Universidade Estadual de Santa Cruz – UESC

Projeto 4D

Discente: Yan Costa Macedo

Matrícula: 201820218

Disciplina Compiladores.

Curso Ciência da Computação

Semestre 2022.2

Professor César Alberto Bravo Pariente

Ilhéus – BA
2022

Índice

1. Introdução

Neste relatório é descrita as primeiras etapas da implementação de um compilador simples, que tem como intuito transformar um código fonte em instruções p-code. O projeto total ainda está incompleto mas a implementação encontra-se em andamento para corresponder o resultado esperado. Nesta seção serão apontados o objetivo de se implementar um compilador, e detalhada a linguagem de programação aceita por este até o momento. Inicialmente, o projeto consistia em fazer um algoritmo que utilizasse um arquivo .txt para definir a linguagem de programação que seria utilizada. Esta metodologia por si só, está em desenvolvimento e encontra-se na pasta “PESSOAL” do projeto 2. Devido ao cronograma e disponibilidade do discente, o projeto seguiu apenas com a definição de uma linguagem. Os módulos desenvolvidos para o funcionamento do compilador foram: Parser, o qual analisa uma programa de entrada e determina uma estrutura segundo uma gramática; ADA2ASA que transforma a árvore de análise descendente (ADA) em árvore de sintaxe abstrata (ASA); Assimilarmente, por enquanto não será tratado sobre os módulos ASA2NP* / NP2GCI / GCI2GCO nesse relatório.

1.1. Objetivo do Compilador

Oferecer a análise de programas, utilizando regras gramaticais e a conversão de códigos fonte para o código objeto p-code, permitindo sua execução em uma p-code-machine.

1.2. Linguagem de Programação

A estrutura da linguagem de programação para a qual o compilador está sendo desenvolvido é primeiramente a utilização de uma função, que pode ser a main (‘m’), uma outra função (‘n’ ou ‘g’). A estrutura dessas funções é, por exemplo, m(){ C; r(E) }, onde C pode ser derivado em atribuições, manipulações(soma, subtração,multiplicação ou divisão), laços de repetições ou condicionais, por fim a função de retorno.

A estrutura de atribuições para uma variável é dada pela utilização do nome da variável(‘h’, ‘i’, ‘j’, ‘k’, ‘z’) e em seguida o símbolo de igual(‘=’) e por fim um número(‘0’ ou ’1’), uma variavel(‘x’ ou ‘y’) ou uma expressão aritmética utilizando os números e variáveis organizadas por parenteses, por exemplo “(x*(1/1)+(y-0))” fazendo uso dos símbolos de operação binária: ‘+’, ‘-’, ‘*’ e ‘/’.

As manipulações como mostradas anteriormente, são essas expressões aritméticas e que são utilizadas também como condicional dos laços de repetição e dos laço if.

Os Laços de repetições contem a estrutura de w(E){C;} ou o(E;E;E){C;}, tal que C continua como a explicação dada anteriormente, e o E é a expressão aritmética. O mesmo se diz sobre a condicional f(E){C;} Como mostrado, o fim de todo conjunto é definido por um ponto-e-vírgula.

A função de retorno é representada por um ‘r’, e segue o padrão r(E);, onde ‘E’ é uma expressão.

```
p1 = S → M
p2 = S → GM
p3 = S → NGM
p4 = F → f(){C;r(E);}
p5 = G → g(){C;r(E);}
p6 = M → m(){C;r(E);}
p7 = E → 0
p8 = E → 1
p9 = E → x
p10 = E → y
p11 = E → (EXE)
p12 = X → +
p13 = X → -
p14 = X → *
p15 = X → /
p16 = C → h=E
p17 = C → i=E
p18 = C → j=E      You, last month • Att readme's
p19 = C → k=E
p20 = C → z=E
p21 = C → (EXE)
p22 = C → w(E){C;}
p23 = C → f(E){C;}
p24 = C → o(E;E;E){C;}
```

2. Módulos do compilador

Esta seção apresenta uma descrição para cada módulo do compilador. Para os módulos implementados é demonstrado como fazer a execução do algoritmo individualmente e como manipular o arquivo de entrada.

2.1. P-Code-Machine(implementation)

Este módulo é o responsável para execução do código objeto (P-CODE). Utilizaremos o módulo 1 para explicação da implementação do executável.

Para executar o código, deve haver um input_file.txt na pasta de origem.

Para compilar o módulo:

```
$ gcc p-code.c -o p-code
```

para executar:

```
$ ./p-code
```

2.2. Scanner (Analisador Léxico)

Este módulo realiza o processo de fazer uma varredura caractere por caractere do programa fonte submetido ao txt, traduzindo tokens. Neste módulo são reconhecidas todas as palavras reservadas, constantes, identificadores e qualquer outra palavra que pertença à linguagem adotada.

Para executar o código, deve haver um input_file.txt na pasta de origem.
Para compilar o módulo:

```
$ gcc proj2b.c -o proj2b
```

para executar:

```
$ ./proj2b
```

2.3. Parser (Analisador Sintático)

Este módulo realiza o processo de analisar uma cadeia de entrada. O parser converte texto na entrada em uma estrutura de dados (No caso desse projeto, uma arvóre). O parser está diretamente ligado ao scanner, o parser o ativa para obter os símbolos (tokens) e assim usar um conjunto de regras para construir a árvore sintática e a tabela de símbolos. A saída deste módulo é a árvore de análise sintática em um vetor.

Para executar o código, deve haver um input_file.txt na pasta de origem.
Para compilar o módulo:

```
$ gcc proj2a.c -o proj2a
```

para executar :

```
$ ./proj2a
```

O código fonte da implementação deste módulo está no Apêndice B.

2.4. ADA2ASA (Árvore Sintática para Árvore Sintática Abstrata)

Este é o módulo responsável por transformar a árvore de análise (ADA) na árvore sintática abstrata (ASA). Consideramos nessa etapa todos os símbolos que são operandos e operadores, entretando os símbolos como parênteses e vírgulas são desconsiderados. A saída deste módulo é a árvore sintática abstrata em forma de vetor com os símbolos e os respectivos índices de cada um (Represantadas por uma struct).

Para executar o código, deve haver um input_file.txt na pasta de origem.
Para compilar o módulo:

```
$ gcc proj2c.c -o proj2c -w
```

para executar :

```
$ ./proj2c
```

O código fonte da implementação deste módulo está no Apêndice C.

2.5. ASA2NP* (Árvore Sintática Abstrata para Notação Polonesa)

Este módulo é responsável por transformar a árvore de sintaxe abstrata (ASA) em notação polonesa ou polonesa reversa (NP*). Convertendo a árvore para notação polonesa para uma busca pré-ordem. Esse módulo não foi implementado até o momento e está em desenvolvimento.

2.6. NP2GCI (Gerador de Código Intermediário)

Este módulo transforma a saída do módulo de notação polonesa para um código intermediário. Esse módulo também não foi implementado até o momento e está em desenvolvimento.

2.7. GCI2GCO (Gerador de Código Objeto)

Este módulo é responsável por transformar o código intermediário em código objeto. O código objeto é gerado em p-code. Assim, este componente gera como saída, um programa semanticamente equivalente ao código fonte dado como entrada pelo usuário, mas escrito em p-code. Esse módulo não foi implementado. Apenas o p-code.

3. Implementações

Nesta seção será introduzida e acompanhada os Códigos que foram utilizados para compilação do compilador, e suas explicações.

