

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра Кафедра информатики и программирования

**РАЗРАБОТКА КОМПЛЕКСА АВТОМАТИЗИРОВАННОГО
ТЕСТИРОВАНИЯ РАЗРАБОТАННОГО ТРАНСЛЯТОРА ИЗ ЯЗЫКА
ПРОГРАММИРОВАНИЯ КУМИР В ЯЗЫК C++**

КУРСОВАЯ РАБОТА

студента 1 курса 173 группы

направления 02.03.04 — Математическое обеспечение и администрирование
информационных систем

КНиИТ

Пронина Антона Алексеевича

Научный руководитель

старший преподаватель

Е. Е. Лапшева

Заведующий кафедрой

к. ф.-м. н.

М. В. Огнева

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Методы, средства и технологии	5
1.1 Методы и средства автоматизированного тестирования	5
1.2 Транслятор с языка КуМир в язык C++	9
2 Реализация системы тестирования	15
2.1 Процесс тестирования	15
2.2 Тесты	15
2.3 Программа-тестер	17
ЗАКЛЮЧЕНИЕ	20
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	22
Приложение А Исходный код dockerfile, унифицирующий настройки сре- ды запуска системы тестирования	23
Приложение Б Исходный код скрипта, производящего тестирование транс- лятора	25

ВВЕДЕНИЕ

В современном школьном образовании среди прочих можно выделить следующие тенденции: растущий интерес к изучению робототехники; глубокая интеграция языка программирования КуМир в школьную информатику и государственную итоговую аттестацию по этому предмету.

В связи с этим будет востребована система программирования роботов с использованием этого языка программирования. Ключевой составной частью этой системы является транслятор программы с языка КуМир [1] в язык C++. Тронуется протестировать разработанный транслятор, провести поиск ошибок и устранить их.

Создание средств и систем обеспечения надежности ПО такого класса как компиляторы, представляет собой весьма сложную задачу, особенно, если речь идет о тестировании компиляторов. Тестирование компиляторов - это необходимый этап в разработке языков программирования и компиляторов, поскольку компиляторы являются ключевыми инструментами для трансформации исходного кода в машинный код, который может выполняться на целевой платформе. Наличие надежного и качественного компилятора обеспечивает уверенность в правильности работы программ и уменьшает риски возникновения ошибок во время выполнения программы. Тестирование компиляторов необходимо для гарантии совместимости программного кода на различных платформах и операционных системах. Кроме того, тестирование компиляторов помогает в оценке работоспособности и эффективности программного обеспечения на целевой платформе. В целом, тестирование компиляторов является важным этапом в процессе разработке программного обеспечения, который может помочь в достижении более высокого качества и уверенности в работе программного кода.

Актуальность транслятора заключается в снижении входного порога в

область робототехники как со стороны ученика, предоставляя возможность, используя полученные навыки программирования в системе КуМир, заниматься разработкой роботов, так и со стороны преподавателя, уменьшая затраты на приобретение программных и аппаратных средств разработки. В качестве аппаратной платформы данной задачи был выбран электронный конструктор и удобная платформа быстрой разработки электронных устройств для новичков и профессионалов [2] - платы Arduino. Ввиду низкой стоимости устройств, периферийных модулей, простоты разработки аппаратных устройств на базе этих плат, высокой модульности систем и их высокой распространенности среди робототехников.

Цель работы - разработка комплекса автоматизированного тестирования из языка программирования КуМир в язык C++.

Для выполнения поставленной цели, требуется выполнить следующие задачи:

- анализ подходов, инструментов и ПО, использующихся при автоматизации тестирования;
- разработка набора тестов и документации для транслятора;
- разработка системы автоматизированного тестирования.

1 Методы, средства и технологии

1.1 Методы и средства автоматизированного тестирования

Автоматизированное тестирование - это процесс тестирования, в котором используются специальные программные средства и скрипты, которые могут автоматически создавать и проводить тесты на соответствие функциональным требованиям. Автоматизированный тест позволяет быстро и эффективно производить тестирование без физического участия тестировщика. Это позволяет повысить качество продукта, уменьшить количество ошибок и рисков при релизе, а также быстро и точно проводить регрессионное тестирование после внесения изменений в проект. Автоматизированное тестирование может уменьшить время, необходимое для тестирования, а также заставить взглянуть на проблемы, с которыми можно столкнуться во время процесса разработки. В качестве основных видов автоматизированного тестирования можно выделить следующие:

- Unit-тестирование: тестирование отдельных компонентов программного обеспечения, таких как функции, классы и модули;
- интеграционное тестирование: тестирование взаимодействия между отдельными компонентами;
- функциональное тестирование: тестирование функциональности программного обеспечения;
- конфигурационное тестирование: тестирование работоспособности системы при изменении настроек и конфигураций;
- нагрузочное тестирование: тестирование производительности системы при различных нагрузках;
- тестирование безопасности: тестирование на возможность атак и нарушений безопасности системы;

- тестирование совместимости: тестирование возможности программного обеспечения работать с различными операционными системами, браузерами и устройствами;
- тестирование пользовательского интерфейса: тестирование удобства и функциональности интерфейса программного обеспечения;
- автоматизированное тестирование: использование специальных средств для создания скриптов тестирования и автоматизации процесса тестирования.

В зависимости от платформы, применяются различные инструменты и методы тестирования. в качестве тестируемого объекта выступает транслятор из языка программирования КуМир в язык C++.

При создании системы тестирования, требуется максимально уменьшить количество внешних эффектов, создаваемых окружением. Для унификации среды для запуска системы тестирования используется технология контейнеризации и Docker.

Оконтейнеризация (или контейнеризация) - это технология виртуализации, которая позволяет запускать и управлять различными приложениями в изолированной среде - контейнере. Контейнеры являются автономными, портативными и легковесными единицами, которые могут быть запущены в любой совместимой с контейнеризацией среде.

Контейнеры позволяют упаковывать приложения и их зависимости в единую единицу, которая может быть развернута и запущена в любом месте, без необходимости инсталляции конкретных зависимостей на хост-системе. Контейнеры дают возможность разработчикам и DevOps-инженерам значительно ускорить процесс разработки, тестирования и распространения приложений.

