1º Trabalho[1]

LPAS - Linguagem de Programação para Aritmética Simples

Quinta-feira, 25 de fevereiro de 2021.

Uma linguagem de máquina denominada LPAS - Linguagem de Programação para Aritmética Simples é usada para escrever programas que executam as operações aritméticas somente com números inteiros. As instruções LPAS são apresentadas na Tabela 1. LPAS exige que:

1. Cada linha de código deve ter apenas uma única instrução.
2. As instruções e os seus argumentos variáveis sejam todos escritos em maiúsculo.
3. Os comentários devem iniciar com ponto-e-vírgula.

Exemplos de programas LPAS:

```
00 ; Programa: soma.lpas
01 ; Descrição: Realização a soma de 2 números inteiros.
02 ; --------------------------------------------------------------------
03 READ X    ; X recebe o valor lido do teclado
04 READ Y    ; Y recebe o valor lido do teclado
05 LOAD X    ; carrega o valor de X no registrador (registrador ⇦ X)
06 ADD Y     ; soma Y com X e coloca o resultado no registrador (registrador ⇦ registrador + Y)
07 STORE Z   ; armazena o resultado da soma na variável Z (Z ⇦ registrador)
08 WRITE Z   ; escreve no vídeo o valor de Z (o resultado da soma)
09 HALT      ; finaliza o programa
```

```
00 ; Programa: contar1a15.lpas
01 ; Descrição: Exibe no vídeo os números ímpares de 1 a 15.
02 ; ----------------------------------------------------------------------
03 MOV X, 1     ; X ⇦ 1
04 LOAD X       ; registrador ⇦ X
05 RDIV 2       ; registrador ⇦ X % 2
06 JPZERO 8     ; não exibe o valor de X se é par, ou seja, o resultado de X % 2 é zero.
07 WRITE X      ; escreve o valor de X no vídeo se X é ímpar, ou seja, o resultado de X % 2 é um.
08 LOAD X       ; registrador ⇦ X
09 SUB 15       ; registrador ⇦ registrador - 15
10 JPZERO 15    ; finaliza o programa se o valor de X é 15, ou seja, registrador é zero
11 LOAD X       ; registrador ⇦ X
12 ADD 1        ; registrador ⇦ X + 1
13 STORE X      ; X ⇦ registrador
14 JUMP 5       ; loop: desvia para o endereço de memória 5 correspondente a instrução RDIV 2
15 HALT         ; finaliza o programa
```

[1] Atualizado em 02/03/2021.

Tabela 1: Instruções de máquina LPAS

Código	Instrução	Descrição
10	READ	Lê um valor do teclado para uma variável. Ex.: **READ X** ; lê um valor do teclado e armazena na variável X
11	WRITE	Escreve no vídeo o valor armazenado em uma variável. Ex.: **WRITE X** ; escreve no vídeo o valor da variável X
20	MOV	Movimenta o valor de uma variável ou um número para outra posição de memória. Ex.: **MOV X, Y** ; X ⇦ Y **MOV X, 20** ; X ⇦ 20
21	LOAD	Carrega o valor de uma variável para o registrador. Ex.: **LOAD X** ; registrador ⇦ X
22	STORE	Armazena o valor do registrador em uma variável. Ex.: **STORE X** ; X ⇦ registrador
30	ADD	Adiciona o valor do registrador ao valor de uma variável ou número e guarda o resultado no registrador. Ex.: **ADD X** ; registrador ⇦ registrador + X **ADD 20** ; registrador ⇦ registrador + 20
31	SUB	Subtrai o valor do registrador do valor de uma variável ou número e guarda o resultado no registrador. Ex.: **SUB X** ; registrador ⇦ registrador - X **SUB 20** ; registrador ⇦ registrador - 20
32	MUL	Multiplica o valor do registrador com o valor de uma variável ou número e guarda o resultado no registrador. Ex.: **MUL X** ; registrador ⇦ registrador * X **MUL 20** ; registrador ⇦ registrador * 20
33	DIV	Divide o valor do registrador pelo valor de uma variável ou número e guarda o resultado no registrador. Ex.: **DIV X** ; registrador ⇦ registrador / X **DIV 20** ; registrador ⇦ registrador / 20
34	RDIV	Calcula o resto da divisão do registrador pelo valor de uma variável ou número e guarda o resultado no registrador. Ex.: **RDIV X** ; registrador ⇦ registrador % X **RDIV 20** ; registrador ⇦ registrador % 20
40	JUMP	Desvia para um endereço de memória. Ex.: **JUMP 20** ; desvia para o endereço 20 da memória
41	JPNEG	Desvia para um endereço de memória se o valor do registrador for negativo. Ex.: **JPNEG 20** ; desvia para o endereço 20 se o acumulador for negativo

