CompilerExpression

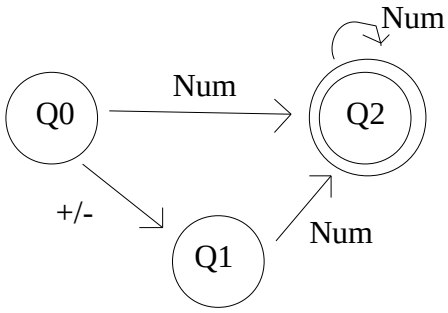
Autor: Luis Henrique Matos Sales

Boa Vista
2020

Este compilador aceita comandos aritméticos das quatro operações básicas, com variáveis do tipo float e int. O uso de ‘(‘ e ‘)’ é aceito pela gramática, o compilador não aceita variáveis maiores do que um caractere, quanto aos números a entrada de números como 12.1 é válida porém devido uma pequena limitação na análise sintática deste compilador somente o primeiro será exibido no código fonte gerado.

Gramáticas regulares

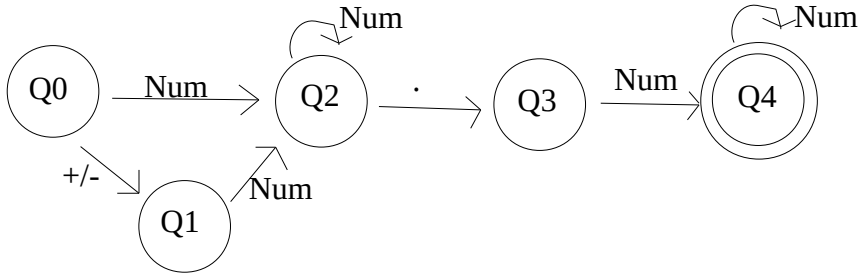
L = {w|w (+|-|num)(num)num*} essa gramática está presente no automato para reconhecer se um número é um inteiro.



	Q0	Q1	Q2
+	Q1	-	-
-	Q1	-	-
Num	Q2	Q2	Q2

Este automato reconhece como inteiros sequências que possuam ou não os sinais de ‘+’ ou ‘-’ na frente e que não possuam e que possuam apenas números.

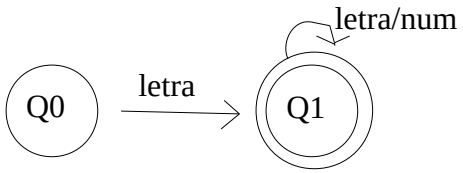
L = {w|w (+|-|num)(num)num*(.)(num)num*} gramática utilizada para reconhecer float.



	Q0	Q1	Q2	Q3	Q4
+	Q1	-	-	-	-
-	Q1	-	-	-	-
Num	Q2	Q2	Q2	Q4	Q4
.	-	-	Q3	-	-

Este automato reconhece como float sequências que possuam ou não os sinais de ‘+’ ou ‘-’ na frente e que possuam ‘.’ entre os caracteres formados apenas por números.

L = {w|w letra(letra|numero)*} gramática utilizada para reconhecer um identificador.



	Q0	Q1
Num	-	Q1
Letra	Q1	Q1

Este automato reconhece como identificador sequências que comecem com letras maiúsculas ou minusculas e que possam ser seguidas por números ou letras. Apesar do Compilador está configurado para rejeitar identificadores de mais de um caractere o automato para reconhecer identificadores é capaz de analisar sequencias maiores.

Gramáticas livres de contexto

Gramática usada no automato da analise sintática para reconhecer as operações suportadas pelo compilador. O tipo de automato escolhido foi de precedência fraca.

