UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
UNIDADE ACADÊMICA DE GARANHUNS Av. Bom Pastor, s/n – Boa Vista – CEP 55292-270 – Garanhuns-PE
Telefones: (087) 3761 -0882 e 3761-0969

CURSO: Bacharelado em Ciência da Computação

Período Letivo: 2018.2

Disciplina: Compiladores

Docente: Igor Medeiros Vanderlei

Discentes: Cleyton Silva Batista e Luan Lins

Compilador

Um compilador é um programa de sistema que traduz um programa descrito em uma linguagem de alto nível para um programa equivalente em código de máquina para um processador. Em geral, um compilador não produz diretamente o código de máquina mas sim um programa em linguagem simbólica *(assembly)* semanticamente equivalente ao programa em linguagem de alto nível. O programa em linguagem simbólica é então traduzido para o programa em linguagem de máquina através de montadores.
Para desempenhar suas tarefas, um compilador deve executar dois tipos de atividade. A primeira atividade é a **análise** do código fonte, onde a estrutura e significado do programa de alto nível são reconhecidos. A segunda atividade é a **síntese** do programa equivalente em linguagem simbólica. Embora conceitualmente seja possível executar toda a análise e apenas então iniciar a síntese, em geral estas duas atividades ocorrem praticamente em paralelo.

Sintaxe de Lua

Palavras reservadas:

and, break, do, else, elseif, end, false, for, function, if, in, local, nil, not, or, repeat, return, then, true, until, while.

Comentário

— comentário de linha

–[[comentário de
bloco]]

Variáveis

LUA é case sensitive

Existem três tipos de variáveis, são elas: Globais, Locais e Tabelas.

Variáveis globais são aquelas que podem ser acessadas em qualquer lugar do algoritmo. Para declarar as variáveis globais basta escrever o nome desde que não seja uma palavra reservada ou que contenha caracteres especiais.
Ex.: número, _valor

Variáveis locais são aquelas que só podem ser acessadas dentro do mesmo bloco em que foi declarada. Para declarar as variáveis locais tem que colocar a palavra 'local' na frente.
Ex.: local número, local _valor

Tabelas

Tabelas em LUA podem ser usadas como vetores, matrizes, listas, filas, pilhas, conjuntos, hash. Veremos agora como manipular tabelas em LUA. A representação de uma tabela se dá através de variável = { }. Podem ser usados qualquer tipo de dados como índice exceto o tipo nil. Para acessar um elemento na tabela pode utilizar qualquer uma dessas formas: variável[valor] ou variável.valor. Quando não se define o índice desejado para aquele valor ele é colocado em uma sequência automática, por exemplo:
a = { 5, 3, 4, 8}; é o mesmo que dizer a[1]=5, a[2]=3, a[3]=4 e a[4]=8;
Outra forma de armazenar é colocando o índice desejado aos valores, por exemplo:
a = { [f(1)] = g; "x", "y"; x = 1, f(x), [30] = 23; 45 } isso é o mesmo que:

```
do
local t = {}
t[f(1)] = g
t[1] = "x"         — primeira exp
t[2] = "y"         — segunda exp
t.x = 1             — t["x"] = 1
t[3] = f(x)       — terceira exp
t[30] = 23
t[4] = 45          — quarta exp
a = t
end
```

Comandos de controle

De Decisão:

```
if(expressão)then
– -bloco de comandos
{elseif (espressão) then
– -bloco de comandos
{else
– -bloco de comandos
}}
end
```

Obs: o trecho que está entre { } é por que pode ser omitido.

– -De repetição:

for v1, v2, v3 do
– -bloco de instruções
end

Onde: v1 é a variável para a variação do índice do for, v2 é a variável do valor final (valor limite), v3 é a variável do valor de variação do índice do for (varia de quanto em quanto).

while (condição) do
– -bloco de instruções
end

repeat
– -bloco de instruções
until (condição)

Obs: Nas estrurutas de repetição é possivel a utilização do comando break para interromper a execução do laço durante sua execução.

Operadores

Op. Aritméticos

+ (Soma)
– (Subtração ou unário (deixa o numero negativo quando utilizado sozinho))
* (Multiplicação)
/ (Divisão)
% (Resto)
^ (Expoente)

Op. Relacionais

= (Atribuição)
== (igual)
~= (Diferente)
< (Menor)
> (Maior)
<= (Menor ou igual)
>= (Maior ou igual)

Op. Lógico

and (e lógico)
or (ou lógico)
not (negação lógica)

Op. de Concatenação
.. (concatenação)

Op. de Tamanho
(pega o índice anterior ao valor nil de uma tabela)

Precedência dos Operadores
1º ^
2º not, #, – (unário)
3º *, /, %
4º +, –
5º ..
6º >, >=, <, <=, ==, ~=
7º and
8º or
Obs: Para os operadores de mesma precedência eles são associativos à esquerda com a exceção do ^ e do .. que são a direita.

Tipos de dados

LUA possui uma tipagem dinâmica, ou seja, o tipo está diretamente nos dados e não as referências.
Em LUA existem 8 tipos de dados, são eles:
nil (ausencia de valor)
boolean (apenas true ou false)
number (apenas tipos numéricos)
string (cadeia de caracteres)
function (funções)
userdata (armazena uma referência de algum ponteiro da linguagem C)
thread (representa fluxos de execução independentes)
table (esse tipo são de vetores associativos)
O tipo numeric aceita diferentes tipos de representação numérica, como por exemplo:
1 1.0 3.2345 323.45e-2 0.32345E1 0xff 0x56
O tipo sting aceita caracteres especiais assim como na linguagem C e estão listados abaixo:
‘\a‘ (campainha),
‘\b‘ (backspace),
‘\f‘ (alimentação de formulário),
‘\n‘ (quebra de linha),
‘\r‘ (retorno de carro),
‘\t‘ (tabulação horizontal),
‘\v‘ (tabulação vertical),
‘\\‘ (barra invertida),
‘\”‘ (citação [aspa dupla])

‘\’‘ (apóstrofo [aspa simples]).
Variáveis do tipo userdata só pode ser utilizada em programas LUA-C, pois não podem ser criadas em programas feito somente em LUA.
O tipo threads não são os processos do sistema operacional, mas sim usado para implementar co-rotinas.

