МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

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

Кафедра Дискретной математики и информационных технологий

ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ТРАНСЛЯТОРА ЯЗЫКА ПРОГРАММИРОВАНИЯ КУМИР В ЯЗЫК С++

БАКАЛАВРСКАЯ РАБОТА

студента 4 курса 421 группы

направления 09.03.01 — Информатика и вычислительная техника

факультета КНиИТ

Пронина Антона Алексеевича

Научный руководитель

доцент _____ Е. А. Синельников

Заведующий кафедрой

к. ф.-м. н. _____ Л. Б. Тяпаев

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Стремительное вхождение в повседневную жизнь информационных и коммуникационных технологий стало возможным благодаря широкому распространению персональных компьютеров и созданию глобальной сети Интернет. В связи с этим вопрос совершенствования и модернизации сложившейся образовательной системы остается актуальным [1]. Одним из наиболее интересных вопросов, требующих особого внимания в обучении информатике, является вопрос системы обучения программированию. Это связано с тем, что профессия специалистов в области информатики и информационных технологий в какой-то мере начинается со школы. Одним из прямых приложений программирования является робототехника. Многие исследователи отмечают актуальность развития данного направления в рамках обучения школьников [2], [3], [4]. Образовательная робототехника - уникальный инструмент обучения, который помогает сформировать привлекательную для детей учебную среду с практически значимыми и занимательными мероприятиями, подкрепляющими интерес учащихся к изучаемым предметам [5].

В то же время, изучение языка программирования КуМир является неотъемлемой частью обучения в школе в рамках дисциплины “Информатика”. Ввиду растущего интереса в сфере образования к обучению робототехнике в школе и широкой распространенности и глубокой степени интеграции современной образовательной системы с этим языком, появилась идея о разработке транслятора с языка КуМир в язык C++.

Актуальность идеи 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 ап-

паратной платформы данной задачи был выбран электронный конструктор и удобная платформа быстрой разработки электронных устройств для новичков и профессионалов [6] - 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.

Целью данной работы является разработка транслятора со школьного алгоритмического языка программирования КуМир в язык программирования C++. Для достижения поставленной цели необходимо решить следующие задачи:

- изучить существующие алгоритмы и методы трансляции;
- изучить набор основных команд платформы Arduino;
- изучить набор основных команд и архитектуру программ в среде программирования КуМир;
- проанализировать исходные коды языка программирования КуМир;
- разработать модуль для предоставления набора основных команд платформы Arduino;
- разработать транслятор с языка КуМир в язык C++.

1 Предметная область. Подходы, понятия, средства

1.1 Среда исполнения КуМир

КуМир (Комплект Учебных МИРов) - система программирования, предназначенная для поддержки начальных курсов информатики и программирования в средней и высшей школе с открытым исходным кодом [7]. Спектр возможностей применения данной системы программирования ограничен, ряд стандартных команд позволяет разрабатывать небольшие приложения с целью изучения основ программирования на процедурных языках. На данный момент язык КуМир - это язык, с которого хорошо начать, чтобы освоить основы алгоритмического подхода и процедурный стиль программирования [8].

Система КуМир [9] в современном ее состоянии (с подсистемой ПиктоМир) состоит из расширяемого набора исполнителей (или роботов), набора систем программирования и вспомогательных утилит и программ. Расширяемый набор исполнителей (роботов) представляет собой отдельные самостоятельные программы и (или) электронно-механические устройства, имеющие собственное пультовое управление (интерфейс), а также возможность локального или сетевого управления из выполняющей системы.

Набор систем программирования:

- система программирования КуМир на школьном алгоритмическом языке, состоящая из редактора-компилятора алгоритмического языка с многооконным интерфейсом, интегрированная с выполняющей системой. Система не является полноценным интерпретатором или компилятором школьного алгоритмического языка. Зачастую, программа на школьном алгоритмическом языке компилируется в некий промежуточный код (подобный подход был использован в языке Java), который затем интерпретируется и исполняется с автоматической генера-

цией точек останова по событиям и шагам (при пошаговом выполнении). В системе программирования КуМир также отсутствует отладчик в его классическом понимании. Это имеет свои причины, так как сильно упрощает процесс освоения и написания системы программирования КуМир, а также отладку программ. Все изменения используемых в школьной программе величин автоматически визуализируются на полях программы. Визуализируются и результаты логических операций. Кроме того, к системе можно подключать любого исполнителя (робота) из набора и программно управлять им;

- система бестекстового, пиктографического программирования ПиктоМир. В ней учащийся собирает программу из команд исполнителя (робота). Управляющие конструкции языка представлены определенной параметрической организацией алгоритмов. Созданная учащимися программа при выполнении передает команды робота и получает его состояние;
- система программирования на производственном языке программирования C++, Python и т.п;

Вспомогательные утилиты и программы – это всевозможные редакторы миров исполнителей (роботов), конвертеры и препроцессоры из пиктографического языка в школьный алгоритмический язык, из школьного алгоритмического языка в C++ и Python [10].

Данная платформа поддерживает возможность добавления модулей и систем программирования, расширяющих спектр возможностей использования языка. Одним из таких модулей является модуль для трансляции исходного кода на языке КуМир в язык C++, использующийся для разработки программ для микропроцессорных систем Arduino.

1.2 Аппаратно-программный комплекс Arduino

Arduino - комплекс аппаратно-программных средств с открытым исходным кодом, обладающий низким порогом вхождения как со стороны программирования, так и со стороны электроники. Электронные платы способны считывать и генерировать входные и выходные сигналы. Для передачи последовательности команд управления плате используется язык программирования Arduino и соответствующая среда исполнения.

Многолетняя история развития платформы Arduino включает в себя тысячи проектов - от небольших устройств до крупных научных инструментов. Сообщество разработчиков данной платформы состоит из программистов различного уровня - от любителей до профессионалов, собравших и систематизировавших свой опыт взаимодействия с Arduino для помощи в вопросах разработки проектов, полезный как новичкам, так и профессионалам.

Arduino стала самой популярной платформой для разработки встроенных систем благодаря простоте взаимодействия и дизайну. Программное обеспечение может быть легко использовано на любой операционной системе. Студенты и преподаватели используют комплекс Arduino для разработки недорогих научных инструментов, для исследования физических и химических законов, а также для изучения основ программирования и робототехники. Дизайнеры и архитекторы создают интерактивные прототипы, музыканты и художники используют его для создания инсталляций и экспериментирования с новыми инструментами. Большим плюсом является огромное число инструкций по созданию устройств шаг-за-шагом, позволяющим любому разработчику повторить и попробовать существующее решение.

Основные плюсы:

- низкая стоимость плат и компонентов

платы Arduino сравнительно недороги в сравнении с прочими программно-

аппаратными платформами. Минимальная конфигурация системы может быть собрана вручную, используя готовые модули, стоимостью не более 50$;

- кросс-платформенность

среда исполнения Arduino исполняется на Windows, Macintosh OSX и Linux, хотя прочие микроконтроллерные системы доступны лишь для разработки, используя ОС Windows;

- простая, чистая среда для программирования

редактор кода Arduino прост и гибок как для новичков, так и для профессионалов в данной сфере. Для преподавателей его удобность кроется в среде программной обработки, позволяя студентам создавать программы на языке программирования высокого уровня без глубоких знаний в низкоуровневом программировании;

- открытое расширяемое ПО

среда программирования Arduino - свободно распространяемая среда исполнения программного кода, написанного на языке программирования, схожем с AVR C, на котором основана платформа;

- открытая и расширяемая аппаратная часть

схемы плат Arduino публикуются в сети с использованием открытой лицензии, позволяющей любому разработчику плат воспроизвести плат в соответствии с чертежом и модернизировать её. Даже неопытные пользователи могут создать кальку модуля, используя макетную плату для понимания механизмов её работы и экономии денег [11].

