# Лабораторная работа № 2.1. Синтаксические деревья

21 февраля 2023 г.

Самохвалова Полина, ИУ9-62Б

## Цель работы

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

## Индивидуальный вариант

Любая неанонимная функция должна в начале выполнения и перед возвратом (оператором return или при достижении конца блока) выводить слово “Starts” и “Ends”, соответственно, своё имя и время, прошедшее с начала выполнения программы (см. функции time.Now() и time.Since(), можно использовать defer).

## Реализация

Демонстрационная программа:

```
package main

import "fmt"

func square(x int) int {
    var y int
    y = x * x
    return y
}

func main() {
    var s int
    s = square(5)
```

```
    fmt.Println(s)
}
```

Программа, осуществляющая преобразование синтаксического дерева:

```
package main

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

func insertTimeDecl(file *ast.File) {
    file.Decls[0].(*ast.GenDecl).Specs = append(file.Decls[0].(*ast.GenDecl).Specs,
        &ast.ImportSpec{
        Doc:  nil,
        Name: nil,
        Path: &ast.BasicLit{
            ValuePos: 0,
            Kind: token.STRING,
            Value:    "\"time\"",
        },
        Comment: nil,
        EndPos: 0,
    })
}

func insertVarT(file *ast.File) {
    file.Decls = append(file.Decls, &ast.GenDecl{
        Doc:    nil,
        TokPos: 0,
        Tok:    token.VAR,
        Lparen: 0,
        Specs:  []ast.Spec {
            &ast.ValueSpec {
                Doc:     nil,
                Names:   []*ast.Ident {
                    {
                        NamePos: 0,
                        Name: "t",
                    },
                },
                Type: &ast.SelectorExpr {
```

```
                    X: &ast.Ident {
                        NamePos: 0,
                        Name: "time",
                    },
                    Sel: &ast.Ident {
                        NamePos: 0,
                        Name: "Time",
                    },
                },
                Values:  nil,
                Comment: nil,
            },
        },
        Rparen: 0,
    })
}

func insertStartEnd(file *ast.File) {
    ast.Inspect(file, func(node ast.Node) bool {
        if FuncDecl, ok := node.(*ast.FuncDecl); ok {
            Name := FuncDecl.Name.Name
            FuncDecl.Body.List = append([]ast.Stmt {
                &ast.DeferStmt {
                    Defer: 0,
                    Call: &ast.CallExpr {
                        Fun: &ast.SelectorExpr {
                            X: &ast.Ident {
                                NamePos: 0,
                                Name: "fmt",
                            },
                            Sel: &ast.Ident {
                                NamePos: 0,
                                Name: "Printf",
                            },
                        },
                        Lparen: 0,
                        Args: []ast.Expr {
                            &ast.BasicLit {
                                ValuePos: 0,
                                Kind: token.STRING,
                                Value: "\"Ends %s %v\\n\"",
                            },
                            &ast.BasicLit {
                                ValuePos: 0,
                                Kind: token.STRING,
                                Value: "\"" + Name + "\"",
```

```
                },
                &ast.CallExpr {
                    Fun: &ast.SelectorExpr {
                        X: &ast.Ident {
                            NamePos: 0,
                            Name: "time",
                        },
                        Sel: &ast.Ident {
                            NamePos: 0,
                            Name: "Since",
                        },
                    },
                    Lparen: 0,
                    Args: []ast.Expr {
                        &ast.Ident {
                            NamePos: 0,
                            Name: "t",
                        },
                    },
                    Ellipsis: 0,
                    Rparen: 0,
                },
            },
            Ellipsis: 0,
            Rparen: 0,
        },
    },
}, FuncDecl.Body.List...)
FuncDecl.Body.List = append([]ast.Stmt {
    &ast.ExprStmt {
        X: &ast.CallExpr {
            Fun: &ast.SelectorExpr {
                X: &ast.Ident {
                    NamePos: 0,
                    Name: "fmt",
                },
                Sel: &ast.Ident {
                    NamePos: 0,
                    Name: "Printf",
                },
            },
            Lparen: 0,
            Args: []ast.Expr {
                &ast.BasicLit {
                    ValuePos: 0,
                    Kind: token.STRING,
```

```
                    Value: "\"Starts %s %v\\n\"",
                },
                &ast.BasicLit {
                    ValuePos: 0,
                    Kind: token.STRING,
                    Value: "\"" + Name + "\"",
                },
                &ast.CallExpr {
                    Fun: &ast.SelectorExpr {
                        X: &ast.Ident {
                            NamePos: 0,
                            Name: "time",
                        },
                        Sel: &ast.Ident {
                            NamePos: 0,
                            Name: "Since",
                        },
                    },
                    Lparen: 0,
                    Args: []ast.Expr {
                        &ast.Ident {
                            NamePos: 0,
                            Name: "t",
                        },
                    },
                    Ellipsis: 0,
                    Rparen: 0,
                },
            },
            Ellipsis: 0,
            Rparen: 0,
        },
    },
}, FuncDecl.Body.List...)
if Name == "main" {
    FuncDecl.Body.List = append([]ast.Stmt {
        &ast.AssignStmt {
            Lhs: []ast.Expr {
                &ast.Ident {
                    NamePos: 0,
                    Name: "t",
                },
            },
            TokPos: 0,
            Tok: token.ASSIGN,
            Rhs: []ast.Expr {
```

```
                            &ast.CallExpr {
                                Fun: &ast.SelectorExpr {
                                    X: &ast.Ident {
                                        NamePos: 0,
                                        Name: "time",
                                    },
                                    Sel: &ast.Ident {
                                        NamePos: 0,
                                        Name: "Now",
                                    },
                                },
                                Lparen: 0,
                                Args: nil,
                                Ellipsis: 0,
                                Rparen: 0,
                            },
                        },
                    },
                }, FuncDecl.Body.List...)
            }
        }
        return true
    })
}

func main() {
    if len(os.Args) != 2 {
        fmt.Printf("usage: astprint <filename.go>\n")
        return
    }

    fset := token.NewFileSet()

    if file, err := parser.ParseFile(
        fset,
        os.Args[1],
        nil,
        parser.ParseComments,
    ); err == nil {
        insertTimeDecl(file)
        insertVarT(file)
        insertStartEnd(file)
        if format.Node(os.Stdout, fset, file) != nil {
            fmt.Printf("Format error: %v\n", err)
        }
        ast.Fprint(os.Stdout, fset, file, nil)
```

```
    } else {
        fmt.Printf("Error: %v", err)
    }
}
```

## Тестирование

Результат трансформации демонстрационной программы:

```
package main

import (
    "fmt"
    "time"
)

func square(x int) int {
    fmt.Printf("Starts %s %v\n", "square", time.Since(t))
    defer fmt.Printf("Ends %s %v\n", "square", time.Since(t))
    var y int
    y = x * x
    return y
}

func main() {
    t = time.Now()
    fmt.Printf("Starts %s %v\n", "main", time.Since(t))
    defer fmt.Printf("Ends %s %v\n", "main", time.Since(t))
    var s int
    s = square(5)
    fmt.Println(s)
}

var t time.Time
```

## Вывод

В результате выполнения лабораторной работы было изучено представление синтаксических деревьев в памяти компилятора, приобретены навыки преобразования синтаксических деревьев, была написана программа, осуществляющая преобразование синтаксического дерева, так, чтобы любая неанонимная функция в начале выполнения и перед возвратом выводила слово “Starts” и “Ends”, соответственно, своё имя и время, прошедшее с начала выполнения программы.