Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ7
(Индекс)
_____ И. В. Рудаков
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

ЗАДАНИЕ
на выполнение курсового проекта

по дисциплине Конструирование компиляторов

Студент группы ИУ7И-21М

Диас Сергей Рамирович

(Фамилия, имя, отчество)

Тема курсового проекта Компилятор Go на Swift для LLVM

Направленность КП (учебный, исследовательский, практический, производственный, др.)
учебный

Источник тематики (кафедра, предприятие, НИР) Кафедра

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание
Разработка компилятора языка Go написанный на Swift с помощью SwiLex и SwiParse для LLVM.

Оформление курсового проекта:

Расчетно-пояснительная записка на 20-30 листах формата А4.

Дата выдачи задания « __ » _____ 2024 г.

Руководитель курсового проекта _____ А.А.Ступников
(Подпись, дата) (И.О.Фамилия)

Студент _____ С.Р.Диас
(Подпись, дата) (И.О.Фамилия)

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение эвм и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Компилятор Go на Swift для LLVM»

Студент группы **ИУ7И-21М** _____ С.Р.Диас
(Подпись, дата) (И.О.Фамилия)

Руководитель _____ А.А.Ступников
(Подпись, дата) (И.О.Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной курсовой работы заключается в разработке компилятора для языка программирования Go. Компилятор должен уметь читать файл, содержащий исходный код на языке Go, и генерировать новый файл с сгенерированным кодом на LLVM IR в качестве результата.

Для достижения цели необходимо решить следующие задачи:

- Проанализировать грамматику языка программирования Go.
- Изучить и выбрать подходящие инструменты для создания компилятора.
- Разработать прототип компилятора для Go.

Аналитическая часть

Компилятор

Компилятор — это программное обеспечение, которое переводит код, написанный на языке программирования высокого уровня (например, C, C++, Go или Java), в машинный код или другой язык более низкого уровня. Результатом обычно является исполняемый файл, который можно запустить напрямую на компьютере без необходимости в исходном коде или компиляторе.

Компилятор можно разделить на три основные стадии: фронтенд, мидленд и бэкенд. Каждая из этих стадий выполняет определённые задачи в процессе перевода исходного кода высокого уровня в машинный код. [1]

1. Фронтенд

Фронтенд отвечает за анализ исходного кода. Он проверяет, что код синтаксически и семантически корректен.

Шаги на стадии фронтенда:

Лексический анализ:

Компилятор разбивает исходный код на последовательность токенов, которые являются мельчайшими единицами (например, ключевые слова, операторы, идентификаторы).
Пример: В выражении int x = 5; токенами будут int, x, =, 5, и ;.

Синтаксический анализ (парсинг):

Токены проверяются на соответствие грамматике языка программирования, чтобы убедиться в правильности структуры. На этом этапе создаётся синтаксическое дерево (также известное как абстрактное синтаксическое дерево или AST).
Пример: В выражении int x = 5; парсер проверяет, что оно соответствует правильному формату объявления переменной.

Семантический анализ:

AST анализируется для проверки смысла. Это включает в себя проверку, что переменные объявлены до их использования, что типы данных совместимы, и что функции вызываются с правильными аргументами.
Пример: Если попытаться сложить число со строкой, семантический анализатор вызовет ошибку.

Генерация промежуточного представления (IR):

Компилятор преобразует код в промежуточное представление (IR), которое является более простым, независимым от языка кодом. Это представление легче оптимизировать и использовать для генерации машинного кода.

2. Мидленд

Мидленд оптимизирует промежуточное представление для повышения производительности и эффективности. Цель — улучшить код без изменения его поведения.

Шаги на стадии мидленда:

Оптимизация:

Промежуточное представление подвергается ряду оптимизаций, таких как:

Свертывание констант: Замена выражений на константы (например, замена 2 + 3 на 5).

Удаление мёртвого кода: Удаление кода, который никогда не выполняется.

Развёртывание циклов: Расширение циклов для уменьшения накладных расходов на их управление.

Инлайн-функции: Замена вызовов функций их телами для избежания накладных расходов на вызов функции.

Эти оптимизации могут выполняться на нескольких уровнях, включая управление потоком, потоком данных и параллелизм на уровне инструкций.

3. Бэкенд

Бэкенд отвечает за генерацию фактического машинного кода для целевой архитектуры.

Шаги на стадии бэкенда:

Генерация кода:

Оптимизированное промежуточное представление преобразуется в машинный код, специфичный для архитектуры (например, x86, ARM).
Компилятор определяет, как переменные и функции соответствуют аппаратным ресурсам, таким как регистры процессора и память.

Распределение регистров:

Компилятор назначает переменные регистрами процессора, стараясь минимизировать количество обращений к памяти.

Выбор и планирование инструкций:

Компилятор выбирает конкретные инструкции из набора инструкций целевой архитектуры и определяет оптимальный порядок их выполнения.

Генерация ассемблерного кода:

Машинный код часто генерируется в виде ассемблерного кода, который является человекочитаемым представлением машинных инструкций.

Компоновка:

Бэкенд связывает объектный код с внешними библиотеками и другими скомпилированными модулями для создания финального исполняемого файла или бинарного кода.

Методы лексического анализа

Лексический анализ, также известный как токенизация, включает преобразование последовательности символов (исходного кода) в последовательность токенов (ключевые слова, идентификаторы, операторы и т.д.). Для выполнения лексического анализа используются несколько техник: [2]

Ручной лексический анализатор (Ad-hoc):

Это самый простой подход, при котором разработчик вручную пишет код для распознавания токенов на основе шаблонов. Часто используется конструкции `if-else` или `switch-case`.

- **Плюсы:** Легко реализовать для небольших или простых языков.
- **Минусы:** Трудно поддерживать и масштабировать для больших или сложных языков.

Конечные автоматы (Finite State Machines, FSM):

Лексер строится как конечный автомат, где состояния представляют различные части лексической структуры. По мере обработки входного потока конечный автомат переходит между состояниями в зависимости от вводимого символа.

- **Плюсы:** Эффективный метод, часто используется для регулярных языков (например, для токенизации).
- **Минусы:** Сложно реализовать вручную для более сложных шаблонов.

Регулярные выражения:

Регулярные выражения (regex) определяют шаблоны для различных типов токенов (например, идентификаторы, ключевые слова, числа, операторы). Лексер сканирует исходный код и использует регулярные выражения для сопоставления токенов.
Пример: Регулярное выражение для идентификатора может быть `[a-zA-Z_][a-zA-Z0-9_]*`, что означает букву или подчеркивание, за которыми следуют буквенно-цифровые символы или подчеркивания.

- **Плюсы:** Очень выразительный, лаконичный и широко используется в сочетании с другими методами.
- **Минусы:** Не подходит для более сложных языков с контекстно-зависимыми правилами.

Генераторы лексеров:

Инструменты, такие как Lex, Flex и ANTLR, могут автоматически генерировать лексер, предоставив список регулярных выражений для различных типов токенов.

- **Плюсы:** Автоматически генерирует эффективный код для лексера.

- **Минусы:** Требует изучения инструмента и может быть избыточным для простых проектов.

Методы синтаксического анализа

Синтаксический анализ, также называемый парсингом, проверяет, соответствует ли последовательность токенов правилам грамматики языка. Парсер обычно создает абстрактное синтаксическое дерево (AST) или дерево разбора. [2]

Восходящий парсинг (Top-Down Parsing):

Парсер начинает с начального символа грамматики и пытается вывести входную строку, рекурсивно разворачивая правила грамматики.

Методы восходящего парсинга:

- **Рекурсивный спуск (Recursive Descent Parsing):**
  Парсер состоит из набора взаимно рекурсивных функций, каждая из которых соответствует правилу грамматики.
  - **Плюсы:** Простой в реализации, особенно для простых грамматик.
  - **Минусы:** Не подходит для лево-рекурсивных грамматик (например, грамматик, где правило ссылается на себя с левой стороны).
- **Предсказательный парсинг (LL Parsing):**
  Особая форма рекурсивного спуска, где парсер может принимать решения на основе следующего токена (также называемого lookahead).
  - **LL(1) парсинг:** Использует один токен для принятия решений.
  - **Плюсы:** Эффективен для некоторых грамматик, легко реализуется для LL(1) грамматик.
  - **Минусы:** Не подходит для всех грамматик, особенно для неоднозначных или лево-рекурсивных.

