Trabalho de Linguagens Formais e Autômatos

Equipe:

Henrique Sartori Siqueira	19240472
Jemis Dievas José Manhiça	19076272
Rafael Silva Barbon	19243633

Código GitHub:
https://github.com/RafaelBarbon/LFA---Gram-tica-Identificadora-de-Linguagem

Descrição da Gramática G

G = ({B,C,D,E,F,G,S},{main,int,scanf,printf,a,b,c,...,z,0,1,2,...,9,*,-,+,/,=,;,),(,{,},,},P,S)
P:

S -> main(){B}
B -> D=C; | int E; | scanf(D); | printf(D); | BB | ε
C -> F | D | (C) | CGC;
D -> aH | bH | ... | zH
H -> D | ε
E -> DJ
J -> ,E | ε
F -> 0I | 1I | ... | 9I
I -> F | ε
G -> + | - | / | *

Figura 1: Gramática criada.

Inicialmente, os símbolos terminais que formam a palavra “main(){B}” é identificada, em seguida o símbolo não terminal B é utilizado para identificar o corpo do programa, realizando também um balanceamento das chaves.

O corpo do programa pode conter quatro diferentes tipos de comandos identificados pelo símbolo não terminal B, sendo eles a declaração de variáveis, atribuição de valor a uma variável por uma expressão aritmética, impressão do valor da variável e a coleta do valor de uma variável.

Os símbolos não terminais D e H são responsáveis pela identificação de uma cadeia de caracteres, que, em outras palavras, identifica nomes de variáveis.

O não terminal C estabelece o valor a ser recebido por uma expressão aritmética composta de variáveis, operadores aritméticos, parênteses (balanceados), e ou números.

O carácter não terminal E realiza a função de estabelecer as declarações das variáveis dentro do corpo do programa main, sendo tais variáveis separadas por vírgula, conforme identificado no não terminal J, que possibilita novas variáveis separadas por vírgula.

Os símbolos não terminais F e I realizam a mesma função dos não terminais D e H, porém para identificação de cadeias de números.

O não terminal G está presente na expressão aritmética desenvolvida por C sendo que sua única função é a de substituir G, não terminal, por uma das operações aritméticas consideradas como terminais.

Descrição do programa reconhecedor e como utilizá-lo

Para a construção do programa reconhecedor, foram desenvolvidas duas bibliotecas para implementação das estruturas e funções de lista e pilha.

A função "monta_simbolo", utilizada para auxiliar o programa reconhecedor, retorna uma lista com os símbolos do programa. A fim de facilitar a tarefa do reconhecedor, os nomes de variáveis retornam um símbolo denominado "id" (desconsiderando as palavras reservadas), números o símbolo "num" e as operações aritméticas (-, +, *, /) retornam o símbolo "op".

A função reconhecedora (com o nome de valida no nosso código) recebe como parâmetro a lista de símbolos gerados pela função descrita no parágrafo anterior. Essa função verifica a lista de palavras de acordo com a gramática, inicialmente a função verifica a existência da main (“main ( ) { “), e logo em seguida é feita a verificação do corpo da main (comandos, declarações de variáveis, atribuição de valores à variáveis), caso um símbolo não mapeado seja identificado, a função já recusa o programa (retornando *false* de imediato), quando a função identifica um “int” ela sai a procura de um “id” e depois de um “;” caso o “;” seja encontrado depois de um “id” então a função já retorna “f*alse*” (como recusa do programa), e quando a função identifica o “printf” ou um “scanf” ela usa a mesma lógica, saí procurando o “(“ (quando o “(“ não for encontrado depois do “scanf ou do “printf” a função já retorna “*false”)* e depois um “id” (se a função não identificar um “id” depois de um “(“ ela já retorna *false*) seguido de um “)” (caso um “)” não for encontrado depois de um “id” a função retorna um *false*) e depois de um “;”(e por fim, se um “;” não for encontrado depois do “)” a função retorna *false*, mostrando que o comando é inválido) para determinar o fim do comando. Para identificar as expressões, a função começa identificando o “id”, e de segunda procura um “=” (caso o “=” não seja encontrado depois do “id” a função já retorna false, invalidando o programa). Também foi utilizada uma flag (“flagOp”) para garantir a intercalação entre num/id e operador, desconsiderando parênteses, assim como a validação dos parênteses no quesito de abertura e fechamento (exemplo: “id = num (op id);” estaria errado assim como “id = num op(id op)num;”), e conforme tal intercalação, o