Контейнеры используют ядро операционной системы хоста и создают определенное окружение, в котором приложение запускается и работает. Это

облегчает многие процессы, такие как масштабирование, управление зависимостями и обновлениями, управление средой и конфигурацией приложения.

Docker - это свободное программное обеспечение, которое позволяет упаковывать, доставлять и запускать приложения в контейнерах. Это популярная технология контейнеризации, которая позволяет разработчикам быстро и легко создавать, тестировать и запускать приложения в различных средах. Docker также предоставляет инструменты для управления контейнерами, включая CLI и API. Это позволяет управлять контейнерами и приложениями из командной строки или с помощью API, что делает автоматизацию и масштабирование процесса разработки и развертывания более эффективным. Docker - это инновационная технология, которая упрощает процессы разработки, тестирования и запуска приложений и может использоваться в любом окружении. Она становится все более популярной в мире разработки приложений благодаря своей надежности, безопасности и высокой скорости запуска.

С точки зрения тестирования, транслятор - просто программа, принимающая информацию на вход и в результате работы имеющая определенный выход. Можно выделить 4 основных способа тестирования компиляторов:

1. наборы небольших статических тестов (regression, unit и т.п.). Это наборы тестов программного обеспечения, которые написаны до запуска приложения и проверяют его на соответствие требованиям и описанию функций. Эти тесты используются для проверки корректности работы приложения до его использования реальными пользователями.;
2. приложения-тестеры. Это приложение позволяет создавать тестовые сценарии, выполнять автоматические тесты и генерировать отчеты о результатах тестирования. Обычно приложения-тестеры используются во время автоматического тестирования, когда тесты выполняются без участия человека. Это может быть полезно для повторяемых тестов, которые нужно

проводить на протяжении всего периода разработки приложения;

3. coverage-тестирование. Это метод тестирования программного обеспечения, который позволяет оценить, насколько хорошо тесты покрывают код программы. Он используется для определения того, сколько кода было протестировано путем анализа выполнения тестов и сравнения с фактическим кодом программы. Результаты coverage-тестирования определяют, какие участки кода были покрыты тестами и какие нет. Если некоторые участки кода не были покрыты тестами, то разработчики смогут сосредоточиться на их тестировании и повышении надежности программного обеспечения.;
4. fuzzing-тестирование. Это метод тестирования программного обеспечения, который заключается в автоматической генерации большого количества случайных или полу-случайных входных данных, и последующем анализе ответов на эти входные данные. Целью такого тестирования является обнаружение ошибок и уязвимостей в программном обеспечении, которые могут привести к потенциальным угрозам безопасности или нарушению работоспособности приложения [3].

Среди готовых решений для тестирования ПО можно выделить следующие:

- IBM Rational Functional Tester[4] предоставляет функции автоматического тестирования для функционального, регрессионного тестирования, тестирования графических пользовательских интерфейсов и тестирования, ориентированного на данные. Данная программа не распространяется бесплатно, для использования необходимо приобрести лицензию;
- IBM Rational Test Workbench[5] так же распространяется по платной лицензией, с пробным периодом бесплатного использования в тридцать дней;
- система автоматизированного тестирования TestComplete[6],

предоставляющая в том числе инструмент для регрессионного тестирования.

На этапе анализа существующего ПО для тестирования, было выяснено, что ни 1 из рассмотренных вариантов не сможет быть применен в виду чрезмерно объемного интерфейса, а также отсутствия бесплатной подписки.

Транслятор из языка программирования КуМир в язык программирования C++ - разработка с открытыми исходными кодами. На данный момент, язык программирования КуМир поддерживает функциональный стиль программирования без создания пользовательских структур или классов. Ввиду наличия исходных кодов транслятора и ограниченности набора инструкций языка КуМир, было решено разработать автоматизированную тестовую среду с набором статических тестов.

1.2 Транслятор с языка КуМир в язык C++

Приложение разрабатывается на языке C++ с использованием фреймворка QT 4 или 5 версии, а также системы автоматизации сборки программного кода CMake. Для сборки проекта на операционной системе Windows, требуется программа-сборщик MS Build 2012, устаревший на данный момент и труднодоступный для скачивания, поэтому было предпринято решение разрабатывать приложение на операционной системе Linux Mint. Для сборки проекта требуется установить ряд дополнительных библиотек, среди которых: ZLib, Boost и интерпретатор языка программирования Python.

Проект состоит из ряда библиотек, используемых внутри проекта, в том числе - AST-дерево, библиотека для работы с низкоуровневой виртуальной машиной, библиотеки для рендеринга pdf-документов и т.д.; плагинов, а также исполнителей и сред разработки, состоящих из файлов с описанием используемых методов, процедур и сущностей и файлов с описанием слоя представ-

ления данных частей. Ввиду большого объема приложения, высокого уровня связности элементов друг с другом и устаревших библиотек, используемых для разработки, сборка проекта осуществляется при помощи командной строки.

В плагине `kumirCodeGenerator` осуществляется трансляция и компиляция программного кода с языка КуМир в язык низкоуровневой виртуальной машины. По имеющемуся файлу с исходным кодом происходят лексический и синтаксический анализы, в случае, если они завершились успешно и не было выявлено ошибок, трансляция продолжается. В результате лексического анализа имеется набор лексем кода исходного языка программирования, использующийся для синтаксического анализа. В результате синтаксического анализа программа создает набор токенов для генерации по ним дерева разбора. Вычисленное дерево разбора усекается до AST-дерева, для трансляции кода. На этом этапе происходит наращивание кода на выходном языке по имеющемуся дереву разбора. Описанные выше этапы осуществляются при помощи вызова функций сторонних модулей.

Для разработки транслятора на основе измененного и переработанного программного кода плагина `kumirCodeGenerator` был создан модуль `arduinoCodeGenerator`. Основные изменения затрагивают сущность `Generator`, содержащую основной объем операций по обработке данных AST-дерева и наращиванию программного кода по нему на выходном языке.