Código	Instrução	Descrição
42	JPZERO	Desvia para um endereço de memória se o valor do registrador for zero. Ex.: **JPZERO 20** ; desvia para o endereço 20 se o acumulador for zero
50	HALT	Finaliza o programa.

Os programas LPAS são executados por uma Máquina de Execução (ME). Essa máquina é responsável por carregar o programa para a memória da ME e executá-lo. A ME possui duas memórias e um único registrador que permite armazenar um número inteiro. Uma memória armazena todos os programas que podem ser executados pela ME e a outra as variáveis do programa que está em execução. Um programa é executado por vez.

Veja nos programas mostrados acima que cada linha de código LPAS é precedida pelo endereço de sua localização na memória do programa. Essa memória é formada pelas linhas de código (instruções) que compõem o programa.

As variáveis do programa LPAS devem ser todas do tipo inteiro. Um programa pode ter no máximo 100 instruções ou linhas de código LPAS.

O limite de memória da ME é de cinco programas, ou seja, esse é o número de programas que podem ser carregados para a ME, sendo que somente um pode ser executado por vez. O programador LPAS pode, por exemplo, carregar três programas e depois escolher qual ele deseja executar. Os programas devem ser nomeados para o programador escolher que programa LPAS ele quer executar.

Desenvolva um programa chamado `LPAS` para executar os programas armazenados na memória da máquina de execução por meio do *prompt* de comandos abaixo.

```
lpas>
```

Esse *prompt* será usado pelo programador LPAS para executar os comandos a seguir. Após a execução de cada comando o *prompt* deve ser reexibido, pois essa é a forma de interação do usuário-programador com a máquina de execução LPAS.

Nos comandos o uso da extensão (.lpas) no nome do arquivo é opcional, ou seja, o usuário pode ou não indicar a extensão, sendo assim o programa deve considerar as duas situações.

1. Carregar programa (*load*)

Carrega para a memória da máquina de execução o programa LPAS armazenado no arquivo indicado no parâmetro `<nomePrograma>` e localizado no diretório definido por `[<localização>]`. Se a localização for omitida o arquivo deve ser lido do diretório atual. O programa lido do arquivo deve ser armazenado como um objeto da classe `Programa` (ver declaração abaixo) na memória da máquina de execução.

Sintaxe: `lpas> load [<localização>]<nomePrograma>`

Exemplos: 1. Carrega os programas `mul.lpas`, `add.lpas` e `div.lpas` para a memória da máquina de execução.

```
lpas> load mul.lpas
lpas> load d:\programas\lpas\add
```

```
lpas> load d:\programas\lpas\div
```

2. É possível carrega os três programas LPAS com um comando apenas, assim:

```
lpas> load mul add div
```

2. Executar programa (*run*)

Executa um programa LPAS armazenado na memória da máquina de execução indicado no parâmetro `<nomePrograma>`.

Sintaxe: `lpas> run <nomePrograma>`

Exemplos: Executa os programas `produto.lpas` e `contador.lpas`.

```
lpas> run produto.lpas
lpas> run contador
```

3. Exibir programas da máquina de execução (*show me*)

Exibe os nomes de todos os programas LPAS armazenados na memória da máquina de execução.

