

Engenharia Gramatical (1^o ano do MEI)
Analisador de Código Fonte
Relatório de Desenvolvimento

João Pereira
PG47325

Luís Vieira
PG47430

Pedro Barbosa
PG47577

25 de abril de 2022

Resumo

O presente trabalho prático, realizado no âmbito da unidade curricular de Engenharia Gramatical inserida no perfil de Engenharia de Linguagens, visa o desenvolvimento de um Analisador de Código para uma evolução de uma Linguagem de Programação Imperativa Simples (LPIS), definida no segundo trabalho prático da unidade curricular de Processamento de Linguagens, de acordo com os guiões práticos 2 e 3 e como proposto pelos docentes.

Este irá começar com uma introdução e contextualização sobre o tema. De seguida, o mesmo irá ser aprofundado através da sua caracterização, descrição e identificação dos seus principais requisitos, serão apresentados e discutidos a GIC, as estruturas e algoritmos de dados utilizados bem como todas as alternativas e decisões tomadas e problemas de implementação. Finalmente, serão apresentados alguns casos de teste realizados e debatidos os resultados obtidos através dos mesmos.

Conteúdo

1	Introdução	2
2	Análise e Especificação	4
2.1	Descrição informal do problema	4
2.2	Especificação de Requisitos	4
2.2.1	Reconhecimento da linguagem de programação	4
2.2.2	Processamento do código fonte	5
2.2.3	Geração e visualização dos resultados	6
3	Concepção/desenho da Resolução	7
3.1	GIC	7
3.2	Estruturas de Dados	8
3.3	Algoritmos	9
4	Codificação e Testes	11
4.1	Alternativas, Decisões e Problemas de Implementação	11
4.2	Testes realizados e Resultados	13
4.2.1	Teste 1	13
4.2.2	Teste 2	15
4.2.3	Teste 3	16
5	Conclusão	18
A	Código do Programa	19

Capítulo 1

Introdução

O desenvolvimento do tema proposto insere-se na continuação do estudo feito no contexto do primeiro trabalho prático. Este consistiu em investigar ferramentas avançadas de análise de código cujo propósito era: detetar situações que comprometem as boas práticas de codificação na linguagem utilizada ou que apresentam vulnerabilidades aquando da execução; avaliar o desempenho do programa de forma estática ou dinâmica; embelezar textualmente a escrita de um programa e sugerir formas mais eficientes de codificar sem alterar o significado do código fonte.

Desta forma, no que concerne ao segundo trabalho prático, o seu principal objetivo consiste em desenvolver um Analisador de Código para uma evolução da sua Linguagem de Programação Imperativa Simples (LPIS) (definida no segundo trabalho prático da unidade curricular de Processamento de Linguagens), designada por LPIS2, conforme proposto nos guiões práticos 2 e 3 desta unidade curricular e pelos docentes. O mesmo, serve também, para pôr em prática o conhecimento obtido nas aulas teórico-práticas sobre a utilização do *Parser* e dos *Visitors* do módulo de geração de processadores de linguagens *Lark.Interpreter*.

Posto isto, o problema apresentado compreende as seguintes funcionalidades: permite declarar variáveis atómicas e estruturadas, instruções condicionais e três variáveis de ciclo diferentes, sendo elas *for*, *while* e *repeat*; através do módulo *Lark.Interpreter* e com recurso a *Parser* e *Visitors* conceber uma ferramenta de análise de código da linguagem que gere em *HTML* um relatório com os resultados da mesma. Este relatório deve conter:

- a) Uma lista de todas as variáveis do programa indicando os casos de redeclaração e não declaração de variáveis, variáveis utilizadas mas não inicializadas e variáveis declaradas e nunca mencionadas;
- b) Total de variáveis declaradas *vs* Tipos de dados estruturados;
- c) Total de instruções que formam o corpo do programa, indicando o número de instruções de cada tipo seja atribuições, leitura e escrita, condicionais ou cíclicas;
- d) Total de situações em que estruturas de controlo surgem aninhadas noutras estruturas de controlo do mesmo tipo ou diferentes;
- e) Adicionar informações acerca da presença de *ifs* aninhados indicando os casos que podem ser substituídos por um só *if*.

Assim sendo, e tendo em consideração todos os tópicos acima abordados, este relatório visa ajudar a compreender e explicar todos os raciocínios e tomadas de decisão feitas de forma a conseguir realizar todas as tarefas pedidas neste projeto e, assim, atingir o resultado final desejado.

Estrutura do Relatório

Este relatório inicia-se no capítulo 1 onde é feita uma contextualização e enquadramento do tema em estudo e uma explicação do problema a ser tratado e os pressupostos a si inerentes.

Segue-se do capítulo 2 onde é feita uma descrição do problema sobre o qual o projeto assenta e uma posterior análise e planeamento detalhados com o intuito de identificar e especificar os requisitos necessários para o desenvolvimento da ferramenta, sendo eles entradas, resultados e formas de transformação.

Posteriormente, no capítulo 3 será explicada a concepção idealizada para a resolução do problema anteriormente mencionado abordando as diferentes estruturas de dados e algoritmos utilizados bem como a GIC idealizada e concebida.

Em seguida, no capítulo 4 serão debatidas as principais tomadas de decisão efetuadas bem como possíveis alternativas para as mesmas e alguns problemas enfrentados durante a implementação da solução. Neste capítulo também serão apresentados alguns resultados e testes.

Finalmente o relatório termina com o capítulo 5 onde é feita uma síntese do documento, uma análise crítica generalizada do trabalho e dos resultados obtidos e uma reflexão sobre o trabalho futuro.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

O principal desafio que se pretende enfrentar ao desenvolver este trabalho prático é a implementação de uma ferramenta de análise de código fonte. Pretende-se que esta ferramenta seja capaz de encontrar, detetar e sinalizar anomalias ou falhas no código e até mesmo fazer recomendações úteis para o desenvolvedor, como por exemplo, sugestões para a reescrita de código.

Com isto, o processo de desenvolvimento de código torna-se mais rico e produtivo do ponto de vista do programador, visto que terá acesso a um conjunto de informações importantes sobre o mesmo e poderá melhorar a sua eficácia no processo de desenvolvimento, aprendendo com os relatórios de erros e sugestões produzidos pela ferramenta, de forma automática.

A interpretação do código fonte é feita através de uma gramática independente do contexto (GIC), cuja função é representar a sintaxe de uma linguagem de programação e reconhecer o código fonte. Depois de reconhecido, o problema pode ser resolvido processando o código e detetando os possíveis erros, para posteriormente serem criados os relatórios.

O utilizador conseguirá visualizar os resultados através da consulta a páginas HTML, para rápida análise e visualização.

2.2 Especificação de Requisitos

Numa fase de análise e planeamento, é necessário identificar e especificar os requisitos para o desenvolvimento desta ferramenta. Em primeiro lugar é necessário dividir os requisitos em 3 categorias principais: reconhecimento da linguagem e do código desenvolvido pelo utilizador; processamento do código fonte reconhecido e por fim geração de resultados de acordo com o mesmo. Desta forma, para cada categoria referida anteriormente existem requisitos um pouco diferentes. Em baixo serão explicados cada um deles e porque é que são importantes.

2.2.1 Reconhecimento da linguagem de programação

Tendo em conta o reconhecimento da linguagem de programação, é fundamental **construir uma GIC** capaz de reconhecer toda a sua sintaxe, sabendo identificar um bloco de código válido para

a linguagem definida. Este requisito torna-se especialmente importante porque é uma base para o correto funcionamento da ferramenta. É no código fonte que existe toda a informação que se pretende analisar e portanto tem de ser analisado duma forma simples e correta.

A linguagem definida pela GIC tem de permitir a **declaração de variáveis** atómicas (inteiros, *doubles*, *strings*, booleanos) e também variáveis mais complexas estruturadas (conjuntos, listas, tuplos e dicionários). Deve também incorporar um conjunto de **instruções**, podendo estas ser de diferentes tipos como: leitura, escrita, atribuição, condicionais e cíclicas. Nas instruções cíclicas incluímos 3 variantes de ciclo: *for*, *while* e *repeat(n)*. Adicionalmente decidimos que a linguagem deveria permitir escrever comentários (ao estilo do C) em frente a cada declaração ou instrução, e que devem ser permitidas **operações** lógicas, relacionais e matemáticas. Nos capítulos seguintes a GIC será descrita com mais detalhe.