Поскольку в изначальном варианте программный код транслировался в язык виртуальной низкоуровневой машины, был кардинально переработан список команд. Список команд для трансляции с языка программирования КуМир в язык программирования C++ приведен в таблице 2.

Таблица 1 – Список команд, используемый для трансляции с языка программирования КуМир в язык программирования C++

Название операции	Код операции
ForLoop	0
WhileLoop	1
VAR	2
ARR	3
FUNC	4
CONST	5
IF	6
ELSE	7
SWITCH	8
CASE	9
INPUT	10
OUTPUT	11
BREAK	12
END_ARG_DELIM	13
END_FUNC_DELIM	14
END_ARR_DELIM	15
STR_DELIM	16
END_ST_DELIM	17
END_VAR_DELIM	18
END_ST_HEAD_DELIM	19
RET	20
SUM	21
SUB	22
MUL	23
DIV	24
POW	25
NEG	26
AND	27
OR	28
EQ	29
NEQ	30
LS	31
GT	32
LEQ	33
GEQ	34
ASG	35
DCR	36
INC	37

Были изменены инструкции для трансляции циклов, условий, вызова и объявления функций. Были добавлены инструкции для объявления переменных и констант. Блок операций подвергся минимальным изменениям — были добавлены инструкции инкремента и декремента и присваивания.

Также изменилась основная сущность, используемая при трансляции — сущность инструкции выходного языка. Были удалены поля для хранения данных о регистре, спецификации строки, спецификации модуля и были добавлены поля для хранения имени операнда и типы операнда, если присутствует.

Трансляция циклов Существует 4 типа циклов в языке программирования КуМир: стандартный для языков программирования высокого уровня цикл, выполняющийся некоторое количество раз, определяемое набором элементов в указанном диапазоне “нц для“, цикл с предусловием “нц пока“, цикл, выполняющийся n раз “нц для n раз“, а также цикл, выполняющийся постоянно до остановки исполнения “нц всегда“. При трансляции в низкоуровневый язык, этап трансляции циклов требовал одной инструкции — “LOOP“, а также тела операций, выполняющихся до данной метки. В начале транслировалось тело цикла, в случае цикла с предусловием до тела транслировалось условие исполнения, на последнем этапе трансляции добавлялась инструкция “LOOP“, объявляющую метку завершения определения цикла.

Для трансляции циклов в язык C++ были добавлены инструкции для соответствующих типов циклов — ForLoop, WhileLoop. Циклы “нц пока“ и “нц всегда“ определяются при помощи инструкции WhileLoop, “нц для“ и “нц для n раз“ - при помощи ForLoop.

В начале трансляции цикла любого типа указывается заголовок, определяющий тип цикла и возможные предусловия. В случае цикла “нц пока“, заголовок содержит условие окончания работы, состоящее из ряда выражений.

После трансляции заголовка инструкции цикла транслируется тело цикла. Трансляция завершается инструкцией END_ST_DELIM.

Трансляция подвыражений Трансляция подвыражений осуществляется при помощи метода `calculate`. Любое подвыражение представляется бинарным деревом, в вершине которого находится операнд, а на листьях - константы или переменные. Для корректной трансляции подвыражений требуется разобрать дерево подвыражения снизу вверх — найти узел, листья которого не содержат дальнейшей вложенности, затем добавить в стек инструкций левый лист узла, затем сам узел, в конце — правый лист. Метод `calculate` используется во множестве мест и содержит логику для трансляции переменных, констант, вызовов функций и подвыражений. Поскольку метод является рекурсивным, невозможно добавить дополнительный блок операций трансляций, описанный выше, в тело метода `calculate`, поэтому было принято решение вынести логику данного метода в новый - `innerCalculation`, превратив метод исходный в обертку, куда и была добавлена вспомогательная логика.

Трансляция условных выражений Изначально, трансляция условных конструкций состояла из определения количества подвыражений транслируемого выражения и поиска ошибок. На каждую конструкцию “если то” и “иначе” добавлялась инструкция безусловного перехода, изменяя ход исполнения программы. Для установки места программы низкоуровневого языка, куда совершался переход используется регистр `IP`.

В разработанной реализации трансляция условных выражений повторяет трансляцию цикла с предусловием за исключением первой инструкции — вначале, в стек инструкций добавляется инструкция “IF” или “ELSE”, указывающая транслятору на объявление соответствующего блока. Процесс повторяется пока сущность, содержащая данные об условном выражении не опустеет.

Трансляция блоков ‘выбор’ происходит следующим образом: вначале в стек заносится инструкция с кодом “SWITCH”, далее добавляется переменная,

значения которой перебираются и на каждый блок ‘при условии’ добавляется инструкция “CASE” и константа, хранящая значение переменной. В случае наличия метки “иначе” в блоке “выбор”, добавляется инструкция “CASE” без константы, транслирующаяся в инструкцию “default” в коде выходного языка.

Трансляция переменных и констант В исходной версии при трансляции констант в выражениях или при инициализации, в стек виртуальной машины записывался индекс переменной или константы среди всех встреченных при трансляции, извлекаемый по ссылке на этапе исполнения. В разработанной реализации в случае использования переменной в стек добавляется инструкция “VAR”, хранящая ссылку на название переменной и на тип, в случае объявления. Трансляция констант начинается с занесения в стек инструкции “CONST”, хранящей индекс значения и тип константы в случае инициализации.

2 Реализация системы тестирования

2.1 Процесс тестирования

Под тестированием компилятора понимается сборка транслятора из исходных кодов, трансляция тестовой программы и сравнение результатов работы транслятора с ожидаемым результатом. Было решено автоматизировать процесс тестирования, для повышения качества и снижения сложности процесса.

Т.к. язык программирования и среда КуМир - кроссплатформенные проекты, в начале процесса автоматизации стоило решить вопрос с внешними условиями - операционной системой, набором используемых пакетов и библиотек, т.о. первым шагом к автоматизации тестирования явилось создание централизованной среды для сборки компилятора, а именно разработка dockerfile файла, содержащий набор инструкций для сборки транслятора из исходных кодов. В качестве операционной системы была выбрана Ubuntu, ввиду широкой распространенности, большого сообществу разработчиков и активной поддержке пакетов с данной системой в репозитории docker.

