PUC Minas

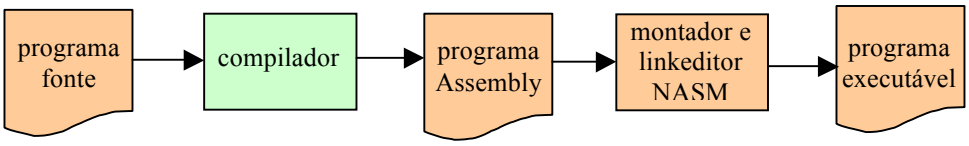
Pontifícia Universidade Católica de Minas Gerais
ICEI - Departamento de Ciência da Computação
Disciplina: Compiladores
Prof. Alexei Machado

Trabalho Prático

A construção de um compilador para uma linguagem imperativa simplificada

Objetivo

O objetivo do trabalho prático é o desenvolvimento de um compilador completo que traduza programas escritos na linguagem fonte "L" para um subconjunto do ASSEMBLY x64. Ambas as linguagens serão descritas durante o semestre. Ao final do trabalho, o compilador deve produzir um arquivo texto que possa ser convertido em linguagem de máquina pelo montador NASM e executado com sucesso em um processador real. No caso do programa conter erros, o compilador deve reportar o primeiro erro e terminar o processo de compilação. **O formato das mensagens de erro será especificado em cada uma das partes do trabalho e deverá ser rigorosamente observado.** O programa deve receber o arquivo-fonte por redirecionamento de modo a poder ser avaliado pelo VERDE. Para a etapa de geração de código, o programa ASSEMBLY gerado deve ser escrito em um arquivo com nome "saida.asm".



Definição da Linguagem-Fonte L

A linguagem "L" é uma linguagem imperativa simplificada que oferece tratamento para 5 tipos básicos: *char*, *string, int, float, boolean*. O tipo *char* é um escalar que varia de 0 a 255, podendo ser escrito em formato alfanumérico ou hexadecimal. Constantes em formato hexadecimal são da forma 0xDD, onde D é um dígito hexadecimal. Constantes em formato alfanumérico são da forma 'c' onde c é um caractere imprimível. Um *string* é um vetor com no máximo 255 caracteres úteis, que quando armazenado em memória, tem seu conteúdo útil terminado pelo byte 0. Constantes que representam strings são delimitadas por aspas e não devem conter aspas ou quebra de linha. O tipo *int* é um escalar que varia de -2^{31} a $2^{31}-1$. O tipo *float* é um número decimal com 6 casas de precisão, sem notação científica, variando de -99999.9 a 99999.9. Por ser ponto flutuante, a precisão é compartilhada entre a parte inteira e a fracionária. Constantes do tipo *float* podem começar ou terminar com o ponto decimal. Constantes do tipo *boolean* são da forma *true* e *false*.

Os caracteres permitidos em um arquivo fonte são as letras, dígitos, espaço, sublinhado, ponto, vírgula, ponto-e-vírgula, dois-pontos, parênteses, colchetes, chaves, mais, menos, aspas, apóstrofo, barra, barra em pé, arroba, e-comercial, porcentagem, exclamação,

interrogação, maior, menor e igual, além da quebra de linha (bytes 0Dh e 0Ah para Windows e 0Ah para Linux e MacOS). Qualquer outro caractere é considerado inválido.

Os identificadores de constantes e variáveis são compostos de letras, dígitos e sublinhado, devem começar com letra ou sublinhado e ter no máximo 32 caracteres. Maiúsculas e minúsculas não são diferenciadas.

As seguintes palavras e suas variações são reservadas:

const	int	char	while	if	float
else	&&	||	!	:=	=
(	)	<	>	!=	>=
<=	,	+	-	*	/
;	{	}	readln	div	string
write	writeln	mod	[	]	true
false	boolean				

Os comandos existentes em "L" permitem atribuição a variáveis através do operador :=, entrada de valores pelo teclado e saída de valores para a tela, estruturas de repetição (repita enquanto), estruturas de teste (se - então - senão), expressões aritméticas com inteiros e reais, expressões lógicas e relacionais, conversão para *int* ou *float*, além de atribuição, leitura/escrita de elementos e comparação de igualdade para strings. A ordem de precedência nas expressões é:

a) Acesso a posições do string, usando [ ];
b) parênteses;
c) conversão de tipos: *int*(expressão) ou *float*(expressão). A expressão tem que ser do tipo inteiro ou real;
d) negação lógica (*!*), aplicado a valores do tipo lógico;
e) multiplicação aritmética (*) e lógica (*&&*), divisão real (/), quociente da divisão de 2 inteiros (div) e resto da divisão (mod) entre inteiros. Uma multiplicação ou divisão real entre inteiro e real resulta em real.
f) subtração (-), adição aritmética (+) e lógica (||). Apenas números inteiros ou reais podem ser somados ou subtraídos. Uma operação entre inteiro e real resulta em real.
g) comparação entre números ou caracteres (=,!=,<,>,<=,>=) e entre strings (=). Números inteiros e reais podem ser comparados entre si. Caracteres só podem ser comparados com outros caracteres.