Sintaxe: `lpas> show me`

Exemplo: `lpas> show me`

```
Os programas LPAS na memória da máquina de execução são:

    1: mul
    2: add
    3: div
```

4. Exibir programa (*show*)

Exibe o código-fonte de um programa LPAS armazenado na memória da máquina de execução indicado no parâmetro `<nomePrograma>`.

Sintaxe: `lpas> show <nomePrograma>`

Exemplos: Exibe o código-fonte dos programas `produto.lpas e soma.lpas`.

```
lpas> show produto.lpas
lpas> show soma
```

5. Encerrar programa (*exit*)

Encerra a máquina de execução LPAS, ou seja, encerra o programa `LPAS`.

Sintaxe: `lpas> exit`

Desenvolva a implementação da classe `Programa` separando-a de sua interface que está listada abaixo. Essa classe representa a estrutura do programa LPAS.

```
#ifndef PROGRAMA_H
#define PROGRAMA_H

#include <string>

using namespace std;

// Número máximo de instruções do programa.
constexpr unsigned short NUMERO_MAXIMO_DE_INSTRUCOES = 100;

// Esta classe representa a estrutura de um programa LPAS.
class Programa {
public:
	// Inicia as variáveis do programa com valores default.
	Programa();

	// Inicia o programa com o nome especificado.
	Programa(string nome);

	/*
	  Lê as instruções de um programa LPAS para a memória.
	  Retorna true se as instruções foram lidas com sucesso e false se ocorreu algum erro durante a leitura.
	*/
	bool carregar();

	// Obtém o número de instruções do programa LPAS.
	unsigned short getNumeroDeInstrucoes();

	/* Obtém uma instrução LPAS armazenada na posição de memória indicada por endereco. Se o
	   endereço de memória for inválido, retorna uma string nula.
	*/
	string obterInstrucao(unsigned short endereco);

	/* Altera uma instrução armazenada na posição de memória indicada por endereco por uma nova
	   instrucao. Retorna true se a instrução foi alterada. Se o endereço de memória for inválido, retorna
	   false.
	*/
	bool alterarInstrucao(string instrucao, unsigned short endereco);

	// Exibe na tela as instruções do programa.
	void exibir();

	void setNome(string nome);
	string getNome();
```

```
private:
        // Número de instruções do programa.
        unsigned short numeroDeInstrucoes;

        // Nome do programa.
        string nome;

        /* Memória que armazena as instruções LPAS que compõem o programa.
           Cada linha do programa é armazenada em uma posição do vetor.
         */
        string instrucoes[NUMERO_ MAXIMO_DE_INSTRUCOES];
};
#endif
```

Desenvolva a implementação da classe ErroExecucao separando-a de sua interface que está listada abaixo. Essa classe representa a estrutura de um erro que pode ocorrer durante a execução de um programa LPAS.

```
#ifndef ERRO_EXECUCAO_H
#define ERRO_EXECUCAO_H

#include <string>

using namespace std;

/* Códigos resultantes da execução do programa LPAS. Os códigos válidos são:

        0 = execução bem sucedida;
        1 = instrução LPAS inválida;
        2 = argumento de instrução LPAS inválido;
        3 = argumento de instrução LPAS ausente;
        4 = duas ou mais instruções LPAS na mesma linha de código;
        5 = símbolo inválido.
*/
enum class Erro { EXECUCAO_BEM_SUCEDIDA, INSTRUCAO_LPAS_INVALIDA,
                  ARGUMENTO_INSTRUCAO_LPAS_INVALIDO,
                  ARGUMENTO_INSTRUCAO_LPAS_AUSENTE, MUITAS_INSTRUCOES, SIMBOLO_INVALIDO };

/*
  Esta classe representa a instrução, o número da linha, o nome do programa e o tipo de erro que pode
  ocorrer durante a execução de um programa LPAS.
*/
class ErroExecucao {
public:
        ErroExecucao();
        ErroExecucao(string instrucao, string nomePrograma, unsigned short numeroLinha, Erro erro);

        string getInstrucao();
        string getNomePrograma();
        unsigned short getNumeroLinha();
        Erro getErro();
```

```
        void setInstrucao(string instrucao);
        void setNomePrograma(string nomePrograma);
        void setNumeroLinha(unsigned short numeroLinha);
        void setErro(Erro erro);

private:
        string instrucao, nomePrograma;
        unsigned short numeroLinha = 0;
        Erro erro;
};
#endif
```