1.3 Arduino в образовании

Развитие современных информационных технологий будет связано не только с появлением новых технических совершенных устройств, но и с раз-

витием робототехники. Основы робототехники еще не является обязательной составляющей ФГОС ООО, поэтому обучение этому курсу возможно в рамках внеурочной деятельности или предпрофильной подготовки (элективные курсы), а также в профильном обучении в 10–11 классах. Хочется подчеркнуть, что существующие образовательные программы по информатике позволяют использовать робототехнику, микроэлектронику (и инженерные составляющие) как методический инструмент учителя, без необходимости изменения рабочей программы педагога. Использование платформы Arduino для образовательных учреждений позволяет получить возможность развить навыки программирования на практике, а также освоить азы схемотехники, дает возможность обучающимся освоить основные приемы разработки аппаратной и программной части автономных автоматизированных комплексов.

Исследователи вопроса актуальности изучения робототехники на базе платформы Arduino в образовательном процессе отмечают повышение креативности учащихся [12], активное формирование и оттачивание профессиональных навыков и умений, а также развиваются и осваиваются компетенции естественно-научной направленности, связанные с физико-техническими дисциплинами [13].

1.4 Грамматики и деревья разбора

Код на языке КуМир исполняется либо на низкоуровневой виртуальной машине, не зависящей от используемой операционной системы, либо транслируется в машинные коды используемого вычислительного устройства. Во время анализа исходных кодов платформы КуМир был выявлен использующийся механизм трансляции кода. Помимо стандартных методов и фаз трансляции, включающих лексический, синтаксический и семантические анализы, так же используются AST-деревья (абстрактные деревья разбора).

Одной из основных структур данных, используемых при компиляции на различных этапах являются деревья разбора.

Определение: Для некоторой грамматики $G =< N, \Sigma, P, S >$, где

N-множество нетерминалов;

Σ - множество терминалов;

P - множество правил;

S - начальный нетерминал(аксиома) грамматики,

дерево вывода (разбора) определяется следующими правилами: каждый узел имеет метку — $n \in N \cup \Sigma \cup \{\varepsilon\}$; метка корня - S; если узел k хотя бы одного потомка $\Rightarrow k \in N$; если узлы $n_1, \ldots, n_k$ - прямые потомки узла n, перечисленные слева направо с метками $A_1, \ldots, A_k$ соответственно, а метка узла $n \in A \Rightarrow A \to A_1 \ldots A_k P$, причем n_i может быть помечен ε только, если n_i - единственный потомок $n, A \to \varepsilon \in P$. Пример дерева разбора для выражения $2 * 7 + 3$ приведен на рисунке 1.

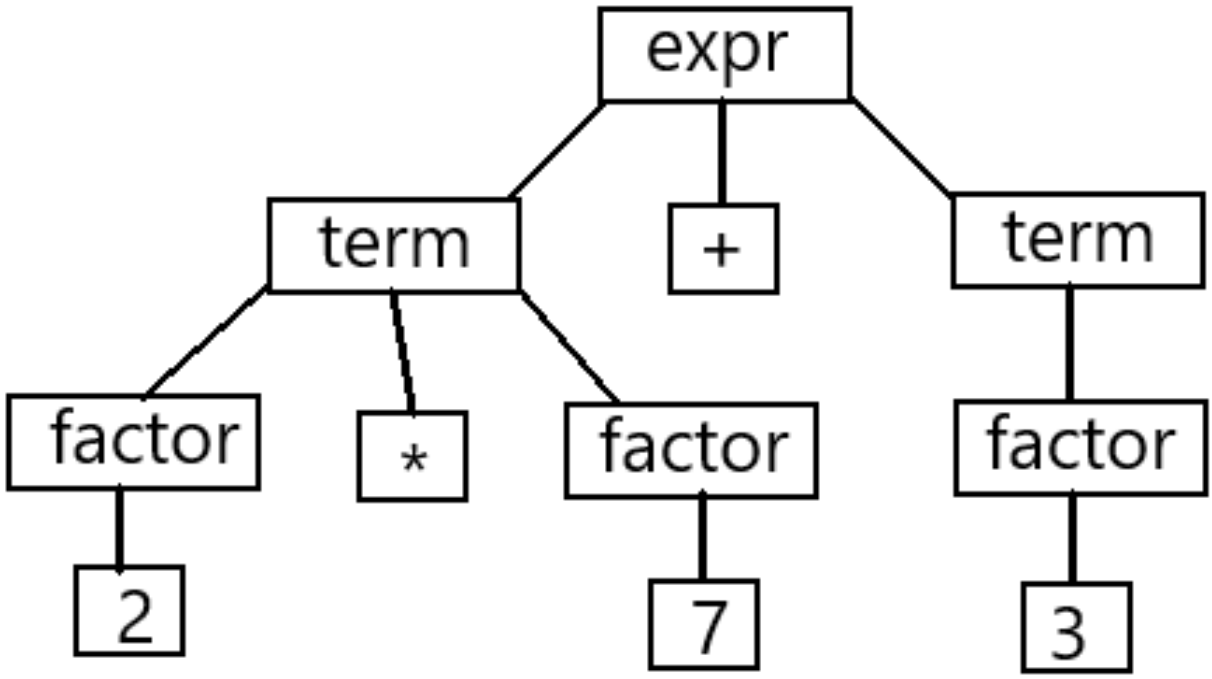

Рисунок 1 — Пример дерева разбора для выражения $2 * 7 + 3$

Зачастую для задач трансляции дерево вывода содержит избыточную информацию. Распространенным средством промежуточного представления между языками в трансляторах выступает дерево абстрактного синтаксиса (AST-дерево). Это дерево, представляющее абстрактную синтаксическую структуру языкового конструкта, где каждому внутреннему узлу(не листу) соответствует оператор, а листьям узла соответствуют операнды при данном операторе. Пример дерева абстрактного синтаксиса для выражения $2 * 7 + 3$ приведен на рисунке 2.

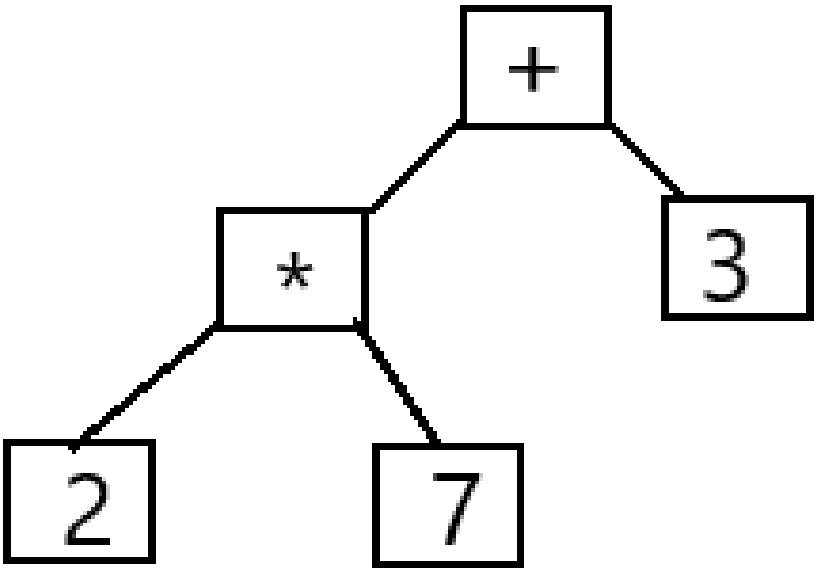

Рисунок 2 — Пример дерева абстрактного синтаксиса для выражения $2 * 7 + 3$

Основные различия между деревом вывода и деревом:

- AST-деревья используют операнды в качестве листьев дерева, а операторы - в качестве внутренних узлов;
- AST-деревья не используют внутренние узлы для представления правил грамматики, в отличие от деревьев разбора;

- AST-деревья не содержат полный вывод синтаксиса(например, отсутствуют узлы для представления скобок и правил);
- AST-деревья гораздо компактнее, в сравнении с деревьями вывода в разрезе совпадающих языковых конструкций [14];

Для приведения дерева вывода к AST-дереву требуется поместить терминальные символы, представляющие операции и команды в качестве корневых узлов соответствующих поддеревьев, оставляя операнды в качестве их детей.

