Versão 2.0

LUCAS RODOLFO CELESTINO DE FARIAS

Mestrando em Ciência da Computação - UFPE

Ex-monitor de Arq. II (2015.2 e 2016.1) e Compiladores (2016.2)

Recife, 2018

SPOILER ALERT!!

O uso deste documento pode tirar a emoção acompanhada da adrenalina do processo da construção do compilador, isto é, aqueles que o usaram, construíram seus compiladores do início ao fim em 8±3 horas!!*

Assim, se deseja estar tão desapontado quanto o Saitama ao derrotar os vilões, desejo-lhe uma boa leitura.**

*** Eu orientei mais de 10 projetos de compiladores do início ao fim, em diversas linguagens de programação, nenhum deles foram acusados como cópias.**

**** Evitem o plágio, somente servirá para facilitar a atribuição de sua nota (zero), seja criativo e programe do seu jeito.**

Sumário

1 Como fazer um SCANNER para iniciantes

1.1 Cabeçalho

```
/////// LEGENDA DO SCANNER
//VERDE Token lido pelo SCANNER
//AMARELO erro detectado pelo SCANNER

// Bibliotecas usadas
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

// Estruturas definidas
typedef enum token {
      // PALAVRAS RESERVADAS
      MAIN, VALOR_INT, VALOR_FLOAT, VALOR_CHAR, IF, ELSE, WHILE, DO, FOR, ID, TIPO_CHAR, TIPO_FLOAT,
TIPO_INT,
      //OPERADOR RELACIONAL
      MAIOR, MAIOR_IGUAL, MENOR, MENOR_IGUAL, IGUALDADE,
      // OPERAÇÕES ARITMÉTICAS
      SOMA, SUBTRACAO, MULT, DIVISAO, DIFERENCA, ATRIBUIR,
      // MARCADORES
      ABRE_PARENTESES, FECHA_PARENTESES, ABRE_CHAVES, FECHA_CHAVES, VIRGULA, PONTO_E_VIRGULA,
FIM_DE_ARQUIVO
} TOKEN;

typedef struct no {
      char lexema [50];
      TOKEN  token;
} TNo;

// Variáveis Globais (esta parte é muito importante, confie em mim vai lhe ajudar a simplificar
// seu compilador, Ana uma vez disse: “não usem variáveis globais”. Ela disse isto porque os alunos
// eram muito verdes para SABER QUANDO USAR, essa hora chegou, use)
int linha = 1, coluna = 0;

// Cabeçalho dos procedimentos e funções utilizados
// Pocedimentos auxiliares para todo o programa
void exibirErro (FILE *source, char msg[], char lexema[]);
// Cabeçalho do SCANNER
TNo SCANNER (FILE * source);
void preencherLexema (FILE * arq, char* ch, char lexema[], int *tamLexema);
// Cabeçalho para chamada do compilador
void PROGRAMA (FILE *source);
```

1.2 Função main e procedimentos auxiliares

```
// A função MAIN não será alterada em todo o nosso projeto
int main (int argc, char *argv[])  // passagem via argumento da linha de COMANDO do arquivo, argc = qtd de argumentos a passar e argv= armazena o nome do PROGRAMA que foi chamado no prompt
{
	FILE *source;
	// source = fopen("teste.txt", "a+t");  // abertura mais fácil de testar
	source = fopen(argv[1], "a+t"); // abertura aprovado por Silvio
	PROGRAMA(source); // executa o compilador no arquivo source
	fclose(source); // Fecha o arquivo source
	return 0;
}

void PROGRAMA (FILE *source) {
	TNo reader;
	do {
		reader = SCANNER (source);
 } while (reader.token != FIM_DE_ARQUIVO);
}

void exibirErro (FILE *source, char msg[], char lexema[]){/ / Proced. para exibir erro elegantemente
	printf("erro na linha %i coluna %i ultimo lexema lido %s : %s\n", linha, coluna, lexema, msg);
	fclose (source);
	exit (0);
}

void preencherLexema (FILE * arq, char* ch, char lexema[], int *tamLexema){ // Proced. aux. do SCANNER
	lexema[*tamLexema] = *ch;
	(*tamLexema)++;
	lexema[*tamLexema] = '\0';
	*ch=getc(arq);
	coluna++;
}
```

1.3 Exemplo “Hello World” da função SCANNER

```
// Irei fazer o SCANNER em 2 etapas
// Primeira etapa: o SCANNER é capaz de ler apenas caracteres em branco e detectar
// o final do arquivo, qualquer coisa diferente disso é ERRO de CARACTER INVALIDO

TNo SCANNER (FILE * source) {
      static char ch = ' '; // variável para percorrer o arquivo
      int tamLexema; // tamanho do lexema retornado
      TNo novo; // estrutura retornada pelo SCANNER contém o lexema e token

      novo.lexema[0] = '0';
      tamLexema = 0;

      // Para ignorar espaços em branco
      while(isspace(ch)) {
            switch (ch){
                  case ' ': coluna++; break;
                  case '\t': coluna = coluna +4; break;
                  case '\n': coluna=1; linha++; break;
            }
            ch=getc(source);
      }
      // FINAL DE ARQUIVO
      if (feof(source)) {
            novo.token = FIM_DE_ARQUIVO;
            return novo;
      }
      // ERRO CARACTER INVALIDO
      else { // Tudo que não estiver na gramática acima é caracter inválido
            preencherLexema (source, &ch, novo.lexema, &tamLexema);
            exibirErro (source,"caracter invalido", novo.lexema);
      }
}
```

