СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью преддипломной практики является проведения ряда аналитических работа применительно к теме ВКРБ «компилятор стекового языка программирования». В процессе работы выполняются аналитический обзор существующих стековых языков программирования, сравнительный анализ стековых языков, анализ средств вывода типов стековых языков программирования, анализ устройства байткода и модели исполнения WebAssembly.

1 Аналитический обзор существующих стековых языков программирования

Начиная с семидесятых годов прошлого века было создано достаточно большое количество стековых языков. Далее рассмотрены несколько значимых языков, каждый из которых привнес важные нововведения в области стековых языков (речь о которых пойдет далее). Для языков рассматриваются аспекты синтаксиса, типизации, работы с памятью и стандартной библиотекой, особенности каждого языка. В обзор не включены, однако также заслуживают внимания:

– конкатенативный язык программирования Kitten, который продолжает идеи языка Cat в статическом анализе и представляет способ явного указания типов;

– .NET CIL и байткод Java, которые также являются языками под стековые виртуальные машины (.NET CLR и JVM соответственно).

Forth – первый известный ранний стековых конкатенативных языков, созданный Чарльзом Муром в начале 1970-х как расширяемый язык, для которого ключевые слова можно добавлять прямо во время работы системы, и таким образом сформировать собственное подмножество языка под конкретную предметную область. Синтаксис основан на обратной польской нотации или постфиксной нотации: программа – это последовательность слов, разделённых пробелами. Интерпретатор идет слева направо по программе и ищет лексемы в словаре доступных слов и либо выполняет их, либо трактует как число и кладёт на стек. Язык и среда исполнения едины: система работает в режимах интерпретации и компиляции, переключаясь между ними. Для добавления нового слова или отложенных вычислений используется режим компиляции, для исполнения – режим интерпретации. Стандартной библиотеки в привычном смысле нет – функциональность наращивается подключаемыми наборами слов (word sets), зависящими от конкретной реализации; при этом ANS Forth (стандарт языка) задаёт базовый набор слов «core word set» со стековыми операциями, арифметикой, сравнениями, управлением потоком и примитивами памяти. Система типов у

Forth фактически «бестиповая»: данные представлены машинными словами фиксированной разрядности, а корректность трактовки содержимого стека контролирует программист, что повышает гибкость и предсказуемость времени выполнения, но усложняет статический анализ и проверку ошибок. Для Forth предложена оптимизация занимаемого программой пространства техникой «нитевого кода»: программа по сути является последовательностью безусловных переходов, каждый из которых ведет на инструкции, выполняющие ту или иную команду. Работа с памятью осуществляется через явные операции чтения и записи по адресу, что дает низкоуровневое управление. Область применения Forth – прошивки, встроенные и ресурсно-ограниченные системы, системы реального времени и космическая техника.

На основе идей Forth в 2001 году Манфред фон Тун представил язык Joy как попытку формализации логики стековых языков: каждая команда в Joy обозначает унарную функцию вида «stack -> stack»: на вход подаётся стек данных (состояние программы), в процессе применения команды происходит его модификация и передача следующей команде, значения и подпрограммы при этом также передаются через стек. Программу на Joy можно рассматривать как математическую функцию «stack -> stack» без побочных эффектов [1]. Язык является динамически типизированным, за проверку типов несет ответственность программист, однако ему в этом могут помогать типы, описанные явно при объявлениях функций. Синтаксис Joy также основан на постфиксной записи [2]. Фон Тун вводит идею комбинаторов – функций высшего порядка, которые принимают другие функции (в терминах стековых языков их принято называть цитатами) и образуют некий алгоритм обработки данных (например, комбинатор линейной рекурсии «primrec» принимает цитату тела рекурсии и цитату условия рекурсии). В стандартной поставке присутствует несколько библиотек, сгруппированных по областям применения: библиотеки «agglib» и «seqlib» содержат обобщённые операции над агрегатами (неупорядоченными множествами, строками и списками) и последовательностями, «numlib» включает числовые функции и численные методы, а «mtrlib» предназначена для работы с матрицами [3].

В 2006 году Кристофер Диггинс опубликовал несколько технических отчетов, в рамках которых описывает алгоритм статической типизации для стековых языков, в качестве примера используя свой экспериментальный языка Cat. Cat наследует идею Joy: любая программа рассматривается как преобразование вида «stack -> stack», а основная операция – композиция таких преобразований, реализуемая простой конкатенацией лексем в исходном тексте программы. В плане синтаксиса Cat аналогичен предыдущим и использует обратную польскую запись. Грамматика показана на листинге 1.