3.1. Projeto 01 - P-CODE-MACHINE

O primeiro projeto foi composto pela geração do código que lê códigos em linguagem P-CODE e os executa. Foram atribuídas ao longo do projeto, 3 Atividades que serão demonstradas a seguir:

p-code.c

-Função que enumera as funções que utilizamos no PCODE

```
enum
{
    LIT,
    OPR,
    LOD,
    STO,
    CAL,
    INT,
    JMP,
    JPC
} instructionC;
char *instructionString[] = {"LIT", "OPR", "LOD", "STO", "CAL", "INT", "JMP", "JPC"};
```

-Estrutura de comandos do P-CODE

```
typedef struct
{
    int operation; // f
    int level;     // l
    int argument;  // a
} instructionSTRUCT;

// Array of code-instructions >INPUT<
instructionSTRUCT instructions[C_MAX_INSTRUCTIONS];
```

-Função main responsavel pela chamada das funções pela obtenção dos dados e pela interface

```
printf("Please enter the instructions (Instruction END 0 0 to stopstack the input)\n");

for (int i = 0; i < C_MAX_INSTRUCTIONS; i++)
{

    scanf("\n%[^ ] %d %d", function, &l, &a);

    if (strcmp(function, "END") == 0)
    {
        printf("——End of Program——\n");
        break;
    }
    else
    {
        int code = getInstructionCode(function);
        if (code != -1)
        {
            instructionCurrent.operation = code;
            instructionCurrent.level = l;
            instructionCurrent.argument = a;

            // Store the instruction for Logs
            instructions[i] = instructionCurrent;
            totalInstructions++;
        }
        else
        {
            printf("Unknow Instruction\n");
            i--; // Rewind the program
        }
    }
}
printf("\n ... Starting the P-code ... \n");
printf("%-10s %-15s %-15s %-15s %-15s %s\n", "Inst", "Level", "Arg", "topstack", "program", "Stack");
do
{
    execute();
} while (program != 0);

return 0;
```

- Função que retorna o nome da função utilizada(LIT OPR ...)

```
char *getInstructionName(int i)
{
    return instructionString[i];
}
```

-função de execução principal do Switch case para tratamentos das funções

```
void execute()
{
    instructionSTRUCT instructionCurrent = instructions[program]; // Stores the Current instruction

    program++;
    switch (instructionCurrent.operation)
    {
    case LIT:
        topstack++;
        stack[topstack] = instructionCurrent.argument;
        break;
    case OPR:
        switch (instructionCurrent.argument)
        {
        case 0: // RTN
            topstack = base - 1;
            program = stack[topstack + 3];
            base = stack[topstack + 2];
            break;
        case 1: // NEG
            stack[topstack] = -1 * stack[topstack];
            break;
        case 2: // ADD
            topstack--;
            stack[topstack] += stack[topstack + 1];
            break;
        case 3: // SUB
            topstack--;
            stack[topstack] -= stack[topstack + 1];
            break;
        case 4: // MUL
```

3.2. Projeto 01 - abcd

3.2.1.Automato:

```
Linha 1  -> Accept : 01010101


Linha 2  -> Accept : 01111111


Linha 3  -> Accept : 01111000


Linha 4  -> Reject : 01150000


Linha 5  -> Accept : 01111110111


Linha 6  -> Reject : 01as001414


Linha 7  -> Accept : 111111100000
```

-Funções Q0 e Q1

```c
int Q1(char *X, int i)
{

    if (X[i] == '\n')
        return 1;
    else if (X[i] == '1' || X[i] == '0')
        return Q1(X, i + 1);
    else
        return 0; // error
}

int Q0(char *X)
{
    if (X[0] == '0' || X[0] == '1')
        return Q1(X, 1);
    else
        return 0; // error
}
```

-Função de Execução do autómato e verificação de aceitação e negação

```c
void automato(char *line)
{
    int result;

    result = Q0(line);

    if (result == 1)
        printf("Accept : %c ", line);
    else if (result == 0)
        printf("Reject : %s ", line);

    printf("\n\n\n");
}
```

-Função main responsavel pela chamada e leitura do archive

```c
void main(int argc, char *argv[])
{
    FILE *archive;
    char line[100];
    char *result;
    int i;

    // Abre um arquivo TEXTO para LEITURA
    archive = fopen("ArqTeste.txt", "rt");

    if (archive == NULL) // Se houve erro na abertura
    {
        printf("Problemas na abertura do arquivo\n");
        return;
    }

    i = 1;
    while (!feof(archive))
    {
        // Lê uma linha (inclusive com o '\n')
        result = fgets(line, 100, archive);

        printf("Linha %d  → ", i);

        if (result) // Se foi possível ler
            automato(line);
        i++;
    }
    fclose(archive);
}
```

-PCODE

3.2.2. Exercício 01 :

Soma de dois números inteiros;

Resolução:

```
...Starting the P-code...
Inst        Level           Arg             topstack        program         Stack
INT         0               4               0               1               0 0 0 0
LIT         0               6               6               2               0 0 0 0 6
LIT         0               4               4               3               0 0 0 0 6 4
OPR         0               2               10              4               0 0 0 0 10
STO         0               3               10              5               0 0 0 10
OPR         0               0               0               0               0
kirito@kirito-Aspire-A515-41G:~/Documentos/Compiladores/PCODE - 1$ _
```

.3.2.3. Exercício 02:

Soma dos números naturais de 1 até 10;

Resolução:

```
➜  PCODE - 1 cd "/home/kirito/Documentos/Compiladores/PCODE - 1"
➜  PCODE - 1 ./"p-code"
Please enter the instructions (Instruction END 0 0 to stopstack the input)
INT 0 4
LIT 0 10
STO 0 3
LOD 0 3
LOD 0 3
LIT 0 1
OPR 0 3
STO 0 3
LOD 0 3
JPC 0 13
LOD 0 3
OPR 0 2
JMP 0 4
STO 0 3
OPR 0 0
END 0 0
---End of Program---
```

```
OPR         0               2               55              12              0 0 0 1 55
JMP         0               4               55              4               0 0 0 1 55
LOD         0               3               1               5               0 0 0 1 55 1
LIT         0               1               1               6               0 0 0 1 55 1 1
OPR         0               3               0               7               0 0 0 1 55 0
STO         0               3               55              8               0 0 0 0 55
LOD         0               3               0               9               0 0 0 0 55 0
JPC         0               13              55              13              0 0 0 0 55
STO         0               3               55              14              0 0 0 55
OPR         0               0               0               0               0
➜  PCODE - 1 _
```

3.2.4. Exercício 03:

Soma dos números naturais de 1 até 100;

```
➜  PCODE - 1 cd "/home/kirito/Documentos/Compiladores/PCODE - 1"
 ./"p-code"
➜  PCODE - 1 ./"p-code"
Please enter the instructions (Instruction END 0 0 to stopstack the input)
INT 0 4
LIT 0 100
STO 0 3
LOD 0 3
LOD 0 3
LIT 0 1
OPR 0 3
STO 0 3
LOD 0 3
JPC 0 13
LOD 0 3
OPR 0 2
JMP 0 4
STO 0 3
OPR 0 0
END 0 0
---End of Program---
STO       0       3       5049    8       0 0 0 1 5049
LOD       0       3       1       9       0 0 0 1 5049 1
JPC       0       13      5049    10      0 0 0 1 5049
LOD       0       3       1       11      0 0 0 1 5049 1
OPR       0       2       5050    12      0 0 0 1 5050
JMP       0       4       5050    4       0 0 0 1 5050
LOD       0       3       1       5       0 0 0 1 5050 1
LIT       0       1       1       6       0 0 0 1 5050 1 1
OPR       0       3       0       7       0 0 0 1 5050 0
STO       0       3       5050    8       0 0 0 0 5050
LOD       0       3       0       9       0 0 0 0 5050 0
JPC       0       13      5050    13      0 0 0 0 5050
STO       0       3       5050    14      0 0 0 5050
OPR       0       0       0       0       0
➜  PCODE - 1
```

