Universidade Federal de Viçosa – Campus UFV-Florestal
Ciência da Computação – Compiladores
Professor: Daniel Mendes Barbosa
Trabalhos Práticos 1, 2 e 3 – Datas de entrega: ver PVANet Moodle

Estes trabalhos práticos deverão ser feitos em grupos de 4 ou 5 alunos, conforme definido com o professor nesta planilha.

Em cada um destes trabalhos práticos, deverá ser entregue um **relatório** através do PVANet Moodle, contendo uma **breve documentação e decisões importantes do projeto (um PDF, como sempre, não zipado)**, além de testes de execução com *screenshots*, de acordo com o que foi pedido em cada trabalho específico. Deverá ser entregue também todo o código do projeto (em um único arquivo .zip). Portanto, serão **3 datas de entrega**, e em cada uma serão entregues dois arquivos: um PDF e um ZIP. Basta que apenas um dos alunos de cada grupo faça as entregas, mas não há problema se mais de um aluno do grupo fizer a entrega. Mas neste caso, o professor poderá corrigir qualquer uma das entregas realizadas, e claro, como sempre, **desde que entregues dentro do prazo**. No PDF da documentação **devem constar os nomes completos de todos os alunos** do grupo que **efetivamente** participaram do trabalho.

Estes trabalhos também deverão ser **apresentados** em aula, mostrando as principais decisões de projeto e o seu funcionamento com **exemplos significativos, diversos e bem justificados**. De uma vez só, cada grupo irá apresentar tudo o que fez nos trabalhos práticos 1, 2 e 3, no final do período letivo. **Tudo o que o grupo for utilizar para apresentar, deve ser entregue junto com o ZIP do trabalho prático 3, incluindo slides, mas o grupo não é obrigado a utilizar slides na apresentação. O tempo de apresentação por grupo será de 20 a 27 minutos.**

A ideia destes trabalhos é que cada grupo trabalhe de forma independente, **sem a interferência dos outros grupos e do professor**, ou seja, faz parte do trabalho a pesquisa e o entendimento de materiais e ferramentas de apoio. Além disso, como cada grupo deve especificar a sua linguagem, é importante que você não saiba o que os outros grupos estão especificando, para não prejudicar a criatividade e originalidade do material produzido pelo seu grupo. No entanto, o professor poderá, **se considerar necessário**, dar algum feedback **em particular** para cada grupo após cada trabalho, solicitando que alguma mudança seja feita para o próximo trabalho.

A seguir estão as especificações de cada um dos trabalhos e é **recomendável** que se faça uma primeira leitura de todo o documento.

Trabalho Prático 1:

Neste trabalho, cada grupo **deverá especificar sua própria linguagem de programação**. Esta especificação, que deverá constar no **PDF da documentação que deverá ser entregue** neste trabalho, **deverá** apresentar:

- nome da linguagem e origem do nome;
- tipos de dados primitivos (deverá haver verificação estática de tipos, mas só será implementada no trabalho prático 3, na análise semântica);
- comandos disponíveis com seu funcionamento;
- paradigma de programação deverá ser **procedural**;
- palavras-chave e palavras reservadas;
- gramática da linguagem, com suas variáveis, terminais (tokens) e padrões de lexema dos tokens;
- outras informações que o grupo julgar necessárias;

Para construir a gramática, tenha como base gramáticas ou pedaços de gramáticas presentes no livro. Você também pode pesquisar na internet para achar gramáticas de outras linguagens. É interessante você fazer algumas derivações para verificar se as produções da gramática realmente funcionam como você pensou inicialmente. Você pode utilizar o **JFLAP** para auxiliar nesta tarefa, na opção de gramáticas.

Observação importante sobre a especificação da linguagem: seja criativo em sua especificação. Não fique com receio de "prometer muito" e depois "não cumprir". Se nem tudo for implementado depois nas demais fases, o grupo irá apenas justificar os motivos disso. É melhor fazer isso do que especificar quase nada e implementar tudo que especificou por ser pouco e fácil. Resumindo: a situação de especificar mais seria melhor valorizada do que a situação de especificar pouco e implementar o pouco que se especificou. Lembre-se que existem linguagens até como a whitespace, mas não precisa ir tão longe assim também na criatividade. Outro detalhe importante é que a própria gramática poderá sofrer alterações à medida que a implementação avança para os trabalhos seguintes, e isso não é problema nenhum. Pelo contrário: representaria o ganho de conhecimento do grupo que teve que fazer as alterações (com justificativas nas documentações) porque aprenderam depois que as alterações eram necessárias. Por fim, é importante esclarecer que você deve ser criativo na sintaxe de sua linguagem, ou seja, para comandos condicionais, para comandos de repetição, arranjos, funções, registros, etc. Não somente para os nomes dos comandos, mas para a sua sintaxe como um todo dentro do programa.

Após esta especificação da linguagem, ainda neste trabalho prático 1, o grupo já deverá implementar o analisador léxico para a sua linguagem, utilizando o gerador de analisador LEX juntamente com a linguagem de programação C, **entregando o código produzido (por exemplo, um arquivo lex.l) no arquivo ZIP e testes e retornos do analisador LEX para arquivos fonte da linguagem no PDF da documentação referenciado anteriormente**. Observe que para este trabalho prático 1, o analisador léxico gerado

deverá fazer impressões na tela parecidas com aquelas do trabalho prático 0 (um analisador léxico *stand-alone*), demonstrando que os tokens foram corretamente identificados. O PDF da documentação deverá conter ainda como compilar para se obter o analisador léxico de sua linguagem e como utilizá-lo.

