Trabalho Prático 2
Segunda, Abril 16, 2018

1 Introdução

Este e os próximos trabalhos práticos consistem na implementação de um compilador para a linguagem Cool. Cada trabalho irá cobrir um componente do compilador: analisador léxico, sintático, semântico e geração de código.

Para este trabalho, você irá implementar o analisador léxico, também chamado de *scanner*, utilizando um *gerador de analisador léxico* chamado flex.

Você irá descrever o conjunto de tokens de Cool em um formato apropriado, e o flex irá gerar o código em C++ que reconhece os tokens dos programas escritos em Cool.

2 Aspectos Técnicos

Você deve seguir a especificação do estrutura léxica do Cool que está disponível na seção 10 e figura 1 do manual do Cool. O seu scanner deve ser robusto, deve funcionar para qualquer entrada possível (tanto programas corretos, quanto incorretos). Por exemplo, ele deve conseguir lidar com EOF (end of file) acontecendo no meio de uma string ou comentário, lidar de maneira correta com strings que são muito grandes etc.

Seu scanner deve terminar de maneira natural. Exceções não tratadas, segfaults etc são inaceitáveis.

2.1 Tratamento de Erro

Todos os erros devem ser enviados ao analisador sintático (parser). Seu analisador léxico (lexer) não deve imprimir nada. Erros são comunicados ao parser retornando o token **ERROR**. Além disso, você deve atribuir a variável **yylval.error_msg** a mensagem de erro correspondente.

(Observação: vocês devem ignorar o token chamado **error** (em minúsculo) para este trabalho; será utilizado apenas na próxima fase - parser).

- Quando um carácter inválido (um que não pode ser o carácter inicial de nenhum token) for encontrado, uma string contendo somente este carácter deve ser retornada como string de erro. Continue a análise léxica a partir do próximo carácter.
- Se houver uma quebra de linha não escapada no meio de uma string, reporte como `‘‘Unterminated string constant’’` e continue a análise na próxima linha — assumiremos que o programador simplesmente esqueceu de fechar a string com quotes (”).
- Quando uma string for muito longa, reporte o erro como `‘‘String constant too long’’`. A análise léxica deve continuar após o final da string.
- Se a string contém caracteres inválidos (por exemplo, null), reporte como `‘‘String contains null character’’`. A análise léxica deve continuar após o final da string.
- O final de uma string é definido como uma das opções abaixo:

1. o começo da próxima linha, se a string tiver uma quebra de linha não escapada dentro dela; ou
2. após o ” que fecha a string

- Se aparecer um EOF enquanto estiver processando um comentário, reporte esse erro como ‘‘`EOF in comment`’’. Mesmo nessa situação, o conteúdo dos comentários deve ser ignorado. De maneira análoga, se um EOF for encontrado enquanto estiver processando uma string, reporte como ‘‘`EOF in string constant`’’.
- Se você encontrar “`*)`” fora de um comentário, reporte esse erro como ‘‘`Unmatched *)`’’, ao invés de gerar os tokens `*` e `)`.
- Se houver um / como último carácter do arquivo, reporte esse erro como ‘‘`backslash at end of file`’’.
- Se houver um carácter null escapado dentro de uma String, você deve reportar esse erro como ‘‘`String contains escaped null character.`’’

2.2 Tabela de Strings

Programas tendem a ter várias ocorrências do mesmo lexema. Por exemplo, um identificador é geralmente referenciado mais de uma vez em um programa (do contrário, ele não seria muito útil!). Para economizar espaço e tempo, uma boa prática é salvar os lexemas em uma *Tabela de String*. Estamos disponibilizando uma implementação de uma tabela de strings.

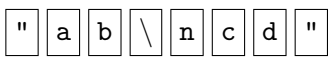
São fornecidas três tabelas de strings: uma tabela para literais inteiros (inttable), uma para literais strings (stringtable) e uma para identificadores (idtable). Para adicionar um elemento a tabela, basta chamar o método add_string, passando como parâmetro a string e seu tamanho. Por exemplo, se quiser armazenar o literal inteiro 124 na tabela de inteiros, chame inttable.add_string(124, 3).

Identificadores especiais como **Object**, **Int**, **Bool**, **String**, **SELF_TYPE** e **self** devem ser tratados como qualquer outro identificador do ponto de vista do analisador léxico (seu tratamento específico será feito em fases posteriores).

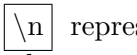
Não teste se os literais de inteiros cabem na sua respectiva representação. Simplesmente crie um símbolo com seu conteúdo textual, independente do seu tamanho.

2.3 Strings

Seu scanner deve converter caracteres escapados para seu valor correto. Por exemplo, se o programador digitar esses oito caracteres:



seu scanner retornará o token **STR_CONST** no qual o valor semântico são os seguintes 5 caracteres:

onde 

representa o código ASCII para a quebra de linha.