Resolução:

3.2.5. Exercício 04:

Soma dos quadrados dos números naturais de 1 até 100(iterativamente);

Resolução:

```
➜  PCODE - 1 cd "/home/kirito/Documentos/Compiladores/PCODE - 1"
 ./"p-code"
➜  PCODE - 1 ./"p-code"
Please enter the instructions (Instruction END 0 0 to stopstack the input)
INT 0 4
LIT 0 100
STO 0 3
LOD 0 3
LOD 0 3
OPR 0 4
LOD 0 3
LIT 0 1
OPR 0 3
STO 0 3
LOD 0 3
JPC 0 17
LOD 0 3
LOD 0 3
OPR 0 4
OPR 0 2
JMP 0 6
STO 0 3
OPR 0 0
END 0 0
```

Compiled successfully!

```
LOD       0           3           1           11          0 0 0 1 338349 1
JPC       0           17          338349      12          0 0 0 1 338349
LOD       0           3           1           13          0 0 0 1 338349 1
LOD       0           3           1           14          0 0 0 1 338349 1 1
OPR       0           4           1           15          0 0 0 1 338349 1
OPR       0           2           338350      16          0 0 0 1 338350
JMP       0           6           338350      6           0 0 0 1 338350
LOD       0           3           1           7           0 0 0 1 338350 1
LIT       0           1           1           8           0 0 0 1 338350 1 1
OPR       0           3           0           9           0 0 0 1 338350 0
STO       0           3           338350      10          0 0 0 0 338350
LOD       0           3           0           11          0 0 0 0 338350 0
JPC       0           17          338350      17          0 0 0 0 338350
STO       0           3           338350      18          0 0 0 338350
OPR       0           0           0           0           0
➜  PCODE - 1 _
```

3.2.6. Exercício 05:

Soma dos cubos dos números naturais de 1 até 100(iterativamente);

Resolução:

```
➜  PCODE - 1 ./"p-code"
Please enter the instructions (Instruction END 0 0 to stopstack the input)
INT 0 4
LIT 0 100
STO 0 3
LOD 0 3
LOD 0 3
LOD 0 3
OPR 0 4
OPR 0 4
LOD 0 3
LIT 0 1
OPR 0 3
STO 0 3
LOD 0 3
JPC 0 21
LOD 0 3
LOD 0 3
LOD 0 3
OPR 0 4
OPR 0 4
OPR 0 2
JMP 0 8
STO 0 3
OPR 0 0
END 0 0_
```

```
LOD       0           3           1           13          0 0 0 1 25502499 1
JPC       0           21          25502499    14          0 0 0 1 25502499
LOD       0           3           1           15          0 0 0 1 25502499 1
LOD       0           3           1           16          0 0 0 1 25502499 1 1
LOD       0           3           1           17          0 0 0 1 25502499 1 1 1
OPR       0           4           1           18          0 0 0 1 25502499 1 1
OPR       0           4           1           19          0 0 0 1 25502499 1
OPR       0           2           25502500    20          0 0 0 1 25502500
JMP       0           8           25502500    8           0 0 0 1 25502500
LOD       0           3           1           9           0 0 0 1 25502500 1
LIT       0           1           1           10          0 0 0 1 25502500 1 1
OPR       0           3           0           11          0 0 0 1 25502500 0
STO       0           3           25502500    12          0 0 0 0 25502500
LOD       0           3           0           13          0 0 0 0 25502500 0
JPC       0           21          25502500    21          0 0 0 0 25502500
STO       0           3           25502500    22          0 0 0 25502500
OPR       0           0           0           0           0
➜  PCODE - 1
```

3.2.7. Exercício 06:

Main chama função Fat(5);

Resolução:

```
➜  PCODE - 1 ./"p-code"
Please enter the instructions (Instruction END 0 0 to stopstack the input)
INT 0 4
LIT 0 5
STO 0 3
LOD 0 3
STO 0 7
CAL 0 9
LOD 0 7
STO 0 3
OPR 0 0
INT 0 4
LOD 0 3
LOD 0 3
LIT 0 1
OPR 0 3
STO 0 3
LOD 0 3
JPC 0 20
LOD 0 3
OPR 0 4
JMP 0 11
STO 0 3
OPR 0 0
END 0 0
```

Compiled successfully!

```
JPC     0       20      120     17      1 1 6 1 120
LOD     0       3       1       18      1 1 6 1 120 1
OPR     0       4       120     19      1 1 6 1 120
JMP     0       11      120     11      1 1 6 1 120
LOD     0       3       1       12      1 1 6 1 120 1
LIT     0       1       1       13      1 1 6 1 120 1 1
OPR     0       3       0       14      1 1 6 1 120 0
STO     0       3       120     15      1 1 6 0 120
LOD     0       3       0       16      1 1 6 0 120 0
JPC     0       20      120     20      1 1 6 0 120
STO     0       3       120     21      1 1 6 120
OPR     0       0       5       6       0 0 0 5
LOD     0       7       120     7       0 0 0 5 120
STO     0       3       120     8       0 0 0 120
OPR     0       0       0       0       0
➜  PCODE - 1
```

3.2.8. Exercício 07:

Main chama função Fib(4);

Extra: Fib(10);

Resolução:

```
➜ PCODE - 1 ./"p-code"
Please enter the instructions (Instruction END 0 0 to stopstack the input)
INT 0 4
LIT 0 4
STO 0 3
LOD 0 3
STO 0 7
CAL 0 9
LOD 0 7
STO 0 3
OPR 0 0
INT 0 6
LIT 0 1
LIT 0 0
STO 0 4
STO 0 5
LOD 0 3
JPC 0 27
LOD 0 5
LOD 0 4
LOD 0 5
OPR 0 2
STO 0 5
STO 0 4
LOD 0 3
LIT 0 1
OPR 0 3
STO 0 3
JMP 0 14
STO 0 3
OPR 0 0
END 0 0
```

```
STO     0       4       5       22      1 1 6 1 3 5
LOD     0       3       1       23      1 1 6 1 3 5 1
LIT     0       1       1       24      1 1 6 1 3 5 1 1
OPR     0       3       0       25      1 1 6 1 3 5 0
STO     0       3       5       26      1 1 6 0 3 5
JMP     0       14      5       14      1 1 6 0 3 5
LOD     0       3       0       15      1 1 6 0 3 5 0
JPC     0       27      5       27      1 1 6 0 3 5
STO     0       3       3       28      1 1 6 5 3
OPR     0       0       4       6       0 0 0 4
LOD     0       7       5       7       0 0 0 4 5
STO     0       3       5       8       0 0 0 5
OPR     0       0       0       0       0
➜ PCODE - 1
```

Fib(10):

```
LOD     0       3       0       15      1 1 6 0 55 89 0
JPC     0       27      89      27      1 1 6 0 55 89
STO     0       3       55      28      1 1 6 89 55
OPR     0       0       10      6       0 0 0 10
LOD     0       7       89      7       0 0 0 10 89
STO     0       3       89      8       0 0 0 89
OPR     0       0       0       0       0
➜ PCODE - 1
```

