Universidade Federal de Viçosa – Campus Florestal
Ciência da Computação – Compiladores
Professor: Daniel Mendes Barbosa
Trabalho Prático Final – Datas de entrega: ver PVANet (várias entregas)

Neste trabalho cada grupo (os grupos foram definidos em sala de aula, e podem ser consultados no arquivo **grupos.pdf**) deverá desenvolver um compilador para a linguagem orion. Cada grupo deverá trabalhar de forma **independente**, sem trocas de ideias entre os grupos, de forma a ter resultados diferentes em cada etapa e no resultado final do trabalho. E isso não somente em relação ao que foi feito de fato, mas como foi feito, além de "personalizações" que cada grupo poderá implementar no caso de terem ideias interessantes. A ideia também é que cada grupo trabalhe sem a interferência do professor no desenvolvimento do trabalho, ou seja, faz parte do trabalho o entendimento das documentações, especificações e materiais de apoio, que deverão ser estudados e discutidos pelos membros de cada grupo.

O trabalho deverá ser implementado com o uso conjunto das ferramentas Lex e Yacc, gerando um código na linguagem C para o compilador, e consequentemente um executável. No entanto, serão três entregas, detalhadas a seguir.

Será disponibilizado, entre outros arquivos, um código em C de um exemplo de implementação de tabela de símbolos, que poderá ou não ser utilizado pelo grupo, além de poder ser alterado no caso do grupo decidir utilizá-lo. No relatório do trabalho deverão ser explicitadas estas decisões e suas justificativas.

Será disponibilizado também um arquivo com a documentação de uma máquina abstrata, chamada TAM. Essa máquina deverá ser estudada, para que o compilador possa então gerar o código para ser executado nessa máquina. Serão disponibilizados também arquivos com os bytecodes (java) dessa máquina.

Abaixo são detalhadas as **etapas** que deverão ser feitas pelo grupo e que serão **avaliadas** pelo professor. Cada etapa será uma entrega a ser feita pelo PVANet (verificar datas de entrega no PVANet). Em cada etapa está descrito o que deve ser entregue.

Etapa 1: criar um analisador léxico "stand-alone" para a linguagem orion utilizando-se o lex (flex). Deverá ser entregue:

- o arquivo lex.l
- um relatório em PDF constando a documentação do que foi produzido até aqui, as decisões tomadas, a forma de se gerar o executável do analisador, a forma

de se usar o analisador, o formato de saída para se exibir o fluxo de tokens (seja criativo), e exemplos de arquivos de entrada (utilizar arquivos fornecidos pelo professor mas também arquivos criados pelo grupo, dizendo quais foram criados pelo grupo) com as respectivas saídas do analisador léxico gerado. O analisador deverá ser capaz de identificar erros léxicos, sempre que for possível, e exibir na saída.

Etapa 2: criar um analisador sintático para a linguagem orion, imprimindo-se na saída:

- o programa fonte com as linhas numeradas;
- o conteúdo da tabela de símbolos;
- "Programa sintaticamente correto" ou então algo do tipo: "Erro próximo a linha x" (onde x é a numeração que foi usada na impressão do fonte na tela) seguido de uma descrição do erro e aí encerra-se a compilação prematuramente.

Nesta etapa deverão ser entregues:

- o arquivo lex.l atualizado, para ser usado em conjunto com o arquivo translate.y;
- o arquivo translate.y;
- código-fonte da tabela de símbolos e outros códigos que o grupo tenha usado;
- um relatório em PDF constando a documentação do que foi produzido até aqui, as decisões tomadas, a forma de se gerar o executável do analisador, a forma de se usar o analisador, o formato de saída criado, e exemplos de arquivos de entrada (utilizar arquivos fornecidos pelo professor mas também arquivos criados pelo grupo, dizendo quais foram criados pelo grupo) com as respectivas saídas do analisador sintático gerado. O analisador deverá ser capaz de identificar erros léxicos e sintáticos, sempre que for possível, e exibir na saída. Erros semânticos ainda não serão identificados nesta etapa.

