Elementos de Sistemas 2017.1
Sprint 9 – Compilador de Alto Nível
Prazo de entrega: 2/06/17

Descrição:

A entrega dessa *sprint* é um compilador que vai traduzir programas escrito na linguagem de alto nível Jack para a linguagem de máquina virtual a pilha do Z0. Este código possui uma complexidade maior que os anteriores e é recomendada ser feita em etapas, façam planos para entregas parciais para reduzir riscos de não completarem a entrega no final.

Organização:

A equipe deverá se organizar para que o compilador gere os códigos dos programas de teste em linguagem de máquina virtual a pilha. Todos os membros de cada grupo devem participar do desenvolvimento. Porém não aceite código pronto de outros. O facilitador é responsável pela organização da equipe e pela completude e consistência do *branch master*. Ele deverá chegar num consenso com a equipe para datas de entregas parciais. Além disso deverá estar pronto para conduzir, no encontro seguinte ao prazo de entrega, reuniões de Revisão e Retrospectiva.

Entrega:

As entregas dos grupos serão consideradas concluídas quando o *pullrequest* for feito e a tarefa tenha sido movida para DONE no Trello. O facilitador terá de verificar se tudo funciona corretamente e emitir um relatório no Trello caso negativo, garantindo que a *sprint* do Trello foi fechada.

Rua Quatá, 300 – Vila Olímpia 04546-042 São Paulo SP Brasil
55 11 45042400 www.insper.edu.br

Instruções:

O compilador de linguagem de alto nível Jack é um programa que carrega um arquivo texto em ASCII com as instruções em Jack (extensão .jack) e deve retornar um arquivo também em texto com as instruções em linguagem de máquina virtual a pilha (extensão .vm). Para testar se o programa funciona, utilize a máquina virtual fornecida pelo site Nand2Tetris, lá você pode carregar um programa e verificar a saída da aplicação. Após validar seu desenvolvimento, submeta um *pull request* com suas contribuições para que o facilitador do projeto aprove e faça o *merge* no *branch master*. O título do *pull requeste* deve ser o nome dos módulos desenvolvidos para facilitar a identificação do mesmo durante os merges. Não há necessidade de colocar os nomes de quem implementou, eles já estão no controle de versão e no Trello.

Módulos:

Devem ser desenvolvidos em Java:

- **JackCompiler:** controlador que cria e invoca os outros módulos;
- **JackTokenizer:** analisador léxico;
- **SymbolTable:** gerenciador a tabela de símbolos;
- **VMWriter:** gerador da saída do código de máquina virtual**;**
- **CompilationEngine:** rotina de compilação recursiva top-down.

Links:

- Github: https://github.com/ElementosDeSistemas/Z0
- Trello: https://trello.com/engcompinsper2017

Rua Quatá, 300 – Vila Olímpia 04546-042 São Paulo SP Brasil
55 11 45042400 www.insper.edu.br

Elementos Léxicos:	A linguagem Jack possui cinco tipos de elementos terminais (tokens):
keyword:	**'class'** | **'constructor'** | **'function'** | **'method'** | **'field'** | **'static'** | **'var'** | **'int'** | **'char'** | **'boolean'** | **'void'** | **'true'** | **'false'** | **'null'** | **'this'** | **'let'** | **'do'** | **'if'** | **'else'** | **'while'** | **'return'**
symbol:	'{' | '}' | '(' | ')' | '[' | ']' | '.' | ',' | ';' | '+' | '-' | '*' | '/' | **'&'** | '|' | '<' | '>' | '=' | '~'
integerConstant:	Um número decimal entre 0 e 32767.
StringConstant:	Uma sequencia de caracteres Unicode não incluindo aspas ou quebra de linha.
identifier:	Sequencia de letras, dígitos ou underscore ('_'), não podendo começar com dígito.
Estrutura do Programa:	Um programa Jack é uma coleção de classes, cada uma aparecendo em um arquivo separado, sendo a classe a unidade de compilação. Classes são uma sequência de *tokens* estruturados conforme a seguinte gramática livre de contexto:
class:	'**class**' className '{' classVarDec* subroutineDec* '}'
classVarDec:	('**static**' | '**field**') type varName (',' varName)* ';'
type:	'**int**' | '**char**' | '**boolean**' | className
subroutineDec:	('**constructor**' | '**function**' | '**method**') ('**void**' | type) subroutineName '(' parameterList ')' subroutineBody
parameterList:	((type varName) (',' type varName)*)?
subroutineBody:	'{' varDec* statements '}'
varDec:	'**var**' type varName (',' varName)* ';'
className:	identifier
subroutineName:	identifier
varName:	identifier
Declaração:	
statements:	statement*
statement:	letStatement | ifStatement | whileStatement | doStatement | returnStatement
letStatement:	'**let**' varName ('[' expression ']')? '=' expression ';'
ifStatement:	'**if**' '(' expression ')' '{' statements '}' ('**else**' '{' statements '}')?
whileStatement:	'**while**' '(' expression ')' '{' statements '}'
doStatement:	'**do**' subroutineCall ';'
ReturnStatement:	'return' expression? ';'
Expressões:	
expression:	term (op term)*
term:	integerConstant | stringConstant | keywordConstant | varName | varName '[' expression ']' | subroutineCall | '(' expression ')' | unaryOp term
subroutineCall:	subroutineName '(' expressionList ')' | (className | varName) '.' subroutineName '(' expressionList ')'
expressionList:	(expression (',' expression)*)?
op:	'+' | '-' | '*' | '/' | **'&'** | '|' | '<' | '>' | '='
unaryOp:	'-' | '~'
KeywordConstant:	'true' | 'false' | 'null' | 'this'

Rua Quatá, 300 – Vila Olímpia 04546-042 São Paulo SP Brasil
55 11 45042400 www.insper.edu.br