1.4 Função SCANNER

```
// a primeira etapa apresenta a ideia da execução de um scanner simples
// SCANNER etapa 2: Todas as funcionalidades estão devidamente implementadas

TNo SCANNER (FILE * source) {
	static char ch = ' '; // variável para percorrer o arquivo
	int tamLexema; // tamanho do lexema retornado
	TNo novo; // estrutura retornada pelo SCANNER contém o lexema e token

	INICIO:
	novo.lexema[0] = '0';
	tamLexema = 0;

	// Para ignorar espaços em branco
	while(isspace(ch)) {
		switch (ch){
			case ' ': coluna++; break;
			case '\t': coluna = coluna +4; break;
			case '\n': coluna=1; linha++; break;
		}
		ch=getc(source);
	}
	// FINAL DE ARQUIVO
	if (feof(source)) {
		novo.token = FIM_DE_ARQUIVO;
		return novo;
	}
	// OPERADORES ARITMETICOS (DIFICULDADE) FÁCEIS E ESPECIAIS
	else if (ch == '+' || ch == '-' || ch == '*' ||
		ch == '(' || ch == ')' || ch == '{'|| ch == '}' || ch == ',' || ch == ';') {
		switch (ch) {
			case '+': novo.token = SOMA; break;
			case '-': novo.token = SUBTRACAO; break;
			case '*': novo.token = MULT; break;
			case '(': novo.token = ABRE_PARENTESES; break;
			case ')': novo.token = FECHA_PARENTESES; break;
			case '{': novo.token = ABRE_CHAVES; break;
			case '}': novo.token = FECHA_CHAVES; break;
			case ',': novo.token = VIRGULA; break;
			case ';': novo.token = PONTO_E_VIRGULA; break;
		}
		preencherLexema (source, &ch, novo.lexema, &tamLexema);
		return novo;
	}
	// OPERADORES ARITMETICOS (DIFICULDADE) MEDIANOS E OPERADORES RELACIONAIS
	else if (ch == '<' || ch == '>' || ch == '=' || ch == '!') {
		switch (ch) {
			case '<': novo.token = MENOR;
					preencherLexema (source, &ch, novo.lexema, &tamLexema);
					if(ch == '='){
						preencherLexema (source, &ch, novo.lexema, &tamLexema);
						novo.token = MENOR_IGUAL;
					}
					break;
			case '>': novo.token = MAIOR;
					preencherLexema (source, &ch, novo.lexema, &tamLexema);
					if(ch == '='){
						preencherLexema (source, &ch, novo.lexema, &tamLexema);
						novo.token = MAIOR_IGUAL;
					}
					break;
```

```
            case '=': novo.token = ATRIBUIR;
                        preencherLexema (source, &ch, novo.lexema, &tamLexema);
                        if(ch == '='){
                              preencherLexema (source, &ch, novo.lexema, &tamLexema);
                              novo.token = IGUALDADE;
                        }
                        break;
            case '!': preencherLexema (source, &ch, novo.lexema, &tamLexema);
                        if(ch != '='){ // ERRO EXCLAMAÇÃO SOZINHA
                              preencherLexema (source, &ch, novo.lexema, &tamLexema);
                              exibirErro (source,"exclamacao sozinha", novo.lexema);
                        }
                        preencherLexema (source, &ch, novo.lexema, &tamLexema);
                        novo.token = DIFERENCA;
                        break;
      }
      return novo;
}
// PALAVRAS RESERVADAS
else if(ch == '_' || isalpha(ch)){
      preencherLexema (source, &ch, novo.lexema, &tamLexema);
      while(ch == '_' || isalnum(ch))  preencherLexema (source, &ch, novo.lexema, &tamLexema);

      if (strcmp ("main", novo.lexema) == 0) novo.token = MAIN;
      else if (strcmp ("int", novo.lexema) == 0) novo.token = TIPO_INT;
      else if (strcmp ("float", novo.lexema) == 0)     novo.token = TIPO_FLOAT;
      else if (strcmp ("char", novo.lexema) == 0) novo.token = TIPO_CHAR;
      else if (strcmp ("if", novo.lexema) == 0) novo.token = IF;
      else if (strcmp ("else", novo.lexema) == 0) novo.token = ELSE;
      else if (strcmp ("while", novo.lexema) == 0) novo.token = WHILE;
      else if (strcmp ("do", novo.lexema) == 0) novo.token = DO;
      else if (strcmp ("for", novo.lexema) == 0) novo.token = FOR;
      else novo.token = ID;

      return novo;
}
// VALOR_CHAR, isto é 'letra' ou 'digito'
else if(ch == 39) { // Na tabela ASCII 39 é o síbolo '
      preencherLexema (source, &ch, novo.lexema, &tamLexema);
      if(isalnum(ch)) {
            preencherLexema (source, &ch, novo.lexema, &tamLexema);
            if(ch == 39) {
                  preencherLexema (source, &ch, novo.lexema, &tamLexema);
                  novo.token=VALOR_CHAR;
                  return novo;
            }
            else { // ERRO CHAR MAL FORMADO
                  preencherLexema (source, &ch, novo.lexema, &tamLexema);
                  exibirErro (source,"char mal formado", novo.lexema);
            }
      }
      else { // ERRO CHAR MAL FORMADO
            preencherLexema (source, &ch, novo.lexema, &tamLexema);
            exibirErro (source,"char mal formado", novo.lexema);
      }
}
// VALOR_INT, isto é dígito+
else if (isdigit(ch)){
      preencherLexema (source, &ch, novo.lexema, &tamLexema);
      while(isdigit(ch)) preencherLexema (source, &ch, novo.lexema, &tamLexema);
      if (ch == '.')
            goto DECLARACAO_DE_FLOATS;
      novo.token=VALOR_INT;
      return novo;
}
```

```
	// VALOR_FLOAT, isto é dígito*.dígito+
	else if(ch == '.') {
		DECLARACAO_DE_FLOATS:
		preencherLexema (source, &ch, novo.lexema, &tamLexema);
		if (!isdigit(ch)) { // ERRO FLOAT MAL FORMADO
			preencherLexema (source, &ch, novo.lexema, &tamLexema);
			exibirErro (source,"float mal formado", novo.lexema);
		}
		while(isdigit(ch)) preencherLexema (source, &ch, novo.lexema, &tamLexema);
		novo.token=VALOR_FLOAT;
		return novo;
	}

	// DIVISÃO, COMENTÁRIO DE UMA E MÚLTIPLAS LINHAS (DIFULCULDADE DIFÍCIL)
	else if (ch == '/')
	{
		preencherLexema (source, &ch, novo.lexema, &tamLexema);
		if (ch == '/') {
			do {
				ch = fgetc (source);
			} while(ch != '\n' && !feof(source));7
			goto INICIO;
		}
		else if (ch == '*') {
		RECOMECAR_VERIFICACAO_DE_COMENTARIO_MULTIPLAS_LINHAS: // ERRO QUANDO HOUVER ('*' SEM O '/'
APÓS); EXEMPLO: /* 123 *5
			while (!feof(source) && ch != '*') {
				switch (ch) {
					case '\t': coluna = coluna +4; break;
					case '\n': coluna=1; linha++; break;
					default: coluna++; break;
				}
				ch=getc(source);
			}
			// ERRO FIM DE ARQUIVO DENTRO DE COMENTARIO
			if (feof(source)) // erro em caso parecido com esse /*123141
				exibirErro (source,"fim de arquivo dentro de comentario", novo.lexema);
			if (ch == '*')
			{
				ch = getc(source);
				while (ch == '*')
					ch = getc(source);
				if (ch == '/') // sai do comentário  /**/ com sucesso
				{
					ch = getc(source);
					coluna++;
					goto INICIO;
				}
				// ERRO FIM DE ARQUIVO DENTRO DE COMENTARIO
				// erro em caso parecido com esse /*123141 * ou  /*123141 ********
				if (feof(source))
					exibirErro (source,"fim de arquivo dentro de comentario", novo.lexema);
				else // caso /* 48326423746 * (não teve o /)
					goto RECOMECAR_VERIFICACAO_DE_COMENTARIO_MULTIPLAS_LINHAS;
			}
		}
		else {
			novo.token = DIVISAO;
			return novo;
		}
	}
	// ERRO CARACTER INVALIDO
	else { // Tudo que não estiver na gramática acima é caracter inválido
		preencherLexema (source, &ch, novo.lexema, &tamLexema);
		exibirErro (source,"caracter invalido", novo.lexema);
	}
}
```

2 Como fazer um PARSER para crianças

2.1 Cabeçalho

```
/////// LEGENDA DO PARSER
//VERDE Chamada do SCANNER ou modificações no cabeçalho para implementar o PARSER
//AZUL Chamada de procedimento
//AMARELO erro detectado pelo PARSER
//CINZA Não é um if para testar se houve erro, mas para verificar o first

// Bibliotecas usadas
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

// Estruturas definidas
typedef enum token {
      // PALAVRAS RESERVADAS
      MAIN, VALOR_INT, VALOR_FLOAT, VALOR_CHAR, IF, ELSE, WHILE, DO, FOR, ID, TIPO_CHAR, TIPO_FLOAT,
TIPO_INT,
      //OPERADOR RELACIONAL
      MAIOR, MAIOR_IGUAL, MENOR, MENOR_IGUAL, IGUALDADE,
      // OPERAÇÕES ARITMÉTICAS
      SOMA, SUBTRACAO, MULT, DIVISAO, DIFERENCA, ATRIBUIR,
      // MARCADORES
      ABRE_PARENTESES, FECHA_PARENTESES, ABRE_CHAVES, FECHA_CHAVES, VIRGULA, PONTO_E_VIRGULA,
FIM_DE_ARQUIVO
} TOKEN;

typedef struct no {
      char lexema [50];
      TOKEN  token;
} TNo;

// Variáveis Globais
int linha = 1, coluna = 0;

// Cabeçalho dos procedimentos e funções utilizados
// Pocedimentos auxiliares para todo o programa
void exibirErro (FILE *source, char msg[], char lexema[]);
// Cabeçalho do SCANNER
TNo SCANNER (FILE * source);
void preencherLexema (FILE * arq, char* ch, char lexema[], int *tamLexema);
// Cabeçalho do PARSER
void PROGRAMA (FILE *source); // modificação na implementação
void BLOCO (FILE * source, TNo *reader);
void DECLARAR_VARIAVEIS (FILE * source, TNo *reader);
void COMANDO (FILE * source, TNo *reader);
void ITERACAO (FILE * source, TNo *reader);
void EXPR_RELACIONAL (FILE * source, TNo *reader);
void COMANDO_BASICO (FILE * source, TNo *reader);
void ATRIBUICAO (FILE * source, TNo *reader);
void EXPRESSAO_ARITMETICA (FILE * source, TNo *reader);
void EXPRESSAO_ARITMETICA_LINHA (FILE * source, TNo *reader);
void TERMO (FILE * source, TNo *reader);
void FATOR (FILE * source, TNo *reader);
```

