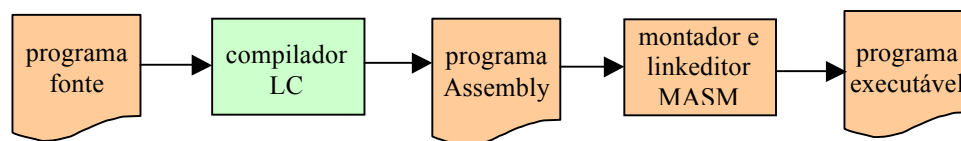


Trabalho Prático

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 2 tipos básicos explícitos: *char* e *integer*, além do tipo lógico que é implícito. 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 DD é um número hexadecimal. Constantes em formato alfanumérico são da forma ‘c’ onde c é um caractere imprimível. O tipo *integer* é um escalar que varia de -32768 a 32767, ocupando 2 bytes. Além dos tipos básicos a linguagem permite a definição de vetores unidimensionais de caracteres e inteiros, com até 4kbytes. Um *string* é um vetor de caracteres que quando armazenado em memória, tem seu conteúdo útil finalizado pelo caractere ‘\$’. Constantes que representam strings são delimitadas por aspas e não devem conter aspas, quebra de linha ou \$. Entretanto, esses caracteres são permitidos nos vetores de caracteres. Dessa forma, vetores e strings são compatíveis entre si, ficando a cargo do programador o controle dos seus conteúdos e tamanhos. tipo lógico assume valores 0 (falso) e 1 (verdadeiro), ocupando um byte de memória.

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, porcentagem, circunflexo, arroba, exclamação, interrogação, maior, menor e igual, além da quebra de linha (bytes 0Dh e 0Ah). Qualquer outro caractere é considerado inválido.

Os identificadores de constantes e variáveis são compostos de letras, dígitos, sublinhado e ponto, não podem começar com dígitos, nem conter apenas sublinhados ou pontos, e têm no máximo 255 caracteres. Maiúsculas e minúsculas não são diferenciadas.

As seguintes palavras e suas variações com maiúsculas e minúsculas são reservadas:

const	var	integer	char	for	if
else	and	or	not	=	to
()	<	>	<>	>=
<=	,	+	-	*	/
;	{	}	then	readln	step
write	writeln	%	[]	do

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 para), estruturas de teste (se - então - senão), expressões aritméticas com inteiros e caracteres, expressões lógicas e relacionais, manipulação de vetores, além de atribuição e comparação de igualdade entre strings. A ordem de precedência nas expressões é:

- parênteses;
- negação lógica (not);
- multiplicação aritmética (*) e lógica (and), quociente da divisão (/) e resto da divisão (%);
- subtração (-), adição aritmética (+) e lógica (or);
- 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 Bloco_de_Comandos fim_arquivo

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

- Declaração de variáveis: inicia-se pela palavra reservada *Var*, à qual seguem-se uma ou mais listas de declarações da forma: *tipo lista-de-ids; ,* onde *tipo* pode ser *integer* ou *char* e *lista-de-ids* é uma série de 1 ou mais identificadores de escalares ou vetores, separados por vírgulas. Variáveis escalares podem ser opcionalmente inicializadas na

forma: $id = valor$, onde id é um identificador. Para inteiros, $valor$ é uma constante decimal precedida ou não de sinal negativo; Para caractere, uma constante hexadecimal ou alfanumérica. Declarações de vetores são da forma $id[tam]$, onde tam é uma constante inteira maior que 0. O tamanho em bytes de um vetor não pode ultrapassar 4Kbytes. Vetores não podem ser inicializados na declaração.

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, hexadecimal ou caractere alfanumérico.
3. Comando de atribuição: é da forma $id = expressão;$
4. Comando de repetição: pode assumir duas formas:

For $id=expressão\ to\ expressão\ step\ constante\ do\ comando$
For $id=expressão\ to\ expressão\ step\ constante\ do\ \{ comandos \}$

onde $expressão$ e $constante$ são do tipo inteiro e $comandos$ é uma lista de zero ou mais comandos da linguagem. A definição de $step$ é opcional, sendo 1 se não estiver especificada.

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

if $expressão\ then\ comando1$
if $expressão\ then\ comando1\ else\ comando2$

$comando1$ e/ou $comando2$ podem ser independentemente substituídos por blocos da forma:

if $expressão\ then\ \{ lista_comandos1 \}\ else\ \{ lista_comandos2 \}$

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, caractere alfanumérico ou string. Caracteres e strings são lidos da mesma maneira, sem que o usuário precise coloca-los entre aspas ou apóstrofes. Números inteiros 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 inteiras, 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 em 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. 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. 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. 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 (10 pontos), em uma única versão final, deverão ser postadas no VERDE até 04/10/2020 em cada um dos testes cadastrados e apresentados conforme cronograma.
 - b) a prática TP3 (10 pontos) deverá ser postada no VERDE até 22/11/2020 em uma versão separada ou juntamente com a prática T4, em cada um dos testes cadastrados.
 - c) a prática TP4 (15 pontos), em uma única versão final com a prática TP3 (compilador completo), deverá ser postada no SGA até 22/11/2020. A documentação consistindo do Esquema de Tradução deverá ser postada no SGA na mesma data. Os trabalhos finais postados devem estar na forma de um arquivo compactado **em formato zip**, com **tamanho máximo de 3MB**, 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. O TP4 será avaliado em vídeo conferência, conforme cronograma, devendo ser usada a versão do código entregue pelo SGA.

Não se admite atraso em qualquer etapa.

5. **Trabalhos iguais, na sua totalidade ou em partes, copiados, reaproveitados, “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. No caso de cópia, ambos os grupos 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)
 - Organização dos arquivos do projeto
 - Parametrização
 - Apresentação individual
 - Documentação