2.2.2 Processamento do código fonte

Depois do código fonte ser reconhecido pela GIC, é preciso processar toda a informação nele contida. De acordo com isto surge a seguinte lista de requisitos:

- Analisar as declarações das variáveis e extrair as suas informações;
- Armazenar todas as variáveis declaradas, assim como os seus tipos;
- Detetar casos em que uma variável é declarada mais do que uma vez;
- Detetar casos em que uma variável é declarada mas não lhe é atribuído nenhum valor (declarada mas não inicializada);
- Analisar todo o tipo de instruções;
- Verificar se as variáveis mencionadas nas instruções estão declaradas e se não foram inicializadas antes do uso;
- Verificar se há variáveis que foram declaradas mas nunca foram utilizadas no código;
- Encontrar situações de aninhamento das estruturas de controlo (ciclos ou ifs);
- Analisar qual o nível de aninhamento das instruções condicionais;
- Sugerir alternativas para juntar ifs aninhados, caso seja possível.

Tendo em conta as necessidades acima identificadas e especificadas, bem como o que se tem vindo a abordar nas aulas de Engenharia Gramatical, surge a oportunidade de utilizar o módulo de geração de processadores de linguagens **Lark.Interpreter** para processar o código fonte. Através da utilização deste módulo é possível visitar a árvore de *parsing* retornada pelo **Lark**, desde a raiz até às folhas, numa abordagem *top-down*, e de acordo com cada nó processar cada parte da gramática da forma pretendida. Este módulo é também um requisito fundamental nesta fase. Visto que os resultados do processamento devem ser persistidos até à fase de visualização dos relatórios, também é necessário utilizar estruturas de dados para armazenar todas as informações enunciadas na listagem de requisitos.

2.2.3 Geração e visualização dos resultados

Por fim, de forma a produzir um relatório de fácil acesso e compreensão, surgem requisitos para a produção de páginas HTML com um relatório dos resultados obtidos após o processamento. Nesta fase identificamos dois requisitos principais.

O primeiro é a disponibilização do código fonte numa página HTML com as devidas anotações (indicação de problemas com variáveis, informar sobre os níveis de aninhamento dos ifs, alertar para variáveis não declaradas, etc.).

O segundo é a criação de uma página HTML adicional com informações gerais sobre o código, relativamente às variáveis, total de instruções, contagem de instruções de cada tipo, informações mais detalhada sobre os aninhamentos e por fim alternativas para os ifs que estejam aninhados e possam ser substituídos por um só.

Para complementar estes requisitos também devem existir funções responsáveis por ler as estruturas de dados resultantes do processamento e transcrever todos os dados necessários para HTML, bem como escrever todo o código fonte devidamente anotado na página principal.

Capítulo 3

Concepção/desenho da Resolução

Para conceber a resolução começamos por nos focar em definir uma GIC que nos permitisse satisfazer todos os requisitos relacionados com a linguagem de programação, tendo sido estes anteriormente identificados. Além disso procuramos obter uma gramática simples e legível, para facilitar também o resto dos componentes da solução.

Posteriormente criámos uma classe denominada `LinguagemProgramacao` que contém um **Interpreter** que permite fazer todas as visitas aos nós da árvore, resultantes do *parsing* do código fonte. Como complemento auxiliar, utilizamos um conjunto de definições para criar as páginas html e para auxiliar em algumas tarefas do processamento. Esta classe contém o código necessário para representar o algoritmo principal que nos permite obter a solução esperada.

3.1 GIC

A GIC final que concebemos para a nossa resolução está incluída no apêndice. Esta é composta por dois blocos principais. O bloco de declarações deve ser a primeira coisa a surgir no código fonte, e posteriormente deve surgir um bloco de instruções. Só são aceites exemplos em que exista pelo menos uma declaração e pelo menos uma instrução, e portanto ambos os conjuntos possuem uma lista de um ou mais elementos.

Relativamente a cada uma das declarações, estas devem todas indicar o **tipo** da variável a ser declarada: INT, BOOL, STR, DOUBLE, LIST, SET, TUPLE ou DICT. A **variável** deve aparecer imediatamente a seguir ao tipo e é representada por uma sequência de letras, minúsculas ou maiúsculas, podendo incluir o `_`. Opcionalmente a declaração pode inicializar a variável, atribuindo um valor, ou somente declará-la sem qualquer valor. Se a **atribuição** aparecer então o valor atribuído deve pertencer à designação **logic**. Esta designação representa qualquer valor, sendo este uma operação lógica, relacional, aritmética ou um fator tal como indicado na gramática. O fator é a designação mais elementar e pode representar números, strings, doubles, booleanos e estruturas mais complexas como listas, tuplos, conjuntos e dicionários. Estas estruturas mais complexas aceitam que os seus elementos sejam de qualquer tipo, desde que encaixem na designação de fator.

A designação instrução, por sua vez, pode ser uma leitura, escrita, uma estrutura de controlo condicional, atribuição ou 3 variantes de estrutura de controlo cíclica.

A **leitura** representa a instrução que permite ler um valor do stdin e armazenar numa variável.

A variável que pode ser utilizada pode ser uma variável normal atômica, ou então um acesso a um campo de uma estrutura, existindo por isso a designação **chave** que representa o índice de acesso às estruturas mais complexas (lista e dicionário). A **escrita** representa a instrução que apresenta um valor no stdout, sendo por isso o seu conteúdo qualquer representação da designação logic. O **if** é acompanhado de uma condição, que é equivalente à designação logic e por isso aceita vários tipos de operações como condição. Opcionalmente este pode ter um else associado. Tanto no if como no else, surge um bloco de instruções dentro. Os ciclos podem ser **for**, **while** e **repeat**, sendo que os dois primeiros podem ter condições associadas e o último repete o corpo do ciclo **NUM** vezes (por exemplo repeat(10){...}). É de notar que todos os ciclos possuem um conjunto de instruções como corpo, permitindo os aninhamentos tal como nos ifs. Por fim, a **atribuição** representa a atualização do valor de uma variável (que pode ser uma variável atômica ou um elemento de uma lista, dicionário ou tuplo), com um novo valor do tipo logic.

3.2 Estruturas de Dados

A principal estrutura de dados utilizada para armazenar as informações do código fonte foi a variável de instância **self.output**. Esta variável é um dicionário que armazena todas as informações necessárias para obter os resultados pretendidos. Através dela é possível ter acesso a todo o conteúdo que depois é apresentado em HTML. Além disso, este dicionário é retornado pela classe sempre que se invoca o programa, para que os programadores também tenham acesso aos resultados de uma forma mais rápida (mas menos detalhada e legível), quando comparada à visualização dos dados nas páginas HTML.

No final da execução do programa, o dicionário self.output irá conter informação sobre **as variáveis declaradas**, **as variáveis não inicializadas**, os **erros** encontrados durante a análise do código (não declaração, redeclaração, variáveis utilizadas mas não declaradas, variáveis declaradas e nunca mencionadas), os **níveis de aninhamento dos ifs** pela ordem em que estes aparecem no código, **as variáveis que foram utilizadas** nas instruções, a **contagem de instruções total e por tipo**, o **total de situações em que uma estrutura de controlo está aninhada noutra** e por fim a **lista de alternativas para os ifs aninhados**, caso existam.

Como complemento ao dicionário principal, são também utilizadas outras estruturas que podem ser encontradas no código do apêndice A, na função `__init__` da classe. Essas estruturas são:

- **self.decls**: Dicionário que armazena todas as variáveis declaradas, com o seu nome como chave e o tipo como valor;
- **self.naoInicializadas**: Conjunto com os nomes de todas as variáveis não inicializadas;
- **self.utilizadas**: Conjunto com os nomes das variáveis utilizadas nas instruções;
- **self.erros**: Dicionário com conjuntos de variáveis que apresentam um erro, que pode ser não-declaração, redeclaração, usado mas não inicializado e declarada mas nunca usado;
- **self.dicinstrucoes**: Dicionário que tem a contagem total de instruções, bem como a contagem das instruções de cada tipo (atribuição, condicional, cíclica, leitura, escrita);

- **self.condicoesIfs**: Lista de condições dos ifs consecutivamente aninhados. Enquanto que os ifs forem consecutivamente aninhados, sem instruções entre si, esta lista armazena as suas condições para mais tarde gerar uma alternativa mais simples para o aninhamento;
- **self.alternativasIfs**: Lista de alternativas encontradas para situações de ifs aninhados. Se for possível substituir uma ou mais situações de aninhamento de ifs por uma estrutura condicional mais simples, é nesta lista que aparecem as substituições em formato de string, para posteriormente serem escritas no html;
- **self.niveisIfs**: Dicionário que armazena os níveis de aninhamento dos ifs. A chave é o nível de aninhamento, e o valor é uma lista de números. Os números representam a ordem em que o if aparece no código. Por exemplo, se existir a entrada 0: [1,2,5] no dicionário significa que o primeiro, segundo e quinto if a aparecer no código, pertencem ao nível 0 de aninhamento.