Листинг 1 — РБНФ грамматики Cat

```
program         ::= { term } ;
term            ::= word | literal | quote | definition ;

definition      ::= "define" name program ";" ;     (*
глобальное и единственное определение *)
quote           ::= "[" program "]" ;               (* цитата/
замыкание *)

literal         ::= number ;
word            ::= name ;
name            ::= letter { letter | digit | "_" | "-" } ;
number          ::= ["+"|"-"] digit { digit } ;
```

Статическая типизация в контексте стековых языков подразумевает, что в момент компиляции возможно построить конфигурацию стека (его размер и «форма» – типы значений, которые на нем лежат) для произвольного момента времени исполнения. Таким образом вывод типов открывает путь к статическому анализу и проведению оптимизаций. Подробности статического вывода типов описываются в разделе 3. В языке присутствует несколько встроенных команд, их описание приведено в таблице 1. Помимо них в язык встроены арифметические, логические, команды и команды сравнения чисел. Язык использует три типа данных – числа, булевы значения и цитаты.

Таблица 1 — Встроенные команды Cat

Команда	Описание
apply	Применить цитату с вершины стека к остальному стеку.
quote	Захватить вершину стека в цитату.

Продолжение таблицы 1

Команда	Описание
compose	Выполнить композицию двух цитат и положить на стек цитату-результат композиции.
dup	Положить на стек дубль вершины стека.
pop	Удалить значение с вершины стека.
swap	Поменять значение на вершине стека с значением под ним.
cond	Комбинатор ветвления, выбирает элемент на стеке или элемент под ним в зависимости от условия.
while	Комбинатор для цикла «пока».

Еще одним ответвлением от Joy стал язык Factor, впервые опубликованный в 2003 году Славой Пестовым. Преемственность Factor прослеживает по нескольким деталям: синтаксис на основе обратной польской записи, аналогичная идея использования и способ создания новых комбинаторов. Ключевая особенность – это первый прикладной высокоуровневый стековый язык общего назначения. Так его можно назвать благодаря богатой стандартной библиотеке (такая включает в себя, например, веб-фреймворк и парсер XML) и инструментарию разработки, собранном в одной IDE и включающим в себя компилятор с развитой системой анализов, интерактивный отладчик, браузер документации, инспектор объектов. Как и Joy, язык динамический, однако существует важное отличие: при объявлении новой функции программист должен указать ее тип функции, в свою очередь компилятор проверяет этот тип на соответствие содержимому функции. Внутренний механизм этой проверки описывается как абстрактная интерпретация программы: компилятор (а точнее, модуль «Stack-checker» внутри него) симулирует эффекты команды на абстрактном стеке, при встрече ветвлений анализирует обе ветви и унифицирует состояния, а несовместимость размера или формы стека превращает в ошибку времени компиляции. На основе анализа типов компилятор проводит высокоуровневые и низкоуровневые оптимизации. Также дополнительно используются динамические проверки

времени исполнения. Все эти особенности помогаю языку Factor быть вполне применимым для решения задач прикладного программирования.

Наконец стоит рассмотреть спецификацию для одноименных стекового языка и виртуальной машины WebAssembly (сокращенно Wasm), которые были разработаны и выпущены в 2017 году группой компаний, в которую входят W3C, Mozilla, Google, Microsoft и Apple. Wasm определяет низкоуровневый переносимый байткод и архитектуру и семантику набора команд под виртуальную стековую машину, запускаемую преимущественно (но не только) в веб-браузерах наравне с движками ECMAScript. В исходных целях проектирования Wasm фиксируются требования к компактности двоичного представления, быстрой однопроходной валидации и компиляции, а также к «песочнице», пригодной для запуска недоверенного кода с производительностью низкоуровневого кода. Как стековый язык Wasm использует неявный стек данных: инструкции потребляют значения с вершины стека и помещают результаты обратно, при этом реализация не обязана хранить «настоящий» стек – спецификация описывает интерпретацию стека как набора анонимных регистров. Это первый из представленных в обзоре языков, который является целью компиляции, а не сам компилируется во что-либо. Поддержка компиляции в Wasm имеется для множества высокоуровневых языков, в первую очередь для полностью компилируемых языков C/C++/ Rust. Для языка существует два представления: текстовое (формат «Wat» с синтаксисом на S-выражениях), удобное для чтения человеком, и бинарное, понимаемое виртуальной машиной.

