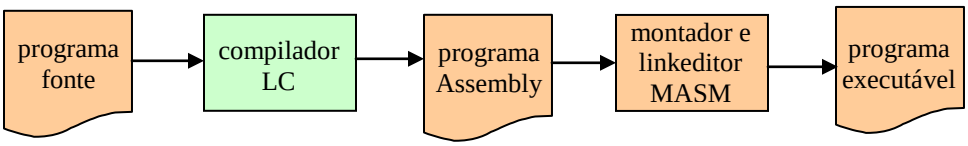
Pontifícia Universidade Católica de Minas Gerais
ICEI - Departamento de Ciência da Computação
Disciplina: Compiladores
Prof. Alexei Machado

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. **O formato das mensagens de erro será especificado posteriormente e deverá ser rigorosamente observado. O programa executável do compilador deve se chamar "LC" e receber 2 parâmetros da linha de comando (argumentos): o nome completo do programa fonte a ser compilado (extensão .L) e o nome completo do programa ASSEMBLY (extensão .ASM) a ser gerado.**



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*, *integer*, *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 XXh, onde XX 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, 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, começando necessariamente por uma letra e podem ter no máximo 255 caracteres. Não há distinção entre maiúsculas e minúsculas.

As seguintes palavras são reservadas:

const	integer	byte	string	while	if
else	and	or	not	==	=
(	)	<	>	<>	>=
<=	,	+	-	*	/
;	{	}	readln	write	writeln
TRUE	FALSE	boolean			

Os comandos existentes em "L" permitem atribuição a variáveis, entrada de valores pelo teclado e saída de valores para a tela, blocos (início - fim), 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 (not);
c) multiplicação aritmética (*), lógica (*and*) e divisão (/);
d) subtração (-), adição aritmética (+), lógica (or) e concatenação de strings (+);
e) comparação aritmética (==,<>,<,>,<=,>=) e entre strings (==).

Comentários se iniciam com // e terminam com a quebra de linha. 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 Bloco

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 *integer*, *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: *const 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. Blocos são da forma:

   { Comandos }

   Dentro do bloco pode haver zero ou mais comandos em qualquer ordem.

4. Comando de atribuição: é da forma *id = expressão;*

5. Comando de repetição: pode assumir duas formas:

   while expressão comando
   while expressão bloco

   onde *expressão* é do tipo lógico.

6. Comando de teste: pode assumir as formas:

   if expressão then comando1
   if expressão then comando1 else comando2

   comando1 e *comando2* podem ser substituídos por blocos. A *expressão* é do tipo lógico.

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

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

9. 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 dois ou três alunos, sem qualquer participação de outros grupos e/ou ajuda de terceiros. Cada aluno deve participar ativamente em todas as etapas do trabalho. Os componentes dos grupos devem ser informados até o dia 27/3, através de e-mail para alexei@pucminas.br e não poderão ser alterados durante o

semestre. Os alunos que não tiverem feito grupos até esta data serão agrupados pelo professor de maneira arbitrária, em grupos de 2 ou 3 alunos.

2. A codificação do trabalho deve ser feita em linguagem C, C++ ou Java, **em ambiente WINDOWS**. Os arquivos enviados devem poder ser compilados **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.

3. O trabalho será avaliado em 2 etapas:
   a) as práticas TP1 e TP2 (10 pontos), em uma única versão final, deverão ser postadas no SGA até às 13:00 horas do dia 06/04/2016, juntamente com a documentação, e apresentadas conforme o cronograma. O atraso na entrega implicará em perda de 3 pontos por dia.
   b) as práticas TP3 e TP4 (15 pontos), em uma única versão final, deverão ser postadas no SGA até às 13:00 horas do dia 02/06/2016, juntamente com a documentação, e apresentadas conforme o cronograma. **Para esta etapa não se admite atraso, ou seja, não serão avaliados trabalhos entregues após 2/6.**

4. Os trabalhos devem ser postados na forma de um arquivo compactado com software disponível no laboratório, com **tamanho máximo de 2MB**, e seu nome deve ser o número de matrícula de um dos componentes (Ex:346542.zip). **Os arquivos fontes devem estar no diretório raiz** e devem conter o nome de todos os componentes do grupo no início do código.

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 à **interface com o usuário**, uma vez que parte da correção será automatizada. Observe principalmente qual deve ser o **nome** do programa executável, seus argumentos de entrada e formatos das mensagens. Trabalhos com interfaces diferentes das especificadas correm o risco de **não serem avaliadas.**

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)
- Organização dos arquivos do projeto
- Parametrização
- Apresentação individual
- Documentação