INF1022 - Trabalho G2

Professor: Vitor Pinheiro	Grupo: Gabriel da Silva Marques Ferreira | 2210442 Rafael Chaves Bayão Ribeiro | 2210461

1. Regras da Gramática:

1. *programa → INICIO varlist monitor_varlist EXECUTE cmds TERMINO*
2. *monitor_varlist → MONITOR varlist*
3. *varlist → ID COMMA varlist | ID*
4. *cmds → cmd cmds | cmd*
5. *cmd → while | if | ifelse | plus | times | eval | zero | equal*
6. *while → ENQUANTO ID FACA cmds FIM*
7. *if → IF ID THEN cmd*
8. *ifelse → IF ID THEN cmd ELSE cmd*
9. *eval → EVAL LPAREN ID COMMA ID COMMA cmds RPAREN*
10. *plus → ID PLUS exp | NUM PLUS exp*
11. *times → ID TIMES exp | NUM TIMES exp*
12. *exp → ID | NUM | plus | times*
13. *zero → ZERO LPAREN ID RPAREN*
14. *equal → ID EQUAL exp*

2. Explicações

Algumas Observações:

- Foi utilizada a biblioteca do python ply (Python Lex Yacc) para construção do código.
- Decidimos que o arquivo gerado pelo nosso código será na linguagem C.
- Funções e variáveis iniciadas por “t_” definem tokens da gramática enquanto funções iniciadas por “p_” definem regras da gramática.
- Implementamos todo o código, tanto o analisador léxico quanto o sintático, no mesmo arquivo main.py.
- Para facilitar a nossa cooperação, utilizamos o Replit como nosso ambiente de codificação, já que ele possibilita que nós dois trabalhemos no código simultaneamente.
- Para executarmos o código, criamos um arquivo “ex.provol” no qual inserimos os nossos testes e ao executar o código é criado um arquivo “saida.c” com o código em C gerado pelo transpilador.

- O não terminal “monitor_varlist” foi criado para otimizar a implementação das nossas funções de definição das regras de gramática, visto que, como utilizamos um vetor global que monitora as variáveis que devem ser exibidas (aquelas que vem depois de MONITOR), é razoável que façamos o tratamento dessas variáveis em uma outra regra que não seja a do não terminal “varlist”.
- Na definição de tokens, criamos um dicionário com as palavras reservadas da nossa gramática (tokens terminais que servem apenas como “marcação”) e depois inserimos os valores na lista de todos os tokens, além de criar a função t_reservados para definir justamente esses tokens reservados.
- Se algum caractere não válido for inserido, uma mensagem de erro aparecerá para o usuário.
- Caso haja algum erro de sintaxe durante a operação, isso será sinalizado como erro para o usuário também. Isso causa, também, com que uma mensagem de que o arquivo .c não foi criado apareça.

3. Restrições na Implementação

- Qualquer if gerado consegue ler apenas se a variável é igual a zero; não é possível realizar operações como if(x == num) ou if(x > num).
- Para otimizar a nossa implementação, não utilizamos tabeamento dentro do código .c.
- Ao ler o terminal EVAL, que representa o for no código .c, só conseguimos realizar iteração decrescente (como pedido no enunciado), visto que a iteração crescente não foi implementada no nosso programa.
- Não conseguimos fazer a saída do programa .c ter \n, visto que quando colocamos o \n dentro do printf descrito nas funções que definem a nossa gramática, esse \n era lido como uma nova linha no código .c em si, o que fazia o print quebrar e não funcionar.
- As nossas operações aritméticas não foram implementadas com o uso de parênteses, o que pode causar resultados diferentes dos esperados ao querer realizar tais operações.

4. Casos de Teste

TESTE 1

Arquivo provolOne	Arquivo C Gerado	Output do Arquivo C

INICIO X, Y
MONITOR Z
EXECUTE
Y = 2
X = 5
Z = Y
ENQUANTO X FACA
Z = Z + 1
FIM
TERMINO
	#include <stdio.h>
	
	int main(){
	int X = 0;
	int Y = 0;
	int Z = 0;
	printf("Z = %d",Z);
	Y = 2;
	
	X = 5;
	
	Z = Y;
	printf("Z = %d",Z);
	while (X)
	{
	Z = Z + 1;
	printf("Z = %d",Z);
	}
	
	return 0;
	}
		o código entra num loop infinito tendo então uma saída infinita

TESTE 2

Arquivo provolOne	Arquivo C Gerado	Output do Arquivo C

INICIO X, Y, B
MONITOR Z
EXECUTE
X = 5
Y = 2
B = 0
EVAL(X, Y,
IF B THEN Z = Z + 2
ELSE Z = Z + 1)
TERMINO
	#include <stdio.h>
	
	int main(){
	int X = 0;
	int Y = 0;
	int B = 0;
	int Z = 0;
	printf("Z = %d",Z);
	X = 5;
	
	Y = 2;
	
	B = 0;
	
	for (int i = X; i > Y; i--)
	{
	if (B)
	{
	Z = Z + 2;
	printf("Z = %d",Z);
	}
	else
	{
	Z = Z + 1;
	printf("Z = %d",Z);
	}
	}
	
	return 0;
	}
		Z = 0Z = 1Z = 2Z = 3

TESTE 3

Arquivo provolOne	Arquivo C Gerado	Output do Arquivo C

INICIO X, Y, L
MONITOR Z
EXECUTE
ZERO(Z)
X = 5
Y = 0
L = 1
IF Y THEN
Z = EVAL (X, L, Z=Z*L)
ELSE
Z = EVAL(X, L, Z=Z*X)
FIM
TERMINO
	#include <stdio.h>
	int main(){
	int X = 0;
	int Y = 0;
	int L = 0;
	int Z = 0;
	printf("Z = %d",Z);
	Z = 0;
	printf("Z = %d",Z);
	X = 5;
	
	Y = 0;
	
	L = 1;
	
	if (Y)
	{
	for (int i = X; i > L; i--)
	{
	Z = Z * L;
	printf("Z = %d",Z);}
	
	}
	else
	{
	for (int i = X; i > L; i--)
	{
	Z = Z * X;
	printf("Z = %d",Z);}
	
	}
	
	return 0;
	}
		Z = 0Z = 0Z = 0Z = 0Z = 0Z = 0