Comentários são delimitados por /* e */ . Espaços em branco e quebra de linha podem ser usados livremente como delimitadores de lexemas.

A estrutura básica de um programa-fonte é da forma:

(Declarações ∪ Comandos)* fim_arquivo

Declarações e comandos são finalizados por ponto-e-vírgula. Um bloco de comandos pode substituir um comando. Nesse caso, o bloco se inicia por *{* contém uma sequência de 0 ou mais comandos e é terminado por *}*. Dentro de um bloco de comandos não pode haver declarações.

A seguir, é feita a descrição informal da sintaxe das declarações e comandos da linguagem:

1. Declaração de variáveis: é da forma: ***tipo lista-de-ids*** , onde *tipo* pode ser *int, float, string, boolean* ou *char* e *lista-de-ids* é uma série de 1 ou mais identificadores, separados por vírgulas. Cada variáveis pode ser opcionalmente inicializadas na forma: ***id := valor*** , onde *id* é um identificador. Para inteiros e reais, *valor* é uma constante precedida ou não de sinal negativo; Para caractere, uma constante hexadecimal ou alfanumérica; Para *string*, uma constante string; Para *boolean*, true ou false.

2. Declaração de constantes escalares: é da forma: ***const id = valor*** , onde *id* é um identificador e *valor* uma constante numérica, precedida ou não de sinal negativo, string, boolean, hexadecimal ou caractere alfanumérico.

3. Comando de atribuição: é da forma ***id := expressão*** para tipos numéricos, lógico, caractere e string ou ***id[expressão] := expressão*** para elementos de strings (caracteres). Identificadores de variáveis reais podem receber números inteiros. Fora isso, a atribuição só pode ser realizada para tipos idênticos.

4. Comando de repetição pode assumir duas formas:

   while (expressão) comando
   while (expressão) { comandos }

   onde *expressão* é do tipo lógico. A repetição do comando ou bloco de comandos será feita enquanto a expressão for verdadeira.

5. Comando de teste: pode assumir as formas, onde *expressão* é do tipo lógico:

   if (expressão) comando1
   if (expressão) comando1 else comando2

   comando1 e/ou *comando2* são comandos da linguagem que podem ser independentemente substituídos por blocos da forma:

   if (expressão) { lista_comandos1 } else { lista_comandos2 }

   onde as listas são sequências de comandos.

6. Comando nulo: Um ponto-e-vírgula não precedido de comando indica um comando nulo. Nada é executado nesse comando.

7. Comando de leitura: é da forma *readln(id)*, onde *id* é um identificador de variável numérica, caractere alfanumérico ou string. Caracteres e strings são lidos da mesma maneira, sem que o usuário precise coloca-los entre aspas ou apóstrofos. Números podem ter sinal negativo.

8. Comandos de escrita: são da forma *write(lista_expressões)* ou *writeln(lista_expressões)* , onde *lista_expressões* é uma lista de uma ou mais expressões numéricas, caracteres ou strings, separadas por vírgulas. A última forma, quando executada, causa a quebra de linha após a impressão.

Considerações gerais para todas as práticas:

1. O trabalho deverá ser feito de forma colaborativa em um repositório **privado** do Github[1] por grupos de dois ou três alunos, sem qualquer participação de outros grupos e/ou ajuda de terceiros nem utilização de trabalhos feitos em semestres anteriores. Grupos com mais que 3 alunos terão notas proporcionalmente reduzidas. As partes devem ser postadas por apenas um dos componentes, tanto no VERDE quanto no Canvas. Entretanto, cada aluno deve participar ativamente em todas as etapas do trabalho. Os componentes dos grupos devem ser informados em um prazo de 2 semanas, através de e-mail para alexeimcmachado@gmail.com e não poderão ser alterados durante o semestre. O email será respondido com um identificador gerado para o grupo, o qual será usado como prefixo do nome da conta no VERDE. Os alunos que não tiverem feito grupos até essa data serão agrupados pelo professor de maneira arbitrária, em grupos de 2 ou 3 alunos.

