Compiladores I

Nome: Ivanir Paulo Cardoso Ignacchitti 2017093437
Nome: Gustavo Ribeiro Alves Rodrigues 2018130972

Introdução

Nesse trabalho prático foi feita a implementação de um analisador léxico da linguagem COOL utilizando FLEX.

Um analisador léxico é um programa que converte uma cadeia de caracteres em uma cadeia de tokens para que possa ser usado pelo analisador sintático de uma linguagem.

O Problema

Neste trabalho estamos trabalhando com um analisador léxico para um compilador da linguagem cool. Para fazer essa implementação foi necessário estudar a linguagem, sua sintaxe e seus tokens.

Além disso, o analisador léxico foi exaustivamente testado para que os erros fossem encontrados tornando o programa mais robusto.

Implementação

A primeira tarefa feita, foi definir os tokens de entrada. Para isso, o manual foi lido para que fosse implementado as palavras chaves da linguagem.

COOL possui as palavras chaves:

class, else, false, fi, if, in, inherits, isvoid, let, loop, pool, then, while, case, esac, new, of, not, true.

Além de definir os tokens, outra tarefa foi definir as expressões regulares que definiriam características da linguagem. O lex é um software que facilita muito na escrita de expressões regulares, então bastou-se estudar a sintaxe da linguagem do lex.

Outra tarefa implementada foi definir algumas características da linguagem, como por exemplo, quais são os valores possíveis para uma string e como tratar comentários e erros no código fonte. Para isso uma grande parcela de tempo foi passado estudando sintaxe do lex, sintaxe do COOL e implementações de outras linguagens.

A última tarefa foi pensar casos de teste para que o analisador léxico seja exaustivamente testado. Com a sintaxe da linguagem em mãos, foi muito mais fácil implementá-los. Na implementação dos testes, vários erros foram encontrados e graças a isso, alguns desses erros foram resolvidos, contudo, um erro específico se manteve e será discutido no próximo tópico.

● Decisões de Implementação

Na implementação do analisador léxico, algumas decisões tiveram que ser tomadas. Desenvolver um programa para essa tarefa pode parecer fácil com a ajuda de ferramentas certas, contudo, diversas decisões críticas precisaram ser tomadas para alcançar um resultado satisfatório.

- **Tratamento de Erros:** Começando pelo tratamento de erros, foi implementado funções na parte de código de usuário para que pudessem ser chamadas nas situações necessárias. As mensagens de erro foram definidas baseadas em erros da linguagem C, porém, algumas mensagens de erro são customizadas, pensando unicamente no nosso compilador.
- **Definições:** Na área de definições, foram setadas os cabeçalhos das funções usadas, além disso, uma flag chamada **commflag**, foi definida. Essa flag é utilizada na parte de comentários, ela avalia se o caractere definido para início de comentário multilinhas foi chamado. A **commflag** ajuda a avaliar se há algum léxico quando os caracteres de finalizar comentário aparecem no código fonte.
  Além disso, foram implementadas as expressões regulares que definem características da linguagem. Como a linguagem COOL é case-insensitive para a maioria das palavras chaves (exceto true e false), foram implementadas expressões regulares que permitem a permutação de maiúsculas e minúsculas nas entradas para a setagem dos tokens. Essa decisão foi tomada como segunda opção. A primeira tentativa foi implementar uma função que converte todas as entradas do yytex para maiúsculas e depois compara com ifs para a geração de tokens. Essa decisão não foi mantida pois é bem mais elegante usar as características sintáticas do flex. Além disso, passar por diversas condicionais depois de um autômato determinístico não é uma decisão melhor do que só passar pelo autômato.
  Outra decisão tomada foi na decisão de quais palavras definiriam um TYPEID ou um OBJECTID. Essa decisão só foi necessária ao perceber que o parser da linguagem (disponibilizado) não possuia o token ID e sim dois tokens, o TYPEID e o OBJECTID. A decisão foi então definir que os identificadores de TIPO deveriam começar com letra maiúscula, como por exemplo: **Int,** enquanto o nome de instâncias de objetos começaram com a letra minúscula.
- **Regras:** Na aba de regras, foram implementadas as regras da tokenização da linguagem. Não tem muito pra se falar aqui exceto de 2 decisões importantes que foram definidas nessa parte. Como lidar com **strings** e **comentários.**

- **Comentários:** Implementar o reconhecimento de comentários de uma linha é muito simples, contudo o mesmo não acontece com múltiplas linhas . Para isso foi criado um “escopo” chamado string. Nesse escopo foi então definido os caracteres que podem ser aceitos em um comentário. Todos os caracteres dentro do escopo dos comentários são ignorados pela linguagem. Outra decisão importante nessa parte foi lidar com erros, ou seja, quando a sintaxe de um comentário não está correta. Isso também poderia ser implementado pelo analisador sintático, contudo, pareceu simples lidar com isso neste momento.
- **Strings:** A implementação de strings é bem parecida com do comentário, exceto que, strings possuem um valor simbólico que será usado no analisador sintático e então não deve ser ignorado. Dessa forma, devido a semelhança com os comentários, foi implementado um escopo específico para as strings, que é iniciado quando as aspas duplas (“) são chamadas. A partir desse escopo, pode-se definir quais caracteres serão aceitos e quais não serão. Uma decisão foi não permitir a quebra de linha (o comando é permitido) porém, quando o analisador encontra uma quebra de linha antes do fechamento das strings, ele automaticamente gera um erro.
  Outra decisão importante foi conferir se o tamanho da string que está em leitura é menor que o tamanho do buffer de string disponível. Essa decisão é importante para evitar falhas de segmentação.

● Considerações Finais:

Uma primeira tentativa de definir os nomes de tokens foi feita, contudo foi percebido durante a implementação que era necessário se adaptar a o parser definido pelo enunciado da tarefa e por isso, a maioria dos TOKENS, como os de caráter especial, mantiveram o “caractere” de entrada. Alguns como o **EQ, DARROW e o ASSIGN,** foram mantidos pois aparentemente, o parser os aceitam como entrada.

- **fontes:** https://theory.stanford.edu/~aiken/software/cool/cool-manual.pdf