В AST-деревьях, выбор ребенка у некоторого узла произволен, поскольку дерево лишь представляет скелет выражения. Возможность использования одного из этих представлений зависит лишь от адекватности и целостности описания конструкций языка. В книгах по компьютерным наукам зачастую путают деревья вывода и AST-деревья - дерево разбора используется для определений обоих видов деревьев. Дело в том, что деревья вывода представляют собой абстрактные выводы, поскольку одно дерево вывода может быть использовано для получения нескольких различных выводов, например - правый и левый. К тому же, AST-дерево само по себе является деревом вывода, ведь несколько разных выводов могут быть схематически описаны при помощи одного дерева, но иметь неидентичный разбор, например $(a+2)-b/3$ и $a+2-(b/3)$ [15].

1.5 Методы компиляции и трансляции

Языки программирования представляют собой средство описания вычислений для людей и машин. Современный мир зависит от языков программирования, поскольку все программное обеспечение на всех компьютерах мира написано на том или ином языке программирования. Однако, прежде чем запустить программу, её необходимо преобразовать в форму, которая может

выполняться на компьютере.

Программные системы, выполняющие такое преобразование, называются компиляторами. Компилятор - программа, считывающая текст программы, написанной на одном языке - исходном, и транслирует его в эквивалентный текст на другом языке - целевом [16]. Концептуально компилятор работает пофазно, причем в процессе каждой фазы происходит преобразование исходной программы из одного представления в другое. Типичное разбиение компилятора на фазы представлено на рисунке 3. На практике некоторые фазы могут быть сгруппированы вместе и промежуточные представления программы внутри таких групп могут явно не строиться.

Рисунок 3 — Фазы компиляции и трансляции

Лексический анализ представляет собой сканирование потока символов

исходной программы слева направо и их группировку в токены, представляющие собой последовательности символов с определенным совокупным значением.

Синтаксический анализ, также называемый иерархическим анализом или парсингом, представляет собой процесс иерархической группировки символов или токенов в грамматические фразы, используемые компилятором для синтеза вывода, зачастую представляемые в виде дерева. Пример дерева разбора приведен на рисунке 4.

Рисунок 4 — Дерево разбора для выражения position := initial + rate*60

2 Компиляция и трансляция в среде исполнения КуМир

2.1 Трансляция кода в среде программирования КуМир

Изучив исходные коды среды исполнения КуМир [17], была выделена основная архитектура, используемая при кодогенерации. Данный процесс можно разбить на следующие этапы:

1. разработка кода на языке программирования КуМир;
2. генерация AST-дерева по исходному коду;
3. генерация кода по полученному AST-дереву в низкоуровневый язык для исполнения на виртуальной машине;
4. исполнение результата трансляции.

Трансляция кода может быть осуществлена как при помощи среды исполнения, так и вручную, при помощи специальной утилиты, создающейся при сборке исходных кодов программы — kumir2-bc.exe [18].

Основными используемыми объектами при трансляции являются плагины — утилиты, предоставляющие интерфейс для выполнения различных действий — компиляции и анализа исходного кода, генерации файла с исполняемым кодом, а также исполнения транслированного кода на виртуальной низкоуровневой машине. Реализация действий инкапсулируется в плагине, скрывая тело основного используемого блока. Например, для генерации кода по имеющемуся AST-дереву используется KumirCodeGeneratorPlugin. Он содержит методы управления процессом трансляции - начало и остановка, добавление дополнительных параметров, влияющих на процесс. Методы реализующие генерацию кода на промежуточном языке реализованы в сущности Generator, использующейся этим плагином.

Генерация исполняемого файла программы происходит при помощи утилиты kumircompilertool. Компиляция и проверка парных условий — открывающие или закрывающие скобки, ... В теле метода start() происходит

вызов лексического и синтаксического анализаторов для получения лексем программы, затем происходит построение AST-дерева разбора по полученному списку лексем и поиск ошибок компиляции. По полученному дереву формируется исполняемый файл программы — вначале транслируется основная информация о программе — информация об используемом модуле, основной и вспомогательные алгоритмы, затем — константы и внешние зависимости. В конце трансляции, в случае, если установлена опция вывода кода в отдельный файл, создается файл с расширением .kod.txt(транслированный код исходной программы в текстовом представлении) или .kod(транслированный код исходной программы в бинарном представлении).

Процесс компиляции инициализируется созданием сущности анализатора кода — analizer $\rightarrow$ createInstance(). В качестве анализатора исходного кода используется kumiranalizerplugin, состоящий из лексического и синтаксического анализаторов, используемых для поиска ошибок, а также МП-преобразователя для дерева разбора и AST-дерева. Создание сущности анализатора исходного кода происходит при помощи конструктора. В качестве входных параметров, функция-конструктор принимает ссылку на используемую версию плагина, указанную в файле сборки(конфигурационный файл, используемый для создания отдельных сред разработки), а также логическую переменную — используемый режим(ученик или учитель). Первым шагом инициализации AnalizerPlugin является подготовка к работе — создание и настройка сущностей лексического и синтаксического анализаторов совместно с автоматом-преобразователем для работы плагина. Далее происходит загрузка модулей, входящих в стандартную библиотеку языка КуМир — пакетов для взаимодействия с файловой системой, строковыми данными и ряда операторов, реализующих стандартные математические функции — $\sin, \cos$, возведение в степень, нахождение целой части деления, ...

Затем определяется расположение разработанной программы на языке КуМир в файловой системе, а также текст исходной программы разбивается на лексемы и группируется в отдельные выражения при помощи методов setSourceDirName и setSourceText (analizer.cpp).

Токенизация текста происходит построчно при помощи метода splitIntoStatements (lexer.cpp). На входе имеется построчно разделенный текст программы. Каждая строка разбивается на лексемы (метод splitLine IntoLexems в том же файле), затем лексемы группируются посредством их упорядочивания в виде списка, с целью упрощения процесса поиска лексических ошибок.

Полученные выражения обрабатываются для поиска парных — если в обрабатываемой строке встречалась лексема начала цикла, программа лишь проверяла существование закрывающей лексемы, более не связывая их.

Разбив текст на отдельные выражения, анализатор вызывает метод для компиляции — doCompilation, принимающий 2 аргумента: набор составленных выражений, а также этап компиляции. Компиляция происходит в 2 этапа: вначале метод работает в режиме построения скелета программы (выделение основных инструкций в виде дерева — определение переменных и структур), затем происходит наращивание тел выделенных ранее блоков кода.