2 Сравнительный анализ стековых языков программирования

2.1 Сравнение подходов к построению синтаксиса

Таблица с сравнением языков по синтаксису представлена в таблице 2.

Таблица 2 — Сравнение синтаксиса языков (сводная таблица)

Язык	Форма записи	Особенности синтаксиса
Forth	Постфиксная	Управляющие конструкции реализуются как слова времени компиляции.
Joy	Постфиксная	Цитаты используются как данные, управление через комбинаторы («i», «ifte», «primrec/linrec/binrec»).
Cat	Постфиксная	Синтаксис близок к Joy.
Factor	Постфиксная	Обязательное указание эффектов над стеком при объявлении функций.
Wasm	S-выражения	Нетрадиционная форма записи синтаксических конструкций

В общем случае в рассмотренных языках прослеживается следующая закономерность. Для языков, которые используются непосредственно программистами для написания программ вручную, применяется характерный конкатенативный синтаксис, что соответствует изначально заложенным в языки данного множества концепциям минималистичности и простоты. Вероятно, некоторую роль в сохранении такой формы играет фактор историчности: исторически первый стековый язык (Forth) обладал конкатенативным синтаксисом.

В то же время синтаксис языков, которые являются целями трансляции, представлен более широким разнообразием. В частности Wasm использует S-выражения – тоже одна из достаточно распространных форм синтаксиса для функциональных языков. S-выражения также соответствуют концепции простоты (РБНФ синтаксиса таких языков имеет относительно малый размер).

Таким образом, отличительная черта синтаксиса стековых языков – минималистичный дизайн, который упрощает как механизмы разбора

программ за счет упрощенного парсера, так и работу программиста за счет меньшего объема требуемых знаний о синтаксисе и высокой степени читаемости (исходя из убеждения, что чем проще синтаксис, тем проще его читать).

2.2 Сравнение подходов к типизации и оптимизации программ

Сравнение подходов к типизации приведено в таблице 3.

Таблица 3 — Сравнение подходов к типизации

Язык	Тип типизации	Доступный анализ программ
Forth (1971)	Бестиповой	Корректность интерпретации содержимого стека обеспечивается соглашениями и выявляется на этапе исполнения.
Joy (2001)	Динамическая	Динамическая проверка типов при выполнении.
Factor (2003)	Динамическая	Модуль «stack-checker» выполняет роль статического анализатора: абстрактная интерпретация эффектов слов на абстрактном стеке и проверка согласованности ветвей.
Cat (2006)	Статическая	Вывод типов и верификация формы и размера стека, гарантии согласованности ветвлений и инвариантов циклов.
Wasm (2017)	Статическая	Однопроходная валидация модуля по типам и структурированному управлению, предотвращение underflow и несогласованности типов до исполнения.

Согласно таблице 3 в процессе развития стековые языки программирования постепенно улучшают свои возможности динамического и статического анализа. Вместе с уровнем анализа растет также уровень безопасности программ.

Основные техники оптимизации показаны в таблице 4.

Таблица 4 — Специфичные техники оптимизации для стековых языков

Оптимизация	Описание
Нитевой код (Forth)	Представление программы как последовательности ссылок на слова; снижает размер кода и удешевляет интерпретацию.
Разворачивание слов времени компиляции (Forth)	Трнсляция управляющих конструкций в примитивные переходы и последовательности слов, уменьшая накладные расходы управления.
Инлайнинг цитат и макрорасширение (Factor)	Снижение накладных расходов вызовов и диспетчеризации за счёт разворачивания и специализации на этапе компиляции.

Оптимизации в стековых языках можно условно разделить на две группы: преобразования представления и диспетчеризации команд (ускоряют интерпретатор/виртуальную машину) и оптимизации, использующие результаты анализа семантики (стек-эффекты, типы, структура управления), что позволяет убирать лишние проверки и преобразовывать программу без нарушения корректности. Стековые языки изначально поддерживают техники первой группы вследствие своей структуры (например, нитевой код Forth). С развитием подходов к статическому анализу становятся доступны оптимизации из второй группы – Cat, Wasm активно применяют оптимизации, основанные на эквивалентных преобразованиях структуры программы, для увеличения скорости работы и снижении размера веса программы.

