МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ 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 №4

«ОЦЕНКА АЛГОРИТМИЧЕСКОЙ СЛОЖНОСТИ

ПРОГРАММНОГО КОДА»

ПО КУРСУ: «МЕТРОЛОГИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»

РАБОТУ ВЫПОЛНИЛ:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4831 |  |  |  | 13.04.2020 |  | К.А. Корнющенков |
|  |  |  | (подпись) |  | (дата отчёта) |  | (инициалы, фамилия) |

Санкт-Петербург 2020

1. Цель работы

Целью данной работы является анализ графа потока управления и оценка алгоритмической сложности программного кода на основе метрики Маккейба.

1. Задание на лабораторную работу

Начертить блок-схему алгоритма программного кода приложения (или его фрагмента), построить граф потока управления, выделить линейно-независимые маршруты и циклы и выполнить расчёт цикломатического числа Маккейба.

Отразить в выводах результаты и проанализировать корректность расчета цикломатической сложности.

Разрешается ограничить исходный код программы (из ЛР 1) и использовать для построения блок-схемы алгоритма и графа потока управления фрагмент кода (или модуль) размером 80-120 строк (из которых не более 15% пустых). Данный фрагмент кода (или модуль) обязательно должен содержать операторы ветвления и/или циклы.

|  |
| --- |
| Вариант задания:  89. Приложение для обмена фотографиями |

1. Оценка алгоритмической сложности программного кода

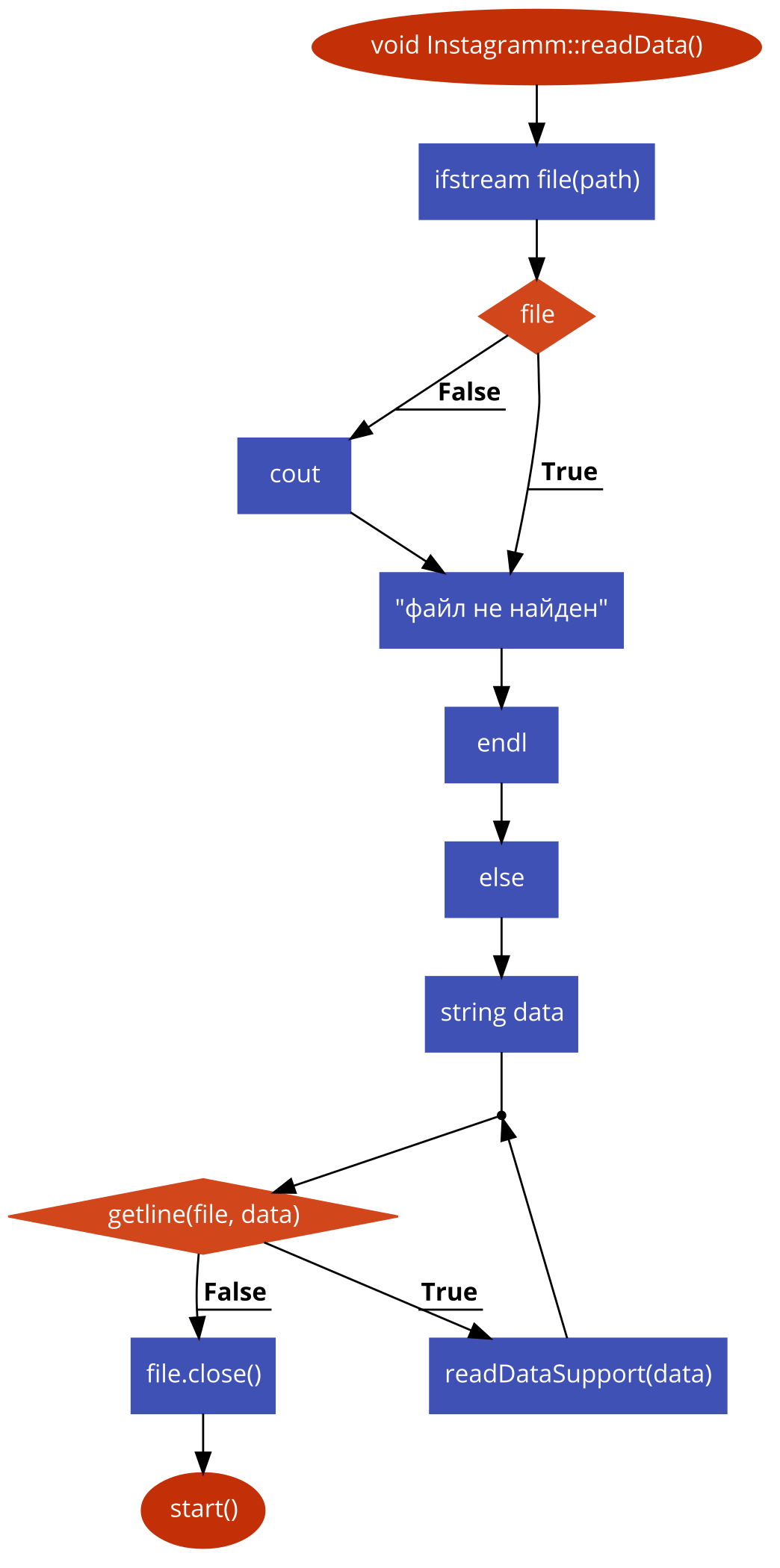


Рисунок — Блок-схема алгоритма анализируемого фрагмента кода функции readData()

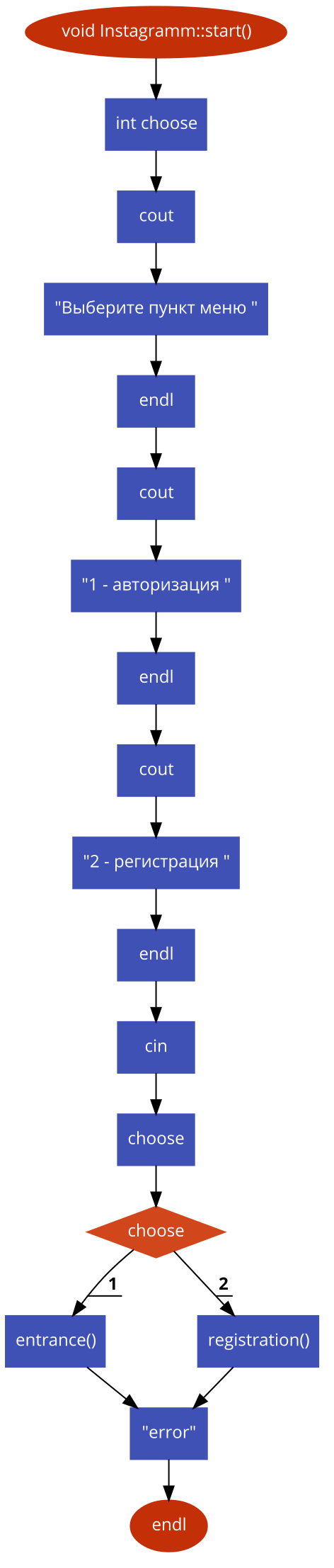


Рисунок — Блок-схема алгоритма анализируемого фрагмента кода функции start()

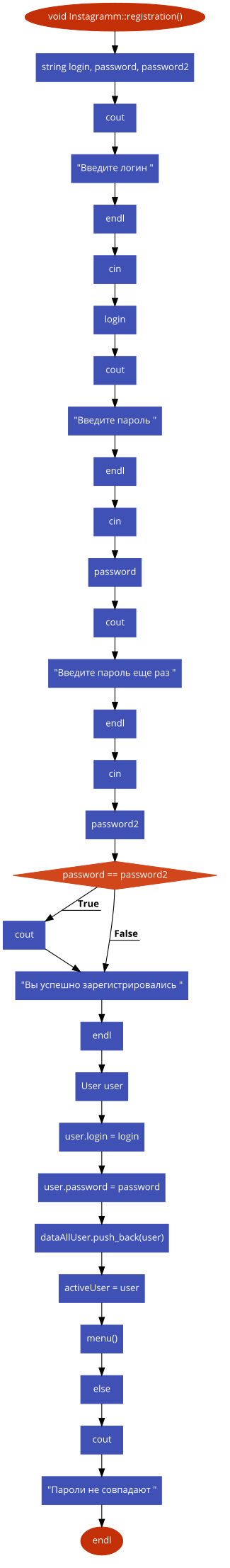


Рисунок — Блок-схема алгоритма анализируемого фрагмента кода функции registration()