Далее встал вопрос автоматизации запуска транслятора. Для решения этой проблемы был разработан скрипт на языке Python 3.

Для автоматизации запуска тестовой системы в репозитории разрабатываемого транслятора были настроены службы автоматического запуска(Github Actions) при отправке изменений и создании запроса на внесение изменений в целевую ветвь. Исходный код, используемый для инициализации системы тестов приведен в приложении А.

2.2 Тесты

Для тестирования были разработаны набор тестов, покрывающие основные инструкции языка программирования КуМир, а именно: работа с переменными(инициализация, присваивание), арифметические и логические выраже-

ния, ветвления, циклы, а также функции и процедуры. Тестирование построено на принципе сравнения ожидаемого результата с результатом работы транслятора. Тесты сгруппированы по подмножествам инструкций исходного языка.



Рисунок 1 — Группы тестов

Группа состоит из секций, представляющих отдельные инструкции. Для создания тестовой секции, например, для тестирования трансляции функций, необходимо 2 файла - файл с расширением .kum на исходном языке программирования, а также файл с расширением .exp - файл с предполагаемым результатом работы транслятора, содержащий программу на языке C++. Пример секции тестов приведен на рисунке 2.

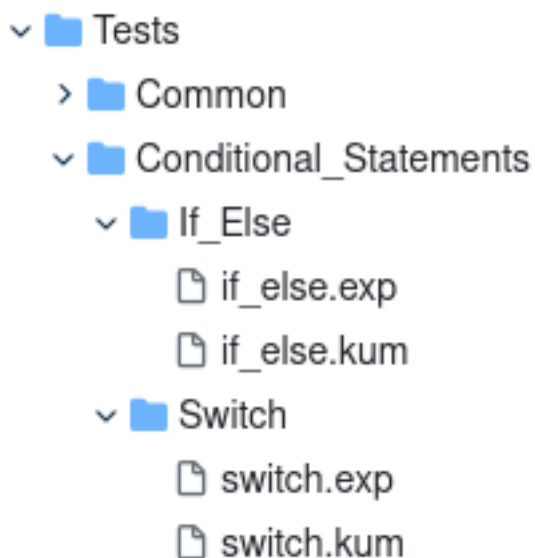


Рисунок 2 — Пример секции тестов

Под тестом транслятора понимается 2 файла, содержащие 1 или несколько инструкций на исходном и выходном языках программирования. На данный момент разработано чуть больше 200 тестов.

2.3 Программа-тестер

Для автоматизации тестирования было разработано консольное приложение-тестер транслятора. В качестве языка программирования был выбран Python 3. Для работы с программой был разработан ряд аргументов командной строки, а также справка. Список аргументов командной строки приведен ниже:

```
1  [-h] [--help] - show help.
2  [-tr] [--translator] ["the path to pre-built kumir2 to arduino translator
3  instance"] - show app what translator instance to use.
4  [-t] [--path-to-tests] ["the path to tests folder"] = show app
5  the folder with test files.
6  [-o] [--output] ["the path to log file"] - show app where to store test logs.
7  [-d] [--duplicate] - duplicate output to console.
8  [-ss] [--skip-successfull] - skip open log info about succesfully completed tests.
9  [-sf] [--skip-failed] - skip open log info about failed tests.
10 [-swe] [--skip-without-expectation] - skip open log info about
11 tests for which the file with expectations was not found.
12 [-sce] [--skip-with-compiler-error] - skip open log info about
13 tests ended with compiler error.
14 [-b] [--brief] - skip open log info about all tests.
```

Было решено разработать программу гибкой и информативной. Для работы необходимы 2 аргумента - `--translator`(путь к транслятору), а также `--path-to-tests`(путь к папке с тестами). Исходный код скрипта для тестирования приведен в приложении Б. По умолчанию результат тестирования записывается в файл, также есть возможность продублировать вывод в консоль. Существует 4 состояния, отражающие результат работы компилятора:

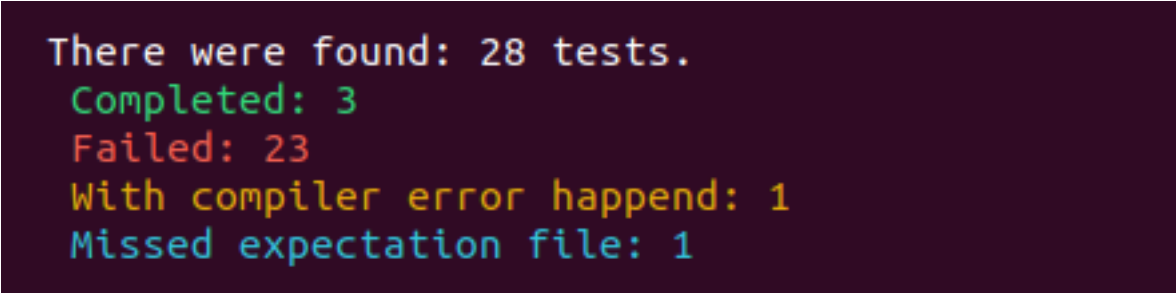
1. успех;
2. провал;
3. ошибка компиляции;
4. не хватает файла с ожиданиями.