fechamento da expressão aritmética com um operador (“id=id op;”) não é válido, este caso também é verificado após encontrar o símbolo de final de expressão (“;”) realizando tal conferência assim como a conferência da pilha estar vazia para validar a expressão aritmética, e no final, a função procura conferir se a main tem a chave de fechamento da mesma (“}”, caso a mesma não seja encontrado, a função retorna false, informando que o programa é inválido).

Para a execução, existem três arquivos “exec” .bat para execução no sistema operacional Windows, .sh para sistemas Linux e .command para MAC. Para utilizar estes comandos de compilação é necessário ou executar como administrador (Windows) ou atribuir permissão de execução utilizando “chmod +x exec(.sh ou .command)”. Estes programas possuem comandos necessários para compilação. Em seguida, para realizar a execução basta chamar o arquivo identificador gerado passando como parâmetro o arquivo .txt com o programa a ser analisado.

Exemplo de execução em MAC:

```
user@user-comp % chmod +x exec.command
user@user-comp % ./exec.command
user@user-comp % ./identificador test.txt
Aceito
```

Exemplo de execução em Linux (Ubuntu):

```
user@user-comp % chmod +x exec.sh
user@user-comp % ./exec.sh
user@user-comp % ./identificador test.txt
Aceito
```

Exemplo de execução em Windows:

```
> .\exec.bat
>gcc -c pilha.c -o pilha.o   && gcc -c lista.c -o lista.o   && gcc -c main.c -o main.o
>gcc lista.o pilha.o main.o -o identificador
> .\identificador test.txt
Aceito
```

Código fonte documentado

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include "lista.h"
```

```c
#include "pilha.h"

// Função responsável por ler todo o arquivo e retornar uma lista com os símbolos encontrados
Lista* monta_simbolo(FILE *arq);
// Função que verifica a lista de palavras de acordo com a gramática
bool valida(Lista *simbolos);

int main(int argc, char **argv){
   if(argc < 1){ // Caso não haja o nome de arquivo como parâmetro
       printf("Erro nome de arquivo inexistente.");
       exit(0);
   }
   char c;
   FILE *arq = fopen(argv[1],"r");
   Lista *palavras = Cria();

   if(!arq){ // Caso não consiga abrir o arquivo por conta do nome incorreto
       printf("Erro ao abrir o arquivo.");
       exit(0);
   }
    palavras = monta_simbolo(arq); // Lista de símbolos (referente a gramática) encontrados no arquivo
   //Imprime(palavras); // Para testes
   //printf("\n");
   printf("%s\n", valida(palavras) ? "Aceito" : "Recusado");
   fclose(arq);
   Libera(&palavras);
   return 0;
}

// Função que verifica a lista de palavras de acordo com a gramática
bool valida(Lista *simbolos){
   Lista *simbolo = simbolos;
   Pilha *parenteses = CriaP();
   bool flagOp = false; // Utilizada para intercalação entre num|id e op, abertura de parênteses e detecção de fechamento com op
   // Conferência do main(){
   if(simbolo && !(strcmp(simbolo->string,"main"))){
       simbolo = simbolo->prox;
       if(simbolo && !(strcmp(simbolo->string, "("))){
           simbolo = simbolo->prox;
```

