Trabalho Final INF1022

2110317 Isabella Gomes
2020273 Thalita Rangel
2023.2

1. O QUE FOI IMPLEMENTADO

Expandimos a gramática introduzindo novos comandos, enriquecendo assim as capacidades da linguagem HoraDoShow-Script e possibilitando a expressão de lógicas mais avançadas nos programas.

Acrescentamos um novo símbolo não terminal chamado `LINEBREAK` após o último símbolo não terminal na primeira regra. Esse símbolo tem a responsabilidade de reconhecer e processar quebras de linha, proporcionando uma manipulação adequada desse aspecto textual.

Para alinhar a linguagem com as convenções da linguagem C, que utiliza um único retorno de variável, substituímos o `varlist` que precedia o comando `HORADOSHOW` por um novo símbolo não terminal chamado `retorn`. Com essa modificação, `varlist` pode derivar várias entradas para a função, enquanto `retorn` representa a derivação de uma única saída.

Na segunda regra, efetuamos a transição da recursão à direita para a recursão à esquerda. Essa modificação foi realizada com o intuito de otimizar o espaço na pilha durante a análise sintática, promovendo uma eficiência aprimorada no processo de compilação.

Introduzimos o não terminal `VIRGULA` para facilitar a visualização do `varlist`, considerando que este representa uma lista de variáveis separadas por vírgula.

Criamos o comando `ENQUANTO` (while), substituindo o comando original. Agora, esse comando utiliza o não terminal `expr`, e implementamos as regras para esse não terminal, incluindo operadores como `MAIOR`, `MENOR`, `MAIORIGUAL` e `MENORIGUAL`.

Adicionamos os comandos `SE-ENTAO` e `SE-ENTAO SENAO`, equivalentes a if-then e if-then-else, respectivamente. Para evitar ambiguidades, acrescentamos ao final de cada expressão desses comandos o símbolo não terminal `END`.

Introduzimos o comando `EXECUTE cmds VEZES`, representando uma repetição definida. Para evitar ambiguidades, adicionamos ao final desse comando o símbolo não terminal `END`.

Acrescentamos os comandos `SOMA`, `MULTIPLICA` e `ZERO`, proporcionando operações matemáticas básicas para ampliar a expressividade da linguagem.

Adicionamos o não terminal `elem`, contendo regras para determinar se um elemento é uma variável (String) ou um número (Int). A inclusão de `elem` foi aplicada em todos os casos em que o segundo elemento pode ser tanto um número quanto uma variável.

Utilizamos a seguinte gramática para o analisador:

program → RECEBA varlist DEVOLVA retorn HORADOSHOW cmds AQUIACABOU LINEBREAK

varlist → varlist VIRGULA VARNAME | VARNAME

retorn → VARNAME

cmds → cmds cmd | cmd

cmd → ENQUANTO VARNAME FACA cmds END

cmd → VARNAME = elem

cmd → VARNAME = cmd

cmd → SE VARNAME ENTAO cmds END

cmd → SE VARNAME ENTAO cmds SENAO cmds END

cmd → ZERO(VARNAME)

cmd → VARNAME + elem

cmd → VARNAME * elem

cmd → EXECUTE cmds VEZES elem END

cmd → ENQUANTO expr FACA cmds END

expr → VARNAME > elem

expr → VARNAME < elem

expr → VARNAME ≥ elem

expr → VARNAME ≤ elem

elem → VARNAME | num

2. COMO FOI IMPLEMENTADO

Durante o processo de implementação do compilador para a linguagem HoraDoShow-Script, desenvolvemos as ferramentas Lex e Yacc, responsáveis pela análise léxica e sintática, respectivamente.

Na fase de análise léxica, utilizamos a ferramenta Lex para reconhecer os tokens na entrada do compilador. Para isso, especificamos as expressões regulares que definem a estrutura da linguagem HoraDoShow-Script.

Cada token reconhecido pelo Lex aciona uma ação, e estas ações foram definidas para executar as operações necessárias durante a compilação. As ações envolvem, por exemplo, a geração de código ou a manipulação de variáveis e estruturas de controle.

A ferramenta Yacc foi utilizada para gerar o parser e trabalhar sobre as funções geradas pelo Lex. Ela lê o input fornecido e constrói a árvore sintática necessária para a compreensão do programa.

Na parte de definição, incluímos as bibliotecas necessárias para a execução do código que reconhece os tokens do input. Também foram declarados os tokens, símbolos não terminais e a gramática de atributos.

Associamos a cada regra gramatical uma sequência de comandos em C. Para cada regra, a sequência de comandos envolve a criação dinâmica de strings, alocação de memória, geração de código em C correspondente à regra, e a atribuição dessa string ao símbolo não terminal mais à esquerda da regra.

O código Yacc foi desenvolvido de forma a gerar uma função nos moldes da linguagem C a partir do input fornecido em HoraDoShow-Script.

A main do Yacc completa o código em C, incluindo a inclusão das bibliotecas necessárias, a análise sintática e a criação da função. Ao final, é impressa a main do arquivo de saída, que exibe o retorno da função gerada pelo parser, utilizando os valores passados no terminal durante a chamada do executável.

3. LIMITAÇÕES

Se a variável não for incluída na varlist após o comando RECEBA, não será possível utilizá-la no restante do código, pois ela não estará declarada. Além disso, o código em HoraDoShow deve ser escrito todo na mesma linha; não permitimos quebras de linha na leitura, a menos que ocorram após o comando AQUIACABOU.