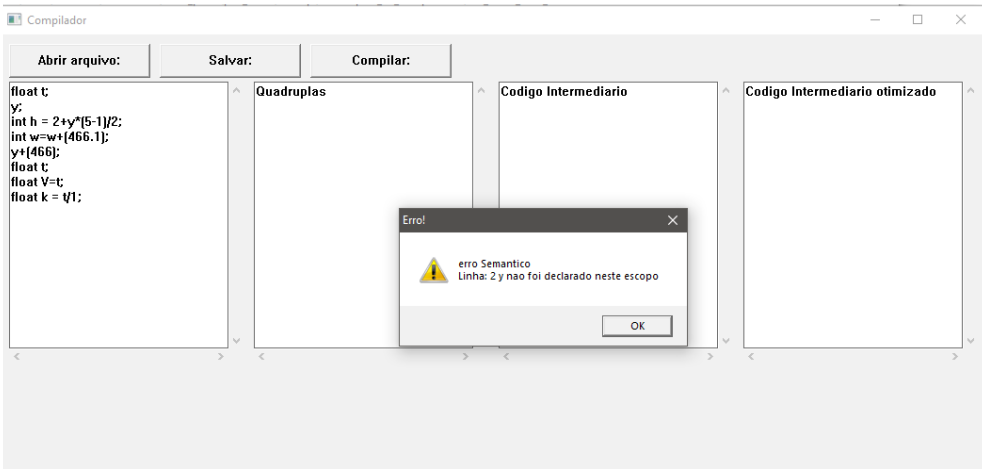
Símbolos não-terminais Vn = {E, S, M, D, P}
Símbolos terminais Vt = {+, -, *, /, (,), v}
Produções P={E → E+S, E → S, S → S-M, S → M, M → M*D, M → D, D → D/P, D → P, P → (E),
P → v}

Tabela de deslocamento e redução

	+	-	*	/	(	)	v	$
E	D					D		
S	R	D				R		R
M	R	R	D			R		R
D	R	R	R	D		R		R
P	R	R	R	R		R		R
+					D		D	
-					D		D	
*					D		D	
/					D		D	
(					D		D	
)	R	R	R	R		R		R
v	R	R	R	R		R		R
$					D		D	

Semântica

Na análise semântica este compilador checa se todas as variáveis utilizadas foram de fato declaradas no escopo através de uma lista onde são guardadas todas as variáveis declaradas para então após a análise sintática se percorrer o código checando se todas as variáveis foram declaras caso contrário é retornado um anunciando que a variável não foi declarada na linha x.



Geração do código intermediário

Após passar pela análise léxica, sintática e semântica o compilador começará o processo de geração do código intermediário. O código é gerado a partir de quádruplas geradas cada vez que uma expressão analisada pelo automato da sintax for aceita.

Exemplo de uma quádrupla onde v simboliza número
ou identificador

-	v	v	_t1
/	_t1	v	_t2
*	v	_t2	_t3
+	v	_t3	_t4
=	_t4	null	var

Na geração do código foi escolhido para este compilador o código de três endereços. Onde a instrução não seja uma declaração ou atribuição é criado variáveis temporárias para que as operações possam ser executadas sempre utilizando no máximo três endereços.

Exemplo: h = 2+y*(5-1)/2;

_t1 := 5 - 1
_t2 := _t1 / 2
_t3 := y * _t2
_t4 := 2 + _t3
h := _t4

Otimização do código intermediário

Para otimiza o código intermediário foram utilizadas as regras abaixo.

Substituir	por
x+ 0	x
0 + x	x
x - 0	x
x * 1	x
1 * x	x
x / 1	x

Para isso a quádrupla é novamente percorrida buscando alguma operação como na tabela acima casa se encontre a operação é substituída pela atribuição da variável.

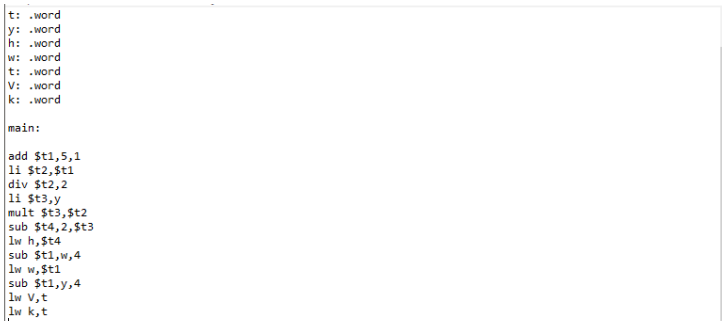
Exemplo:

