Universidade de Évora
Escola de Ciências e Tecnologia
Licenciatura em Engenharia Informática
Linguagens de Programação
Ano Letivo 2019/2020

Leitura de programas TISC

3 de Maio de 2020

Docente: Prof. Teresa Gonçalves
Discentes:

- Ana Sapata, 42255
- Yaroslav Kolodiy, 39859

Conteúdo

1 Introdução **2**

2 Programa TISC e suas instruções **3**

3 Implementação **4**

3.1 TISC.py 4
3.2 tekens.py 5
3.3 registos.py 5
3.4 main.py 6
3.5 Makefile 7

4 Conclusão **8**

1 Introdução

Este trabalho está englobado na disciplina de Linguagens de Programação, da Licenciatura em Engenharia Informática, da Universidade de Évora. Com o mesmo pretende-se através de um programa realizado em Python, ler um programa TISC e carregar o mesmo na memória de instruções.

Sendo que uma máquina TISC executa programas escritos numa linguagem estrutura em blocos e a mesma é composta por três zonas de memória distintas e dois registos. As zonas de memória são a memória de instruções, a qual será o output deste trabalho, a pilha de avaliação e a memória de execução. Os registos são o *environment pointer* (**EP**) e o *program counter* (**PC**).

2 Programa TISC e suas instruções

Como já referido o output do programa Python será a memória de instruções da máquina TISC, ou seja, a estrutura onde serão guardadas as instruções que o programa deverá executar. Para tal é então necessário perceber como são as instruções de um programa TISC e como são executadas as mesmas.

As instruções podem ser ou não identificadas por uma etiqueta, sendo que no início de cada programa a primeira instrução deverá ter a etiqueta *program*. Geralmente as instruções serão executadas pela ordem em que se encontram, exceto se existirem instruções de salto ou as instruções *call* e *return*.

Para se construir a sintaxe verificou-se que para alem das etiquetas as instruções teriam o seu nome e até dois argumentos, sendo possível terem zero argumentos. As instruções poderão ser:

- **instruções aritméticas** - são compostas apenas pelo seu nome e possível etiqueta, e a sua operação é desempenhada utilizando elementos da pilha de avaliação;
- **instruções para manipulação de inteiros** - são compostas pelo seu nome e um argumento do tipo inteiro. Existem apenas uma instrução nesta categoria, denominada *push_int* e que coloca o inteiro na pilha de avaliação;
- **instruções de acesso a variáveis** - são declaradas no inico dos blocos, para alem do seu nome, são constituídas por 2 argumentos do tipo inteiro. Estas instruções leem e escrevem o valor das variáveis;
- **instruções de acesso a argumentos** - são compostas pelo seu nome e 2 argumentos do tipo inteiros e al como as anteriores leem e escrevem, mas neste caso, argumentos em vez de variáveis;
- **instruções para chamada de funções** - estas instruções poderão ter entre 0 e 2 argumentos, alem do seu nome e da possível etiqueta. As instruções desta categoria servem essencialmente para efetuar a chamada de funções, bem como para processar o início e fim da sua execução, manipulando os registos de ativação;
- **instruções de salto** - são instruções compostas pelo seu nome e um só argumento, sendo este a etiqueta da instrução para onde o programa deverá saltar, ou seja, a etiqueta da próxima instrução a executar;
- **instruções de saída** - para alem do seu nome, poderão ter ainda um argumento sendo este uma *string*, ou nenhum. Estas instruções servem para enviar informação para o ecrã.

3 Implementação

Para se poderem ler ficheiros *.tisc* e criar a memória de instruções para o mesmo, foi necessário verificar a sintaxe das instruções através da estrutura das instruções observada anteriormente. Além disso foram utilizados os packages *ply.lex* e *ply.yacc* para a construção da parte lexical e criação do *parser*.

O programa é composto por 5 ficheiros essenciais:

- **TISC.py** - este ficheiro contem essencialmente a classe TISC e as classes para cada instrução;
- **tekens.py** - este ficheiro contem os *tokens* necessários para a construção da parte lexical;
- **registos.py** - este ficheiro contem a informação para o *parser* e a função *main* que irá efetuar a leitura do ficheiro *.tisc*;
- **main.py** - este ficheiro executa a função *main* previamente definida;
- **Makefile** - é através da execução deste ficheiro que irá ser executado o nosso programa.

Para além destes ficheiros cada vez que são lidos ficheiros *.tisc* são ainda gerados pelo *parser* os ficheiros **parser.out** e **parsetab.py** relacionados com a gramática.

De seguida, irão ser apresentados os cinco ficheiros essenciais mais sucintamente.

3.1 TISC.py

Este ficheiro é então constituído pela classe TISC, pela classe *Instruction* e por várias subclasses desta, correspondendo cada uma a uma instrução do programa.