2.2 Novidades

```
void PROGRAMA (FILE *source) { // modificação na implementação
	TNo reader;
	reader = SCANNER (source);
	if (reader.token != TIPO_INT) exibirErro (source,"token INT nao encontrado", reader.lexema);
	reader = SCANNER (source);
	if (reader.token != MAIN) exibirErro (source,"token MAIN nao encontrado", reader.lexema);
	reader = SCANNER (source);
	if (reader.token != ABRE_PARENTESES) exibirErro (source,"token ABRE PARENTESES nao encontrado",
reader.lexema);
	reader = SCANNER (source);
	if (reader.token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao encontrado",
reader.lexema);
	reader = SCANNER (source);
	BLOCO (source, &reader);
	if (reader.token != FIM_DE_ARQUIVO) exibirErro (source,"Existem tokens declarados fora do main",
reader.lexema);
}
void BLOCO (FILE * source, TNo *reader) {
	if (reader->token != ABRE_CHAVES) exibirErro (source,"token ABRE CHAVES nao encontrado", reader-
>lexema);
	*reader = SCANNER (source);
	while (reader->token == TIPO_INT || reader->token == TIPO_FLOAT || reader->token == TIPO_CHAR)
		DECLARAR_VARIAVEIS (source, reader);
	while (reader->token == ID || reader->token == ABRE_CHAVES || reader->token == WHILE || reader-
>token == DO || reader->token == IF)
		COMANDO (source, reader);
	if (reader->token != FECHA_CHAVES) exibirErro (source,"token FECHA CHAVES nao encontrado",
reader->lexema);
	*reader = SCANNER (source);
}
void DECLARAR_VARIAVEIS (FILE * source, TNo *reader) {
	do {
		*reader = SCANNER (source);
		if (reader->token != ID) exibirErro (source,"token IDENTIFICADOR nao encontrado", reader-
>lexema);
		*reader = SCANNER (source);
	} while (reader->token == VIRGULA);
	if (reader->token != PONTO_E_VIRGULA) exibirErro (source,"token PONTO E VIRGULA nao encontrado",
reader->lexema);
	*reader = SCANNER (source);
}
void COMANDO (FILE * source, TNo *reader) {
	if (reader->token == ID || reader->token == ABRE_CHAVES)  {
		COMANDO_BASICO (source, reader);
	}
	else if (reader->token == IF)    {
		*reader = SCANNER (source);
		if (reader->token != ABRE_PARENTESES) exibirErro (source,"token ABRE PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		EXPR_RELACIONAL (source, reader);
		if (reader->token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		COMANDO (source, reader);
		if (reader->token == ELSE) {
			*reader = SCANNER (source);
			COMANDO (source, reader);
		}
	}
	else if (reader->token == DO || reader->token == WHILE) ITERACAO (source, reader);
	else if (reader->token == TIPO_INT || reader->token == TIPO_CHAR || reader->token == TIPO_FLOAT)
		exibirErro (source,"Nao se pode declarar variavel fora de bloco", reader->lexema);
}
```

```
void ITERACAO (FILE * source, TNo *reader) {
	if (reader->token == WHILE)
	{
		*reader = SCANNER (source);
		if (reader->token != ABRE_PARENTESES) exibirErro (source,"token ABRE PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		EXPR_RELACIONAL (source, reader);
		if (reader->token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		COMANDO (source, reader);
	}
	else if (reader->token == DO)
	{
		*reader = SCANNER (source);
		COMANDO (source, reader);
		if (reader->token != WHILE) exibirErro (source,"token \"WHILE\" do comando \"DO WHILE\" nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		if (reader->token != ABRE_PARENTESES) exibirErro (source,"token ABRE PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		EXPR_RELACIONAL (source, reader);
		if (reader->token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		if (reader->token != PONTO_E_VIRGULA) exibirErro (source,"token PONTO E VIRGULA nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
	}
}

void EXPR_RELACIONAL (FILE * source, TNo *reader) {
	EXPRESSAO_ARITMETICA (source, reader);
	if (reader->token != MAIOR && reader->token != MAIOR_IGUAL && reader->token != MENOR && reader-
>token != MENOR_IGUAL && reader->token != DIFERENCA && reader->token != IGUALDADE)
		exibirErro (source,"token OPERADOR RELACIONAL nao encontrado", reader->lexema);
	*reader = SCANNER (source);
	EXPRESSAO_ARITMETICA (source, reader);
}

void COMANDO_BASICO (FILE * source, TNo *reader) {
	if (reader->token == ID)   {
		*reader = SCANNER (source);
		ATRIBUICAO (source, reader);
	}
	else if (reader->token == ABRE_CHAVES)     {
		BLOCO (source, reader);
	}
}

void ATRIBUICAO (FILE * source, TNo *reader) {
	if (reader->token != ATRIBUIR) exibirErro (source,"token ATRIBUICAO nao encontrado", reader-
>lexema);
	*reader = SCANNER (source);
	EXPRESSAO_ARITMETICA (source, reader);
	if (reader->token != PONTO_E_VIRGULA) exibirErro (source,"token PONTO E VIRGULA nao encontrado",
reader->lexema);
	*reader = SCANNER (source);
}
```

```
void EXPRESSAO_ARITMETICA (FILE * source, TNo *reader) { // Implementado sem recursividade a esquerda
	TERMO (source, reader);
	EXPRESSAO_ARITMETICA_LINHA (source, reader);
}

void EXPRESSAO_ARITMETICA_LINHA (FILE * source, TNo *reader) {
	if (reader->token == SOMA || reader->token == SUBTRACAO) {
		*reader = SCANNER (source);
		TERMO (source, reader);
		EXPRESSAO_ARITMETICA_LINHA (source, reader);
	}
}

void TERMO (FILE * source, TNo *reader) { // implementação sem recursividade
	FATOR (source, reader);
	while(reader->token == MULT || reader->token == DIVISAO){
		*reader = SCANNER (source);
		FATOR (source, reader);
	}
}

void FATOR (FILE * source, TNo *reader) {
	if (reader->token == ID || reader->token == VALOR_INT || reader->token == VALOR_FLOAT || reader-
>token == VALOR_CHAR) {
		*reader = SCANNER (source);
		return;
	}
	else if (reader->token == ABRE_PARENTESES) {
		*reader = SCANNER (source);
		EXPRESSAO_ARITMETICA (source, reader);
		if (reader->token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
	}
	else { // Devido a Fator aceitar palavra vazia é preciso printer erro aqui
		exibirErro (source,"Expressao aritmetica mal formada", reader->lexema);
	}
}
```

3 Como fazer uma Tabela de Símbolos de maneira prática

3.1 Introdução

Como a tabela de símbolos é implementada?

A tabela de símbolos (TS) é implementada pela estrutura de dados lista simplesmente encadeada (LSE), com inserção e remoção no início. A estrutura da TS é composta por 4 campos: lexema, TOKEN, escopo e um ponteiro para o próximo nó.