В зависимости от этапа компиляции, метод doCompilation работает поразному. Шаги работы алгоритма в случае первого прохода:

1. инициализация хранилищ данных о сторонних модулях — unnamedUserModule и unnamedTeacherModule;
2. обнуление счетчика ошибок;
3. группировка выражений по модулям при помощи проверки парности открывающих/закрывающих структур и разделения выражений в соответствии с границами модулей(выражений “начало модуля” или ”конец

модуля"); проверки дополнительного парного "конец модуля" и поиска непарных выражений "начало модуля";

4. выбор рассматриваемого блока из списка найденных;
5. определение начальной и конечной строк модуля;
6. в случае, если исследуемый блок является точкой входа в программу, т.е. список выражений, полученный алгоритмом на входе, начинается и заканчивается на первую и последнюю строчку данного модуля, крайние выражения удаляются;
7. если начало блока совпадает с первым выражением, определяется тип модуля(учитель или ученик). Иначе блок сохраняется в unnamedUserModule или unnamedTeacherModule для соответствующего типа модуля;
8. если блок кода не был очищен, указывается используемый модуль и в случае, если модуль не является сторонним (отдельно подключаемые модули, сохраняющиеся в unnamedUserModule или unnamedTeacherModule) добавляем блок к AST-дереву разбора. Иначе — шаг 9;
9. если список исследуемых выражений не пуст, инициализируется преобразователь и запускаются методы обработки и пост-обработки имеющегося блока и набора содержащихся в нем выражений. Первичная обработка заключается в построении левого разбора входной цепочки. Пост-обработка заключается в сопоставлении типов данных с компилируемыми аналогами;
10. по результату пост-обработки генерируется AST-дерево;
11. повторять шаг 1, пока не закончатся выражения;
12. в момент опустошения списка полученных выражений, зависимости AST-дерева пополняются обнаруженными модулями из хранилищ;
13. инициализация синтаксического анализатора для проверки и удаления неиспользуемых зависимостей;

14. запуск синтаксического анализатора для проверки доступности внешних модулей.

Шаги 1-4 алгоритма приведены в блок-схеме на рисунке 5. Шаги 5-7 алгоритма приведены в блок-схеме на рисунке 6. Шаги 8-14 алгоритма приведены в блок-схеме на рисунке 7.

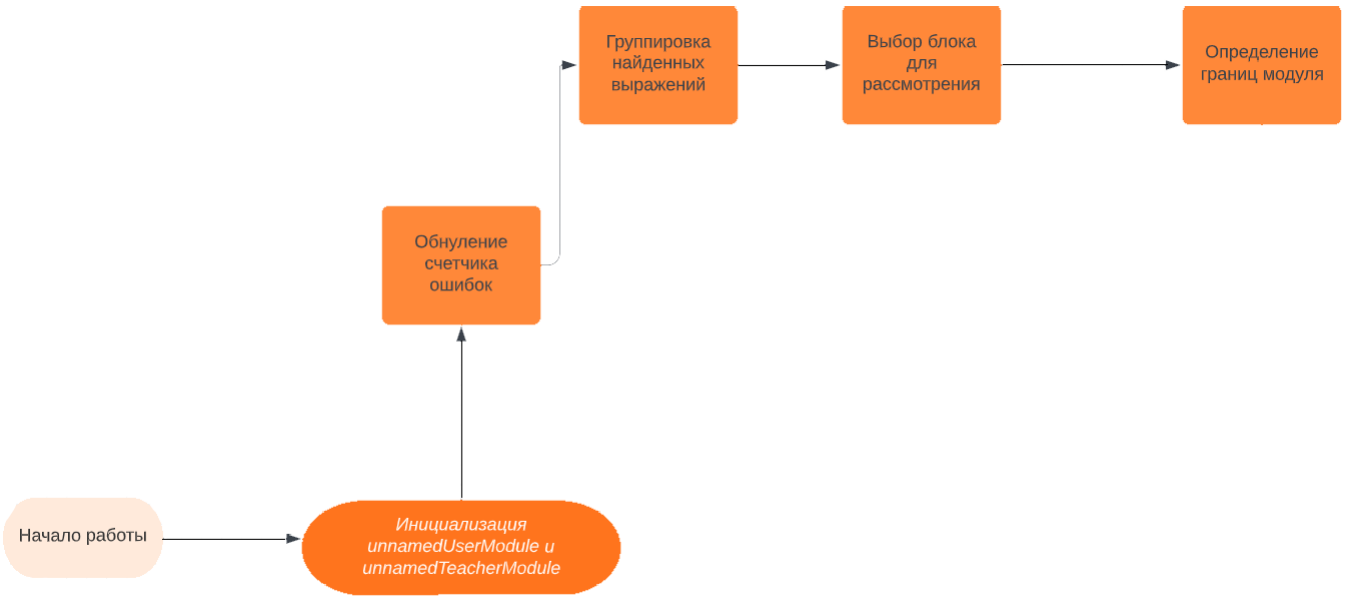

Рисунок 5 — Блок-схема первого прохода работы алгоритма doCompile(шаги 1-4)

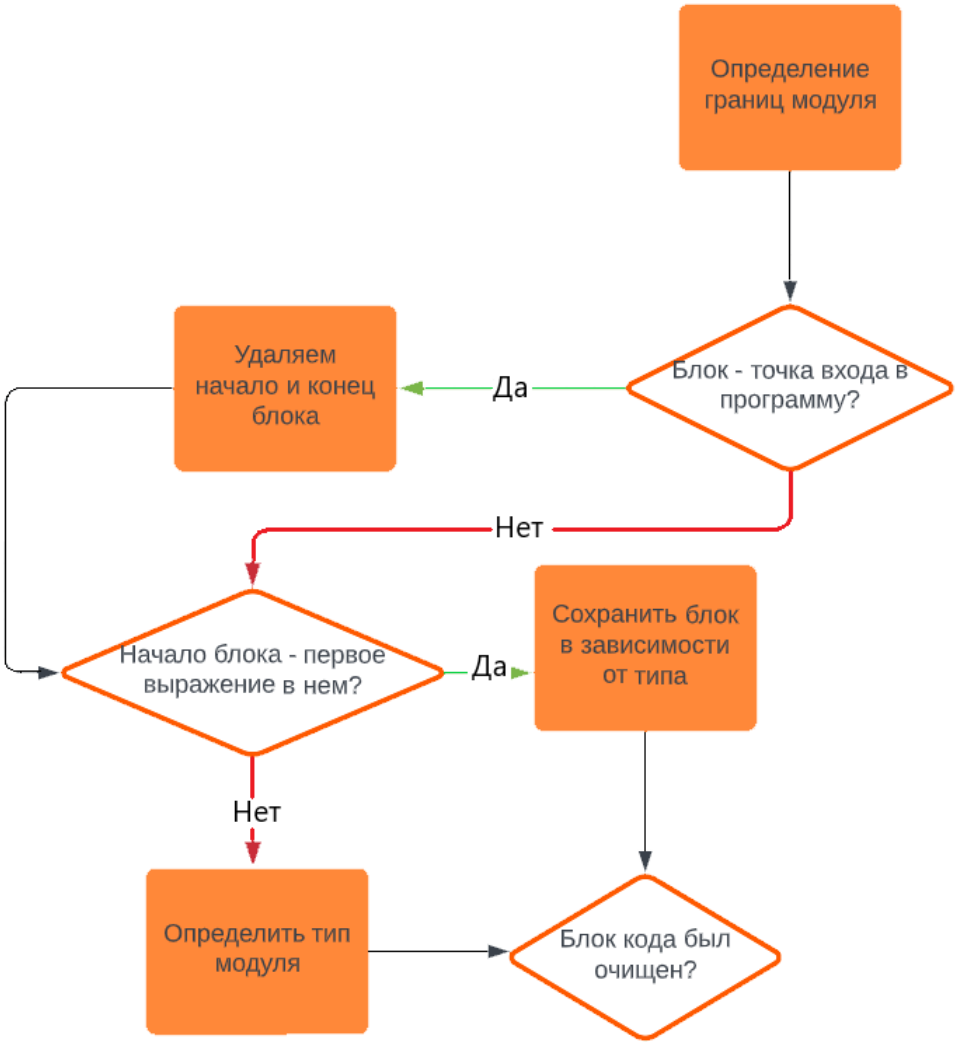

Рисунок 6 — Блок-схема первого прохода работы алгоритма doCompile(шаги 5-7)

