Enunciado Laboratório 02 de Compiladores.
Entrega 12/04/2022

Você deve:

- **Fazer o trabalho individualmente;**
- **Enviar um arquivo .zip contendo todos os arquivos necessários para rodar o parser em cima da sua gramática;**
- **Gravar e enviar um vídeo de 7min a 15min explicando a sua gramática e mostrando o programa executando com ela.**
- **O vídeo também deve explicar eventuais adaptações do programa que possam ter sido necessárias.**
- **O vídeo também deve explicar eventuais adaptações ao programa da fase 1 que possam ter sido necessárias.**
- **Realizar os envios SOMENTE pelo Classroom, até 12/04/2022.**

Crie uma gramática para a linguagem que foi sorteada para você no lab 1. Esta gramática deve expressar no mínimo as seguintes estruturas da sua linguagem. Outras estruturas além destas podem ser inseridas. Sua gramática deve expressar no mínimo isto, podendo adicionar outros.

- Operadores aritméticos,: +, -, *, /, %, parênteses, etc
- Operadores de comparação: <,<=,>,>=,==,!=, etc.
- Operadores lógicos: ||, &&, !
- Atribuições
- Controles de fluxo: if, if-else, while. (Não são necessário, mas são permitidos: for, do-while, switch, goto, break, continue
- Chamada de funções, pondendo ser parte de expressões
- Declaração de funções (Não é necessário suportar função aninhada ou apontadores de função)
- Retorno de função
- Declaração de tipo de variável
- Declaração de variáveis, variáveis array (vetores)
- Acesso a índice de array/vetor
- Escopo: { }, begin-end, etc.
- Caso sua linguagem necessite de definição de classe para um programa básico, então definição de classe deve fazer parte de sua gramática.

Não são necessárias as seguintes estruturas, apesar de serem permitidas:

- Apontadores: uso e declaração (se sua linguagem incluir)
- Operadores de incremento/decremento: ++, --
- Operadore de shift: >>, <<
- Declaração de struct/registros
- Declaração de classes
- Acesso a campo de struct/classe (se sua linguagem suportar)
- Controles de fluxo extras: for, do-while, switch, goto, break, continue

Você deve criar a sua gramática, em seguida utilizar um dos seguintes sites para gerar um parser LR ou LALR para sua gramática:
http://jsmachines.sourceforge.net/machines/lr1.html
http://jsmachines.sourceforge.net/machines/lalr1.html

Lá uma tabela LR é gerada. Esta tabela pode ser copiada para dentro do arquivo Excel .xlsx dentro do anexo do trabalho. (Obs. Esta cópia deve selecionar as células da tabela a partir da linha marcada com 0 e **evitar a última célula** para funcionar em pelo menos uma versão do Excel (a minha). Em seguida, copie a linha com a legenda de colunas.) Dentro do arquivo excel, há a linearização da tabela, feita na segunda planilha. Combinada com um "| grep [0-9] | sort -n", a linearização gera o arquivo tabela_lr1.conf. Desta forma, é possível converter sua tabela em uma lista de regras de transição do parser LR, que é a configuração do parser. O formato de cada regra do arquivo tabela_lr1.conf é:
estado lookahead ação
onde ação pode ser um dentre sNUM, rNUM, NUM, acc (NUM sendo um número natural) e **sNUM** indicando um shift com consumo da entrada e ida ao estado NUM, **rNUM** indicando uma redução pela regra NUM da gramática e **NUM** indicando um **goto** para o estado NUM, e **acc** indicando aceitação.
Por exemplo,
0 a s2
0 b r5
0 c acc
0 S 4
indica as transições: [estado 0 com lookahead a faz um shift consumido a da entrada e indo para o estado 2, etc.]

Em seguida você também deve transcrever sua gramática para o formato de lista de regras, onde cada regra é uma linha no formato:
LadoEsquerdo -> LadoDireito
, onde os símbolos do lado direito estão separados por espaços, e o símbolo -> tem um espaço antes dele e um após ele. Note que o lado direito pode ser vazio.
P. Ex.
S -> S S

A entrada para o seu programa será a saída da fase, um arquivo que contenha uma linha por token léxico. Cada uma das linhas começa com uma string sem espaços seguida de um espaço, seguida de uma sequência de caracteres terminada com '\n'. A primeira string é a classe de token a última sequência de caracteres descrita é a sequência de caracteres de entrada que correspondem ao token identificado.

Finalmente, você precisa adaptar, caso necessário, o seu parser léxico da fase 1. Esta adaptação será para que cada símbolo da gramática que você gerar tenha um tipo distinto vindo da fase sintática. Por exemplo, se você identificou na fase 1 tanto "+" quanto "*" como operadores, você precisará diferenciá-los em uma modificação da fase 1 para que sua gramática possa diferenciá-los, tal como no exemplo abaixo:
S -> E
E -> E + T
E -> T
T -> T * F
T -> F
F -> a

Neste exemplo, o operador + tem papel distinto na gramática do operador * e portanto eles precisam ser diferenciados na entrada. No entanto, diferentes números, tais como 123 e 3456, podem ser identificados na fase léxica como do mesmo tipo, pois terão o mesmo papel na gramática. Por exemplo, uma entrada como a seguinte é suficiente para o parser sintático:
NUM 123
+ +
NUM 456
* *
NUM 5678

A execução final do seu parser deve respeitar a precedência de operadores definida na sua linguagem. Isto pode ser implementado na definição da gramática, ou em uma adaptação manual da tabela LR gerada a partir da gramática. Você deve escolher uma das duas soluções e explicar isto no vídeo, mostrando como você executou sua escolha.

Em seguida, você deve executar o programa parserLR no arquivo anexo. A compilação do programa pode ser feita com [g++ *.cpp -o ../bin/parserLR] dentro do diretório "src/". Para executar, você pode entrar no diretório "tabela-X" que contém os arquivos gramatica.conf e tabela_lr1.conf e chamar o binário com [../bin/parserLR]. Aí a entrada padrão lida deve ser a saída da fase 1, **com a adição de uma linha marcadora de final que é igual a**
$ $
. A saída de seu programa será o debug que já está presente no código.

Os principais de sua submissão são os seguintes itens:
- arquivo gramatica.conf
- arquivo tabela_lr1.conf
- vídeo explicando sua gramática, como ela implementa a linguagem e exemplos de execução.

Em fases posteriores, o objeto do tipo Arvore será utilizado.