Relatório

Analisadores Léxicos e Sintáticos - INF1022

Leo Land Bairos Lomardo-2020201
Matheus Valejo Gomes Pereira - 2011536

Professor: Vitor Pinheiro de Almeida

Rio de Janeiro, RJ
Novembro, 2023
Período 2023.2

Conteúdo

1 Introdução

O projeto final da disciplica Analisadores Léxicos e Sintáticos(INF1022), consiste no desenvolvimento de um analisador sintático para linguagem *HoraDoShow-Script*. Nele, o analisador recebe como entrada um programa escrito na linguagem *HoraDoShow*, e retorna um programa escrito na linguagem escolhida pelo grupo, no nosso caso optamos pela saída ser na linguagem C.

Como informado no enunciado do trabalho, devemos utilizar um gerador de analisadores sintáticos que implemente o método *LaLR(1)* ou outro ascendente. Nessse caso optamos por utilziar o gerador *Bison/Flex*.

2 Desenvolvimento

Para entender melhor a estratégia de desenvolvimento, devemos antes analisar a gramática da linguagem, que está definida abaixo:

$$
\begin{aligned}
\textit{program} &\rightarrow \text{RECEBA } \textit{varlist} \text{ DEVOLVA } \textit{varlist} \text{ HORADOSHOW } \textit{cmds} \text{ AQUIACABOU} \\
\textit{varlist} &\rightarrow \textit{varname varlist} \mid \textit{varname} \\
\textit{cmds} &\rightarrow \textit{cmd cmds} \mid \textit{cmd} \\
\textit{cmd} &\rightarrow \text{ENQUANTO } \textit{varname} \text{ FACA } \textit{cmds} \text{ FIM} \\
\textit{cmd} &\rightarrow \textit{varname} = \textit{varname}
\end{aligned}
$$

Onde todas as variáveis da linguagem são do tipo inteiro não negativo(*unsigned int*), o não-terminal *varlist* descreve uma lista de variáveis separados por vírgula. O não-terminal RECEBA indica o começo de todo o programa, informando uma lista de variáveis que serão utilizadas no programa.

O não-terminal DEVOLVA sinaliza que logo após ele, virá uma lista de variáveis de saída do programa. O não-terminal HORADOSHOW informa que após ele virá uma sequência de comandos (*cmds*) que devem ser executados. E no final do código tem o comando AQUIACABOU, informando o fim do programa.

Já para o loop ENQUANTO, abordamos as duas pricipais maneiras mostradas no trabalho:

$$
\begin{aligned}
\textit{cmd} &\rightarrow \text{ENQUANTO } \textit{varname} \text{ FACA } \textit{cmds} \text{ FIMENQUANTO} \\
\textit{cmd} &\rightarrow \text{ENQUANTO } \textit{expr} \text{ FACA } \textit{cmds} \text{ FIMENQUANTO}
\end{aligned}
$$

Onde *expr* é uma expressão lógica entre duas variáveis ou entre uma variável e um número. No enunciado do trabalho, mostra que podemos implementar de duas maneiras diferentes, sendo elas utilizando o símbolo do operador ($>$, $<$, $>=$, $<=$) ou então utilizando expressões no formado abaixo. Optamos por utiliar como demonstrado abaixo.

- GT(A,B): A maior que B
- GE(A,B): A maior ou igual a B
- LE(A,B): A menor ou igual a B
- LT(A,B): A menor que B

Com base nas instruções do trabalho, a gramática também precisa ser capaz de interpretar os seguintes comandos:

- SE condição ENTÃO *cmds* FIMSE
- SE condição ENTÃO *cmds* SENÃO *cmds* FIMSE
- SOMA(A,B) | A + B → Optamos por utilizar SOMA(A,B)
- MULT(A,B) | A * B → Optamos por utilizar MULT(A,B)
- EXECUTE(*num*,*cmds*)|EXECUTE *cmds* VEZES *num* FIMEXE→Optamos pela 1^a opção
- ZERO(A)

Vale ressaltar que, como demonstraremos nos exemplos, implementamos na linguagem os não-terminais que marcam o final dos comandos:

- SE condição ENTÃO *cmds* FIMSE
- ENQUANTO *expr* FACA *cmds* FIMENQUANTO

3 Código

O nosso analisador, está estruturado em três partes principais: A parte léxica do nosso analisados, a parte gramatical e um código escrito em C para implementar fila.

