Compiladores – Linguagem para Manipulação de Tabelas

ALFA-D: Documentação

Lucas Silva	85036
João Alegria	85048
André Pedrosa	85098
Filipe Pires	85122
Duarte Castanho	85133

Introdução

Este documento foi produzido de forma a dar a conhecer a estrutura da linguagem de programação para manipulação de tabelas, a que demos o nome de ALFA-D. O documento abarca a descrição da solução por nós implementada no âmbito da unidade curricular Compiladores, utilizando os conceitos absorvidos ao longo do semestre. Nesta descrição são enumeradas as capacidades da linguagem, em conjunto com as funcionalidades implementadas para o suporte e manipulação de tabelas. A informação disponibilizada é acompanhada por código descritivo, diretamente retirado da linguagem e de programas exemplo escritos na mesma.

O trabalho envolve duas linguagens: uma para o compilador, que valida programas de manipulação de tabelas; outra para ler e validar informação estruturada contida em ficheiros CSV e representativa de tabelas. A primeira linguagem trata-se da principal, onde ocorre a compilação do conteúdo para código Java, já a segunda envolve um interpretador do conteúdo de ficheiros de forma a capacitar o utilizador de utilizar tabelas externas ao programa.

É esperado que o leitor termine a leitura deste relatório com uma noção geral da forma como a linguagem ALFA-D está estruturada e do tipo de programas que podem ser escritos com ela. Em conjunto com o código desenvolvido, são disponibilizados alguns programas exemplo. É também o nosso intuito dar a conhecer o trabalho futuro caso o projeto tivesse continuidade após a entrega final e as considerações finais acerca do projeto.

ALFA-D

A motivação por detrás do desenvolvimento da linguagem de programação ALFA-D foi procurar ter disponível uma ferramenta de fácil compreensão capaz de manipular tabelas de informação de uma forma dinâmica e de conseguir aceder/guardar a informação de/em ficheiros externos, num formato CSV.

A implementação está feita de uma forma que permite ao programador desenvolver scripts de interação de tabelas com variáveis de tipos primitivos, conseguindo também executar impressões de dados quer na consola quer em ficheiros.

Ainda que muito limitada, esta linguagem é de certa forma orientada a tabelas e segue algumas características típicas das linguagens de programação SQL (a nível de operações sobre tabelas) e Java (a nível de sintaxe).

Na tabela a seguir apresentamos a lista dos tipos de dados suportados pela linguagem e as respetivas descrições. Dada a natureza da linguagem, é definido que todos os tipos de dados não diretamente relacionadas com tabulação (bool, int, real, String) são consideradas Primitivas, enquanto que as restantes (Table, Column, Line) são consideradas Não-Primitivas.

Tipo de Dados	Descrição
bool	Tipo de dados Primitivo, com apenas dois possíveis valores: *true* e *false*. Este tipo de dados representa um bit de informação.
int	Tipo de dados Primitivo, um inteiro com 32 bits com sinal e complemento a 2. Tem um valor mínimo de -2^{63} e máximo de $2^{63}-1$.
real	Tipo de dados Primitivo, um ponto flutuante de 64 bits com dupla precisão.
String	Tipo de dados Primitivo, conjunto de caracteres entre aspas. O número de bits depende do tamanho da *String*.
Table	Tipo de dados Não-Primitivo, estrutura de dados organizada por linhas e colunas (cada uma com um cabeçalho). Cada coluna suporta apenas um tipo de dados, definido pelo programador. O número de bits depende do tamanho da *Table*.
Column	Tipo de dados Não-Primitivo, estrutura de dados com um cabeçalho organizada por linhas. Este tipo de dados é um caso particular de uma *Table*, com certas operações limitadas. O programador define o tipo de dados que a coluna suporta. O número de bits depende do tamanho da *Column*.
Line	Tipo de dados Não-Primitivo, estrutura de dados organizada por colunas e uma só linha. Este tipo de dados é um caso particular de uma *Table*, com certas operações limitadas. O programador define o tipo de dados que cada coluna suporta. O número de bits depende do tamanho da *Column*.