```
            if(simbolo && !(strcmp(simbolo->string, ")"))){
                simbolo = simbolo->prox;
                if(simbolo && !(strcmp(simbolo->string, "{"))){
                    simbolo = simbolo->prox;
                }else{
                    LiberaP(&parenteses);
                    return false;
                }
            }else{
                LiberaP(&parenteses);
                return false;
            }
        }else{
            LiberaP(&parenteses);
            return false;
        }
    }else{
        LiberaP(&parenteses);
        return false;
    }
    //Conferência do corpo do programa (4 possibilidades de comandos - declaração de variáveis, coleta de valor de varíavel, exibição de valor de variável e atribuição de valor para variável)
    while(simbolo && strcmp(simbolo->string, "}") != 0){
          if(!strcmp(simbolo->string,"?")){ // Caso seja um símbolo não mapeado, já recusa o programa
            LiberaP(&parenteses);
            return false;
        }else if(!strcmp(simbolo->string,"int")){ // Caso declaração de variáveis
            simbolo = simbolo->prox;
                                while(simbolo   &&   !strcmp(simbolo->string,"id")   && (strcmp(simbolo->string,";")  != 0)){ // Confere o próximo identificador ou o final da declaração
                simbolo = simbolo->prox;
                  if(simbolo  && !strcmp(simbolo->string,",")){ // Confere se há uma próxima variável a partir da vírgula
                    simbolo = simbolo->prox;
                         if(simbolo  && !strcmp(simbolo->string,";")){ // Verifica se termina sem o identificador (id,;)
                        LiberaP(&parenteses);
                        return false;
                    }
```

```
                }
            }
                                }else    if(!strcmp(simbolo->string,"printf")    ||
!strcmp(simbolo->string,"scanf")){  // Caso seja impressão ou coleta de valor de
variável
            simbolo = simbolo->prox;
            if(simbolo && !strcmp(simbolo->string,"(")){
                simbolo = simbolo->prox;
                if(simbolo && !strcmp(simbolo->string,"id")){
                    simbolo = simbolo->prox;
                    if(simbolo && !strcmp(simbolo->string,")")){
                        simbolo = simbolo->prox;
                        if(simbolo && !strcmp(simbolo->string,";")){
                            simbolo = simbolo->prox;
                            continue;
                        }else{
                            LiberaP(&parenteses);
                            return false;
                        }
                    }else{
                        LiberaP(&parenteses);
                        return false;
                    }
                }else{
                    LiberaP(&parenteses);
                    return false;
                }
            }else{
                LiberaP(&parenteses);
                return false;
            }
             }else if(simbolo  &&  !strcmp(simbolo->string,"id")){  // Caso seja uma
expressão (,),+,-,*,/,id,num
            simbolo = simbolo->prox;
            if(simbolo && !strcmp(simbolo->string,"=")){
                simbolo = simbolo->prox;
                  flagOp = false; // Flag para contar com que o próximo símbolo seja
um operador (desconsiderando parênteses)
                while(simbolo && strcmp(simbolo->string,";")){//Detecta expressão
                      if(simbolo && (!strcmp(simbolo->string,"(")) && (!flagOp)){ //
Verifica se não é do tipo "num ( op"
                        Push(parenteses, '('); //joga na pilha
```

```
                }else if(simbolo && !strcmp(simbolo->string,")") && flagOp){
                        if(Pop(parenteses) == -1){ // Verifica se foi possível
realizar o pop (caso nao a pilha esta vazia)
                        LiberaP(&parenteses);
                        return false;
                    }
                                                        }else   if(simbolo   &&
(!strcmp(simbolo->string,"num")||!strcmp(simbolo->string,"id")) && !flagOp){
                         // Espera-se um operador aritmético no próximo símbolo
                    flagOp = true;
                }else if(simbolo && (!strcmp(simbolo->string,"op")) && flagOp){
                    // Já foi identificado um operador aritmético neste símbolo
                    flagOp = false;
                    }else{ // Caso seja um símbolo inválido para a expressão ou
símbolo
                    LiberaP(&parenteses);
                    return false;
                }
                simbolo = simbolo->prox;
            }
              if(!VaziaP(parenteses) || !flagOp){ // Verifica o balanceamento dos
parênteses e o fechamento com op
                LiberaP(&parenteses);
                return false;
            }
        }else{ // Se não houver "=" como segundo símbolo do comando
            LiberaP(&parenteses);
            return false;
        }
      }else{ // Caso o comando inicie com um símbolo terminal, porém não válido
para iniciar o comando
        LiberaP(&parenteses);
        return false;
    }
    simbolo = simbolo->prox;
}
if(simbolo && !strcmp(simbolo->string,"}")){
    LiberaP(&parenteses);
    return true;
}
LiberaP(&parenteses);
return false;
```

