Desenvolvimento de um Compilador

Prof. Charles Ferreira

1 Instruções

- Este trabalho deverá ser desenvolvido em grupos de, **no máximo**, 4 alunos.
- Entregas: Projeto e documentação.
- A **documentação** deverá ser entregue em formato **pdf** e deverá conter:
  - Título do trabalho.
  - Componentes do grupo.
  - As expressões regulares de todos os tokens.
  - A gramática do analisador sintático.
  - Instruções de como executar seu compilador e características da linguagem criada.
  - Exemplos de código na sua linguagem criada e a tradução equivalente.
  - Data de entrega: até **10/11/2024**.
  - **A não entrega da documentação até a data especificada causará redução de 1 ponto na nota final do projeto.**
- O **projeto** deve conter:
  - O código fonte e a aplicação executável (se houver).
  - Data de entrega: **10/11/2024**.
  - **Qualquer atraso na entrega e/ou apresentação resultará em desconto de 2 pontos na nota final do projeto.**
- Durante as aulas dos dias 11/11/2024 e 12/11/2024 (turma do vespertino) e 13/11/2024 e 14/11/2024 (turma do noturno) cada grupo deverá apresentar a Linguagem de Programação implementada. Todos os integrantes do grupo deverão estar presentes. Caso algum integrante do grupo não possa estar presente então esta pessoa deverá apresentar o trabalho em uma outra data.

2 Enunciado

O compilador deve fazer a conversão de um programa desenvolvido na Linguagem definida pelo grupo para a Linguagem Java, C ou Python (fica a critério do professor definir para qual linguagem). **A verificação da corretude do programa** será realizada compilando o arquivo gerado pelo compilador desenvolvido.

Seu compilador deverá receber como entrada um arquivo contendo um programa escrito na Linguagem definida pelo grupo e gerar um arquivo (.java, .py ou .c) com o código equivalente na linguagem destino, que deverá ser compilada, executada e não deverá conter erros.

OBS: a gramática não pode conter recursividade à esquerda. Caso seja necessário, elimine a recursividade e efetue a sua fatoração.

Cada grupo deve definir a sua própria gramática e todos os tokens necessários. Os requisitos mínimos são:

- Deve ter, no mínimo, 3 tipos de variáveis
- Deve ter a estrutura de controle if . . . else;
- Deve possibilitar ifs encadeados
- Deve ter ao menos duas estruturas de repetição (while, do . . . while, for);
- Deve possibilitar laços encadeados
- A parte de expressões envolvendo os operadores matemáticos deve ser realizada de maneira correta, respeitando a precedência.
- As atribuições também devem ser realizadas;
- Os comandos de leitura do teclado (Scanner, scanf, input, etc) e de impressão na tela (print) devem ser disponibilizados.
- O compilador tem que aceitar números decimais.
- O compilador deve ter mensagens de erros.
- A cada utilização de uma variável, é necessário verificar se ela já foi declarada.
- É necessário verificar se é possível realizar as operações, devido aos tipos das variáveis e ao seu escopo.

3 Exemplo de um Compilador

A descrição a seguir ilustra um exemplo de uma gramática[1] e sua utilização por um compilador que faz a conversão de um programa desenvolvido em uma linguagem fictícia para uma forma equivalente na linguagem C.

Os termos em negrito significam palavras reservadas. Preste atenção aos sinais de pontuação.

```
prog → 'programa' declara bloco 'fimprog'.

declara → (inteiro | decimal) Id (, Id)*.

bloco → (cmd)+

cmd → cmdLeitura | cmdEscrita | cmdExpr | cmdSe

cmdLeitura → leia '(' Id ')'.

cmdEscrita → escreva '(' Texto | Id ')'.

cmdSe → se '(' expr op_rel expr ')' '{' cmd+ '}' (senao '{' cmd+ '}')?

cmdExpr → Id ':' '=' expr.

cp_rel → '<' | '>' | '<' '=' | '>' '=' | '!' '=' | '=' '='

expr → expr '+' fator | expr '-' fator | expr '*' fator | expr '/' fator | fator

fator → NUM | ID | '(' expr ')'

texto → ' "' (0-9 | a-z | A-Z | ' ' )+ ' "'

NUM → [0-9]+

ID → [a-z]+
```

OBS: espaços em branco, tabs e enter podem aparecer e devem ser eliminados.

[1] Esta gramática não atende a todos os requisitos do projeto mas pode ser usada como ponto de partida

```
programa

inteiro a, b, c.
decimal d.
escreva("Programa Teste").
escreva("Digite A").
leia(a).
escreva("Digite B").
leia(b).

se(a<b)
{
  c:= a + b.
}senao{
  c:= a - b.
}

escreva("C e igual a ").
escreva(c).

d := c / (a + b).

escreva("D e igual a ").
escreva(d).

fimprog.
```

```
#include <stdio.h>

void main(void)
{
int a,b,c;
double d;
printf("Programa Teste");
printf("Digite A");
scanf("%d",&a);
printf("Digite B");
scanf("%d",&b);

if(a<b)
{
  c=a+b;
}else
{
  c = a-b;
}

printf("C e igual a ");
printf("%d",c);

d = c / (a + b);

printf("D e igual a ");
printf("%lf", d):
}
```

4 Critério de Avaliação

O projeto será avaliado mediante os seguintes critérios:

- Analisador Léxico: 3 pontos.
  - fazer todos os tokens mínimos exigidos: 1 ponto.
  - ter a possibilidade de imprimir a lista de tokens: 1 ponto.
  - não utilizar bibliotecas de expressões regulares: 1 ponto.
- Analisador Sintático: 4 pontos.
  - fazer a análise sintática corretamente de todos os tokens: 3 pontos.
  - apresentar mensagens de erros corretas: 1 pontos.
- Analisador Semântico: 1 ponto.
- Diferença da linguagem criada em relação a linguagem traduzida: 1 ponto.
- Adicionais: até 2 pontos (dependendo da complexidade do adicional).

A apresentação terá caráter eliminatório, ou seja, se não souberem responder as perguntas do professor o projeto será desconsiderado.

Projetos iguais e/ou plagiados serão zerados e encaminhados para a Comissão Disciplinar.

5 Adicionais

Cada grupo deve implementar funcionalidades adicionais que não foram previstas no escopo do projeto.

Exemplos de funcionalidades:

- Fazer uma mini IDE para codificar sua linguagem.
- Criar novas regras na gramáticas que não foram pedidas (Exemplo: classes, funções, interfaces, herança, entre outros).
- Fazer seu compilador ser capaz de compilar para duas linguagens diferentes. Sendo que todas as funcionalidades da sua linguagem deverão ser atendidas para ambas linguagens. Neste caso, não precisa se limitar a java, python ou C/C++. Contudo, será avaliado a diferença entre a linguagem criada e linguagem traduzida.
- Fazer um language server, ou seja, o compilador rodar como servidor e fica esperando requisições de outros clientes (computadores) com código para ser compilado e devolve o código na nova linguagem ou o erro do programa.
- Outras ideias (conversar com o professor para validar se a ideia possui um mínimo de desafio que faça valer os pontos adicionais.