Состояния 3 и 4 могут быть получены по причине некорректного содержания тестовых файлов - либо написанная на языке программирования КуМир программа имеет ошибки и не может быть скомпилирована, либо не был найден файл с расширением .ехр, содержащий ожидания работы компилятора. В других случаях, текст сообщения с результатами тестирования содержит предложение вызвать команду `vimdiff` с путями до файлов и прореферировать результат. Можно посмотреть как три файла сразу (исходный код, ожидаемый результат, результат работы компилятора), так и только ожидания и результат работы. Пример сообщения о результатах работы приведен на рисунке 3.

```
Test №0. Test /home/anton/Sources/kumir2/kumir_tests/Sources/2.
Test is not completed(
Sources for the test: /home/anton/Sources/kumir2/kumir_tests/Sources/2.kum
Test expectations: /home/anton/Sources/kumir2/kumir_tests/Expectations/2.c
Test results: /home/anton/Sources/kumir2/kumir_tests/Results/2.kumir.c.
To get more detail info about comparison results, print:
vimdiff /home/anton/Sources/kumir2/kumir_tests/Sources/2.kum /home/anton/Sources/kumir2/
kumir_tests/Expectations/2.c /home/anton/Sources/kumir2/kumir_tests/Results/2.kumir.c
or vimdiff /home/anton/Sources/kumir2/kumir_tests/Expectations/2.c /home/anton/Sources/kumir2/
kumir_tests/Results/2.kumir.c
```

Рисунок 3 — Пример сообщения с возможностью подробно посмотреть результаты тестирования

В начале вывода информации приведена краткая статистика - сколько тестов завершилось и с каким состоянием. Объем вывода можно фильтровать и убирать подробную информацию о сообщениях в состояниях 1-4, либо полностью отключить подробный вывод информации, добавив флаг `-b`. При дублировании вывода в консоль, в зависимости от состояния, сообщение будет выделено цветом. Пример краткого вывода по результатам тестирования приведен на рисунке 4.

A screenshot of a terminal window with a dark purple background. It displays the results of 28 tests in a monospaced font. The text is color-coded: 'There were found: 28 tests.' is white, 'Completed: 3' is green, 'Failed: 23' is red, 'With compiler error happend: 1' is yellow, and 'Missed expectation file: 1' is cyan.

```
There were found: 28 tests.  
Completed: 3  
Failed: 23  
With compiler error happend: 1  
Missed expectation file: 1
```

Рисунок 4 — Пример краткого вывода по окончании тестирования

На данный момент, система тестирования демонстрирует следующие результаты:

1. 3 теста завершились успешно;
2. 23 теста провалились;
3. во время запуска 1 теста произошла ошибка компиляции;
4. во время запуска 1 теста не был найден файл с ожиданиями.

Полученные результаты демонстрируют большой процент неточностей в работе разработанного транслятора. Найденные ошибки в работе необходимо исправить.

ЗАКЛЮЧЕНИЕ

В ходе тестирования разработанного транслятора с языка программирования КуМир в язык программирования С++ был выявлен ряд ошибок. Автоматизация процесса тестирования позволяет быстро расширять список тестов и увеличивать процент покрытия, влияющий на надежность и безотказность работы транслятора. Создание тестовых случаев позитивно сказывается на процессе разработки, ведь тестовые сценарии являются документацией. Автоматизация запуска программы-тестера при помощи Github Actions позволит быстрее отсеивать некачественный код, влияющий на работоспособность транслятора. В дальнейшем разработанная система тестов может быть доработана при помощи методов Fuzzing-тестирования и автоматизированной генерации тестовых файлов. Исходный код разработанной программы тестера и транслятора можно найти в открытом репозитории [7].

В рамках ВКР был разработан транслятор с языка программирования КуМир в язык С++. Программа имеет ряд недостатков, которые предстоит исправить и список улучшений, которые планируется реализовать. Среди задач по улучшению транслятора можно выделить следующие:

- создание отдельного клиента среды программирования КуМир, специально для разработки роботов;
- добавление возможности прошивки робота из клиента;
- добавление инструмента выбора порта с подключенным роботом для прошивки;
- добавление настраиваемого алгоритма прошивки, определяющего роль результата трансляции в архитектуре программы для прошивки робота.

После реализации клиента среды программирования КуМир для разработки роботов и исправления ошибок транслятора найденных в ходе тестиро-

вания планируется спроектировать и разработать робота на базе аппаратного комплекса Arduino для опробации разработки в школах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Статья о КуМире на электронном образовательном портале Фоксфорд [Электронный ресурс] URL: *[https : //foxford.ru/wiki/informatika/sredaprogrammirovaniya-kumir](https://foxford.ru/wiki/informatika/sredaprogrammirovaniya-kumir)*
- 2 Статья о предназначении комплектов Arduino [Электронный ресурс] URL: *[http : //arduino.ru/](http://arduino.ru/)*
- 3 Максименков Д.А., Рогов Р.Ю. Применение метода инструментирования тестовых программ при отладке оптимизирующих компиляторов. Вопросы радиоэлектроники, 2010, вып. 3, стр. 50-61
- 4 IBM. IBM Rational Functional Tester // IBM.
- 5 IBM. IBM Rational Test Workbench // IBM.
- 6 SmartBear. TestComplete Platform // SmartBear.
- 7 Репозиторий с исходным кодом транслятора [Электронный ресурс] URL:*[https : //github.com/CaMoCBaJL/kumir2/tree/translator_tests](https://github.com/CaMoCBaJL/kumir2/tree/translator_tests)*
- 8 Система программирования Кумир 2.x А.Г.Кушниренко , М. А. Ройтберг , Д.В.Хачко, В. В. Яковлев [Электронный ресурс] URL:*[http : //roytberg.lpm.org.ru/pdfs/kumir2x2015.pdf](http://roytberg.lpm.org.ru/pdfs/kumir2x2015.pdf)*
- 9 Леонов А.Г., Кушниренко А.Г. Методика преподавания основ алгоритмизации на базе системы «КуМир». М.: «Первое сентября», 2009.
- 10 Кушниренко А.Г., Рогожкина И.Б., Леонов А.Г. // Большой московский семинар по методике раннего обуч. информатике (ИТО-РОИ-2012): сб. докл. ПиктоМир: пропедевтика алгоритмического языка (опыт обучения программированию старших дошкольников). М.: Конгресс конференций ИТО-РОИ, 2012.