```
}

// Função responsável por ler todo o arquivo e retornar uma lista com os símbolos
encontrados
Lista* monta_simbolo(FILE *arq){
   char *palavra = NULL,c;
   int i = 0;
   bool flag = false, flagTI = false;
   Lista *l = Cria();
   c = getc(arq);
   while(c != EOF){
       //são letras -> cadeia de letras
       while(c >= 97 && c <= 122){
           palavra = realloc(palavra,i+1); // Realoca espaço para o novo caractere
              if(!palavra){ // Caso não haja memória disponível retornará NULL no
realloc
               printf("Erro: armazenamento insuficiente para alocar");
               exit(1);
           }

                palavra[i++] = c; // Adiciona o novo caractere na palavra a ser
identificada

           c = getc(arq); // Lê o próximo caractere do arquivo
            flag = true; // Booleano que identifica se é uma palavra que entrou para
colocar na lista posteriormente
       }
       if(flag){
                 palavra[i] ^= palavra[i]; // Atribuir \0 no final da cadeia de
caracteres
           i ^= i; // Reinicializa o index
           // Verifica se a palavra formada é uma das reservadas
            if((strcmp(palavra,"printf") == 0) || (strcmp(palavra,"scanf") == 0) ||
(strcmp(palavra,"int") == 0) || (strcmp(palavra,"main") == 0)){
               Insere(&l,palavra);//insere na lista a palavra reservada
               //printf("Inseriu: %s\t",palavra);
           }
           else
               Insere(&l,"id");//insere na lista a variável
           flag = false;
            flagTI = true; // Booleano que identifica se é uma palavra que entrou
para colocar na lista posteriormente
```

```
        }
          //cadeia de números (Apenas verifica se tem um conjunto de números ou um
apenas um número (Nao importa qual é o numero, ou seja não precisa salvar os
números))
        while(c >= 48 && c <= 57){
            c = getc(arq);
            flag = true;
        }
        if(flag){
            Insere(&l,"num");//insere na lista o número
            flag = false;
              flagTI = true; // Booleano que identifica se é uma palavra que entrou
para colocar na lista posteriormente
        }
         if((c == '(') || (c == ')') ||(c == '{') ||( c == '}') || (c == ';') || (c
== ',') || (c == '=' )){ // Detecta símbolo
            palavra = realloc(palavra,2);
            if(!palavra){
                printf("Erro: armazenamento insuficiente para alocar");
                exit(1);
            }
            palavra[0] = c;
            palavra[1] = 0;
            Insere(&l,palavra);//insere na lista o símbolo detectado
            c = getc(arq);
              flagTI = true; // Booleano que identifica se é uma palavra que entrou
para colocar na lista posteriormente
        }
           if((c == '/') || (c == '+') ||(c == '-')|| (c == '*')){ // Detecta um
operador aritmético
            Insere(&l,"op");//insere na lista o operador aritmético
            c = getc(arq);
              flagTI = true; // Booleano que identifica se é uma palavra que entrou
para colocar na lista posteriormente
        }
         if(c == ' ' || c == '\n' || c == '\t'){// Caso haja espaço vazio, ignora o
caractere e coleta o próximo
            c = getc(arq);
              flagTI = true; // Booleano que identifica se é uma palavra que entrou
para colocar na lista posteriormente
        }
        if(!flagTI){ // Caso seja um caractere desconhecido
```

```
            Insere(&l,"?");//insere na lista
            c = getc(arq);
        }
        flagTI = false;
    }
    free(palavra);
    return l;
}
```

Biblioteca da PILHA:

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "pilha.h"

// Criar pilha
Pilha* CriaP(){
    Pilha *p = (Pilha*)malloc(sizeof(Pilha));
    p->no = NULL;
    return p;
}

// Verificação se está vazia
bool VaziaP(Pilha *p){
    if(p->no == NULL)
        return true;
    return false;
}

// Push para a pilha
No* InsertP(No *n, char p){
    No *novo = (No*)malloc(sizeof(No));
    novo->parentese = p;
    novo->prox = n;
    return novo;
}

void Push(Pilha *p, char pa){
    p->no = InsertP(p->no,pa);
```

```
}

// Pop para a pilha
No* RemoveP(No *n){
   No *nn = n->prox;
   free(n);
   return nn;
}

// Retorna o elemento que foi retirado ou -1 caso a pilha esteja vazia
char Pop(Pilha *p){
   if(VaziaP(p)){
       printf("\nPilha vazia."); // Para debug
       return -1;
   }
   char v = p->no->parentese;
   p->no = RemoveP(p->no);
   return v;
}

// Impressão da pilha para debug
void ImprimeP(Pilha *p){
   if(VaziaP(p))
       printf("\nPilha Vazia.");
   else
       for(No *aux = p->no; aux != NULL; aux = aux->prox)
           printf("\n%c", aux->parentese);
}

// Desalocamento de memória da pilha
void LiberaP(Pilha **p){
   No *aux = ((*p)->no), *aux2;
   while(aux != NULL){
       aux2 = aux->prox;
       free(aux);
       aux = aux2;
   }
   free(*p);
   (*p) = NULL;
}
```

