UNIVERSIDADE FEDERAL DE ALAGOAS

ANNA VITÓRIA SOARES QUEIROZ VASCONCELOS
LUCAS CARVALHO FLORES

COMPILADOR MINIPAR

Maceió
2024

ANNA VITÓRIA SOARES QUEIROZ VASCONCELOS
LUCAS CARVALHO FLORES

COMPILADOR MINIPAR

Projeto de um compilador desenvolvido para a disciplina de Compiladores, ministrada pelo professor Arturo Hernandez Dominguez.

Maceió
2024

Relatório de Implementação de Interpretador

Introdução:

O MiniPar foi criado como projeto final da disciplina de Compiladores, com o intuito de aplicar os conhecimentos adquiridos ao longo do curso. Desde os conceitos básicos de análise léxica e sintática até as técnicas mais avançadas de interpretação de linguagens de programação.

Execução do Projeto:

O programa foi desenvolvido em Python e está disponível em um repositório no GitHub. Para executá-lo, é necessário ter o Python 3.12 e o pipenv instalados no computador.

Configuração do Ambiente e Execução do Programa

1. **Instalação do Pipenv e Dependências**
   Primeiramente, instalamos o gerenciador de ambientes virtuais **pipenv**. Em seguida, utilizamos o comando abaixo para instalar todas as dependências necessárias para o projeto:

```
PS C:\Users\annav> pipenv install
```

2. **Execução do Programa**
   Para executar o programa corretamente, siga os passos abaixo:
   - **Gere a gramática**: Compile ou gere os arquivos de gramática necessários para a execução do interpretador.

```
PS C:\Users\annav> .\update_grammar
```

   - **Inicie o servidor**:

```
PS C:\Users\annav> pipenv run python server.py
```

   - **Execute o interpretador**: Execute o programa client.py com o arquivo de teste (exemplo test1.mini):

```
PS C:\Users\annav> pipenv run python client.py test1.mini
```

Testes de execução

Estaremos executando três testes para verificar a efetividade do programa, analisando o código e os resultados.

Teste 1

Na primeira linha é criado o canal no qual serão executadas as operações matemáticas. Depois dela temos a primeira operação, no caso do teste um é uma sequência. Por fim temos os resultados das operações solicitadas na ordem solicitada.

tests > test1.mini

```
1   chan (server,127.0.0.1:50007);
2   seq {
3       x = 1;
4       print(x);
5       y = 2;
6       print(y);
7       z = x + y;
8       print(z);
9   }
10  seq {
11      x = 3;
12      print(x);
13      y = 4;
14      print(y);
15      z = x + y;
16      print(z);
17  }
```

código do teste 1.

```
PS C:\Users\annav\OneDrive\Área de Trabalho\Pastas\CC\Compiladores 24.1\Projeto Minipar\minipar\minipar> python client.py .\tests\test1.mini
1
2
3
3
4
7
```

Resultados do teste 1.

Teste 2

No primeiro teste, avaliamos a efetividade do programa em executar a tarefa solicitada em sequência. Neste segundo teste, realizaremos a execução em paralelo. Novamente, um canal de execução é criado. Com o resultado, observamos que o programa realizou a execução de forma paralela, processando primeiro as operações mais simples, como x = 1 e x = 3, para depois calcular a soma do valor de z nas duas operações.

tests > test2.mini

```
1   chan (server,127.0.0.1:50007);
2   par {
3       x = 1;
4       print(x);
5       y = 2;
6       print(y);
7       z = x + y;
8       print(z);
9   }
10  par {
11      x = 3;
12      print(x);
13      y = 4;
14      print(y);
15      z = x + y;
16      print(z);
17  }
```

Código do teste 2.

```
PS C:\Users\annav\OneDrive\Área de Trabalho\Pastas\CC\Compiladores 24.1\Projeto Minipar\minipar\minipar> python client.py .\tests\test2.mini
1
2
3
4
5
7
```

Resultados do teste 2.

Teste 3

O objetivo do teste 3 é executar a sequência de comandos solicitada pelo professor no projeto: a thread 1 realiza um cálculo fatorial, enquanto a thread 2 calcula a série de Fibonacci, com ambas rodando em paralelo. Conforme o padrão, o código começa com a criação do canal de execução, seguido pelas operações a serem executadas e seus respectivos resultados.

tests > test6.mini

```
chan (server,127.0.0.1:50007);
par {
    n = 12;
    a = 0;
    b = 1;
    temp = 0;
    i = 1;
    while (i <= n) {
        print(a);
        temp = a + b;
        a = b;
        b = temp;
        i = i + 1;
    }
    print(a);
}
par {
    n = 10;
    resultado = 1;
    while (n > 1) {
        resultado = resultado * n;
        n = n - 1;
        print(resultado);
    }
    print(resultado);
}
```

Código do teste 3.

```
PS C:\Users\annav\OneDrive\Área de Trabalho\Pastas\CC\Compiladores 24.1\Projeto Minipar\minipar\minipar> python client.py .\tests\test6.mini
0
10
1
90
1
720
2
5040
3
30240
5
151200
604800
1814400
3628800
3628800
```

Resultados do teste 3.

Gramática

A gramática da linguagem do interpretador do MiniPar foi gerada pela biblioteca antlr4. e está contida na pasta *grammar*.

```
grammar MiniPar;

prog: channel (seq | par)* EOF;
channel: 'chan' '(' ID ',' ADDR ')' ';';
seq: 'seq' '{' stmt* '}';
par: 'par' '{' stmt* '}';
stmt: atribuicao
    | condicional
    | loop
    | print;
atribuicao: ID '=' expr ';';
condicional: IF '(' expr ')' '{' stmt* '}' condicional_else?;
condicional_else: ELSE '{' stmt* '}';
loop: WHILE '(' expr ')' '{' stmt* '}';
print: PRINT '(' expr ')' ';';
expr: expr op=('+'|'-') expr
    | expr op=('*'|'/') expr
    | expr op=('<'|'>'|'<='|'>='|'=='|'!=') expr
    | '(' expr ')'
    | ID
    | INT
    | STRING;

IF: 'if';
ELSE: 'else';
WHILE: 'while';
PRINT: 'print';

NEWLINE: [\r\n]+ -> skip;
WHITESPACE: [ \t]+ -> skip;
ID: [a-zA-Z] [a-zA-Z0-9]*;
INT: [0-9]+;
ADDR: [a-zA-Z.0-9]+ ':' [0-9]+;
STRING: '"' ~[\r\n"]* '"';
```

Conclusão

O MiniPar é uma implementação funcional dos conceitos de análise sintática, análise léxica e comunicação cliente-servidor. Apesar dos desafios que aconteceram durante o desenvolvimento, a equipe conseguiu sucesso na implementação, entregando um produto capaz de executar programas escritos na linguagem MiniPar de maneira eficiente e confiável.

Referências

GITHUB. ANTLR 4 Documentation. Disponível em: https://github.com/antlr/antlr4/blob/4.13.2/doc/index.md. Acesso em: 08 nov. 2024.

Link do Repositório

https://github.com/kallyous/Compiladores-MiniPar