Desenvolva a implementação da classe MaquinaExecucao separando-a de sua interface que está listada abaixo. Essa classe representa a estrutura da máquina de execução LPAS.

```
#ifndef MAQUINA_EXECUCAO_H
#define MAQUINA_EXECUCAO_H

// Número máximo de variáveis do programa.
constexpr unsigned short NUMERO_MAXIMO_DE_VARIAVEIS = 25;

// Número máximo de programas da máquina de execução.
constexpr unsigned short NUMERO_MAXIMO_DE_PROGRAMAS = 5;

// Indica endereço inválido na memória da máquina de execução.
constexpr unsigned short ENDERECO_INVALIDO = 100;

// Esta classe representa a estrutura da máquina de execução de um programa  LPAS.
class MaquinaExecucao {
public:
        // Inicia as variáveis da máquina de execução.
        MaquinaExecucao();

        /* Carrega um programa LPAS para a memória. Retorna true se o programa foi carregado, caso
           contrário retorna false.
        */
        bool carregar(Programa programa);

        // Obtém o número de programas LPAS carregados na memória da máquina de execução.
        int getNumeroDeProgramas();

        /* Pesquisa o nome do programa LPAS na memória da máquina de execução. Retorna o endereço de
           memória do programa ou ENDERECO_INVALIDO se o nome não for localizado.
        */
        unsigned short pesquisarPrograma(string nome);

        /* Obtém um programa LPAS armazenado na posição de memória indicada por endereco. Se o
           endereço de memória for inválido, retorna um objeto com atributos nulos.
        */
        Programa obterPrograma(unsigned short endereco);
```

```
        /* Executa o programa armazenado na posição de memória indicada por endereco. Esta função deve
           usar a função obterCodigoInstrucao() para obter o código de máquina da instrução LPAS a ser
           executada. Retorna um objeto que permite identificar a instrução, o número da linha, o nome do
           programa e o tipo de erro que pode ocorrer durante a execução de um programa LPAS.
        */
        ErroExecucao executarPrograma(unsigned short endereco);

        /* Retorna um objeto que permite identificar a instrução, o número da linha, o nome do programa e o
           tipo de erro que pode ocorrer durante a execução de um programa LPAS.
        */
        ErroExecucao getErroExecucao();

private:
        // Número de programas LPAS carregados na memória da máquina de execução.
        unsigned short numeroDeProgramas;

        // Armazena os programas LPAS na memória da máquina de execução.
        Programa memoria[NUMERO_DE_PROGRAMAS];

        // Registrador da máquina de execução LPAS.
        int registrador;

        // Variáveis do programa LPAS que está sendo executado na máquina de execução.
        int variaveis[NUMERO_DE_VARIAVEIS];

        /* Identifica a instrução, o número da linha, o nome do programa e o tipo de erro que pode ocorrer
           durante a execução de um programa LPAS.
        */
        ErroExecucao erroExecucao;

        /* Converte a instrução mnemônica LPAS para o seu código de máquina equivalente. Ver Tabela 1.
           Retorna o código de máquina da instrução LPAS.
        */
        unsigned short obterCodigoInstrucao(string instrucao);

        /* Define a instrução, o número da linha, o nome do programa e o erro ocorrido na execução do
           programa.
        */
        void definirErroExecucao(string nomePrograma, string instrucao, unsigned short linha, Erro erro);

        /* Executa a instrução LPAS usando o valor armazenado em argumento ou na posição de memória
           indicada por enderecoVariavel. Retorna o código da instrução executada.
        */
        unsigned short executarInstrucao(unsigned short codigoInstrucao, unsigned short enderecoVariavel,
                                         int argumento);
};
#endif
```