2.3 Выводы

В пути развития стековых языков видна тенденция: изначально работающие на низких уровнях абстракций благодаря простоте рантайма (в сравнении с нестековыми языками), высокой переносимости и оптимизациям, доступным при меньшей стоимость за счет структуры программ и модели исполнения (например, нитевой код Forth), стековые языки обзаводятся формальной базой, которая позволяет точно описывать работу программ и проводить развитой статический анализ и трансформации кода, нацеленные на уменьшение объема потребляемой памяти и увеличения скорости работы.

Одной из вершин достижений стековых языков можно считать Wasm. Исходные цели проектирования предлагают во многом подходящие для использования стекового языка условия, на основе которых выросла система, сочетающая в себе сразу несколько изначально трудносовмещаемых преимуществ: высокая портируемость под разные платформы (в сравнении с регистровыми виртуальными машинами), но при этом большая производительность (например, в сравнении с традиционными движками ECMAScript) за счет низкоуровневых интерфейсов и оптимизаций и верифицируемость (и оттого частично безопасность исполнения). Wasm вбирает в себя теоретические и практические наработки в области стековых языков, решая с помощью них сложную задачу исполнения прикладного кода с учетом заданных требований.

В результате анализа можно сделать вывод, что современная ниша стековых языков – это быстрые виртуальные машины, запускаемые на широком спектре устройств. Это утверждение также подтверждается фактом, что спецификации самых распространенных виртуальных машин (JVM, .NET CLR, Wasm) используют стековую модель исполнения кода.

3 Анализ средств вывода типов стековых языков программирования

Использование стека данных в качестве единого хранилища порождает несколько возможных проблем типизации. Первая – проблема несовпадение типов, когда на вход некоторой команде с вершины стека поступают данные неправильного типа. Вторая – проблема исчерпания стека, когда команда при выполнении пытается взять данные с пустого стека. Такие ошибки приводят к аварийным завершениям программ и тем самым снижают их надежность. В то же время от подобных ошибок можно защититься, если в момент компиляции программы статически определять состояния стека во все будущие моменты времени исполнения и в случае нахождения ошибки блокировать компиляцию программы как небезопасной. Просчет возможных состояний стека для программы на стековом языке называется выводом ее типа.

В стековых языках тип для некоторого выражения определяется через эффект, который оказывает это выражение на конфигурацию стека данных при выполнении. В свою очередь конфигурация стека состоит из размера стека (сколько данных на нем лежит) и «формы» стека (данные каких типов на нем лежат). Задача вывода типов для стековых языков сводится к нахождению типа для всей программы – как программа поменяет конфигурацию стека при выполнении.

В общем случае выводом типов в языках программирования называется процесс, который позволяет установить неуказанный явно тип некоторого выражения в тексте программы по ограничениям окружения, где это выражение встречается. Разные языки обладают разными возможностями вывода в зависимости от размера области программы, которая используется для вывода типа. Некоторые языки допускают простые выводы, основанные на анализе только самого выражения, для которого выводится тип. Например, вывод типов в язык программирования Java ограничивается автоматическим определением типа переменной при ее объявлении по выражению, которое инициализирует данную переменную. Другие языки идут дальше и позволяют выводить типы по совокупности выражений и совокупности переменных, задействованных в этих выражениях в рамках функций. К примеру, Rust,

который позволяет выводить тип переменной из контекста выражений, где эта переменная используется – так, если функция возвращает целое 32-битное число, то для гипотетической переменной, из которой делается возврат значения и которая может быть объявлена где угодно внутри такой функции, выводится соответственно тип целого 32-битного числа (то есть тип вывелся исходя из как минимум двух выражений, где переменная была задействована). Наконец существуют языки, вывод типов в которых осуществляется в рамках всей единицы трансляции – зачастую это языки с функциональной парадигмой. В таком случае типы для всей программы можно не указывать вовсе, так как все они будут выведены неявно при компиляции.

Для стековых языков алгоритмы вывода типов всей программы (последний случай, рассмотренный в предыдущем абзаце) актуальны, так как зачастую программисту затруднительно самому вычислить тип программы, что и порождает ошибки работы со стеком данных.