Нисходящий парсинг (Bottom-Up Parsing):

Парсер начинает с входных токенов и пытается свернуть их до начального символа, применяя правила грамматики в обратном порядке.

Методы нисходящего парсинга:

- **Парсинг методом сдвига и свертки (Shift-Reduce Parsing):**
  Парсер использует стек для сдвига токенов (записывая их в стек) и сворачивает их, используя правила грамматики.
  - **Плюсы:** Работает для более широкого диапазона грамматик, чем методы восходящего парсинга.
  - **Минусы:** Сложнее реализовать вручную.
- **LR парсинг:**
  Тип парсера сдвига и свертки, который более мощен, чем LL парсинг. Он использует lookahead для принятия решений и способен обрабатывать левую рекурсию.
  - **LR(0), SLR(1), LALR(1) и LR(1):** Это все версии LR парсеров, которые различаются по объему lookahead и способам разрешения неоднозначностей.
  - **Плюсы:** Очень мощные; могут обрабатывать большинство грамматик языков программирования.

- **Минусы:** Сложно реализовать вручную, часто генерируются с помощью генераторов парсеров.

Генераторы парсеров:

Инструменты, такие как Yacc, Bison и ANTLR, генерируют парсеры на основе формальной грамматики.

- **Плюсы:** Автоматизирует процесс создания парсера, снижая вероятность ошибок.
- **Минусы:** Требует изучения инструмента, а отладка может быть сложнее, чем с ручными парсерами.

Парсинг по методу Эрли (Earley Parsing):

Общий алгоритм парсинга, который может обрабатывать любую контекстно-свободную грамматику, включая лево-рекурсивные и неоднозначные грамматики.

- **Плюсы:** Очень мощный, обрабатывает неоднозначные и сложные грамматики.
- **Минусы:** Менее эффективен, чем LR и LL парсеры для некоторых грамматик.

Алгоритм CYK (Cocke-Younger-Kasami):

Алгоритм нисходящего парсинга, использующий динамическое программирование. В основном используется для парсинга контекстно-свободных грамматик в нормальной форме Хомского (CNF).

- **Плюсы:** Работает для любой контекстно-свободной грамматики в CNF.
- **Минусы:** Не очень эффективен для больших входных данных или сложных грамматик; в основном представляет теоретический интерес.

SwiLex и SwiParse: библиотеки для лексического и синтаксического анализа на языке Swift

SwiLex и SwiParse — это библиотеки, обычно используемые в языке программирования Swift для выполнения лексического и синтаксического анализа.

SwiLex: [3]

SwiLex — это генератор лексеров для Swift, который генерирует лексические анализаторы. Разработчики определяют регулярные выражения или шаблоны токенов, а SwiLex выполняет обработки процесса токенизации.
SwiLex в основном использует регулярные выражения для определения шаблонов токенов и автоматизирует процесс создания лексера.

SwiParse: [4]

SwiParse — это библиотека для генерации парсеров, которая используется для реализации синтаксического анализа в Swift.

SwiParse позволяет разработчикам определять правила грамматики, которые напоминают парсеры рекурсивного спуска, и библиотека выполняет разбор на основе этих правил.

LLVM

LLVM (Low Level Virtual Machine) — это мощный, модульный и гибкий фреймворк для компиляторов, который предоставляет набор инструментов для создания компиляторов и связанных технологий. Изначально разработанный как исследовательский проект, LLVM стал широко используемой и производственной системой для многих языков программирования. Его часто используют для создания промежуточных представлений (IR), оптимизации кода и генерации машинного кода. [5]

Основные особенности LLVM:

Промежуточное представление (IR):

- LLVM предоставляет собственное промежуточное представление (LLVM IR), которое является низкоуровневым, платформо-независимым и строго типизированным языком, используемым как промежуточный шаг в процессе компиляции.
- LLVM IR позволяет разработчикам писать оптимизации и трансформации портативным образом для различных архитектур.

Модульная архитектура:

- LLVM обладает высокой модульностью, что означает, что он состоит из отдельных библиотек для различных этапов компиляции, включая фронтенды, оптимизаторы и бэкенды.
- Вы можете использовать LLVM только для части процесса компиляции (например, для оптимизации) или для всего процесса — от исходного кода до исполняемого файла.

Оптимизация кода:

- LLVM предлагает широкий набор агрессивных оптимизаций, которые можно применить к промежуточному представлению, помогая улучшить производительность, сократить размер кода и повысить эффективность выполнения.
- Оптимизация может выполняться на этапе компиляции, линковки и даже во время выполнения (Just-In-Time, или JIT, компиляция).

Независимый от платформы бэкенд:

- LLVM IR может быть преобразован в машинный код для различных архитектур. LLVM поддерживает широкий спектр целевых платформ (например, x86, ARM, RISC-V), что делает его очень портативным.
- Это означает, что вы можете генерировать код для различных аппаратных архитектур из одного и того же IR без изменения исходного кода.

Just-In-Time компиляция (JIT):

- LLVM поддерживает JIT-компиляцию, что позволяет компилировать код во время выполнения, а не заранее (AOT). Это полезно для динамических языков и сред, где важна производительность.
- Возможности JIT в LLVM используются в средах, таких как интерпретаторы или виртуальные машины (например, JavaScript-движки).

Фронтенды:

- LLVM поддерживает фронтенды для многих языков программирования. Фронтенд переводит высокоуровневый исходный код (например, C, C++ или Swift) в LLVM IR.
- Популярные фронтенды, использующие LLVM, включают Clang (для C, C++, Objective-C), Swift и Rust.

Экосистема LLVM:

- LLVM — это не просто фреймворк для компиляторов; он превратился в полноценную экосистему, включающую такие проекты, как Clang, LLD (линкер LLVM), LLDB (отладчик LLVM) и многие другие инструменты и библиотеки, связанные с компиляторами, оптимизацией и разработкой.

Примеры использования LLVM:

- **Создание компиляторов:** LLVM может использоваться как бэкенд для создания компилятора для любого языка. Многие современные языки, такие как Rust, Swift и Julia, используют LLVM как часть своего компиляционного пайплайна.
- **Оптимизация кода:** Оптимизатор LLVM может использоваться независимо для улучшения производительности кода, сгенерированного другими компиляторами.
- **JIT-компиляция:** Языки и фреймворки, которым необходима высокая производительность во время выполнения (например, игровые движки или виртуальные машины), часто используют возможности JIT-компиляции в LLVM.
- **Кроссплатформенная разработка:** LLVM упрощает компиляцию на различные архитектуры из одной кодовой базы, что делает его ключевой технологией для кроссплатформенных языков.

Выводы

В этом разделе были описаны основные этапы работы компилятора. Также были рассмотрены библиотеки SwiLex и SwiParse, которые были выбраны для разработки лексического и синтаксического анализа в данном проекте компилятора. Наконец, был описан LLVM, который был выбран в качестве формата выходного кода для компилятора.

Конструкторская часть

IDEF0

Концептуальная модель разрабатываемого компилятора в нотации IDEF0 представлена на рисунках 1 и 2.