3.3 Algoritmos

Depois de levantados todos os requisitos da ferramenta, bem como a sua especificação, utilizamos um algoritmo capaz de resolver o problema, tirando partido das estruturas de dados apresentadas anteriormente e um conjunto de instruções bem definido. A primeira parte do algoritmo consiste em ler o código fonte a partir de um ficheiro (`codigoFonte.txt`) e gerar a árvore resultante do *parsing* utilizando o módulo **Lark**.

A fase seguinte consiste no processamento e visita aos nós da árvore, juntamente com a escrita do código num ficheiro HTML. A escrita no ficheiro **codigoAnotado.html** é feita ao mesmo tempo que o processamento, isto é, à medida que se vão visitando os diferentes nós da árvore, é processada a informação e o código é escrito no ficheiro html, podendo conter anotações ou permanecer igual ao ficheiro de *input*. O ficheiro **informacoesAdicionais.html** só é escrito no final do processamento, quando toda a informação já tiver sido recolhida. O algoritmo é maioritariamente influenciado pela classe `LinguagemProgramacao` e os seus métodos definem o processamento que cada nó da árvore faz. A junção de todos esses métodos permite que a abordagem geral para atingir a solução seja o processamento independente feito por cada um dos métodos, sendo que cada um deles processa a informação de forma especial e a retorna para os nós pai. De modo a entender o funcionamento geral de cada um dos métodos, é feita uma breve explicação em seguida:

linguagem

Este método é responsável, inicialmente, por visitar o nó das declarações, obtendo todo o processamento das mesmas. Em seguida é utilizada uma flag **self.nasInstrucoes** para indicar que se vai passar a visitar o nó das instruções, que fará todo o processamento da parte das instruções. Por fim, é criado o dicionário de output de acordo com os resultados obtidos e é feita a escrita da segunda página html.

declaracoes

Responsável por visitar cada uma das declarações (nós filho).

declaracao

Responsável por processar uma declaração, percorre todos os filhos e recolhe o tipo da variável, a variável, e o valor que possa ser atribuído. Tem em conta diferentes situações que possam ocorrer, e para tal faz verificações de redeclaração, não inicialização, se a variável é uma string, estrutura, etc. Para tal é necessário obter o tipo, a variável e o valor dos nós filho, que retornam essa informação. No final é também escrita a variável no dicionário **self.decls**.

instrucoes

Responsável por visitar cada uma das instruções (nós filho), e retornar o código correspondente em formato de string. Este retorno de informação é importante para se conseguir obter o corpo das instruções de controlo, especialmente dos ifs.

instrucao

Um dos **métodos mais complexos do algoritmo**. Processa o nó da instrução, armazena o nível de aninhamento do if, o nível de profundidade (utilizado para a indentação) e as condições dos ifs que podem ser aninhados. À medida que a instrução é analisada a variável resultado é preenchida, para depois retornar essa informação ao nó pai. Caso a instrução atualmente analisada seja um if, é visitado o nó da condição para a obter, e depois são feitas várias operações: adição do nível do if ao dicionário, verificação da existência de instruções dentro do if e verificação da existência de ifs aninhados. Sempre que existe um conjunto de instruções dentro do if, o nível de aninhamento é incrementado.

Caso um dos nós filho da instrução seja um token, este é processado de forma especial dependendo dos casos. Finalmente, se for do tipo *tree*, é visitado o nó. No final, ainda é feita uma verificação para o caso dos ifs aninhados, se a lista das condições condicionais tiver mais do que um elemento, significa que podemos juntá-las e obter um só if. Desta forma é preenchida a lista das alternativas para os ifs.

Resto dos métodos

Como o resto dos métodos são mais simples e seguem todos a mesma abordagem, podem ser explicados de uma forma mais simples. Cada um deles processa a informação do nó que está a visitar, e retorna para os nós pai, que fazem um processamento mais detalhado envolvendo as estruturas de dados. Estes métodos acabam por processar as condições, os componentes das estruturas de dados complexas, o conteúdo de algumas instruções mais simples que envolvam variáveis, etc. O **fator**, por sua vez, faz também verificações relativamente à utilização de variáveis e situações de erro, quando estas são utilizadas nas instruções.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Neste projeto adotamos abordagens diferentes entre o processamento das declarações e das instruções. Para as declarações decidimos visitar todos os nós filho para recolher as informações da declaração, e fazer o seu processamento ao nível da declaração, em vez de o fazer em níveis mais baixos da árvore. Isto permitiu tornar o código legível e funcional.

No entanto, para as instruções encontrámos alguns problemas em utilizar a mesma abordagem das declarações, especialmente devido à deteção de estruturas de controlo aninhadas e da escrita de alternativas para os ifs aninhados. Como nas estruturas de controlo podem existir outras instruções aninhadas, tornou-se muito complexo manter a abordagem. Com isto, a nossa abordagem foi dividir o processamento pelos vários níveis, em vez de o reunir todo na instrução, apesar deste último continuar a ser o nível mais complexo do algoritmo. Apesar desta abordagem, encontrámos uma vantagem na eficiência, visto que o processamento passa a ser feito em níveis mais baixos da árvore.

Um dos principais **problemas** da abordagem que adotamos foi o facto de ao longo do desenvolvimento do código, este se ter tornado cada vez mais complexo, principalmente nas funcionalidades de verificar aninhamentos nos ifs, e na criação de alternativas para a sua escrita. Inicialmente a estratégia era simples de implementar mas no final tornou-se mais complicada.

Relativamente às **decisões** tomadas na codificação, é apresentada em seguida uma lista das que consideramos relevantes apontar:

- Consideramos que o nível de aninhamento dos ifs deveria voltar a 0 caso aparecesse uma estrutura de controlo cíclica. Desta forma, os ifs só serão aninhados se estiverem uns dentro dos outros, e não tiverem instruções cíclicas entre eles;
- Consideramos que as situações de aninhamento seriam qualquer situação em que uma estrutura condicional ou cíclica surge dentro de outra estrutura de qualquer um desses tipos. Ou seja, se existir um *for* aninhado num *while*, e este último estiver aninhado num *if*, então há 2 situações de aninhamento no código.
- Consideramos que os ifs aninhados só poderiam ser reescritos por um if só, caso o aninhamento fosse constituído por ifs imediatamente seguidos, sem instruções entre eles, por exemplo:

```
1   if(a){  
2       if(b){  
3           if(c){  
4               print("1");  
5           }  
6       }  
7   }  
8
```

- Decidimos que se uma variável não fosse inicializada, mas estivesse a ser utilizada numa instrução **READ**, não seria gerado nenhum erro, visto que esta instrução indica que será atribuído um valor à variável.
- Decidimos que a nossa ferramenta não considera os valores atribuídos às variáveis, e portanto, não os armazena nas estruturas nem faz as operações aritméticas entre eles. Focamo-nos apenas na análise estática do código e nunca no que poderá acontecer em *runtime*.

4.2 Testes realizados e Resultados

Vamos agora apresentar testes que evidenciam os resultados finais do nosso interpretador, bem como as possibilidades e capacidades do mesmo.

4.2.1 Teste 1

Neste primeiro teste estão evidenciadas as redeclarações de variáveis, possíveis aninhamentos, uso de variáveis não declaradas e variáveis declaradas mas não utilizadas.

```
1 list l = [1,2,3];
2 string str = "s";
3 string str = "d";
4 print(str);
5 if (a){
6     input(c);
7     if(b){
8         if(c){
9             print("1");
10        }
11    }
12 }
13 if(str){
14     while(str > "1"){
15         print("t");
16         input(c);
17         c = c*2;
18     }
19 }
```

Listing 4.1: Teste Exemplo 1

Análise de código

```
list l = [1, 2, 3];
string str = "s";
string str = "d";

print(str);
if(a){
    input(c);
    if(b){
        if(c){
            print("1");
        }
    }
}
if(str){
    while(str>"1"){
        print("t");
        input(c);
        c=c*2;
    }
}
```

Figura 4.1: Análise do Código Teste 1

Variáveis declaradas e os seus tipos

Variável	Tipo
l	list
str	string

Outras informações sobre as variáveis

Variáveis sem inicialização: Nenhuma

Variáveis não declaradas: {'c', 'a', 'b'}

Variáveis redeclaradas: {'str'}

Variáveis usadas e não inicializadas: Nenhuma

Variáveis declaradas mas nunca mencionadas: {'t'}

Informações sobre as instruções

Total de instruções: 11

Total de atribuições: 1

Total de leituras: 2

Total de escritas: 3

Total de instruções condicionais: 4

Total de instruções cíclicas: 1

Total de situações de aninhamento: 3

Informações sobre os ifs e os seus aninhamentos

Níveis de aninhamento dos ifs

Nível 0: Ifs pela ordem em que aparecem no código - [1, 4]

Nível 1: Ifs pela ordem em que aparecem no código - [2]