3.1 Léxico

Como citado anteriormente, utilizamos o gerador de analisador sintático *Flex/Bison* para construir nosso analisador sintático. Após analisar o problema, geramos o arquivo `lexic.l`.

Este arquivo define as diferentes regras para identificar diferentes tokens do programa escrito na linguagem *HoraDoShow-Script*. Esses tokens serão usados pelo analisador sintático gerado, pelo Bison para construir a árvore de análise sintática do programa.

3.2 Gramática

Utilizando a ferramenta *Bison*, criamos o arquivo `grammar.y`. Onde definimos nele as regras sintáticas, para a linguagem *HoraDoShow-Script*, e como o código deve se comportar em cada caso.

3.3 Auxiliar Lista

Para facilitar a leitura de *varname* e das operações, utilizamos lista duplamente encadeada, onde cada nó representa uma linha, e cada linha possui 2 variáveis do tipo *char*(representando *varname*) e uma variável do tipo *short*(representando *cmd*).

Segue abaixo como montamos as estruturas de dados do nosso programa, no arquivo `llist.c`:

Listing 1: Estrutura Lista

```
typedef struct llist{
    struct llist *ant;
    struct llist *prox;
    LINE line;
}LLIST;
```

Listing 2: Estrutura Nó

```
typedef struct line{
    short cmd;
    char *v1;
    char *v2;
    char *v3;
    char *v4;
}LINE;
```

Neste trecho do código, realizamos a operação de adicionar um comando lido a posição final da lista.

```
void addLLISTend(LLIST *newC, LLIST *llist){
    LLIST *temp = llist;

    while (temp->prox != NULL){
        temp = temp->prox;
    }
    temp->prox = newC;
    newC->ant = temp;
}
```

Aqui realizamos a operação inversa, adicionando o comando lido, ao começo da lista e ajustando os ponteiros($*ant$ e $*prox$).

```
void addLLISTstart(LLIST *newC, LLIST *llist){
    llist->ant = newC;
    newC->prox = llist;
}
```

Nesta parte do código, implementamos uma função que lê o valor de uma variável, e verifica se ela já foi adicionada ao buffer de variáveis. Nós implementamos isso para sanar a necessidade de inicializar variáveis novas.

```
int existsInBuffer( char *var) {
    char *bufferCopy = strdup(buffer);
    char *result = strstr(bufferCopy, var);

    if (bufferCopy == NULL) {
        printf("Buffer vazio.\n");
        return 0;
    }
    if (result != NULL) {
        printf("Substring encontrada: %s\n", result);
        free(bufferCopy);
        return 1;
    } else {
        printf("Substring nao encontrada.\n");
        free(bufferCopy);
        return 0;
    }
}
```

4 Implementações Adicionais

Buscando complementar nossa gramática, adicionamos novas regras, operações e expressões. Segue abaixo a lista do que adicionamos:

- SE *expr* ENTÃO *cmds* FIMSE
- SE *expr* ENTÃO *cmds* SENÃO *cmds* FIMSE
- RSHIFT(A,B) → Realizar bit shift para direita (A = A >> B)
- LSHIFT(A,B) → Realizar bit shift para esquerda (A = A << B)
- DIV(A,B) → Realizar divisão (A = A ÷ B)
- SUB(A,B) → Realizar subtração (A = A − B)
- EQUAL(A,B) → Expressão de igualdade absoluta (A == B)
- DIF(A,B) → Expressão de diferença (A ! = B)
- VETOR(A,B,C) → Declarar Vetor/Matriz (int A[B][C])
- VALORVETOR(A,B,C,D) → Preenche Vetor/Matriz (A[B][C] = D)
- EQUALM(A,B,C,D) → Preenche Vetor/Matriz (D = A[B][C])

5 Execução do código

Para executar o trabalho, utilizar esses comandos:

```
$ bison -d grammar.y
$ flex lexic.l
$ gcc -c lex.yy.c grammar.tab.c
$ gcc -o compile lex.yy.o grammar.tab.o -lfl
$ ./compile Testes/<arquivo>.show
```

6 Pontos a melhorar

6.1 Nome de variável

Durante o trabalho, buscamos seguir a estratégia recomendada no enunciado do trabalho, começamos com um analisador simples e funcional, e fomos complementando ele aos poucos. Porém, tiveram decisões importantes do trabalho, que percebemos estarem equivocadas apenas nas etapas finais do trabalho.

