Geração de Código na implementação de um compilador para a Linguagem T++

Guilherme Vasco da Silva

Departamento de Ciências da Computação
Universidade Tecnológica Federal do Paraná (UTFPR) – Campo Mourão, PR – Brazil

guilhermevasco@alunos.utfpr.edu.br

Resumo. *Este artigo é um trabalho com o intuito de documentar um programa/ferramenta para realização da geração do código intermediário em códigos na linguagem de programação T++.*

1. Linguagem T++

A linguagem a ser lida pelo gerador de código intermediário será a T++, que suporta dados inteiros e flutuantes e necessidade da especificação de tipos das variáveis locais e globais, porém as funções podem ter tipo omitido. A linguagem possui também operações de adição, multiplicação, subtração, divisão e também operadores lógicos E, OU e NÃO.

2. Geração de Código

A quarta e última parte do projeto no desenvolvimento de um compilador para a linguagem T++ trata sobre a geração de código intermediário LLVM (*Low Level Virtual Machine*). Nesta fase, a ASTO gerada na análise sintática deverá ser percorrida novamente para a construção do código intermediário. Nesse processo, será gerado um código em formato *.ll* (texto legível) e outro em formato *.bc* (bitcode), após isso será gerado o arquivo de saída a ser executado.

3. LLVM

LLVM (Low Level Virtual Machine) é uma infraestrutura de compilador que utiliza C++ para otimizar e auxiliar na compilação, provendo camadas intermediárias, lendo a representação intermediária (IR) de um compilador e retornando outra representação otimizada, que pode ser então convertida, com auxílio do clang, e ligada em código de montagem para determinada plataforma.

4. Detalhes do Funcionamento

Para auxiliar na geração do código intermediário é utilizado o pacote LLVMLite do Python, juntamente com os outros pacotes necessários para as outras partes. Também

será necessário o Clang para a geração do *output* a partir do arquivo bitcode. Para instalar esses pacotes necessários utilizam-se, respectivamente, os comandos *pip install llvmlite* e *pip install clang*. Também serão necessários os pacotes complementares Numpy (*pip install numpy*) e LLC (*pip install llc*).

O LLVMLite é utilizado para a criação do arquivo de texto legível (*.ll*) através de funções de IR disponibilizadas nele, nesse momento é utilizado também o arquivo de apoio, escrito em C, "*io.c*". Também é utilizado na passagem do texto legível para o bitcode (*.bc*). O arquivo bitcode, então, será passado ao Clang, que será responsável por compilar o código em um arquivo "*.o*". Por fim, esse arquivo "*.o*" pode ser executado pelo usuário, resultando assim no código completamente compilado.

5. Execução

O código pode ser executado através do arquivo "*tppGenerator.py*" fornecido, junto a ele está uma pasta contendo arquivos para realização de testes (geracao-codigo-testes). A execução é realizada pelo comando "*python tppGenerator.py codigo.tpp*, onde "codigo.tpp" deve ser trocado pelo arquivo a ser executado.

A execução resultará em um arquivo de mesmo nome da origem, mas com extensão ".o", esse arquivo pode ser executado através do comando *./codigo.o*, onde "codigo.o" deve ser trocado pelo nome do arquivo gerado. O resultado do código compilado então será impresso na tela e o sucesso da execução pode ser testado com o comando "*echo $?*", que deve imprimir 0 caso o código tenha rodado sem erros.

6. Testes

Para testar o código podemos pegar qualquer um dos arquivos disponibilizados na pasta geracao-codigo-testes, a exemplo aqui utilizaremos o arquivo *gencode-004.tpp*.

Primeiramente, o arquivo contendo o código deve ser compilado, isso pode ser feito através do comando *python tppGenerator.py geracao-codigo-testes/gencode-004.tpp* e a execução não deve retornar erros. A seguir, se executa o arquivo de saída *.o*, que pode ser feito com o comando *./geracao-codigo-testes/gencode-004.o* , que imprimirá na tela o resultado esperado. Também é possível verificar o sucesso da execução com o *echo $?* e verificando que o resultado é 0.