4. TESTES REALIZADOS

No arquivo zip que contém o projeto, foram deixados alguns arquivos horadoshow contendo programas de exemplo, que podem ser utilizados para gerarem código objeto em C. Os códigos gerados por tais arquivos estão exibidos a seguir.

Teste1.horadoshow:

Na gramática HoraDoShow:

RECEBA Y,X DEVOLVA Y HORADOSHOW Y=X AQUIACABOU

Saída no arquivo .c:

Teste1.c > ...

```c
#include <stdio.h>
#include <stdlib.h>

        int horaDoShow(int Y, int X) {

    Y = X;

    return Y;

}

int main(int argc, char *argv[]) {

    printf("Saida -> %d\n", horaDoShow(atoi(argv[1]), atoi(argv[2])));
    return 0;

}
```

Teste2.horadoshow

(Exemplo do enunciado do trabalho com modificação na varlist após o RECEBA que recebe X, Y e Z)

Na gramática HoraDoShow:

RECEBA X,Y,Z DEVOLVA Z HORADOSHOW EXECUTE Y=Y+2 VEZES X END Z=Y AQUIACABOU

Saída no arquivo .c:

Teste2.c > ...

```c
#include <stdio.h>
#include <stdlib.h>

        int horaDoShow(int X, int Y, int Z) {

    for (int i=0; i<X; i++) {
        Y = Y + 2;
    }
    Z = Y;

    return Z;

}

int main(int argc, char *argv[]) {

    printf("Saida -> %d\n", horaDoShow(atoi(argv[1]), atoi(argv[2]), atoi(argv[3])));
    return 0;

}
```

Teste3.horadoshow

(Exemplo do enunciado do trabalho com modificação na varlist após o RECEBA que recebe X, Y e Z)

Na gramática HoraDoShow:

RECEBA X,Y,Z DEVOLVA Z HORADOSHOW ZERO(Z) SE X ENTAO Z=X*2 END SE Y ENTAO Z=Y+3 END AQUIACABOU

Saída no arquivo .c:

```c
// Teste3.c > horaDoShow(int, int, int)
#include <stdio.h>
#include <stdlib.h>

                int horaDoShow(int X, int Y, int Z) {

    Z = 0;
    if (X) {
        Z = X * 2;
    }
    if (Y) {
        Z = Y + 3;
    }

    return Z;

}

int main(int argc, char *argv[]) {

    printf("Saida -> %d\n", horaDoShow(atoi(argv[1]), atoi(argv[2]), atoi(argv[3])));
    return 0;

}
```

Teste4.horadoshow

(Exemplo do enunciado do trabalho com modificação na varlist após o RECEBA que recebe X, Z, NUM e RESULT)

Na gramática HoraDoShow:

RECEBA X,Z,NUM,RESULT DEVOLVA Z HORADOSHOW NUM=0 RESULT=1 ENQUANTO X>NUM FACA NUM=NUM+1 RESULT=RESULT*NUM Z=RESULT END AQUIACABOU

Saída no arquivo .c:

```c
// Teste4.c > main(int, char * [])
#include <stdio.h>
#include <stdlib.h>

            int horaDoShow(int X, int Z, int NUM, int RESULT) {

    NUM = 0;
    RESULT = 1;
    while (X > NUM) {
        NUM = NUM + 1;
    RESULT = RESULT * NUM;
    Z = RESULT;
    }

    return Z;

}

int main(int argc, char *argv[]) {

    printf("Saida -> %d\n", horaDoShow(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]), atoi(argv[4])));
    return 0;

}
```

Teste5.horadoshow

Na gramática HoraDoShow:

RECEBA X,Z DEVOLVA Z HORADOSHOW SE X ENTAO EXECUTE Z=Z+9 VEZES X END SENAO ENQUANTO X<=Z FACA Z=Z*4 X=X+1 END END AQUIACABOU

Saída no arquivo .c:

Teste5.c > horaDoShow(int, int)

```c
#include <stdio.h>
#include <stdlib.h>

int horaDoShow(int X, int Z) {

    if (X) {
        for (int i=0; i<X; i++) {
        Z = Z + 9;
    }
    }
    else{
        while (X <= Z) {
        Z = Z * 4;
    X = X + 1;
    }
    }

    return Z;

}

int main(int argc, char *argv[]) {

    printf("Saida -> %d\n", horaDoShow(atoi(argv[1]), atoi(argv[2])));
    return 0;

}
```

Para realizar os testes demonstrados acima, utilizamos os seguintes comandos no terminal tendo como exemplo o Teste1:

```
isabellagomes@MacBook-Pro-de-Isabella HoraDoShow---INF1022 % yacc -d HoraDoShow.y
isabellagomes@MacBook-Pro-de-Isabella HoraDoShow---INF1022 % lex HoraDoShow.l
isabellagomes@MacBook-Pro-de-Isabella HoraDoShow---INF1022 % gcc -c lex.yy.c y.tab.c
isabellagomes@MacBook-Pro-de-Isabella HoraDoShow---INF1022 % gcc -o parser lex.yy.o y.tab.o -ll
isabellagomes@MacBook-Pro-de-Isabella HoraDoShow---INF1022 % ./parser <Teste1.horadoshow>  Teste1.c
isabellagomes@MacBook-Pro-de-Isabella HoraDoShow---INF1022 % gcc -Wall -o parser Teste1.c
isabellagomes@MacBook-Pro-de-Isabella HoraDoShow---INF1022 % ./parser Y X
Saida -> 0
```

Para rodar outros testes basta trocar “Teste1" para o nome de seu arquivo e na linha de comando do ./parser colocar os argumentos que você quer passar para a função.