Uma imagem com texto, Tipo de letra, Azul elétrico, logótipo

Descrição gerada automaticamente

|  |  |
| --- | --- |
| David Martinho (25620)  João Ferreira (25986)  5 de junho de 2023 |  |

|  |
| --- |
| Processamento de Linguagem |
| Relatório do Trabalho Prático 02 |
| Docente: Óscar Ribeiro |

Índice

[Introdução 3](#_Toc168609492)

[Estrutura 4](#_Toc168609493)

[Analisador Léxico (FCALexer.py) 5](#_Toc168609494)

[FCAGrammar.py 7](#_Toc168609495)

[FCAEval.py 9](#_Toc168609496)

[FCA.py 11](#_Toc168609497)

# Introdução

Este relatório descreve o desenvolvimento do projeto prático para a disciplina de Processamento de Linguagens. O objetivo do projeto é criar uma aplicação que processa uma linguagem funcional chamada Linguagem Funcional do Cávado e do Ave (FCA). Esta aplicação deve ler um ficheiro de texto com um programa escrito em FCA e executar os comandos nele contidos. Opcionalmente, a aplicação pode também gerar um ficheiro de texto com o código C correspondente.

Para realizar este projeto, seguimos várias etapas:

Definição da gramática da linguagem: Especificamos as regras gramaticais que definem a estrutura da linguagem FCA.

Criação de um analisador léxico: Usamos a biblioteca lex para construir um analisador que identifica os símbolos terminais da gramática FCA. Testamos este analisador com exemplos de palavras da linguagem.

Criação de um analisador sintático: Usamos a biblioteca yacc para construir um analisador que reconhece a gramática completa da linguagem FCA. Testamos este analisador com exemplos de frases da linguagem.

Planeamento da árvore de sintaxe abstrata: Desenhamos uma estrutura para representar a sintaxe dos programas em FCA e associamos ações semânticas às regras da gramática para construir esta árvore.

Este relatório explica cada uma dessas etapas, detalhando como foram implementadas e testadas. O projeto permitiu-nos aprender e aplicar conceitos importantes sobre processamento de linguagens, proporcionando uma experiência prática valiosa.

# Estrutura

O nosso projeto está distribuído nos seguintes ficheiros:

**FCALexer.py**

Este ficheiro contém o analisador léxico, responsável por transformar o código-fonte numa sequência de tokens. Os tokens são elementos básicos da linguagem, como números, identificadores, operadores e palavras-chave.

**FCAGrammar.py**

Este ficheiro contém o analisador sintático, que usa a biblioteca PLY yacc para transformar a sequência de tokens numa árvore de sintaxe abstrata (AST). Ele também trata da interpolação de strings.

**FCAEval.py**

Este ficheiro contém o avaliador, que executa a árvore de sintaxe abstrata (AST) gerada pelo analisador sintático, avaliando as expressões e executando os comandos.

**FCA.py**

Este ficheiro contém o módulo de semântica, que verifica a semântica do programa, garantindo que o código seja válido e consistente.

**Executar o interpretador:**

No terminal, depois de navegar até ao diretório onde os ficheiros estão armazenados, quando executamos o seguinte comando:

*python fca.py programa.fca*

O interpretador vai ler o ficheiro programa.fca, vai analisar o código, executar as instruções e irá imprimir os resultados, lidando corretamente com a interpolação de strings, concatenação de strings e comentários.

Também é possível usar o modo interativo, onde o utilizador escreve o seu próprio código na linha de comandos.

*python fca.py*

# Analisador Léxico (FCALexer.py)

Uma imagem com texto, captura de ecrã, Software de multimédia, software

Descrição gerada automaticamente

Os tokens como NUMBER, IDENTIFIER, STRING, e palavras-chave específicas são definidos, junto com literais que são caracteres únicos.

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

t\_NUMBER reconhece e converte números inteiros.

t\_STRING reconhece strings e remove as aspas ao redor delas.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Funções como t\_ESCREVER, t\_ENTRADA, e t\_ALEATORIO reconhecem palavras-chave específicas da linguagem, considerando diferentes combinações de maiúsculas e minúsculas.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

t\_comment\_multi ignora comentários de múltiplas linhas e atualiza a contagem de linhas.

t\_comment\_single ignora comentários de única linha.

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente

t\_error lida com tokens inesperados, imprime uma mensagem de erro e salta o token inválido.



O lexer é criado usando a função lex.lex(), que compila todas as definições anteriores num lexer funcional.

# FCAGrammar.py

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

O módulo ply.yacc e os tokens do lexer são importados.

Um dicionário functions é criado para armazenar as funções definidas.

A precedência dos operadores é estabelecida para resolver corretamente expressões complexas.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

p\_program define que um programa é uma lista de declarações.

p\_statement\_list define como se agrupam múltiplas declarações.

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

p\_function\_declaration define como uma função é declarada, guardando o nome, os parâmetros e o corpo no dicionário functions.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

p\_command\_statement define como processar comandos específicos como ESCREVER e ALEATORIO, permitindo diferentes formas de sintaxe.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

p\_expression\_binop define operações binárias como adição, subtração, multiplicação e divisão.

p\_expression\_concat define a operação de concatenação.