Seguindo a especificação na página 15 do manual do Cool, você deve retornar um erro para uma string contendo o carácter literal null. Contudo, a sequência de dois caracteres:

é permitida mas deve ser convertida para o carácter único

 .

2.4 Outras Considerações

- Seu scanner deve manter a variável **curr_lineno**, que indica qual linha do código fonte está sendo lida no momento. Esta funcionalidade irá auxiliar o parser a imprimir mensagens de erros úteis.
- Você deve ignorar o token **LET_STMT**. Ele será utilizado apenas pelo parser (TP3).
- Caso a sua especificação léxica estiver incompleta (algumas entradas não possuem expressões regulares correspondentes), o seu scanner gerado pelo flex poderá ter comportamentos indesejados. Tenha certeza que a sua especificação está completa.
- Cada vez que o scanner é chamado (**cool_yylex** é chamada), ele retorna o próximo token e lexema da entrada. O valor retornado é um inteiro que representa o token que foi lido (e.g. literal inteiro, ponto e virgula, palavra reservada if etc). Os códigos dos tokens estão definidos no arquivo cool-parse.h, que é incluído em cool.flex. O valor semântico do token lido é armazenado na variável **yylval**, que é de tipo **YYSTYPE**. Ele também é definido em cool-parse.h. Os tokens para caracteres individuais (por exemplo, ";" e ",") são representados somente pelo valor inteiro (ASCII) do próprio carácter. Todos os tokens de caracteres individuais estão listados na gramática do Cool, presente no seu manual.
- Para identificadores de classe, identificadores de objetos, inteiros, e strings, o valor semântico deve ser um **Symbol** armazenado no campo **yylval.symbol**. Para constantes binárias, o valor semântico é armazenado no campo **yylval.boolean**. Exceto por erros (veja abaixo), os lexemas para os outros tokens não carregam nenhuma informação interessante.
- Nós disponibilizamos para você uma implementação da Tabela de String, que é discutida em detalhes em *A Tour of the Cool Support Code* e na documentação do código. Por enquanto, você só precisa saber que o tipo das entradas da Tabela de String é **Symbol** e que a função add_string retorna o símbolo que corresponde a string que foi adicionada.
- Quando um erro léxico é encontrado, a rotina **cool_yylex** deve retornar o token **ERROR**. O valor semântico é a string que representa a mensagem de erro, que é armazenada no campo **yylval.error_msg** (perceba que esse campo é uma string qualquer, não um símbolo). Veja a sessão anterior para saber o que colocar nas mensagens de erro.

3 Implementação e Execução

A implementação deverá ser realizada no arquivo cool.flex. Não remova o código que já está presente nele.

Uma vez implementado, execute

```
> make
```

para compilar o código. Isso gerará um arquivo chamado **lexer**, que recebe um arquivo como parâmetro e imprime o resultado do seu analisador léxico executado sobre ele.

Estamos fornecendo uma pasta, examples, com três arquivos de teste: *test1.cl*,*test2.cl* e *test3.cl*. A saída da execução do **lexer** com eles como entrada está nos arquivos *test1.cl.out*, *test2.cl.out* e *test3.cl.out*.

O seu **lexer** deve produzir a mesma saída, caso esteja implementado corretamente.

Note que esses exemplos não abrangem todos os casos possíveis que podem acontecer. Mas são um bom ponto de partida para que criem seus próprios casos de teste.

Vocês podem testar seu analisador léxico linkando ele com as outras fases do compilador. Para isso, a partir de alguma máquina dos laboratórios da graduação (máquinas .grad), crie um diretório de trabalho e execute o seguinte comando:

```
> make -f /home/prof/renato/cool/student/assignments/PA2/Makefile
```

Este comando irá copiar alguns arquivos para o seu diretório. Alguns desses arquivos serão copiados como "read-only" (utilizando links simbólicos para o arquivo original). Você não deve modificar estes arquivos.

O único arquivo que você deve modificar é o cool.flex, que conterá sua implementação do analisador léxico.

Para gerar o lexer, execute o comando:

```
> make lexer
```

Após gerar o lexer, vocês podem executar *mycoolc*. Ele representa o compilador da linguagem Cool utilizando o lexer que vocês desenvolveram. Se tiverem implementado corretamente, o *mycoolc* deveria se comportar de maneira idêntica ao *coolc* que foi utilizado no TP 1.

Observação: é necessário que o programa **flex** esteja instalado na sua máquina. Foi pedido ao CRC que instalassem **flex** nas máquinas da graduação. Caso, por alguma razão, não estiver instalado, realizem SSH para *pomba.grad* ou *velhas.grad*, pois o **flex** já está instalado nelas.

4 Submissão

Vocês precisam enviar apenas o arquivo *cool.flex* contendo sua implementação do analisador léxico.