Nível 2: Ifs pela ordem em que aparecem no código - [3]

Alternativa para os ifs aninhados

```
if(b && c){ print("1"); }
```

Figura 4.2: Informações Adicionais Teste 1

4.2.2 Teste 2

Já no segundo teste é evidenciado o uso de variáveis declaradas mas não inicializadas, bem como o uso de variáveis não declaradas, juntamente com aninhamento de ifs e demonstração de diversas instruções reconhecidas pelo interpretador.

```
1 int a;
2 int b;
3 a=12;
4 print(str);
5 if (a){
6     input(c);
7     if(b){
8         if(c){
9             print("1");
10        }
11    }
12    if(a<b){
13        while(a>0){
14            b = b + 1;
15        }
16        repeat(5){
17            print(f);
18        }
19    }
20 }
```

Listing 4.2: Código de Teste 2

Análise de código

```
int a;
int b;

a=12;
print(str);
if(a){
    input(c);
    if(b){
        if(c){
            print("1");
        }
    }
    if(a<b){
        while(a>0){
            b=b+1;
        }
        repeat(5){
            print(f);
        }
    }
}
```

Figura 4.3: Análise do Código Teste 2

Variáveis declaradas e os seus tipos

Variável	Tipo
a	int
b	int

Outras informações sobre as variáveis

Variáveis sem inicialização:	Nenhuma
Variáveis não declaradas:	{'f', 'str', 'c'}
Variáveis redeclaradas:	Nenhuma
Variáveis usadas e não inicializadas:	{'b'}
Variáveis declaradas mas nunca mencionadas:	Nenhuma

Informações sobre as instruções

Total de instruções:	12
Total de atribuições:	2
Total de leituras:	1
Total de escritas:	3
Total de instruções condicionais:	4
Total de instruções cíclicas:	2
Total de situações de aninhamento:	5

Informações sobre os ifs e os seus aninhamentos

Níveis de aninhamento dos ifs
Nível 0: Ifs pela ordem em que aparecem no código - [1]
Nível 1: Ifs pela ordem em que aparecem no código - [2, 4]
Nível 2: Ifs pela ordem em que aparecem no código - [3]
Alternativa para os ifs aninhados
if(b && c){ print("1"); }

Figura 4.4: Informações Adicionais Teste 2

4.2.3 Teste 3

Por fim, este último teste serve para evidenciar as possibilidades de aninhamento de ifs, um dos maiores obstáculos deste trabalho.

```
1 int s=2;
2 if(a){
3     if(b){
4         if(c){
5             print("1");
6         }
7     }
8     if(a<d){
9         if(d<c){
10            if(s==2){
11                if(True){
12                    print("False");
13                }
14            }
15        }
16    }
17 }
```

Listing 4.3: Código de Teste 3

Análise de código

```

int s = 2;

if(a){
    if(b){
        if(c){
            print("1");
        }
    }
    if(a<d){
        if(d<c){
            if(s==2){
                if(True){
                    print("False");
                }
            }
        }
    }
}

```

Figura 4.5: Análise do Código Teste 3

Variáveis declaradas e os seus tipos

Variável	Tipo
s	int

Outras informações sobre as variáveis

Variáveis sem inicialização:	Nenhuma
Variáveis não declaradas:	{'c', 'a', 'd', 'b'}
Variáveis redeclaradas:	Nenhuma
Variáveis usadas e não inicializadas:	Nenhuma
Variáveis declaradas mas nunca mencionadas:	Nenhuma

Informações sobre as instruções

Total de instruções:	9
Total de atribuições:	0
Total de leituras:	0
Total de escritas:	2
Total de instruções condicionais:	7
Total de instruções cíclicas:	0
Total de situações de aninhamento:	6

Informações sobre os ifs e os seus aninhamentos

Níveis de aninhamento dos ifs
Nível 0: Ifs pela ordem em que aparecem no código - [1]
Nível 1: Ifs pela ordem em que aparecem no código - [2, 4]
Nível 2: Ifs pela ordem em que aparecem no código - [3, 5]
Nível 3: Ifs pela ordem em que aparecem no código - [6]
Nível 4: Ifs pela ordem em que aparecem no código - [7]

Alternativa para os ifs aninhados
if(b && c){ print("1"); }
if(a < d && d < c && s == 2 && True){ print("False"); }

Figura 4.6: Informações Adicionais Teste 3

Capítulo 5

Conclusão

O presente documento visa apresentar de forma sucinta e concreta o projeto desenvolvido, desde a formulação do problema até à implementação final e respetivos resultados apresentados, tendo sempre como base a explicação do raciocínio utilizado para a resolução do problema em questão.

Reverendo todo o trabalho feito até aqui, o projeto desenvolvido encontra-se completo e a equipa acredita que este está ao nível que pretendia desde o seu início. Todos os resultados citados como objetivo no enunciado foram alcançados e realizados sempre da forma que se pensou ser a mais correta. Porém, conforme mencionado anteriormente, um dos maiores obstáculos enfrentados foi o aumento da complexidade do código à medida que o projeto foi avançando, principalmente no que concerne ao aninhamentos dos *ifs* e na conceção de alternativas para a sua escrita. Estes dois desafios acabaram por ser mais desafiantes do que o que era esperado na abordagem inicial ao problema tornando a implementação de uma solução final correta e completa mais complicada.

Tendo como foco um possível trabalho futuro, a complexidade do projeto poderia ser atenuada e melhorada no caso dos aninhamentos dos *ifs*. Tentaríamos desenhar uma solução que torná-se esta funcionalidade possível mesmo existindo uma instrução cíclica entre os *ifs* ou mesmo estes não se encontrarem no código consecutivamente, existindo instruções entre eles. Outro ponto seria também realizar algumas pequenas mudanças na GIC idealizada pelo grupo com o intuito de a tornar o mais geral e menos específica possível.

No âmbito geral, o grupo sente que desenvolveu o trabalho prático da melhor forma possível e conseguiu alcançar todos os objetivos propostos no mesmo.

Apêndice A

Código do Programa

Lista-se a seguir o código em Python do programa desenvolvido.

```
1 from errno import ESOCKETNOTSUPPOT
2 from lark import Lark,Token,Tree
3 from lark.tree import pydot__tree_to_png
4 from lark import Transformer
5 from lark.visitors import Interpreter
6 from lark import Discard
7
8 grammar = '''
9 linguagem: declaracoes instrucoes
10 declaracoes: comentario? declaracao PV comentario? (declaracao PV comentario?)*
11 declaracao: tipo VAR (ATRIB logic)?
12 instrucoes: instrucao comentario? (instrucao comentario?)*
13 instrucao: READ PE conteudoread PD PV
14 | PRINT PE logic PD PV
15 | IF PE condicao PD CE instrucoes CD (ELSE CE instrucoes CD)?
16 | FOR PE atribuicao PV condicao PV atribuicao PD CE instrucoes CD
17 | WHILE PE condicao PD CE instrucoes CD
18 | REPEAT PE NUM PD CE instrucoes CD
19 | atribuicao PV
20 comentario: C_COMMENT
21 atribuicao: VAR (PER chave PDR)? ATRIB logic
22 condicao: logic
23 conteudoread: VAR (PER chave PDR)?
24 logic: PE? logicnot AND logic PD? | PE? logicnot OR logic PD? | PE? logicnot PD?
25 logicnot: PE? NOT logic PD? | PE? relac PD?
26 relac: PE? logic EQ exp PD?
27 | PE? logic DIFF exp PD?
28 | PE? logic GRT exp PD?
29 | PE? logic GEQ exp PD?
30 | PE? logic LWR exp PD?
31 | PE? logic LEQ exp PD?
32 | PE? exp PD?
33 exp: PE? exp ADD termo PD?
34 | PE? exp SUB termo PD?
35 | PE? termo PD?
36 termo: PE? exp MUL termo PD?
37 | PE? exp DIV termo PD?
```

```

38 | PE? exp MOD termo PD?
39 | PE? factor PD?
40 factor: NUM
41 | BOOLEANO
42 | STRING
43 | NUMDOUBLE
44 | VAR (PER chave PDR)?
45 | PER conteudo? PDR
46 | CE conteudo especial? CD
47 | PE conteudo? PD
48 conteudo: factor (VIR factor)*
49 conteudo especial: conteudo dicionario | conteudo
50 conteudo dicionario: entrada (VIR entrada)*
51 entrada: STRING PP factor
52 tipo: INT | BOOL | STR | DOUBLE | LIST | SET | TUPLE | DICT
53 chave: NUM | STRING | VAR
54 BOOLEANO: "True" | "False"
55 NUM: ("0".."9")+
56 NUMDOUBLE: ("0".."9")+ "." ("0".."9")+
57 STRING: ESCAPED_STRING
58 INT: "int"
59 STR: "string"
60 BOOL: "bool"
61 DOUBLE: "double"
62 LIST: "list"
63 SET: "set"
64 TUPLE: "tuple"
65 DICT: "dict"
66 VIR: ","
67 PE: "("
68 PD: ")"
69 PER: "["
70 PDR: "]"
71 CE: "{"
72 CD: "}"
73 PV: ";"
74 PP: ":"
75 ADD: "+"
76 SUB: "-"
77 DIV: "/"
78 MUL: "*"
79 MOD: "%"
80 EQ: "=="
81 DIFF: "!="
82 GRT: ">"
83 GEQ: ">="
84 LWR: "<"
85 LEQ: "<="
86 AND: "and"
87 OR: "or"
88 NOT: "not"
89 READ: "input"
90 PRINT: "print"
91 ATRIB: "="

