Compiladores
Trabalho Prático – Projeto de um Compilador

1 Introdução

A melhor forma de aprender sobre compiladores é construindo um! Esse é o objetivo do trabalho prático da disciplina: um projeto de desenvolvimento de um compilador, realizado ao longo de todo o curso, permitindo o estudo do conteúdo de forma prática.

2 Avaliação

Dado que não há provas parciais na disciplina, a avaliação do projeto do compilador determina de forma completa a sua média parcial no curso. A avaliação tem dois objetivos fundamentais:

1. Determinar o aprendizado do conteúdo por meio do código implementado.
2. Medir o nível de participação dos integrantes do grupo no desenvolvimento do projeto.

Para o item 1 acima, o professor vai analisar a versão final do código do compilador entregue no término do curso. A avaliação/correção da implementação do trabalho leva em conta os seguintes aspectos:

- **Corretude**: um compilador que gera código errado obviamente é inútil. Assim, é de se esperar que o ponto fundamental na avaliação do seu compilador seja o seu correto funcionamento.
- **Cobertura dos testes**: esse ponto está diretamente ligado ao anterior porque um bom conjunto de testes tem uma relação direta com o funcionamento adequado de um programa. Certifique-se de construir uma quantidade considerável de casos de testes para o seu projeto. Utilize os casos de teste dos laboratórios como base.
- **Qualidade/organização do código**: assumindo um compilador funcional, o ponto seguinte na avaliação leva em consideração a qualidade do código desenvolvido no projeto. Existem inúmeras recomendações sobre o que distingue um código bom de um ruim (por exemplo, Clean Code); escolha uma que se adeque ao seu estilo de programação e use-a de forma consistente.
- **Documentação**: o código do seu compilador deve estar bem comentado. Note que **bem** não é sinônimo de **muito**, pelo contrário! Citando um dos princípios de Clean Code: *"Clear and expressive code with few comments is far superior to cluttered and complex code with lots of comments."* Via de regra, a sua intenção como programador deve estar diretamente expressa no código, deixando os comentários para esclarecimento dos pontos mais complexos da implementação.

Já para o item 2 da avaliação, serão realizados *checkpoints* (CPs – pontos de controle), onde serão determinados marcos específicos que o andamento do projeto deve atender. A quantidade, formato e pontuação dos CPs, bem como as informações sobre cálculo da média estão disponíveis no programa da disciplina. Fique particularmente atento às datas dos CPs, informadas no calendário da disciplina. A avaliação dos CPs vai se basear nos seguintes critérios:

- **Domínio do conteúdo**: todos os integrantes do grupo devem ter um completo entendimento de todas as partes do código do compilador, *mesmo aquelas que não tenham sido diretamente escritas pela pessoa.*
- **Participação no trabalho do grupo**: todos os integrantes do grupo devem ter uma participação razoavelmente igualitária no desenvolvimento do projeto. Alunos que não seguirem esse princípio vão prejudicar todo o grupo, porque no dia da entrega de um

CP serão feitas perguntas individuais para cada integrante, com as respostas afetando diretamente a nota **coletiva** do CP.

3 Requisitos mínimos do projeto e simplificações

Dado que os projetos envolvem criar compiladores para linguagens de programação (LPs) reais, é certo que será necessário realizar várias simplificações dos aspectos da linguagem fonte, visto que praticamente todas as LPs possuem uma grande quantidade de funcionalidades.

Segue abaixo uma lista mínima de elementos que o seu compilador deve tratar corretamente:

- Operações aritméticas e de comparação básicas (+, $*$, $<$, $==$, etc).
- Comandos de atribuição.
- Execução de blocos sequenciais de código.
- Pelo menos uma estrutura de escolha (`if-then-else`) e uma de repetição (`while`, `for`, etc).
- Declaração e manipulação de tipos básicos como `int`, `real`, `string` e `bool` (quando aplicável à LP).
- Declaração e manipulação de pelo menos um tipo composto (vetores, listas em Python, etc).
- Declaração e execução correta de chamadas de função com número de parâmetros fixos (não precisa ser `varargs`).
- Sistema de tipos que trata adequadamente todos os tipos permitidos.
- Operações de IO básicas sobre `stdin` e `stdout` para permitir testes.

É claro que funcionalidades adicionais são bem-vindas mas certifique-se primeiro que o seu compilador atende os requisitos mínimos acima antes de implementar mais coisas. Se houver qualquer dúvida sobre esse ponto, o grupo deve conversar com o professor para definir adequadamente o que deve ser feito.

Eis uma lista de elementos das LPs de entrada que **não** precisam ser considerados no projeto:

- Qualquer parte de compilação separada, tais como `imports`, etc. Basta que o compilador consiga tratar programas monolíticos em um único arquivo.
- Operações *bitwise* tais como *shifts* para esquerda e direita, operações bit-a-bit, etc. (Operadores lógicos convencionais ainda são necessários.)
- Qualquer parte de chamadas de funções não convencional, tais como funções com `varargs`, valores *default*, chamada com nomes de parâmetros, *packing/unpacking*, etc.
- Qualquer parte relacionada a anotações de programas, como por exemplo os `@` (*decorators*) em Python e Java.
- Comentários com estrutura como por exemplo JavaDoc e afins.
- Tratamento de exceções.
- Asserções.
- Macros e outras formas de pré-processamento como `#define` e `#include` em C, e construções similares de outras linguagens.
- Construções de concorrência ou paralelismo. Por exemplo, `async` e `yield` em Python, e `synchronized` em Java.
- Parte mais complexa de gerência de memória tal como *garbage collection*, etc.