3.2.9. Exercício 08:

Main chama função Fat(5) Recursiva;

Resolução:

```
1   INT 0 4
2   LIT 0 5
3   STO 0 3
4   LOD 0 3
5   STO 0 7
6   CAL 0 8
7   LOD 0 7
8   OPR 0 0
9
10  INT 0 4
11  LOD 0 3
12  JPC 0 21 // SE FOR 0 PULA PRA
13  LOD 0 3
14  LIT 0 1
15  OPR 0 3 // O BUFFER ESTÁ COM n-1
16  STO 0 7 //  ENVIA n-1 COMO PARAMETRO
17  CAL 0 8 /   CHAMADA RECURSIVA
18  LOD 0 7 //  LOAD O VALOR RESULTADO DA RECURSÃO
19  LOD 0 3
20  OPR 0 4 // MULTIPLICA
21  STO 0 3 // ARMAZENA SER ACESSADO NO RETORNO
22  OPR 0 0
23
24  LIT 0 1
25  STO 0 3 // ARMAZENA 1 NO RETORNO CASO n == 0
26  OPR 0 0
```

```
LOD     0       3       4       19      5 5 17 4 6 4
OPR     0       4       24      20      5 5 17 4 24
STO     0       3       24      21      5 5 17 24
OPR     0       0       5       17      1 1 6 5
LOD     0       7       24      18      1 1 6 5 24
LOD     0       3       5       19      1 1 6 5 24 5
OPR     0       4       120     20      1 1 6 5 120
STO     0       3       120     21      1 1 6 120
OPR     0       0       5       6       0 0 0 5
LOD     0       7       120     7       0 0 0 5 120
STO     0       3       120     8       0 0 0 120
OPR     0       0       0       0       0
➜  PCODE - 1 _
```

3.2.10. Exercício 09:

Main chama função Fib(4) Recursiva;

Extra: Fib(10) Recursiva;

Resolução:

```
INT 0 4
LIT 0 4
STO 0 3
LOD 0 3
STO 0 7
CAL 0 9
LOD 0 7
STO 0 3
OPR 0 0
```

```
INT 0 4
LOD 0 3
JPC 0 30        // SE FOR 0 PULA PARA RETURN
LOD 0 3
LIT 0 1        //CASO FOR 1 ELE DIMINUI 1 E VERIFICA SE É 1 NA PROXIMA LINHA
OPR 0 3
JPC 0 30        // SE FOR 1 PULA PARA RETURN
LOD 0 3
LIT 0 1
OPR 0 3  // O BUFFER ESTÁ COM | n-1 |
STO 0 7     //ENVIA n-1 COMO PARAMETRO
CAL 0 9     //CHAMADA RECURSIVA
LOD 0 7     // LOAD O VALOR RESULTADO DA RECURSÃO
LOD 0 3
LIT 0 2
OPR 0 3  // O BUFFER ESTÁ COM  | fib(n-1) | n-2 |
STO 0 8     //ENVIA n-2 COMO PARAMETRO
CAL 0 9     //CHAMADA RECURSIVA
LOD 0 8     // LOAD O VALOR RESULTADO DA RECURSÃO
OPR 0 2     // SOMA O BUFFER ATUAL = | fib(n-1) | fib(n-2) |
STO 0 3     // ARMAZENA PARA O VALOR RETURN
OPR 0 0
END 0 0
```

```
STO       0           3           1           30          5 5 27 1
OPR       0           0           2           27          1 1 6 4 2
LOD       0           8           1           28          1 1 6 4 2 1
OPR       0           2           3           29          1 1 6 4 3
STO       0           3           3           30          1 1 6 3
OPR       0           0           4           6           0 0 0 4
LOD       0           7           3           7           0 0 0 4 3
STO       0           3           3           8           0 0 0 3
OPR       0           0           0           0           0
➜  PCODE - 1
```

3.2.11. Fib(10) Recursiva:

```
OPR       0           0           34          27          1 1 6 10 34
LOD       0           8           21          28          1 1 6 10 34 21
OPR       0           2           55          29          1 1 6 10 55
STO       0           3           55          30          1 1 6 55
OPR       0           0           10          6           0 0 0 10
LOD       0           7           55          7           0 0 0 10 55
STO       0           3           55          8           0 0 0 55
OPR       0           0           0           0           0
```

3.3. Projeto 02

O projeto 02 engloba desde a Análise do programa submetido pelo usuário, utilizando o Parser(Analisador Sintatico) juntamente com o Analisador Léxico para gerarem a ADA(Arvóre de Análise) até a construção da ASA(Arvóre Sintática Abstrata) a partir do ADA na etapa ADA2ASA, o qual é a saída utilizada como entrada para o Projeto 03 onde utilizaremos a ASA para transformarmos em NP* e por fim construirmos o GCI e o GCO.

