Relatório do trabalho 2

Matheus Doretto Compri 7151885
Thaís Emanuele dos Santos 6453087
Vinícius Alvarenga Lovato 7696455

1. Introdução

Neste segundo trabalho prático foi implementado o analisador sintático em linguagem C utilizando a ferramente Yacc. Como método de recuperação de erros foi utilizado, como requerido, o modo pânico. Algumas decisões tiveram que ser feitas em relação a recuperação de erros devido a limitações da ferramenta e da própria gramática do LALG, fornecida pelo professor em sala de aula.

2. Decisões de projeto

Como decisão de projeto, implementamos o modo panico da mesma forma como estudado em sala de aula, desta forma, quando o parser encontra um erro, ele consome os tokens ate que encontre um token pertencente ao conjunto de sincronização da regra no qual o erro foi encontrado. Nesse conjunto, estão contidos os tokens seguidores de cada regra.

Quando o Flex consome os tokens, esses são removidos dos buffers de leitura, impedindo assim que esses sejam lidos novamente pela próxima regra. Assim, a cada vez que encontramos o token de sincronização, devemos incluí-lo novamente no buffer de leitura.

Toda a manipulação com os buffers de leitura é feita utilizando as funções fornecidas pela API do Flex. Os tokens lidos são inseridos em um novo buffer e adicionados na pilha de buffers do flex. Quando um buffer é finalizado, o flex verifica se esse é o ultimo buffer, e caso seja, envia EOF para o parser, o qual finaliza a análise.

Segue o conjunto de sincronização de cada regra do gramática do LALG:

dc_c : {var, begin}
dc_v : {procedure, begin}
tipo_var : {;,)}
variaveis: {:, (}
mais_var:{:,)}
dc_p: {begin}
dc_f: {begin}
parametros: {;}
lista_par: {)}
corpo_p: {procedure, begin}
lista_arg: {;}

argumentos: {)}
mais_ident: {)}
pfalsa: {;}
comandos: {end}
cmd: {;}
condicao: {), then}
relacao: {+, -, id, real, integer,)}
expressao: {;,)}
op_un: {id, real, integer, (}
outros_termos: {:,)}
op_ad: {+, -, id, real, integer}
termo: {;,), +, -, id, real, integer, (}
mais_fratores: {;, (, +, -}
op_mul: {id, real, integer, (}
fator: {*, /, +, id, real, integer, (}
var_numero: {id, real, integer, (}

Devido a maneira no qual a gramática foi construída, a recuperação de erros que acontecem na regra cmd não acontece de forma satisfatória. No Yacc uma vez que um erro é encontrado, após sincronizar com o caracter específicado o processo de parsing retorna a regra no qual foi chamado; dessa forma, quando o usuário não coloca o caracter “;” ao final de um cmd, não é possível sincronizar com o próximo “;”, pois esse caracter esta no meio da regra “comandos”. Ao se recuperar do erro o Yacc, então retorna a regra que chamou “comandos”, sincronizando dessa forma apenas com o caracter “end”. Isto acontece devido a forma no qual a gramática do LALG foi implementada, deixando limitada as opções de recuperação de erro no modo pânico.

Quando o analisador sintático não retornar nenhum erro na linha de comando, significa que a analise foi finalizada sem nenhum erro. Caso erros sejam encontrados a saída indicará a linha com o erro encontrado, a regra que possui o erro e os tokens que foram consumidos pelo modo panico. Exemplo:

Parser: syntax error at line 12, token 'read' not expected
comandos error
** Entering in panic mode **
Token skipped: '(' - code: 278
Token skipped: 'vari' - code: 258
Token skipped: ')' - code: 279
Token skipped: ';' - code: 281
** Exiting panic mode **

No caso acima, o token read encontrado não era esperado; o erro foi encontrado na regra relacionada a comandos, o modo pânico então pula os tokens seguintes “(“,”vari”, “)” e “;” e então sincroniza a análise.

3. Passo a passo de compilação

O trabalho foi desenvolvido no sistema operacional Linux Ubuntu 14.04.
A versão do Flex utilizada foi 2.5.35.
A versão do Bison/Yacc utilizada foi 3.02.

.Para compilar, basta executar o comando

./make.sh.

Par executar basta abrir o arquivo executavel *parser*, e passar como parâmetro o arquivo a ser analisado.

Exemplo:

./parser testeSintatico/t1parser.pas