Для вывода типов в функциональных языках используется алгоритм Хиндли-Милнера. Вкратце алгоритм при запуске над несколькими выражениями сводится к двум шагам: сначала вычисляются ограничения, которые задаются формой выражений (например, одна переменная равна другой или переменная участвует в сложении, а потому должна быть складываемой), ограничения составляют систему уравнений, для которой вторым шагом ищется решение – такой набор типов, при подстановке которых, выражения останутся непротиворечивыми. Если набор найден, то типы выведены успешно, иначе – ошибка типизации.

Согласно идее, предложенной Кристофером Диггинсом, для стековых языков использование алгоритма Хиндли-Милнера также возможно, но с поправкой на row-полиморфизм. Row-полиморфизм – полиморфизм по размеру строки («row»), состоянию стека данных. Все команды row-полиморфны и оперируют всем стеком сразу, который состоит из хвоста (типовой вектор, обозначается большими латинскими буквами) и единичных значений, лежащих на вершине стека (обозначаются маленькими латинскими буквами либо названиями типов, например «a» и «Bool»). Для цитат

используется запись со скобками, показывающая отложенность описанных вычислений (пример, конфигурация стека, на котором лежит цитата «A (A -> B)»).

Диггинс демонстрирует возможность статического вывода типов на примере своего экспериментального стекового языка Cat. Для встроенных команд приводятся типы, показанные на листинге 2.

Листинг 2 — Типы встроенных команд Cat

```
// Применение цитаты, которая приводит конфигурацию S к
конфигурации R, к стеку S
apply   : (S (S -> R) -> R)
// Формирование цитаты, путем захвата значения с вершины стека
quote   : (S a -> S (R -> R a))
// Композиция двух цитат
compose : (S (B -> C) (A -> B) -> S (A -> C))
// Дублирование значения на вершине стека
dup     : (S a -> S a a)
// Удаление значения с вершины стека
pop     : (S a -> S)
// Перестановка двух значений с вершины стека
swap    : (S a b -> S b a)
// Примитив ветвления, выбирающий то или иное значения со
стека в зависимости от значения Bool
cond    : (S Bool a a -> S a)
// Примитив циклических операций, выполняющий тело (цитата на
вершине) пока верно условие (цитата под вершиной)
while   : ((S -> R Bool) (R -> S) S -> S)
```

Стоит обратить внимание на ограничения, которые ставят описанные в листинге 2 типы. Для условного оператора «if» тип требует, чтобы обе ветви, принимая на вход одну и ту же конфигурацию стека, возвращали тоже одну и ту жу измененную конфигурацию стека (при этом конкретные значения на стеке, конечно, могут различаться). Для цикла «while» тип требует, чтобы после выполнения цикла конфигурация стека оставалась такой же, как и на момент начала цикла.

Алгоритм вывода типа заключается в последовательном «сцеплении» типов команд, образующих программу. В своем техническом отчете Диггинс приводит текстовое описание алгоритма и пример, в котором происходит вывод типа для программы «[dup] apply». В результате применения алгоритма получается следующая цепочка типов: «A a -> A a (A a ->

A a a) -> A a a» (примечание, в отчете приводятся только начальная и конечная конфигурации стека, но аналогичным образом возможно построить и промежуточные конфигурации). Как видно, цитата и команда «apply» взаимно уничтожаются, из-за чего результирующий тип эквивалентен применению одиночной команды «dup» с точностью до промежуточных конфигураций.

Предложенный алгоритм имеет ограничение, заключающееся в невозможности вычислять рекурсивные типы. Из-за этого ограничения нельзя вычислить тип для программ, которые имеют рекурсивный поток управления. Также нельзя вычислить тип программ, предполагающих рекурсию за счет использования конструкции «dup apply» (объяснить это можно так: «apply» предполагает, что на стеке лежит некоторая цитата, которая переводит стек под собой из начального состояния в некоторое другое, но при этом «dup» говорит, что сама цитата также является частью этого начального состояния под собой). Такие ограничения можно снять, если расширить систему типов дополнительными конструкциями.

Таким образом для стековых языков возможно статически вывести форму и размер стека, и в случае наличия ошибок заранее их заметить. Признаком исчерпания стека служит наличие значений на вершине стека в начальной конфигурации программы – то есть программа ожидает, что на стеке до ее запуска уже что-то лежит, хотя это не так. Признак несовпадения типов выражается невозможностью найти решение системы уравнений в процессе очередной сцепки типов команд в программе.

