Relatório Técnico MinC

Francisco Castro Del'Gaudio Junior

4 de junho de 2025

Sumário

1 Descrição do Funcionamento do Software **2**

2 Descrição da Linguagem **3**

3 Tipos de Erros Tratáveis **5**

4 Documentação dos Principais Métodos/Funções **7**

5 Processo de Construção do Software **9**

6 Referências Bibliográficas **10**

1 Descrição do Funcionamento do Software

O software desenvolvido realiza a análise léxica de programas escritos na linguagem `MinC`. Ele foi implementado com o auxílio da ferramenta `Flex`, responsável por gerar o analisador léxico a partir de um conjunto de expressões regulares definidas no arquivo `main.l`. O programa identifica e classifica os tokens do código-fonte e organiza os resultados em listas de palavras reservadas e símbolos.

Etapas de Execução do Software

A seguir, estão listados os passos para compilar e executar o analisador léxico da linguagem `MinC`:

1. Abra o terminal na pasta onde está localizado o arquivo `main.l` (arquivo contendo as regras léxicas escritas em Flex).
2. Gere o código-fonte em C a partir do arquivo léxico utilizando o Flex:

   ```
   flex main.l
   ```

   Isso criará automaticamente um arquivo chamado `lex.yy.c`.
3. Compile o arquivo gerado com o `gcc`:

   ```
   gcc lex.yy.c lista.c -o main.exe
   ```

   Isso produzirá o executável `main.exe` (ou `./main` em sistemas Unix).
4. Execute o programa:

   ```
   ./main.exe
   ```

5. Ao iniciar, o programa solicitará o nome de um arquivo de entrada contendo o código a ser analisado (por exemplo, `exemplo.minc`).
6. Após a análise, será exibido um menu com opções:
   - `0` - Sair;
   - `1` - Imprimir lista de palavras reservadas;
   - `2` - Imprimir lista de símbolos;
   - `3` - Abrir outro arquivo.

Funcionamento Geral

O analisador percorre o conteúdo do arquivo de entrada linha por linha e reconhece os seguintes elementos:

- Palavras-chave, operadores e delimitadores;
- Identificadores válidos;
- Números inteiros e reais;
- Strings e caracteres;
- Comentários de linha ou bloco;
- Macros pré-processadas.

Cada token identificado é armazenado em uma estrutura de lista (implementada via TAD `Lista`), permitindo posterior exibição ao usuário.

Tratamento de Erros

Durante a análise, o software também reconhece e trata erros léxicos, emitindo mensagens informativas com a linha e coluna do problema. Entre os erros tratados, destacam-se:

- Identificadores iniciados incorretamente com números;
- Números reais malformados;
- Strings e caracteres não finalizados corretamente.

2 Descrição da Linguagem

A linguagem `MinC` possui uma estrutura léxica inspirada na linguagem C, com suporte a tipos primitivos, operadores, estruturas de controle, entrada e saída padrão, strings, caracteres, macros, entre outros. A análise léxica da linguagem é realizada com base em expressões regulares que descrevem os padrões válidos para cada tipo de token.

Expressões Regulares Utilizadas

Abaixo estão listadas as expressões regulares utilizadas no analisador léxico, junto com sua descrição:

- **Palavras-chave (PALAVRA_CHAVE)**:

```
(int|float|double|char|void|if|else|for|while|switch|
 case|cout|cin|endl|return)
```

  Representam as instruções e comandos reservados da linguagem.

- **Identificadores (IDENTIFICADOR)**:

```
[_a-zA-Z]+[_a-zA-Z0-9]*
```

Nomes válidos de variáveis, funções e identificadores em geral.

- **Números inteiros (NUMERO_INTEIRO)**:

```
[\+\-]?[0-9]+
```

Valores inteiros, com ou sem sinal.

- **Números reais (NUMERO_REAL)**:

```
{NUMERO_INTEIRO}"."[0-9]+
```

Valores reais compostos por parte inteira e parte decimal.

- **Operadores aritméticos (OPERADOR_ARITMETICO)**:

```
[+\-*/%]
```

Operações matemáticas básicas.

- **Operadores relacionais (OPERADOR_RELACIONAL)**:

```
(==|<=|>=|!=|>|<)
```

Comparações entre valores.

