ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

RELATÓRIO SISTEMAS DE PROGRAMAÇÃO COMPILADOR C–

Gustavo Trivelatto Gabriel
Brian Andrade Nunes

São Paulo

2021

SUMÁRIO

RESUMO

O projeto aqui desenvolvido utiliza conceitos apresentados pela disciplina PCS3616 - Sistemas de Programação para realizar a implementação de um compilador simples para a linguagem C–, desenvolvida especificamente para utilização neste projeto. De acordo com as especificações apresentadas, foram implementadas as máquinas de estados para o analisador léxico e para o analisador sintático do compilador, assumindo-se também algumas restrições para simplificar a implementação do compilador em questão.

Por conta de restrições de tempo e conhecimento quanto ao projeto, não foi possível implementar o analisador semântico, que seria a última etapa do compilador. O restante do projeto será explicado em mais detalhes no decorrer deste relatório.

1 INTRODUÇÃO

1.1 Objetivos

Os objetivos deste projeto eram implementar um compilador funcional para converter da linguagem C–, uma linguagem C simplificada, para assembly, utilizando as peculiaridades apresentadas em aula sobre as instruções utilizadas, e também se fosse possível, integrar com o projeto de algum grupo que tivesse feito o projeto do ambiente de execução.

1.2 Estrutura do Trabalho

O trabalho será apresentado neste relatório em módulos, primeiro especificaremos os detalhes do analisador léxico, em seguida o sintático, e por fim o semântico, justificando decisões como as restrições definidas e problemas encontrados durante a implementação.

2 DESENVOLVIMENTO

2.1 Restrições

O primeiro ponto que será abordado neste relatório são as restrições assumidas durante o desenvolvimento do projeto, por via de regra, as restrições que serão aqui tratadas foram escolhidas para simplificar a implementação, em troca de um aumento na dificuldade da escrita do código em C–.

- Restrição de caracteres em nomes de variáveis e rótulos
  - Por questão de simplicidade, foram limitados em 2 bytes (2 caracteres)
  - Esta escolha se deve ao limite de 2 bytes por endereço de memória na MVN
  - Com esta restrição, foi possível utilizar apenas um endereço de memória para armazenar cada nome, simplificando verificações no compilador
- Restrição do tipo inteiro por caracteres, e não por tamanho do número de fato
  - As instruções do C– indicavam um limite de 2 bytes para o tipo int
  - Foi assumido pela equipe que poderíamos utilizar 2 bytes como 2 caracteres, em oposição a uma representação de número com tamanho total de 2 bytes
  - Reduzimos assim o limite de inteiros de 0 a 65536 para 0 a 99
  - A simplificação foi assumida pois seria possível implementar funções necessárias mesmo sem ter um grande limite de valores, para questões do projeto aceitamos que seria o suficiente de aprofundamento
- Restrição de uma operação por linha
  - Somente uma operação é aceita por linha de código
  - Esta restrição evitou que a equipe precisasse se preocupar com ordem de operações, ao custo de que cada operação deve ser feita de forma individual, não podendo ser aninhadas em uma só linha
- Restrição de parâmetros para as funções if e print
  - Assim como na restrição anterior, esta foi feita para simplificar a ordem de operações e limitar o número de operações em uma linha
  - Restringimos a função print para aceitar apenas variáveis como parâmetro
  - Quanto ao if, foi restringido para aceitar apenas comparações entre variáveis

  - Nenhuma das duas funções aceita expressões, estas ficam restritas apenas à atribuição nas variáveis

- Restrição de número de variáveis e rótulos
  - Foi restringido em 16 a quantidade máxima de cada tipo de variável e de rótulos
  - Esta restrição foi assumida para poder utilizar uma lista estática para armazenar os valores

2.2 Aspectos gerais

Para realizar a análise completa utilizando os três analisadores, seria necessário ler exatamente três vezes o arquivo fonte em C–, por particulariedades da MVN da disciplina, a instrução de leitura não consegue resetar sua posição, portanto tivemos de desenvolver uma solução para este impasse. A solução encontrada foi instanciar o mesmo arquivo em três dispositivos diferentes no arquivo disp.lst, permitindo assim que ele fosse acessado tantas vezes quanto fosse necessário para a compilação completa.