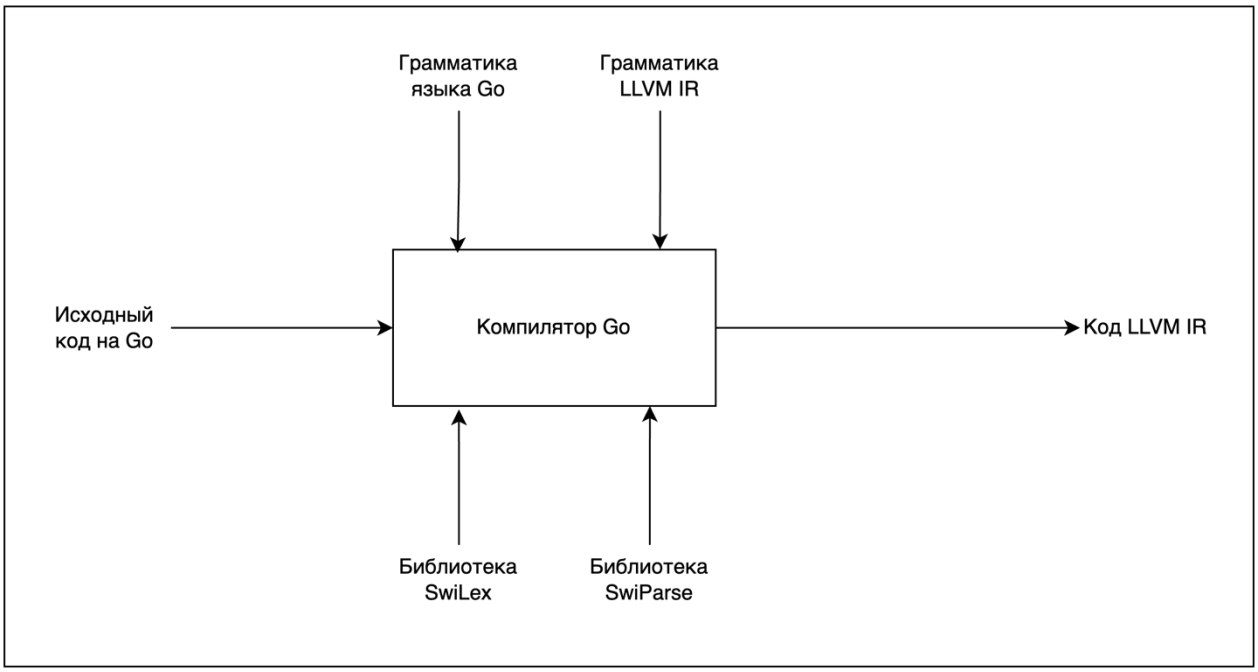

Рис. 1 IDEF0

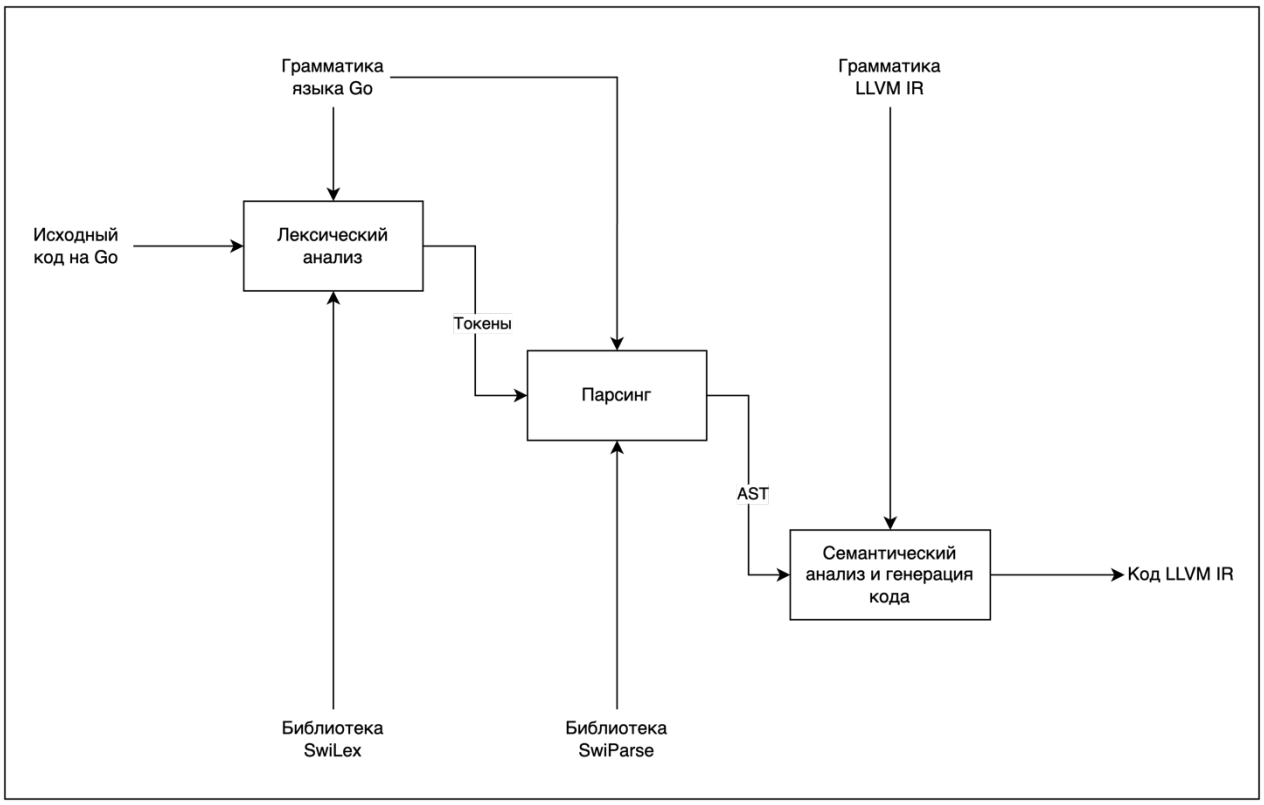

Рис. 2 IDEF0

Язык Go

Go (также называемый Golang) — это язык программирования с открытым исходным кодом, статически типизированный и компилируемый, разработанный компанией Google. Go был создан для того, чтобы быть простым, эффективным и масштабируемым, с целью решения общих проблем в системном программировании, особенно в области конкуренции, простоты и быстрой компиляции. [6]
Грамматика, используемая для разработки компилятора в этом проекте, представлена в приложении А.

Лексический и синтаксический анализаторы

Библиотеки SwiLex и SwiParse используются в этой работе для лексического и синтаксического анализа. SwiLex принимает исходный код на языке Go на входе и генерирует токены на выходе. Токены поступают на вход SwiParse, который генерирует AST на выходе. Это показано на рисунке 2.

Семантический анализ

AST анализируется снизу вверх, выполняя специфические методы для каждого узла. В процессе компиляции проверяются правильность определения переменных и типы значений при присваивании, а также отслеживаются размеры массивов и типы объектов.

Выводы

В этом разделе была предоставлена диаграмма IDEF0 компилятора. Также было дано краткое описание языка программирования Go и грамматики, используемой в рамках этой работы. Наконец, были описаны процессы лексического, синтаксического и семантического анализа.

Технологическая часть

Выбор средств программной реализации

Для разработки компилятора был выбран язык программирования Swift по следующим причинам:
- Уже приобретенный опыт использования этого языка программирования.
- Существующие библиотеки для этого языка для разработки компиляторов, такие как SwiLex и SwiParse.

Тестирование

Компилятор тестировался в основном путем компиляции и выполнения двух исходных кодов, написанных на языке программирования Go:
- Алгоритм Bubble sort
- Алгоритм инвертирования связанного списка

Пример работы программы

Коды 1 и 3 показывают соответственно входные исходные коды, написанные на языке программирования Go для алгоритма Bubble sort и алгоритма инвертирования связанного списка. Коды 2 и 4 показывают соответственно сгенерированный вывод в LLVM IR упомянутых алгоритмов.

```
func bubbleSort(arr []int, n int) {
    var i int;
    var j int;
    var t1 int;

    t1 = n - 1 ;
    i = 0;

    for i < t1{
        var t2 int;
        t2 = n-i-1;
        j = 0;
        for j < t2 {
            var nextPos int;
            nextPos = j + 1;
            var current int;
            current = arr[j];
            var next int;
            next = arr[nextPos];
            if current > next {
                arr[j] = next;
                arr[nextPos] = current;
            };
            j = j + 1;
            };
            i = i + 1;
        };
```

```
    }

    func main() {
        var arr []int;
        var length int;
        var i int;
        var n int;
        var val int;
        fmt.Println("Insert the number of values:");
        fmt.Scanln(&length);
        arr = make([]int, length);
        i = 0 ;
        fmt.Println("Insert the values:");
        for i < length{
            var l int;
            fmt.Scanln(&l);
            arr[i] = l;
            i = i + 1;
        };
        n = len(arr);
        i = 0;
        fmt.Println("Unsorted array:");
        for i < n {
            val = arr[i];
            fmt.Print(val);
            fmt.Print(" ");
            i = i + 1;
        };
        fmt.Println("");
        bubbleSort(arr, n);
        fmt.Println("Sorted array:");
        i = 0;
        for i < n {
            val = arr[i];
            fmt.Print(val);
            fmt.Print(" ");
            i = i + 1;
        };
        fmt.Println("");
    }
```