3.3.1. Parser(Geração de ADA)

```
                 --> Table - Parsing <--
i     q                  .w                              Stack             δ        p
0     Q0    .g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1));}                     δ0       -
1     Q1    .g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1));}                   S δ2      p2
2     Q1    .g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1));}                  GM δ5      p5
3     Q1    .g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1));}        g(){C;r(E);}M δ26     -
4     Q1    g.(){w(1){(x-1);};r(x);}m(){i=1;r((1+1));}         (){C;r(E);}M δ28     -
5     Q1    g(.){w(1){(x-1);};r(x);}m(){i=1;r((1+1));}          ){C;r(E);}M δ29     -
6     Q1    g().{w(1){(x-1);};r(x);}m(){i=1;r((1+1));}           {C;r(E);}M δ30     -
7     Q1    g(){.w(1){(x-1);};r(x);}m(){i=1;r((1+1));}            C;r(E);}M δ22    p22
8     Q1    g(){.w(1){(x-1);};r(x);}m(){i=1;r((1+1));}     w(E){C;};r(E);}M δ48     -
9     Q1    g(){w.(1){(x-1);};r(x);}m(){i=1;r((1+1));}      (E){C;};r(E);}M δ28     -
10    Q1    g(){w(.1){(x-1);};r(x);}m(){i=1;r((1+1));}       E){C;};r(E);}M δ8      p8
11    Q1    g(){w(.1){(x-1);};r(x);}m(){i=1;r((1+1));}       1){C;};r(E);}M δ35     -
12    Q1    g(){w(1.){(x-1);};r(x);}m(){i=1;r((1+1));}        ){C;};r(E);}M δ29     -
13    Q1    g(){w(1).{(x-1);};r(x);}m(){i=1;r((1+1));}         {C;};r(E);}M δ30     -
14    Q1    g(){w(1){.(x-1);};r(x);}m(){i=1;r((1+1));}          C;};r(E);}M δ21    p21
15    Q1    g(){w(1){.(x-1);};r(x);}m(){i=1;r((1+1));}      (EXE);};r(E);}M δ28     -
16    Q1    g(){w(1){(.x-1);};r(x);}m(){i=1;r((1+1));}       EXE);};r(E);}M δ9      p9
17    Q1    g(){w(1){(.x-1);};r(x);}m(){i=1;r((1+1));}       xXE);};r(E);}M δ36     -
18    Q1    g(){w(1){(x.-1);};r(x);}m(){i=1;r((1+1));}        XE);};r(E);}M δ13    p13
19    Q1    g(){w(1){(x.-1);};r(x);}m(){i=1;r((1+1));}        -E);};r(E);}M δ39     -
20    Q1    g(){w(1){(x-.1);};r(x);}m(){i=1;r((1+1));}         E);};r(E);}M δ8      p8
21    Q1    g(){w(1){(x-.1);};r(x);}m(){i=1;r((1+1));}         1);};r(E);}M δ35     -
22    Q1    g(){w(1){(x-1.);};r(x);}m(){i=1;r((1+1));}          );};r(E);}M δ29     -
23    Q1    g(){w(1){(x-1).;};r(x);}m(){i=1;r((1+1));}           ;};r(E);}M δ33     -
24    Q1    g(){w(1){(x-1);.};r(x);}m(){i=1;r((1+1));}            };r(E);}M δ31     -
25    Q1    g(){w(1){(x-1);}.;r(x);}m(){i=1;r((1+1));}             ;r(E);}M δ33     -
26    Q1    g(){w(1){(x-1);};.r(x);}m(){i=1;r((1+1));}              r(E);}M δ32     -
27    Q1    g(){w(1){(x-1);};r.(x);}m(){i=1;r((1+1));}               (E);}M δ28     -
28    Q1    g(){w(1){(x-1);};r(.x);}m(){i=1;r((1+1));}                E);}M δ9      p9
29    Q1    g(){w(1){(x-1);};r(.x);}m(){i=1;r((1+1));}                x);}M δ36     -
30    Q1    g(){w(1){(x-1);};r(x.);}m(){i=1;r((1+1));}                 );}M δ29     -
31    Q1    g(){w(1){(x-1);};r(x).;}m(){i=1;r((1+1));}                  ;}M δ33     -
32    Q1    g(){w(1){(x-1);};r(x);.}m(){i=1;r((1+1));}                   }M δ31     -
33    Q1    g(){w(1){(x-1);};r(x);}.m(){i=1;r((1+1));}                   M δ6     p6
34    Q1    g(){w(1){(x-1);};r(x);}.m(){i=1;r((1+1));}        m(){C;r(E);} δ27     -
35    Q1    g(){w(1){(x-1);};r(x);}m.(){i=1;r((1+1));}         (){C;r(E);} δ28     -
36    Q1    g(){w(1){(x-1);};r(x);}m(.){i=1;r((1+1));}          ){C;r(E);} δ29     -
37    Q1    g(){w(1){(x-1);};r(x);}m().{i=1;r((1+1));}           {C;r(E);} δ30     -
38    Q1    g(){w(1){(x-1);};r(x);}m(){.i=1;r((1+1));}            C;r(E);} δ17   p17
39    Q1    g(){w(1){(x-1);};r(x);}m(){.i=1;r((1+1));}          i=E;r(E);} δ43     -
40    Q1    g(){w(1){(x-1);};r(x);}m(){i.=1;r((1+1));}           =E;r(E);} δ47     -
41    Q1    g(){w(1){(x-1);};r(x);}m(){i=.1;r((1+1));}            E;r(E);} δ8     p8
42    Q1    g(){w(1){(x-1);};r(x);}m(){i=.1;r((1+1));}            1;r(E);} δ35     -
43    Q1    g(){w(1){(x-1);};r(x);}m(){i=1.;r((1+1));}             ;r(E);} δ33     -
44    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;.r((1+1));}              r(E);} δ32     -
45    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r.((1+1));}               (E);} δ28     -
46    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r(.(1+1));}                E);} δ11   p11
47    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r(.(1+1));}            (EXE));} δ28     -
48    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((.1+1));}             EXE));} δ8     p8
49    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((.1+1));}             1XE));} δ35     -
50    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((1.+1));}              XE));} δ12   p12
51    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((1.+1));}              +E));} δ38     -
52    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((1+.1));}               E));} δ8     p8
53    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((1+.1));}               1));} δ35     -
54    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1.));}                ));} δ29     -
55    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1).);}                 );} δ29     -
56    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1)).;}                  ;} δ33     -
57    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1));.}                   } δ31     -
58    Q1    g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1));}.                     -        -

Accepted : g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1));}
```

```
--------------------------------------
              --> Tree - Parsing <--
[0|0| S ]
[1|1| G ]
[2|2| M ]
[3|13| g ]
[4|14| ( ]
[5|15| ) ]
[6|16| { ]
[7|17| C ]
[8|18| ; ]
[9|19| r ]
[10|20| ( ]
[11|21| E ]
[12|22| ) ]
[13|23| ; ]
[14|24| } ]
[15|205| w ]
[16|206| ( ]
[17|207| E ]
[18|208| ) ]
[19|209| { ]
[20|210| C ]
[21|211| ; ]
[22|212| } ]
[23|2485| 1 ]
[24|2521| ( ]
[25|2522| E ]
[26|2523| X ]
[27|2524| E ]
[28|2525| ) ]
[29|30265| x ]
[30|30277| - ]
[31|30289| 1 ]
[32|253| x ]
[33|25| m ]
```

```
[34|26| ( ]
[35|27| ) ]
[36|28| { ]
[37|29| C ]
[38|30| ; ]
[39|31| r ]
[40|32| ( ]
[41|33| E ]
[42|34| ) ]
[43|35| ; ]
[44|36| } ]
[45|349| i ]
[46|350| = ]
[47|351| E ]
[48|4213| 1 ]
[49|397| ( ]
[50|398| E ]
[51|399| X ]
[52|400| E ]
[53|401| ) ]
[54|4777| 1 ]
[55|4789| + ]
[56|4801| 1 ]
```

-Função de leitura do arquivo .txt

```
// ReadFileFunctions
int readINPUT(FILE *file_IN); // Retorna quantidade de linhas de entradas
```

-Função do autómato de pilha do Parser

```
//————————< AUTOMATO >————————
void automato(char *in);
```

-Funções de Manipulação da Pilha

```
// ————————<StackFunctions>————————
void inversePush(char *line);
void push(char value);
void pop();
```

-Funções para geração da Arvóre

```
//————————< TREE >————————
void start_tree();
void parsing_tree(Ntree * tree, int curr_node);
```

-Funções de Exibição

```
// ————————<PrintFunctions>————————
void printTableHead();
void printTreeHead();
void printLineTable(char *q, char *w, char *d, char *p, int token);
```