Uma delas foi na maneira de armazenar nomes de variáveis. No nosso código, armazenamos todos os nomes em um vetor de caracteres, onde cada string está separada por uma vírgula, e existe um marcados no meio desse vetor que diferencia as variáveis de entrada e de saída.

Tal implementação acarretou em um erro imporante, onde ao declararmos uma variável, no meio do código, onde seu nome é uma substring de outro varíavel já inicializada, o programa não inicializa essa nova variável.

Segue abaixo um exemplo:

Listing 3: Código de Entrada

```
RECEBA XX
DEVOLVA Z
HORADOSHOW
X = 3
Z = X
AQUIACABOU
```

Listing 4: Código de Saída

```
#include <stdio.h>
    int main(void) {
    int XX;
    printf("RECEBA[XX]:");
    scanf("%d", &XX);
    int Z;
    X = 3;
    Z = X;
    printf("DEVOLVA[Z]:%d", Z);
    return 0;
}
```

Como observado, a variável X não foi inicializada, uma vez que a verificação da existência da variável no buffer (`existsInBuffer( char *var)`) indicou que X é uma substring de XX.

6.2 Operadores Lógicos

No decorrer do desenvolvimento dos comandos "ENQUANTO condição"e "SE condição ENTÃO", pensamos em implementar operadores lógicos para que pudesse haver mais de uma condição a ser satisfeita para que o comando especificado seja executado. Por exemplo:

Listing 5: Código de Saída

```
RECEBA X
DEVOLVA Z
HORADOSHOW
X = 3
Z = 0
ENQUANTO DIF(X,4) AND EQUAL(Z,0) FACA
SOMA(X,1)
SOMA(Z,1)
FIMENQUANTO
AQUIACABOU
```

Porém não conseguimos implementar nenhum dos operadores (AND, OR, XOR).

6.3 Indentação do Código

Outro ponto que não conseguimos implementar, embora nossa escolha de linguagem de destino não tenha impacto, é a indentação do código. A linguagem C não exige indentação para que o código seja compilado corretamente, já que os blocos de código em C são delimitados pelo uso de chaves {}. Porém tal implementação se torna indispensável para determinadas linguagens como *Python*.

6.4 Preenchimento do Vetor/Matriz

Buscamos implementar a declaração e o preenchimento de vetores e matrizes de inteiros, porém não conseguimos implementar uma verificação na hora de atribuir valores a posições específicas. Tal erro se mostra claro quando tentamos atribuir um valor a uma posição que não existe no valor inicialmente declarado. Tentamos implementar uma solução porém não conseguimos.

Listing 6: Código de Entrada

```
RECEBA X
DEVOLVA Z
HORADOSHOW
VETOR(A,1,0)
VALORVETOR(A,3,2,100)
AQUIACABOU
```

Listing 7: Código de Saída

```
#include <stdio.h>
int main(void) {
        int X;
        printf("RECEBA [X]: ");
        scanf("%d", &X);
        int Z;
        int A[1];
        A[3][2] = 100;
        printf("DEVOLVA [Z]: %d", Z);
        return 0;
}
```

Como se pode ver no arquivo de sáida, ele acabou imprimindo a atribuição de valor, mesmo ela gerando erro ao compilar o arquivo .c.

7 Exemplos

Montamos diversos exemplos para comprovar que todas as estruturas do analisador estão funcionando. Segue abaixo alguns exemplos:

7.1 Exemplo 1

Listing 8: Código de Entrada

```
RECEBA X, Y
DEVOLVA Z
HORADOSHOW
EXECUTE(X, SOMA(Y,2))
Z=Y
AQUIACABOU
```

Listing 9: Código de Saída

```
#include <stdio.h>
int main(void) {
	int X;
	printf("RECEBA [X]: ");
	scanf("%d", &X);
	int Y;
	printf("RECEBA [Y]: ");
	scanf("%d", &Y);
	int Z;
	for (int i = 0; i < X; i++) {
	Y = Y + 2 ;
	}
	Z = Y;
	printf("DEVOLVA [Z]: %d", Z);
	return 0;
}
```