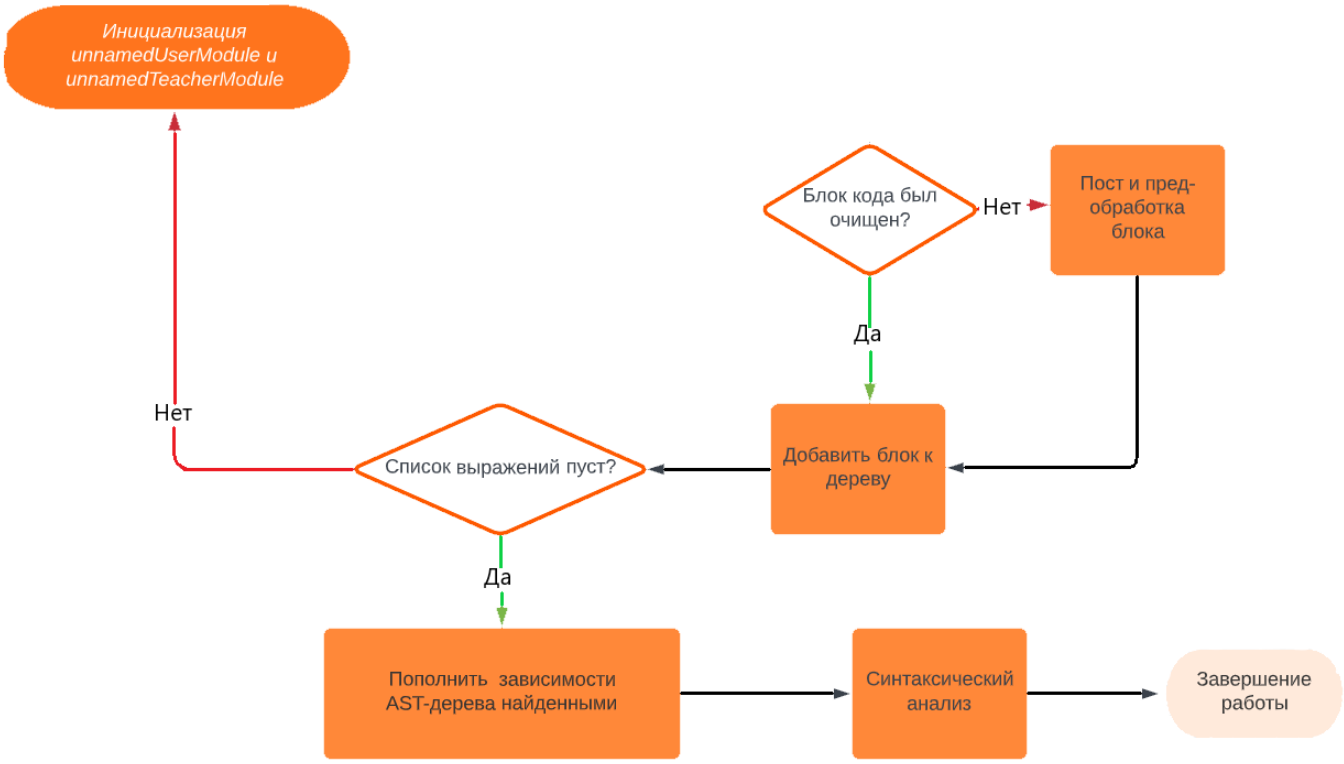

Рисунок 7 — Блок-схема первого прохода работы алгоритма doCompile(шаги 8-14)

Второй этап компиляции заключается в вызове метода processAnalisys у синтаксического анализатора. Метод отвечает за отображение ошибок, полученных в процессе компиляции и передачу их на уровень представления для отображения пользователю. Проверяются все допущенные конструкции в дереве AST-разбора, добавляя сообщение об ошибке в скрытый список errors.

2.2 Реализация AST-дерева

За хранение информации о AST-дереве разбора отвечает соответствующая структура, содержащая определения для различных выражений, модулей и утверждений, типов и переменных (ast_expression, ast_module, ast_statement, ast_type, ast_variable, а также ast_algorithm, содержащий информацию об исходной программе в целом — подключаемые заголовки, тело программы и т.д.)

Сущность ast_variable состоит из трех основных частей — заголовочного файла ast_variabletype.h, определяющего перечисление с информацией о доступных типах данных, а также перечисление, определяющее назначение переменной — тип параметра как аргумента (входной или выходной), тип использования параметра(чтение или запись); заголовочного файла ast_variable.h, определяющего структуру переменной — название, базовый тип, область определения и значение по умолчанию, а также число размерностей для массивов и список ограничений; файл с реализацией ast_variable.cpp, определяющий реализацию данной сущности — функции для создания и обновления копий экземпляров.

Сущность ast_type реализуется при помощи файла — ast_type.h, описывающего структуру абстрактного типа данных — его название, а также ссылка на исполнителя(actor), представляющего собой подключаемый программный модуль, который может быть использован при написании программ.

Сущность ast_statement описывает основные встроенные языковые утверждения — циклы, управляющие метки(break, continue), условные выражения, команды взаимодействия с пользовательским вводом. В случае цикла, утверждение содержит информацию о его типе(LoopFor, LoopWhile, LoopTimes, LoopForever), числе итераций, для цикла типа LoopTimes, условии остановки или пограничных значений итератора, тело цикла и команды завершения цикла. В случае условной конструкции, утверждение содержит информацию о месте вызова, теле утверждения, ошибках, возникших во время компиляции.

Сущность ast_expression описывает выражения — переменные, константы, вызовы функций, а также вложенные выражения. По умолчанию, любое выражение настраивается при помощи флагов, определяющих взаимодей-

ствие выражения с кешем — keepInCache, useFromCache и clearCacheOnFailure. Для проверки наличия скобок в выражении присутствует флаг expressionIsClosed, необходимый для избежания проблем при приведении выражения в конъюктивную нормальную форму. В случае описания вызова функции, в тело выражения передается ссылка на вызываемую функцию. Аналогично для констант и переменных.

Сущность ast_module описывает модуль — комплекс программного кода(библиотеку), состоящий из заголовка(содержит информацию о названии модуля, файле с исходным кодом модуля, списке методов модуля, а также о валидации модуля) и тела(содержит информацию о глобальных переменных и константах, функциях, объявленных перед телом модуля, а также о лексемах определяющих границы реализации модуля).