Код 1 алгоритм Bubble sort (GO)

```
define void @print_str(i8* %str) {
entry:
   ; Print the string using printf
   call i32 (i8*, ...) @printf(i8* %str)
   ret void
}

define void @print_int(i32 %num) {
entry:
   ; Print the integer using printf
```

```
    %strIntFormat = getelementptr [4 x i8], [4 x i8]* @.strIntFormat, i32 0, i32 0
    call i32 (i8*, ...) @printf(i8* %strIntFormat, i32 %num)
    ret void
}

define void @println_str(i8* %str) {
entry:
    ; Print the string
    call void @print_str(i8* %str)

    ; Print a newline character
    %strNewlinePtr = getelementptr [2 x i8], [2 x i8]* @.strNewline, i32 0, i32 0
    call i32 (i8*, ...) @printf(i8* %strNewlinePtr)

    ret void
}

define void @println_int(i32 %num) {
entry:
    ; Print the integer
    call void @print_int(i32 %num)

    ; Print a newline character
    %strNewlinePtr = getelementptr [2 x i8], [2 x i8]* @.strNewline, i32 0, i32 0
    call i32 (i8*, ...) @printf(i8* %strNewlinePtr)

    ret void
}
@.strIntFormatRead = private unnamed_addr constant [3 x i8] c"%d\00", align 1

; External function declaration for malloc (memory allocation)
declare i8* @malloc(i64)

define i32* @expand_and_copy_array(i32* %srcArray, i32 %N) {
entry:
    ; Calculate the size of the new array (10 x (N + 1))
    %newSize = add i32 %N, 1
    %totalElements = mul i32 %newSize, 10

    ; Calculate the total size in bytes (4 bytes per i32)
    %totalSizeInBytes = mul i32 %totalElements, 4
    %totalSizeInBytes64 = sext i32 %totalSizeInBytes to i64

    ; Allocate memory for the new array on the heap
    %newArray = call i8* @malloc(i64 %totalSizeInBytes64)

    ; Bitcast the allocated memory to an i32 pointer
    %newArrayPtr = bitcast i8* %newArray to i32*

    ; Loop counters
    %i = alloca i32, align 4
    %j = alloca i32, align 4
```

```
    store i32 0, i32* %i, align 4
    store i32 0, i32* %j, align 4

    br label %copy_outer_loop

copy_outer_loop:                                  ; Outer loop for rows
    %i_val = load i32, i32* %i, align 4
    %N_cond = icmp slt i32 %i_val, %N
    br i1 %N_cond, label %copy_inner_loop, label %initialize_new_row

copy_inner_loop:                                  ; Inner loop for columns
    %j_val = load i32, i32* %j, align 4
    %j_cond = icmp slt i32 %j_val, 10
    br i1 %j_cond, label %copy_elements, label %increment_row

copy_elements:                                    ; Copy elements from srcArray to newArray
    %srcIdx = mul i32 %i_val, 10
    %srcOffset = add i32 %srcIdx, %j_val
    %srcElementPtr = getelementptr inbounds i32, i32* %srcArray, i32 %srcOffset
    %srcElement = load i32, i32* %srcElementPtr, align 4

    %newIdx = mul i32 %i_val, 10
    %newOffset = add i32 %newIdx, %j_val
    %newElementPtr = getelementptr inbounds i32, i32* %newArrayPtr, i32 %newOffset
    store i32 %srcElement, i32* %newElementPtr, align 4

    ; Increment inner loop counter
    %j_next = add i32 %j_val, 1
    store i32 %j_next, i32* %j, align 4
    br label %copy_inner_loop

increment_row:                                    ; Increment outer loop counter
    %i_next = add i32 %i_val, 1
    store i32 %i_next, i32* %i, align 4

    ; Reset inner loop counter
    store i32 0, i32* %j, align 4
    br label %copy_outer_loop

initialize_new_row:                               ; Initialize the new row (N-th index)
    %i_newRow = load i32, i32* %i, align 4
    %j_new = load i32, i32* %j, align 4
    %j_newCond = icmp slt i32 %j_new, 10
    br i1 %j_newCond, label %init_elements, label %end_new_row

init_elements:                                    ; Initialize elements in the new row
    %newRowIdx = mul i32 %N, 10
    %newRowOffset = add i32 %newRowIdx, %j_new
    %newRowElementPtr = getelementptr inbounds i32, i32* %newArrayPtr, i32 %newRowOffset
    store i32 0, i32* %newRowElementPtr, align 4  ; Example: initialize with 0

    %j_newNext = add i32 %j_new, 1
```

```
    store i32 %j_newNext, i32* %j, align 4
    br label %initialize_new_row

end_new_row:
    ret i32* %newArrayPtr
}
@.strIntFormat = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.strNewline = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
declare i32 @scanf(i8*, ...)
declare i32 @printf(i8*, ...)
@.strConst1 = private unnamed_addr constant [29 x i8] c"Insert the number of values:\00", align 1
@.strConst2 = private unnamed_addr constant [19 x i8] c"Insert the values:\00", align 1
@.strConst3 = private unnamed_addr constant [16 x i8] c"Unsorted array:\00", align 1
@.strConst4 = private unnamed_addr constant [2 x i8] c" \00", align 1
@.strConst5 = private unnamed_addr constant [1 x i8] c"\00", align 1
@.strConst6 = private unnamed_addr constant [14 x i8] c"Sorted array:\00", align 1
@.strConst7 = private unnamed_addr constant [2 x i8] c" \00", align 1
@.strConst8 = private unnamed_addr constant [1 x i8] c"\00", align 1
define i32 @main() {
entry:
%length = alloca i32, align 4
store i32 0, i32* %length
%i = alloca i32, align 4
store i32 0, i32* %i
%n = alloca i32, align 4
store i32 0, i32* %n
%val = alloca i32, align 4
store i32 0, i32* %val
%strConstPtr1 = getelementptr [29 x i8], [29 x i8]* @.strConst1, i32 0, i32 0
call void @println_str(i8* %strConstPtr1)
call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.strIntFormat, i64 0, i64
0), i32* %length)
%length1 = load i32, i32* %length, align 4
%arraySize2 = mul i32 %length1, 4
%arraySize3 = sext i32 %arraySize2 to i64
%arrRaw4 = call i8* @malloc(i64 %arraySize3)
%arr = bitcast i8* %arrRaw4 to i32*
store i32 0, i32* %i
%strConstPtr2 = getelementptr [19 x i8], [19 x i8]* @.strConst2, i32 0, i32 0
call void @println_str(i8* %strConstPtr2)
br label %condBlock21

loopBlock42:
%loadedInt31 = alloca i32, align 4
%loadedInt32 = load i32, i32* %i
store i32 %loadedInt32, i32* %loadedInt31
%arrptr29 = getelementptr inbounds [0 x i32], [0 x i32]* %arr, i32 0, i32 %loadedInt32
%tempNumber30 = load i32, i32* %arrptr29
store i32 %tempNumber30, i32* %val
%loadedInt33 = load i32, i32* %val
call void @print_int(i32 %loadedInt33)
%strConstPtr4 = getelementptr [2 x i8], [2 x i8]* @.strConst4, i32 0, i32 0
```

```
call void @print_str(i8* %strConstPtr4)
%temp35 = alloca i32, align 4
store i32 1, i32* %temp35
%temp37 = load i32, i32* %i, align 4
%temp38 = load i32, i32* %temp35, align 4
%temp36 = add i32 %temp37, %temp38
%temp39 = alloca i32, align 4
store i32 %temp36, i32* %temp39
%loadedInt40 = load i32, i32* %temp39
store i32 %loadedInt40, i32* %i
br label %condBlock41

loopBlock62:
%loadedInt51 = alloca i32, align 4
%loadedInt52 = load i32, i32* %i
store i32 %loadedInt52, i32* %loadedInt51
%arrptr49 = getelementptr inbounds [0 x i32], [0 x i32]* %arr, i32 0, i32 %loadedInt52
%tempNumber50 = load i32, i32* %arrptr49
store i32 %tempNumber50, i32* %val
%loadedInt53 = load i32, i32* %val
call void @print_int(i32 %loadedInt53)
