Compilador fase 2: Análise Semântica e Geração de Código[1]

Na **fase 1** do projeto foi implementado um compilador que fazia a leitura de um **arquivo-fonte** na linguagem **Pascal+-** e realizava a análise léxica e sintática do programa fonte. O objetivo dessa fase é implementar as fases de **Análise Semântica** e **Geração de Código Intermediário** do compilador, a implementação dessa fase será baseada na implementação realizada na **fase 1**, caso você não tenha implementado a **fase 1**, para essa fase, você **terá que implementar tanto a fase 1 quanto a fase 2**.

Análise Semântica

Na **Análise Semântica** o seu compilador deverá verificar se as construções sintáticas da fase anterior estão coerentes, o compilador implementado na fase anterior deve manter as funcionalidades de identificação de erros **léxicos** e **sintáticos,** e adicionalmente, emitir as mensagens de erros semânticos, caso ocorram.

Basicamente o Compilador fará somente a verificação semântica para variáveis em dois momentos:

- **Declaração**: Na seção de declaração de variável **`<declaracao_de_variaveis>`** o compilador deve garantir que os identificadores usados no nome de variável sejam únicos, ou seja, não podendo ter duas variáveis declaradas com o mesmo identificador. Caso aconteça uma repetição de identificador o **compilador deve ser finalizado informando que ocorreu um erro semântico**. Para isso deverá ser implementado uma **minitabela de símbolos** que armazenará as variáveis declaradas (**identificador** e **endereço**), o endereço da variável seria a ordem em que a variável foi declarada, dessa forma a primeira variável tem endereço **`0`**, a segunda endereço **`1`** e assim sucessivamente.

- **Corpo do programa:** As variáveis declaradas na seção de declaração de variáveis podem ser referenciadas nos comandos de atribuição, nas expressões e nas chamadas das funções de entrada e saída. Assim toda vez que uma variável for referenciada no corpo de programa, o compilador deve verificar se a variável foi declarada corretamente, **caso não tenha sido declarada é gerado um erro semântico** explicativo e compilador é finalizado.

Para simplificar a análise semântica e geração de código intermediário, o compilador não precisará realizar a distinção entre expressões inteiras e lógicas, assim, para os testes do compilador, só teremos variáveis e expressões do tipo **`integer`**, consequentemente, não teremos construções do tipo **`25+(x>y)`** e nem atribuição das constantes **`true`** e **`false`** às variáveis, por exemplo: **`var=false`**.

Geração de Código Intermediário

A **Geração de Código Intermediário** será baseada na proposta do livro do professor **Tomasz Kowaltowiski** Implementação de Linguagem de Programação (**Seção 10.3 Análise Sintática e Geração de Código**). A geração das instruções da MEPA será realizada nas funções mutuamente recursivas implementadas no analisador sintático, para tanto basta **imprimir as instruções da MEPA** nas mesmas funções que fazem análise sintática e semântica do compilador. Nos exemplos de geração de código intermediário são apresentadas duas funções: **`proximo_rotulo()`** e **`busca_tabela_simbolos()`**, que é recomendável que você implemente conforme proposto, pois tornam a implementação modularizada, a geração de código mais simples e poderão ser utilizadas questões da prova da disciplina.

Como primeiro exemplo considere a produção abaixo para o comando **`<comando_condicional>`**, conforme visto na gramática da **fase 1 do compilador**.

```
<comando_condicional> ::= if <expressao> “:” <comando> [elif <comando>]
```

[1] **Importante**: A especificação desse trabalho pode sofrer modificações de acordo com discussões que tivermos em sala de aula.

A implementação da função correspondente para gerar código intermediário para produção do **<comando_condicional>** segue abaixo:

```
void comando_condicional(){
   int L1 = proximo_rotulo();
   int L2 = proximo_rotulo();
   consome(IF);
   expressao();
   consome(DOIS_PONTOS);
   printf("\tDSVF L%d\n",L1);
   comando();
   printf("\tDSVS L%d\n",L2);
   printf("L%d:\tNADA\n",L1);
   if(lookahead == ELFI){
       consome(ELIF);
       comando();
   }
   printf("L%d:\tNADA\n",L2);
}
```

Considere que a função **proximo_rotulo()** retorna o valor do próximo rótulo consecutivo positivo (por exemplo **1**, **2**, **3**, ...). **Importante:** Como todas as funções são recursivas, deve-se tomar o cuidado na atribuição das variáveis locais que irão receber o retorno da função e/ou a ordem de chamadas da função **proximo_rotulo()**.

