**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)»**

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

**Лабораторная работа № 2**

“Синтаксические деревья”

по дисциплине «Конструирование компиляторов»

Вариант 16

Работу выполнил

студент группы ИУ9-62Б

Сербин Денис

Москва, 2022

# **1. Цель работы**

# Целью данной работы является изучение представления синтаксических деревьев в памяти компилятора и приобретение навыков преобразования синтаксических деревьев.

# **2. Исходные данные**

В качестве исходного языка и языка реализации программы преобразования синтаксических деревьев выберем язык Go. Пакеты "go/token", "go/ast" и "go/parser" из стандарт ной библиотеки этого языка содержат готовый «front-end» компилятора языка Go, a пакет "go/format" восстанавливает исходный текст программы по её синтаксическому дереву. Документацию по этим пакетам можно посмотреть по адресу http://golang.org/pkg/go/. Построение синтаксического дерева по исходному тексту программы выполняется функцией parser.ParseFile, возвращающей указатель типа \*ast. File на корень дерева. Синтаксические деревья в памяти представляются значениями структур из пакета "go/ast". Изучать синтаксические деревья удобно по их листингам, порождаемым функцией ast. Fprint. Небольшая программа astprint, которая, ко всему прочему, демонстрирует вызов парсера для построения синтаксического дерева программы, представлена на листинге 1. Напомним, что для компиляции программы astprint нужно выполнить команду

go build astprint.go

Обход синтаксического дерева в глубину реализован в функции ast. Inspect, которая вы зывает переданную ей в качестве параметра функцию для каждого посещённого узла дерева. С помощью этой функции удобно осуществлять поиск узлов определенного типа в дереве. Например, представленная на листинге 2 функция insertHello выполняет поиск всех опера торов if в дереве и вставляет в начало положительной ветки каждого найденного оператора печать строки "hello".

Восстановление исходного текста программы из синтаксического дерева осуществляется функцией format.Node. Эта функция не обращает внимания на координаты узлов дерева, выполняя полное переформатирование текста программы, поэтому при преобразовании дерева координаты новых узлов прописывать не нужно.

# **3. Задание**

Выполнение лабораторной работы состоит из нескольких этапов:

1. подготовка исходного текста демонстрационной программы, которая в дальнейшем будет выступать в роли объекта преобразования (демонстрационная программа должна размещаться в одном файле и содержать функцию main);

2. компиляция и запуск программы astprint для изучения структуры синтаксического дерева демонстрационной программы:

3. разработка программы, осуществляющей преобразование синтаксического дерева и порождение по нему новой программы;

4. тестирование работоспособности разработанной программы на исходном тексте демонстрационной программы.

**4. Индивидуальный вариант**

Заменить вхождения глобальной константы FUNCNAME на имя неанонимной функции, в которой константы упоминается, либо на "(global)", если константа упоминается в глобальном контексте. В исходной программе константа FUNCNAME должна быть объявлена в глобальной области видимости типа string (в противном случае следует выдать ошибку), в новой программе она должна отсутстовать.

# **5. Результат выполнения**

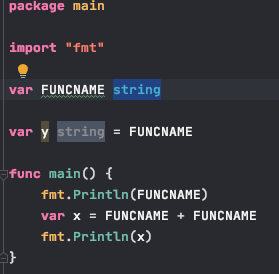


Рисунок 1. Исходный текст демонстрационной программы

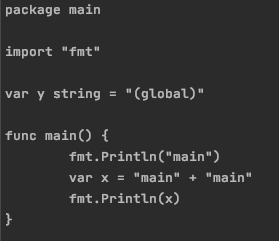


Рисунок 2. Тестирование работоспособности разработанной программы на исходном тексте демонстрационной программы

# **6. Вывод**

В рамках данной лабораторной работы был реализована программа, осуществляющая преобразование синтаксического дерева и порождение по нему новой программы.