Como é manipulada quando entra e sai de um escopo local?

Quando o programa entra em um novo bloco, isto é, o token ABRE_CHAVES é lido pelo scanner, a variável global escopo (no meu programa eu o nomeio como escopoAtual, mas na Figura 1, é apenas escopo) é incrementada e a TS não sofre alterações.

Ao sair do bloco, isto é, o token FECHA_CHAVES é lido pelo scanner, a variável escopo é decrementada e as variáveis decladaras somente naquele escopo antigo são removidas da TS.

Como as variáveis declaradas no bloco interno são inseridas e depois removidas ao término do bloco?

As variáveis são inseridas no início da lista e ao sair do bloco são removidas todas aquelas que pertenciam aquele escopo.

Como funcionam as buscas e qual a diferença da busca no escopo atual e todos os escopos?

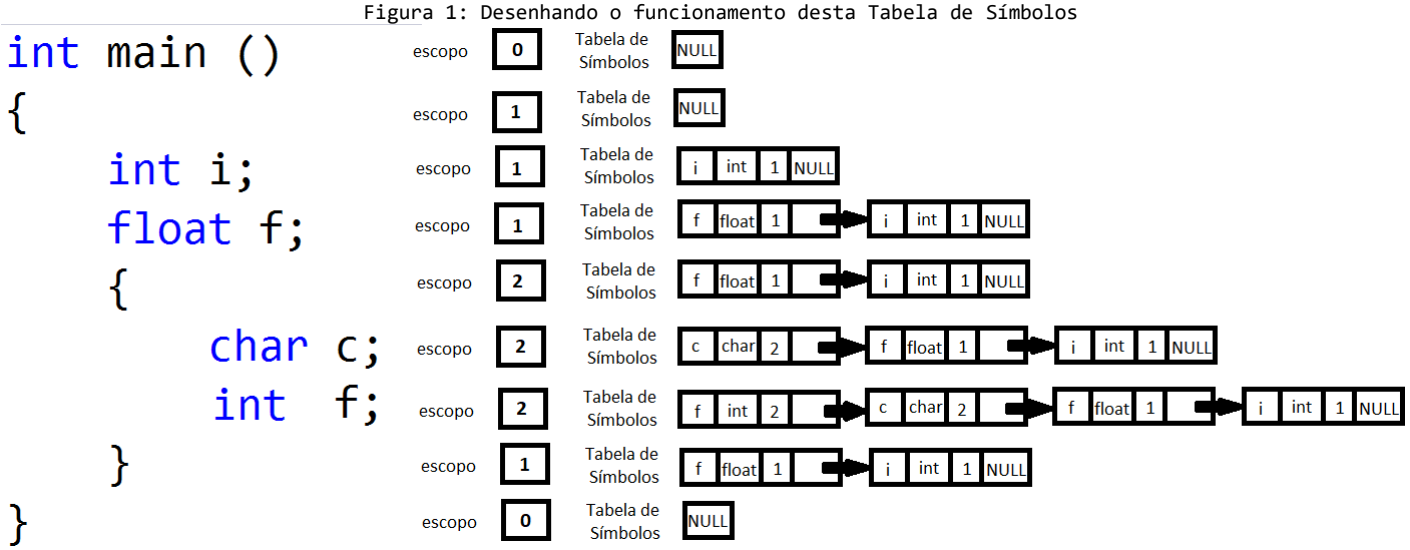
As buscas começam percorrendo o início da LSE até o objetivo da chamada, isto é, percorrem o escopo atual ou todos os escopos.

Busca no escopo atual: Ao declarar uma variável é verificado se existe outra com o mesmo nome já declarada naquele bloco/escopo/contexto. Caso haja, erro variável já declarada neste bloco.

Busca em todos os escopos: Numa expressão aritmética é verificado se a variável foi declarada, para isso percorre-se a TS, porém neste compilador, a variável não necessariamente estará declarada no mesmo escopo em que foi usada, pode ser um anterior. Assim, é feito uma busca em todos os escopos. Caso ela não seja encontrada é um erro de variável não declarada.

Para a conveniência dos leitores, uma representação é apresentada na Figura 1.

Figura 1: Desenhando o funcionamento desta Tabela de Símbolos



3.2 Cabeçalho

```
/////// LEGENDA DA TABELA DE SIMBOLOS
//VERDE Modificações da tabela de símbolos

// Bibliotecas usadas
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

// Estruturas definidas
typedef enum token {
      // PALAVRAS RESERVADAS
      MAIN, VALOR_INT, VALOR_FLOAT, VALOR_CHAR, IF, ELSE, WHILE, DO, FOR, ID, TIPO_CHAR, TIPO_FLOAT,
TIPO_INT,
      //OPERADOR RELACIONAL
      MAIOR, MAIOR_IGUAL, MENOR, MENOR_IGUAL, IGUALDADE,
      // OPERAÇÕES ARITMÉTICAS
      SOMA, SUBTRACAO, MULT, DIVISAO, DIFERENCA, ATRIBUIR,
      // MARCADORES
      ABRE_PARENTESES, FECHA_PARENTESES, ABRE_CHAVES, FECHA_CHAVES, VIRGULA, PONTO_E_VIRGULA,
FIM_DE_ARQUIVO
} TOKEN;

// Estruturas definidas
typedef struct no {
      char lexema [50];
      TOKEN   token;
      int escopo;
      struct no * prox;
} TNo;

// Variáveis Globais
int linha = 1, coluna = 0, escopoAtual = 0;
TNo * tabelaDeSimbolos = NULL;

// Cabeçalho dos procedimentos e funções utilizados

// Pocedimentos auxiliares para todo o programa
void exibirErro (FILE *source, char msg[], char lexema[]);// modificação na implementação

// Cabeçalho do SCANNER
TNo SCANNER (FILE * source);
void preencherLexema (FILE * arq, char* ch, char lexema[], int *tamLexema);

// Cabeçalho do PARSER
void PROGRAMA (FILE *source);
void BLOCO (FILE * source, TNo *reader); // modificação na implementação
void DECLARAR_VARIAVEIS (FILE * source, TNo *reader); // modificação na implementação
void COMANDO (FILE * source, TNo *reader);
void ITERACAO (FILE * source, TNo *reader);
void EXPR_RELACIONAL (FILE * source, TNo *reader);
void COMANDO_BASICO (FILE * source, TNo *reader); // modificação na implementação
void ATRIBUICAO (FILE * source, TNo *reader);
void EXPRESSAO_ARITMETICA (FILE * source, TNo *reader);
void EXPRESSAO_ARITMETICA_LINHA (FILE * source, TNo *reader);
void TERMO (FILE * source, TNo *reader);
void FATOR (FILE * source, TNo *reader); // modificação na implementação

// Novos procedimentos e funções para a Tabela de Simbolos
void InserirTabelaDeSimbolos (FILE *source, char lexema_ID[50], TOKEN token_ID, TNo *reader);
void removerEscopoAntigoDaTabelaDeSimbolos (int escopoAtual);
TNo* buscar_ID_no_EscopoAtual_Apenas (char lexema_ID[50]);
TNo* buscar_ID_em_todos_escopos (char lexema_ID[50]);
```