2.3 Генерация кода по построенному AST-дереву

За генерацию целевого кода для исполнения отвечает плагин kumirCodeGenerator. Для построения набора инструкций по исходной программе используется сущность Generator, инкапсулирующая логику трансляции. В начале процесса кодогенерации (функция-конструктор) создается экземпляр класса-генератора, затем в методе initialize происходит первичная подготовка процесса — проверяется наличие точек останова и тип выходного файла(текст, содержащий список инструкций или байт-код, напрямую используемый виртуальной машиной). Затем определяется режим работы кодогенератора и необходимых зависимостей, глобальных переменных и алгоритмов.

Метод addKumirModule отвечает за наращивание инструкций по готовому дереву разбора. В начале определяется бинарный поток данных, в который заносятся все данные по глобальным переменным. Далее определя-

ется точка входа в программу и, происходит трансляция дерева инструкций основного алгоритма. Метод instructions получает список выражений, полученных в ходе разборов и наращивает код в зависимости от типа инструкции. Существует 12 групп команд, формируемых AST-деревом: ошибки, операции присваивания, операции тестирования существующего кода, инициализации переменных, ввод/вывод данных, операции ветвления и цикла, а также операции приостановки исполнения.

При трансляции описанных выше структур используется метод calculate, отвечающий за корректность трансляции. Метод определяет тип переменной или функции, входные или выходные параметры и заносит эти данные в поток.

Выявление ошибок в коде существующей программы осуществляется при помощи выгрузки ошибок с фазы лексического/синтаксического анализов при помощи плагина kumirAnalizer.

Процесс трансляции зависимостей определен в двух методах класса Generator — generateConstantTable и generateExternTable. Первый метод осуществляет трансляцию констант и сохранение всех метаданных — размерность, тип и количество данных. Второй метод отвечает за трансляцию зависимостей как явно, так и неявно импортируемых. Тип зависимости зависит от факта принадлежности модуля к составу стандартной библиотеки КуМир.

Для более глубокого понимания работы кодогенератора, были проанализированы плагины для исполнения кода.

2.4 Исполнение кода в среде КуМир

За выполнение кода программы на языке КуМир отвечает runplugin. Среда исполнения позволяет исполнять код в нескольких режимах — существует отдельный класс Run, определяющий объект сущности исполнения.

Данный подход к построению архитектуры предоставляет ряд способов запуска кода на исполнение помимо стандартного — пошаговый, без отладки, исполнение до точки останова, исполнение, начиная с точки останова, без параметров. Каждый режим добавляет аргументы, влияющие на ход исполнения целевого кода. Наличие аргументов изменяет режим работы виртуальной машины — меняется контекст исполнения программы на низком уровне.

2.5 Исполнение кода на виртуальной машине

Альтернативой стандартному способу исполнению кода на персональном компьютере, является вариант исполнения кода программы при помощи низкоуровневой виртуальной машины. Для этого используется программный интерфейс низкоуровневой системы компилятора.

Используемый интерфейс имеет ряд преимуществ над стандартным компилятором языка C++, а именно:

- больший набор поддерживаемых архитектур, включающий все современные, среди которых: X86, X64, ARM, ...
- встроенный JIT-компилятор предоставляющий возможность компилирования отдельных участков кода, только в случае, их использования;
- оптимизированный сборщик мусора;
- возможность переиспользования, за счет поддержки и развития библиотеки [19].

Программа, написанная школьником в среде, транслируется на низкоуровневый кроссплатформенный язык. Набор команд виртуальной машины приведен в таблице 1.

Набор инструкций виртуальной машины состоит из следующих кодов:

- 0x0C-0x13 - функции взаимодействия с переменными в памяти - за-

грузка и выгрузка, добавление, подготовка переменных к выводу;

- 0x14-0x16 отвечают за условные и безусловные переходы;
- 0x18, 0x19 - извлечение и запись значений из стека;
- 0x1B - возврат из подпрограммы в точку вызова;
- 0x1D - ожидание ввода со стороны пользователя;
- 0x1E - ошибка исполнения;
- 0x1F - определение границ инструкции в исходном коде на языке КуМир;
- 0x20, 0x21 - получение ссылки на адрес в памяти переменной/массива;
- 0x22 - вывод информации о данных в регистрах;
- 0x23 - удаление отступов;
- 0x24 - установка ссылки на адрес в памяти для хранения значения переменной;
- 0x26 - остановка выполнения;
- 0x27 - ручной воздействие на виртуальную среду исполнения;
- 0x28 - извлечение 4 значений из верхушки стека a, b, c, x для сравнения данных следующим образом: если c >= 0, то a<x<=b, иначе a<=x<b;
- 0x29 - обновление данных о границах массива;
- 0x30 - кэширование значения из головы стека;
- 0x31 - извлечение значение из кэша и помещение его на верх стека;
- 0x32 - сброс кэшированного значения в случае наличия нуля в указанном регистре;
- 0x33 - установка начала области памяти для кэширования значений;
- 0x34 - установка конца области памяти для кэширования значений;
- 0xF1-0xFE используются для реализации операторов сравнения, ариф-

метических и логических операций;

Программа на виртуальной машине исполняется в заранее определенном контексте исполнения. Он определяет используемые регистры и их начальное значение, адреса расположения исполняемой программы, а также контекст модуля — зависимости, ссылки и глобальные переменные, определенные в начале программы. Исполнение происходит, пока в стеке команд виртуальной машины имеется хотя бы 1 команда. Перед выполнением действия, описанного в теле программы, происходит проверка инструкции — находится ли она на вершине стека (значение регистра IP). Далее команда извлекается из стека и исполняется в зависимости от типа. В случае, если тип команды не определен, происходит переход к следующей команде. Для трансляции кода на язык низкоуровневого устройства существуют соответствующие сущности — типы данных, функции, модули, а также плагин с отдельной сущностью генератора кода. Виртуальная машина позволяет работать как в режиме выпуска приложения, так и в отладочном режиме, добавляя возможности пошагового исполнения и определения точек останова.

2.6 Модуль для взаимодействия с аппаратно-программным комплексом Arduino

Программный код для языка программирования КуМир группируется в библиотеки для возможности переиспользования, поддержки и распространения. Одной из таких библиотек является разработанный модуль, предоставляющий набор основных команд, используемых при программировании роботов на базе аппаратной платформы Arduino.

Разработанный модуль определяется в виде использования исполнителя в среде программирования КуМир.

Исполнитель системы «Кумир 2.0» представляет собой подключаемый

Таблица 1 – Набор инструкций низкоуровневой виртуальной машины

Инструкция	Код операции
NOP	0x00
CALL	0x0A
INIT	0x0C
SETARR	0x0D
STORE	0x0E
STOREARR	0x0F
LOAD	0x10
LOADARR	0x11
SETMON	0x12
UNSETMON	0x13
JUMP	0x14
JNZ	0x15
JZ	0x16
POP	0x18
PUSH	0x19
RET	0x1B
PAUSE	0x1D
ERRORR	0x1E
LINE	0x1F
REF	0x20
REFARR	0x21
SHOWREG	0x22
CLEARMARG	0x23
SETREF	0x24
CTL	0x27
INRANGE	0x28
UPDARR	0x29
CSTORE	0x30
CLOAD	0x31
CDROPZ	0x32
CACHEBEGIN	0x33
CACHEEND	0x34
SUM	0xF1
SUB	0xF2
MUL	0xF3
DIV	0xF4
POW	0xF5
NEG	0xF6
AND	0xF7
OR	0xF8
EQ	0xF9
NEQ	0xFA
LS	0xFB
GT	0xFC
LEQ	0xFD
GEQ	0xFE

программный модуль, который может быть использован при написании программ как на языке Кумир, так и на других языках программирования, поддерживаемых системой.