-Função de Produções da arvóre

```
void get_productions(Ntree * tree, int curr_node);
```

```c
void automato(char *in)
{
    int token = 0;
    char tree[100] = {'\0'}; // Foi obrigatorio o uso de vetor ao invés de struct dinamica
    int count_tree = 0;
    int visited[100] = {0};
    int aux_f[100] = {0};
    int aux_ind = 0;
    int aux;
    goto Q0;

Q0:
    tree[count_tree] = 'S';
    visited[count_tree] = TRUE;

    printLineTable("Q0", in, "δ0", "-", 0);
    push('S'); // stack[0] = 'S'
    goto Q1;
```

```c
    }
    else if ((in[token] == 'w') && (STACK[STACK_TOP] == 'w'))
    { // token_accepted = w     //t48
        printLineTable("Q1", in, "δ48", "-", token);
        pop();
        goto Q1get;
    }
    else if ((in[token] == 'f') && (STACK[STACK_TOP] == 'f'))
    { // token_accepted = f      //t49
        printLineTable("Q1", in, "δ49", "-", token);
        pop();
        goto Q1get;
    }
    else if ((in[token] == 'o') && (STACK[STACK_TOP] == 'o'))
    { // token_accepted = o     //t50
        printLineTable("Q1", in, "δ50", "-", token);
        pop();
        goto Q1get;
    }
    else if (in[token] == ' ')
    {
        goto Q1get;
    }
    else
    {
        goto ERROR;
    }
ACCEPT:
    printf("\nAccepted : %s\n", in);
    printf("————————————————————————————————————————————————————————————————————————————————————————————————————————————\n");
    return;

ERROR:
    printf("\nRejected : %s\n", in);
    printf("————————————————————————————————————————————————————————————————————————————————————————————————————————————\n");
    return;
}
```

```c
    // P4: N → n(){ C; r(E); }
    // P5: G → g(){ C; r(E); }
    // P6: M → m() { C; r(E); }
    else if(productions[prod] ≥ 4 && productions[prod] ≤ 6) {
        for (int i = 0; i < 12; i++){ tree[end_array+i].visited = TRUE; }

        if (productions[prod] == 4) {
            if (tree[curr_node].token ≠ 'N')
                return;
            tree[end_array].token = 'n';
        }
        else if (productions[prod] == 5) {
            if (tree[curr_node].token ≠ 'G')
                return;
            tree[end_array].token = 'g';
        }
        else {
            if (tree[curr_node].token ≠ 'M')
                return;
            tree[end_array].token = 'm';
        }
        tree[end_array+1].token = '(';
        tree[end_array+2].token = ')';
        tree[end_array+3].token = '{';
        tree[end_array+4].token = 'C';
        tree[end_array+5].token = ';';
        tree[end_array+6].token = 'r';
        tree[end_array+7].token = '(';
        tree[end_array+8].token = 'E';
        tree[end_array+9].token = ')';
        tree[end_array+10].token = ';';
        tree[end_array+11].token = '}';
        tree[end_array+4].visited = FALSE;
        tree[end_array+8].visited = FALSE;
        tree[end_array].children = 12;
    }
```

3.3.2. Parser(Geração de ASA)

```
                 --> Table - Parsing <--
 i     q                 .w                          Stack                    δ        p
 0     Q0    .m(){i=1;r((1+1));}                          δ0       -
 1     Q1    .m(){i=1;r((1+1));}                        S δ1      p1
 2     Q1    .m(){i=1;r((1+1));}                        M δ6      p6
 3     Q1    .m(){i=1;r((1+1));}           m(){C;r(E);} δ27       -
 4     Q1    m.(){i=1;r((1+1));}            (){C;r(E);} δ28       -
 5     Q1    m(.){i=1;r((1+1));}             ){C;r(E);} δ29       -
 6     Q1    m().{i=1;r((1+1));}              {C;r(E);} δ30       -
 7     Q1    m(){.i=1;r((1+1));}               C;r(E);} δ17     p17
 8     Q1    m(){.i=1;r((1+1));}             i=E;r(E);} δ43       -
 9     Q1    m(){i.=1;r((1+1));}              =E;r(E);} δ47       -
 10    Q1    m(){i=.1;r((1+1));}               E;r(E);} δ8       p8
 11    Q1    m(){i=.1;r((1+1));}               1;r(E);} δ35       -
 12    Q1    m(){i=1.;r((1+1));}                ;r(E);} δ33       -
 13    Q1    m(){i=1;.r((1+1));}                 r(E);} δ32       -
 14    Q1    m(){i=1;r.((1+1));}                  (E);} δ28       -
 15    Q1    m(){i=1;r(.(1+1));}                   E);} δ11     p11
 16    Q1    m(){i=1;r(.(1+1));}               (EXE));} δ28       -
 17    Q1    m(){i=1;r((.1+1));}                EXE));} δ8       p8
 18    Q1    m(){i=1;r((.1+1));}                1XE));} δ35       -
 19    Q1    m(){i=1;r((1.+1));}                 XE));} δ12     p12
 20    Q1    m(){i=1;r((1.+1));}                 +E));} δ38       -
 21    Q1    m(){i=1;r((1+.1));}                  E));} δ8       p8
 22    Q1    m(){i=1;r((1+.1));}                  1));} δ35       -
 23    Q1    m(){i=1;r((1+1.));}                   ));} δ29       -
 24    Q1    m(){i=1;r((1+1).);}                    );} δ29       -
 25    Q1    m(){i=1;r((1+1)).;}                     ;} δ33       -
 26    Q1    m(){i=1;r((1+1));.}                      } δ31       -
 27    Q1    m(){i=1;r((1+1));}.                        -          -

 Accepted : m(){i=1;r((1+1));}
```

```
-----------------------------------------------------------------------------------------------
                --> Tree - Parsing <--
0, S
1, M
5, C
6, E
21, E
25, E
26, X
27, E
85, 1
101, 1
105, +
109, 1
```

- Função para leitura do arquivo de Input

```c
// ReadFileFunctions
int readINPUT(FILE *file_IN); // Retorna quantidade de linhas de entradas
```

-Função do autómato que recebe a string do input

```c
//────────< AUTOMATO >────────
void automato(char *in);
```

-Funções de Manipulação da pilha

```
// ————————〈StackFunctions〉————————
void inversePush(char *line);
void push(char value);
void pop();
```

-Função da Arvóre

```
//————————〈 TREE 〉————————
void parsing_tree();
```

-Funções de exibição

```
// ————————〈PrintFunctions〉————————
void printTableHead();
void printTreeHead();
void printLineTable(char *q, char *w, char *d, char *p, int token);
```

-Definição:

```
#define TRUE 1
#define FALSE -1
```

-Algumas produções:

```
// P1: S → M
if(productions[i] == 1){
    if (tree[top] == 'S' && tree[tam_max_deriv*top+1] == ' '){
        tree[tam_max_deriv*top + 1] = 'M';
        tree[tam_max_deriv*top + 2] = ' ';
        tree[tam_max_deriv*top + 3] = ' ';
        tree[tam_max_deriv*top + 4] = ' ';

        visited[tam_max_deriv*top+2] = TRUE;
        visited[tam_max_deriv*top+3] = TRUE;
        visited[tam_max_deriv*top+4] = TRUE;
        i++;
    }
}
```

```
// P21: C → (EXE)
else if(productions[i] == 21) {
    if (tree[top] == 'C' && tree[tam_max_deriv*top+1] == ' '){
        tree[tam_max_deriv*top + 1] = 'E';
        tree[tam_max_deriv*top + 2] = 'X';
        tree[tam_max_deriv*top + 3] = 'E';
        tree[tam_max_deriv*top + 4] = ' ';

        visited[tam_max_deriv*top+4] = TRUE;
        i++;
    }
}
```

```
// P23: C → f(E){ C; }
else if(productions[i] == 23) {
    if (tree[top] == 'C' && tree[tam_max_deriv*top+1] == ' '){
        tree[tam_max_deriv*top + 1] = 'E';
        tree[tam_max_deriv*top + 2] = 'C';
        tree[tam_max_deriv*top + 3] = ' ';
        tree[tam_max_deriv*top + 4] = ' ';

        visited[tam_max_deriv*top+3] = TRUE;
        visited[tam_max_deriv*top+4] = TRUE;
        i++;
    }
}
```

