1. Gramática

A gramática que descreve a linguagem criada é descrita por:
G = (V, Σ, P, S)

V = {Programa_principal, Comandos, Comando, Declaracao, Tipo, Decl, LISTA_VAR, Atribui_valor, Atribuicao, Exp, x, SINAL, FUNCAOWHILE, FUNCAOIF, IFARG, IFARG2}

Σ = {NUM, SIGN, DSIGN, VAR, INV, INT, FLOAT, DOUBLE, ATRIB, SC, COMP, EOU, COMM, ABREPAR, FECHAPAR, ABRECOL, FECHACOL, ABRECHAVE, FECHACHAVE, MAIN, WHILE, IF, ELSE}

P = {

Programa_principal → MAIN ABREPAR FECHAPAR ABRECHAVE Comandos FECHACHAVE

Comandos → Comando Comandos | Comando | ϵ

Comando → Declaracao | Atribuicao | FUNCAOWHILE | FUNCAOIF

Declaracao → Tipo Decl SC

Tipo → INT | FLOAT | DOUBLE

Decl → LISTA_VAR

LISTA_VAR → VAR Atribui_valor COMM LISTA_VAR | VAR Atribui_valor

Atribui_valor → ATRIB SINAL NUM | ϵ

Atribuicao → VAR ATRIB SINAL Exp SC

Exp → NUM x | VAR x

x → SINAL Exp | ϵ

SINAL → SIGN | ϵ

FUNCAOWHILE → WHILE ABREPAR IFARG FECHAPAR ABRECHAVE Comandos FECHACHAVE

FUNCAOIF → IF ABREPAR IFARG FECHAPAR ABRECHAVE Comandos FECHACHAVE

IFARG → VAR COMP VAR | VAR COMP VAR IFARG2 | VAR COMP NUM | VAR COMP NUM IFARG2| VAR

IFARG2 → EOU IFARG

}

S = {Programa_principal}

2. Diferenças em relação a primeira entrega

2.1 Inclusão de Nova Biblioteca

No código mais recente, foi incluída a biblioteca “parser.tab.h”, que será gerada automaticamente ao se executar as instruções dadas no manual de uso.

2.2 Descarte da Função Principal

No código mais recente, a função “main” foi deixada comentada, pois agora será usada a função “main” inclusa no analisador sintático (arquivo parser.y) e, portanto, a função principal do analisador léxico não será mais necessária.

2.3 Tokens Adicionados

No código mais recente, foram adicionados novos tokens para representar operadores matemáticos duplos (DSIGN) e operadores lógicos (EOU). Além disso, foram adicionados tokens separados para cada tipo de parênteses, colchetes e chaves (ABREPAR, FECHAPAR, ABRECOL, FECHACOL, ABRECHAVE, FECHACHAVE), enquanto no código anterior todos eles eram representados por um único token (PAR). Também foi adicionado um novo token para representar a função “while” (WHILE) a ser implementada na sintaxe, “else” (ELSE), que não foi utilizado, e “double” (DOUBLE), e removido o token “for” (FOR).

2.4 Alterações nas Regras Léxicas

No código mais recente, foram adicionadas regras léxicas para reconhecer as palavras-chave while, main, if, else, int, float e double. No código anterior, apenas as palavras-chave for, int, main, float e if eram reconhecidas. Além disso, foi adicionada a regra léxica para reconhecer a palavra “\n”, que é uma quebra de linha, e, ao ser identificada, incrementa o número de linhas lidas até então. Foi adicionado a todas as regras léxicas o “return” seguido do tipo de símbolo terminal que foi lido pela regra, para assim ser possível entregar (retornar) o que foi lido no analisador léxico para o analisador sintático.

2.5 Contagem de Linhas e Erros

O código mais recente inclui a contagem de linhas e erros, o que pode ser útil para depuração e relatórios de erros. A contagem de linhas facilita a localização dos erros, e a contagem de erros facilita depurar se tudo ocorreu como esperado no final ou não.

3. Manual de uso

Com os arquivos em seu computador, abra o WSL e realize a instalação do bison da seguinte maneira:

sudo apt update && sudo apt install flex bison

Com o WSL aberto no diretório em que os arquivos estão, execute os seguintes comandos:

bison -d parser.y
flex trabalho.lex
gcc -o compilador.flex.o -c lex.yy.c
gcc -o compilador.y.o -c parser.tab.c
gcc -o compilador compilador.flex.o compilador.y.o -lfl

Por fim, teste o arquivo de entrada dado “teste.txt” com o compilador criado da seguinte forma:

./compilador teste.txt

Caso tudo ocorra como esperado será exibido “Análise concluída com sucesso” ao final das operações.