- **Operadores lógicos (OPERADOR_LOGICO)**:

```
(|||&&|!)
```

Operações booleanas.

- **Operador de atribuição (OPERADOR_ATRIBUICAO)**:

```
=
```

- **Delimitadores (DELIMITADOR)**:

```
[{}()[\]]
```

Símbolos usados para delimitar blocos, parâmetros e expressões.

- **Pontuação (PONTUACAO)**:

  ```
  (;|,|\.)
  ```

  Usada para separar instruções e elementos.

- **Strings (STRING)**:

  ```
  \"([^\\\"]|\\.)*\"
  ```

  Sequência de caracteres entre aspas duplas, com suporte a escape.

- **Caracteres (CARACTERE)**:

  ```
  \'.\'
  ```

  Um único caractere entre aspas simples.

- **Comentários (COMENTARIO)**:

  ```
  (\/\/[^\n]*|\/\*([^*]|\*+[^*/])*\*+\/)
  ```

  Comentários de linha (//) ou de bloco (/* */).

- **Macros (MACROS)**:

  ```
  \#[a-zA-Z_]+
  ```

  Diretivas do pré-processador, como `#include` ou `#define`.

3 Tipos de Erros Tratáveis

Durante a execução do analisador léxico da linguagem `MinC`, o sistema é capaz de detectar diversos tipos de erros relacionados à formação incorreta de tokens. Esses erros são identificados com base em padrões específicos de expressões regulares, e quando encontrados, geram mensagens informativas contendo o tipo de erro, a linha e a coluna onde ele ocorreu.

A seguir, estão descritos os principais tipos de erros léxicos tratados pelo software:

1. Identificador Malformado

Padrão reconhecido:

```
{NUMERO_INTEIRO}{IDENTIFICADOR}
```

Esse erro ocorre quando um identificador começa com um número, o que é inválido segundo as regras da linguagem. Por exemplo:

```
123abc
```

Mensagem gerada:

```
Erro lexico [linha X, coluna Y] IDENTIFICADOR malformado: 123abc
```

2. Número Real Malformado

Padrão reconhecido:

```
{NUMERO_INTEIRO}"."[^0-9|^\n]+
```

Indica que um ponto decimal foi utilizado sem um número válido após ele. Exemplo de entrada inválida:

```
42.a
```

Mensagem gerada:

```
Erro lexico [linha X, coluna Y] NUMERO_REAL malformado: 42.a
```

3. String Não Finalizada

Padrão reconhecido:

```
\"([^\\\"\n]|\\.)*(\n|$)
```

Ocorre quando uma cadeia de caracteres é iniciada com aspas duplas, mas não é corretamente finalizada antes do fim da linha ou do arquivo. Exemplo:

```
"texto sem fechamento
```

Mensagem gerada:

```
Erro lexico [linha X, coluna Y] STRING malformado: "texto sem fechamento
```

4. Caractere Malformado

Padrão reconhecido:

```
\'([^\\'\n]|\\.)*(\n|$)
```

Esse erro ocorre quando um literal de caractere é iniciado por aspas simples, mas não contém exatamente um caractere válido ou não é encerrado corretamente. Exemplo:

```
'a
```

Mensagem gerada:

```
Erro lexico [linha X, coluna Y] CARACTERE malformado: 'a
```

5. Símbolos Desconhecidos

O analisador ignora ou apenas contabiliza o avanço de coluna para símbolos não reconhecidos por nenhuma regra válida. Embora não gere uma mensagem de erro diretamente, esse comportamento permite que o sistema continue analisando o restante do código sem interrupções.

4 Documentação dos Principais Métodos/Funções

A seguir estão descritos os principais métodos e funções utilizados no código-fonte do analisador léxico da linguagem `MinC`. Cada função está acompanhada de sua finalidade, parâmetros de entrada e comportamentos principais.

1. `Inserir`

- **Localização:** `lista.c`
- **Protótipo:**

```
Lista* Inserir(Lista *l, Classe classe, char *lexema, int linha, int coluna);
```

- **Descrição:** Insere um novo elemento na lista ligada, evitando duplicatas. Cada nó armazena a classe léxica, lexema, linha e coluna do token.
- **Parâmetros:**
  - `l`: ponteiro para a lista atual.
  - `classe`: tipo do token (por exemplo, `PALAVRA_CHAVE`, `IDENTIFICADOR`).
  - `lexema`: texto do token reconhecido.
  - `linha`, `coluna`: posição do token no código-fonte.
- **Retorno:** Ponteiro para a lista atualizada.

