Лабораторная работа № 2.1. Синтаксические деревья

25 февраля 2025 г.

Александр Старовойтов, ИУ9-61Б

Цель работы

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

Индивидуальный вариант

Каждый оператор for вида `for  expr  {...}` преобразовать в оператор `for fmt.Println("init"); expr; fmt.Println("next") {...}`.

Реализация

Демонстрационная программа:

```
package main

import (
    "fmt"
)

func main() {
    for i := 0; i < 5; i++ {
        fmt.Println("cycle with init and post")
    }

    for {
        break
    }

    i := 0
```

```
    for i < 5 {
        fmt.Println(i)
        i += 1
        j := 2
        for j >= 0 {
            fmt.Println("    ", j)
            j -= 1
        }
    }
}
```

Программа, осуществляющая преобразование синтаксического дерева:

```
package main

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

// Init: *ast.ExprStmt {
//     83  .  .  .  .  .  .  .  X: *ast.CallExpr {
//     84  .  .  .  .  .  .  .  .  Fun: *ast.SelectorExpr {
//     85  .  .  .  .  .  .  .  .  .  X: *ast.Ident {
//     86  .  .  .  .  .  .  .  .  .  .  NamePos: main.go:9:6
//     87  .  .  .  .  .  .  .  .  .  .  Name: "fmt"
//     88  .  .  .  .  .  .  .  .  .  .  Obj: nil
//     89  .  .  .  .  .  .  .  .  .  }
//     90  .  .  .  .  .  .  .  .  .  Sel: *ast.Ident {
//     91  .  .  .  .  .  .  .  .  .  .  NamePos: main.go:9:10
//     92  .  .  .  .  .  .  .  .  .  .  Name: "Println"
//     93  .  .  .  .  .  .  .  .  .  .  Obj: nil
//     94  .  .  .  .  .  .  .  .  .  }
//     95  .  .  .  .  .  .  .  .  }
//     96  .  .  .  .  .  .  .  .  Lparen: main.go:9:17
//     97  .  .  .  .  .  .  .  .  Args: []ast.Expr (len = 1) {
//     98  .  .  .  .  .  .  .  .  .  0: *ast.BasicLit {
//     99  .  .  .  .  .  .  .  .  .  .  ValuePos: main.go:9:18
//    100  .  .  .  .  .  .  .  .  .  .  Kind: STRING
//    101  .  .  .  .  .  .  .  .  .  .  Value: "\"oao\""
//    102  .  .  .  .  .  .  .  .  .  }
//    103  .  .  .  .  .  .  .  .  }
//    104  .  .  .  .  .  .  .  .  Ellipsis: -
```

```
//    105  .  .  .  .  .  .  .  .  Rparen: main.go:9:23
//    106  .  .  .  .  .  .  .  }
//    107  .  .  .  .  .  .  }


func insertInitNext(file *ast.File) {
    ast.Inspect(file, func(node ast.Node) bool {
        if forStmt, ok := node.(*ast.ForStmt); ok {
            if forStmt.Init != nil || forStmt.Post != nil {
                return true
            }
            forStmt.Init = &ast.ExprStmt{
                X: &ast.CallExpr{
                    Fun:      &ast.SelectorExpr{
                        X:   ast.NewIdent("fmt"),
                        Sel: ast.NewIdent("Println"),
                    },
                    Args:     []ast.Expr{
                        &ast.BasicLit{
                            Kind:     token.STRING,
                            Value:    `"init"`,
                        },
                    },
                },
            }
            forStmt.Post = &ast.ExprStmt{
                X: &ast.CallExpr{
                    Fun:      &ast.SelectorExpr{
                        X:   ast.NewIdent("fmt"),
                        Sel: ast.NewIdent("Println"),
                    },
                    Args:     []ast.Expr{
                        &ast.BasicLit{
                            Kind:     token.STRING,
                            Value:    `"next"`,
                        },
                    },
                },
            }
        }
        return true
    })
}

func main() {
    if len(os.Args) != 3 {
```

```
        fmt.Printf("usage: lab2 <in.go> <out.go>\n")
        os.Exit(1)
    }

    fset := token.NewFileSet()

    file, err := parser.ParseFile(
        fset,                // данные об исходниках
        os.Args[1],          // имя файла с исходником программы
        nil,                 // пусть парсер сам загрузит исходник
        parser.ParseComments, // приказываем сохранять комментарии
    )

    if err != nil {
        fmt.Printf("Error: %v", err)
        os.Exit(1)
    }

    insertInitNext(file)

    outFile, err := os.Create(os.Args[2])
    if err != nil {
        fmt.Printf("Error: %v", err)
        os.Exit(1)
    }

    err = format.Node(outFile, fset, file)
    if err != nil {
        fmt.Printf("Error: %v", err)
        os.Exit(1)
    }
}
```

Тестирование

Результат трансформации демонстрационной программы:

```
package main

import (
    "fmt"
)

func main() {
    for i := 0; i < 5; i++ {
```

```
        fmt.Println("cycle with init and post")
    }

    for fmt.Println("init"); ; fmt.Println("next") {
        break
    }

    i := 0
    for fmt.Println("init"); i < 5; fmt.Println("next") {
        fmt.Println(i)
        i += 1
        j := 2
        for fmt.Println("init"); j >= 0; fmt.Println("next") {
            fmt.Println("    ", j)
            j -= 1
        }
    }
}
```

Вывод тестового примера на `stdout` (если необходимо)

```
cycle with init and post
cycle with init and post
cycle with init and post
cycle with init and post
cycle with init and post
init
init
0
init
     2
next
     1
next
     0
next
next
1
init
     2
next
     1
next
     0
next
next
2
```

```
init
     2
next
     1
next
     0
next
next
3
init
     2
next
     1
next
     0
next
next
4
init
     2
next
     1
next
     0
next
next
```

Вывод

Изучил представления синтаксических деревьев в памяти компилятора языка Go. Приобрел навык преобразования синтаксических деревьев для внесения в программу дополнительные возможности.