2.3 Analisador Léxico

O analisador léxico foi pensado para implementar fielmente a máquina de estados apresentada na proposta do compilador. Sua linha de execução basicamente verifica qual é o caractere atual e dependendo do tipo o direciona para a verificação correspondente. Após verificar todo o arquivo do código fonte, a execução é imediatamente direcionada para o nalisador sintático.

Por ser um analisador simples que verifica apenas sequências de caracteres individuais, ele não armazena nada, não escreve nada e não possui nenhuma saída, exceto a mensagem de erro caso o analisador tenha encontrado algum erro no arquivo.

2.4 Analisador Sintático

Podemos dizer que o analisador sintático é o principal ponto de interesse do compilador, pois é ele que permite analisar realmente se os comandos, as funções e as variáveis estão dentro das regras estabelecidas pelo C– e também pelas restrições assumidas no planejamento. Seu funcionamento foi simplificado por conta da restrição de 2 bytes por nome de variável ou rótulo, já que cada verificação poderia ser armazenada dentro de um único endereço auxiliar, além de ocupar apenas um endereço na memória após ser armazenado.

Sua estrutura basica se baseia em testar caracteres em sequência, de forma semelhante a uma máquina de estados. Como um exemplo, temos a verificação para a instanciação de uma variável do tipo *char*, ela inicia identificando um caractere C no inicio da linha, neste caso, ele testará o próximo caractere em busca de um H, caso não encontre um H vai passar para o teste de nome de variável ou de rótulo, caso contrário continua a execução em busca de um A, um R e por fim um espaço. Todas as outras palavras reservadas são analisadas da mesma forma.

Após a identificação do comando, podemos iniciar o teste das regras daquele comando em específico, continuando o exemplo do *char*, ele deve testar se depois do espaço há dois caracteres válidos em sequência, caso os caracteres sejam válidos o compilador segue para uma busca nas variáveis já armazenadas para verificar se há espaço para uma nova na lista, ou então se ela já foi instanciada anteriormente. Por fim, o analisador irá testar a existência de um ponto e vírgula no final da linha, logo após o nome da variável, assim que ele confirmar sua existência, a execução retorna ao início como um looping.

Cada comando segue uma regra em específico, mas em linhas gerais todos são analisados de formas semelhantes. Ao fim da execução do analisador sintático é esperado que tenhamos populado a lista de chars, a lista de ints e a lista de rótulos, de certa forma este é o único retorno do analisador sintático além dos casos de erro, pois assim como o léxico ele não escreve nada no arquivo final.

2.5 Analisador Semântico

O analisador semântico é o último passo para a compilação do programa em C–, é neste passo que será escrito o arquivo em Assembly. Por já ter passado tanto pelo analisador léxico quanto pelo semântico, podemos neste ponto assumir que não haverão erros de escrita e nem de lógica, logo, podemos focar apenas em regras de escrita para o código final. A única exceção a esta regra é o comando *goto*, pois no analisador sintático é possível saber se um rótulo é duplicado, porém, um *goto* pode referenciar um rótulo posterior ao uso do comando, portanto, ao identificar um *goto* nesta etapa da compilação iremos verificar na lista de rótulos gerada pelo analisador sintático se o rótulo em específico existe.

Além desta ressalva, não há muito mistério no desenvolvimento do analisador semântico, ele deve utilizar a mesma lógica de máquina de estados para identificar os comandos e as variáveis, e de acordo com o que for identificado, entrar em uma lógica de escrita específica. Para cada comando iremos utilizar uma instrução específica da MVN, assim como para cada variável iremos utilizar seu endereço na lista correspondente. A única exceção a esta regra é o uso de rótulos, estes poderão ser armazenados de maneira literal, sendo simplesmente re-escritos dentro código final em Assembly.