Como explicado acima, só vamos considerar variáveis do tipo **integer**, assim a produção **<fator>** só precisa gerar código para **identificador**, **numero** e expressão entre parênteses, como abaixo:

<fator> ::= **identificador** | **numero** |“(“ <expressao> “)”

Abaixo segue o segundo exemplo de implementação de geração de código intermediário:

```
void fator(){
    if(lookahead == IDENTIFICADOR){
        int endereco = busca_tabela_simbolos(InfoAtomo.atributo_ID);
        printf("\tCRVL %d\n",endereco);
        consome(lookahead);
    }
    else if(lookahead == NUMERO){
        printf("\tCRCT %d\n", InfoAtomo.atributo_numero);
        consome(lookahead);
    }else{
        consome('(');
        E();
        consome(')');
    }
}
```

Nesse exemplo utilizamos a função **busca_tabela_simbolos()**, que recebe como parâmetro o atributo **atributo_ID** do átomo corrente (um vetor de caracteres) e retorna o endereço da variável armazenado na minitabela de símbolos, caso o identificador **não conste da tabela de símbolos a função gera um erro semântico**, que é informado na tela do computador, em seguida o processo de compilação é finalizado.

Lembre-se que a variável **`InfoAtomo`** é uma variável global do tipo **`TInfoAtomo`** e é atualizada na função **`consome()`** e armazena os atributos do **átomo** reconhecido no **analisador léxico**.

Execução do Compilador – fase 2

A seguir temos um programa em **Pascal+-** que calcula o fatorial de um número informado ao programa, considere que o programa **`fatorial`** não possui erros léxicos e sintáticos.

```
 1 {-
 2 programa calcula o fatorial de um numero lido
 3 -}
 4 programa fatorial;
 5    integer fat,num,cont;
 6 begin
 7    read(num);
 8    set fat to 0b1;
 9    for cont of 0b10 to num:
10       set fat to fat * cont;
11
12    write(fat) # imprime o fatorial calculado
13 end.
```

Na saída do **compilador**, caso o programa não tenha erro léxico sintático ou semântica, deve ser impresso, no monitor do computador, a **tradução do código fonte para linguagem MEPA e a tabela de símbolos com as variáveis declaradas com seus endereços**.
Se o código fonte tiver erros, deve ser gerada uma mensagem informativa imprimindo o número da linha no arquivo fonte, onde o primeiro erro ocorre, e qual o erro foi identificado. **Não é necessário imprimir os átomos reconhecidos na fase léxica.**

Saída do compilador:

```
    INPP
    AMEM 3
    LEIT
    ARMZ 1
    CRCT 1
    ARMZ 0
    CRCT 2
    ARMZ 2
L1: NADA
    CRVL 2
    CRVL 1
    CMEG
    DVSF L2
    CRVL 0
    CRVL 2
    MULT
    ARMZ 0
    CRVL 2
    CRCT 1
    SOMA
    ARMZ 2
    DSVS L1
L2: NADA
    CRVL 0
    IMPR
    PARA
```

```
TABELA DE SIMBOLOS
fat  | Endereco: 0
num  | Endereco: 1
cont | Endereco: 2
```

Observações importantes:

O programa deve estar bem documentado e pode ser feito em grupo de até **2 alunos**, não esqueçam de colocar o **nome dos integrantes** do grupo no arquivo fonte do trabalho e sigam as **Orientações para Desenvolvimento de Trabalhos Práticos** disponível no **Moodle**.

O trabalho será avaliado de acordo com os seguintes critérios:

- Funcionamento do programa, caso programa apresentarem ***warning*** ao serem compilados serão penalizados. Após a execução o programa deve finalizar com **retorno igual a 0**;
- Caso o **programa não compile ou não execute** será penalizado com a **NOTA 0.0**;
- O trabalho deve ser desenvolvido na **linguagem C** e será testado usando o compilador do **MinGW** com **VSCode**, para configurar sua máquina no Windows acesse: https://www.doug.dev.br/2022/Instalacoes-e-configuracoes-para-programar-em-C-usando-o-VS-Code/
- Compile seu programa com o seguinte comando abaixo, considere que o programa fonte do seu compilador seja compilador.c:
  `gcc -g -Og -Wall compilador.c -o compilador`

- O quão fiel é o programa quanto à descrição do enunciado, principalmente ao formato de do **arquivo de entrada**;
- Clareza e organização, programas com código confuso (linhas longas, variáveis com nomes não-significativos, ...) e desorganizado (sem indentação, sem comentários, ...) também serão levados em consideração na correção.

- Entrega de um arquivo **Readme.txt** explicando até a parte do trabalho que foi feito.