Uma imagem com texto, Tipo de letra, captura de ecrã, file

Descrição gerada automaticamente

O parser é criado usando yacc.yacc(), compilando todas as regras gramaticais definidas.

A função parse recebe uma string de entrada e retorna a árvore de sintaxe abstrata (AST) gerada pelo parser.

# FCAEval.py

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

A função evaluate é a principal responsável por avaliar nós da árvore de sintaxe abstrata (AST). Ela processa diferentes tipos de nós e operadores, manipulando variáveis e funções conforme necessário.

Esta secção lida também com operações aritméticas básicas (+, -, \*, /) e concatenação (concat). Cada operação é avaliada recursivamente.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Esta parte trata de comandos como escrever, que imprime valores, e assign, que atribui valores ás variáveis.

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

O código inclui a lógica para a avaliação de chamadas de funções, incluindo funções internas como map e fold.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

A função interpolate\_string permite substituir variáveis dentro de strings pelo seu valor atual no contexto das variáveis.

# FCA.py

Uma imagem com texto, Tipo de letra, captura de ecrã, file

Descrição gerada automaticamente

O código importa funções e variáveis essenciais dos módulos FCAGrammar e FCAEval. A função parse é usada para analisar a entrada, enquanto evaluate avalia a árvore de sintaxe abstrata (AST).

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

A função main é o ponto de entrada do programa, que lida com dois modos de operação: execução de um ficheiro de entrada ou modo interativo.

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Se um ficheiro for passado como argumento de linha de comando, o conteúdo do ficheiro é lido, analisado e avaliado. Erros durante o processo são capturados e exibidos.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

No modo interativo, o utilizador pode escrever expressões para serem analisadas e avaliadas em tempo real. O laço continua até que o utilizador digite uma linha vazia. Erros são capturados e exibidos.



O código dentro do bloco if \_\_name\_\_ == "\_\_main\_\_": garante que a função main seja executada apenas se o script for executado diretamente, não quando importado como módulo.

# Programa.fca

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Neste código, temos o código fonte da nossa linguagem.

No programa, são realizadas operações matemáticas como multiplicação, subtração e soma, atribuindo os resultados a variáveis. Além disso, são manipuladas strings, com a concatenação de texto e a interpolação de variáveis.

A definição e invocação de funções são exemplificadas com as funções soma e area, demonstrando a modularização do código e a capacidade de reutilização de blocos de código.

Outro destaque é a manipulação de listas, com o uso das funções map e fold para aplicar operações a cada elemento da lista ou reduzi-la a um único valor, respectivamente.

Por fim, o código também ilustra operações de entrada e saída de dados, como a leitura de valores do usuário e a exibição de resultados na tela.

# Conclusão

Este projeto prático para a disciplina de Processamento de Linguagens proporcionou uma oportunidade valiosa para desenvolver e aplicar conhecimentos na criação de uma aplicação que processa a Linguagem Funcional do Cávado e do Ave (FCA). Através das várias etapas do projeto, conseguimos construir um sistema funcional que lê programas escritos em FCA, executa os comandos e, opcionalmente, gera código C correspondente. Durante o desenvolvimento do projeto, seguimos os seguintes passos: Primeiro, definimos a gramática da linguagem, especificando as regras gramaticais que definem a estrutura da FCA, estabelecendo uma base sólida para a análise sintática e semântica da linguagem. Em seguida, criamos um analisador léxico utilizando a biblioteca `lex`, construindo um analisador capaz de identificar os símbolos terminais da gramática FCA. Este analisador foi testado com diversos exemplos, garantindo a precisão na identificação dos tokens. Posteriormente, desenvolvemos um analisador sintático com a biblioteca `yacc`, que reconhece a gramática completa da FCA. Testamos este componente com várias frases da linguagem para assegurar sua correção e eficiência. Além disso, planejamos a árvore de sintaxe abstrata (AST), desenhando sua estrutura para representar a sintaxe dos programas em FCA e associamos ações semânticas às regras da gramática. Isso permitiu a construção de uma representação interna coerente dos programas, fundamental para a etapa de avaliação e execução dos comandos. A implementação dessas etapas foi detalhada neste relatório, demonstrando o processo de desenvolvimento, os desafios enfrentados e as soluções adotadas. O projeto não apenas reforçou nosso entendimento teórico sobre o processamento de linguagens, mas também nos forneceu uma experiência prática significativa na construção de analisadores léxicos e sintáticos, bem como na manipulação de árvores de sintaxe abstrata. Em resumo, o projeto de desenvolvimento da aplicação para a Linguagem Funcional do Cávado e do Ave (FCA) foi um sucesso, alcançando os objetivos propostos e oferecendo uma aplicação prática dos conceitos estudados. Esta experiência foi fundamental para consolidar o conhecimento na área de processamento de linguagens, preparando-nos para futuros desafios na programação e análise de linguagens.