É importante esclarecer ainda que nos trabalhos 2 e 3 o analisador léxico deverá sofrer alterações, não mais fazendo estas impressões, mas apenas retornando os tokens, ou seja, deixando de ser *stand-alone* e funcionando de forma integrada ao restante. Além disso, vocês poderão inclusive fazer alterações na própria especificação da linguagem, na gramática da mesma (que talvez tenha que sofrer alterações para a implementação), etc. Como já foi dito, isso não é problema nenhum, desde que seja documentado.

Trabalho Prático 2:

Neste trabalho vocês deverão utilizar o gerador de analisador sintático Yacc, juntamente com o LEX e com a linguagem de programação C, para então fazerem a análise sintática de códigos de entrada de sua linguagem de programação. Ao se usar o executável de seu compilador para a sua linguagem, deverá ser impresso na saída:

- o programa fonte com as linhas numeradas;
- o conteúdo da tabela de símbolos;
- "Programa sintaticamente correto" ou "Programa sintaticamente incorreto", dependendo da entrada que for passada. No caso do programa incorreto, pode-se também imprimir 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.

Neste trabalho **deverão ser entregues** no arquivo ZIP:

- o arquivo "lex.l" atualizado, para ser usado em conjunto com o arquivo "translate.y";
- o arquivo "translate.y";
- código-fonte da implementação da tabela de símbolos e outros códigos que o grupo tenha usado;
- arquivos de teste para serem utilizados com o seu compilador;

Uma nova versão do PDF da documentação deverá ser entregue, mantendo-se a especificação da linguagem, mas com uma nova seção documentando-se possíveis alterações em relação à documentação anterior. No caso da gramática ter sido alterada, crie uma nova seção, colocando a nova gramática inteira e depois justificando as mudanças.

Este arquivo PDF deverá trazer também a documentação da análise sintática, como foi realizada, etc, incluindo-se como compilar o seu projeto, isto é, como obter o executável de seu compilador e como utilizá-lo. Além disso, é fundamental que a documentação traga exemplos de execuções, evidenciando exemplos relevantes de códigos

sintaticamente corretos e de códigos sintaticamente incorretos, mostrando-se as saídas de seu compilador para ambos os casos.

Seu compilador 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.

Trabalho Prático 3:

Nesta etapa vocês deverão **primeiramente** atualizar seu arquivo "translate.y" para fazer também a análise semântica. Exemplo: verificação de tipos, que não era feita na versão do trabalho prático 2.
Deve constar na documentação como esta análise semântica foi feita, como a tabela de símbolos foi utilizada, etc.

Em seguida, vocês deverão ainda fazer a geração de código. Possibilidades, todas utilizando linguagem de programação C com exceção da LLVM, onde C++ será permitido se bem documentado (**escolha pelo menos uma**):

- definição de um código intermediário com operações de três endereços (podendo ser parecido com os exemplos que vimos na disciplina), e geração destas instruções de código intermediário; neste caso apenas seria gerado este "texto" com as instruções de três endereços, ou seja, uma representação intermediária linear do código, também podendo-se usar triplas ou quádruplas;
- geração de uma árvore sintática abstrata como representação intermediária;
- geração da IR (representação intermediária) da LLVM a partir da representação intermediária do grupo ou diretamente; em ambos os casos seria possível depois, com a LLVM, gerar um código objeto a partir da IR da LLVM e, portanto, executá-lo na máquina real;
- geração de código para algum simulador ou alguma outra plataforma encontrada, escolhida e documentada pelo grupo, sendo possível ou não a execução, o que também deve ser documentado.

Importante: Observem que independente da possibilidade escolhida pelo grupo, pode ser que, devido a limitações de tempo, o grupo faça uma **implementação parcial** disso, ou seja, seria gerado um código mas não para todas as construções possíveis da linguagem. Neste caso, deve fazer parte da documentação para quais construções é gerado algum código e quais as limitações. Na documentação também devem constar exemplos que se enquadram nestas limitações, isto é, exibindo o código na linguagem fonte, e o código gerado.

Como sempre vocês **deverão entregar** um arquivo ZIP com tudo o que for produzido e um arquivo PDF com a documentação atualizada, trazendo possíveis alterações em relação aos documentos dos trabalhos anteriores e também as explicações do que foi desenvolvido e exemplos de entrada e saída para o seu compilador, com as devidas explicações.

Materiais complementares, disponíveis na internet e que talvez sejam úteis:

JFLAP

Guia Compacto de Lex e Yacc

Compiler Construction using Flex and Bison

The LLVM Compiler Infrastructure

Tutorial usando Lex, Yacc e LLVM

Obs.: se você utilizar quaisquer destas ou outras referências para fazer os trabalhos, as mesmas devem aparecer na seção de referências da documentação de cada trabalho.

Critérios de avaliação para estes trabalhos:

As notas de cada um destes trabalhos serão atribuídas considerando-se o que o grupo entregou de fato dentro do que foi pedido na respectiva especificação de cada trabalho presente neste documento, mas também considerando o quão bem cada tarefa foi planejada e executada pelo grupo. A criatividade e vontade de fazer algo diferente e melhor, o que pode ser facilmente identificado na documentação apresentada e nos códigos apresentados, também podem ser consideradas. Também é um critério de correção a qualidade da documentação, no que diz respeito às especificações construídas, as justificativas, decisões de projeto e exemplos de teste com entrada, saída e explicações. Parte da nota dos trabalhos 2 e 3 especificamente será atribuída considerando-se a apresentação destes trabalhos.
Vale ressaltar ainda que se um grupo não entregar um dos trabalhos, por qualquer motivo que seja, ou entregar pouco em um deles, isso não impede que o grupo recupere o tempo perdido depois e faça a entrega dos trabalhos seguintes.

Bons trabalhos!