3.3 Novidades

```
////////---Novas implementações para usar a tabela de simbolos---///////////

void InserirTabelaDeSimbolos (FILE *source, char lexema_ID[50], TOKEN token_ID, TNo *reader) {
	TNo *Novo;

	if (buscar_ID_no_EscopoAtual_Apenas(lexema_ID) == NULL)
	{
		Novo = (TNo*)malloc(sizeof(TNo));
		strcpy (Novo->lexema, lexema_ID);
		Novo->token = token_ID;
		Novo->escopo = escopoAtual;
		if (tabelaDeSimbolos == NULL)
		{
			tabelaDeSimbolos = Novo;
			Novo->prox = NULL;
		}
		else
		{
			Novo->prox = tabelaDeSimbolos;
			tabelaDeSimbolos = Novo;
		}
	}
	else{
		exibirErro (source,"Variavel ja declarada neste escopo", reader->lexema);
	}
}
void removerEscopoAntigoDaTabelaDeSimbolos (int escopoAtual) {
	TNo *aux = tabelaDeSimbolos;

	while (aux != NULL) {
		if (aux->escopo > escopoAtual)
		{
			tabelaDeSimbolos = tabelaDeSimbolos -> prox;
			free (aux);
			aux = tabelaDeSimbolos;
		}
		else
			break;
	}
}
TNo* buscar_ID_no_EscopoAtual_Apenas (char lexema_ID[50]) {
	TNo *aux;
	if (tabelaDeSimbolos != NULL)
	{
		for (aux = tabelaDeSimbolos; aux != NULL ; aux = aux->prox)
		{
		if (aux->escopo < escopoAtual)
				break;
			if (strcmp (aux -> lexema, lexema_ID) == 0)
				return aux;
		}
	}
	return NULL;
}
TNo* buscar_ID_em_todos_escopos (char lexema_ID[50]) {
	TNo *aux = tabelaDeSimbolos;

	while (aux != NULL)
	{
		if (strcmp (aux -> lexema, lexema_ID) == 0)
				return aux;
		 aux = aux->prox;
	}
	return NULL;
}
```

3.4 Modificações

```
////////---Modificações no código para usar a tabela de símbolos---///////////

void BLOCO (FILE * source, TNo *reader){
	if (reader->token != ABRE_CHAVES) exibirErro (source,"token ABRE CHAVES nao encontrado", reader->lexema);
	escopoAtual++;
	*reader = SCANNER (source);
	while (reader->token == TIPO_INT || reader->token == TIPO_FLOAT || reader->token == TIPO_CHAR)
		DECLARAR_VARIAVEIS (source, reader);
	while (reader->token == ID || reader->token == ABRE_CHAVES || reader->token == WHILE || reader->token == DO || reader->token == IF)
		COMANDO (source, reader);
	if (reader->token != FECHA_CHAVES) exibirErro (source,"token FECHA CHAVES nao encontrado", reader->lexema);
	escopoAtual--;
	removerEscopoAntigoDaTabelaDeSimbolos (escopoAtual);
	*reader = SCANNER (source);
}

void DECLARAR_VARIAVEIS (FILE * source, TNo *reader){
	char lexema_ID[50];
	TOKEN token_ID;

	token_ID = reader->token; // Pega o tipo das variáveis que serão declaradas

	do {
		*reader = SCANNER (source);
		if (reader->token != ID) exibirErro (source,"token IDENTIFICADOR nao encontrado", reader->lexema);
		strcpy (lexema_ID,reader->lexema); // Pega o lexema da variável declarada
		InserirTabelaDeSimbolos (source, lexema_ID, token_ID, reader);
		*reader = SCANNER (source);
	} while (reader->token == VIRGULA);
	if (reader->token != PONTO_E_VIRGULA) exibirErro (source,"token PONTO E VIRGULA nao encontrado", reader->lexema);
	*reader = SCANNER (source);
}

void exibirErro (FILE *source, char msg[], char lexema[]){ // Proced. para exibir erro elegantemente
	printf("erro na linha %i coluna %i ultimo lexema lido %s : %s\n", linha, coluna, lexema, msg);
	removerEscopoAntigoDaTabelaDeSimbolos (0);
	fclose (source);
	exit (0);
}

void COMANDO_BASICO (FILE * source, TNo *reader) {
	TNo *verify;

	if (reader->token == ID) {
		verify = buscar_ID_em_todos_escopos(reader->lexema);
		if (verify == NULL) exibirErro (source,"variavel nao declarada em nenhum escopo", reader->lexema);
		*reader = SCANNER (source);
		ATRIBUICAO (source, reader);
	}
	else if (reader->token == ABRE_CHAVES)	{
		BLOCO (source, reader);
	}
}
```

```
void FATOR (FILE * source, TNo *reader) {
      TNo *verify;

      if (reader->token == ID || reader->token == VALOR_INT || reader->token == VALOR_FLOAT || reader-
>token == VALOR_CHAR) {
            if(reader->token == ID){
                  verify=buscar_ID_em_todos_escopos(reader->lexema);
                  if (verify == NULL) exibirErro (source,"variavel nao declarada em nenhum escopo",
reader->lexema);
            }
            *reader = SCANNER (source);
            return;
      }
      else if (reader->token == ABRE_PARENTESES) {
            *reader = SCANNER (source);
            EXPRESSAO_ARITMETICA (source, reader);
            if (reader->token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao
encontrado", reader->lexema);
            *reader = SCANNER (source);
      }
      else { // Devido a Fator aceitar palavra vazia é preciso printer erro aqui
            exibirErro (source,"Expressao aritmetica mal formada", reader->lexema);
      }
}
```

4 Como fazer o Semântico para aprendizes

4.1 Cabeçalho

```
/////// LEGENDA DO SEMÂNTICO
//VERDE Modificações do semântico no parser

// Bibliotecas usadas
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
// Estruturas definidas
typedef enum token {
      // PALAVRAS RESERVADAS
      MAIN, VALOR_INT, VALOR_FLOAT, VALOR_CHAR, IF, ELSE, WHILE, DO, FOR, ID, TIPO_CHAR, TIPO_FLOAT,
TIPO_INT,
      //OPERADOR RELACIONAL
      MAIOR, MAIOR_IGUAL, MENOR, MENOR_IGUAL, IGUALDADE,
      // OPERAÇÕES ARITMÉTICAS
      SOMA, SUBTRACAO, MULT, DIVISAO, DIFERENCA, ATRIBUIR,
      // MARCADORES
      ABRE_PARENTESES, FECHA_PARENTESES, ABRE_CHAVES, FECHA_CHAVES, VIRGULA, PONTO_E_VIRGULA,
FIM_DE_ARQUIVO,
      NIL // Novo TOKEN para auxiliar no semântico
} TOKEN;
typedef struct no {
      char lexema [50];
      TOKEN  token;
      int escopo;
      struct no * prox;
} TNo;

// Variáveis Globais
int linha = 1, coluna = 0, escopoAtual = 0;
TNo * tabelaDeSimbolos = NULL;

// Pocedimentos auxiliares de todo o programa
void exibirErro (FILE *source, char msg[], char lexema[]);

// Cabeçalho do SCANNER
TNo SCANNER (FILE * source);
void preencherLexema (FILE * arq, char* ch, char lexema[], int *tamLexema);

// Cabeçalho do PARSER
void PROGRAMA (FILE *source);
void BLOCO (FILE * source, TNo *reader);
void DECLARAR_VARIAVEIS (FILE * source, TNo *reader);
void COMANDO (FILE * source, TNo *reader);
void ITERACAO (FILE * source, TNo *reader);
void EXPR_RELACIONAL (FILE * source, TNo *reader);
void COMANDO_BASICO (FILE * source, TNo *reader);
TNo ATRIBUICAO (FILE * source, TNo *reader);
TNo EXPRESSAO_ARITMETICA (FILE * source, TNo *reader);
TNo EXPRESSAO_ARITMETICA_LINHA (FILE * source, TNo *reader);
TNo TERMO (FILE * source, TNo *reader);
TNo FATOR (FILE * source, TNo *reader);

// Cabeçalho da Tabela de Simbolos
void InserirTabelaDeSimbolos (FILE *source, char lexema_ID[50], TOKEN token_ID, TNo *reader);
void removerEscopoAntigoDaTabelaDeSimbolos (int escopoAtual);
TNo* buscar_ID_no_EscopoAtual_Apenas (char lexema_ID[50]);
TNo* buscar_ID_em_todos_escopos (char lexema_ID[50]);

// Cabeçalho do Semântico
void verificarOperadores (FILE*source, TNo *reader, TNo *operando1, TNo *operando2, int divisao_ocorreu);
void verificarOperadoresEmUmaAtribuicao (FILE*source, TNo *reader, TNo *operando1, TNo *operando2);
```