Requisitos do compilador:

a) Pelo menos dois tipos de dados (inteiro e ponto flutuante). Tipo string; opcional e vale ponto extra;
b) Arrays;
c) expressões aritméticas (-, +, *, /, MOD), lógicas (AND, OR e NOT) e relacionais (Maior que, menor que, maior ou igual, menor ou igual, igual e diferente)
d) precedência de operadores (verificar regras da linguagem) / utilização de parênteses para "alterar" a precedência;
e) 1 comando condicional (if-else, switch ou equivalente);
f) 1 comando de repetição (for, while, do ... while, repeat ou equivalente);
g) definição e chamada de métodos ou funções;
h) a passagem de parâmetros pode ser por valor ou por referência (escolham um);
i) aninhamento entre os blocos;
j) contexto das variáveis (global, local e bloco);
k) algum comando para exibir o valor das variáveis (print ou equivalente)
l) algum comando para receber o valor das variáveis (scan ou equivalente)
m) verificações de tipo de declaração de variáveis e tipos de expressão
A saída do compilador deve ser um programa na linguagem mips, que será executado no simulador spim;

O compilador pode ser dividido em alguns módulos para efetuar o processo de tradução. A lista abaixo foi proposta por Louden (2004):
• Analisador Léxico;
• Analisador Sintático;
• Analisador Semântico;
• Gerador de Código.

O Analisador Léxico é o responsável por agrupar os caracteres, contidos no arquivo do programa-fonte, em unidades significativas, chamadas tokens, e encaminhá las para o Analisador Sintático.
Por sua vez, o Analisador Sintático verifica se o uxo de tokens recebidos pelo Analisador Léxico é válido para a gramática (ou linguagem) que foi definida. Usualmente o Analisador Sintático produz uma estrutura (tipicamente uma do tipo árvore) que representa o programa fonte (AHO et al., 2008).

O Analisador Semântico recebe a estrutura produzida pelo Analisador Sintático e, principalmente, verificar se as operações são coerentes para os tipos de dados utilizados e faz a inferência dos tipos de dados. Ao nal do processo temos uma Árvore Anotada. As notas (informações de inferência, por exemplo) podem ser incluídas diretamente na estrutura recebida do Analisador Sintático, ou numa estrutura auxiliar como uma Tabela de Símbolos.

Estando a Árvore Anotada disponível para o Gerador de Código, este executa a tradução das estruturas recebidas para o programa-objeto. Nessa fase da compilação podem ser incluídas otimizações para que o programa traduzido execute de forma mais eficiente.

Podem haver outras fases intermediárias durante o processo, como, por exemplo, as fases de Otimização de Código, dependentes ou não da máquina-alvo.

GERADORES DE ANALISADORES SINTÁTICOS – YACC / BISON

O YACC/BISON serve para gerar automaticamente programas para análise sintática (usualmente em “C”) de códigos fonte de uma linguagem específica qualquer. Estas ferramentas possuem como entrada uma descrição de uma gramática, que especifica uma determinada linguagem, e gera como saída um programa em C ou C++ que será o parser desta linguagem. Uma vez compilado este parser, gerado automaticamente pelo YACC/BISON, ele terá como entrada arquivos com códigos fonte da linguagem especificada pela gramática, executando assim a “validação” códigos fonte através da execução deste parser. O parser gerado pelo YACC/BISON permite realizar apenas a validação do código fonte, indicando se está correto sintaticamente ou não, mas também permite implementar de modo bastante simples e direto ferramentas como interpretadores (executando diretamente ações a medida que vai reconhecendo comandos corretos na linguagem especificada) ou tradutores (realizando uma tradução dirigida pela sintaxe) e pode também servir de base para a criação de compiladores mais completos e sofisticados. Note que os tokens gerados pelos programas criados pelo LEX/FLEX são usualmente fornecidos como entrada para o processo seguinte de análise sintática realizado pelo programa criado pelo YACC/BISON, ou seja, ambos atuam de forma integrada.

Requisitos do compilador:

a) Pelo menos dois tipos de dados (inteiro e ponto flutuante). Tipo string; opcional e vale ponto extra;

b) Arrays;
c) expressões aritméticas (-, +, *, /, MOD), lógicas (AND, OR e NOT) e relacionais (Maior que, menor que, maior ou igual, menor ou igual, igual e diferente)
d) precedência de operadores (verificar regras da linguagem) / utilização de parênteses para "alterar" a precedência;
e) 1 comando condicional (if-else, switch ou equivalente);
f) 1 comando de repetição (for, while, do ... while, repeat ou equivalente);
g) definição e chamada de métodos ou funções;
h) a passagem de parâmetros pode ser por valor ou por referência (escolham um);
i) aninhamento entre os blocos;
j) contexto das variáveis (global, local e bloco);
k) algum comando para exibir o valor das variáveis (print ou equivalente)
l) algum comando para receber o valor das variáveis (scan ou equivalente)
m) verificações de tipo de declaração de variáveis e tipos de expressão
A saída do compilador deve ser um programa na linguagem mips, que será executado no simulador spim;