# Processamento de Linguagens

## Engenharia Informática (3º ano)

Projeto Final

18 de Março de 2024

## Objectivos e Organização

Este trabalho prático tem como principais objectivos:

- aumentar a experiência em engenharia de linguagens e em programação generativa (gramatical), reforçando a capaci- dade de escrever gramáticas, quer independentes de contexto (GIC), quer tradutoras (GT);
- desenvolver processadores de linguagens segundo o método da tradução dirigida pela sintaxe, a partir de uma gramática tradutora;
- desenvolver um compilador gerando código para um objetivo específico;
- utilizar geradores de compiladores baseados em gramáticas tradutoras, concretamente o Yacc, versão PLY do Python, completado pelo gerador de analisadores léxicos Lex, também versão PLY do Python.

Na resolução dos trabalhos práticos desta UC, aprecia-se a imaginação/criatividade dos grupos em todo o processo de desenvolvimento! Deve entregar a sua solução até Domingo dia 12 de Maio.

O ficheiro com o relatório e a solução deve ter o nome 'pl2024- projeto-grNN.zip' de deverá estar no formato ZIP, em que NN corresponderá ao número de grupo. O número de grupo será ou foi atribuído no registo das equipes do projeto.

A submissão deverá ser feita no Black Board da UC.

Na defesa, a realizar na semana de 13 a 17 de Maio, o programa desenvolvido será apresentado aos membros da equipa docente, totalmente pronto e a funcionar (acompanhado do respectivo relatório de desenvolvimento) e será defendido por todos os elementos do grupo, em data e hora a marcar. O relatório a elaborar, deve ser claro e, além do respectivo enunciado, da descrição do problema, das decisões que lideraram o desenho da solução e sua implementação, deverá conter exemplos de utilização (textos fontes diversos e respectivo resultado produzido). Como é de tradição, o relatório será escrito em LaTeX.

## Recursos e documentação

Estão disponíveis na internet vários recursos quer de documentação quer de ambientes de programação para Forth que o aluno poderá usar para estudar a linguagem e perceber melhor o problema.

Podemos destacar os seguintes recursos que foram usados na preparação do projeto e para criar os programas exemplo neste enunciado:

- **Documentação**: um dos muito manuais online apadrinhado pelo criador da linguagem: *"...I hope this book is not so easy and enjoyable that it seems trivial. Be warned that there is heavy content here*

*and that you can learn much about computers and compilers as well as about programming." — Charles Moore*;

- **IDE de desenvolvimento e testes**: IDE para escrita e teste de programas, neste IDE os programas são compilados;
- **Máquina Virtual (VM)** a ser usada para a geração de código: Documentação e IDE para testes e execução.

## Forth: uma linguagem de programação

O Forth é uma linguagem de programação de baixo nível, baseada numa stack, que foi criada por Charles H. Moore na década de 1960. É conhecida pela sua simplicidade e eficiência, sendo amplamente utilizada em sistemas embebidos, controlo de hardware, sistemas operativos e outras aplicações que requerem desempenho e compactação.

A linguagem Forth tem como principais caraterísticas:

- **Stack**: Todas as operações em Forth são realizadas utilizando uma stack de dados. Os valores são empilhados e desempilhados para execução das operações;
- **Notação pós-fixa (RPN - Reverse Polish Notation)**: Forth utiliza a notação pós-fix, onde os operadores são colocados após os seus operandos. Por exemplo, 2 3 + realiza a operação de adição entre 2 e 3;
- **Extensível**: Forth é uma linguagem extremamente extensível, permitindo que novas palavras (ou funções) sejam definidas pelo utilizador de forma rápida e fácil. Isso permite uma grande flexibilidade na criação de programas específicos para determinadas aplicações;
- **Interpretada ou compilada**: Forth pode ser interpretada diretamente a partir do código fonte ou compilada para código nativo, dependendo da implementação. No nosso caso, iremos compilá-la;
- **Eficiência e compactação**: Forth é conhecida pela sua eficiência e compactação de código. Isso a torna uma escolha popular para sistemas com recursos limitados de hardware.

