UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Projeto de Implementação de um Compilador para a Linguagem **T++**

Geração de Código (Trabalho – 4ª parte)

Prof. Rogério Aparecido Gonçalves[1]

[1] *Universidade Tecnológica Federal do Paraná (UTFPR)*

Departamento de Computação (DACOM)

rogerioag@utfpr.edu.br

27 de agosto de 2019

Resumo

Este documento apresenta a especificação da 4ª e última parte do trabalho de implementação da disciplina. O objetivo nessa etapa é projetar e implementar a fase de *Geração de Código* do compilador para a linguagem **T++**. O código a ser gerado é o LLVM-IR (código intermediário) utilizando as APIs do projeto LLVM.

Sumário

1 Geração de Código

1.1 Instruções Gerais

1. Faça download do arquivo do modelo de estrutura do trabalho e relatório disponível na página da disciplina no moodle. Descompacte e trabalhe nos arquivos e estrutura fornecida, pois será a mesma estrutura que deverá ser entregue ao final do projeto.

2. Siga a estrutura fornecida para desenvolver o trabalho.

3. O relatório deve ter a descrição do trabalho e dos programas, o código fonte dos programas, uma explicação sobre o funcionamento do programa, o processo de tradução com exemplos de instruções dos três formatos e um exemplo de execução do seu programa reproduzindo a saída gerada.

4. Deverão ser entregues:

a) O código fonte dos programas (C, assembly, binário).
b) Relatório em pdf que pode ser feito no formato do OpenOffice ou no Latex.

8. O projeto deve seguir a estrutura de diretórios e arquivos, disponível no formato. A estrutura do projeto é apresentada na Figura 1.

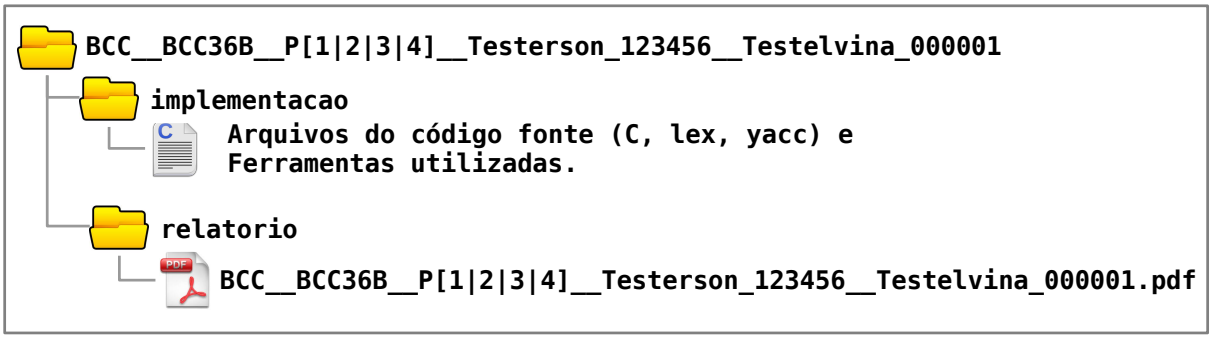

Figura 1: Formato de Entrega

1.2 Implementação

Este é o ununciado da última parte do trabalho, que trata sobre a geração do código intermediário **LLVM**. A análise semântica produz uma árvore sintática abstrata anotada (ASTO). A tabela de símbolos produzida por meio desta árvore anotada e a própria estrutura da árvore deve ser utilizada para a construção do código intermediário

Na fase de geração de código intermediário (RI), a ASTO deverá ser percorrida novamente, classificando assim como a "terceira passada" ou a "segunda passada" do compilador, como foi comentado em sala de aula, para geração da RI. A linguagem alvo irá gerar a RI da LLVM **(LLVM-IR)** (LLVM 2013)(LLVMPY 2014) (LLVM 2017) (Smith 2015) (Bertolacci 2016) e todas as estratégias para a geração do código são apresentadas em sala de aula, além de materiais de apoio postados no moodle.

Ao criar o código para a chamada de função, o tipo dos parâmetros reais deve ser comparado com o tipo dos parâmetros formais. Essa análise é possível pois os parâmetros formais estão declarados na tabela de símbolos. Um erro/aviso deste tipo deverá ser do tipo semântico.

É esperado no fim desse trabalho um código IR gerado em `LLVM` *otimizado*. Para a realização de otimizações, deverá somente aplicar algumas configuração no ambiente de geração de código (API) do `LLVM`. Tais considerações serão apresentadas durante a aula.