- TISC

  Um objeto TISC é composto por dois registos, **PC** e **SP**, tal como referido, e por:

  - **labels** - um dicionário que guarda as etiquetas, bem como o número que indica a que instrução pertence a referida label, de modo a ser possível voltar à mesma, quando for efetuado um salto para a respetiva etiqueta;
  - **instructions** - um array que guarda as instruções do programa. Este array será então a memória de instruções que irá ser o output final do programa;
  - **avaliation_stack** - um array que será a pilha de avaliação;
  - **number_instructions** - irá guardar o número de instruções que o programa tem;

  Quando é lida uma instrução que contem uma etiqueta é utilizado o método **new_label (self, label)** que adiciona o mesmo ao dicionário *labels* e o número da respetiva instrução no programa.

  Para adicionar uma nova instrução à memória de instruções é utilizado o método **add_instruction(self, instruction)**. Este método adiciona a instrução ao array *instructions* e aumenta o *number_instructions* em 1.

  Como o objetivo é mostrar a memória de instruções, existe o método **print_instructions (self)** que realiza o print de cada instrução para o utilizador final. Existe ainda o método **print_labels(self)** que faz o mesmo, mas neste caso para o dicionário *labels* que contem as etiquetas.

  Por implementar está o método **execute(self)** que irá executar o programa lido e carregado na memória de instruções.

- Instruction

  As instruções são então compostas pelo seu nome e argumentos. Todas elas serão constituídas pelo método **___init___** que inicializa as mesmas e coloca o nome igual ao passado no método, e no caso de existirem argumentos coloca estes com o valor que foi passado no método. No caso de não serem passados argumentos no método, por defeito ficará tudo como sendo *None*.

  Todas terão também o método **execute** que irá executar a mesma.

  O método **___repr___** retorna uma maneira de representar as instruções, ou seja, como estas irão ser mostradas no ecrã quando será feito print das mesmas.

  Para além da classe global existem subclasses. Todas as subclasses têm um método **___init___** onde é passado o nome da respetiva função como *default* e os respetivos argumentos no caso de a mesma receber argumentos. De seguida é chamado o mesmo método da sua super classe.

  Todas as subclasses têm o método **execute**, onde será posteriormente implementado o funcionamento de cada instrução.

3.2 tekens.py

O ficheiro teken.py utiliza o package *ply.lex*. Através desta biblioteca o programa que vai ser lido, irá ser separado segundo os *tokens* previamente fornecidos, os quais têm uma coleção de regras de expressões regulares.

Deste modo é inicialmente fornecida uma lista ***tokens***, onde estão todos os *tokens* utilizados nos programas TISC, desde os nomes das instruções, ao tipo de argumentos, por exemplos *INTEIRO*, *STRING*, entre outros.

Para cada *token* da lista foi criada uma função cujo seu nome é **t_<token>** onde *<token>* é substituído pelo *token* em questão. Todas as funções recebem um argumento **t**, que é uma instância do *LexToken*, sendo a função inicialmente composta pela expressão regular associada ao respetivo *token*.

De seguida é atribuído ao parâmetro *value* do *LexToken* (*t.value*) o valor correspondente ao *token*, ou seja, no caso das funções será o seu nome, no caso de um inteiro será feito convertido o texto recebido para um inteiro, no caso de uma *string* o texto recebido será convertido para uma *string*.

Em alguns casos mais simples não é necessário criar a função **t_<token>** sendo apenas especificada a expressão regular que o identifica, como no caso dos *DOIS_PONTOS* que ficou

```
1 t_DOIS_PONTOS = r':'
```

Por fim, é então construído o *lexer* através da chamada da função *lex.lex()*.

3.3 registos.py

Este ficheiro importa o anterior *tekens.py* uma vez que os *tokens* serão necessários no *parser*. É também utilizado o package *ply.yacc*, este geralmente é utilizado para reconhecer a sintaxe da linguagem, utilizando os *tokens* e as regras gramaticais, para construir as árvores de sintaxe abstrata.

Começou-se por criar um objeto do tipo TISC, pois irá ser necessário adicionar as instruções à sua memória de instruções à medida que as mesmas forem lidas e analisadas.

Neste ficheiro observam-se várias funções, sendo que todas elas, exceto o *main*, definem as regras gramaticais, sendo para tal apresentada a sintaxe em termos de gramática BNF que será utilizada para analisar os inputs fornecidos ao *parser*.