%strConstPtr7 = getelementptr [2 x i8], [2 x i8]* @.strConst7, i32 0, i32 0
call void @print_str(i8* %strConstPtr7)
%temp55 = alloca i32, align 4
store i32 1, i32* %temp55
%temp57 = load i32, i32* %i, align 4
%temp58 = load i32, i32* %temp55, align 4
%temp56 = add i32 %temp57, %temp58
%temp59 = alloca i32, align 4
store i32 %temp56, i32* %temp59
%loadedInt60 = load i32, i32* %temp59
store i32 %loadedInt60, i32* %i
br label %condBlock61

endBlock43:
%strConstPtr5 = getelementptr [1 x i8], [1 x i8]* @.strConst5, i32 0, i32 0
call void @println_str(i8* %strConstPtr5)
%n44 = load i32, i32* %n, align 4
call void @bubbleSort(i32* %arr, i32 %n44)
%strConstPtr6 = getelementptr [14 x i8], [14 x i8]* @.strConst6, i32 0, i32 0
call void @println_str(i8* %strConstPtr6)
store i32 0, i32* %i
br label %condBlock61
ret i32 0

endBlock23:
%arr24 = load i32, i32* %length, align 4
store i32 %arr24, i32* %n
store i32 0, i32* %i
%strConstPtr3 = getelementptr [16 x i8], [16 x i8]* @.strConst3, i32 0, i32 0
call void @println_str(i8* %strConstPtr3)
br label %condBlock41
```

```
ret i32 0

endBlock63:
%strConstPtr8 = getelementptr [1 x i8], [1 x i8]* @.strConst8, i32 0, i32 0
call void @println_str(i8* %strConstPtr8)
ret i32 0

condBlock21:
%temp7 = load i32, i32* %i, align 4
%temp8 = load i32, i32* %length, align 4
%temp6 = icmp slt i32 %temp7, %temp8
br i1 %temp6, label %loopBlock22, label %endBlock23

condBlock61:
%temp46 = load i32, i32* %i, align 4
%temp47 = load i32, i32* %n, align 4
%temp45 = icmp slt i32 %temp46, %temp47
br i1 %temp45, label %loopBlock62, label %endBlock63

loopBlock22:
%l = alloca i32, align 4
store i32 0, i32* %l
call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.strIntFormat, i64 0, i64
0), i32* %l)
%loadedInt12 = alloca i32, align 4
%loadedInt13 = load i32, i32* %i
store i32 %loadedInt13, i32* %loadedInt12
%l10 = load i32, i32* %l, align 4
%arrptr11 = getelementptr inbounds [0 x i32], [0 x i32]* %arr, i32 0, i32 %loadedInt13
store i32 %l10, i32* %arrptr11
%temp15 = alloca i32, align 4
store i32 1, i32* %temp15
%temp17 = load i32, i32* %i, align 4
%temp18 = load i32, i32* %temp15, align 4
%temp16 = add i32 %temp17, %temp18
%temp19 = alloca i32, align 4
store i32 %temp16, i32* %temp19
%loadedInt20 = load i32, i32* %temp19
store i32 %loadedInt20, i32* %i
br label %condBlock21

condBlock41:
%temp26 = load i32, i32* %i, align 4
%temp27 = load i32, i32* %n, align 4
%temp25 = icmp slt i32 %temp26, %temp27
br i1 %temp25, label %loopBlock42, label %endBlock43

}
define void @bubbleSort(i32* %arr64,i32 %n66) {
entry:
%arr65 = alloca i32*, align 8
store i32* %arr64, i32** %arr65, align 8
```

```
%arr = load i32*, i32** %arr65, align 8
%n = alloca i32, align 8
store i32 %n66, i32* %n, align 8
%i = alloca i32, align 4
store i32 0, i32* %i
%j = alloca i32, align 4
store i32 0, i32* %j
%t1 = alloca i32, align 4
store i32 0, i32* %t1
%temp68 = alloca i32, align 4
store i32 1, i32* %temp68
%temp70 = load i32, i32* %n, align 4
%temp71 = load i32, i32* %temp68, align 4
%temp69 = sub i32 %temp70, %temp71
%temp72 = alloca i32, align 4
store i32 %temp69, i32* %temp72
%loadedInt73 = load i32, i32* %temp72
store i32 %loadedInt73, i32* %t1
store i32 0, i32* %i
br label %condBlock140

endBlock142:
ret void

block114:
%temp124 = alloca i32, align 4
store i32 1, i32* %temp124
%temp126 = load i32, i32* %j, align 4
%temp127 = load i32, i32* %temp124, align 4
%temp125 = add i32 %temp126, %temp127
%temp128 = alloca i32, align 4
store i32 %temp125, i32* %temp128
%loadedInt129 = load i32, i32* %temp128
store i32 %loadedInt129, i32* %j
br label %condBlock130

endBlock132:
%temp134 = alloca i32, align 4
store i32 1, i32* %temp134
%temp136 = load i32, i32* %i, align 4
%temp137 = load i32, i32* %temp134, align 4
%temp135 = add i32 %temp136, %temp137
%temp138 = alloca i32, align 4
store i32 %temp135, i32* %temp138
%loadedInt139 = load i32, i32* %temp138
store i32 %loadedInt139, i32* %i
br label %condBlock140

block113:
%loadedInt117 = alloca i32, align 4
%loadedInt118 = load i32, i32* %j
store i32 %loadedInt118, i32* %loadedInt117
```

```
%next115 = load i32, i32* %next, align 4
%arrptr116 = getelementptr inbounds [0 x i32], [0 x i32]* %arr, i32 0, i32 %loadedInt118
store i32 %next115, i32* %arrptr116
%loadedInt121 = alloca i32, align 4
%loadedInt122 = load i32, i32* %nextPos
store i32 %loadedInt122, i32* %loadedInt121
%current119 = load i32, i32* %current, align 4
%arrptr120 = getelementptr inbounds [0 x i32], [0 x i32]* %arr, i32 0, i32 %loadedInt122
store i32 %current119, i32* %arrptr120
br label %block114

condBlock140:
%temp75 = load i32, i32* %i, align 4
%temp76 = load i32, i32* %t1, align 4
%temp74 = icmp slt i32 %temp75, %temp76
br i1 %temp74, label %loopBlock141, label %endBlock142

loopBlock141:
%t2 = alloca i32, align 4
store i32 0, i32* %t2
%temp81 = load i32, i32* %n, align 4
%temp82 = load i32, i32* %i, align 4
%temp80 = sub i32 %temp81, %temp82
%temp83 = alloca i32, align 4
store i32 %temp80, i32* %temp83
%temp84 = alloca i32, align 4
store i32 1, i32* %temp84
%temp86 = load i32, i32* %temp83, align 4
%temp87 = load i32, i32* %temp84, align 4
%temp85 = sub i32 %temp86, %temp87
%temp88 = alloca i32, align 4
store i32 %temp85, i32* %temp88
%loadedInt89 = load i32, i32* %temp88
store i32 %loadedInt89, i32* %t2
store i32 0, i32* %j
br label %condBlock130
br label %condBlock140

condBlock130:
%temp91 = load i32, i32* %j, align 4
%temp92 = load i32, i32* %t2, align 4
%temp90 = icmp slt i32 %temp91, %temp92
br i1 %temp90, label %loopBlock131, label %endBlock132

loopBlock131:
%nextPos = alloca i32, align 4
store i32 0, i32* %nextPos
%temp95 = alloca i32, align 4
store i32 1, i32* %temp95
%temp97 = load i32, i32* %j, align 4
%temp98 = load i32, i32* %temp95, align 4
%temp96 = add i32 %temp97, %temp98
```

```
%temp99 = alloca i32, align 4
store i32 %temp96, i32* %temp99
%loadedInt100 = load i32, i32* %temp99
store i32 %loadedInt100, i32* %nextPos
%current = alloca i32, align 4
store i32 0, i32* %current
%loadedInt103 = alloca i32, align 4
%loadedInt104 = load i32, i32* %j
store i32 %loadedInt104, i32* %loadedInt103
%arrptr101 = getelementptr inbounds [0 x i32], [0 x i32]* %arr, i32 0, i32 %loadedInt104
%tempNumber102 = load i32, i32* %arrptr101
store i32 %tempNumber102, i32* %current
%next = alloca i32, align 4
store i32 0, i32* %next
%loadedInt107 = alloca i32, align 4
%loadedInt108 = load i32, i32* %nextPos
store i32 %loadedInt108, i32* %loadedInt107
%arrptr105 = getelementptr inbounds [0 x i32], [0 x i32]* %arr, i32 0, i32 %loadedInt108
%tempNumber106 = load i32, i32* %arrptr105
store i32 %tempNumber106, i32* %next
%temp110 = load i32, i32* %current, align 4
%temp111 = load i32, i32* %next, align 4
%temp109 = icmp sgt i32 %temp110, %temp111
br i1 %temp109, label %block113, label %block114
br label %condBlock130

}
```