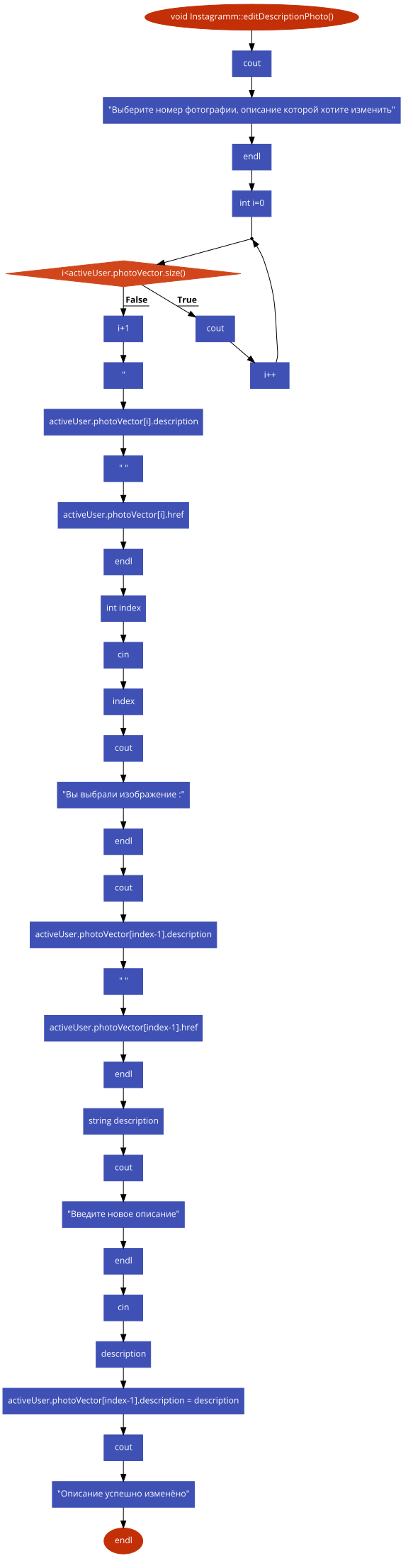


Рисунок — Блок-схема алгоритма анализируемого фрагмента кода функции editDescriptionPhoto ()

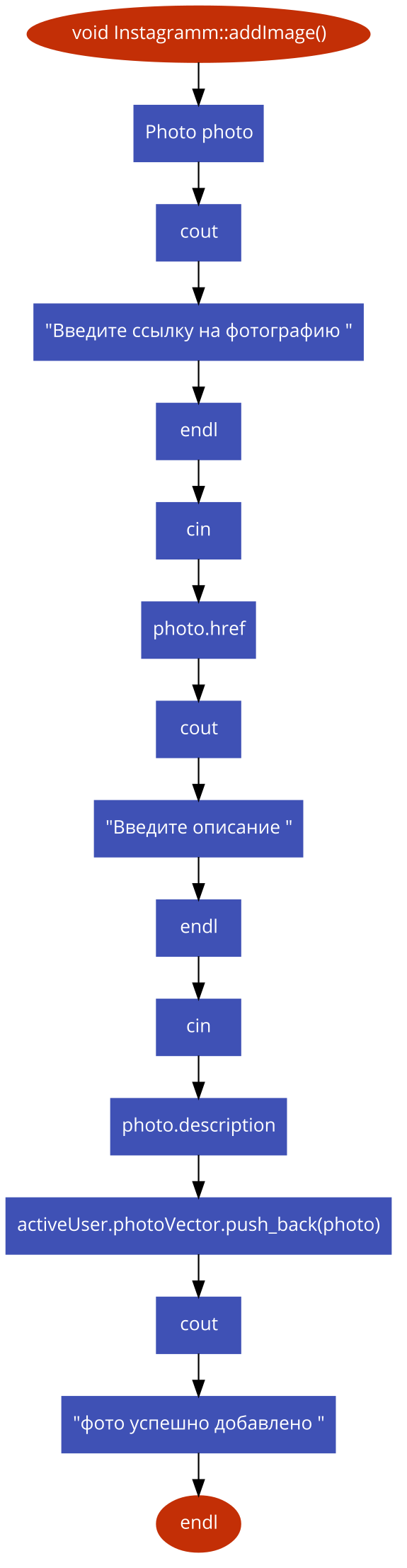


Рисунок — Блок-схема алгоритма анализируемого фрагмента кода функции addImage ()