ПРИЛОЖЕНИЕ А

Исходный код dockerfile, унифицирующий настройки среды запуска системы тестирования

```
1 FROM ubuntu
2
3 ARG path_to_tests_folder="./Tests"
4
5 LABEL EMAIL=gorka19800@gmail.com
6
7 #install all the necessary libs and apps
8 RUN apt-get update
9 RUN apt-get dist-upgrade -y
10 RUN echo "8"| apt-get install -y qttools5-dev-tools
11 RUN apt-get install -y git python3 cmake qtbase5-dev g++ libqt5svg5-dev
12   libqt5x11extras5-dev qtscript5-dev libboost-system-dev zlib1g zlib1g-dev
13 #setup git
14 RUN git config --global user.email "gorka19800@gmail.com"
15 RUN git config --global user.name "Test suit"
16 #clone repo and prepare for building kumir-to-arduino translator
17 RUN mkdir /home/Sources
18 WORKDIR /home/Sources/
19 RUN git clone https://github.com/CaMoCBaJL/kumir2
20 WORKDIR kumir2/
21 RUN git pull
22 RUN git checkout translator_tests
23 RUN git merge -s ours --no-edit origin/ArduinoFixes
24 RUN mkdir build
25 WORKDIR build/
26 #build translator
27 RUN cmake -DUSE_QT=5 -DCMAKE_BUILD_TYPE=Release ..
28 RUN make -j 18
29 #start tests
```

```
30 WORKDIR ../kumir_tests/  
31 RUN touch test_results.log  
32 RUN python3 test_script.py -d -o ./test_results.log  
33 -tr ../build/bin/kumir2-arduino -t \${path_to_tests_folder}
```


ПРИЛОЖЕНИЕ Б

Исходный код скрипта, производящего тестирование транслятора

```
1  import os
2  import sys
3  import subprocess
4
5  #constants
6  class CONSOLE_BG_COLORS:
7      HEADER = '\033[95m'
8      OKBLUE = '\033[94m'
9      OKCYAN = '\033[96m'
10     OKGREEN = '\033[92m'
11     WARNING = '\033[93m'
12     FAIL = '\033[91m'
13     ENDC = '\033[0m'
14     BOLD = '\033[1m'
15     UNDERLINE = '\033[4m'
16
17  class TEST_RESULT_STATE:
18      COMPLETED = 1
19      MISSING_EXPECTATION = 2
20      COMPILER_ERROR_HAPPEND = 3
21      FAILED = -1
22      NONE = 0
23
24  COMPILER_ERROR_LABEL = "ERROR!"
25
26  EXPECTATION_FILE_EXTENTION = ".exp"
27  SOURCE_FILE_EXTENTION = ".kum"
28
29  CONTROL_CHARACTERS = ["\n", "\r", "\t", " "]
30  BYTES_TO_READ_COUNT = 1024
```

```

31 CHAR_THRESHOLD = 0.3
32 TEXT_CHARACTERS = ''.join(
33     [chr(code) for code in range(32, 127)] +
34     list('\b\f\n\r\t')
35 )
36 BINARY_CHAR_EXAMPLE = '\x00'
37
38 TESTS_FOLDER_NAME = "Tests"
39
40 ARGS = {"help": ["-h", "--help"],
41         "translator": ["-tr", "--translator"],
42         "output": ["-o", "--output"],
43         "duplicate": ["-d", "--duplicate"],
44         "skip-successfull": ["-ss", "--skip-successfull"],
45         "skip-failed": ["-sf", "--skip-failed"],
46         "skip-without-expectation": ["-swe", "--skip-without-expectation"],
47         "skip-with-compiler-error": ["-sce", "--skip-with-compiler-error"],
48         "brief": ["-b", "--brief"],
49         "path-to-tests": ["-t", "path-to-tests"]
50     }
51
52 ERRORS = {
53     "wrong input": "Wrong args input - you should type path to file after -c or
54     -o flags!",
55     "file not exist": "File doesn't exist!",
56     "wrong file extention": "File extention for translator is not correct! It should
57     be a bin-file.",
58     "no path to tests": "Wrong args input - you should type path to tests folder
59     after -t flag!"
60 }
61
62 #data structure to store test results
63 class TestResult:

```

```

64
65     __text_color = CONSOLE_BG_COLORS.OKGREEN
66     __header_text = ""
67     SKIPPED_TEST_TYPES = [TEST_RESULT_STATE.NONE]
68
69     def __init__(self, name: str, source: str, expectation: str, result: str):
70         self.name = name
71         self.source_file_name = source
72         self.expectation_file_name = expectation
73         self.resultFileName = result
74         self.state = TEST_RESULT_STATE.NONE
75
76     def __str__(self):
77         self.__setup_output()
78         additional_test_data = f'''
79         Test group expectations: {self.expectation_file_name}
80         Test group results: {self.resultFileName}.
81         To get more detail info about comparison results, print:
82         vimdiff {self.source_file_name} {self.expectation_file_name}
83             {self.resultFileName}
84         or vimdiff {self.expectation_file_name} {self.resultFileName}
85         {CONSOLE_BG_COLORS.ENDC}
86         '''
87         return f'''{self.__text_color}
88         Test {self.name}.
89         {self.__header_text}
90         Sources for the test: {self.source_file_name}
91         {additional_test_data if self.state is TEST_RESULT_STATE.COMPLETED
92         or self.state is TEST_RESULT_STATE.FAILED else ""}
93         '''
94
95     def __setup_output(self):
96         if self.state == TEST_RESULT_STATE.COMPLETED:

```

```

97         self.__text_color = CONSOLE_BG_COLORS.OKGREEN
98         self.__header_text = "Congratulations! Test completed successfully!"
99     elif self.state == TEST_RESULT_STATE.MISSING_EXPECTATION:
100         self.__text_color = CONSOLE_BG_COLORS.OKCYAN
101         self.__header_text = "Oh! Test didn't complete: no expectation"
102     elif self.state == TEST_RESULT_STATE.COMPILER_ERROR_HAPPEND:
103         self.__text_color = CONSOLE_BG_COLORS.WARNING
104         self.__header_text = "Oh! Test didn't complete: compiler error"
105     elif self.state == TEST_RESULT_STATE.FAILED:
106         self.__text_color = CONSOLE_BG_COLORS.FAIL
107         self.__header_text = "Sorry, but test failed..."
108
109     class TestSection:
110
111         def __init__(self, section_name) -> None:
112             self.name = section_name
113             self.test_results = []
114
115         def __str__(self) -> str:
116             columns, _ = os.get_terminal_size()
117             if (len(self.test_results) > 0):
118                 return f"""
119                 {CONSOLE_BG_COLORS.WARNING + "-" *columns + CONSOLE_BG_COLORS.ENDC}
120                 Test section {self.name} starts here:
121                 {os.linesep.join(list(map(lambda test_result: str(test_result),
122                 self.test_results))))}
123                 End of {self.name} section tests
124                 {CONSOLE_BG_COLORS.WARNING + "-" *columns + CONSOLE_BG_COLORS.ENDC}
125                 """
126
127             return ''
128
129     #functions