Код 2 алгоритм Bubble sort (LLVM IR)

```
type Node struct {
    value int;
    next  *Node;
}

type LinkedList struct {
    head *Node;
}

func Append(list *LinkedList, value int) {
    var newNode *Node;

    newNode = &Node{value: value};

    var tempHead *Node;

    tempHead = list.head;

    if tempHead == nil {
        list.head = newNode;
    } else {
        var current *Node;
        var next *Node;
        current = list.head;
```

```
        next = current.next;

        for next != nil {
            current = next;
            next = current.next;
        };

        current.next = newNode;
    };
}

func main() {
    var list *LinkedList;
    var current *Node;
    var prev *Node;
    var next *Node;
    var value int;
    var length int;
    var i int;

    list = &LinkedList{};
    fmt.Println("Insert the number of values:");
    fmt.Scanln(&length);

    i = 0 ;
    fmt.Println("Insert the values:");
    for i < length{
        var v int;
        fmt.Scanln(&v);
        Append(list, v);
        i = i + 1;
    };

    fmt.Println("Original linked list: ");
    current = list.head;
    for current != nil {
        value = current.value;
        fmt.Print(value, " -> ");
        current = current.next;
    };

    fmt.Println("nil");

    current = list.head;
    for current != nil {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    };
    list.head = prev;

    fmt.Println("Reversed linked list: ");
```

```
    current = list.head;
    for current != nil {
        value = current.value;
        fmt.Print(value, " -> ");
        current = current.next;
    };
    fmt.Println("nil");
}
```

