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 9

Работу выполнил

студент группы ИУ9-62Б

Жук Дмитрий

Москва, 2022

Цель работы

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

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.

Индивидуальный вариант

Динамические срезы, создаваемые функцией make, должны иметь в два раза большую вместимость, чем в исходной программе (при этом их длина меняться не должна).

Реализация

```
package main

import (
    "fmt"
    "go/ast"
    "go/format"
    "go/parser"
    "go/token"
    "os"
)

func mulTwo(node *ast.Expr) *ast.BinaryExpr {
    return &ast.BinaryExpr {
        X: *node,
        Op: token.MUL,
        Y: &ast.BasicLit {
            Kind: token.INT,
            Value: "2",
        },
    }
}

func insertMulTwoForMake(file *ast.File, fset *token.FileSet) {
    ast.Inspect(file, func(node ast.Node) bool {
        // fmt.Println("=")

        if callExpr, ok := node.(*ast.CallExpr); ok {
            fun := callExpr.Fun;
            if ident, ok := fun.(*ast.Ident); ok && ident.Name == "make" {
                // ast.Fprint(os.Stdout, fset, callExpr, nil)
                if len(callExpr.Args) == 2 {
                    callExpr.Args = append(callExpr.Args,
mulTwo(&callExpr.Args[1]));
                } else {
                    callExpr.Args[2] = mulTwo(&callExpr.Args[2]);
                }
                // ast.Fprint(os.Stdout, fset, callExpr, nil)
            }
        }

        return true
    })
}

func main() {
    if len(os.Args) != 2 {
        return
    }

    fset := token.NewFileSet()
    if file, err := parser.ParseFile(fset, os.Args[1], nil,
parser.ParseComments); err == nil {
        insertMulTwoForMake(file, fset)
```

```
        if format.Node(os.Stdout, fset, file) != nil {
            fmt.Printf("Formatter error: %v\n", err)
        }
        //ast.Fprint(os.Stdout, fset, file, nil)
    } else {
        fmt.Printf("Errors in %s\n", os.Args[1])
    }
}
```

Листинг 1 — Код программы

```
package main

import "fmt"

func main() {

    s := make([]string, 3)
    len1 := 3;
    p := make([]string, len1, len1*2)
    fmt.Println("emp:", s)
    fmt.Println("emp:", p)

    s[0] = "a"
    s[1] = "b"
    s[2] = "c"
    fmt.Println("set:", s)
    fmt.Println("get:", s[2])

    fmt.Println("len:", len(s))

    s = append(s, "d")
    s = append(s, "e", "f")
    fmt.Println("apd:", s)

    var c = make([]string, len(s) + 1)
    copy(c, s)
    fmt.Println("cpy:", c)

    l := s[2:5]
    fmt.Println("sl1:", l)

    l = s[:5]
    fmt.Println("sl2:", l)

    l = s[2:]
    fmt.Println("sl3:", l)

    t := []string{"g", "h", "i"}
    fmt.Println("dcl:", t)

    twoD := make([][]int, 3)
    for i := 0; i < 3; i++ {
        innerLen := i + 1
        twoD[i] = make([]int, innerLen)
```

```
        for j := 0; j < innerLen; j++ {
            twoD[i][j] = i + j
        }
    }
    fmt.Println("2d: ", twoD)
}
```

Листинг 3 — Программа для тестирования изменений

Вывод

В ходе лабораторной работы было изучено представление синтаксических деревьев в памяти компилятора и приобретен навык преобразования синтаксических деревьев.