4.2 Novidades

```
////////---Procedimentos do semântico---///////////
void verificarOperadores (FILE*source, TNo *reader, TNo *operando1, TNo *operando2, int
divisao_ocorreu) {
	if (operando1->token == TIPO_CHAR && operando2->token != TIPO_CHAR) exibirErro (source,"tipo char
nao eh compativel com um tipo diferente de char", reader->lexema);
	if (operando1->token == TIPO_INT && operando2->token == TIPO_CHAR) exibirErro (source,"tipo int
nao eh compativel com um tipo char", reader->lexema);
	if (operando1->token == TIPO_FLOAT && operando2->token == TIPO_CHAR) exibirErro (source,"tipo
float nao eh compativel com um tipo char", reader->lexema);
	if (operando1->token == TIPO_INT && operando2->token == TIPO_FLOAT) {
		operando1->token = TIPO_FLOAT;
	}
	if (operando1->token == TIPO_FLOAT && operando2->token == TIPO_INT) {
		operando2->token = TIPO_FLOAT;
	}
	if (divisao_ocorreu == 1 && operando1->token == TIPO_INT && operando2->token == TIPO_INT) {
		operando1->token = TIPO_FLOAT;
	}
}

void verificarOperadoresEmUmaAtribuicao (FILE*source, TNo *reader, TNo *operando1, TNo *operando2) {
	if (operando1->token == TIPO_CHAR && operando2->token != TIPO_CHAR) exibirErro (source,"tipo char
so pode receber valores do tipo char", reader->lexema);
	if (operando1->token == TIPO_INT && operando2->token != TIPO_INT) exibirErro (source,"tipo int so
pode receber valores do tipo int", reader->lexema);
	if (operando1->token == TIPO_FLOAT && operando2->token == TIPO_CHAR) exibirErro (source,"tipo
float so pode receber valores do tipo float", reader->lexema);
	if (operando1->token == TIPO_FLOAT && operando2->token == TIPO_INT) {
		operando2->token = TIPO_FLOAT;
	}
}
```

4.3 Modificações

```
void EXPR_RELACIONAL (FILE * source, TNo *reader) {
	TNo operando1, operando2;

	operando1 = EXPRESSAO_ARITMETICA (source, reader);
	if (reader->token != MAIOR && reader->token != MAIOR_IGUAL && reader->token != MENOR && reader-
>token != MENOR_IGUAL && reader->token != DIFERENCA && reader->token != IGUALDADE)
		exibirErro (source,"token OPERADOR RELACIONAL nao encontrado", reader->lexema);
	*reader = SCANNER (source);
	operando2 = EXPRESSAO_ARITMETICA (source, reader);

	verificarOperadores (source, reader, &operando1, &operando2, 0);
}


void COMANDO_BASICO (FILE * source, TNo *reader) {
	TNo operando1, operando2;
	TNo *verify;

	if (reader->token == ID) {
		verify = buscar_ID_em_todos_escopos(reader->lexema);
		if (verify == NULL) exibirErro (source,"variavel nao declarada em nenhum escopo", reader-
>lexema);
		else {
			strcpy (operando1.lexema, verify->lexema);
			operando1.token = verify->token;
		}
		*reader = SCANNER (source);
		operando2 = ATRIBUICAO (source, reader);
		verificarOperadoresEmUmaAtribuicao (source, reader, &operando1, &operando2);
	}
	else if (reader->token == ABRE_CHAVES)	{
		BLOCO (source, reader);
	}
}


TNo ATRIBUICAO (FILE * source, TNo *reader) {
	TNo operando;
	if (reader->token != ATRIBUIR) exibirErro (source,"token ATRIBUICAO nao encontrado", reader-
>lexema);
	*reader = SCANNER (source);
	operando = EXPRESSAO_ARITMETICA (source, reader);
	if (reader->token != PONTO_E_VIRGULA) exibirErro (source,"token PONTO E VIRGULA nao encontrado",
reader->lexema);
	*reader = SCANNER (source);
	return operando;
}


TNo EXPRESSAO_ARITMETICA (FILE * source, TNo *reader) {
	TNo operando1, operando2;

	operando1.token = NIL; operando2.token = NIL;
	operando1 = TERMO (source, reader);
	operando2 = EXPRESSAO_ARITMETICA_LINHA (source, reader);

	if (operando2.token == TIPO_CHAR || operando2.token == TIPO_INT || operando2.token == TIPO_FLOAT)
	{
		verificarOperadores (source, reader, &operando1, &operando2, 0);
	}
	return operando1;
}
```

```
TNo EXPRESSAO_ARITMETICA_LINHA (FILE * source, TNo *reader) {
	TNo operando1, operando2;

	operando1.token = NIL; operando2.token = NIL;
	if (reader->token == SOMA || reader->token == SUBTRACAO) {
		*reader = SCANNER (source);
		operando1 = TERMO (source, reader);
		operando2 = EXPRESSAO_ARITMETICA_LINHA (source, reader);
	}
	if (operando2.token == TIPO_CHAR || operando2.token == TIPO_INT || operando2.token == TIPO_FLOAT)
	{
		verificarOperadores (source, reader, &operando1, &operando2, 0);
	}
	return operando1;
}


TNo TERMO (FILE * source, TNo *reader) {
	TNo operando1, operando2;
	int divisao_ocorreu = 0;

	operando1.token = NIL; operando2.token = NIL;
	operando1 = FATOR (source, reader);
	while(reader->token == MULT || reader->token == DIVISAO){
		if (reader->token == DIVISAO) divisao_ocorreu = 1;
		*reader = SCANNER (source);
		operando2 = FATOR (source, reader);
		verificarOperadores (source, reader, &operando1, &operando2, divisao_ocorreu);
		divisao_ocorreu = 0;
	}
	return operando1;
}
```

```
TNo FATOR (FILE * source, TNo *reader) {
      TNo operando;
      TNo *verify;

      if (reader->token == ID || reader->token == VALOR_INT || reader->token == VALOR_FLOAT || reader-
>token == VALOR_CHAR)
      {
            if(reader->token == ID)
            {
                  verify=buscar_ID_em_todos_escopos(reader->lexema);
                  if (verify == NULL) exibirErro (source,"variavel nao declarada em nenhum escopo",
reader->lexema);
                  else {
                        strcpy (operando.lexema, verify->lexema);
                        operando.token = verify->token;
                  }
            }
            else
            {
                  strcpy (operando.lexema, reader->lexema);
                  switch (reader->token)
                  {
                        case VALOR_CHAR: operando.token = TIPO_CHAR; break;
                        case VALOR_INT: operando.token = TIPO_INT; break;
                        case VALOR_FLOAT: operando.token = TIPO_FLOAT; break;
                  }
            }
            *reader = SCANNER (source);
            return operando;
      }
      else if (reader->token == ABRE_PARENTESES) {
            *reader = SCANNER (source);
            operando = EXPRESSAO_ARITMETICA (source, reader);
            if (reader->token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao
encontrado", reader->lexema);
            *reader = SCANNER (source);
            return operando;
      }
      else { // Devido a Fator aceitar palavra vazia é preciso printer erro aqui
            exibirErro (source,"Expressao aritmetica mal formada", reader->lexema);
      }
}
```