Существует следующие типы реализации исполнителей:

1. реализация исполнителя на целевом языке программирования (Кумир, Python и т. д.);
2. реализация исполнителя с использованием технологий QML/JavaScript;
3. реализация исполнителя на базе WebKit с использованием технологий HTML 5;
4. реализация исполнителя на языке C++ с использованием средств Qt.

Для реализации исполнителей на языке C++ в системе Кумир 2.0 был создан инструмент, позволяющий создать единый шаблон разработки исполнителей. Преимущества, предоставляемые данной разработкой:

1. обеспечение единого программного кода исполнителя, назависящего от изменений в программном интерфейсе системы Кумир 2x;
2. реализация только функциональной части исполнителя, абстрагированной от деталей взаимодействия с системой Кумир;
3. возможность создания программных интерфейсов исполнителя для различных языков программирования на основе декларативного описания команд исполнителя.

Создание нового исполнителя на языке C++ системы Кумир 2.x состоит из следующих этапов:

1. декларативное описание исполнителя: его системы команд и, при необходимости, пользовательского интерфейса;
2. создание на основе декларативного описания проекта на языке C++, содержащего заготовки файлов, необходимых для реализации;
3. реализация функциональности исполнителя.

Под декларативным описанием подразумевается файл в формате JSON, который является частью проекта. Каждый исполнитель представляется отдельным проектом при компиляции исходных кодов.

Для создания проекта на основе декларативного описания, необходимо использовать скрипт генерации кода исполнителя при помощи команды — gen_actor_source.py, в качестве аргументов указываются флаг –project и имя файла с декларативным описанием исполнителя в формате JSON.

После этого будут созданы файлы: CMakeLists.txt –– файл проекта CMake, имя_исполнителямodule.cpp и имя_исполнителямodule.h – файлы заготовок для реализации функциональности исполнителя. Данные файлы создаются только один раз при создании проекта и впоследствии не перезаписываются автоматически.

Замечание: на этапе генерации исходных текстов выполняется только синтаксический, но не семантический анализ на возможные ошибки. Таким образом, если результатом работы скрипта gen_actor_source.py является заведомо некомпилируемый текст, это означает, что исходный файл описания содержит ошибку.

Разработанный модуль приведен в приложении А.

2.7 Транслятор с языка КуМир в язык C++

Сложность разработки 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 библио-

тек, 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, ...; плагинов, а также исполнителей и сред разработки, состоящих из файлов с описанием использующихся методов, процедур и сущностей и файлов с описанием слоя представления 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.

Изначально, для сборки исходных кодов представления проекта использовалась 4 версия фреймворка QT. Разработчики добавили возможность сборки проекта с использованием 5 версии фреймворка для повышения поддерживаемости и актуальности библиотек. Как оказалось, добавление возможности сборки проекта с использованием более современной версии фреймворка QT повлекло за собой негативные последствия, затрудняющие разработку — изучив ряд современных сред разработки для работы с используемыми в проекте библиотеками, оказалось, что ни одна из них не способна скомпилировать проект. Из-за объема различий фреймворка QT, отладка проекта в момент исполнения в привычном понимании недоступна — возможность отслеживания изменений при помощи точек останова отсутствует. Единственный способ отладки приложения в момент исполнения — вывод данных в формате сообщений об ошибках в момент исполнения программы.

Ещё одной сложностью, повышающей трудоемкость задачи является отсутствие документации к исходным кодам системы программирования КуМир. Названия методов и сущностей не всегда позволяют понять предназна-

чение программного кода. Зачастую в программном коде встречаются некорректно именованные блоки кода, не позволяющие понять их роль в программе. Иногда встречаются комментарии, поясняющие процесс выполнения кода программы, но их объем не покрывает и доли процента программного кода проекта, что снижает доступность кода.

Разработка плагина для трансляции 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 2 – 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, а также оператор 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 низкоуровнего 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 по-

вторяется 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 условие‘ 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 заносился 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.

Программный код разработанного транслятора приведен в приложении В.

ЗАКЛЮЧЕНИЕ

В ходе работы были изучены и проанализированы исходные кода среды программирования КуМир, был разработан транслятор с языка программирования КуМир в язык C++, а также модуль для взаимодействия с программной платформой Arduino при помощи языка программирования КуМир, а также решен ряд поставленных задач:

- были изучены существующие алгоритмы и методы трансляции;
- был изучен набор основных команд платформы Arduino;
- был исследован набор основных команд и архитектура программ в среде программирования КуМир;
- были проанализированы и переработаны исходные коды языка программирования КуМир;
- был разработан модуль для предоставления набора основных команд платформы Arduino;
- был разработан транслятор с языка программирования КуМир в язык C++.

На данный момент планируется расширить возможности разработанного транслятора до отдельного приложения для персональных компьютеров, а также добавить возможность прошивки робота из клиента, инструмент выбора порта с подключенным роботом для прошивки с настраиваемым алгоритмом прошивки, определяющим роль транслированной программы в архитектуре программы для прошивки робота.

С учетом ряда реализованных задач, поставленную в начале работы цель можно считать достигнутой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Сармантаева Л. С. Исследование технологий обучения программированию в школе // Проблемы и перспективы развития образования в России. 2012. №16. [Электронный ресурс] URL: https://cyberleninka.ru/article/n/issledovanie-tehnologiy-obucheniya-programmirovaniyu-v-shkole (дата обращения: 08.05.2022).

2 Челнокова Елена Александровна, Хижная Анна Владимировна, Казначеев Дмитрий Александрович РОБОТОТЕХНИКА В ОБРАЗОВАТЕЛЬНОЙ ПРАКТИКЕ ШКОЛЫ // Проблемы современного педагогического образования. 2019. №65-1. [Электронный ресурс] URL: https://cyberleninka.ru/article/n/robototehnika-v-obrazovatelnoy-praktike-shkoly (дата обращения: 26.12.2021).

3 Сенюшкин Н.С. Изучение робототехники в школе - путь интеграции в инженерное образование // Актуальные проблемы авиации и космонавтики. 2014. №10. [Электронный ресурс] URL: https://cyberleninka.ru/article/n/izuchenie-robototehniki-v-shkole-put-integratsii-v-inzhenernoe-obrazovanie (дата обращения: 26.12.2021).

4 Брянцева Р.Ф. Занимательная робототехника в современной школе // Наука и перспективы. 2018. №1. [Электронный ресурс] URL: https://cyberleninka.ru/article/n/zanimatelnaya-robototehnika-v-sovremennoy-shkole (дата обращения: 26.12.2021).

5 Образовательная робототехника: дайджест актуальных материалов / ГАОУ ДПО «Институт развития образования Свердловской области»; Библиотечно-информационный центр; сост. Т. Г. Попова. – Екатеринбург: ГАОУ ДПО СО «ИРО», 2015. – 70 с. [Электронный ресурс]. URL: http://cmitsuperlab.ru/assets/upload/files/19-dajdzhest-

aktualnyix-materialov-poobrazovatelnoj-robototexnike.pdf (дата обращения: 26.12.2021).

6 Статья о предназначении комплектов Arduino [Электронный ресурс] URL: http://arduino.ru/ (дата обращения: 26.12.2021).

7 Оффициальный сайт КуМир [Электронный ресурс] URL: https://www.niisi.ru/kumir/ (дата обращения: 26.12.2021).