7.2 Exemplo 2

Listing 10: Código de Entrada

```
RECEBA X, Y
DEVOLVA Z
HORADOSHOW
ZERO(Z)
SE X ENTAO Z= MULT(X,2)
FIMSE
SE Y ENTAO Z= SOMA(Y,3)
FIMSE
AQUIACABOU
```

Listing 11: Código de Saída

```
#include <stdio.h>
int main(void) {
	int X;
	printf("RECEBA [X]: ");
	scanf("%d", &X);
	int Y;
	printf("RECEBA [Y]: ");
	scanf("%d", &Y);
	int Z;
	Z = 0;
	if (X != 0) {
	X = X * 2;
	Z = X;
	}
	if (Y != 0) {
	Y = Y + 3 ;
	Z = Y;
	}
	printf("DEVOLVA [Z]: %d", Z);
	return 0;
}
```

7.3 Exemplo 3

Listing 12: Código de Entrada

```
RECEBA X
DEVOLVA Z
HORADOSHOW
NUM = 0
RESULT=1
ENQUANTO GT(X,NUM) FACA
SOMA(NUM,1)
RESULT = MULT(RESULT,NUM)
Z=RESULT
FIMENQUANTO
AQUIACABOU
```

Listing 13: Código de Saída

```
#include <stdio.h>
int main(void) {
        int X;
        printf("RECEBA [X]: ");
        scanf("%d", &X);
        int Z;
        int NUM;
        NUM = 0;
        int RESULT;
        RESULT = 1;
        while (X > NUM){
        NUM = NUM + 1 ;
        RESULT = RESULT * NUM;
        RESULT = RESULT;
        Z = RESULT;
        }
        printf("DEVOLVA [Z]: %d", Z);
        return 0;
}
```

7.4 Exemplo 4

Listing 14: Código de Entrada

```
RECEBA X,XX
DEVOLVA Z
HORADOSHOW
ZERO(Z)
Z = SOMA(Z,10)
SE GE(Z,3) ENTAO
ENQUANTO GE(Z,3) FACA
Z = SUB(Z,4)
FIMENQUANTO
FIMSE
AQUIACABOU
```

Listing 15: Código de Saída

```
#include <stdio.h>
int main(void) {
        int X;
        printf("RECEBA [X]: ");
        scanf("%d", &X);
        int XX;
        printf("RECEBA [XX]: ");
        scanf("%d", &XX);
        int Z;
        Z = 0;
        Z = Z + 10 ;
        Z = Z;
        if(Z >= 3){
        while(Z >= 3){
        Z = Z - 4 ;
        Z = Z;
        }
        }
        printf("DEVOLVA [Z]: %d", Z);
        return 0;
}
```

7.5 Sequência de Fibonacci

Listing 16: Código de Entrada

```
RECEBA NUM,T1,T2,PROX
DEVOLVA TERMO
HORADOSHOW
T1 = 0
T2 = 1
PROX = SOMA(T1,T2)
EXECUTE(NUM, T1=T2 T2=PROX PROX=SOMA(T1
    ,T2))
TERMO=PROX
AQUIACABOU
```

Listing 17: Código de Saída

```
#include <stdio.h>
int main(void) {
        int NUM;
        printf("RECEBA [NUM]: ");
        scanf("%d", &NUM);
        int T1;
        printf("RECEBA [T1]: ");
        scanf("%d", &T1);
        int T2;
        printf("RECEBA [T2]: ");
        scanf("%d", &T2);
        int PROX;
        printf("RECEBA [PROX]: ");
        scanf("%d", &PROX);
        int TERMO;
        T1 = 0;
        T2 = 1;
        T1 = T1 + T2 ;
        PROX = T1;
        for (int i = 0; i < NUM; i++) {
        T1 = T2;
        T2 = PROX;
        T1 = T1 + T2 ;
        PROX = T1;
        }
        TERMO = PROX;
        printf("DEVOLVA [TERMO]: %d",
            TERMO);
        return 0;
}
```

7.6 Fatorial

Listing 18: Código de Entrada

```
RECEBA X
DEVOLVA Z
HORADOSHOW
Z = 1
ENQUANTO GT(X,1) FACA
MULT(Z,X)
SUB(X,1)
FIMENQUANTO
AQUIACABOU
```