Ao final da execução, esperamos ter um código Assembly funcional para o código fonte em C–. Como não foi possível terminar o desenvolvimento do compilador, não podemos dizer que as subrotinas e métodos desenvolvidos funcionam da maneira esperada, contudo, mais detalhes sobre os problemas serão apresentados na conclusão do relatório.

2.6 Perguntas

1. Os algoritmos propostos podem ser mais eficientes? Como? Se sim, por que a forma menos eficiente foi escolhida?

    - Poderiam, os algoritmos utilizados fazem testes simples caractere a caractere, com um pouco mais de pensamento empregado no desenvolvimento destes algoritmos seria possível encontrar algum que simplificasse este processo de leitura das informações. Além disso, acreditamos que também seja possível paralelizar a execução dos analisadores do compilador, com esta alteração o código também acabaria se tornando mais eficiente. O algoritmo mais simples foi escolhido justamente por sua simplicidade, ele pode não ser o mais rápido ou o que consome menos processamento, porém, ele foi o mais rápido de se implementar, atingindo o mesmo objetivo.

2. Os códigos escritos podem ser mais eficientes? Como? Se sim, por que a forma menos eficiente foi escolhida?

    - Podem, diversas vezes a dupla acabou identificando no código pontos que poderiam ser simplificados ou até mesmo transformados em uma subrotina, porém, por conta da dificuldade já empregada em desenvolver o compilador, acabamos fazendo de uma forma mais rebuscada, porém mantendo a funcionalidade, o que nos permitiu poupar tempo, em troca de não ter um código simplificado.

3. Qual foi a maior dificuldade em implementar o compilador?

    - A maior dificuldade encontrada ao implementar o compilador foi trabalhar com condições recursivas, tanto que parte das nossas restrições foram escolhidas apenas com o objetivo de mitigar a dificuldade neste aspecto. Variáveis com nomes variáveis foram transformadas em variáveis com 2 bytes obrigatóriamente no nome, expressões foram forçadas a ter apenas um operando por linha e assim por diante, tudo que conseguimos simplificar neste sentido foi feito.

4. O que teria que ser mudado no seu código para poder suportar definição de funções? E para suportar uso de arrays (tanto de int como de char)?

    - Para as funções, seria necessário apenas criar uma nova palavra reservada para definir uma função e criar os tratamentos necessários para este comando, após

ter o tratamento implementado, a função acabaria funcionando como um rótulo, com a diferença que seria necessário passar parâmetros para a função, o que poderia ser feito acessando a própria lista de variáveis.

- Já para arrays, seria necessário gerar uma lista de tamanho variável, porém da mesma forma que a lista já existente de chars e ints, tendo um indexador que armazena o nome da variável, neste caso, o nome do array. A partir daí, o acesso poderia se dar normalmente através dos índices do array, sendo necessário apenas alterar alguns detalhes do tratamento base de variáveis.

3 CONCLUSÕES

Ao fim do desenvolvimento do projeto, pudemos entender mais a fundo o funcionamento de um compilador e perceber quão extenso e complexo ele pode ser. Um compilador funcional envolve diversas verificações e testes para garantir que não houveram erros de escrita ou de lógica no código, além de precisar de um método robusto de escrita para o código alvo, garantindo que seja escrito na sequência correta e de forma oganizada.

Por conta de toda a complexidade envolvida no projeto, não foi possível concluí-lo a tempo para a entrega final. O grupo foi capaz de desenvolver até o início do analisador semântico, porém sem concluí-lo e também sem poder testar os outros dois analisadores. Além disso, também houveram complicações no desenvolvimento da análise do comando *if* e da análise de expressões, de forma que terminamos o projeto apenas com uma espécie de "esqueleto"do compilador final. Para um compilador funcional, seria necessário empregar muito mais tempo e dedicação ao projeto, algo que não foi possível no tempo disponível para o desenvolvimento.