5 Como fazer o Gerador de Código Intermediário para jovens

5.1 Cabeçalho

```
/////// LEGENDA do Gerador de Código Intermediário
//VERDE Modificações do Gerador de Código Intermediário
// Bibliotecas usadas
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
// Estruturas definidas
typedef enum token {
      // PALAVRAS RESERVADAS
      MAIN, VALOR_INT, VALOR_FLOAT, VALOR_CHAR, IF, ELSE, WHILE, DO, FOR, ID, TIPO_CHAR, TIPO_FLOAT,
TIPO_INT,
      //OPERADOR RELACIONAL
      MAIOR, MAIOR_IGUAL, MENOR, MENOR_IGUAL, IGUALDADE,
      // OPERAÇÕES ARITMÉTICAS
      SOMA, SUBTRACAO, MULT, DIVISAO, DIFERENCA, ATRIBUIR,
      // MARCADORES
      ABRE_PARENTESES, FECHA_PARENTESES, ABRE_CHAVES, FECHA_CHAVES, VIRGULA, PONTO_E_VIRGULA,
FIM_DE_ARQUIVO,
      NIL // Novo token para auxiliar no semântico
} TOKEN;
typedef struct no {
      char lexema [50];
      TOKEN  token;
      int escopo;
      int T_id;
      struct no * prox;
} TNo;
// Variáveis Globais
int linha = 1, coluna = 0, escopoAtual = 0, Tn = 0, Ln = 0;
TNo * tabelaDeSimbolos = NULL;
// Pocedimentos auxiliares de todo o programa
void exibirErro (FILE *source, char msg[], char lexema[]);
// Cabeçalho do SCANNER
TNo SCANNER (FILE * source);
void preencherLexema (FILE * arq, char* ch, char lexema[], int *tamLexema);
// Cabeçalho do PARSER
void PROGRAMA (FILE *source);
void BLOCO (FILE * source, TNo *reader);
void DECLARAR_VARIAVEIS (FILE * source, TNo *reader);
void COMANDO (FILE * source, TNo *reader);
void ITERACAO (FILE * source, TNo *reader);
TNo EXPR_RELACIONAL (FILE * source, TNo *reader);
void COMANDO_BASICO (FILE * source, TNo *reader);
TNo ATRIBUICAO (FILE * source, TNo *reader);
TNo EXPRESSAO_ARITMETICA (FILE * source, TNo *reader);
TNo EXPRESSAO_ARITMETICA_LINHA (FILE * source, TNo *reader);
TNo TERMO (FILE * source, TNo *reader);
TNo FATOR (FILE * source, TNo *reader);
// Cabeçalho da Tabela de Simbolos
void InserirTabelaDeSimbolos (FILE *source, char lexema_ID[50], TOKEN token_ID, TNo *reader);
void removerEscopoAntigoDaTabelaDeSimbolos (int escopoAtual);
TNo* buscar_ID_no_EscopoAtual_Apenas (char lexema_ID[50]);
TNo* buscar_ID_em_todos_escopos (char lexema_ID[50]);
// Cabeçalho do Semântico
void verificarOperadores (FILE*source, TNo *reader, TNo *operando1, TNo *operando2, int
divisao_ocorreu);
void verificarOperadoresEmUmaAtribuicao (FILE*source, TNo *reader, TNo *operando1, TNo *operando2);
// Cabeçalho do Gerador de Código Intermediário
TNo geradorDeCodigoIntermediario (TNo operando1, char *operacao, TNo operando2);
```

5.2 Novidades

```
// Implementação do cabeçalho do Gerador de Código Intermediário
TNo geradorDeCodigoIntermediario (TNo operando1, char *operacao, TNo operando2)
{
      printf ("T%i = %s",Tn, operando1.lexema);
      if (operando1.T_id >= 0) printf ("%i", operando1.T_id);
      printf (" %s", operacao);
      printf (" %s", operando2.lexema);
      if (operando2.T_id >= 0) printf ("%i", operando2.T_id);
      printf ("\n");
      strcpy (operando1.lexema, "T");
      operando1.T_id = Tn;
      Tn++;
      return operando1;
}
```

