Introdução

No âmbito da unidade curricular de Processamento de Linguagens, foi nos proposto que desenvolvesse-mos um compilador que converta código Forth para a VM. Neste sentido, foram criados dois ficheiros python, um contendo o analisador léxico e outro o sintático, necessários para realizar essa tarefa. Este trabalho requer um foco extra no sentido em que a linguagem utilizada é pós-fixa, formatação que não nos é tão familiar.

Estrutura geral

Inicialmente, definimos que o programa seria composto por uma Frase, sendo que essa Frase podia ser simples (uma única frase), ou composta por diversas Frases recursivamente. Notar também que a Frase em questão pode ser vazia, no caso em que o input fornecido não contem nenhum código Forth.

```
Prog : Frase
```

Expressões Relacionais

Para suportar as operações que envolvem expressões relacionais em Forth, realizamos a introdução de duas novas produções em Frase, onde as mesmas representam uma Frase que é composta apenas por uma expressão, ou por uma expressão e a impressão do seu respetivo valor, respetivamente.

```
Frase : Frase Exp
      | Frase Exp '.'
```

A produção de uma Expressão, pode ser composta simplesmente por um Fatores ou conter também um simbolo da operação a realizar, tais como:

```
Exp : Fatores
    | Exp '+'
    | Exp '-'
    | Exp '*'
    | Exp '/'
    | Exp 'MOD'
    | Exp 'GREATEREQUAL'
    | Exp 'GREATER'
    | Exp 'LESSEQUAL'
    | Exp 'LESS'
    | Exp 'EQUAL'
    | Exp 'NOTEQUAL'
    | Exp 'DUP'
```

De notar que a produção Exp tem também outras duas entradas que suportam emits e comentários, que serão abordados mais à frente.

Fatores e Fator

A produção Fatores, limita-se a ser uma procura recursiva de tokens Fator de maneira consecutiva, agrupando-os nessa nova abstração, sendo a produção Fator um Número ou um Valor perviamente atribuído a uma variável (usando-se o token @, que em forth é usado para ir buscar o valor de uma certa variável).

```
Fatores : Fator
        | Fatores Fator

Fator : NUM
      | VAR '@'
```

Variáveis

Na produção Frase, as variáveis podem estar em dois formatos, sendo essas, a sua declaração e possível atribuição de um valor a essa mesma variável.

```
Frase : Frase Decl
      | Frase Atrib
```

A declaração de uma variável em Forth, consiste em usar a palavra reservada VARIABLE, seguida do nome a da variável.

```
Decl : VARIABLE VAR
```

Quanto à atribuição de um valor a essa mesma variável, a mesma pode ser obtida de duas maneiras, atribuindo um Fator (encapsulado em Fatores), ou atribuindo um Expressão Relacional, onde a mesma irá guardar o resultado dessa mesma expressão. Nos dois casos, a Frase termina com o uso do token '!', que em Forth indica que se trata de uma atribuição a uma variável.

```
Atrib : Fatores VAR '!'
      | Exp VAR '!'
```

Funções

De maneira análoga às variáveis, as funções também são resumidas em duas entradas na produção Frase, sendo elas a FuncDecl e FuncAtrib.

```
Frase : Frase FuncDecl
      | Frase FuncAtrib
```

Em Forth, quando queremos declarar uma função, começamos com o token ':', seguido do nome da função, das transformações que a mesma irá realizar (FuncCont), terminando a Frase com o token ';'

```
FuncDecl : ':' VAR FuncCont ';'
```

A produção FuncCont determina todas as possibilidades de ações que uma Função pode realizar, desde fatores e operacões relacionais a prints, podendo também conter Ciclos e Condicionais. Notar também que a função pode ser declarada com o seu conteúdo a vazio, sendo nessa caso uma função que não irá realizar transformações nos seus possíveis argumentos.

```
FuncCont :
         | FuncCont Fatores
         | FuncCont '+'
         | FuncCont '-'
         | FuncCont '*'
         | FuncCont '/'
         | FuncCont MOD
         | FuncCont GREATEREQUAL
         | FuncCont GREATER
         | FuncCont LESSEQUAL
         | FuncCont LESS
         | FuncCont EQUAL
         | FuncCont NOTEQUAL
         | FuncCont DUP
         | FuncCont '.'
         | FuncCont STRING
         | FuncCont Condicional1
         | FuncCont Condicional2
         | FuncCont Ciclo1
         | FuncCont Ciclo2
         | FuncCont COMMENT
```