```

```

92 VAR: ("a".. "z" | "A".. "Z" | "_" )+
93 IF: "if"
94 ELSE: "else"
95 FOR: "for"
96 WHILE: "while"
97 REPEAT: "repeat"
98 %import common.WS
99 %import common.ESCAPED_STRING
100 %import common.C_COMMENT
101 %ignore WS
102 '''
103
104 def criarFicheiroHtml(nome):
105     try:
106         f = open(nome, 'r+', encoding='utf-8')
107         f.truncate(0)
108         return f
109     except:
110         f = open(nome, 'a', encoding='utf-8')
111         return f
112
113 def preencherInicio(ficheiro):
114     conteudo = '''
115 <!DOCTYPE html>
116 <html>
117 <head>
118 <meta charset="UTF-8">
119 <title>Análise Estática</title>
120 </head>
121 <style>
122     .info {
123         position: relative;
124         display: inline-block;
125         border-bottom: 1px dotted black;
126         color: rgb(142, 142, 248);
127     }
128     .info .infotext {
129         visibility: hidden;
130         width: 200px;
131         background-color: #555;
132         color: #fff;
133         text-align: center;
134         border-radius: 6px;
135         padding: 5px 0;
136         position: absolute;
137         z-index: 1;
138         bottom: 125%;
139         left: 50%;
140         margin-left: -40px;
141         opacity: 0;
142         transition: opacity 0.3s;
143     }
144     .info .infotext::after {
145         content: "";

```

```

146     position: absolute;
147     top: 100%;
148     left: 20%;
149     margin-left: -5px;
150     border-width: 5px;
151     border-style: solid;
152     border-color: #555 transparent transparent transparent;
153 }
154 .info:hover .infotext {
155     visibility: visible;
156     opacity: 1;
157 }
158 .error {
159     position: relative;
160     display: inline-block;
161     border-bottom: 1px dotted black;
162     color: red;
163 }
164 .code {
165     position: relative;
166     display: inline-block;
167 }
168 .error .errortext {
169     visibility: hidden;
170     width: 200px;
171     background-color: #555;
172     color: #fff;
173     text-align: center;
174     border-radius: 6px;
175     padding: 5px 0;
176     position: absolute;
177     z-index: 1;
178     bottom: 125%;
179     left: 50%;
180     margin-left: -40px;
181     opacity: 0;
182     transition: opacity 0.3s;
183 }
184 .error .errortext::after {
185     content: "";
186     position: absolute;
187     top: 100%;
188     left: 20%;
189     margin-left: -5px;
190     border-width: 5px;
191     border-style: solid;
192     border-color: #555 transparent transparent transparent;
193 }
194 .error:hover .errortext {
195     visibility: visible;
196     opacity: 1;
197 }
198 </style>
199 <body>

```



```

200     <h2>Análise de código</h2>
201     <pre>
202         <code>'''
203     ficheiro.write(conteudo + '\n')
204
205 def preencherFim(ficheiro):
206     conteudo = '''
207         </code>
208     </pre>
209 </body>
210 </html>'''
211     ficheiro.write(conteudo)
212
213 def criarSegundaPagina(dicionario, ficheiro):
214     inicio = '''
215     <!DOCTYPE html>
216     <html lang="pt">
217     <head>
218         <link rel="stylesheet" href="w3.css">
219         <meta charset="UTF-8">
220         <title>Informações</title>
221     </head>
222     <body class="w3-container">
223     '''
224
225     #Variáveis e os tipos
226     ficheiro.write(inicio)
227     ficheiro.write('\t<h2>Variáveis declaradas e os seus tipos</h2>\n')
228     ficheiro.write('\t<table class="w3-table-all">\n\t\t<tr>\n')
229     ficheiro.write('\t\t\t<th>Variável</th>\n')
230     ficheiro.write('\t\t\t<th>Tipo</th>\n')
231     ficheiro.write('\t\t</tr>\n')
232     for k,v in dicionario['decls'].items():
233         ficheiro.write('\t\t<tr>\n')
234         ficheiro.write('\t\t\t<td>' + k + '</td>\n')
235         ficheiro.write('\t\t\t<td>' + v + '</td>\n')
236         ficheiro.write('\t\t</tr>\n')
237     ficheiro.write('\t</table>\n')
238
239     ficheiro.write('\t<h2>Outras informações sobre as variáveis</h2>\n')
240     ficheiro.write('\t<ul class="w3-ul">\n')
241
242     naoInicializadas = dicionario['naoInicializadas']
243     if len(naoInicializadas) > 0:
244         ficheiro.write('\t\t<li><b>Variáveis sem inicialização: </b>' + str(
245             naoInicializadas) + '</li>\n')
246     else:
247         ficheiro.write('\t\t<li><b>Variáveis sem inicialização: </b>Nenhuma</li>\n')
248
249     naoDeclaradas = dicionario['erros']['1: Não-declaração']
250     if len(naoDeclaradas) > 0:
251         ficheiro.write('\t\t<li><b>Variáveis não declaradas: </b>' + str(
252             naoDeclaradas) + '</li>\n')
253     else:

```

```

252     ficheiro.write('\t\t<li><b>Variáveis não declaradas: </b>Nenhuma</li>\n')
253
254 redeclaradas = dicionario['erros']['2: Redeclaração']
255 if len(redeclaradas) > 0:
256     ficheiro.write('\t\t<li><b>Variáveis redeclaradas: </b>' + str(redeclaradas)
257 + '</li>\n')
258 else:
259     ficheiro.write('\t\t<li><b>Variáveis redeclaradas: </b>Nenhuma</li>\n')
260
261 usadasNaoInicializadas = dicionario['erros']['3: Usado mas não inicializado']
262 if len(usadasNaoInicializadas) > 0:
263     ficheiro.write('\t\t<li><b>Variáveis usadas e não inicializadas: </b>' + str(
264 usadasNaoInicializadas) + '</li>\n')
265 else:
266     ficheiro.write('\t\t<li><b>Variáveis usadas e não inicializadas: </b>Nenhuma
267 </li>\n')
268
269 nuncaMencionadas = dicionario['erros']['4: Declarado mas nunca mencionado']
270 if len(nuncaMencionadas) > 0:
271     ficheiro.write('\t\t<li><b>Variáveis declaradas mas nunca mencionadas: </b>'
272 + str(nuncaMencionadas) + '</li>\n')
273 else:
274     ficheiro.write('\t\t<li><b>Variáveis declaradas mas nunca mencionadas: </b>
275 Nenhuma</li>\n')
276
277 ficheiro.write('\t</ul>\n')
278
279 ficheiro.write('\t<h2>Informações sobre as instruções</h2>\n')
280 ficheiro.write('\t<ul class="w3-ul">\n')
281 ficheiro.write('\t\t<li><b>Total de instruções: </b>' + str(dicionario['
282 instrucoes']['total']) + '</li>\n')
283 ficheiro.write('\t\t<li><b>Total de atribuições: </b>' + str(dicionario['
284 instrucoes']['atribuicoes']) + '</li>\n')
285 ficheiro.write('\t\t<li><b>Total de leituras: </b>' + str(dicionario['
286 instrucoes']['leitura']) + '</li>\n')
287 ficheiro.write('\t\t<li><b>Total de escritas: </b>' + str(dicionario['
288 instrucoes']['escrita']) + '</li>\n')
289 ficheiro.write('\t\t<li><b>Total de instruções condicionais: </b>' + str(
290 dicionario['instrucoes']['condicionais']) + '</li>\n')
291 ficheiro.write('\t\t<li><b>Total de instruções cíclicas: </b>' + str(dicionario
292 ['instrucoes']['ciclicas']) + '</li>\n')
293 ficheiro.write('\t\t<li><b>Total de situações de aninhamento: </b>' + str(
294 dicionario['totalSituacoesAn']) + '</li>\n')
295 ficheiro.write('\t</ul>\n')
296
297 ficheiro.write('\t<h2>Informações sobre os ifs e os seus aninhamentos</h2>\n')
298 ficheiro.write('\t<h4>Níveis de aninhamento dos ifs</h4>\n')
299 ficheiro.write('\t<ul class="w3-ul">\n')
300 niveis = dicionario['niveisIf']
301 for k, v in niveis.items():
302     ficheiro.write('\t\t<li><b>Nível ' + str(k) + ': </b>Ifs pela ordem em que
303 aparecem no código - ' + str(v) + '</li>\n')
304 ficheiro.write('\t</ul>\n')
305
306 alternativas = dicionario['alternativasIfs']