1.3 Documentação

Durante toda a disciplina o aluno criará uma documentação formal da implementação do compilador para a linguagem. Sendo os relatório com conteúdo acumulativo, isto é, as fases subsequentes irão complementar o conteúdo existente das fases anteriores. Neste ponto do trabalho o aluno deverá incluir na documentação:

- Explicar se houve alguma estratégia diferenciada para a geração do código IR `LLVM`;
- Explicar a(s) otimizações utilizadas na linguagem de programação; e
- Explicar como o compilador deverá ser executado.

Utilize o formato de artigo da SBC[1] para fazer o relatório.

1.4 Avaliação

Será avaliado se o compilador realiza todas as fases de compilação corretamente para um exemplo simples escrito pelo professor. O essencial da linguagem deve ser mantido (conforme especificado nos trabalhos anteriores) para que o mesmo exemplo funcione para testar todos os trabalhos com o mínimo de alteração.

- Serão avaliados, dentre outros critérios:

  a) Da implementação:
  - O funcionamento do programa.
  - O capricho e a organização na elaboração do projeto.
  - A corretude da implementação em relação ao que foi pedido no trabalho;

[1] [1] Formato para publicação de artigos da SBC: http://www.sbc.org.br/documentos-da-sbc/summary/169-templates-para-artigos-e-capitulos-de-livros

  - A colocação em prática dos conceitos que foram discutidos em sala de aula de forma correta;
  - A qualidade do projeto e da implementação (descrição e elaboração do projeto e o passo a passo da implementação);

b) Do relatório:

  - O conteúdo e a forma que foi apresentado, se o formato é o mesmo solicitado.
  - Organização das ideias e do processo de tradução.
  - O capricho na elaboração e na formatação do texto, bem como o conteúdo do texto.

- Não serão avaliados os trabalhos:

a) Que chegarem fora do prazo;
b) Que não forem feitos nas ferramentas solicitadas;
c) Que não estão no formato especificado;
d) Que não foram compactados em um só arquivo;
e) Que não tiverem identificação (nome e matrícula);
f) Que forem cópias de outros trabalhos ou materiais da internet.
g) Que não seguirem todas estas instruções;

- Não se esqueça que o trabalho contribui com 50% da nota.

1.5 Entrega e apresentação

O trabalho será **individual** e deverá ser entregue até o dia **12/12/2019** no moodle da disciplina em um pacote compactado e apresentado no mesmo dia. Não haverá extensão de tempo, então se organizem para fazer os trabalhos completamente.

A estrutura do projeto com os arquivos do projeto (fonte e relatório) deve ser compactada (zipados) e o arquivo compactado deve ser enviado pelo moodle utilizando a opção de submissão "Trabalho 4a. parte - Geração de Código Intermediário", o nome do arquivo compactado deve seguir o padrão de nomes do formato.

Deverá ser especificado na entrega o mecanismo de execução da varredura para a realização da correção.

Obs.: Favor utilizar ZIP como forma de compactação. O RELATÓRIO DEVE SER TAMBÉM ENTREGUE IMPRESSO, NO HORÁRIO DA AULA, PARA O PROFESSOR.

Referências

AHO, Alfred V., Monica S. LAM, Ravi SETHI, e Jeffrey D. ULLMAN. 2008. *Compiladores: princípios, técnicas e ferramentas*. 2 ed. São Paulo, SP: Pearson Addison-Wesley.

Bertolacci, Ian. 2016. «Writing Fibonacci in LLVM with llvmlite». https://ian-bertolacci.github.io/llvm/llvmlite/python/compilers/programming/2016/03/06/LLVMLite_fibonacci.

html.

JARGAS, Aurélio Marinho. 2012. *Expressões regulares: uma abordagem divertida*. 4 ed. São Paulo, SP: Novatec.

LLVM. 2013. «The LLVM Compiler Infrastructure Site». http://llvm.org.

———. 2017. «Tutorial Kaleidoscope escrito em C». http://llvm.org/docs/tutorial/.

LLVMPY. 2014. «Site da API escrita Python para LLVM». http://www.llvmpy.org/.

LOUDEN, Kenneth C. 2004. *Compiladores: Princípios e Práticas*. 1st ed. São Paulo, SP: Thomson.

Smith, Paul. 2015. «Tutorial não oficial utilizando a API em C para LLVM». https://pauladamsmith.com/blog/2015/01/how-to-get-started-with-llvm-c-api.html.