De seguida é apresentada a tabela com a sintaxe das funcionalidades e operações base da linguagem ALFA-D, bem como as respetivas descrições.

Sintaxe	Descrição
-- single line comment	Comentário de uma linha. Todo o conteúdo da linha da instrução a partir do símbolo '--' é ignorado e descartado pelo compilador.
-* multi line comment *-	Comentário de número variável de linhas. Todo o conteúdo entre os símbolos '-*' e '*-' é ignorado e descartado pelo compilador.
DataType ID;	Declaração de uma variável do tipo Primitivo. DataType: tipo de dados da variável; ID: identificador da variável.
ID = Value;	Definição de uma variável (atribuição de um valor) do tipo Primitivo. ID: identificador da variável; Value: valor a inserir no endereço de memória da variável.
DataType ID = Value;	Instanciação de uma variável (declaração com atribuição de um valor) do tipo Primitivo. DataType: tipo de dados da variável; ID: identificador da variável; Value: valor a inserir no endereço de memória da variável.
Value Operator Value;	Operação aritmética sobre valores. Value: valor de um dos tipos de dados primitivos suportados
if (Condition) { }	Estrutura condicional inicial. O código descrito no interior dos símbolos '{' e '}' é executado caso a condição se verifique. Condition: condição (valor booleano) a ser verificada.
else if(Condition) { }	Estrutura condicional intermédia ou final. Esta estrutura é sempre precedida de um *If* ou de outro *Else If*. O código descrito no interior dos símbolos '{' e '}' é executado caso a condição se verifique e a condição da estrutura anterior não se verifique. Condition: condição (valor booleano) a ser verificada.
else { }	Estrutura condicional final. Esta estrutura é sempre precedida de um *If* ou de um *Else If*. O código descrito no interior dos símbolos '{' e '}' é executado caso a condição da estrutura anterior não se verifique.
for (Init; Term; Inc) { }	Estrutura cíclica. Método compacto de iterar sobre um intervalo de valores. O código descrito no interior dos símbolos '{' e '}' é executado repetidamente enquanto a condição presente no *Term* se verificar. Init(ialization): definição ou instanciação de uma variável do tipo int; executa uma vez e inicializa o ciclo. Term(ination): quando a condição aqui descrita deixa de se verificar, o ciclo termina; esta condição envolve a variável presente no primeiro campo. Inc(rement): invocada no final de cada iteração e incrementa / decrementa a variável presente no primeiro campo.
while (Condition) { }	Estrutura cíclica. O código descrito no interior dos símbolos '{' e '}' é executado iterativamente enquanto a condição se verifica. Condition: condição (valor booleano) a ser verificada.
do { } while (Condition);	Estrutura cíclica. Funciona da mesma forma que o *While*, mas o código descrito no interior dos símbolos '{' e '}' é obrigatoriamente executado uma vez antes de ser feita a verificação da condição. Condition: condição (valor booleano) a ser verificada.
def Type Name Args { Code};	Estrutura para definição de uma função em que Type é o tipo do objeto que a função retorna, Name é o nome da função, Args é a lista de argumentos separada por vírgulas e Code são os statements da função.

A terceira tabela deste documento enumera a sintaxe de todo o tipo de utilização de tabelas suportado pela linguagem, em conjunto com a descrição de cada instrução.