Biblioteca da LISTA:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "lista.h"

// Criar lista encadeada
Lista* Cria(){
   return NULL;
}

// Conferência de vazia
bool Vazia(Lista *l){
   if(l == NULL)
       return true;
   return false;
}

// Inserção de elementos no final da lista
void Insere(Lista **l, char *v){
   Lista *novo = malloc(sizeof(Lista)), *aux = *l;
   if(!novo){
       printf("Erro: armazenamento insuficiente para alocar");
       exit(1);
   }
   novo->string = malloc((strlen(v)+1)*sizeof(char)); // strlen(v)+1 considera-se o
\0 além dos caracteres
   if(!novo->string){
       printf("Erro: armazenamento insuficiente para alocar");
       exit(1);
   }
   strcpy(novo->string,v);
   if(!(*l)){ // Caso seja o primeiro elemento da lista
       novo->prox = NULL;
       *l = novo;
   }
   else{
       while(aux->prox != NULL){ // Avança o ponteiro até o último elemento da
lista
           aux = aux->prox;
       }
       novo->prox = aux->prox;
```

```
        aux->prox = novo; // Para adicionar o novo elemento à lista
    }
}

// Retirar elemento da lista a partir da posição fornecida
void Retira(Lista **l,int v){
    Lista *aux, *aux2;
    int i;
    for(i = 0, aux = (*l), aux2 = NULL; aux != NULL; i++, aux2 = aux, aux =
aux->prox){
        if(i == v){
            if(v == 0)
                (*l) = aux->prox;
            else
                aux2->prox = aux->prox;
            free(aux->string);
            free(aux);
            break;
        }
    }
}

// Busca se o dado existe na lista e retorna a posição deste
int Busca(Lista *l, char *v){
    Lista *aux;
    int i;
    for(i = 0, aux = l; aux != NULL; i++, aux = aux->prox)
        if(strcmp(aux->string,v) == 0)
            return i;
    return -1;
}

// Impressão da lista para testes
void Imprime(Lista *l){
    if(Vazia(l))
        printf("\nLista vazia.");
    else
        for(Lista *aux = l; aux != NULL; aux = aux->prox)
            printf("\n%s",aux->string);
}

// Desalocamento da lista
```

```c
void Libera(Lista **l){
   Lista *aux = (*l), *aux2;
   while(aux != NULL){ // Inicia-se do início e libera a memória de cada nó
        aux2 = aux->prox;
        free(aux->string);
        free(aux);
        aux = aux2;
   }
   (*l) = NULL;
}
```

Casos de testes

Foram produzidos dois casos de teste, o primeiro deles sem erros, ou seja, aceitado pelo programa reconhecedor:

```
main(){
   int james, rafael;
   int henrique;
   printf(dievas);
   scanf(barbon);
   nada = (9 * (6 + (5 / (barbon * 9))));
   james = nada;
   fael = math + (num -(num / (silva * (0 + one)))) + comp;
   freitas = 10;
   nota = 11;
}
```

>ACEITO

E o segundo com alguns erros, utilizando palavras chaves como nome de variáveis, atribuições incompletas, falta de “;”, e utilização incorreta das chaves:

```
main(){
   int sartori, int, amigomeu;
   int freitas, otimo;
   printf(dievas);
   scanf(barbon);
   james = nadamain(){
   int james, rafael;
   int henrique;
```

```
    printf(dievas);
    scanf(barbon);
    james = ;

    sartori = 6 + henrique * amigomeu;
    printf(lfa);
}
```

>REJEITADO