```

```

293 if len(alternativas) > 0:
294     ficheiro.write('\t<h4>Alternativa para os ifs aninhados</h4>\n')
295     for a in alternativas:
296         ficheiro.write('\t<p>' + str(a) + '</p>\n')
297
298 def eDigito(palavra):
299     res = True
300     i = 0
301     size = len(palavra)
302     while res and i < size:
303         if palavra[i] < "0" or palavra[i] > "9":
304             res = False
305             i += 1
306     return res
307
308 class LinguagemProgramacao(Interpreter):
309
310     def __init__(self):
311         self.fHtml = criarFicheiroHtml('codigoAnotado.html')
312         self.f2Html = criarFicheiroHtml('informacoesAdicionais.html')
313         preencherInicio(self.fHtml)
314         self.decls = {}
315         self.naoInicializadas = set()
316         self.utilizadas = set()
317         self.erros = {
318             '1: Não-declaração' : set(),
319             '2: Redecclaração' : set(),
320             '3: Usado mas não inicializado' : set(),
321             '4: Declarado mas nunca mencionado' : set()
322         }
323         self.dicinstrucoes = {
324             'total' : 0,
325             'atribuicoes' : 0,
326             'leitura' : 0,
327             'escrita' : 0,
328             'condicionais' : 0,
329             'ciclicas' : 0
330         }
331         self.condicoesIfs = []
332         self.alternativasIfs = []
333         self.niveisIfs = {}
334         self.nivelIf = -1
335         self.nivelProfundidade = 0
336         self.totalSituacoesAn = 0
337         self.nasInstrucoes = False
338         self.instrucaoAtual = ''
339         self.output = {}
340
341     def linguagem(self, tree):
342         self.visit(tree.children[0]) #Declarações
343         self.nasInstrucoes = True
344         self.fHtml.write(' '*10 + '<div class="instrucoes">\n')
345         self.visit(tree.children[1]) #Instruções
346         self.fHtml.write(' '*10 + '</div>\n')

```

```

347 preencherFim(self.fHtml)
348 self.fHtml.close()
349 #Verificar as variáveis declaradas mas nunca mencionadas
350 declaradas = set(self.decls.keys())
351 self.erros['4: Declarado mas nunca mencionado'] = declaradas - self.
utilizadas
352 #Criar o output
353 self.output['decls'] = self.decls
354 self.output['naoInicializadas'] = self.naoInicializadas
355 self.output['erros'] = self.erros
356 self.output['niveisIf'] = self.niveisIfs
357 self.output['utilizadas'] = self.utilizadas
358 self.output['instrucoes'] = self.dicinstrucoes
359 self.output['totalSituacoesAn'] = self.totalSituacoesAn
360 self.output['condicoesIf'] = self.condicoesIfs
361 self.output['alternativasIfs'] = self.alternativasIfs
362 #Preencher a segunda página com informações adicionais
363 criarSegundaPagina(self.output, self.f2Html)
364 return self.output
365
366 def declaracoes(self, tree):
367     #Visita todas as declarações e cada uma processa a sua parte
368     self.fHtml.write(' '*10 + '<div class="declaracoes">\n')
369     for decl in tree.children:
370         if isinstance(decl, Tree):
371             self.visit(decl)
372     self.fHtml.write(' '*10 + '</div>\n')
373
374 def declaracao(self, tree):
375     self.fHtml.write('\t')
376     #Inicialização de variáveis
377     var = None
378     tipo = None
379     valor = None
380     #É preciso percorrer cada filho e processar de acordo com as situações
381     for child in tree.children:
382         if isinstance(child, Token) and child.type == 'VAR' and child.value in self
.decls.keys(): #Visita a variável declarada
383             var = child.value
384             self.erros['2: Redeclaração'].add(var)
385             self.fHtml.write('<div class="error">' + var + '<span class="errortext">
Variável redeclarada</span></div>')
386         elif isinstance(child, Token) and child.type == 'VAR':
387             var = child.value
388             self.fHtml.write(var)
389         elif isinstance(child, Tree) and child.data == 'tipo': #Visita o tipo da
declaração
390             tipo = self.visit(child)
391             self.fHtml.write(tipo + ' ')
392         elif isinstance(child, Token) and child.type == 'ATRIB': #Se houver atribui
ção visita o valor que está a ser declarado
393             valor = self.visit(tree.children[3])
394             if valor != None and eDigito(valor):
395                 valor = int(valor)

```

```

396         if valor == None:
397             self.naoInicializadas.add(var)
398             self.decls[var] = tipo
399             self.fHtml.write(';\n')
400         elif isinstance(valor, str) and valor[0] != '':
401             self.fHtml.write(' = ')
402             if ('[' or ']') in valor: #0 valor resulta de um acesso a uma estrutura
403                 infoEstrutura = valor.split('[')
404                 variavel = infoEstrutura[0]
405                 if variavel not in self.decls.keys(): #Se a variável não tiver sido
declarada antes é gerado um erro
406                     if variavel != '':
407                         self.erros['1: Não-declaração'].add(variavel)
408                         self.fHtml.write('<div class="error">' + variavel + '<span class="
errortext">Variável redeclarada</span></div>' + '[' + infoEstrutura[1] + ';\n'
)
409                     elif variavel in self.naoInicializadas:
410                         self.fHtml.write('<div class="error">' + variavel + '<span class="
errortext">Variável não inicializada</span></div>' + '[' + infoEstrutura[1] +
';\n')
411                     else:
412                         self.fHtml.write(variavel + '[' + infoEstrutura[1] + ';\n')
413             elif valor not in self.decls.keys(): #Verificar se a variável atômica j
á existe
414                 self.erros['1: Não-declaração'].add(valor)
415                 self.fHtml.write('<div class="error">' + valor + '<span class="
errortext">Variável redeclarada</span></div>;\n')
416             else:
417                 self.fHtml.write(' = ')
418                 self.fHtml.write(str(valor) + ';\n')
419
420         if valor == None: #Caso nunca tenha existido atribuição na declaração
421             self.naoInicializadas.add(var)
422             self.decls[var] = tipo
423             self.fHtml.write(';\n')
424
425         self.decls[var] = tipo
426
427     def instrucoes(self, tree):
428         r = self.visit_children(tree)
429         return r
430
431     def instrucao(self, tree):
432         instrucaoAtual = self.instrucaoAtual
433         nivelIf = self.nivelIf
434         nivelProfundidade = self.nivelProfundidade
435         numTabs = (nivelProfundidade * '\t') + '\t'
436         self.fHtml.write(numTabs)
437         condicoesParaAninhar = []
438         corpo = ''
439         sairDoCiclo = False
440         resultado = ''
441         for child in tree.children:
442             if not sairDoCiclo:

```

```

443         if isinstance(child,Token) and child.type == 'IF':
444             self.instrucaoAtual = "condicional"
445             self.dicinstrucoes['condicionais'] += 1
446             self.dicinstrucoes['total'] += 1
447
448             if self.nivelIf == -1: #Primeiro if do código
449                 nivelIf = self.nivelIf = 0
450             elif nivelProfundidade > 0:
451                 self.totalSituacoesAn += 1
452
453             self.niveisIfs.setdefault(nivelIf, list())
454             self.niveisIfs[nivelIf].append(self.dicinstrucoes['condicionais'
455 ])
456
457             self.fHtml.write('<div class="info">' + child.value + '<span
458 class="infotext">Nível de aninhamento: ' + str(nivelIf) + '</span></div>')
459             self.fHtml.write('(')
460             condicaoIfAtual = self.visit(tree.children[2])[0]
461             self.fHtml.write('){\n')
462             if len(tree.children[5].children) > 0: #Se existirem instruções
dentro do if
463                 existeElse = 'else' in tree.children[5].children[0].children
464                 proxInstrucao = str(tree.children[5].children[0].children[0])
465                 if proxInstrucao != 'if' or existeElse or len(tree.children
[5].children) > 1:
466                     self.condicoesIfs.append(condicaoIfAtual)
467                     condicoesParaAninhar = self.condicoesIfs
468                     self.condicoesIfs = []
469                     self.nivelProfundidade += 1
470                     self.nivelIf += 1
471                     res = self.visit(tree.children[5])
472                     self.nivelIf = nivelIf
473                     self.nivelProfundidade = nivelProfundidade
474                     numTabs = (self.nivelProfundidade * '\t') + '\t'
475                     self.fHtml.write(numTabs + '}')
476                     corpo = res[0]
477                     for r in res[1:]:
478                         corpo += '\n' + r
479                     if 'else' in tree.children:
480                         self.fHtml.write('else{\n')
481                         self.nivelProfundidade += 1
482                         self.nivelIf += 1
483                         resElse = self.visit(tree.children[9])
484                         self.nivelIf = nivelIf
485                         self.nivelProfundidade = nivelProfundidade
486                         numTabs = (self.nivelProfundidade * '\t') + '\t'
487                         self.fHtml.write(numTabs + '}')
488                         resultado += child.value + '(' + condicaoIfAtual + '
489 {\n' + str(res) + '}else{\n' + str(resElse) + '}'
490                     else:
491                         resultado += child.value + '(' + condicaoIfAtual + '
492 {\n' + str(res) + '}'
493                     else:
494                         self.condicoesIfs.append(condicaoIfAtual)

