PUC Minas

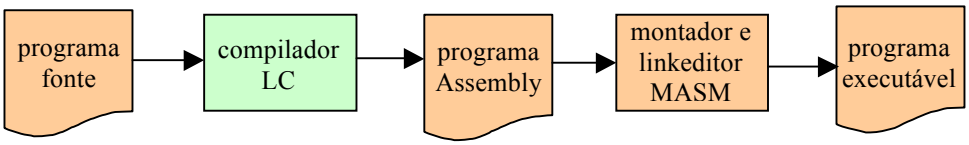
Pontifícia Universidade Católica de Minas Gerais
ICEI - Departamento de Ciência da Computação
Disciplina: Compiladores
Prof. Alexei Machado

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 da família 80x86. 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 MASM 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, caso contrário deverá emitir uma mensagem de sucesso com o número de linhas compiladas. **O formato das mensagens de erro será especificado posteriormente e deverá ser rigorosamente observado.**



Definição da Linguagem-Fonte L

A linguagem "L" é uma linguagem imperativa simplificada, com características do C e Pascal. A linguagem oferece tratamento para 4 tipos básicos: *byte*, *int, boolean* e *string*. O tipo *byte* é um escalar que varia de 0 a 255, podendo ser escrito em formato decimal ou hexadecimal. Constantes em formato hexadecimal são da forma 0xDD, onde DD é um número hexadecimal. O tipo *integer* é um escalar que varia de –32768 a 32767, ocupando 2 bytes. O tipo *string* é um arranjo que pode conter até 255 caracteres úteis e quando armazenado em memória, é finalizado pelo caracter '$'. Variáveis do tipo string ocupam 256 bytes de memória. O tipo boolean pode ter os valores TRUE e FALSE, ocupando um byte de memória (0h para falso e FFh para verdadeiro).

Os caracteres permitidos em um arquivo fonte são as letras, dígitos, espaço, sublinhado, ponto, vírgula, ponto-e-vírgula, e_comercial, dois-pontos, parênteses, colchetes, chaves, mais, menos, aspas, apóstrofo, barra, exclamação, interrogação, maior, menor e igual, além da quebra de linha (bytes 0Dh e 0Ah). Qualquer outro caractere é considerado inválido.

Strings são delimitados, no programa-fonte, por aspas e não podem conter quebra de linha ou aspas.

Os identificadores de constantes e variáveis são compostos de letras, dígitos e o sublinhado, não podem começar com dígitos e têm no máximo 255 caracteres. Maiúsculas e minúsculas são diferenciadas.

As seguintes palavras são reservadas:

final	int	byte	string	while	if
else	&&	||	!	<-	=
(	)	<	>	!=	>=
<=	,	+	-	*	/
;	begin	endwhile	endif	endelse	readln
write	writeln	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 (enquanto), estruturas de teste (se - então - senão), expressões aritméticas com inteiros e bytes, expressões lógicas e relacionais, além de atribuição, concatenação e comparação de igualdade entre strings. A ordem de precedência nas expressões é:

a) parênteses;
b) negação lógica (!);
c) multiplicação aritmética (*), lógica (&&) e divisão (/);
d) subtração (-), adição aritmética (+), lógica (||) e concatenação de strings (+);
e) comparação aritmética (=,!=,<,>,<=,>=) e entre strings (=).

Comentários são delimitados por /* */. A quebra de linha e o espaço podem ser usados livremente como delimitadores de lexemas.

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

Declarações Comandos

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, boolean, byte* ou *string* e *lista-de-ids* é uma série de 1 ou mais identificadores, separados por vírgulas. Variáveis podem ser opcionalmente inicializadas na forma: *id* <- *valor* , onde *id* é um identificador e *valor* uma constante decimal, precedida ou não de sinal negativo, hexadecimal, lógica ou do tipo string.

2. Declaração de constantes: é da forma: final *id* <- *valor;* , onde *id* é um identificador e *valor* uma constante numérica, precedida ou não de sinal negativo, hexadecimal, lógica ou do tipo string.

3. Comando de atribuição: é da forma *id* <- *expressão;*

4. Comando de repetição: pode assumir duas formas:

   while (*expressão*) *comando*
   while (*expressão*) *begin comandos endwhile*

   onde *expressão* é do tipo lógico e *comandos* é uma lista de zero ou mais comandos da linguagem.

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* podem ser substituídos por blocos da forma:

   if (expressão) begin lista_comandos1 endif else begin lista_comandos2 endelse

   onde as listas são sequências de comandos.

6. Comando nulo: é da forma *;* . Nada é executado neste comando.

7. Comando de leitura: é da forma *readln(id);* , onde *id* é um identificador de variável inteira, byte ou string.

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 ou do tipo string, 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 em grupos de até 3 alunos, sem qualquer participação de outros grupos e/ou ajuda de terceiros. No caso de grupos, as partes devem ser postadas por apenas um dos componentes, tanto no VERDE quanto no SGA. 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.

2. Os arquivos de código postados devem conter um cabeçalho com a identificação da disciplina e dos componentes do grupo. 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%.

3. A codificação do trabalho deve ser feita em linguagem C, C++ ou Java para avaliação no Ambiente VERDE ou em um compilador de linha (gcc, javac). 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.

4. O trabalho será avaliado em 3 etapas:
   a) as práticas TP1 e TP2 (15 pontos), em uma única versão final, deverão ser postadas no VERDE até 26/04/2020 em cada um dos testes cadastrados.
   b) a prática TP3 (10 pontos), em uma única versão final, deverá ser postada no VERDE até às 13:00 horas do dia 01/06/2020 em cada um dos testes cadastrados.
   c) a prática TP4 (15 pontos), em uma única versão final, deverá ser postadas no SGA até às 13:00 horas do dia 01/06/2020. A documentação consistindo do Esquema de Tradução deverá ser postada no SGA na mesma data. O TP4 será avaliado em vídeo conferência, conforme cronograma, devendo ser usada a versão do código entregue pelo SGA.

5. **Trabalhos iguais, na sua totalidade ou em partes, copiados, "encomendados" ou outras barbaridades do gênero, serão severamente penalizados. É responsabilidade do aluno manter o sigilo sobre seu trabalho, evitando que outros alunos tenham acesso a ele. No caso de cópia, ambos os trabalhos serão penalizados, independentemente de quem lesou ou foi lesado no processo.**

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

7. 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.

8. 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.

9. 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