Em resumo, Forth é uma linguagem de programação simples, eficiente e flexível, que se destaca pela sua abordagem baseada numa stack e pela capacidade de se adaptar a uma ampla gama de aplicações, desde sistemas embebidos até sistemas operativos.

Existem muitos manuais disponíveis na internet pelo que não descreveremos mais da linguagem neste documento. No entanto, apresentam-se uma série de programas comentados que poderão ajudar a perceber a linguagem e poderão ser usados como teste no compilador.

## Enunciado: o problema

Neste projeo, deverás implementar um compilador de Forth que deverá gerar código para a máquina virtual criada no contexto desta UC e disponível online.

Ambas as linguagens, o Forth e o código máquina da máquina virtual serão apresentados e trabalhados nas aulas teóricas.

Na introdução deste documento, tens as localizações da documentação e ambientes de teste e desenvolvimento que terás de usar.

Este projeto será avaliado pelo seu grau de completude e apresentam-se já os seguintes níveis, do mais básico para o mais completo:

1. Suporte a todas as expressões aritméticas: soma, adição, subtração, divisão, resto da divisão inteira;
2. Suporte à criação de funções;
3. Suporte ao `print`de carateres e strings (`.`, `." string"`, `emit`, `char`);
4. Suporte a condicionais;
5. Suporte a ciclos;
6. Suporte a variáveis;
7. ...

## Sintaxe

Um programa em FORTH é uma lista de palavras (`words`) separadas por espaço:

```
WAKE.UP EAT.BREAKFAST WORK EAT.DINNER PLAY SLEEP
```

ou

```
." #S SWAP ! @ ACCEPT . *
```

## Baseada em Stacks

Todas as operações em FORTH assentam na manipulação de Stacks. A mais simples e mais usada é a Stack de Dados (dataStack).

A Stack de Dados está inicialmente vazia. Para colocarmos valores na stack, basta introduzirmos esses valores como palavras:

```
17 34 23
```

Para imprimir o valor no topo da stack usamos a palavra `.`:

```
< 17 34 23
< .
> 23
```

E agora usa os recursos apontados para estudares a linguagem.

## Aritmética

Exemplo:

```
2 3 + .
2 3 + 10 + .
```

As expressões aritméticas escrevem-se no formato RPN ("Reverse Polish Notation").

```
30 5 - . ( 25=30-5 )
30 5 / . ( 6=30/5 )
30 5 * . ( 150=30*5 )
30 5 + 7 / . \ 5=(30+5)/7
```

Atalhos: `1+ 1- 2+ 2- 2* 2/`

Experimentar no interpretador online:

```
10 1- .
7 2* 1+ . ( 15=7*2+1 )
```

### Exercício: Conversão de expressões em formato infixo para pós-fixo

Escreve expressões FORTH para as seguintes expressões aritméticas:

```
(12 * ( 20 - 17 ))
(1 - ( 4 * (-18) / 6) )
( 6 * 13 ) - ( 4 * 2 * 7 )
```

## Definição de uma nova palavra ou função

Utilizam-se duas novas palavras: `:` e `;` Que marcam o início e o fim de uma definição de uma nova palavra ou função.

Exemplo: `: AVERAGE ( a b -- avg ) + 2/ ;`

```
: AVERAGE ( a b -- avg ) + 2/ ;
10 20 AVERAGE .
```

## Input e Output de carateres

```
CHAR W .
CHAR % DUP . EMIT
CHAR A DUP .
32 + EMIT
```

```
CHAR ( <char> -- char , get ASCII value of a character )
```

### Strings

```
: TOFU ." Yummy bean curd!" ;
TOFU
```

```
: SPROUTS ." Miniature vegetables." ;
: MENU
    CR TOFU CR SPROUTS CR
;
MENU
```

### Input

```
: TESTKEY ( -- )
    ." Hit a key: " KEY CR
    ." That = " . CR
;
TESTKEY
```

Resumindo:

```
EMIT ( char -- , output character )
KEY ( -- char , input character )
SPACE ( -- , output a space )
SPACES ( n -- , output n spaces )
CHAR ( <char> -- char , convert to ASCII )
CR ( -- , start new line , carriage return )
." ( -- , output " delimited text )
```

## Alguns programas exemplo em Forth

A seguir apresentam-se alguns programas em Forth que poderão ajudar a compreender melhor a linguagem e que podem ser usados para testar o compilador desenvolvido.