4 Анализ устройства байткода и модели исполнения WebAssembly

Байткод WebAssembly (или сокращенно Wasm) спроектирован как низкоуровневая переносимая платформа с околонативной производительностью вычислений. Поддерживается 2 формата описания программ Wasm: бинарное представление, понимаемое виртуальной машиной Wasm, и текстовое представление в синтаксисе S-выражений, удобное для чтения человеком. В зависимости от цели, одну и ту же программу возможно представить в обоих вариантах.

Единицей трансляции Wasm является модуль – описанный спецификацией Wasm файл, содержащий коды программ, наборы метаданных, информацию об импортах и экспортах и другие составляющие. Модуль является самостоятельной единицей, готовой к запуску на виртуальной машине. Для представления чисел используется формат little-endian (младшие разряды чисел записываются в младших адресах). При кодировании векторов (последовательностей значений) в начале блока данных записываются 4 байт значения длины векторов, а затем кодируемые данные. Строки кодируются как байтовые вектора utf-8.

Содержимое модуля состоит преамбулы (значение «\0asm» и 4 байт номера версии) и основной части – набор секций. Каждая секция состоит из байта-идентификатора, 4 байт размера секции и полезной нагрузки в соответствии с идентификатором секции. Секции могут идти в произвольном порядке, но рекомендуется располагать их по мере увеличения идентификаторов.

Помимо модуля Wasm определяет также нескольких других сущностей, которые описываются в модуле:

– функции – блоки исполняемого кода;

– таблицы ссылок – механизм реализации указателей на функции, динамических вызовов;

– глобальные переменные и константы – набор типизированных изменяемых или неизменяемых значений фиксированного типа;

– линейная память – непрерывные последовательности байт;

– теги – механизм исключений.

Сущности каждого вида объединяются в индексные пространства, которые используются для ссылок между секциями. Если сущность из адресного пространства задается несколькими секциями (что верно для функций, таблиц и линейной памяти, у которых объявление и определение разделены на разные секции, подробнее в таблице 5), то нумерация в этих секциях идет параллельно (например, функция с индексом 1 задается объявлением с индексом 1 и определением с индексом 1). Внутри индексного пространства нумерация сквозная и начинается с 0. Сначала нумеруются импортированные сущности в порядке перечисления в секции импорта, затем аналогично в порядке перечисления в соответствующих секциях нумеруются внутренние сущности.

Описание каждой секции приведено в таблице 5.

Таблица 5 — Описание секций модуля

Ид.	Название	Описание
0	Custom section	Игнорируется виртуальной машиной, хранит произвольные данные. Может использоваться для различных целей, в том числе для отладки.
1	Type section	Объявления доступных для использования типов функций (типы параметров и типы результатов). Остальные секции ссылаются на данную при необходимости указать тип.
2	Import section	Объявления сущностей, которые импортируются в модуль для работы. Определения импортируемых сущностей предоставляют модули, которые их экспортируют.
3	Function section	Объявления типов для определенных в модуле функций, имеет параллельную с секцией Code нумерацию.
4	Table section	Объявления таблиц ссылок, имеет параллельную с секцией Element нумерацию.
5	Memory section	Объявления линейной памяти.

Продолжение таблицы 5

Ид.	Название	Описание
6	Global section	Объявления глобальных переменных.
7	Export section	Объявления публичных сущностей, которые могут быть импортированы другими модулями.
8	Start section	Объявление индекса функции, которую необходимо вызвать при инстанцировании модуля.
9	Element section	Объявления элементных сегментов – заполнения таблиц.
10	Code section	Объявления тел функций.
11	Data section	Объявление сегментов данных.
12	Data count section	Опциональное указание числа сегментов данных (Data section), которое ускоряет валидацию модуля.
13	Tag section	Объявления тегов для реализации механизма исключений.

Вызов функций возможен либо напрямую через указание индекса нужной функции либо косвенно через таблицу ссылок. Для операций над данными используется стек данных. Чтобы выполнить некую операцию, необходимо положить на стек требуемые данные, операция берет операнды со стека и кладет на стек результат выполнения. Функции действуют аналогично. Аналогично описанию в разделе 3, тип функции описывает преобразование верхушки стека, которое осуществляет функция.

Спецификация Wasm использует идею «непрозрачных» указателей, которая выражается в том, что с точки зрения программы представление указателей не раскрывается, они выступают лишь значениями, которые ссылаются на что-то. Оперирование указателями осуществляется через отдельные команды. Указатели нельзя сохранять в линейную память – только в таблицы ссылок.