3.3.3. LEX

```
lines to input: 2


------<Parser>--------
g(){w(1){(x-1);};r(x);}m(){i=1;r((1+1));}
------<Tree Parser>--------
-----------------
0 >S<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
-----------------
1 >S<  >G<  >M<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
-----------------
2 >S<  >G<  >M<  > <  > <  >C<  >E<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
-----------------
3 >S<  >G<  >M<  > <  > <  >C<  >E<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
-----------------
4 >S<  >G<  >M<  > <  > <  >C<  >E<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
-----------------
4 >S<  >G<  >M<  > <  > <  >C<  >E<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
-----------------
4 >S<  >G<  >M<  > <  > <  >C<  >E<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
-----------------
4 >S<  >G<  >M<  > <  > <  >C<  >E<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
-----------------
5 >S<  >G<  >M<  > <  > <  >C<  >E<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
-----------------
6 >S<  >G<  >M<  > <  > <  >C<  >E<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
-----------------
6 >S<  >G<  >M<  > <  > <  >C<  >E<  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <  > <
```

```
15 >S<  >G<  >M<  > <  > <  >C<
-----------------
16 >S<  >G<  >M<  > <  > <  >C<
0 -> [S]
1 -> [G]
2 -> [M]
5 -> [C]
6 -> [E]
21 -> [E]
22 -> [C]
25 -> [x]
85 -> [1]
89 -> [E]
90 -> [X]
91 -> [E]
357 -> [x]
361 -> [-]
365 -> [1]


------<Parser>--------
m(){i=1;r((1+1));}
```

-Função de Leitura do arquivo de input

```
int readINPUT(FILE *file_IN);
```

-Função Léxica

```
void lex();
```

-Funções de Verificação da sintaxe para um trecho do código que possui valor semantico

```
int S();
int M();
int G();
int N();
int E();
int X();
int C();
```

-Função de error

```
void err(int error, char *on_value, char on_function);
```

-Funções de geração da arvore

```
void start_tree();
void parsing_tree(char *tree, int *visited, int curr_node);
void put_production(char *tree, int *visited, int curr_node);
```

```
int main()
{
    // Abre um arquivo TEXTO para LEITURA
    file_IN = fopen("input_file.txt", "rt");
    if (file_IN == NULL) // Se houver erro na abertura…
    qtd_entries = readINPUT(file_IN);
    printf("lines to input: %d\n\n", qtd_entries);

    for (line = 0; line < qtd_entries; line++)
    {
        for (int i = 0; i<100; i++)
            productions[i] = -1;
        index = 0;
        prod = 0;
        p_count = 0;
        error_founded = FALSE;
        printf("\n------<Parser>-------\n");
        S();
        printf("\n------<Tree Parser>-------\n");
        start_tree();
        //Se depois da analise, o error continuar FALSE, então printa que foi sucesso
        if(error_founded == FALSE)
            printf(" ");
    }
    fclose(file_IN);
    return 1;
}
```

-Algumas Produções:

```
// S→ P1: M | P2: G M | P3: F M G
int S()
{
    lex();
    if (token == 'm')
    {
        productions[p_count++] = 1;
        M();
    }
    else if (token == 'g')
    {
        productions[p_count++] = 2;
        G();
        M();
    }
    else if (token == 'f')
    {
        productions[p_count++] = 3;
        N();
        G();
        M();
    }
    else
    {
        err(1, "m | g | f", 'S');
    }
    return 0;
}
// P4: M → m(){ C; r(E); }
int M()⋯
// P5: G → g(){ C; r(E); }
int G()⋯
// P6: N → n(){ C; r(E); }
```

```
// C → P16: h=E | P17: i=E | P18: j=E | P19: k=E | P20: z=E | P21: (EXE) | P22: w(E){ C; } | P23: f(E){ C; } | P24: o(E; E; E){ C; }
int C()
{

    if (token == 'h'){
        productions[p_count++] = 16;
        lex();
        if (token == '=') {
            lex();
            E();
        }
        else
            err(1, "h", 'C');
    }
    else if (token == 'i'){
        productions[p_count++] = 17;
        lex();
        if (token == '=') {
            lex();
            E();
            }
        else
            err(1, "i", 'C');
    }
    else if (token == 'j'){
        productions[p_count++] = 18;
        lex();
        if (token == '=') {
            lex();
            E(); }
        else
            err(1, "j", 'C');
    }
```

```
// X → P12: + | P13: - | P14: * | P15: /
int X()
{
    if (token == '+'){
        productions[p_count++] = 12;
        lex();
    }
    else if (token == '-'){
        productions[p_count++] = 13;
        lex();
        }
    else if (token == '*'){
        productions[p_count++] = 14;
        lex();
    }
    else if (token == '/'){
        productions[p_count++] = 15;
        lex();
    }
    else err(1, "+⊢⊣*|/", 'X');
    return 0;
}
```

```
int E()
{
    if (token == '0'){
        productions[p_count++] = 7;
        lex();
    }
    else if (token == '1'){
        productions[p_count++] = 8;
        lex();
    }
    else if (token == 'x'){
        productions[p_count++] = 9;
        lex();
    }
    else if (token == 'y'){
        productions[p_count++] = 10;
        lex();
    }
    else if (token == '('){
        productions[p_count++] = 11;
        lex();
        E();
        X();
        E();
        if (token == ')')
            lex();
        else
```

-Modularização de ERROR

```
                else
                {
                    err(6, "O→;", 'C');
                }
            }
            else
            {
                err(7, "O→(", 'C');
            }
        }
        else
        {
            err(3, "h | i | j | k | z | ( | w | f | o", 'C');
        }
        return 0;
    }
```

```
void err(int error, char *on_value, char on_function)
{
    error_founded = TRUE;
    printf("\n————————————————————————————————————————————————————————————\n");
    printf("\tErro: [%d] \t Função: '%c' \n", error, on_function);
    printf("\tEra esperado: '%s \t foi lido: '%c'\n", on_value, token);
    printf("————————————————————————————————————————————————————————————\n");
    exit(1);
}
```

-exemplo de um erro onde não é colocado ponto e virgula no final do while. Era esperado um ; no final do W

```
------<Parser>--------
g(){w(1){(x-1)}
-----------------------------------------------------------------
        Erro: [2]        Função: 'C'
        Era esperado: 'W->;      foi lido: '}'
-----------------------------------------------------------------
```

3.3.4. ADA2ASA

```
[21|211| ; ]
[22|212| } ]
[23|2485| 1 ]
[24|2521| ( ]
[25|2522| E ]
[26|2523| X ]
[27|2524| E ]
[28|2525| ) ]
[29|30265| x ]
[30|30277| - ]
[31|30289| 1 ]
[32|253| x ]
[33|25| m ]
[34|26| ( ]
[35|27| ) ]
[36|28| { ]
[37|29| C ]
[38|30| ; ]
[39|31| r ]
[40|32| ( ]
[41|33| E ]
[42|34| ) ]
[43|35| ; ]
[44|36| } ]
[45|349| i ]
[46|350| = ]
[47|351| E ]
[48|4213| 1 ]
[49|397| ( ]
[50|398| E ]
[51|399| X ]
[52|400| E ]
[53|401| ) ]
[54|4777| 1 ]
[55|4789| + ]
```

```
0 -> [S]
1 -> [G]
2 -> [M]
5 -> [C]
6 -> [E]
21 -> [E]
22 -> [C]
25 -> [x]
85 -> [1]
89 -> [E]
90 -> [X]
91 -> [E]
357 -> [x]
361 -> [-]
365 -> [1]
```

-Função de Leitura do Arquivo de input

```
// ReadFileFunctions
int readINPUT(FILE *file_IN); // Retorna quantidade de linhas de entradas
```

-Função do Automato

```
//————————< AUTOMATO >————————
void automato(char *in);
```

-Funções de exibição

```
// ————————<PrintFunctions>————————
void printTableHead();
void printTreeHead();
void printLineTable(char *q, char *w, char *d, char *p, int token);
```