```

```

130 def remove_control_characters(data_array):
131     result = []
132     for i in data_array:
133         for cc in CONTROL_CHARACTERS:
134             i = i.replace(cc, "")
135
136         if i:
137             result.append(i)
138
139     return result
140
141 def get_file_data(filename):
142     if not os.path.exists(filename):
143         return ''
144     file = open(filename, "r")
145     result = file.readlines()
146     file.close()
147     return result
148
149 def has_errors(text):
150     return COMPILER_ERROR_LABEL in text
151
152 def process_sources(source_filename, path_to_translator):
153     path, _ = os.path.splitext(source_filename)
154     result_filename = path + ".kumir.c"
155
156     #call kumir2-arduino with params:
157     --out="path_to_cwd/results/test_name.kumir.c" -s ./test_name.kum
158     popen = subprocess.Popen([path_to_translator,
159                               f'--out={result_filename}',
160                               '-s',
161                               source_filename],
162                               stdout=subprocess.PIPE)

```

```

163     popen.wait()
164     popen.stdout.read()
165
166     return result_filename
167
168 def compare_data(expected_data, processed_data) -> TEST_RESULT_STATE:
169     result_without_kumir_ref = remove_control_characters(processed_data[2:])
170     expected_data = remove_control_characters(expected_data)
171
172     if (has_errors(result_without_kumir_ref)):
173         return TEST_RESULT_STATE.COMPILER_ERROR_HAPPEND
174
175     for i in range(len(result_without_kumir_ref)):
176         if result_without_kumir_ref[i] != expected_data[i]:
177             return TEST_RESULT_STATE.FAILED
178
179     return TEST_RESULT_STATE.COMPLETED
180
181 def get_test_result_type_counters(test_sections):
182     counters = {
183         TEST_RESULT_STATE.COMPLETED: 0,
184         TEST_RESULT_STATE.FAILED: 0,
185         TEST_RESULT_STATE.COMPILER_ERROR_HAPPEND: 0,
186         TEST_RESULT_STATE.MISSING_EXPECTATION: 0,
187     }
188
189     for test_section in test_sections:
190         for test_result in test_section.test_results:
191             counters[test_result.state] += 1
192
193
194     return counters
195

```

```

196 #log comparison results to file
197 def log_test_results(logs_filename, data_to_log: TestSection, log_to_console):
198     result_type_counters = get_test_result_type_counters(data_to_log)
199     completed_tests_count, failed_tests_count,
200     compiler_error_happend_tests_count,
201     missing_expectations_tests_count = \
202     [result_type_counters[k] for k in result_type_counters]
203
204     log_data = []
205     for test_section in data_to_log:
206         test_section.test_results = list(filter
207         (lambda test_result: test_result.state not in
208         TestResult.SKIPPED_TEST_TYPES, test_section.test_results))
209         log_data.append(test_section)
210
211     log_data_strings = list(map(lambda x: str(x), log_data))
212     log_data_strings.insert(0, f'''
213     There were found: {len(log_data)} tests.
214     {CONSOLE_BG_COLORS.OKGREEN} Completed:
215     {completed_tests_count} {CONSOLE_BG_COLORS.ENDC}
216     {CONSOLE_BG_COLORS.FAIL} Failed:
217     {failed_tests_count} {CONSOLE_BG_COLORS.ENDC}
218     {CONSOLE_BG_COLORS.WARNING} With compiler error happend:
219     {compiler_error_happend_tests_count} {CONSOLE_BG_COLORS.ENDC}
220     {CONSOLE_BG_COLORS.OKCYAN} Missed expectation file:
221     {missing_expectations_tests_count} {CONSOLE_BG_COLORS.ENDC}''')
222
223     if not os.path.exists(logs_filename):
224         open(logs_filename, "a").close()
225
226     log_file = open(logs_filename, "a")
227     log_file.writelines(log_data_strings)
228     log_file.close()

```

```

229
230     if log_to_console:
231         print(f"{os.linesep}".join(log_data_strings))
232
233 def is_binary_file(filename):
234     file_stream = open(filename, 'rb')
235     file_content = file_stream.read(BYTES_TO_READ_COUNT)
236     file_stream.close()
237
238     if not len(file_content):
239         #file is empty, nothing to read
240         return False
241
242     if ord(BINARY_CHAR_EXAMPLE) in file_content:
243         #file contains binary symbols
244         return True
245
246     binary_chars = file_content.translate(TEXT_CHARACTERS)
247     return float(len(binary_chars)) / len(file_content) > CHAR_THRESHOLD
248
249 def validate_file_name(filename: str, is_log_file: bool):
250     if not os.path.exists(filename):
251         print(filename + ERRORS.get("file not exist"))
252         sys.exit(2)
253     if not os.path.isfile(filename):
254         print(ERRORS.get("wrong input"))
255         sys.exit(2)
256     if not is_log_file and not is_binary_file(filename):
257         print(ERRORS.get("wrong file extention"))
258         sys.exit(2)
259
260 def show_help():
261     print(""" kumir2-arduino tester.

```


Description:

Approach of this app is to debug the work of kumir2 to arduino translator. It uses compiled translator's instance, pre-built locally on PC.

To start the work you should input path to compiler and path to logs file.

Flags:

`[-h] [--help]` - show help.

`[-tr] [--translator] ["the path to pre-built kumir2 to arduino translator instance"]` - show app what translator instance to use.