Sintaxe	Descrição
Table ID = create (Arguments);	Estrutura de construção de tabela segundo os argumentos do tipo “Type HeaderName, … ,Type HeaderName”.
Table ID = load (Filename);	Estrutura de construção de tabela que faz import dos dados diretamente de um ficheiro CSV indicado em “FileName”.
Column ID = createCol(Arguments);	Estrutura de construção de coluna que recebe como argumentos “Type HeaderName”.
Line ID = <Arguments>;	Estrutura de construção de linhas segundo os argumentos do tipo “Arg1, Arg2, …, ArgN”.
Table union Table	Método para proceder à união de duas tabelas.
Table intersect Table	Método para proceder à interseção de duas tabelas.
Table difference Table	Método para proceder à diferença entre duas tabelas.
Table join Table	Método para proceder ao “join” entre duas tabelas.
Table join Table on Conditional	Método para proceder ao “join” entre duas tabelas segundo uma condição de comparação entre duas colunas. Conditional é do tipo “arg1 (‘<’ ’>’ ‘<=’ ‘>=’ ‘==’ ‘!=’) arg2” .
Table remove where (Conditional)	Método para remoção proceder à remoção de elementos da tabela segundo certa condição. Conditional pode ser tipo “arg1 (‘<’ ’>’ ‘<=’ ‘>=’ ‘==’ ‘!=’) arg2” ou “@Number”, que remove a linha na posição Number ou “@(Number--Number)” ou “@*” que procede à remoção de todos os elementos.
Table increase (Arguments)	Método para fazer aumento lateral da tabela, Arguments pode ser do tipo Column ou então do tipo “Type arg0”.
Table decrease (Arguments)	Método para fazer redução lateral da tabela, Arguments é do tipo “Arg0”.
Table add (Arguments)	Método para adicionar linhas à tabela, Arguments pode ser do tipo “Line” que adiciona ao fim ou do tipo “Line, Number” que adiciona a linha na posição indicado por Number.
Table clear(Arguments)	Método para remover elementos da tabela, Arguments pode ser do tipo “ ”, caso se pretendam remover todos os elementos, ou “Column” que apaga uma coluna ou “Line” que apaga uma linha.
Table save(FileName)	Método para guardar a tabela num ficheiro CSV com o nome FileName, caso o ficheiro já exista, a escrita é feita por cima.
Column ID = function()	Método para atribuir os valores de uma coluna conforme o return de uma função previamente definida pelo utilizador.
table.ID	Método que devolve a coluna cujo header seja igual a “ID”.
table[Number]	Método que devolve a coluna no index Number da table.
[Number]table	Método que devolve a linha no index Number da table.
[Number1--Number2]table	Método que devolve uma tabela com as linhas da table que vão desde Number1 até Number2.
table[Number1--Number2]	Método que devolve uma tabela com as colunas da table que vão desde Number1 até Number2.
[N1--N2]table[N3--N4]	Método que devolve a sub-tabela com cujas colunas vão desde N3 a N4 e as linhas vão de N1 a N2.

ALFA-Interpreter

O interpretador é uma linguagem complementar à nossa linguagem principal que tem como objetivo auxiliar-nos na leitura dos ficheiros que possam conter tabelas. Esta leitura é feita para garantir que os ficheiros se encontram num formato pré-definido respeitando certas regras que são validadas lexicamente por essa gramática.

O formato tem de ter um header e tem de ter uma lista de colunas e depois de elements separados por vírgulas, essencialmente, do tipo CSV.

O interpretador não é um compilador e avalia cada linha em runtime e usa Listeners.

Programas Teste

Os programas abaixo descritos têm como objetivo exemplificar a utilização da linguagem ALFA-D. Estes programas foram utilizados como testes durante o desenvolvimento do projeto e mais tarde reorganizados pelas funcionalidades que cada um testa mais aprofundadamente.

Operações Básicas

O código que se segue exemplifica a simples utilização de variáveis do tipo *int*, *real* e *String* em operações aritméticas, com o auxílio de ferramentas como *Loops* e *Conditionals*.

```
--comentário de uma linha

-*
Comentário
multi
linha
*-

String nice = "StringExample";
print(nice + "\n");

int num1 = 123;
print(num1 + "\n");

real num2 = 2.3;
print(num2 + "\n");

bool cond = true;
print(cond + "\n");

real mul = num1 * num2;
print(mul + "\n");

real div = num1 / num2;
print(div + "\n");

int sum = num1 + num2;
print(sum + "\n");

real exp = num1 ^ 2.32;
```

```
print(exp + "\n");

int a;

for(int i = 0; i < 10; i++){
  if(i == 0){
    a == num1;
    print(a + "\n");
  }
  if(cond){
    a++;
    print(a + "\n");
  }
}

int count = 0
while(cond){
  print(count + nice + "\n");
  count = count + 1;
  if(count > 3){
    cond = false;
    print(cond + "\n");
  }
}

do{
  print(count + "OutraStringExemplo\n");
  count++;
}while(count < 5)
```