Код 3 алгоритма инвертирования связанного списка (GO)

```
define void @print_str(i8* %str) {
entry:
   ; Print the string using printf
   call i32 (i8*, ...) @printf(i8* %str)
   ret void
}

define void @print_int(i32 %num) {
entry:
   ; Print the integer using printf
   %strIntFormat = getelementptr [4 x i8], [4 x i8]* @.strIntFormat, i32 0, i32 0
   call i32 (i8*, ...) @printf(i8* %strIntFormat, i32 %num)
   ret void
}

define void @println_str(i8* %str) {
entry:
   ; Print the string
   call void @print_str(i8* %str)

   ; Print a newline character
   %strNewlinePtr = getelementptr [2 x i8], [2 x i8]* @.strNewline, i32 0, i32 0
   call i32 (i8*, ...) @printf(i8* %strNewlinePtr)

   ret void
}

define void @println_int(i32 %num) {
entry:
   ; Print the integer
   call void @print_int(i32 %num)

   ; Print a newline character
   %strNewlinePtr = getelementptr [2 x i8], [2 x i8]* @.strNewline, i32 0, i32 0
   call i32 (i8*, ...) @printf(i8* %strNewlinePtr)

   ret void
}
@.strIntFormatRead = private unnamed_addr constant [3 x i8] c"%d\00", align 1

; External function declaration for malloc (memory allocation)
```

```
declare i8* @malloc(i64)

define i32* @expand_and_copy_array(i32* %srcArray, i32 %N) {
entry:
  ; Calculate the size of the new array (10 x (N + 1))
  %newSize = add i32 %N, 1
  %totalElements = mul i32 %newSize, 10

  ; Calculate the total size in bytes (4 bytes per i32)
  %totalSizeInBytes = mul i32 %totalElements, 4
  %totalSizeInBytes64 = sext i32 %totalSizeInBytes to i64

  ; Allocate memory for the new array on the heap
  %newArray = call i8* @malloc(i64 %totalSizeInBytes64)

  ; Bitcast the allocated memory to an i32 pointer
  %newArrayPtr = bitcast i8* %newArray to i32*

  ; Loop counters
  %i = alloca i32, align 4
  %j = alloca i32, align 4
  store i32 0, i32* %i, align 4
  store i32 0, i32* %j, align 4

  br label %copy_outer_loop

copy_outer_loop:                                  ; Outer loop for rows
  %i_val = load i32, i32* %i, align 4
  %N_cond = icmp slt i32 %i_val, %N
  br i1 %N_cond, label %copy_inner_loop, label %initialize_new_row

copy_inner_loop:                                  ; Inner loop for columns
  %j_val = load i32, i32* %j, align 4
  %j_cond = icmp slt i32 %j_val, 10
  br i1 %j_cond, label %copy_elements, label %increment_row

copy_elements:                                    ; Copy elements from srcArray to newArray
  %srcIdx = mul i32 %i_val, 10
  %srcOffset = add i32 %srcIdx, %j_val
  %srcElementPtr = getelementptr inbounds i32, i32* %srcArray, i32 %srcOffset
  %srcElement = load i32, i32* %srcElementPtr, align 4

  %newIdx = mul i32 %i_val, 10
  %newOffset = add i32 %newIdx, %j_val
  %newElementPtr = getelementptr inbounds i32, i32* %newArrayPtr, i32 %newOffset
  store i32 %srcElement, i32* %newElementPtr, align 4

  ; Increment inner loop counter
  %j_next = add i32 %j_val, 1
  store i32 %j_next, i32* %j, align 4
  br label %copy_inner_loop
```

```
increment_row:                                    ; Increment outer loop counter
   %i_next = add i32 %i_val, 1
   store i32 %i_next, i32* %i, align 4

   ; Reset inner loop counter
   store i32 0, i32* %j, align 4
   br label %copy_outer_loop

initialize_new_row:                               ; Initialize the new row (N-th index)
   %i_newRow = load i32, i32* %i, align 4
   %j_new = load i32, i32* %j, align 4
   %j_newCond = icmp slt i32 %j_new, 10
   br i1 %j_newCond, label %init_elements, label %end_new_row

init_elements:                                    ; Initialize elements in the new row
   %newRowIdx = mul i32 %N, 10
   %newRowOffset = add i32 %newRowIdx, %j_new
   %newRowElementPtr = getelementptr inbounds i32, i32* %newArrayPtr, i32 %newRowOffset
   store i32 0, i32* %newRowElementPtr, align 4  ; Example: initialize with 0

   %j_newNext = add i32 %j_new, 1
   store i32 %j_newNext, i32* %j, align 4
   br label %initialize_new_row

end_new_row:
   ret i32* %newArrayPtr
}
@.strIntFormat = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.strNewline = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
declare i32 @scanf(i8*, ...)
declare i32 @printf(i8*, ...)
%LinkedList = type { %Node* }
%Node = type { i32, %Node* }
@.strConst1 = private unnamed_addr constant [29 x i8] c"Insert the number of values:\00", align 1
@.strConst2 = private unnamed_addr constant [19 x i8] c"Insert the values:\00", align 1
@.strConst3 = private unnamed_addr constant [23 x i8] c"Original linked list: \00", align 1
@.strConst4 = private unnamed_addr constant [5 x i8] c" -> \00", align 1
@.strConst5 = private unnamed_addr constant [4 x i8] c"nil\00", align 1
@.strConst6 = private unnamed_addr constant [23 x i8] c"Reversed linked list: \00", align 1
@.strConst7 = private unnamed_addr constant [5 x i8] c" -> \00", align 1
@.strConst8 = private unnamed_addr constant [4 x i8] c"nil\00", align 1
define void @Append(%LinkedList* %list75,i32 %value77) {
entry:
%list76 = alloca %LinkedList*
store %LinkedList* %list75, %LinkedList** %list76
%list = load %LinkedList*, %LinkedList** %list76
%value = alloca i32, align 8
store i32 %value77, i32* %value, align 8
%newNode = call i8* @malloc(i96 12)
store %Node* null, %Node** %newNode
%struct80 = call i8* @malloc(i96 12)
%stuctObject78 = bitcast i8* %struct80 to %Node*
```

```
%fieldPtr81 = getelementptr %Node, %Node* %stuctObject78, i32 0, i32 1
store %Node* null, %Node** %fieldPtr81
%structAttPtr79 = getelementptr %Node, %Node* %stuctObject78, i32 0, i32 0
%loadedInt83 = load i32, i32* %value
store i32 %loadedInt83, i32* %structAttPtr79
store %Node* %stuctObject78, %Node** %newNode
%tempHead = call i8* @malloc(i96 12)
store %Node* null, %Node** %tempHead
%loadedStruct86 = load %LinkedList*, %LinkedList** %list
%attributePtr84 = getelementptr %LinkedList, %LinkedList* %loadedStruct86, i32 0, i32 0
%valPtr85 = load %Node*, %Node** %attributePtr84
store %Node* %valPtr85, %Node** %tempHead
%temp88 = load %Node*, %Node** %tempHead
%temp87 = icmp eq %Node* %temp88, null
br i1 %temp87, label %block91, label %block92

endBlock112:
%objectPtr114 = load %Node*, %Node** %current
%objectVal115 = load %Node*, %Node** %newNode
%attributePtr113 = getelementptr %Node, %Node* %objectPtr114, i32 0, i32 1
store %Node* %objectVal115, %Node** %attributePtr113
br label %block116

block92:
%current = call i8* @malloc(i96 12)
store %Node* null, %Node** %current
%next = call i8* @malloc(i96 12)
store %Node* null, %Node** %next
%loadedStruct98 = load %LinkedList*, %LinkedList** %list
%attributePtr96 = getelementptr %LinkedList, %LinkedList* %loadedStruct98, i32 0, i32 0
%valPtr97 = load %Node*, %Node** %attributePtr96
store %Node* %valPtr97, %Node** %current
%loadedStruct101 = load %Node*, %Node** %current
%attributePtr99 = getelementptr %Node, %Node* %loadedStruct101, i32 0, i32 1
%valPtr100 = load %Node*, %Node** %attributePtr99
store %Node* %valPtr100, %Node** %next
br label %condBlock110
br label %block116

block91:
%objectPtr94 = load %LinkedList*, %LinkedList** %list
%objectVal95 = load %Node*, %Node** %newNode
%attributePtr93 = getelementptr %LinkedList, %LinkedList* %objectPtr94, i32 0, i32 0
store %Node* %objectVal95, %Node** %attributePtr93
br label %block116

block116:
ret void

condBlock110:
%temp103 = load %Node*, %Node** %next
%temp102 = icmp ne %Node* %temp103, null
```

```
br i1 %temp102, label %loopBlock111, label %endBlock112

loopBlock111:
%objectVal106 = load %Node*, %Node** %next
store %Node* %objectVal106, %Node** %current
%loadedStruct109 = load %Node*, %Node** %current
%attributePtr107 = getelementptr %Node, %Node* %loadedStruct109, i32 0, i32 1
%valPtr108 = load %Node*, %Node** %attributePtr107
store %Node* %valPtr108, %Node** %next
br label %condBlock110

}
define i32 @main() {
entry:
%list = call i8* @malloc(i64 8)
store %LinkedList* null, %LinkedList** %list
%current = call i8* @malloc(i96 12)
store %Node* null, %Node** %current
%prev = call i8* @malloc(i96 12)
store %Node* null, %Node** %prev
%next = call i8* @malloc(i96 12)
store %Node* null, %Node** %next
%value = alloca i32, align 4
store i32 0, i32* %value
%length = alloca i32, align 4
store i32 0, i32* %length
%i = alloca i32, align 4
store i32 0, i32* %i
%struct3 = call i8* @malloc(i64 8)
%stuctObject1 = bitcast i8* %struct3 to %LinkedList*
%fieldPtr4 = getelementptr %LinkedList, %LinkedList* %stuctObject1, i32 0, i32 0
store %Node* null, %Node** %fieldPtr4
store %LinkedList* %stuctObject1, %LinkedList** %list
%strConstPtr1 = getelementptr [29 x i8], [29 x i8]* @.strConst1, i32 0, i32 0
call void @println_str(i8* %strConstPtr1)
call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.strIntFormat, i64 0, i64
0), i32* %length)
store i32 0, i32* %i
%strConstPtr2 = getelementptr [19 x i8], [19 x i8]* @.strConst2, i32 0, i32 0
call void @println_str(i8* %strConstPtr2)
br label %condBlock17

loopBlock53:
%loadedStruct46 = load %Node*, %Node** %current
%attributePtr44 = getelementptr %Node, %Node* %loadedStruct46, i32 0, i32 1
%valPtr45 = load %Node*, %Node** %attributePtr44
store %Node* %valPtr45, %Node** %next
%objectPtr48 = load %Node*, %Node** %current
%objectVal49 = load %Node*, %Node** %prev
%attributePtr47 = getelementptr %Node, %Node* %objectPtr48, i32 0, i32 1
store %Node* %objectVal49, %Node** %attributePtr47
%objectVal50 = load %Node*, %Node** %current
```

```
store %Node* %objectVal50, %Node** %prev
%objectVal51 = load %Node*, %Node** %next
store %Node* %objectVal51, %Node** %current
br label %condBlock52

endBlock36:
%strConstPtr5 = getelementptr [4 x i8], [4 x i8]* @.strConst5, i32 0, i32 0
call void @println_str(i8* %strConstPtr5)
%loadedStruct39 = load %LinkedList*, %LinkedList** %list
%attributePtr37 = getelementptr %LinkedList, %LinkedList* %loadedStruct39, i32 0, i32 0
%valPtr38 = load %Node*, %Node** %attributePtr37
store %Node* %valPtr38, %Node** %current
br label %condBlock52
ret i32 0

condBlock34:
%temp24 = load %Node*, %Node** %current
%temp23 = icmp ne %Node* %temp24, null
br i1 %temp23, label %loopBlock35, label %endBlock36

loopBlock73:
%loadedStruct67 = load %Node*, %Node** %current
%attributePtr65 = getelementptr %Node, %Node* %loadedStruct67, i32 0, i32 0
%loadedInt66 = load i32, i32* %attributePtr65
store i32 %loadedInt66, i32* %value
%loadedInt68 = load i32, i32* %value
call void @print_int(i32 %loadedInt68)
%strConstPtr7 = getelementptr [5 x i8], [5 x i8]* @.strConst7, i32 0, i32 0
call void @print_str(i8* %strConstPtr7)
%loadedStruct71 = load %Node*, %Node** %current
%attributePtr69 = getelementptr %Node, %Node* %loadedStruct71, i32 0, i32 1
%valPtr70 = load %Node*, %Node** %attributePtr69
store %Node* %valPtr70, %Node** %current
br label %condBlock72

endBlock54:
%objectPtr56 = load %LinkedList*, %LinkedList** %list
%objectVal57 = load %Node*, %Node** %prev
%attributePtr55 = getelementptr %LinkedList, %LinkedList* %objectPtr56, i32 0, i32 0
store %Node* %objectVal57, %Node** %attributePtr55
%strConstPtr6 = getelementptr [23 x i8], [23 x i8]* @.strConst6, i32 0, i32 0
call void @println_str(i8* %strConstPtr6)
%loadedStruct60 = load %LinkedList*, %LinkedList** %list
%attributePtr58 = getelementptr %LinkedList, %LinkedList* %loadedStruct60, i32 0, i32 0
%valPtr59 = load %Node*, %Node** %attributePtr58
store %Node* %valPtr59, %Node** %current
br label %condBlock72
ret i32 0

condBlock52:
%temp41 = load %Node*, %Node** %current
%temp40 = icmp ne %Node* %temp41, null
```

```
br i1 %temp40, label %loopBlock53, label %endBlock54

loopBlock35:
%loadedStruct29 = load %Node*, %Node** %current
%attributePtr27 = getelementptr %Node, %Node* %loadedStruct29, i32 0, i32 0
%loadedInt28 = load i32, i32* %attributePtr27
store i32 %loadedInt28, i32* %value
%loadedInt30 = load i32, i32* %value
call void @print_int(i32 %loadedInt30)
%strConstPtr4 = getelementptr [5 x i8], [5 x i8]* @.strConst4, i32 0, i32 0
call void @print_str(i8* %strConstPtr4)
%loadedStruct33 = load %Node*, %Node** %current
%attributePtr31 = getelementptr %Node, %Node* %loadedStruct33, i32 0, i32 1
%valPtr32 = load %Node*, %Node** %attributePtr31
store %Node* %valPtr32, %Node** %current
br label %condBlock34

loopBlock18:
%v = alloca i32, align 4
store i32 0, i32* %v
call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.strIntFormat, i64 0, i64
0), i32* %v)
%v9 = load i32, i32* %v, align 4
call void @Append(%LinkedList* %list, i32 %v9)
%temp11 = alloca i32, align 4
store i32 1, i32* %temp11
%temp13 = load i32, i32* %i, align 4
%temp14 = load i32, i32* %temp11, align 4
%temp12 = add i32 %temp13, %temp14
%temp15 = alloca i32, align 4
store i32 %temp12, i32* %temp15
%loadedInt16 = load i32, i32* %temp15
store i32 %loadedInt16, i32* %i
br label %condBlock17

condBlock17:
%temp6 = load i32, i32* %i, align 4
%temp7 = load i32, i32* %length, align 4
%temp5 = icmp slt i32 %temp6, %temp7
br i1 %temp5, label %loopBlock18, label %endBlock19

endBlock19:
%strConstPtr3 = getelementptr [23 x i8], [23 x i8]* @.strConst3, i32 0, i32 0
call void @println_str(i8* %strConstPtr3)
%loadedStruct22 = load %LinkedList*, %LinkedList** %list
%attributePtr20 = getelementptr %LinkedList, %LinkedList* %loadedStruct22, i32 0, i32 0
%valPtr21 = load %Node*, %Node** %attributePtr20
store %Node* %valPtr21, %Node** %current
br label %condBlock34
ret i32 0

condBlock72:
```

```
%temp62 = load %Node*, %Node** %current
%temp61 = icmp ne %Node* %temp62, null
br i1 %temp61, label %loopBlock73, label %endBlock74

endBlock74:
%strConstPtr8 = getelementptr [4 x i8], [4 x i8]* @.strConst8, i32 0, i32 0
call void @println_str(i8* %strConstPtr8)
ret i32 0

}
```