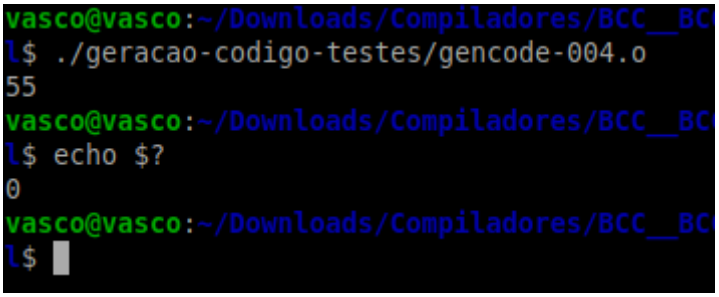

Figura 1. Execução do arquivo compilado *gencode-004.o* e teste de sucesso com *echo $?*.

Nem todos os códigos serão compilados com sucesso, há ainda os que falham em alguma das fases da compilação, seja por erros sintáticos ou semânticos. Para estes, não é possível gerar um arquivo executável de saída, então já no processo de execução é retornado o erro na tela e cancelada a compilação.

A exemplo, pode-se utilizar o arquivo *gencode-014.o*, que possui um erro sintático, então executá-lo com o comando *python tppGenerator.py geracao-codigo-testes/gencode-014.tpp* apresenta na tela o seguinte erro:

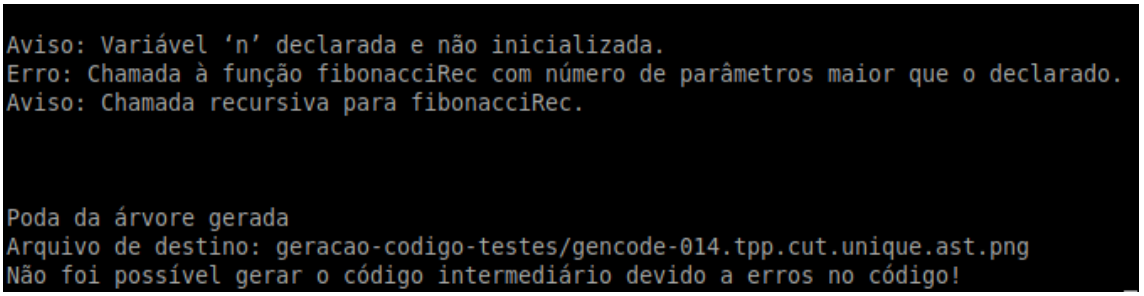

Figura 2. Erro apresentado na tela ao tentar executar o arquivo *gencode-014.tpp*.

Como não é terminada a compilação, não é gerado arquivo de saída que possa ser executado.

7. Conclusões

O módulo de geração de código intermediário está concluído e em funcionamento, o que conclui a última parte das quatro etapas para a compilação de códigos em T++.

O compilador desenvolvido está de acordo com a proposta de compilar códigos na linguagem T++. As quatro etapas ensinadas ao decorrer do período foram aplicadas para funcionar em conjunto em um compilador completo, capaz de gerar executáveis, ou indicar erros que impeçam a compilação de acontecer, tal qual outros compiladores que estamos acostumados a utilizar.

Referências

Gonçalves, R. A. (2017) “Documentação online da Gramática da TPP”, https://docs.google.com/document/d/1oYX-5ipzL_izj_hO8s7axuo2OyA279YEhnAItgXzXAQ, Dezembro.

Ricarte, I. L. M. (2003) “Aula 18 – Geração de Código Intermediário”, https://moodle.utfpr.edu.br/pluginfile.php/1112014/mod_resource/content/4/aula-18-geracao-de-codigo-ir-llvm-ir.md.slides.pdfl, Dezembro.

Gonçalves, R. A. (2021) “Projeto de Implementação de um Compilador para a Linguagem T++”, https://moodle.utfpr.edu.br/pluginfile.php/185440/mod_resource/content/12/trabalho-04.md.notes.pdf, Dezembro.

Wikipedia (2021) “LLVM”, https://pt.wikipedia.org/wiki/LLVM, Dezembro.