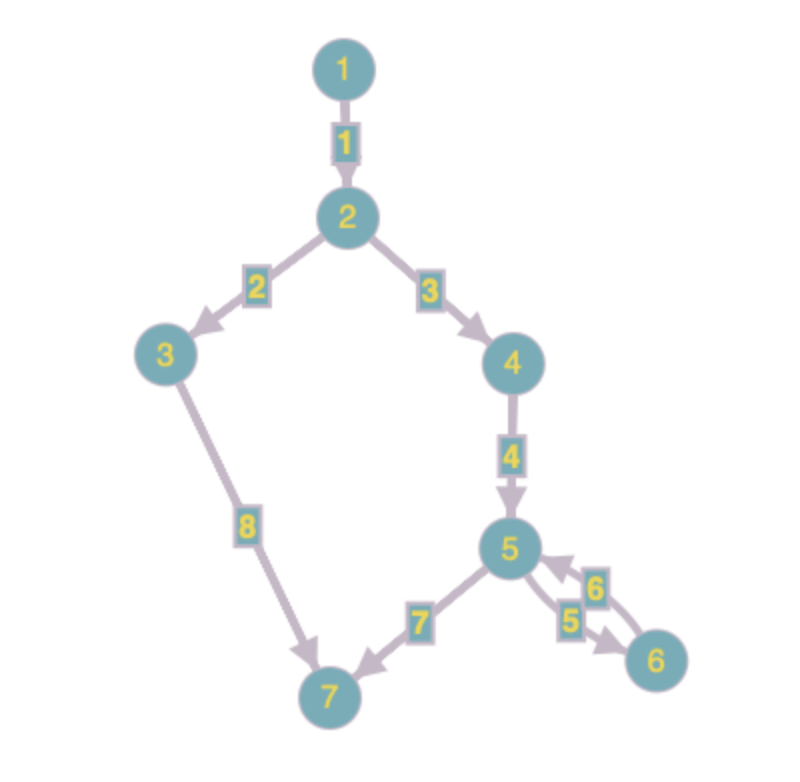


Рисунок — Граф потока управления функции readData()

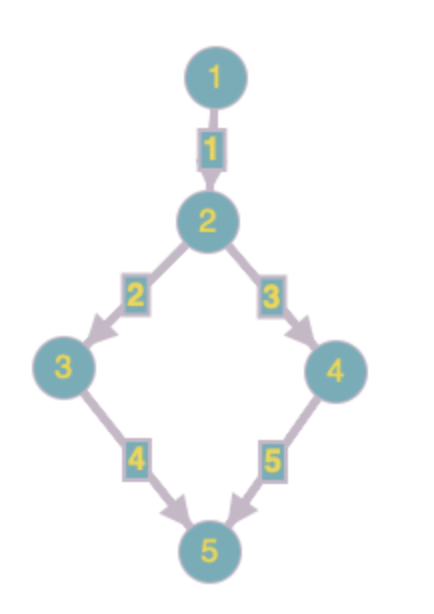


Рисунок — Граф потока управления функции start()

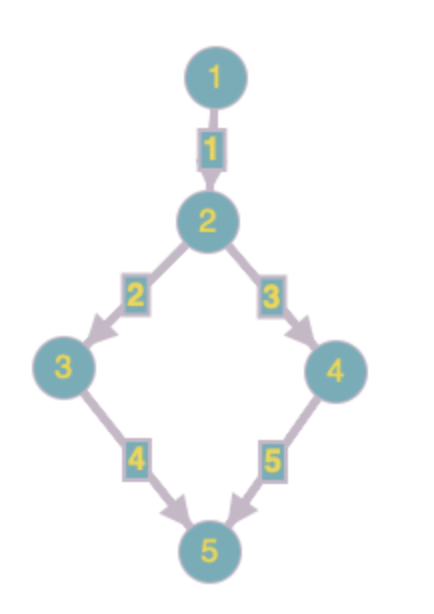


Рисунок — Граф потока управления функции registration()

