0; i < MAP\_SIZE; i++)

if (map[i] != nullptr)

cout << i << ": " << map[i]->getKey() << " - " << map[i]->getValue()

<< endl;

}

private:

HashNode<V> \*\*map;

int \*exprt;

int MAP\_SIZE;

vector<int> primes\_h2;

};