Construção de Compiladores
Daniel Lucrédio
Especificação e Critérios de notas do Trabalho 4
(Última revisão: ago/2020)

O trabalho 4 (T4) da disciplina consiste em implementar um compilador completo para uma linguagem de sua escolha. Não é necessário implementar uma linguagem de programação completa (como LA, ou Alguma). Na verdade, isso não é recomendável, dado que irá tomar muito tempo. A ideia é criar uma linguagem pequena, de preferência declarativa, para alguma aplicação específica de interesse dos estudantes. O único requisito é que o compilador tenha:

- Análise léxica + sintática: em outras palavras, uma gramática;
- Análise semântica: três ou quatro (ou mais) verificações de conformidade que não são feitas pela gramática;
- Geração de código ou interpretação: algo de útil ou interessante feito com a linguagem.

Por exemplo, uma linguagem para escrita de melodias teria:

- Análise léxica + sintática: uma gramática para escrita de notas, acidentes, frases, compassos musicais, refrões, trilhas de instrumentos, etc;
- Análise semântica: verificação de número máximo de notas por compasso, verificação de adequação das notas com o tom da melodia, verificação da duração das notas em relação a uma frase, reutilização de refrões por meio de nomes, etc;
- Geração de código: geração de código MIDI para ser reproduzido em um player; ou
- Interpretação: execução da música usando uma biblioteca de sons.

Recomenda-se que os estudantes já comecem a pensar na linguagem com antecedência, de preferência desde o início da disciplina, para poder discutir com o professor sua viabilidade, complexidade, tamanho, etc.

Algumas ideias já executadas por alunos em outras ofertas da disciplina:

1. Linguagem para criação de programas Arduino. A entrada são especificações de comandos, em alto nível, e o resultado é código C que roda na plataforma.
2. Linguagem para especificação de questionários, que gera um HTML com as questões do tipo múltipla escolha.
3. Uma linguagem para descrição de circuitos lógicos, com portas lógicas e conexões. O resultado é uma imagem com o circuito.
4. Compilador de jogos descritivos para text adventures. A entrada seria uma sequência de cenas, salas e ações, e o resultado é um jogo que roda conforme as cenas e ações previstas.
5. Linguagem para geração de DAO (código para Data Access Objects). A especificação seria uma classe estilo Java, e iria gerar código em cima do próprio código-fonte que implementa acesso a dados.
6. Linguagem geradora de composições para o jogo League of Legends. Essa linguagem gera um script em python que, se executado, gera imagens com as características da composição, os campeões organizados por posição e, também, gera automaticamente sugestões de bans, baseados nas características definidas pelo usuário.
7. Linguagem para edição de vídeos. A entrada é uma especificação de comandos em alto nível de edição de vídeos, e a saída é uma linha de comando executável que efetivamente realiza as operações solicitadas.
8. Linguagem para escrita de monografias e artigos, baseada em Markdown, convertendo para Latex. O resultado é um PDF do documento.

9. Linguagem para matrizes curriculares, para especificação de disciplinas por semestre, pré-requisitos, conteúdo, ementa. O resultado é um HTML com a grade curricular e todas as informações.
10. Linguagem para descrever contatos (nome, email, número, endereço, ...) com atividades e tarefas. O resultado é um HTML ou PDF com a descrição dos contatos/tarefas.
11. Linguagem para especificação de receitas culinárias (ingredientes, processo, atividades, …). O resultado é um HTML ou PDF com a receita formatada.
12. Uma linguagem para especificação de conteúdo web que gera para diferentes frameworks CSS. Possibilita criar conteúdos diversos, inclusive alguns de mais alto nível. O resultado são HTMLs diferentes, em frameworks diferentes, para um mesmo conteúdo.
13. Uma linguagem para listas de casamento, contendo vários itens de uma festa, desde convidados, presentes, serviços, fotógrafo, etc. O resultado é um HTML ou PDF contendo os detalhes da festa.
14. Uma linguagem para criação de fluxograma, com atividades, lógica e fluxos de dados. O resultado é uma imagem com o diagrama
15. Uma linguagem para geração de páginas acadêmicas, com dados de professores e sua pesquisa, alunos e disciplinas. O resultado é um HTML com a página dos professores.
16. Uma linguagem para conteúdo jornalístico (notícias). O resultado é uma página de jornal formatada com as notícias organizadas de acordo com seus metadados.
17. Linguagem para gerar mapas de jogos, com base em diferentes algoritmos de geração de conteúdo procedural. O resultado é um código que implementa o mapa gerado.
18. Linguagem para acordes musicais. Duas utilizações: dado um conjunto de notas, encontrar o acorde; dado um acorde, encontrar as notas.
19. Linguagem para geração de plantas baixas de um tipo de imóvel. Possibilita a especificação de itens da planta (cômodos, quartos, paredes) e o resultado é um desenho da planta.
20. Linguagem para simulação de horta. Seria tipo um jogo onde os comandos da linguagem são ações a serem executadas para conseguir crescer a horta. O objetivo é conseguir compilar corretamente o programa.
21. Linguagem para geração de cronogramas. O objetivo é especificar tarefas e prazos, e o resultado é um HTML ou PDF com as tarefas formatadas.
22. Linguagem para visualização de gráficos. O arquivo de entrada terá especificação dos dados, e será gerado código JavaScript que renderiza os gráficos.
23. Linguagem para especificação de esquemas táticos de futebol. A entrada consiste nos detalhes do esquema e a saída é uma imagem com o esquema.
24. Linguagem para criação de animações. A entrada é uma descrição de imagens e animações e o resultado é um programa JavaScript que roda as animações.
25. Linguagem de especificação para autômatos finitos determinísticos. A entrada é a especificação do autômato e a saída é código em C++ que implementa o autômato E uma representação gráfica (para visualizar com o Graphviz)
26. Linguagem de especificação de consultas em bancos de dados. A saída é código SQL das consultas.
27. Linguagem de manipulação de grafos, que gera imagens com representações dos grafos.
28. Linguagem para especificação de polinômios. O resultado é a determinação de valores do polinômio.
29. Linguagem para representação de consultas SQL em HTML. A entrada é uma consulta SQL, e a saída é um HTML que descreve a consulta em mais alto nível.
30. Linguagem para orientação a objetos em português. A entrada é uma definição de classes em português e a saída são classes em uma linguagem de programação OO.
31. Linguagem para descrição de famílias. A entrada são os membros de uma família e a saída é um HTML/PDF com a descrição e uma árvore genealógica.
32. Linguagem para notas fiscais. A entrada é uma descrição dos dados da nota e a saída é um

