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

Вариант 7

Работу выполнил

студент группы ИУ9-61

Бакланова А.Д.

Москва, 2020

Цель работы:

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

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 стандартной библиотеки этого языка содержат готовый «front-end» компилятора языка Go, а 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 выполянется 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 вызывает переданную ей в качестве параметра функцию для каждого посещённого узла дерева. С помощью этой функции удобно осуществлять поиск узлов определённого типа в дереве. Например, представленная на листинге 2 функция `insertHello` выполняет поиск всех операторов `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.

Индивидуальный вариант:

7	Любая неанонимная функция, имеющая ровно один параметр типа `int`, должна выводить своё имя и значение параметра.

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.

Преобразование синтаксического дерева должно вносить в преобразуемую программу дополнительные возможности, перечисленные в таблице 1, 2 и 3.

Реализация:

Демонстрационной программой является файл demo-1.go

```
▶ ./astprint demo-1.go
     0  *ast.File {
     1  .  Doc: nil
     2  .  Package: demo-1.go:1:1
     3  .  Name: *ast.Ident {
     4  .  .  NamePos: demo-1.go:1:9
     5  .  .  Name: "main"
     6  .  .  Obj: nil
     7  .  }
     8  .  Decls: []ast.Decl (len = 4) {
     9  .  .  0: *ast.GenDecl {
    10  .  .  .  Doc: nil
    11  .  .  .  TokPos: demo-1.go:3:1
    12  .  .  .  Tok: import
    13  .  .  .  Lparen: demo-1.go:3:8
    14  .  .  .  Specs: []ast.Spec (len = 6) {
    15  .  .  .  .  0: *ast.ImportSpec {
    16  .  .  .  .  .  Doc: nil
    17  .  .  .  .  .  Name: nil
    18  .  .  .  .  .  Path: *ast.BasicLit {
    19  .  .  .  .  .  .  ValuePos: demo-1.go:4:2
    20  .  .  .  .  .  .  Kind: STRING
    21  .  .  .  .  .  .  Value: "\"fmt\""
```

Разработка программы, осуществляющей преобразование синтаксического дерева и порождение по нему новой программы

```
func insertNameAndVal(file *ast.File) {
    var funName string;
    var funParam string;
    ast.Inspect(file, func(node ast.Node) bool {
        if FuncDecl, ok := node.(*ast.FuncDecl); ok {
            if len(FuncDecl.Type.Params.List) == 1 {
                if a, ok:= FuncDecl.Type.Params.List[0].Type.(*ast.Ident); ok {
                    if a.Name =="int" {

                        funParam = FuncDecl.Type.Params.List[0].Names[0].Name;
                        funName = FuncDecl.Name.Name;

                        FuncDecl.Body.List = append(
                            []ast.Stmt{
                                &ast.ExprStmt{
                                    X: &ast.CallExpr{
                                        Fun: &ast.SelectorExpr{
                                            X:   ast.NewIdent("fmt"),
                                            Sel: ast.NewIdent("Printf"),
                                        },
                                        Args: []ast.Expr{
                                            &ast.BasicLit{
                                                Kind:  token.INT,
                                                Value: "\""+funName+"\"",
                                            },
                                        },
                                    },
                                },
                            },
                            FuncDecl.Body.List...,
                        )
                        FuncDecl.Body.List = append(
                            []ast.Stmt{
                                &ast.ExprStmt{
                                    X: &ast.CallExpr{
                                        Fun: &ast.SelectorExpr{
                                            X:   ast.NewIdent("fmt"),
                                            Sel: ast.NewIdent("Printf"),
                                        },
                                        Args: []ast.Expr{
                                            &ast.BasicLit{
                                                Kind:  token.INT,
                                                Value: "\"%d\","+funParam,
                                            },
                                        },
                                    },
                                },
                            },
                            FuncDecl.Body.List...,
                        )
                    };
                };
                //fmt.Printf("%s",Type.Params.List[0]);
        };
        return true
    })
}
```

Тестирование работоспособности программы на исходном тексте демонстрационной программы:

```
76    func test(a int) {
77
78    }
```

```
func test(a int) {
    fmt.Printf("%d", a)
    fmt.Printf("test")
}
func insertNameAndVal(file *ast.File) {
    var funName string
    var funParam string
    ast.Inspect(file, func(node ast.Node) bool {
        if FuncDecl, ok := node.(*ast.FuncDecl); ok {
            if len(FuncDecl.Type.Params.List) == 1 {
```

Т.е. преобразования появились только у функций, содержащих 1 входной параметр типа int.

Вывод:

В результате выполнения лабораторной работы были получены знания о представлении синтаксических деревьев в памяти компилятора, также было преобразовано синтаксическое дерево путем добавления информации о названии функции и значения ее входного параметра.