```

```

491         self.nivelProfundidade += 1
492         self.nivelIf += 1
493         res = self.visit(tree.children[5])
494         self.nivelIf = nivelIf
495         self.nivelProfundidade = nivelProfundidade
496         numTabs = (self.nivelProfundidade * '\t') + '\t'
497         self.fHtml.write(numTabs + '}')
498         corpo = res[0]
499         for r in res[1:]:
500             corpo += '\n' + r
501         if 'else' in tree.children:
502             self.fHtml.write('else{\n')
503             self.nivelProfundidade += 1
504             self.nivelIf += 1
505             resElse = self.visit(tree.children[9])
506             self.nivelIf = nivelIf
507             self.nivelProfundidade = nivelProfundidade
508             numTabs = (self.nivelProfundidade * '\t') + '\t'
509             self.fHtml.write(numTabs + '}')
510             resultado += child.value + '(' + condicaoIfAtual + ')\n' + str(res) + '}else{\n' + str(resElse) + '}'
511         else:
512             resultado += child.value + '(' + condicaoIfAtual + ')\n' + str(res) + '}'
513         sairDoCiclo = True
514         elif isinstance(child, Token) and child.type == 'CE':
515             #0 \n é necessário porque a seguir a este token chegam instruções
516             aninhadas
517             self.fHtml.write(child.value)
518             self.fHtml.write('\n')
519             resultado += '{\n' + numTabs
520             elif isinstance(child, Token) and child.type == 'CD':
521                 self.fHtml.write(numTabs + child.value)
522                 resultado += numTabs + '}'
523             elif isinstance(child, Token) and (child.type == 'FOR' or child.type
524             == 'WHILE' or child.type == 'REPEAT'):
525                 if self.nivelProfundidade > 0:
526                     self.totalSituacoesAn += 1
527                     self.fHtml.write(child.value)
528                     self.dicinstrucoes['ciclicas'] += 1
529                     self.dicinstrucoes['total'] += 1
530                     self.instrucaoAtual = "ciclo"
531                     resultado += child.value
532             elif isinstance(child, Token) and child.type == 'READ':
533                 self.fHtml.write(child.value)
534                 self.instrucaoAtual = "leitura"
535                 self.dicinstrucoes['leitura'] += 1
536                 self.dicinstrucoes['total'] += 1
537                 resultado += child.value
538             elif isinstance(child, Token) and child.type == 'PRINT':
539                 self.fHtml.write(child.value)
540                 instrucaoAtual = self.instrucaoAtual = "escrita"
541                 self.dicinstrucoes['escrita'] += 1
542                 self.dicinstrucoes['total'] += 1

```

```

541         resultado += child.value
542     elif isinstance(child, Token):
543         self.fHtml.write(child.value)
544         resultado += child.value
545     elif isinstance(child, Tree) and child.data == 'atribuicao':
546         instrucaoAtual = self.instrucaoAtual = "atribuicao"
547         self.dicinstrucoes['atribuicoes'] += 1
548         self.dicinstrucoes['total'] += 1
549         resultado += str(self.visit(child))
550     elif isinstance(child, Tree):
551         if child.data == 'contendoread':
552             resultado += str(self.visit(child))
553         elif child.data == 'logic':
554             resultado += str(self.visit(child))
555         elif child.data == 'condicao':
556             resultado += str(self.visit(child))
557         elif child.data == 'instrucoes':
558             self.nivelProfundidade += 1
559             self.nivelIf = 0
560             resultado += str(self.visit(child))
561             self.nivelIf = nivelIf
562             self.nivelProfundidade = nivelProfundidade
563
564     #Se a lista de condições tiver mais do que um elemento então podemos aninhar
565     #os ifs
566     if len(condicoesParaAninhar) > 1:
567         alternativaIf = 'if(' + condicoesParaAninhar[0]
568         for cond in condicoesParaAninhar[1:]:
569             alternativaIf += ' && ' + cond
570             alternativaIf += '){\n' + numTabs + '\t\t'
571             alternativaIf += corpo
572             alternativaIf += '\n' + numTabs + '}'
573         #print('ALTERNATIVA PARA O IF: ', alternativaIf)
574         self.alternativasIfs.append(alternativaIf)
575         self.fHtml.write('\n')
576         return resultado
577
578 def condicao(self, tree):
579     r = self.visit_children(tree)
580     return r
581     #print('CONDICAO: ', r)
582
583 def atribuicao(self, tree):
584     res = ''
585     for child in tree.children:
586         if (isinstance(child, Token) and child.type == 'VAR') and child.value in
587         self.naoInicializadas:
588             self.fHtml.write(child.value)
589             self.naoInicializadas.remove(child.value)
590             res += child.value
591         elif (isinstance(child, Token) and child.type == 'VAR') and child.value not
592         in self.decls.keys():
593             self.fHtml.write('<div class="error">' + child.value + '<span class="
594             errortext">Variável não declarada</span></div>')

```



```

591         res += '<div class="error">' + child.value + '<span class="errortext">
Variável não declarada</span></div>'
592     elif (isinstance(child,Token) and child.type == 'VAR'):
593         self.fHtml.write(child.value)
594         res += child.value
595     elif isinstance(child,Token):
596         self.fHtml.write(child.value)
597         res += str(child.value)
598     elif isinstance(child,Tree) and child.data == 'chave':
599         chave = self.visit(child)
600         self.fHtml.write(str(chave))
601         res += str(chave)
602     elif isinstance(child,Tree):
603         res += str(self.visit(child))
604     return res
605
606 def conteudoread(self,tree):
607     res = ''
608     for child in tree.children:
609         if isinstance(child,Token) and child.type == 'VAR' and child.value in self.
naoInicializadas:
610             self.fHtml.write(child.value)
611             self.naoInicializadas.remove(child.value)
612             self.utilizadas.add(child.value)
613             res += child.value
614         elif isinstance(child,Token) and child.type == 'VAR' and child.value not in
self.decls.keys():
615             self.fHtml.write('<div class="error">' + child.value + '<span class="
errortext">Variável não declarada</span></div>')
616             res += '<div class="error">' + child.value + '<span class="errortext">
Variável não declarada</span></div>'
617         elif isinstance(child,Token):
618             self.fHtml.write(child.value)
619             res += str(child.value)
620         elif isinstance(child,Tree) and child.data == 'chave':
621             chave = str(self.visit(child))
622             self.fHtml.write(chave)
623             res += chave
624     return res
625
626 def tipo(self, tree):
627     return str(tree.children[0])
628
629 def logic(self,tree):
630     res = ''
631     for child in tree.children:
632         if isinstance(child,Token) and self.nasInstrucoes:
633             self.fHtml.write(child.value)
634             res += str(child.value)
635         elif isinstance(child,Tree) and child.data == 'logicnot':
636             visit = self.visit(child)
637             if visit != None:
638                 res += str(visit)
639         elif isinstance(child,Tree):