`[-t] [--path-to-tests] ["the path to tests folder"]` = show app the folder with test files.

`[-o] [--output] ["the path to log file"]` - show app where to store test logs.

`[-d] [--duplicate]` - duplicate output to console.

`[-ss] [--skip-successfull]` - skip open log info about successfully completed tests.

`[-sf] [--skip-failed]` - skip open log info about failed tests.

`[-swe] [--skip-without-expectation]` - skip open log info about tests for which the file with expectations was not found.

`[-sce] [--skip-with-compiler-error]` - skip open log info about tests ended with compiler error.

```

295     [-b] [--brief] - skip open log info about all tests.
296     """)
297
298 def process_args():
299     result = ["", "", False, ""]
300     if ARGS["help"][0] in sys.argv or ARGS["help"][1]
301     in sys.argv:
302         show_help()
303         sys.exit(2)
304
305     if ARGS["skip-successfull"][0] in sys.argv or ARGS["
306     skip-successfull"][1] in sys.argv:
307         TResult.SKIPPED_TEST_TYPES.append(
308             TEST_RESULT_STATE.COMPLETED)
309
310     if ARGS["skip-without-expectation"][0] in sys.argv
311     or ARGS["skip-without-expectation"][1] in sys.argv:
312         TResult.SKIPPED_TEST_TYPES.append(TEST_RESULT_STATE
313             .MISSING_EXPECTATION)
314
315     if ARGS["skip-with-compiler-error"][0] in sys.argv or
316     ARGS["skip-with-compiler-error"][1] in sys.argv:
317         TResult.SKIPPED_TEST_TYPES.append(TEST_RESULT
318             _STATE.COMPILER_ERROR_HAPPEND)
319
320     if ARGS["skip-failed"][0] in sys.argv or ARGS[
321     "skip-failed"][1] in sys.argv:
322         TResult.SKIPPED_TEST_TYPES.append(
323             TEST_RESULT_STATE.FAILED)
324
325     if ARGS["brief"][0] in sys.argv or ARGS["brief"][1] in sys.argv:
326         TResult.SKIPPED_TEST_TYPES = [
327             TEST_RESULT_STATE.COMPILER_ERROR_HAPPEND,

```

```

328         TEST_RESULT_STATE.COMPLETED,
329         TEST_RESULT_STATE.FAILED,
330         TEST_RESULT_STATE.MISSING_EXPECTATION
331     ]
332
333     args = sys.argv[1:]
334     for i in range(1, len(args)):
335         if args[i - 1] in ARGS["translator"] or args[i - 1]
336             in ARGS["output"] or args[i - 1] in ARGS
337             ["path-to-tests"]:
338             if (os.path.isfile(args[i])):
339                 validate_file_name(args[i], False if args[i - 1]
340                     in ARGS["translator"] else True)
341
342             if args[i - 1] in ARGS["translator"]:
343                 result[0] = os.path.abspath(args[i])
344             elif args[i - 1] in ARGS["output"]:
345                 result[1] = os.path.abspath(args[i])
346             elif args[i - 1] in ARGS["path-to-tests"]:
347                 result[3] = os.path.abspath(args[i])
348             elif args[i-1] in ARGS["duplicate"]:
349                 result[2] = True
350
351     return result
352
353 def get_files_with_absolute_paths(folder_name):
354     path_to_folder = os.path.join(os.getcwd(), folder_name)
355     files =
356     list(map(lambda x: os.path.join(path_to_folder, x),
357         os.listdir(path=path_to_folder)))
358     files.sort()
359
360     return files

```

```

361
362 def get_source_and_expectation(dir_files):
363     sources = []
364     expectations = []
365     for file in dir_files:
366         if (os.path.isfile(file)):
367             ext = os.path.splitext(file)[1]
368             if (ext == EXPECTATION_FILE_EXTENTION):
369                 expectations.append(file)
370             elif (ext == SOURCE_FILE_EXTENTION):
371                 sources.append(file)
372
373     return [sources, expectations]
374
375 def get_folder_contents_full_paths(path_to_folder):
376     return list(
377         map(
378             lambda x: os.path.join(os.sep, path_to_folder, x),
379             os.listdir(path_to_folder)
380         )
381     )
382
383 def calculate_test_sections(path_to_tests_folder):
384     result = []
385     test_folder_paths = get_folder_contents_full
386     _paths(path_to_tests_folder)
387     for test_folder_path in test_folder_paths:
388         if os.path.isfile(test_folder_path):
389             continue
390
391         result.append(TestSection(test_folder_path.
392             split(os.sep)[-1]))
393         test_paths = get_folder_contents_full_paths

```

```

394         (test_folder_path)
395     for test_dir_path in test_paths:
396         source_files, expectation_files =
397         get_source_and_expectation(get_folder_contents_full_paths
398         (test_dir_path))
399         if os.path.isfile(test_dir_path) or len
400         (expectation_files) > 1 or len(source_files)
401         < 1:
402             continue
403
404         result[-1].test_results.append(TestMethod(
405             test_dir_path.split(os.sep)[-1],
406             source_files[0],
407             "", ""
408         ))
409     )
410
411     if len(expectation_files) == 1:
412         result[-1].test_results[-1]
413         .expectation_file_name = expectation_files[0]
414         result[-1].test_results[-1].resultFileName
415         = process_sources(source_files[0],
416             args_data[0])
417         result_data = get_file_data(result[-1]
418
419             .test_results[-1].resultFileName)
420         expected_data =
421         get_file_data(result[-1].test_results[-1].expectation_file_name)
422         result[-1].test_results[-1].state
423         = compare_data(expected_data, result_data)
424     else:
425         result[-1].test_results[-1].state =
426         TEST_RESULT_STATE.MISSING_EXPECTATION

```

```
427
428     return result
429
430 if __name__=="__main__":
431     args_data = process_args()
432
433     if not args_data[3]:
434         print("Didn't find any test to execute. Shutting down.")
435         sys.exit()
436
437     test_results = calculate_test_sections(args_data[3])
438
439     log_test_results(args_data[1], test_results, args_data[2])
```