2. `ImprimirLista`

- **Localização:** `lista.c`
- **Protótipo:**

```
void ImprimirLista(Lista *l);
```

- **Descrição:** Percorre recursivamente a lista e imprime os tokens nela contidos. Cada item exibido mostra a classe do token e seu lexema.
- **Parâmetros:**
  - `l`: ponteiro para a lista de tokens.
- **Retorno:** Nenhum (procedimento).

3. `ClasseParaString`

- **Localização:** `lista.c`
- **Protótipo:**

```
const char* ClasseParaString(Classe classe);
```

- **Descrição:** Converte o valor enumerado da classe léxica em uma string legível para exibição. Utilizado pela função `ImprimirLista`.
- **Parâmetros:**
  - `classe`: valor da enumeração `Classe`.
- **Retorno:** Ponteiro para string com o nome da classe.

4. `main`

- **Localização:** `main.l`
- **Protótipo:**

```
int main();
```

- **Descrição:** Função principal do programa. Solicita ao usuário um nome de arquivo, realiza a análise léxica e oferece um menu interativo para exibir os tokens reconhecidos.
- **Fluxo principal:**
  1. Solicita o nome de um arquivo de entrada.
  2. Executa `yylex()` para realizar a análise léxica.
  3. Exibe opções ao usuário para imprimir listas ou carregar outro arquivo.
  4. Garante que o programa continue operando até que o usuário escolha encerrar.

5. `yylex`

- **Localização:** Gerado por `flex` a partir de `main.l`
- **Descrição:** Função de varredura principal. Reconhece padrões com base nas expressões regulares definidas e executa ações específicas, como inserir tokens ou emitir erros léxicos.
- **Ações principais:**
  - Chamada de `Inserir` para armazenar tokens válidos.
  - Impressão de mensagens para erros léxicos com linha e coluna.
  - Atualização dos contadores de linha e coluna para rastreamento preciso.

6. `yywrap`

- **Localização:** `main.l`
- **Protótipo:**

```
int yywrap();
```

- **Descrição:** Função obrigatória em scanners gerados pelo Flex. Indica o fim da entrada ao retornar 1. Neste projeto, está implementada de forma simples para encerrar o `yylex()`.

5 Processo de Construção do Software

O processo de construção do analisador léxico da linguagem `MinC` envolveu a utilização de ferramentas bem estabelecidas para o desenvolvimento de compiladores, com foco na automação da geração de código e facilidade de edição e testes.

Ambiente de Desenvolvimento

- **IDE utilizada:** Visual Studio Code (VS Code)
- **Compilador:** MinGW (Minimalist GNU for Windows)
- **Sistema operacional:** Windows 10

O `MinGW`, utilizado neste projeto, já vem com o compilador `gcc` e com o `Flex` pré-instalado, o que permite a geração e compilação do código diretamente via terminal, sem necessidade de instalação adicional de ferramentas separadas.

Ferramentas e Bibliotecas Utilizadas

- **Flex:** Utilizado para a geração automática do analisador léxico a partir do arquivo `main.l`, onde estão definidas todas as expressões regulares e regras de análise.
- **GCC:** Utilizado para compilar o código gerado pelo `Flex` (`lex.yy.c`) e os arquivos auxiliares (`lista.c`, `lista.h`).
- **Bibliotecas padrão da linguagem C:**
  - `stdio.h`: entrada e saída padrão.
  - `stdlib.h`: alocação de memória e controle de fluxo.
  - `string.h`: manipulação de strings.

Configuração e Compilação do Projeto

O projeto é composto por três arquivos principais:

- `main.l` — contém as regras léxicas do analisador (em Flex).
- `lista.h` — definição da estrutura de dados utilizada para armazenar tokens.
- `lista.c` — implementação das funções de manipulação da lista.

Esses comandos podem ser executados diretamente no terminal integrado do Visual Studio Code ou no prompt do Windows.

6 Referências Bibliográficas

- OpenAI. *ChatGPT* (mai. 2025). Assistente utilizado para apoio técnico e redação deste relatório. Disponível em: `https://chat.openai.com/`