JSON no formato correto para processamento da nota.

A linguagem pode também ser uma adaptação de uma linguagem existente, desde que:

- A fonte (documentação original) seja citada; e
- As diferenças entre a proposta dos estudantes e a linguagem original estejam claras.

Exemplos: `https://github.com/antlr/grammars-v4`

Os estudantes são livres para escolher a tecnologia que quiserem para implementar o compilador, mas recomenda-se fortemente utilizar um gerador de parser.

Critérios de avaliação do Trabalho 4 e descontos nas notas

O trabalho 4 deve ser desenvolvido em **grupos de até 3 estudantes** (máximo).

A nota do trabalho 4 será composta de 4 parcelas, cada uma valendo de 0 a 10, conforme o conteúdo exigido para o trabalho, e com os pesos assim distribuídos:

ALS - Análise léxica/sintática: 25%
AS - Análise semântica: 25%
GCI - Geração de código ou interpretação: 25%
E - Entrega: 25%

ALS - Análise léxica/sintática: seu compilador deve fazer a análise léxica e sintática da linguagem projetada. Normalmente, basta projetar a gramática corretamente, e o gerador de parser faz o resto. IMPORTANTE: forneça exemplos e casos de teste para demonstrar que a análise léxica e sintática está funcionando!

AS - Análise semântica: seu compilador deve realizar algum tipo de análise semântica (que vai além da análise léxica/sintática), como checagem de consistência, tipos, qualquer coisa vale. Normalmente, basta enriquecer a gramática com alguma verificação adicional. Implemente um mínimo de 3 ou 4 verificações (dependendo da complexidade pode ser mais do que isso, o professor irá acompanhar os grupos nesse sentido). IMPORTANTE: forneça exemplos e casos de teste para demonstrar que a análise semântica está funcionando!

GCI - Geração de código ou interpretação: seu compilador deve fazer algo de "útil" após a análise, seja gerando código ou interpretando os programas. Siga a ideia apresentada no trabalho 2. IMPORTANTE: forneça exemplos e casos de teste para demonstrar que a geração de código ou interpretação está funcionando!

Obs: em todos os itens anteriores, os exemplos e casos de teste não serão os únicos critérios para avaliação. O professor irá modificar alguns e criar outros para realmente testar seu compilador.

E - Entrega: os estudantes deverão criar um repositório em um sistema de controle de versões (ex: GitHub, GitLab, etc) para disponibilizar seu trabalho, de preferência em formato de código aberto. Deve ser um repositório auto-descritivo, contendo:

- Um vídeo curto demonstrativo para "vender" a sua linguagem. Explicando para que ela serve, e como utilizá-la
- Descrição detalhada, possibilitando qualquer um compreender do que se trata a linguagem/compilador
- Descrição de como utilizar a linguagem/compilador
- Devem ser fornecidos exemplos de uso (casos de teste) para a linguagem/compilador
- Descrição de como compilar o compilador, para possibilitar que outras pessoas possam modificar seu código
- Todo o código-fonte, incluindo a gramática, deve ser documentado

Caso o grupo não queira abrir seu código-fonte, deve criar um repositório privado compartilhado com o professor ou um arquivo .zip, enviado por e-mail, contendo as mesmas informações acima.