### Hello world!

```
: hello-world  ( -- )
  ." Hello, World!"  cr ;

hello-world  \ Call the defined word
```

Notas:

- `: hello-world` inicia a definição de uma nova palavra designada `hello-world`;
- `( -- )` indica que `hello-world` não recebe argumentos e não deixa nada na stack;
- `." Hello, World!"` coloca na saída (stdout) a string `"Hello, World!"` sem alterar o estado da stack;
- `cr` escreve um caráter de mudança de linha '\n' na saída;
- `;` termina a definição da nova palavra.

### Maior de 2 números passados como argumento

```
: maior2 2dup > if swap . ." é o maior " else . ." é o maior " then ;
77 156 maior2
```

### Maior de 3 números passados como argumento

```
: maior2 2dup > if swap then ;
: maior3 maior2 maior2 . ;
2 11 3  maior3
```

### Maior de N números passados como argumento

```
: maior2 2dup > if drop else swap drop then ;
: maior3 maior2 maior2 ;
: maiorN depth 1 do maior2 loop ;
2 11 3 4 45 8 19 maiorN .
```

### Somatório de 1 até n-1

```
: somatorio 0 swap 1 do i + loop ;
11 somatorio .
```

### Playing with chars and strings

```
( May the Forth be with you)
: STAR 42 EMIT ;
: STARS   0 DO  STAR  LOOP ;
: MARGIN  CR 30 SPACES ;
: BLIP MARGIN STAR ;
: IOI MARGIN STAR 3 SPACES STAR ;
```

```
: IIO MARGIN STAR STAR 3 SPACES ;
: OIO MARGIN 2 SPACES STAR 2 SPACES ;
: BAR  MARGIN 5 STARS ;
: F    BAR BLIP BAR BLIP BLIP CR ;
: O BAR IOI IOI IOI BAR CR ;
: R BAR IOI BAR IIO IOI CR ;
: T BAR OIO OIO OIO OIO CR ;
: H IOI IOI BAR IOI IOI CR ;
F O R T H
```

## Fatorial

```
: factorial  ( n -- n! )
  dup 0 = if
    drop 1
  else
    dup 1 - recurse *
  then ;

\ Example usage:
5 factorial .  \ Calculate factorial of 5 and print result
```

Notas:

Explanation:

- `: factorial` inicia a definição de uma nova palavra designada `factorial`;
- `( n -- n! )` indica que `factorial` recebe um argumento (n) e deixa um resultado (n!) na stack;
- `dup 0 = if` testa se o argumento é 0;
- Se o argumento for 0, este é descartado e o valor 1 é colocado na stack (será o caso de base parao cálculo do fatorial);
- Se o argumento não for 0, este é duplicado, subtrai-se 1 ao duplicado, a função é chamada recursivamente, e o seu resultado é multiplicado pelo argumento original;
- `then` marca o fim do bloco condicional;
- `\ Example usage:` é um comentário que indica como usar a palavra/função nova;
- `5 factorial .` calcula o fatorial de 5 e escreve o resultado na saída.

Este programa mostra a utilização da recursividade em Forth, neste caso, para calcular o fatorial.

## Somatório de n números

```
: sum ( n -- sum )
  0 swap 1 do
    i +
  loop ;

\ Example usage:
5 sum .
```

Notas:

- `: sum` inicia a definição de uma nova palavra designada `sum`;
- `( n -- sum )` indica que `sum` recebe 1 argumento (n) e deixa1 resultado (sum) na stack;
- `0 swap` inicializa `sum` a 0 e troca o argumento com o topo da stack;
- `1 do` inicia o ciclo de 1 até ao valor do argumento;
- `i +` adiciona o índice do ciclo corrente (i) a `sum`;
- `loop` termina o ciclo;
- `\ Example usage:` é um comentário;
- `5 sum .` calcula e imprime o somatório dos números inteiros de 1 a 5.