Код 4 алгоритма инвертирования связанного списка (LLVM IR)

ЗАКЛЮЧЕНИЕ

В заключение, в этой работе описаны основные этапы работы компилятора. Также был описан LLVM и используемые инструменты для лексического и синтаксического анализа. Разработан прототип компилятора языка программирования Go с использованием языка программирования Swift и библиотек SwiLex и SwiParse. Компилятор генерирует код LLVM IR в качестве выходных данных. Работа разработанного компилятора была описана в виде дриаграмм IDEF0. Наконец, компилятор был протестирован путем компиляции и выполнения алгоритма Bubble sort и алгоритма инвертирования связанного списка.

СПИСОК ЛИТЕРАТУРЫ

1. АХО А.В, ЛАМ М.С., СЕТИ Р., УЛЬМАН Дж.Д. Компиляторы: принципы, технологии и инструменты. – М.: Вильямс, 2008.
2. Aho A. V., Ullman J. D. The theory of parsing, translation, and compiling January 1972.
3. SwiLex // github URL: https://github.com/yassram/SwiLex
4. SwiParse // github URL: https://github.com/yassram/SwiParse
5. The LLVM Compiler Infrastructure Project // llvm URL: https://llvm.org/
6. GO // go URL: https://go.dev/

ПРИЛОЖЕНИЕ

```
program : defList

defList : function defList
        | structDefinition defList
        | function
        | structDefinition

function : funcKeyword identifier lParenthesis parametersList rParenthesis block
         | funcKeyword identifier lParenthesis rParenthesis block

structDefinition : typeKeyword identifier structKeyword structDefinitionBlock

structInit : identifier lCurlyParenthesis valueListSpecific rCurlyParenthesis
           | andSymbol identifier lCurlyParenthesis valueListSpecific rCurlyParenthesis
           | identifier lCurlyParenthesis rCurlyParenthesis
           | andSymbol identifier lCurlyParenthesis rCurlyParenthesis

parametersList : parameter
               | parameter colon parametersList

type : arrayValueType
     | valueType
     | identifier
     | star identifier
     | star valueType

arrayValueType : lSquareParenthesis rSquareParenthesis valueType

parameter : identifier type

block : lCurlyParenthesis statementsList rCurlyParenthesis

structDefinitionBlock : lCurlyParenthesis declarationStatementList rCurlyParenthesis

declarationStatementList : identifier type semicolon
                         | identifier type semicolon declarationStatementList

statementsList : statement semicolon
               | statement semicolon statementsList

statement : assignationStatement
          | declarationStatement
          | functionCallStatement
          | forStatement
          | ifStatement
          | ifElseStatement
          | printStatement
          | scanStatement
```

```
printStatement : fmt dot printLn lParenthesis valueList rParenthesis
               | fmt dot print lParenthesis valueList rParenthesis

scanStatement : fmt dot scanln lParenthesis pointerValue rParenthesis

value : valueScalar
      | valueArray

pointerValue : andSymbol identifier
             | andSymbol arrayPositionValue

arrayPositionValue : identifier lSquareParenthesis number rSquareParenthesis
                   | identifier lSquareParenthesis identifier rSquareParenthesis

booleanOperator : more
                | less
                | moreEquals
                | lessEquals
                | equalsEquals
                | exclamationEquals

valueScalar : number
            | nilKeyword
            | string

valueArray : lSquareParenthesis oneTypeValueList rSquareParenthesis

numberList : number colon numberList
           | number

stringList : string colon numberList
           | string

oneTypeValueList : numberList
                 | stringList

valueList : value colon valueList
          | identifier colon valueList
          | arrayValueType colon valueList
          | value
          | identifier
          | arrayValueType

valueListSpecific : identifier doubleDot value colon valueList
                  | identifier doubleDot identifier colon valueList
                  | identifier doubleDot arrayValueType colon valueList
                  | identifier doubleDot value
                  | identifier doubleDot identifier
                  | identifier doubleDot arrayValueType

arithmeticOperation : arithmeticOperation plus arithmeticOperationTerm
                    | arithmeticOperation minus arithmeticOperationTerm
```

```
                | arithmeticOperationTerm

arithmeticOperationTerm : arithmeticOperationTerm star arithmeticOperationFactor
                        | arithmeticOperationTerm slash arithmeticOperationFactor
                        | arithmeticOperationFactor

arithmeticOperationFactor : lParenthesis arithmeticOperation rParenthesis
                          | number
                          | identifier
                          | arrayPositionValue

booleanOperation : booleanOperation booleanOperator booleanOperationTerm
                 | booleanOperationTerm

booleanOperationTerm : identifier
                     | valueScalar

assignationStatement : identifier equals value
                     | identifier equals functionCallStatement
                     | identifier equals arithmeticOperation
                     | identifier equals arrayPositionValue
                     | identifier equals structInit
                     | identifier equals structAttribute
                     | structAttribute equals identifier
                     | arrayPositionValue equals value
                     | arrayPositionValue equals functionCallStatement
                     | arrayPositionValue equals arithmeticOperation
                     | arrayPositionValue equals arrayPositionValue

declarationStatement : varKeyword identifier type

functionCallStatement : identifier lParenthesis valueList rParenthesis

forStatement : forKeyword booleanOperation block

ifStatement : ifKeyword booleanOperation block

ifElseStatement : ifStatement elseKeyword block
```

Приложение А Грамматика