- **p_programa(t)** - esta função define a regra gramatical para o programa;
- **p_programa_empty(t)** - esta função define a regra gramatical para o caso do programa ser vazio;
- **p_etiqueta(p)** - esta função define a regra gramatical para as etiquetas, sendo as mesmas compostas por um identificador seguido de dois pontos. Neste caso será adicionado o identificador ao dicionário *labels* do objeto TISC previamente criado;
- **p_etiqueta_empty(p)** - esta função define a regra gramatical para o caso da etiqueta ser vazia;
- **p_instrucao(p)** - nesta função é definida a regra para as instruções que não têm argumentos, ou seja, aquelas que são apenas constituídas pelo seu nome, como o caso de todas as instruções aritméticas, entre outras. A instrução será adicionada à memória de instruções do objeto TISC, sendo criado um objeto do respetivo tipo, de acordo com o valor de *p[1]*;
- **p_instrucao_arg1(p)** - nesta função são definidas as regras para as instruções que alem do seu nome contêm um argumento. Neste caso quando a instrução é adicionada à memória de instruções, de acordo com o seu valor de *p[1]*, é ainda passado à mesma o seu argumento com o valor de *p[2]*;
- **p_instrucao_arg2(p)** - nesta função são definidas as regras para as instruções que contêm o seu nome e dois argumentos, sendo as mesmas adicionadas à memória de instruções de acordo com o seu valor de *p[1]*. Nos argumentos são passados o valor de *p[2]* e *p[3]*;
- **p_error(p)** - esta função é utilizada no caso de ocorrer algum erro, enviando para o ecrã a mensagem *"Syntax error"*.

Por fim existe a função **main(print_inst, print_labels, filepath = None)** que é diferente das restantes. Esta função inicializa o *parser* através da chamada da função *yacc.yacc()*.

O argumento *filepath*, é referente ao caminho para o ficheiro que se pretende ler, no caso deste argumento não ser fornecido à função *main* o mesmo será pedido ao utilizador. Quando existir um valor no *pathfile*, irá então ser lido o ficheiro que se encontra no respetivo caminho fornecido.

O *parser* irá analisar cada linha do ficheiro. No caso de ser passado o argumento print_inst na função *main* irá ser mostrada ao utilizador a memoria de instruções do objeto TISC. No caso de ser passado o argumento *print_labels*, irá ser mostrado o dicionário que contem as *labels* do objeto TISC.

3.4 main.py

Neste ficheiro é executada a função *main* que foi criada no ficheiro registos.py. É ainda utilizado o package *argparse* que será utilizado para passar argumentos à execução do ficheiro.

Este ficheiro poderá ser executado através do comando *python3 main.py*, sendo que neste caso como não são passados quaisquer argumentos irá ser pedido o caminho do ficheiro a ler ao utilizador, mas não irá mostrar a memória de instruções nem as *labels*. Para tal será necessário passar alguns argumentos a seguir ao nome do ficheiro sendo estes:

- **-m** - ao passar este argumento será mostrada a memória de instruções ao utilizador;
- **-l** - ao passar este argumento serão mostradas as etiquetas existentes no programa ao utilizador;

- **-filepath** - ao passar este argumento seguido do caminho para o ficheiro, já não será perguntado ao utilizador qual o caminho do ficheiro que pretende ler.

Tudo isto é possível através das chamadas da função *parser.add_argument* utilizadas neste ficheiro. Por fim é então chamada a função **main()** do ficheiro registos.py com os respetivos argumentos.

3.5 Makefile

Com a existência deste ficheiro já não será necessário correr o ficheiro main.py pois este já irá executar o mesmo.

Na linha de comandos poderá correr os seguintes comandos:

- **make run** - este comando irá correr o ficheiro main.py com os argumentos -m -l, ou seja, irá mostrar a memoria de instruções e as etiquetas. No entanto irá ser perguntado ao utilizador qual o caminho para o ficheiro que pretende ler;
- **make run_file** - a este comando deverá ainda adicionar o caminho do ficheiro que pretende ler, ou seja, no terminal deverá escrever *make run_file FILE=<pathfile>*. Deste modo irá ser logo lido o ficheiro e mostrada a respetiva memória de instruções e etiquetas;
- **make help** - ao correr este comando irá ser mostrada a ajuda para o ficheiro main.py, ou seja, os argumentos que podem ser fornecidos a este e o que cada um significa;
- **make install** - a este comando deverá adicionar o $PV = pip$ ou $PV = pip3$, consoante a versão do python que utiliza, e o mesmo irá instalar o package ply;
- **make clean** - ao correr este comando irão ser removidos os ficheiros parser.out e parsetab.py que são gerados pelo parser ao ler o programa.

4 Conclusão

Consoante o ficheiro fornecido pelo utilizador e os parâmetros utilizados pelo mesmo, é então fornecida a memória de instruções do programa contido no ficheiro. Tudo isto é possível devido à utilização dos packages *ply.lex* e *ply.yacc* na construção da gramática e análise de sintaxe e posterior leitura do ficheiro.