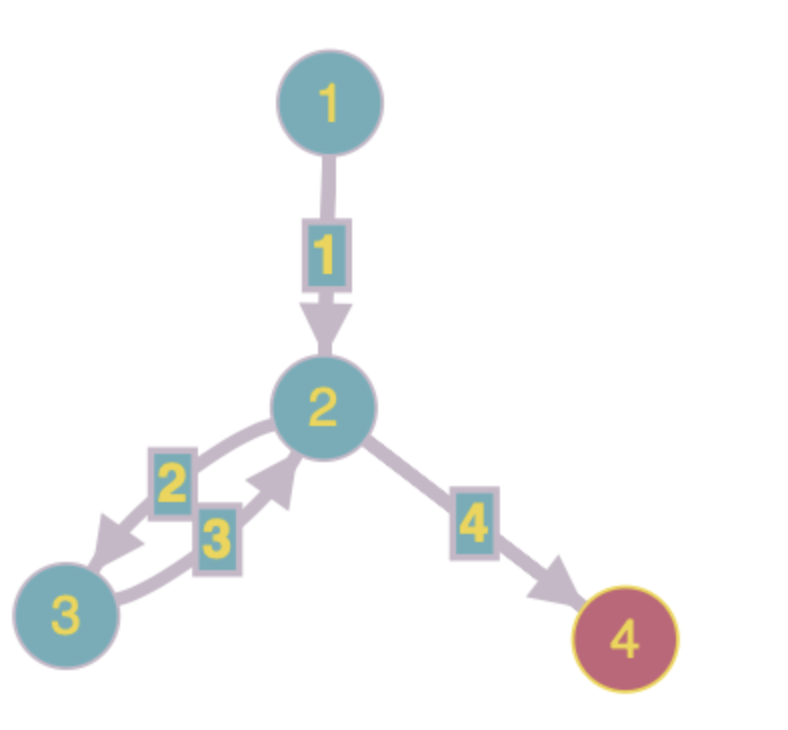


Рисунок — Граф потока управления функции editDescriptionPhoto ()



Рисунок — Граф потока управления функции addImage ()

Таблица 1 — Расчет метрики Маккейба для функции readData:

|  |  |  |
| --- | --- | --- |
| Количество рёбер (дуг) | *m* | 8 |
| Количество узлов (вершин) | *n* | 7 |
| Цикломатическое число Маккейба | *M = m – n + 2* | 3 |

Для исчерпывающего тестирования программного кода (см. приложение А) потребуется 3 тестовых проходов, чтобы покрыть все пути исполнения.

Линейно-независимые маршруты и циклы для данного графа:

1) 1 – 2 – 3 – 7

2) 1 – 2 – 4 – 5 – 7

3) 5 – 6

Таким образом, количество линейно-независимых маршрутов и циклов равно цикломатическому числу, что свидетельствует о корректно выполненном расчёте.

Таблица 2 — Расчет метрики Маккейба для функции start:

|  |  |  |
| --- | --- | --- |
| Количество рёбер (дуг) | *m* | 5 |
| Количество узлов (вершин) | *n* | 5 |
| Цикломатическое число Маккейба | *M = m – n + 2* | 2 |

Для исчерпывающего тестирования программного кода (см. приложение А) потребуется 2 тестовых проходов, чтобы покрыть все пути исполнения.

Линейно-независимые маршруты и циклы для данного графа:

1) 1 – 2 – 3 – 5

2) 1 – 2 – 4 – 5

Таким образом, количество линейно-независимых маршрутов и циклов равно цикломатическому числу, что свидетельствует о корректно выполненном расчёте.

Таблица 3 — Расчет метрики Маккейба для функции registration:

|  |  |  |
| --- | --- | --- |
| Количество рёбер (дуг) | *m* | 5 |
| Количество узлов (вершин) | *n* | 5 |
| Цикломатическое число Маккейба | *M = m – n + 2* | 2 |

Для исчерпывающего тестирования программного кода (см. приложение А) потребуется 2 тестовых проходов, чтобы покрыть все пути исполнения.

Линейно-независимые маршруты и циклы для данного графа:

1) 1 – 2 – 3 – 5

2) 1 – 2 – 4 – 5

Таблица 4 — Расчет метрики Маккейба для функции editDescriptionPhoto:

|  |  |  |
| --- | --- | --- |
| Количество рёбер (дуг) | *m* | 4 |
| Количество узлов (вершин) | *n* | 4 |
| Цикломатическое число Маккейба | *M = m – n + 2* | 2 |

Для исчерпывающего тестирования программного кода (см. приложение А) потребуется 2 тестовых проходов, чтобы покрыть все пути исполнения.

Линейно-независимые маршруты и циклы для данного графа:

1) 1 – 2 – 4

2) 2 – 3

Таким образом, количество линейно-независимых маршрутов и циклов равно цикломатическому числу, что свидетельствует о корректно выполненном расчёте.