5.3 Modificações

```
void COMANDO (FILE * source, TNo *reader){
	TNo operando;
	int label_local = Ln;

	if (reader->token == ID || reader->token == ABRE_CHAVES)  {
		COMANDO_BASICO (source, reader);
	}
	else if (reader->token == IF)    {
		*reader = SCANNER (source);
		if (reader->token != ABRE_PARENTESES) exibirErro (source,"token ABRE PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		operando = EXPR_RELACIONAL (source, reader);
		if (reader->token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);

		// Gerador de código intermediário do Token IF
		printf ("if %s", operando.lexema);
		if (operando.T_id >= 0)    printf ("%i",operando.T_id);
		printf (" == 0 goto L%i\n",label_local);
		Ln++;
		COMANDO (source, reader);
		if (reader->token == ELSE) {
			// Gerador de código intermediário do Token ELSE
			printf ("goto L%i\n", Ln);
			printf ("L%i:\n", label_local);
			label_local = Ln;
			*reader = SCANNER (source);
			COMANDO (source, reader);
			printf ("L%i:\n",label_local);
		}
		else // if (reader->token != ELSE)
		{
			// Caso não haja declaração de else
			printf ("L%i:\n",label_local);
		}
	}
	else if (reader->token == DO || reader->token == WHILE)    {
		ITERACAO (source, reader);
	}
	else if (reader->token == TIPO_INT || reader->token == TIPO_CHAR || reader->token == TIPO_FLOAT)
	{
		exibirErro (source,"Nao se pode declarar variavel fora de bloco", reader->lexema);
	}
}
```

```
void ITERACAO (FILE * source, TNo *reader){
	TNo operando;
	int label_local = Ln;

	printf ("L%i:\n",label_local);
	if (reader->token == WHILE)
	{
		Ln = Ln + 2;
		*reader = SCANNER (source);
		if (reader->token != ABRE_PARENTESES) exibirErro (source,"token ABRE PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		operando = EXPR_RELACIONAL (source, reader);
		if (reader->token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		// Gerador de código intermediário do Token WHILE
		printf ("if %s", operando.lexema);
		if (operando.T_id >= 0)	printf ("%i",operando.T_id);
		printf (" == 0 goto L%i\n",label_local+1);
		COMANDO (source, reader);
		printf ("goto L%i\n",label_local);
		printf ("L%i:\n",label_local+1);
	}
	else if (reader->token == DO)
	{
		Ln++;
		*reader = SCANNER (source);
		COMANDO (source, reader);
		if (reader->token != WHILE) exibirErro (source,"token \"WHILE\" do comando \"DO WHILE\" nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		if (reader->token != ABRE_PARENTESES) exibirErro (source,"token ABRE PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		operando = EXPR_RELACIONAL (source, reader);
		if (reader->token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao
encontrado", reader->lexema);
		*reader = SCANNER (source);
		if (reader->token != PONTO_E_VIRGULA) exibirErro (source,"token PONTO E VIRGULA nao
encontrado", reader->lexema);
		*reader = SCANNER (source);

		// Gerador de código intermediário do Token DO WHILE
		printf ("if %s", operando.lexema);
		if (operando.T_id >= 0) printf ("%i",operando.T_id);
		printf (" != 0 goto L%i\n",label_local);
	}
}

TNo EXPR_RELACIONAL (FILE * source, TNo *reader) {
	TNo operando1, operando2;
	char operacao[50];

	operando1 = EXPRESSAO_ARITMETICA (source, reader);
	strcpy(operacao,reader->lexema);
	if (reader->token != MAIOR && reader->token != MAIOR_IGUAL && reader->token != MENOR && reader-
>token != MENOR_IGUAL && reader->token != DIFERENCA && reader->token != IGUALDADE)
		exibirErro (source,"token OPERADOR RELACIONAL nao encontrado", reader->lexema);
	*reader = SCANNER (source);
	operando2 = EXPRESSAO_ARITMETICA (source, reader);

	verificarOperadores (source, reader, &operando1, &operando2, 0);
	operando1 = geradorDeCodigoIntermediario (operando1, operacao, operando2);
	return operando1;
}
```

```
void verificarOperadores (FILE*source, TNo *reader, TNo *operando1, TNo *operando2, int
divisao_ocorreu) {
	if (operando1->token == TIPO_CHAR && operando2->token != TIPO_CHAR) exibirErro (source,"tipo char
nao eh compativel com um tipo diferente de char", reader->lexema);
	if (operando1->token == TIPO_INT && operando2->token == TIPO_CHAR) exibirErro (source,"tipo int
nao eh compativel com um tipo char", reader->lexema);
	if (operando1->token == TIPO_FLOAT && operando2->token == TIPO_CHAR) exibirErro (source,"tipo
float nao eh compativel com um tipo char", reader->lexema);
	if (operando1->token == TIPO_INT && operando2->token == TIPO_FLOAT) {
		operando1->token = TIPO_FLOAT;
	}
	if (operando1->token == TIPO_FLOAT && operando2->token == TIPO_INT) {
		operando2->token = TIPO_FLOAT;
		printf ("T%i = (float) %s", Tn, operando2->lexema);
		if (operando2->T_id >= 0) printf ("%i",operando2->T_id);
		printf ("\n");
		strcpy (operando2->lexema, "T");
		operando2->T_id = Tn;
		Tn++;
	}
	if (divisao_ocorreu == 1 && operando1->token == TIPO_INT && operando2->token == TIPO_INT) {
		operando1->token = TIPO_FLOAT;
		printf ("T%i = (float) %s", Tn, operando1->lexema);
		if (operando1->T_id >= 0) printf ("%i",operando1->T_id);
		printf ("\n");
		strcpy (operando1->lexema, "T");
		operando1->T_id = Tn;
		Tn++;
	}
}

void verificarOperadoresEmUmaAtribuicao (FILE*source, TNo *reader, TNo *operando1, TNo *operando2) {
	if (operando1->token == TIPO_CHAR && operando2->token != TIPO_CHAR) exibirErro (source,"tipo char
so pode receber valores do tipo char", reader->lexema);
	if (operando1->token == TIPO_INT && operando2->token != TIPO_INT) exibirErro (source,"tipo int so
pode receber valores do tipo int", reader->lexema);
	if (operando1->token == TIPO_FLOAT && operando2->token == TIPO_CHAR) exibirErro (source,"tipo
float so pode receber valores do tipo float", reader->lexema);
	if (operando1->token == TIPO_FLOAT && operando2->token == TIPO_INT) {
		operando2->token = TIPO_FLOAT;
		printf ("T%i = (float) %s", Tn, operando2->lexema);
		if (operando2->T_id >= 0) printf ("%i",operando2->T_id);
		printf ("\n");
		strcpy (operando2->lexema, "T");
		operando2->T_id = Tn;
		Tn++;
	}
	printf ("%s = %s", operando1->lexema, operando2->lexema);
	if (operando2->T_id >= 0) printf ("%i", operando2->T_id);
	printf ("\n");
}
TNo EXPRESSAO_ARITMETICA (FILE * source, TNo *reader) {
	TNo operando1, operando2;
	char operacao[50];

	operando1.token = NIL; operando2.token = NIL;
	operando1 = TERMO (source, reader);
	strcpy(operacao,reader->lexema);
	operando2 = EXPRESSAO_ARITMETICA_LINHA (source, reader);

	if (operando2.token == TIPO_CHAR || operando2.token == TIPO_INT || operando2.token == TIPO_FLOAT)
	{
		verificarOperadores (source, reader, &operando1, &operando2, 0);
		operando1 = geradorDeCodigoIntermediario (operando1, operacao, operando2);
	}
	return operando1;
}
```

```
TNo EXPRESSAO_ARITMETICA_LINHA (FILE * source, TNo *reader) {
	TNo operando1, operando2;
	char operacao[50];

	operando1.token = NIL; operando2.token = NIL;
	if (reader->token == SOMA || reader->token == SUBTRACAO) {
		strcpy(operacao,reader->lexema);
		*reader = SCANNER (source);
		operando1 = TERMO (source, reader);
		operando2 = EXPRESSAO_ARITMETICA_LINHA (source, reader);
	}
	if (operando2.token == TIPO_CHAR || operando2.token == TIPO_INT || operando2.token == TIPO_FLOAT)
{
		verificarOperadores (source, reader, &operando1, &operando2, 0);
		operando1 = geradorDeCodigoIntermediario (operando1, operacao, operando2);
	}
	return operando1;
}

TNo TERMO (FILE * source, TNo *reader) {
	TNo operando1, operando2;
	int divisao_ocorreu = 0;
	char operacao[50];

	operando1.token = NIL; operando2.token = NIL;
	operando1 = FATOR (source, reader);
	while(reader->token == MULT || reader->token == DIVISAO){
		strcpy(operacao,reader->lexema);
		if (reader->token == DIVISAO) divisao_ocorreu = 1;
		*reader = SCANNER (source);
		operando2 = FATOR (source, reader);
		verificarOperadores (source, reader, &operando1, &operando2, divisao_ocorreu);
		operando1 = geradorDeCodigoIntermediario (operando1, operacao, operando2);
		divisao_ocorreu = 0;
	}
	return operando1;
}
```

```
TNo FATOR (FILE * source, TNo *reader) {
	TNo operando;
	TNo *verify;

	operando.T_id = -1;
	if (reader->token == ID || reader->token == VALOR_INT || reader->token == VALOR_FLOAT || reader->token == VALOR_CHAR)
	{
		if(reader->token == ID)
		{
			verify=buscar_ID_em_todos_escopos(reader->lexema);
			if (verify == NULL) exibirErro (source,"variavel nao declarada em nenhum escopo", reader->lexema);
			else {
				strcpy (operando.lexema, verify->lexema);
				operando.token = verify->token;
			}
		}
		else
		{
			strcpy (operando.lexema, reader->lexema);
			switch (reader->token)
			{
				case VALOR_CHAR: operando.token = TIPO_CHAR; break;
				case VALOR_INT: operando.token = TIPO_INT; break;
				case VALOR_FLOAT: operando.token = TIPO_FLOAT; break;
			}
		}
		*reader = SCANNER (source);
		return operando;
	}
	else if (reader->token == ABRE_PARENTESES) {
		*reader = SCANNER (source);
		operando = EXPRESSAO_ARITMETICA (source, reader);
		if (reader->token != FECHA_PARENTESES) exibirErro (source,"token FECHA PARENTESES nao encontrado", reader->lexema);
		*reader = SCANNER (source);
		return operando;
	}
	else { // Devido a Fator aceitar palavra vazia é preciso printer erro aqui
		exibirErro (source,"Expressao aritmetica mal formada", reader->lexema);
	}
}
```

O Autor

L. R. C. Farias é graduado em Ciência da Computação pela Universidade Católica de Pernambuco (UNICAP) (2016), onde recebeu láurea acadêmica. Atualmente, é aluno de mestrado em Ciência da Computação pela Universidade Federal de Pernambuco (UFPE), desenvolvendo pesquisas na área de inteligência computacional e otimização.

Atuou como monitor das disciplinas de Introdução à Programação, Arquitetura e Organização de Computadores II e Compiladores na UNICAP, orientando projetos de estudantes e unindo teoria e prática. Também participou de projetos de iniciação científica na área de lógica fuzzy e algoritmos genéticos aplicados a problemas de engenharia química, com foco na modelagem e otimização de processos.

Seus principais interesses incluem computação evolucionária, modelagem de problemas de otimização e o desenvolvimento de algoritmos inspirados na natureza.

Caso deseje entrar em contato, seguem meus canais oficiais:

- **E-mail**: lrcf@cin.ufpe.br
- **Currículo Lattes**: http://lattes.cnpq.br/1167531021165184
- **Google Scholar**: https://scholar.google.com.br/citations?user=I08lnEsAAAAJ&hl