- Critérios de avaliação

1. O trabalho será avaliado considerando:

a. A validação dos dados fornecidos pelo usuário.
b. A lógica empregada na solução do problema.
c. O funcionamento do programa.
d. O conhecimento da linguagem de programação C++.
e. A implementação dos conceitos de orientação a objetos.
f. O uso do princípio do menor privilégio[2].
g. Código fonte sem erros e sem advertências do compilador.
h. Código fonte legível, indentado, organizado e comentado.
i. Identificadores significativos para aprimorar a inteligibilidade do código fonte.

2. O programa deve ser desenvolvido integralmente usando apenas os recursos da linguagem C++ e do *Microsoft Visual Studio Community* 2019, versão 16.8. Programas desenvolvidos em outras linguagens, mesmo que parcialmente, receberão nota zero.

3. Para que o programa seja avaliado o código deve executar com sucesso. Programas que apresentarem erros de compilação e/ou ligação receberão nota zero.

4. Trabalhos com plágio, ou seja, programas com código fonte copiados de outra pessoa (cópia integral ou parcial) receberão nota zero.

5. O desenvolvimento do trabalho é individual.

6. Incluir em cada classe apenas as definições de tipos de dados, variáveis, constantes e métodos que forem essenciais para a funcionalidade da mesma.

7. Escrever funções e métodos específicos, ou seja, com atribuição clara e objetiva.

Exemplo: Pesquisa um nome em um vetor de *strings*. Retorna a posição do nome no vetor se ele for encontrado ou -1 caso contrário.

```
int pesquisarNome(string vetor[], string nome);
```

A descrição dessa função deixa claro que não é atribuição dela ler o nome via algum dispositivo de E/S e nem exibir o resultado da consulta, somente realizar a pesquisa do nome no vetor e devolver o resultado.

8. Não escrever código redundante.

9. É proibido modificar os nomes de arquivos, identificadores, os protótipos de função, as declarações e/ou definições de métodos e classes fornecidos neste texto ou em anexo.

10. É permitido acrescentar novas declarações e/ou definições de classes, métodos, variáveis e constantes desde que estejam de acordo com os critérios acima.

11. Use a classe ArquivoTexto fornecida no projeto ArquivoTexto.7z para acessar os arquivos de texto que possui os códigos-fontes LPAS.

[2] O **princípio do menor privilégio** declara que deve ser concedido ao código somente a quantidade de privilégio e acesso de que ele precisa para realizar sua tarefa designada, não mais que isso.

- Instruções para entrega do trabalho

1. Crie uma solução com o nome LPAS e um projeto com o seu nome e sobrenome, por exemplo: AyrtonSenna.
2. Limpe a solução para apagar todos os arquivos OBJ da pasta *Debug* do projeto. Confira se esses arquivos realmente foram excluídos da pasta *Debug*. De preferência exclua todo o conteúdo dessa pasta.

3. Antes de submeter os exercícios via SIGAA, compacte apenas o diretório do projeto para criar um arquivo 7z com o seu nome e sobrenome, por exemplo: AyrtonSenna.7z.

   Não inclua no arquivo 7z o diretório da solução LPAS, somente o diretório do projeto que possui o seu nome e sobrenome.

   Assim o tamanho final do arquivo 7z não ultrapassará o limite de 10 MB do SIGAA, porque o conteúdo do diretório oculto .vs, criado dentro da pasta da solução, não será compactado, reduzindo drasticamente o tamanho final do arquivo.

   Para compactar o projeto use o *software* livre de código aberto 7-Zip, que está disponível em https://www.7-zip.org/download.html.

- Data de entrega

Segunda-feira, 8 de março de 2021.

- Valor do trabalho

10,0 pontos.

Prof**. M**árlon **O**liveira **d**a **S**ilva
marlon.silva@ifsudestemg.edu.br