8 Статья о КуМире на электронном образовательном портале Фоксфорд [Электронный ресурс] URL: https://foxford.ru/wiki/informatika/sreda-programmirovaniya-kumir (дата обращения: 26.12.2021).

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.

11 Леонов А. Г. Тенденции объектно-ориентированного программирования в разработке системы КуМир // Программные продукты и системы. 2012. №4. [Электронный ресурс] URL: URL: https://cyberleninka.ru/article/n/tendentsii-obektno-orientirovannogo-programmirovaniya-v-razrabotke-sistemy-kumir (дата обращения: 03.01.2022).

12 Оффициальный сайт платформы Arduino [Электронный ресурс] URL: https://www.arduino.cc/en/Guide/Introduction (дата обращения: 26.12.2021).

13 Серёгин М.С. Использование платформы arduino в образовательной деятельности // Инновационная наука. 2019. №6. [Электронный ресурс] https://cyberleninka.ru/article/n/ispolzovanie-platformy-arduino-v-obrazovatelnoy-deyatelnosti (дата обращения: 02.01.2022).

14 Сорокин Алексей Николаевич ОСОБЕННОСТИ СОСТАВЛЕНИЯ ЛАБОРАТОРНЫХ РАБОТ ПО ИЗУЧЕНИЮ ЭЛЕКТРИЧЕСКИХ ПРОГРАММИРУМЫХ УСТРОЙСТВ НА ПЛАТФОРМЕ ARDUINO // БГЖ. 2019. №4 (29). [Электронный ресурс] https://cyberleninka.ru/article/n/osobennosti-sostavleniya-laboratornyh-rabot-po-izucheniyu-elektricheskih-programmirumyh-ustroystv-na-platforme-arduino (дата обращения: 02.01.2022).

15 Let's Build A Simple Interpreter. Part 7: Abstract Syntax Trees [Электронный ресурс] URL: https://ruslanspivak.com/lsbasi-part7/ (дата обращения: 26.12.2021).

16 Ахо А. Теория синтаксического анализа, перевода и компиляции. Синтаксический анализ: Том 1 / Ахо А., Ульман Дж. – М.: Книга по Требованию, 2012. – 613 с

17 Library of Congress Cataloging-in-Publication Data Slonneger, Kenneth. Formal syntax and semantics of programming languages: a laboratory based approach / Kenneth Slonneger, Barry L. Kurtz. p.cm. Includes bibliographical references and index. ISBN 0-201-65697-3 1.Programming languages (Electronic computers)–Syntax. 2.Programming languages (Electronic computers)–Semantics. I. Kurtz, Barry L. II. Title. QA76.7.S59 1995 005.13'1–dc20

18 Ахо, Альфред В., Лам, Моника С., Сети, Рави, Ульман, Джеффри Д. К63 Компиляторы: принципы, технологии и инструментарий, 2-е изд. :Пер. с

англ. - М.: ООО “И.Д. Вильямс”, 2008. — 1184 с. : ил. — Парал. тит. англ.

19 Репозиторий с исходным кодом среды исполнения КуМир [Электронный ресурс] URL: https://github.com/a-a-maly/kumir2 (дата обращения: 26.12.2021)

20 Система программирования Кумир 2.х А.Г.Кушниренко , М. А. Ройтберг , Д.В.Хачко, В. В. Яковлев [Электронный ресурс] URL:$http : //roytberg.lpm.org.ru/pdfs/kumir2x_2015.pdf$ (дата обращения: 26.12.2021)

21 LLVM API [Электронный ресурс] URL: $https : //llvm.org/docs$ (дата обращения: 26.12.2021)

ПРИЛОЖЕНИЕ А

Декларативное описание исполнителя Arduino

```
{
        "name": { "ascii": "Arduino", "ru_RU": "Ардуино" },
        "dynamic": { "methods": true },
        "methods": [
                {
                        "name": {"ascii": "digitalRead", "ru_RU": "цифрВвод" },
                        "async": false,
                        "returnType": "bool",
                        "arguments": [
                                { "name": "pin", "baseType": "int"}
                        ]
                },
                {
                        "name": {"ascii": "digitalWrite", "ru_RU": "цифрВывод" },
                        "async": false,
                        "arguments": [
                                { "name": "pin", "baseType": "int"},
                                { "name": "value", "baseType": "bool"}
                        ]
                },
                {
                        "name": {"ascii": "analogRead", "ru_RU": "аналогВвод" },
                        "async": false,
                        "returnType": "int",
                        "arguments": [
                                { "name": "pin", "baseType": "int"}
                        ]
                },
                {
                        "name": {"ascii": "analogWrite", "ru_RU": "аналогВывод" },
```

```
31                      "async": false,
32                      "arguments": [
33                              { "name": "pin", "baseType": "int"},
34                              { "name": "value", "baseType": "int"}
35                      ]
36              },
37              {
38                      "name": {"ascii": "delay", "ru_RU": "задержкаС" },
39                      "async": false,
40                      "arguments": [
41                              { "name": "ms", "baseType": "int"}
42                      ]
43              },
44              {
45                      "name": {"ascii": "milis", "ru_RU": "задержкаМС" },
46                      "async": false,
47                      "returnType": "int"
48              },
49              {
50                      "name": {"ascii": "Serial.begin", "ru_RU": "открытьПорт" },
51                      "async": false,
52                      "arguments": [
53                              { "name": "rate", "baseType": "int"}
54                      ]
55              },
56              {
57                      "name": {"ascii": "Serial.println", "ru_RU": "выводВПорт" },
58                      "async": false,
59                      "arguments": [
60                              { "name": "data", "baseType": "int"}
61                      ]
62              },
63              {
```

```
64                          "name": {"ascii": "INPUT", "ru_RU": "режимВвод" },
65                          "async": false,
66                          "returnType": "int",
67              "arguments": []
68                  },
69                  {
70                          "name": {"ascii": "OUTPUT", "ru_RU": "режимВывод" },
71                          "async": false,
72                          "returnType": "int",
73                          "arguments": []
74                  },
75                  {
76                          "name": {"ascii": "HIGH", "ru_RU": "высокийСигнал" },
77                          "async": false,
78                          "returnType": "int",
79                          "arguments": []
80                  },
81                  {
82                          "name": {"ascii": "LOW", "ru_RU": "низкийСигнал" },
83                          "async": false,
84                          "returnType": "int",
85                          "arguments": []
86                  }
87          ]
88  }
```

ПРИЛОЖЕНИЕ Б

Примеры работы кодогенератора

Пример 1. Исходная программа на языке программирования КуМир:

```
алг
нач
цел a = 6;
нц a раз
вывод 1;
кц
кон
```

Результат кодогенерации:

```
const int constName1 = 1;
const int a = 6;
void main(){
        while(a > 0){
                Serial.println(constName1);
        }
}
```

Пример 2. Исходная программа на языке программирования КуМир:

```
алг
нач
цел a = 3;
нц для i от 1 до 5
a = a+i;
кц
кон
```

Результат кодогенерации:

```
1 const int a = 6;
2 void main(){
3         for(int i = 1; i < 5; i++){
4                 a += i;
5         }
6 }
```

ПРИЛОЖЕНИЕ В

Программный код приложения

Программный код приложения приведен в git-репозитории: https://github.com/CaMoCBaJL/kumir2/tree/ArduinoFixes

Бакалаврская работа “Проектирование и реализация транслятора языка программирования Кумир в язык C++“ выполнена мною самостоятельно, и на все источники, имеющиеся в работе, даны соответствующие ссылки.

подпись, дата	инициалы, фамилия