Funções

Este segundo programa recorre a *Functions* para efetuar operações simples.

```
def void printLine(String info) {
	print(info + "\n");
	return;
}

def int sqrt(int a) {
	int result = a ^ (1/2);
	return result;
}

def bool isGoodName(String name){
	if(name == "Lucas") {
		return true;
	}else{
		return false;
	}
	return;
}

def void work(int a, String name){
	if(isGoodName(name)){
		printLine(name);
		printLine(sqrt(a));
	}
	return;
}

String name = "Lucas";
int a = 3;

work(a, name);
```

Manipulação de Tabelas

O último programa (e o mais complexo) introduz o leitor à utilização das tabelas em programas escritos na linguagem ALFA-D, manipulando-as através de manipulações de tabelas e operações entre as mesmas.

```
Table familia1  = create(String Nome, String Apelido, int Idade, bool Male);
familia1 increase(String Parentesco);

familia1 add(<"Duarte","Castanho",20,true,"Pai">);
familia1 add(<"André","Pedrosa",20,false,"Mae">);
familia1 add(<"Lucas ","Silva",6,false,"Filha">);
familia1 add(<"Joao","Alegria",4,true,"Filho">);
familia1 add(<"Filipe","Pires",11,true,"Filho">);

print(familia1);

Table familia2  = create(String Nome2, String Apelido2, int Idade2, bool Male2, String
Parentesco2);
familia2 add(<"Duarte","Castanho",20,true,"Pai">);
familia2 add(<"Maria","Albertina",20,false,"Mae">);
familia2 add(<"Alberto","Cortez",7,true,"Filho">);
familia2 add(<"Mário","Ravia",12,true,"Tio">);
familia2 add(<"Van","Dogh",32,true,"Cão">);


print("\nUnion the tables:\n");
Table unionTables = familia1 union familia2;
print(unionTables);

print("\nIntersect the tables:\n");
print(familia1 intersect familia2);

print("\nDifference the Tables:\n");
print(familia1 difference familia2);

print("\nJoin the Tables\n");
print(familia1 join familia2 on Nome == Nome2);
```

Trabalho Futuro

Tendo em conta as capacidades atuais da linguagem de programação ALFA-D, consideramos que o passo a seguir caso o desenvolvimento prosseguisse após a entrega final do projeto seria concretizar a funcionalidade de aplicar funções a colunas de tabelas. Isto permitiria ao utilizador definir que colunas teriam comportamentos específicos e definir os próprios comportamentos (p.e.: incrementar automaticamente o valor de uma coluna). Em acréscimo, seria possível definir que os valores de uma coluna fossem calculados a partir dos respetivos valores de outras.

Considerações Finais

Ao fazer uma análise crítica sobre todas as etapas do desenvolvimento da linguagem de manipulação de tabelas e do seu resultado final, podemos concluir que os principais objetivos do projeto foram cumpridos, dentro dos prazos estabelecidos, com a robustez, a estrutura e a performance que procurávamos.

A nível da performance e do empenho dos elementos do grupo de trabalho, podemos afirmar que foram bastante positivos, que houve um ritmo de trabalho constante e uma distribuição de tarefas equilibrada, aproveitando os pontos fortes de cada elemento o melhor possível. Podemos também dizer que tivemos gosto em criar uma linguagem de programação útil, ainda que relativamente primitiva, e que consideramos que os conhecimentos por nós absorvidos ao longo deste projeto serão uma mais valia para nós no futuro.

Repositório

https://code.ua.pt/git/compiladores-1718-g11

Fontes de Pesquisa:

https://elearning.ua.pt/course/view.php?id=4637
https://docs.oracle.com/javase/8/docs/