Sem otimização	Com otimização
_t1 := t / 1 k := _t1	k := t

Geração do código em linguagem simbólica

A linguagem simbólica escolhida para a geração do código intermediário foi o Mips que possui uma arquitetura de 32 bits. Para Gerar o código para Mips primeiro é escrito as variáveis no arquivo de saída e então com a quádrupla já otimizada é escrito as operações existentes da seguinte forma caso seja uma adição é primeiramente escrito add caso seja subtração é escrito sub logo após é escrito variável temporária que guardara o valor da operação e os outros dois elementos da instrução. Caso seja multiplicação ou divisão a variável temporária será carregada com o primeiro argumento e em seguida é será escrito mult ou div multiplicação e divisão respectivamente junto com a variável temporária com o valor carregado anteriormente para que a operação seja executada com o segundo argumento. No caso de atribuição é feita o carregamento(load) de uma variável x em uma y.

Exemplo:

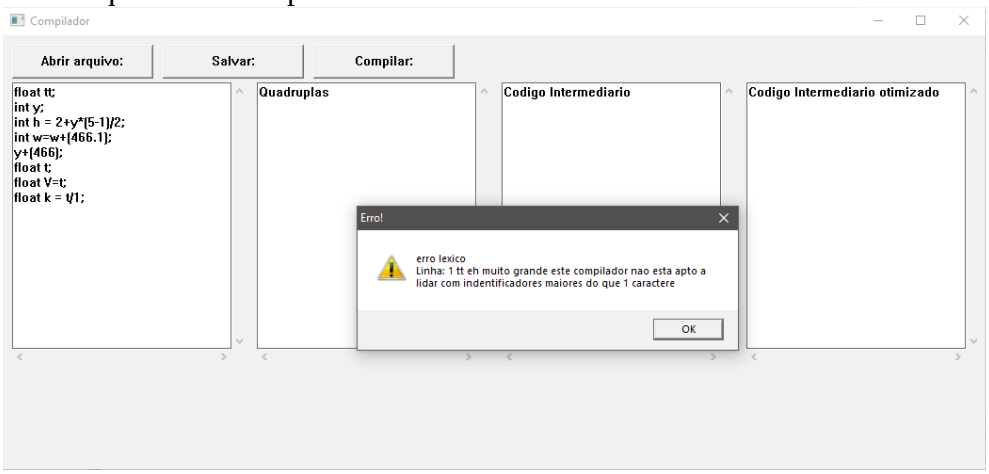


```
t: .word
y: .word
h: .word
w: .word
t: .word
V: .word
k: .word

main:

add $t1,5,1
li $t2,$t1
div $t2,2
li $t3,y
mult $t3,$t2
sub $t4,2,$t3
lw h,$t4
sub $t1,w,4
lw w,$t1
sub $t1,y,4
lw V,t
lw k,t
```

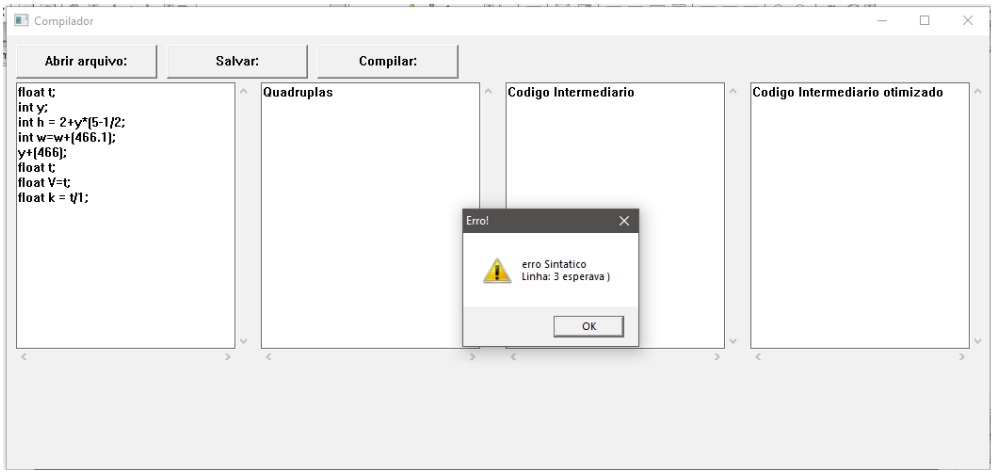
Mensagens corretivas

Algumas mensagens corretivas serão apresentadas mostrando o tipo do erro léxico, semântico e sintático.