2. Os arquivos-fontes submetidos no Verde devem ser alterados para não terem argumentos de entrada e sim um redirecionamento do arquivo-fonte L para a entrada padrão. **Deve ser criada apenas uma conta no VERDE por grupo com o prefixo informado por email**, na linguagem escolhida para desenvolvimento (C++ ou Java).

3. O VERDE é um ambiente experimental sobre o qual o professor não tem controle. Podem ocorrer instabilidades e erros de diversos tipos. É importante testar o ambiente com uma versão inicial do compilador o mais cedo possível para se evitarem problemas de última hora. No caso da impossibilidade de uso do VERDE, o trabalho deve ser postado no Canvas no prazo determinado e apresentado posteriormente de forma síncrona na aula planejada no cronograma para esse fim.

4. Tire suas dúvidas sobre o trabalho até a aula anterior à data de entrega, pois o professor nem sempre estará disponível para dúvidas de última hora.

[1] https://fullcycle.com.br/git-e-github/

5. Os arquivos de código postados devem conter um cabeçalho com a identificação da disciplina e dos componentes do grupo. Nomes não informados no cabeçalho não receberão a nota relativa àquela atividade. O código deve ser extensamente comentado, com a descrição da lógica usada. **Arquivos pouco comentados e difíceis de ler terão sua nota reduzida em 30%.**

6. A codificação do trabalho deve ser feita em linguagem C++ ou Java para avaliação no Ambiente VERDE em um compilador de linha (gcc, javac) com redirecionamento de entrada e saída na tela. O arquivo (único) enviado deve poder ser compilado **sem necessidade de arquivos de projeto específicos de IDEs**. Não poderão ser utilizados bibliotecas gráficas ou qualquer recurso que não esteja instalado oficialmente nos laboratórios do ICEI.

7. O trabalho será avaliado em 3 etapas:
   a) as práticas TP1 e TP2 (10 pontos), em uma única versão final, deverão ser postadas no VERDE até 05/10/2022 em cada um dos testes cadastrados e apresentados posteriormente nas aulas do cronograma destinadas a esse fim.
   b) as práticas TP3 (10 pontos) e TP4 (15 pontos) em uma única versão final com as práticas TP1 e TP2 (compilador completo), deverão ser postada no Canvas até 20/11/2022. A documentação consistindo do Esquema de Tradução e do relatório do Github deverá ser postada no Canvas na mesma data. Os trabalhos finais postados devem estar na forma de um arquivo compactado **em formato zip**, com **tamanho máximo de 5MB**, e seu nome deve ser o identificador do grupo (Ex:G01.zip). **Os arquivos fontes devem estar no diretório raiz** e devem conter o nome de todos os componentes do grupo no início do código. O compilador completo será avaliado de forma presencial conforme cronograma, devendo ser usada a versão do código entregue pelo Canvas.

   Não se admite atraso em qualquer etapa. A apresentação individual é obrigatória para a confirmação da nota. Alunos que não apresentarem o trabalho receberão nota nula em todas as partes.

8. **Trabalhos iguais, na sua totalidade ou em partes, copiados, reaproveitados de semestres anteriores, por todos ou alguns dos componentes, "encomendados" ou outras barbaridades do gênero, serão severamente penalizados. As notas parciais eventualmente lançadas serão reajustadas para 0, com a perda dos 35 pontos correspondentes. É responsabilidade do aluno manter o sigilo sobre seu trabalho, evitando que outros alunos tenham acesso a ele e mantendo o repositório do Github como privado. No caso de cópia, ambos os grupos serão penalizados, independentemente de quem lesou ou foi lesado no processo.**

9. Será pedida ao Colegiado uma advertência formal no caso de cópia por má fé.

10. Durante a apresentação poderão ser feitas perguntas relativas ao trabalho, as quais serão consideradas para fim de avaliação. Todos os componentes devem comparecer e serem

capazes de responder a **quaisquer perguntas e/ou alterar o código de qualquer parte do trabalho**. A avaliação será individual e levará em consideração o relatório de participação de cada aluno no Github.

11. É fundamental que a especificação do trabalho seja **rigorosamente obedecida**, principalmente com relação às mensagens de erro, uma vez que parte da correção será automatizada. O código L a ser compilado deverá ser lido da entrada padrão, linha a linha e a mensagem de sucesso ou erro escrita na saída padrão.

12. A avaliação será baseada nos seguintes critérios:
    - Correção e robustez dos programas
    - Conformidade às especificações
    - Clareza de codificação (comentários, endentação, escolha de nomes para identificadores)
    - Parametrização
    - Apresentação individual
    - Documentação