-Funções De analise lexica e de error

```
int S();
int M();
int G();
int N();
int E();
int X();
int C();
void lex();
void err(int error, char *on_value, char on_function);
```

-Funções da ADA

```
//————————< SYNTAX TREE >————————
void start_tree();
void parsing_tree(Ntree *tree, int curr_node);
void get_production(Ntree *tree, int curr_node);
```

-Funções da para geração da ASA

```
//————————< ABSTRACT TREE  >————————
void start_abstract_tree(Nsyntax *syntax_tree);
void abstract_tree(Nsyntax *syntax_tree, Ntree *tree, int curr_node);
void get_token(Nsyntax *syntax_tree, Ntree *tree, int curr_node);
```

```
// P24: C → o(E;E;E){ C; }
else if(productions[prod] == 24) {
    if (tree[curr_node].token ≠ 'C')
        return;

    for (int i = 0; i < 12; i++)
        tree[end_tree+i].visited = TRUE;
    tree[end_tree].token = 'o';
    tree[end_tree+1].token = '(';
    tree[end_tree+2].token = 'E';
    tree[end_tree+3].token = ';';
    tree[end_tree+4].token = 'E';
    tree[end_tree+5].token = ';';
    tree[end_tree+6].token = 'E';
    tree[end_tree+7].token = ')';
    tree[end_tree+8].token = '{';
    tree[end_tree+9].token = 'C';
    tree[end_tree+10].token = ';';
    tree[end_tree+11].token = '}';
    tree[end_tree+2].visited = FALSE;
    tree[end_tree+4].visited = FALSE;
    tree[end_tree+6].visited = FALSE;
    tree[end_tree+9].visited = FALSE;
    tree[end_tree].children = 12;
}
```

```
// P23: C → f(E){ C; }
else if(productions[prod] == 22 || productions[prod] == 23) {
    if (tree[curr_node].token ≠ 'C')
        return;

    for (int i = 0; i < 8; i++)
        tree[end_tree+i].visited = TRUE;
    if (productions[prod] == 22)
        tree[end_tree].token = 'w';
    else
        tree[end_tree].token = 'f';
    tree[end_tree+1].token = '(';
    tree[end_tree+2].token = 'E';
    tree[end_tree+3].token = ')';
    tree[end_tree+4].token = '{';
    tree[end_tree+5].token = 'C';
    tree[end_tree+6].token = ';';
    tree[end_tree+7].token = '}';
    tree[end_tree+2].visited = FALSE;
    tree[end_tree+5].visited = FALSE;
    tree[end_tree].children = 8;
}
// P24: C → o(E;E;E){ C; }
```

```
void start_abstract_tree(Nsyntax *syntax_tree){
    Ntree tree[end_tree];
    for (int i = 0; i < end_tree; i++) {
        tree[i].token = ' ';
        tree[i].visited = FALSE;
        tree[i].index_tree = 0;
        tree[i].children = 1;
        tree[i].hash = 0;
    }
    end_abs_tree = 1;
    tree[0].token = 'S';

    abstract_tree(syntax_tree, tree, 0);
```

```
tree[end_abs_tree].token = syntax_tree[valores[1]].token;
tree[end_abs_tree].hash = syntax_tree[valores[1]].index_tree;
tree[end_abs_tree].index_tree = 2*tree[curr_node].index_tree+1;
index_w = tree[end_abs_tree++].index_tree;
tree[end_abs_tree].token = syntax_tree[valores[1]].token;
tree[end_abs_tree].hash = syntax_tree[valores[1]].index_tree;
tree[end_abs_tree++].index_tree = 2*index_w+1;
tree[end_abs_tree].token = syntax_tree[valores[0]].token;
tree[end_abs_tree].hash = syntax_tree[valores[0]].index_tree;
tree[end_abs_tree].index_tree = 2*tree[curr_node].index_tree+2;
index_w = tree[end_abs_tree++].index_tree;
tree[end_abs_tree].token = syntax_tree[valores[0]].token;
tree[end_abs_tree].hash = syntax_tree[valores[0]].index_tree;
tree[end_abs_tree++].index_tree = 2*index_w+1;
tree[end_abs_tree].token = syntax_tree[valores[3]].token;
tree[end_abs_tree].hash = syntax_tree[valores[3]].index_tree;
tree[end_abs_tree].index_tree = 2*index_w+2;
tree[end_abs_tree].children = 2;
index_w = tree[end_abs_tree++].index_tree;
tree[end_abs_tree].token = syntax_tree[valores[3]].token;
tree[end_abs_tree].hash = syntax_tree[valores[3]].index_tree;
tree[end_abs_tree++].index_tree = 2*index_w+1;
tree[end_abs_tree].token = syntax_tree[valores[2]].token;
tree[end_abs_tree].hash = syntax_tree[valores[2]].index_tree;
tree[end_abs_tree++].index_tree = 2*index_w+2;
```

```
tree[curr_node].token = syntax_tree[tokens].token;
tree[curr_node].hash = syntax_tree[tokens].index_tree;
tree[curr_node].children = 2;

tree[end_abs_tree].token = syntax_tree[valores[0]].token;
tree[end_abs_tree].hash = syntax_tree[valores[0]].index_tree;
tree[end_abs_tree++].index_tree = 2*tree[curr_node].index_tree+1;

tree[end_abs_tree].token = syntax_tree[valores[1]].token;
tree[end_abs_tree].hash = syntax_tree[valores[1]].index_tree;
tree[end_abs_tree++].index_tree = 2*tree[curr_node].index_tree+2;
```

3.4. Projeto 03

O projeto 03 não foi finalizado para ser postado no relatório.

3.4.1 Geração de nova linguagem e ASA

Foi iniciado mas não finalizado.

3.4.2 ASA2NP*

3.4.3. NP2GCI

3.4.4. GCI2GCO

4. Casos de Uso

Nesta seção serão detalhados os casos de uso para a implementação do compilador. Será mostrado a listagem de casos de uso sucedidos e mal sucedidos.

4.1. Caso de Uso Programa bem sucedida

- Usuário entra com o código fonte.
- O compilador repassa o código fonte para o parser.
- O parser envia o código fonte para o scanner.
- O scanner realiza a análise de linhas de caracteres e gera a tabela de símbolos.
- O parser utiliza a tabela de símbolos para construção da árvore de análise sintática.
- A árvore de análise sintática então é convertida em árvore de análise sintática abstrata.

4.2. Casos de Uso de erros sintáticos

3.3.1. Cadeia consumida e pilha não vazia

- Parser consome totalmente a cadeia
- Parser verifica que a pilha não está vazia
- Ainda faltam ser incluídos símbolos na cadeia de entrada para que esta seja aceita
- É gerado um erro

3.3.2. Cadeia não consumida e pilha vazia

- Parser verifica que a pilha está vazia
- Parser verifica que a cadeia não foi totalmente consumida
- Prefixo da cadeia de entrada é o suficiente para que esta seja aceita
- É gerado um erro

3.3.3. Token recebido diferente do esperado pelo parser

- Parser verifica que lookahead é diferente do topo da pilha
- Token recebido, o qual fez tal produção estar no topo da pilha, não é o esperado pelo parser.
- É gerado um erro

Referências

1. **P-Code<https://en.wikipedia.org/wiki/P-code_machine>**
2. **Sebesta<https://trendspdf.prograd.uesc.br/MaterialApoio/Diario/Aula/2002681954/Sebesta_Cap10pp.pdf>**
3. **GITHUB Material de Acesso <https://github.com/KiritoKi/Compilers>**