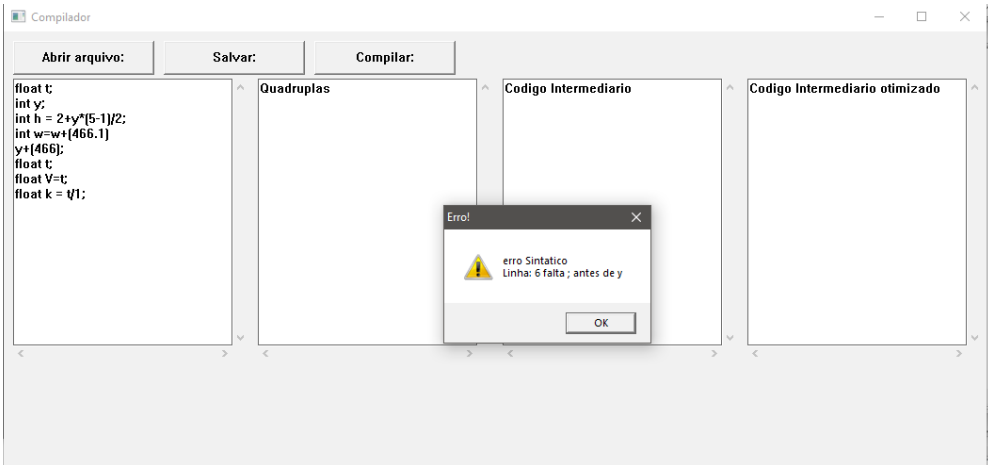
Erro léxico em que a variável possui mais de um caractere



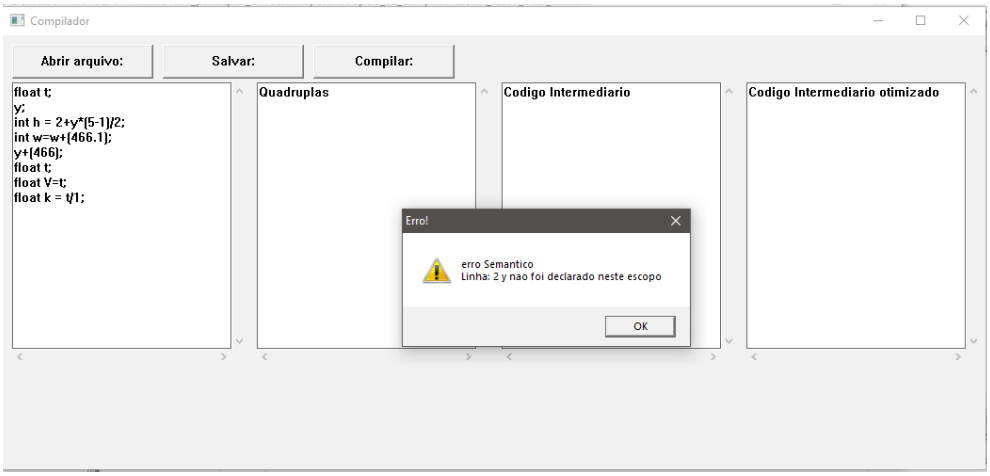
Erro sintático devido a falta de um dos parênteses.



Erro sintático devido a falta de ;.



Erro semântico por conta de variável não declarada.



OBS: Para compilar o código deste compilador é necessário compilar junto da biblioteca libcomdlg32.a caso se esteja no windows e utilizando a ide codeblocks basta clicar em settings, compiler, Linker settings, add , o caminho da biblioteca normalmente é C:\Program Files (x86)\ CodeBlocks\MinGW\lib\libcomdlg32.a

Este compilador foi desenvolvido como trabalho final da disciplina construção de compiladoresDCC605 com o professor Luciano Ferreira.