```

```

640         visit = self.visit(child)
641         if visit != None:
642             res += str(visit)
643     return res
644
645 def logicnot(self,tree):
646     res = ''
647     for child in tree.children:
648         if isinstance(child,Token) and self.nasInstrucoes:
649             self.fHtml.write(child.value)
650             res += str(child.value)
651         elif isinstance(child,Tree) and child.data == 'relac':
652             visit = self.visit(child)
653             if visit != None:
654                 res += str(visit)
655         elif isinstance(child,Tree):
656             visit = self.visit(child)
657             if visit != None:
658                 res += str(visit)
659     return res
660
661 def relac(self,tree):
662     res = ''
663     for child in tree.children:
664         if isinstance(child,Token) and (child.value == '<' or child.value == '>' or
665             child.value == '<=' or child.value == '>=' or child.value == '==') and self.
666             nasInstrucoes:
667             self.fHtml.write(child.value)
668             res += str(' ' + child.value + ' ')
669         elif isinstance(child,Token) and self.nasInstrucoes:
670             self.fHtml.write(child.value)
671             res += str(child.value)
672         elif isinstance(child,Tree) and child.data == 'exp':
673             visit = self.visit(child)
674             if visit != None:
675                 res += str(visit)
676         elif isinstance(child,Tree):
677             visit = self.visit(child)
678             if visit != None:
679                 res += str(visit)
680     return res
681
682 def exp(self,tree):
683     res = ''
684     for child in tree.children:
685         if isinstance(child,Token) and self.nasInstrucoes:
686             self.fHtml.write(child.value)
687             res += str(child.value)
688         elif isinstance(child,Tree) and child.data == 'termo':
689             visit = self.visit(child)
690             if visit != None:
691                 res += str(visit)
692         elif isinstance(child,Tree):
693             visit = self.visit(child)

```

```

692         if visit != None:
693             res += str(visit)
694         return res
695
696 def termo(self, tree):
697     res = ''
698     for child in tree.children:
699         if isinstance(child, Token) and self.nasInstrucoes:
700             self.fHtml.write(child.value)
701             res += str(child.value)
702         elif isinstance(child, Tree) and child.data == 'factor':
703             visit = self.visit(child)
704             if visit != None:
705                 res += str(visit)
706         elif isinstance(child, Tree):
707             visit = self.visit(child)
708             if visit != None:
709                 res += str(visit)
710     return res
711
712 def factor(self, tree):
713     for child in tree.children:
714         if isinstance(child, Token) and child.type == 'VAR' and len(tree.children)
> 1:
715         self.utilizadas.add(child.value)
716         chave = self.visit(tree.children[2])
717         if self.nasInstrucoes and child.value not in self.decls.keys():
718             self.fHtml.write('<div class="error">' + child.value + '<span class="
errortext">Variável não declarada</span></div>' + '[' + str(chave) + ']')
719             self.erros['1: Não-declaração'].add(child.value)
720         elif self.nasInstrucoes and child.value in self.naoInicializadas:
721             self.fHtml.write('<div class="error">' + child.value + '<span class="
errortext">Variável não inicializada</span></div>' + '[' + str(chave) + ']')
722             self.erros['3: Usado mas não inicializado'].add(child.value)
723         elif self.nasInstrucoes:
724             self.fHtml.write(child.value)
725             self.fHtml.write('[' + str(chave) + ']')
726             return str(child.value) + '[' + str(chave) + ']'
727         elif isinstance(child, Token) and child.type == 'VAR':
728             if self.nasInstrucoes and child.value not in self.decls.keys():
729                 self.fHtml.write('<div class="error">' + child.value + '<span class="
errortext">Variável não declarada</span></div>')
730                 self.erros['1: Não-declaração'].add(child.value)
731             elif self.nasInstrucoes and child.value in self.naoInicializadas:
732                 self.fHtml.write('<div class="error">' + child.value + '<span class="
errortext">Variável não inicializada</span></div>')
733                 self.erros['3: Usado mas não inicializado'].add(child.value)
734             elif self.nasInstrucoes:
735                 self.fHtml.write(child.value)
736                 #Adicionar a variável à lista das utilizadas
737                 self.utilizadas.add(child.value)
738                 return str(child.value)
739             elif isinstance(child, Tree) and child.data == 'chave':
740                 #Obter o índice da estrutura se existir

```

```

741     chave = self.visit(child)
742     if self.nasInstrucoes:
743         self.fHtml.write(chave)
744     elif isinstance(child, Token) and child.type == 'NUM':
745         if self.nasInstrucoes:
746             self.fHtml.write(child.value)
747             return int(child.value)
748     elif isinstance(child, Token) and child.type == 'BOOLEANO':
749         if self.nasInstrucoes:
750             self.fHtml.write(child.value)
751             if child.value == "False":
752                 return False
753             elif child.value == "True":
754                 return True
755     elif isinstance(child, Token) and child.type == 'STRING':
756         if self.nasInstrucoes:
757             self.fHtml.write(child.value)
758             return str(child.value)
759     elif isinstance(child, Token) and child.type == 'NUMDOUBLE':
760         if self.nasInstrucoes:
761             self.fHtml.write(child.value)
762             return float(child.value)
763     elif isinstance(child, Token) and child.type == 'PER' and isinstance(tree.
children[1], Token) and (tree.children[1]).type == 'PDR':
764         if self.nasInstrucoes:
765             self.fHtml.write('[]')
766             return list()
767     elif isinstance(child, Token) and child.type == 'PER' and isinstance(tree.
children[1], Tree):
768         lista = None
769         if self.nasInstrucoes:
770             self.fHtml.write('[')
771             lista = self.visit(tree.children[1])
772             self.fHtml.write(']')
773         else:
774             lista = self.visit(tree.children[1])
775         return lista
776     elif isinstance(child, Token) and child.type == 'CE' and isinstance(tree.
children[1], Token) and (tree.children[1]).type == 'CD':
777         if self.nasInstrucoes:
778             self.fHtml.write("{}")
779             return {}
780     elif isinstance(child, Token) and child.type == 'CE' and isinstance(tree.
children[1], Tree):
781         estrutura = None
782         if self.nasInstrucoes:
783             self.fHtml.write('{')
784             estrutura = self.visit(tree.children[1])
785             self.fHtml.write('}')
786         else:
787             estrutura = self.visit(tree.children[1])
788         if isinstance(estrutura, dict):
789             return estrutura
790         else:

```

```

791         return set(estrutura)
792     elif isinstance(child, Token) and child.type == 'PE' and isinstance(tree.
children[1], Token) and (tree.children[1]).type == 'PD':
793         if self.nasInstrucoes:
794             self.fHtml.write('()')
795         return tuple()
796     elif isinstance(child, Token) and child.type == 'PE' and isinstance(tree.
children[1], Tree):
797         tuplo = None
798         if self.nasInstrucoes:
799             self.fHtml.write('(')
800             tuplo = self.visit(tree.children[1])
801             self.fHtml.write(')')
802         else:
803             tuplo = self.visit(tree.children[1])
804         return tuple(tuplo)
805     elif isinstance(child, Token) and self.nasInstrucoes:
806         self.fHtml.write(child.value)
807
808 def conteudoespecial(self, tree):
809     r = self.visit(tree.children[0])
810     return r
811
812 def conteudodicionario(self, tree):
813     estrutura = dict()
814     i = 0
815     for child in tree.children:
816         if i == 0 and isinstance(child, Tree):
817             entrada = self.visit(child)
818             chave = entrada[0][1:-1]
819             estrutura[chave] = entrada[2]
820         elif isinstance(child, Tree):
821             if self.nasInstrucoes:
822                 self.fHtml.write(',')
823             entrada = self.visit(child)
824             chave = entrada[0][1:-1]
825             estrutura[chave] = entrada[2]
826         else:
827             entrada = self.visit(child)
828             chave = entrada[0][1:-1]
829             estrutura[chave] = entrada[2]
830     i += 1
831     return estrutura
832
833 def entrada(self, tree):
834     res = []
835     for child in tree.children:
836         if self.nasInstrucoes and isinstance(child, Token):
837             self.fHtml.write(child.value)
838             res.append(child.value)
839         elif isinstance(child, Token):
840             res.append(child.value)
841         elif isinstance(child, Tree):
842             r = self.visit(child)

```

```

843         res.append(r)
844     return res
845
846 def conteudo(self, tree):
847     # print('Entrei no conteúdo de uma estrutura...')
848     res = list()
849     i = 0
850     for child in tree.children:
851         if i == 0 and isinstance(child, Tree):
852             r = self.visit(child)
853             res.append(r)
854         elif isinstance(child, Tree):
855             r = None
856             if self.nasInstrucoes:
857                 self.fHtml.write(',')
858                 r = self.visit(child)
859             else:
860                 r = self.visit(child)
861             res.append(r)
862         i += 1
863     return res
864
865 def chave(self, tree):
866     if tree.children[0].type == 'NUM':
867         return int(tree.children[0].value)
868     elif self.nasInstrucoes and tree.children[0].type == 'STRING':
869         r = (tree.children[0].value)
870         return r
871     elif tree.children[0].type == 'STRING':
872         r = (tree.children[0].value)[1:-1]
873         return r
874     elif tree.children[0].type == 'VAR':
875         return str(tree.children[0].value)
876
877 l = Lark(grammar, start='linguagem')
878
879 f = open('codigoFonte.txt', 'r')
880 input = f.read()
881
882 tree = l.parse(input)
883 #print(tree.pretty())
884
885 data = LinguagemProgramacao().visit(tree)
886 print('-'*100)
887 print('Output: ', data)
888 print('-'*100)

```