Таблица 5 — Расчет метрики Маккейба для функции:

|  |  |  |
| --- | --- | --- |
| Количество рёбер (дуг) | *m* | 0 |
| Количество узлов (вершин) | *n* | 1 |
| Цикломатическое число Маккейба | *M = m – n + 2* | 1 |

Для исчерпывающего тестирования программного кода (см. приложение А) потребуется 2 тестовых проходов, чтобы покрыть все пути исполнения.

Линейно-независимые маршруты и циклы для данного графа:

1) 1

Таким образом, количество линейно-независимых маршрутов и циклов равно цикломатическому числу, что свидетельствует о корректно выполненном расчёте.

Выводы по работе

В результате выполнения данной работы представлена блок-схема алгоритма рассматриваемого программного кода, по которой был сформирован граф потока управления и выделены линейно-независимые маршруты и циклы. На основе графа потока управления выполнен расчет цикломатического числа Маккейба. При этом количество выделенных линейно-независимых маршрутов и циклов совпадает с цикломатическим числом, соответственно оценка цикломатической сложности программного кода выполнена корректно.

По отношению к рассматриваемому программному коду можно говорить о низкой алгоритмической сложности и соответственно высоких показателях свойств анализируемости и тестируемости. Высокая оценка данных характеристик основана на простой функциональности анализируемого кода и не связана с особенностями реализации.

Таким образом, можно заключить, что выполненная работа соответствует поставленной задаче и отвечает всем сформулированным в методических указаниях требованиям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Павлов Е. В. Методические рекомендации к выполнению лабораторных работ: Метрология программного обеспечения / Евгений Васильевич Павлов. —   
   СПб ГУАП, 2020
2. Черников, Б. В. Управление качеством программного обеспечения: учебник /   
   Б. В. Черников. — М.: ИД «ФОРУМ»: ИНФРА-М, 2012. — 240 с.: ил.
3. Широков, А. И. Стандартизация, сертификация и оценка качества программного обеспечения: учебное пособие / А. И. Широков, Е. П. Потоцкий. —   
   М.: ИД «МИСиС», 2013. — 208 с.
4. Graph Description Language [Электронный ресурс]: Documentation / Emden R. Gansner, Eleftherios Koutsofios, Stephen North. — 2020. — URL: <https://graphviz.gitlab.io/_pages/pdf/dotguide.pdf> (дата обращения: 23.03.2020).

ПРИЛОЖЕНИЕ А

Анализируемый программный код

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123 | void Instagramm::readData(){  ifstream file(path);  if (!file) {  cout << "файл не найден" << endl;  } else {  string data;  while (getline(file, data)) {  readDataSupport(data);  }  file.close();  }  start();  }  //MARK: Вход в приложение  void Instagramm::start(){  int choose;  cout << "Выберите пункт меню " << endl;  cout << "1 - авторизация " << endl;  cout << "2 - регистрация " << endl;  cin >> choose;  switch (choose) {  case 1:  entrance();  break;  case 2:  registration();  break;  default:  cout << "error" << endl;  break;  }  }  void Instagramm::registration(){  string login, password, password2;  cout << "Введите логин " << endl;  cin >> login;  cout << "Введите пароль " << endl;  cin >> password;  cout << "Введите пароль еще раз " << endl;  cin >> password2;  if (password == password2){  cout << "Вы успешно зарегистрировались " << endl;  User user;  user.login = login;  user.password = password;  dataAllUser.push\_back(user);  activeUser = user;  menu();  }else{  cout << "Пароли не совпадают " << endl;  }  }  void Instagramm::editDescriptionPhoto(){  cout << "Выберите номер фотографии, описание которой хотите изменить" << endl;  for (int i=0;i<activeUser.photoVector.size();i++){  cout << i+1 << ":" << activeUser.photoVector[i].description << " " << activeUser.photoVector[i].href << endl;  }  int index;  cin >> index;  cout << "Вы выбрали изображение :" << endl;  cout << activeUser.photoVector[index-1].description << " " << activeUser.photoVector[index-1].href << endl;  string description;  cout << "Введите новое описание" << endl;  cin >> description;  activeUser.photoVector[index-1].description = description;  cout << "Описание успешно изменёно" << endl;  }  void Instagramm::addImage(){  Photo photo;  cout << "Введите ссылку на фотографию " << endl;  cin >> photo.href;  cout << "Введите описание " << endl;  cin >> photo.description;  activeUser.photoVector.push\_back(photo);  cout << "фото успешно добавлено " << endl;  } |