1ª Tarefa - PCS2056
Linguagens e Compiladores

Professor: Ricardo Luis de Azevedo Rocha

Grupo:
Filipe Morgado Simões de Campos 5694101
Rafael Barbolo Lopes 5691262

Assunto 1: Introdução e terminologia básica

Tarefas:

1. Utilizar o compilador de exemplo escrito em C e escrever um pequeno texto identificando cada componente lógico do compilador no código utilizado:

Analisador léxico

Existe um *parser* no compilador que utiliza o analisador léxico. O arquivo **parser.i** usa o método *get_next_token* que está implementado no arquivo **lex.c**. Este arquivo possui a implementação do analisador léxico; ele lê o código fonte do terminal e identifica o caracter que foi escrito, gerando um token que pode representar um dígito (algarismo entre 0 e 9), um EOF (indicador de fim de caracteres de entrada do código fonte) ou um outro caracter.

```
void get_next_token(void) {
    int ch;

    /* get a non-layout character: */
    do {
        ch = getchar();
        if (ch < 0) {
            Token.class = EoF; Token.repr = '#';
            return;
        }
    } while (Layout_char(ch));

    /* classify it: */
    if ('0' <= ch && ch <= '9') {Token.class = DIGIT;}
    else {Token.class = ch;}

    Token.repr = ch;
}
```

Figura 1 - Método get_next_token do arquivo lex.c

Uma observação interessante é que o Token foi implementado como uma variável global e não como uma variável local trocada entre os métodos do compilador.

Analisador sintático

O analisador sintático também é requisitado pelo **parser.i**. Ele está implementado nos arquivo **parserbody.i** através dos métodos *Parse_expression* e *Parse_operator*. O primeiro é responsável por criar uma expressão lógica com os dígitos e operadores identificados pelo analisador léxico.

```
static int Parse_operator(Operator *oper) {
    if (Token.class == '+') {
        *oper = '+'; get_next_token(); return 1;
    }
    if (Token.class == '*') {
        *oper = '*'; get_next_token(); return 1;
    }
    return 0;
}
static int Parse_expression(Expression **expr_p) {
    Expression *expr = *expr_p = new_expression();

    /* try to parse a digit: */
    if (Token.class == DIGIT) {
        expr->type = 'D'; expr->value = Token.repr - '0';
        get_next_token();
        return 1;
    }

    /* try to parse a parenthesized expression: */
    if (Token.class == '(') {
        expr->type = 'P';
        get_next_token();
        if (!Parse_expression(&expr->left)) {
            Error("Missing expression");
        }
        if (!Parse_operator(&expr->oper)) {
            Error("Missing operator");
        }
        if (!Parse_expression(&expr->right)) {
            Error("Missing expression");
        }
        if (Token.class != ')') {
            Error("Missing )");
        }
        get_next_token();
        return 1;
    }

    /* failed on both attempts */
    free_expression(expr); return 0;
}
```

Figura 2 - parserbody.i

As expressões que o analisador sintático gera são do tipo: D | (EOE), em que D é um dígito (algarismo entre 0 e 9), E é uma expressão válida e O é uma operação de soma ou multiplicação, ou seja + ou *.
O código de entrada possuí parêntesis que o analisador sintático utiliza para gerar a expressão. Esses parêntesis são importantes para validar o código fonte de entrada.

Gerador de código objeto

Existem geradores de código objeto no projeto SimpleCompiler que processam recursivamente a expressão gerada pelo analisador sintático e imprimem o código objeto final. Esses geradores são usados separadamente, dependendo da chamada que o compilador recebe.

O gerador localizado no arquivo **codegen.c** cria um código objeto em assembly. Os geradores localizados nos arquivos **vnaivcg.c** e **vnaivcg2.c** são geradores triviais que criam código em C.

```
static void Code_gen_expression(Expression *expr) {
    switch (expr->type) {
    case 'D':
        printf("PUSH %d\n", expr->value);
        break;
    case 'P':
        Code_gen_expression(expr->left);
        Code_gen_expression(expr->right);
        switch (expr->oper) {
        case '+': printf("ADD\n"); break;
        case '*': printf("MULT\n"); break;
        }
        break;
    }
}
```

Figura 3 - Função presente em codegen.c

Outros comentários

Existem também interpretadores que permitem que o código fonte original seja executado em tempo de execução. Esses interpretadores estão localizados nos arquivos **interpr.c** e **itinter.c**.

Não foram localizados os componentes lógicos: **analisador semântico**, **gerador de código intermediário** e **otimizador de código**.