Compiladores – 2019/2

Trabalho Prático T1

29 de agosto de 2019

1 Objetivo

O objetivo deste trabalho é a criação de um analisador léxico (*scanner*) para a linguagem C-Minus, como primeiro componente do *front-end* de um compilador.

2 A Linguagem C-Minus

Definimos aqui uma linguagem de programação denominada C-Minus, que é uma linguagem apropriada para um projeto de compilador por ser mais complexa que a linguagem TINY (vista nos laboratórios), pois inclui funções e vetores.

A linguagem C-Minus é essencialmente um subconjunto de C, mas com muitas simplificações, o que justifica o seu nome. Uma lista (não-exaustiva) de diferenças e semelhanças para o C é dada abaixo:

- Não existe um pré-processador de macro.
- O único tipo de dado de variáveis é `int`.
- Todas as variáveis devem ser declaradas no início de cada bloco. Não existem variáveis globais.
- Só é possível declarar uma variável por linha, sem inicialização.
- Existem constantes booleanas e *strings*, mas estas não podem ser guardadas em variáveis.
- Não existe `printf`. No lugar, temos `write`, uma função que exibe uma *string* e `output`, uma função que exibe um inteiro.
- A entrada de dados é feita pela função `input`.
- Os comandos `if` e `while` são como em C, com a diferença que todos os comandos aninhados (internos ao `if` ou `while`) devem ser colocados em blocos demarcados por `{` e `}`.
- Não existe o comando `for`.
- Vetores só podem ter uma dimensão.
- Um programa é uma sequência de funções, sendo que não há declaração de cabeçalhos e a última função deve ser sempre a `main`, declarada com a assinatura `void main(void)`.
- Existem comentários de linha `//` e de blocos `/* ... */`. Os comentários de blocos não podem ser aninhados.

IMPORTANTE: caso haja alguma discrepância entre informações no livro e na especificação, vale o que está escrito aqui.

3 Convenções Léxicas de C-Minus

O seu *scanner* deve obrigatoriamente reconhecer os *tokens* listados na tabela abaixo. A primeira coluna da tabela indica o tipo do *token*, **bem como o que deve ser impresso na tela** quando o *token* for identificado.

Tipo	Exemplo de lexema
ELSE	`else`
IF	`if`
INPUT	`input`
INT	`int`
OUTPUT	`output`
RETURN	`return`
VOID	`void`
WHILE	`while`
WRITE	`write`
PLUS	`+`
MINUS	`-`
TIMES	`*`
OVER	`/`
LT	`<`
LE	`<=`
GT	`>`
GE	`>=`
EQ	`==`
NEQ	`!=`
ASSIGN	`=`
SEMI	`;`
COMMA	`,`
LPAREN	`(`
RPAREN	`)`
LBRACK	`[`
RBRACK	`]`
LBRACE	`{`
RBRACE	`}`
NUM	`42`
ID	`x17y`
STRING	`"Hello\n"`

Valem as seguintes observações:

- O primeiro bloco da tabela lista as palavras reservadas da linguagem. O seu *scanner* deve reconhecer todas elas.
- O segundo bloco da tabela lista os operadores aritméticos e de comparação. Eles têm o mesmo funcionamento dos operadores em C. O mesmo vale para os caracteres do terceiro bloco.
- O último bloco da tabela lista os tipos de *tokens* que admitem mais de um lexema. Um número é formado por **um ou mais** dígitos. Identificadores começam com uma letra seguida por **zero ou mais** letras ou dígitos. As letras podem ser maiúsculas ou minúsculas e existe diferença entre elas. *Strings* são sequências de caracteres cercados por aspas duplas (`"`). *Strings* não podem ser aninhadas, isto é, não existe escape para o caractere de delimitação de *strings*.
- Deverão ser desconsiderados os caracteres não significativos: espaços, tabulações, e quebra de linha, além de comentários de linha (`//`) e bloco (`/*` e `*/`).

4 Implementando e Testando o Trabalho

Você deve criar um *scanner* que reconhece todos os elementos léxicos da linguagem C-Minus. Para tal, utilize o `flex`. Para cada *token* reconhecido no programa de entrada, exiba no terminal:

1. Número da linha do *token* seguido de `:`.

2. Lexema reconhecido seguido de `->`.
3. Tipo do *token* associado ao lexema. Caso o lexema identificado não esteja associado a nenhum tipo de *token*, exiba `UNKNOWN` (desconhecido).

O trabalho será testado da seguinte maneira:

```
./trab1 < entrada.cm
```

onde `entrada.cm` é o arquivo com o programa na linguagem C-Minus que deve ser analisado.

Exemplo de um arquivo de entrada (`hello.cm`):

```
// First program in C-Minus

void main(void) {
    write("Hello, World!\n");
}
```

Executando `./trab1 < hello.cm`, teremos como resposta no terminal:

```
3: void -> VOID
3: main -> ID
3: ( -> LPAREN
3: void -> VOID
3: ) -> RPAREN
3: { -> LBRACE
4: write -> WRITE
4: ( -> LPAREN
4: "Hello, World!\n" -> STRING
4: ) -> RPAREN
4: ; -> SEMI
5: } -> RBRACE
```

Observações importantes:

- Veja mais arquivos de entrada de exemplo na sala da disciplina no AVA.
- Valide o seu programa usando os gabaritos disponibilizados na sala da disciplina no AVA. Garanta que seu programa gerará exatamente o mesmo resultado do gabarito usando o utilitário `diff` (no Linux). Exemplo:

  ```
  ./trab1 <  in/c01.cm | diff out1/c01.out -
  ```

  ATENÇÃO: seu programa deve produzir como saída (NA TELA) SOMENTE o resultado no formato acima. Nada além disto. Ou seja, nenhuma mensagem e nenhuma formatação adicional deverá ser exibida. Isto é absolutamente necessário porque será usada uma bateria de testes para validação de seu trabalho, que verifica se sua resposta está correta baseado na saída do seu programa.
- **IMPORTANTE**: Ao submeter o seu trabalho para correção, além dos códigos-fonte envie um arquivo `Makefile` que gera como executável para o seu *scanner* um arquivo de nome `trab1`.

5 Regras para Desenvolvimento e Entrega do Trabalho

- **Data da Entrega**: O trabalho deve ser entregue até às 23:55 h do dia 18/09/2019 (quarta-feira). Não serão aceitos trabalhos após essa data.
- **Grupo**: O trabalho é **individual**.

- **Linguagem de Programação e Ferramentas**: Para implementar o seu analisador léxico você deve obrigatoriamente usar o `flex`.
- **Como entregar**: Pela atividade criada no AVA. Envie um arquivo compactado com todo o seu trabalho.
- **Recomendações**: Modularize o seu código adequadamente. Crie códigos claros e organizados. Utilize um estilo de programação consistente. Comente o seu código extensivamente. Não deixe para começar o trabalho na última hora.

6 Avaliação

- O trabalho vale 1.0 ponto na média parcial do semestre.
- Trabalhos com erros de compilação receberão nota zero.
- Caso seja detectado plágio (entre alunos ou da internet), todos os envolvidos receberão nota zero.
- Serão levadas em conta, além da correção da saída do seu programa, a clareza e simplicidade de seu código.
- A critério do professor, poderão ser realizadas entrevistas com os alunos, sobre o conteúdo do trabalho entregue. Caso algum aluno seja convocado para uma entrevista, a nota do trabalho será dependente do desempenho na entrevista. (Vide item sobre plágio, acima.)