Relatório Compiladores

Felipe Gabriel Comin Scheffel RA117306
Prof. Felippe Fernandes

1. Introdução

Este relatório foi desenvolvido durante o desenvolvimento do projeto para a disciplina de Compiladores. O projeto teve como objetivo criar um compilador utilizando C e ferramentas externas para uma linguagem fictícias. Os pontos a serem desenvolvidos foram a Análise Léxica, Análise Sintática, Construção da Árvore de Sintaxe, Construção da tabela de símbolos, Análise Semântica e Geração de Código intermediário.

---

2. Especificação da Linguagem

A linguagem fictícia especificada é uma linguagem de alto nível, semelhante ao Pascal em sua estrutura de declarações, mas com blocos e expressões semalhantes a C.

Estrutura Principal:

```
Declarações globais
Declarações de funções
Bloco principal de execução
```

- Não é possível declarar funções dentro do bloco principal, e declarações de variáveis devem estar sempre no topo de seu escopo (global, de função ou de bloco).
- Comandos devem ser finalizados com o delimitor `;`. Declarações de funções e blocos de execução são exceções.
- O bloco principal deve estar após todos as definições de variáveis globais e funções, e deve ser delimitado, assim como todos os outros blocos, por chaves `{}`.

Tipos de Dados e Variáveis:

- A linguagem suporta os seguintes tipos de dados:
  - **int**: Números inteiros.
  - **float**: Números de ponto flutuante.
  - **char**: Caracteres individuais.
  - **string**: Sequências de caracteres.
  - **boolean**: Valor booleano.
  - **array**: Arrays unidimensionais de tamanho fixo de um dos tipos já mencionado.
- As variáveis podem ser declaradas utilizando a seguinte sintaxe:

```
var nome: tipo;
```

- Ou declarando mais de uma variável do mesmo tipo:

```
var nome1, nome2: tipo;
```

- Declarando e acessando array:

```
var nome: array [tamanho] of tipo;
nome[indice];
```

- Funções podem ser definidas com a seguinte estrutura:

```
function nome_da_funcao (param1, param2: tipo1; param3: tipo2): tipo_de_retorno
{
        // Corpo da função
}
```

- As chamadas devem seguir exatamente a assinatura da função na questão de tipo e quantidade de parâmetros.

```
function mdc (a, b: int): int
{
    ...
}

{
    var x, y, result: int;
    result = mdc(x, y);
}
```

- O tipo de retorno pode ser omitido, e a função se torna então um procedimento. Os parâmetros também são opcionais.

Estruturas de Controle:

- A linguagem suporta as estruturas de controle condicional `if-else` e de repetição `while`. O corpo das estruturas podem ser uma expressão simples ou um bloco composto.

```
if (condicao)
    corpo

if (condicao)
    corpo
else
    corpo

while (condicao)
    corpo
```

Operações:

- São suportadas as operações matemáticas básicas (adição +, subtração -, multiplicação * e divisão /), operadores de sinal (+, -) e módulo (%), tanto para números inteiros quanto para números de ponto flutuante.
- Realizar uma operação matemática com pelo menos um float e quantos int ao mesmo tempo sempre terá como resultado um float.
- São suportadas as operações relacionais (igual ==, diferente !=, maior >, menor <, maior igual >=, menor igual <=) entre númericos.
- Entre booleanos, além dos operadores igual e diferente, podem ser utilizados os operadores and, or e not.
- Não é possível realizar operações com strings e chars.

---

3. Ferramentas Utilizadas

Para o desenvolvimento da linguagem fictícia, foram utilizadas as seguintes ferramentas:

- **Linguagem de Programação:** C
- **Ferramentas de Compilação:** O código-fonte da linguagem foi compilado usando um compilador C padrão.
- **Ambiente de Desenvolvimento:** Um editor de texto simples foi utilizado para escrever o código-fonte da linguagem.

---

4. Análise Léxica

A análise léxica foi realizada com flex, definindo regex para criar tokens para os identificadores, literais, operadores de mais de um caracter e keywords. Os delimitadores são enviados como texto, e comentários são todos ignorados.

---

5. Análise Sintática

A análise sintática foi feita com Bison para definir a gramática. A grande maioria dos símbolos não terminais recebe um valor de nó, definidos em um header. Foram utilizadas as ações, que são disparadas ao encontrar regras sendo aplicadas no código, para gerar os nós da arvore de sintaxe. Ela é impressa ao fim da análise.

---

6. Análise Semântica

A análise semântica é feita a partir da árvore sintática. Percorrendo a partir do nó inicial, funções e variáveis declaradas são alocadas para buckets em uma tabela hash. Então, é realizada a verificação de tipo (por exemplo, condições de IF devem ser booleanas), de declaração (impedindo chamada de funções e acesso

a variáveis não declaradas) e algumas outras verificações, como expressões de retorno fora de funções.

Referências

Fora o material das aulas, foram utilizados apenas os livros Compiladores - Princípios, Técnicas e Ferramentas, e, principalmente Modern Compiler Implementation em C. Além disso, foram utilizadas as documentações do Flex e Bison.

Dificuldades e Facilidades

Infelizmente não foi possível realizar todos os pontos requeridos na especificação, principalmente devido ao prazo. Não foi implementado suporte para classes, e nem as funções de input e output (a ideia inicial era implementá-las em forma de biblioteca, ao fim do desenvolvimento). A análise semântica também não foi totalmente finalizada: foram definidas todas as funções de verificação para até onde foi desenvolvida a gramática, mas não foi terminado o processo de debugging, com alguns casos de Segmentation Fault na busca de registros ainda ocorrendo. Por fim, a geração de código não passou das etapas de pesquisa, sem qualquer avanço na implementação.

A maior dificuldade foi encontrar material atualizado para utilizar como referência. Os livros são focados muito mais na teoria, com poucos exemplos de aplicações práticas de, por exemplo, a geração da AST e da tabela de símbolos. Mesmo o livro do Appel, que tem uma abordagem mais prática, apresenta código difícil de decifrar e com poucas implementações concretas.