Nas atribuições a funções, gerou-se três entradas na produção FuncAtrib, onde as mesmas tratam dos casos onde a Função recebe Fatores como argumento, recebe uma Exp como argumento ou até nem recebe argumentos. Nos três casos, terminam com uma chamada à produção FuncPrint, que verifica se o valor da funcão será, ou não, imprimido, verificando a existência do token '.' após a atribuição da função.

```
FuncAtrib : Fatores VAR FuncPrint
          | Exp VAR FuncPrint
          | VAR FuncPrint

FuncPrint :
          | '.'
```

Print de carateres e strings

Em relação ao suporte de prints de caracteres e strings, adicionamos três tokens ao analisador léxico sendo elas o 'STRING', 'EMIT', 'CHAR'.

Token '.'

O token '.' é uma instrução utilizada no Forth para imprimir o valor no topo da pilha de dados. Sempre que é necessário imprimir o resultado de uma operação aritmética ou o valor de uma variável, é comum acrescentar um ponto '.' no final da linha para indicar que o valor deve ser impresso.

Por exemplo:

```
1 2 + .
```

Isso resultaria na impressão do valor 3.

Para isso foi criado a seguinte produção:

```
Frase : Frase Exp '.'
      | Frase CHAR VAR '.'

FuncCont : FuncCont '.'

FuncPrint : '.'
```

." string"

A construção `." string"` é usada para imprimir uma string no Forth.

Por exemplo:

```
." Hello, world!"
```

Isso resultaria na impressão da mensagem "Hello, world!".

Para isso foi criado os seguintes processos:

```
Frase : Frase STRING

FuncCont : FuncCont STRING
```

emit

O token `EMIT` é usado para imprimir um único caractere no Forth.

```
65 Emit
```

Isso resultaria na impressão do caractere 'A'.

Para isso foi criado o seguinte processo:

```
Exp : Exp EMIT
```

char

O token `CHAR` é usado para representar caracteres no Forth em seu valor ASCII.

Por exemplo:

```
A CHAR .
```

Isso resultaria na impressão do valor ASCII do caractere 'A', que é 65.

Para isso foi criado o seguinte processo:

```
Frase : Frase CHAR VAR '.'
```

Condicionais

No Forth, as condicionais são implementadas principalmente usando as palavras-chave IF, ELSE e THEN.

```
10 20 > IF
    ." 10 é maior que 20"
THEN
```

```
10 20 > IF
    ." 10 é maior que 20"
ELSE
    ." 10 não é maior que 20"
THEN
```

Para implementar as condicionais, foram criados os tokens 'IF', 'THEN' e 'ELSE'. O processo de implementação envolve o uso de dois padrões sintáticos distintos para lidar com as estruturas condicionais.

```
Condicional1 : IF FuncCont THEN

Condicional2 : IF FuncCont ELSE FuncCont THEN
```

Na produção Condicional1 é lida com o caso simples de uma estrutura IF ... THEN. Ela verifica se a condição é verdadeira e executa o código no bloco IF se for.

Na produção Condicional2 tanto o bloco IF quanto o bloco ELSE são considerados. Se a condição no bloco IF for verdadeira, o código no bloco IF é executado; caso contrário, o código no bloco ELSE é executado.

Em ambas as produções as variável parser.cont é utilizada para criar labels únicas para cada condicional.

No caso do bloco IF simples, a label endif é utilizada para marcar o fim do bloco IF.

Já no caso do bloco IF/ELSE, são utilizadas duas labels: else para marcar o início do bloco ELSE e endif para marcar o fim do bloco IF/ELSE.

Essas labels garantem o controlo do fluxo de execução do código nas condicionais.