Wasm различает восстановимые и невосстановимые ошибки. Невосстановимые ошибки называются «trap» и приводят к аварийному завершению программы. Восстановимые ошибки реализованы как исключения. При

возникновении исключения указывается его тег и машина раскручивает стек вызовов программы, пока не найдется обработчик соответствующего тега или стек вызовов будет раскручен полностью, и программа аварийно завершится.

Таким образом, спецификация WebAssembly предлагает низкоуровневую виртуальную машину. За счет низкого уровня и небольшого количества абстракций и ряда инженерных решений (таких как стековая архитектура, непрозрачные указатели) достигается во-первых малая удельная стоимость исполнения команд, что делает программы Wasm сопоставимы по скорости с нативными программами, а во-вторых снижается сложность реализаций виртуальной машины для различных аппаратных платформ (чем проще машина, тем проще ее реализовать) и вследствие этого увеличивается география устройств, для которых реализована виртуальная машина и возможно исполнение Wasm-программ. Эти преимущества делают Wasm подходящей платформой для запуска аппаратно-независимых программ на множества устройств, что активно используется в веб-разработке и облачных сервисах.

ЗАКЛЮЧЕНИЕ

В результате выполнения преддипломной практики были проанализированы различные аспекты стековых языков программирования, алгоритм статического вывода типов для стековых языков, а также модель исполнения и структура WebAssembly. Полученные знания использованы при разработке программного обеспечения согласно теме «компилятор стекового языка программирования».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Mathematical foundations of Joy [Электронный ресурс]. URL: https://hypercubed.github.io/joy/html/j02maf.html (дата обращения: 10.10.2025).

2. An informal tutorial on Joy [Электронный ресурс]. URL: https://hypercubed.github.io/joy/html/j01tut.html (дата обращения: 10.10.2025).

3. The prototype implementation of Joy [Электронный ресурс]. URL: https://hypercubed.github.io/joy/html/j09imp.html (дата обращения: 10.10.2025).

4. ANS Forth standard [Электронный ресурс]. URL: https://forth-standard.org/standard/words (дата обращения: 25.09.2025).

5. Forth documentation [Электронный ресурс]. URL: https://www.forth.com/resources/forth-programming-language/ (дата обращения: 25.09.2025).

6. The Algebra of Joy [Электронный ресурс]. URL: https://hypercubed.github.io/joy/html/j04alg.html (дата обращения: 10.10.2025).

7. Simple Type Inference for Higher-Order Stack-Oriented Languages [Электронный ресурс]. URL: https://dcreager.net/remarkable/Diggins2008a.pdf (дата обращения: 13.10.2025).

8. Typing Functional Stack-Based Languages [Электронный ресурс]. URL: https://dcreager.net/remarkable/Diggins2008b.pdf (дата обращения: 13.10.2025).

9. Cat language repository [Электронный ресурс]. URL: https://github.com/cdiggins/cat-language (дата обращения: 13.10.2025).

10. Factor: a dynamic stack-based programming language [Электронный ресурс]. URL: https://factorcode.org/slava/dls.pdf (дата обращения: 20.11.2025).

11. Factor wiki [Электронный ресурс]. URL: https://concatenative.org/wiki/view/Factor (дата обращения: 20.11.2025).

12. Understanding WebAssembly text format [Электронный ресурс]. URL: https://developer.mozilla.org/en-US/docs/WebAssembly/Guides/Understanding_the_text_format (дата обращения: 11.11.2025).

13. WebAssembly Specification 1.0 [Электронный ресурс]. URL: https://webassembly.github.io/spec/versions/core/WebAssembly-1.0.pdf (дата обращения: 11.11.2025).

14. A foundation for typed concatenative languages [Электронный ресурс]. URL: https://www2.ccs.neu.edu/racket/pubs/dissertation-kleffner.pdf (дата обращения: 15.10.2025).

15. Implementation of stack-based languages on register machines [Электронный ресурс]. URL: https://repositum.tuwien.at/handle/20.500.12708/13464 (дата обращения: 10.12.2025).

16. Stack Caching for Interpreters [Электронный ресурс]. URL: https://www2.cs.arizona.edu/~collberg/Teaching/553/2011/Resources/ertl94sc.pdf (дата обращения: 10.12.2025).

17. Combining Stack Caching with Dynamic Superinstructions [Электронный ресурс]. URL: https://dl.acm.org/doi/10.1145/1059579.1059583 (дата обращения: 10.12.2025).