Etapa 3: análise semântica e geração de código para a TAM: quando o programa estiver sintaticamente correto e semanticamente correto (acrescente análise semântica, como por exemplo, verificação de tipos em atribuições e outros comandos), será gerado um arquivo de saída padrão (por exemplo, `arquivo.tam`). Esse arquivo irá conter instruções para a máquina TAM de acordo com a especificação da mesma, e poderá ser executado pelo seu interpretador. Nesta etapa deverão ser entregues:

- o arquivo lex.l atualizado;
- o arquivo translate.y atualizado;
- código-fonte da tabela de símbolos e outros códigos que o grupo tenha usado, todos atualizados;
- um relatório em PDF constando a documentação final do que foi produzido, as decisões tomadas, a forma de se gerar o executável do compilador, a forma de se usar o compilador, o formato de saída criado, e exemplos de arquivos

de entrada (utilizar arquivos fornecidos pelo professor mas também arquivos criados pelo grupo, dizendo quais foram criados pelo grupo) com as respectivas saídas do compilador, bem como os resultados das respectivas execuções de cada exemplo na TAM. O compilador deverá ser capaz de identificar erros léxicos, sintáticos e semânticos, sempre que for possível, e exibir na saída.

Observações sobre todas as etapas:

1) Alterações que simplifiquem o trabalho obviamente não são interessantes. Mas se forem feitas, por qualquer motivo, deverão ser relatadas e justificadas. O mesmo vale para qualquer outro tipo de alteração.
2) Seja criativo e comente bem os seus códigos, e faça boas documentações.
3) Como personalização, você pode gerar código em outro formato também, documentando é claro a forma de execução deste código em algum ambiente. Se você optar apenas pela personalização (sem gerar código para a TAM), você ainda assim pode ganhar todos os pontos referentes a esta etapa, desde que os resultados sejam bons o suficiente. Fazendo-se os dois, pontos extras poderão ser concedidos.

Dicas importantes:

1) Leia com atenção os arquivos de especificação e discuta com os membros de seu grupo;
2) Teste cada etapa realizada, e aos poucos, a fim de achar possíveis erros com mais facilidade, partindo de exemplos mais simples, e quando funcionarem, avance para exemplos mais complexos;
3) Em cada etapa e principalmente na geração de código, comece por programas com construções mais simples e com poucos "recursos" da linguagem, e vá adicionando novas construções apenas quando estas estiverem funcionando; documente aquilo que funciona e aquilo que não funcione.
4) Para utilizar a TAM (implementada em Java), crie um subdiretório tam com os arquivos `.class` disponibilizados no seu diretório de trabalho (aquele onde estão os arquivos fonte do Lex e Yacc). Para executar um arquivo chamado `arquivo.tam` na máquina abstrata, execute `java tam.Interpreter arquivo.tam` a partir de seu diretório de trabalho.
5) Para simplificar o código de seu compilador, use constantes e funções auxiliares sempre que possível, e bem documentadas.

O arquivo `tpfinal-compiladores.zip` (também disponível no PVANet) tem o seguinte conteúdo:

`grupos.pdf` – arquivo contendo os nomes dos alunos que fazem parte de cada grupo.

`tp final - linguagem orion.pdf`: descrição da linguagem orion (desconsidere descrições não diretamente ligadas à linguagem orion).
`TAM.pdf` – descrição da máquina abstrata tam.
`LEXYACC.PDF` – manual de lex e yacc.
`testes-comp.pdf` – alguns códigos-fonte na linguagem orion para testes.
`tabela.c e tabela.h` - código-fonte exemplo de implementação de tabela de símbolos.
Pasta `tam`: bytecodes da máquina abstrata tam.

Apresentação:

Na última aula do semestre, cada grupo irá fazer uma breve apresentação de seu compilador.

Bom trabalho!