Ciclos

Em Forth existem diferentes tipos de ciclos. Uma das estruturações possíveis para os ciclos é a seguinte:

```
10 1 do
    i .
loop
```

```
variable a

20 a !
a @ 10 do 5 . loop
```

Outra formatação possível é a seguinte:

```
2 begin
    dup .
    1 +
    dup
    10 >
until
```

Relativamente ao analisador sintático, isto obrigou-nos à criação de mais 4 tokens (*DO*, *LOOP*, *BEGIN* e *UNTIL*) para que os ciclos possam ser.

Para os implementar no analisador léxico, foram criada mais duas frases (*p_Frase9* e *p_Frase10*), seguindo a seguinte estrutura:

```
Frase : Frase Ciclo1
      | Frase Ciclo2
```

Os ciclos, por sua vez, seguem a seguinte formatação:

```
Ciclo1 : Fatores DO corpoCiclo LOOP

Ciclo2 : Fatores BEGIN FuncCont UNTIL
       | BEGIN FuncCont UNTIL
```

No caso do primeiro ciclo, par implementar o analisador sintático, primeiro recolhem-se os limites do ciclo. Para as armazenar, reservam-se 2 espaços de memória na stack global da VM. Após isso, é criada uma label para o ciclo. É introduzido o corpo do ciclo nessa label. No ciclo é ainda incrementada a variável que verifica os limites do ciclo e é verificada se a condição de fim de ciclo se verifica. Se se verificar, então o ciclo é terminado. Caso contrário, ocorre um salto para a label anteriormente criada. Neste caso, é ainda implementado *i*. Para o fazer, tivemos de perceber se as operações contidas no ciclo eram 'i' ou diferentes de 'i'. Se fossem diferentes, escrevia-se para a VM o próprio código da instrução. Caso contrário, íamos ver o valor do limite inferior do ciclo e escrevia-se na VM esse valor.

No caso do segundo ciclo, começa por identificar se recebe argumentos. Em caso afirmativo, entra no caso do *Ciclo2* em que começa por escrever os argumentos para a stack. Após isso, é criada uma label para o ciclo e posteriormente é escrito o código do corpo do ciclo. No final, verifica-se se o ciclo terminou, a partir do valor da última instrução contida no ciclo. Em caso afirmativo sai do ciclo. Caso contrário, faz um salto para a label criada. No caso de não conter argumentos, entra no outro caso do *Ciclo2*, em todo idêntico ao anteriormente descrito, retirando-se apenas o processo de escrita dos argumentos para a stack.

As labels dos ciclos tem um nome estruturado, sendo ele *ciclo*<*num*> em que <\num> é o número de ciclo, variável do parser que é incrementada quando um ciclo é criado.

Comentários

No Forth, para realizar comentários, basta colocar o bloco que se deseja que seja comentado dentro de parênteses curvos.

Para identificar os comentários, o analisador léxico teve de ser adaptado para reconhecer mais um token, o *COMMENT*. Esse token reconhece todo o conteúdo presente dentro de parêntese.

O analisador sintático, em qualquer parte do código, seja numa linha apenas formada por um comentário, dentro de ciclos ou funções, se detetar um comentário, limita-se a não escrever nenhum código para a VM.

Conclusão

Neste trabalho, desenvolvemos um compilador para a linguagem Forth, que converte código Forth em instruções para uma máquina virtual (VM). Focámo-nos em expandir o analisador sintático para suportar os vários recursos da linguagem, como expressões relacionais, impressão de caracteres e strings, condicionais, ciclos e funções.

No geral, o trabalho proporcionou uma compreensão mais profunda da linguagem Forth e maquina virtual e dos desafios associados ao desenvolvimento de um compilador. A expansão do analisador sintático permitiu suportar uma variedade de recursos adicionais, tornando o compilador mais robusto e funcional. Este trabalho representa um passo importante na nossa jornada como programadores, ajudando a consolidar os conhecimentos sobre expressões regulares, análise léxica e sintática, além de promover uma maior familiaridade com a implementação de linguagens de programação e o desenvolvimento de compiladores.