Listing 19: Código de Saída

```
#include <stdio.h>
int main(void) {
        int X;
        printf("RECEBA [X]: ");
        scanf("%d", &X);
        int Z;
        Z = 1;
        while (X > 1){
        Z = Z * X;
        X = X - 1 ;
        }
        printf("DEVOLVA [Z]: %d", Z);
        return 0;
}
```

7.7 Potência

Listing 20: Código de Entrada

```
RECEBA X,POWER
DEVOLVA X
HORADOSHOW
SUB(POWER,1)
EXECUTE(POWER,MULT(X,X))
AQUIACABOU
```

Listing 21: Código de Saída

```
#include <stdio.h>
int main(void) {
        int X;
        printf("RECEBA [X]: ");
        scanf("%d", &X);
        int POWER;
        printf("RECEBA [POWER]: ");
        scanf("%d", &POWER);
        int X;
        POWER = POWER - 1 ;
        for (int i = 0; i < POWER; i++)
           {
        X = X * X;
        }
        printf("DEVOLVA [X]: %d", X);
        return 0;
}
```

7.8 Vetor

Listing 22: Código de Entrada

```
RECEBA X
DEVOLVA Z
HORADOSHOW
VETOR(VET,X,0)
T=0
ENQUANTO GT(X,1) FACA
    VALORVETOR(VET,X,0,T)
    SOMA(T,1)
    SUB(X,1)
FIMENQUANTO
AQUIACABOU
```

Listing 23: Código de Saída

```
#include <stdio.h>
int main(void) {
        int X;
        printf("RECEBA [X]: ");
        scanf("%i", &X);
        int Z;
        int VET[X];
        int T;
        T = 0;
        while (X > 1){
        VET[X] = T;
        T = T + 1 ;
        X = X - 1 ;
        }
        printf("DEVOLVA [Z]: %d", Z);
        return 0;
}
```

7.9 Matriz

Listing 24: Código de Entrada

```
RECEBA X
DEVOLVA Z
HORADOSHOW
AUX1=0
AUX2=0
INC=1
TOTAL=0
VETOR(MAT, 10, 5)
EXECUTE(10,
    EXECUTE(5,
        VALORVETOR(MAT,AUX1,AUX2,INC)
        SOMA(INC,1)
        EQUALM(X,MAT,AUX1,AUX2)
        SOMA(TOTAL,X)
        SOMA(AUX2,1)
        ZERO(X)
    )
    ZERO(AUX2)
    SOMA(AUX1,1)
)
Z=TOTAL
AQUIACABOU
```

Listing 25: Código de Saída

```
#include <stdio.h>
int main(void) {
        int X;
        printf("RECEBA [X]: ");
        scanf("%i", &X);
        int Z;
        int AUX1;
        AUX1 = 0;
        int AUX2;
        AUX2 = 0;
        int INC;
        INC = 1;
        int TOTAL;
        TOTAL = 0;
        int MAT[10][5];
        for (int a = 0; a < 10; a++) {
        for (int b = 0; b < 5; b++) {
        MAT[AUX1][AUX2] = INC;
        INC = INC + 1 ;
        X=MAT[AUX1][AUX2];
        TOTAL = TOTAL + X ;
        AUX2 = AUX2 + 1 ;
        X = 0;
        }
        AUX2 = 0;
        AUX1 = AUX1 + 1 ;
        }
        Z = TOTAL;
        printf("DEVOLVA [Z]: %d", Z);
        return 0;
}
```

Para a realização do trabalho, utilizamos vídeos no *YouTube*, conteúdo disponível na plataforma EAD e tutoriais da internet. Segue abaixo o endereço web do conteúdo utilizado:

Referências

[1] *Documentação Bison 3.8.1*. 2021. URL: https://www.gnu.org/software/bison/manual/bison.html.

[2] *Documentação Desenho de Diagramas LaTeX*. 2023. URL: https://www.overleaf.com/learn/latex/CircuiTikz_package.

[3] *Documentação LaTeX*. 2023. URL: https://www.latex-project.org/help/documentation/.

[4] *Playlist de vídeos com aulas sobre Compiladores*. 2020. URL: https://www.youtube.com/watch?v=TQzFCHQ-muc&list=PLDDV68kBkoTYavaJmTFt8poB_F8vSCFOd&ab_channel=Universo%20Comput%C3%A1vel.