Faculdade de Computação e Informática
Ciência da Computação
Estrutura de Dados I – 3ª etapa – 2025.2
Professores: Alexandre dos Santos Mignon, Charles Boulhosa Rodamilans, Ivan Carlos Alcântara de Oliveira
Elaborado por: André Kishimoto

Atividade Apl2 – Interpretador de linguagem Assembly simplificada

No campo da Ciência da Computação, as linguagens de programação podem ser classificadas em duas categorias principais: *linguagens compiladas* e *linguagens interpretadas*.

Linguagens compiladas, como C e C++, são traduzidas inteiramente em código de máquina antes da execução. O compilador realiza essa tradução, gerando um arquivo executável. Uma vez compilado, o programa pode ser executado diretamente pela máquina, sem a necessidade de um interpretador. Isso geralmente resulta em programas mais rápidos, mas uma desvantagem é que o código precisa ser recompilado a cada alteração.

Por outro lado, linguagens interpretadas, como Python e JavaScript, são executadas por um programa especial chamado *interpretador*. Em vez de traduzir o código inteiro para código de máquina de uma vez, o interpretador lê o código-fonte linha por linha e executa as instruções diretamente. Essa abordagem permite uma maior flexibilidade, já que o código pode ser alterado e testado sem a necessidade de recompilação, mas tem a desvantagem de que os programas podem ser mais lentos.

Nesta atividade, você implementará um REPL (*Read-Evaluate-Print-Loop*) que aceita entradas via teclado e que também é um editor e *interpretador* de uma versão simplificada de Assembly.

Desenvolvimento do programa

O programa Java a ser desenvolvido na atividade deve implementar um REPL (*Read-Evaluate-Print-Loop*) que aceita entradas via teclado e que avalia e valida as entradas de acordo com os requisitos descritos a seguir.

REPL: Comandos válidos (*case insensitive*)

Comando	Descrição
LOAD <ARQUIVO.ED1>	Carrega o arquivo indicado em **<ARQUIVO.ED1>**, cujo conteúdo deve ser um código escrito na linguagem Assembly simplificada que é descrita neste documento. Cada linha do arquivo deve ser mapeada para um nó de uma lista encadeada. O programa deve verificar se já existe um arquivo aberto e, caso existam modificações não salvas na memória, deve solicitar a confirmação se a pessoa deseja salvar as alterações em arquivo antes de abrir outro. O programa deve exibir uma mensagem de notificação (tais como arquivo carregado com sucesso, erro ao carregar o arquivo, operação cancelada).
LIST	Exibe em tela o conteúdo completo do código-fonte existente na memória, de 20 em 20 linhas.
RUN	Ativa o interpretador, que executa o código-fonte carregado na memória. Caso não exista código na memória, o programa deve exibir uma mensagem de notificação.
INS <LINHA> <INSTRUÇÃO>	Insere uma nova linha de código, na posição correta da lista, sendo que **<LINHA>** não pode ser negativa (as linhas que antecedem as instruções Assembly indicam a ordem de execução do código). Caso a **<LINHA>** já exista na lista, seu conteúdo deve ser atualizado com a nova instrução. O programa deve exibir uma mensagem de notificação (por exemplo, linha inserida ou linha atualizada).
DEL <LINHA>	Remove a **<LINHA>** de código da lista, caso exista, e exibe uma mensagem de notificação (linha removida). Caso a **<LINHA>** não exista, o programa deve exibir uma mensagem de notificação (linha inexistente).

Comando	Descrição
DEL <LINHA_I> <LINHA_F>	Remove as linhas de código da lista no intervalo **<LINHA_I>** e **<LINHA_F>** (inclusive) e exibe uma mensagem de notificação, indicando quais linhas e instruções foram removidas. O programa deve validar o intervalo inicial e final e exibir uma mensagem de notificação, caso os intervalos sejam inválidos.
SAVE	Salva o código-fonte da memória para o arquivo atualmente aberto. O programa deve exibir uma mensagem de notificação (por exemplo, arquivo salvo com sucesso ou erro ao salvar o arquivo).
SAVE <ARQUIVO.ED1>	Salva o código-fonte da memória no arquivo especificado em **<ARQUIVO.ED1>**. Caso o arquivo já exista, o programa deve perguntar se a pessoa deseja sobrescrever o arquivo existente. O programa deve exibir uma mensagem de notificação (por exemplo, arquivo salvo com sucesso, erro ao salvar o arquivo, arquivo não salvo).
EXIT	Encerra o programa. Caso existam modificações não salvas na lista, o programa deve solicitar confirmação se a pessoa deseja salvar as alterações em arquivo antes de encerrar o programa.

Observações importantes para a implementação do programa

1) O uso de uma lista encadeada, *conforme estudada e implementada em aula*, é um requisito do programa.

2) A equipe desenvolvedora do programa ficou encarregada de definir os critérios de validação das entradas inseridas pela pessoa usuária do programa, além das indicadas neste documento.

3) Para cada comando realizado, o programa deve exibir uma mensagem adequada. A equipe deve definir as mensagens, seguindo as especificações e critério de validação do programa.

4) Ao exibir o código-fonte em tela, o número da linha de cada instrução correspondente também deve ser exibido.

5) Os tipos de dados criados para representar um nó da lista e uma lista encadeada devem estar em seus próprios arquivos `.java` e o código contendo o método `main()` também deve estar localizado em seu próprio arquivo `.java`. Dessa forma, o projeto deve ter, no mínimo, 3 arquivos `.java`.

A linguagem Assembly simplificada

A linguagem Assembly simplificada desta atividade suporta as seguintes instruções:

Instrução	Descrição
`mov x y`	Copia o valor `y` para o registrador `x`.
`inc x`	Incrementa o valor armazenado no registrador `x` em 1.
`dec x`	Decrementa o valor armazenado no registrador `x` em 1.
`add x y`	Adiciona o valor de `y` ao registrador `x`, armazenando o resultado em `x`.
`sub x y`	Subtrai o valor de `y` do registrador `x`, armazenando o resultado em `x`.
`mul x y`	Multiplica o valor de `x` pelo valor de `y`, armazenando o resultado em `x`.
`div x y`	Divide o valor de `x` pelo valor de `y`, armazenando o resultado em `x`.
`jnz x y`	Se o valor no registrador `x` for diferente de zero, o interpretador pula para a linha de número `y`, desde que a linha seja válida.
`out x`	Exibe o valor do registrador `x` em tela e pula uma linha.

Regras da linguagem

Considerando a tabela de instruções da linguagem desta atividade,

- `x` representa um registrador, que é uma variável identificada por uma única letra do alfabeto (`A`-`Z`). Ou seja, a linguagem assume que existem 26 registradores disponíveis na memória.
- `y` pode ser tanto um valor constante (número inteiro positivo ou negativo) quanto o conteúdo de outro registrador.
- Um registrador só pode ser usado como leitura caso algum valor já tenha sido atribuído a ele. Isto é, um registrador só pode aparecer nas instruções `inc`, `dec`, `add`, `sub`, `mul`, `div`, `jnz`, `out` e no lugar de `y` na instrução `mov`, se ele já teve algum valor atribuído com a instrução `mov` (apareceu no lugar de `x` na instrução `mov`).
- A linguagem é *case insensitive*, ou seja, `MOV` e `mov` são considerados a mesma instrução.
- Antes de cada instrução, deve ser especificado um número de linha, similar ao estilo das linguagens antigas como o BASIC, sendo que não podem existir duas ou mais instruções iniciando com o mesmo número de linha.
- A instrução `jnz` faz um salto condicional para o número de linha especificado em `y`, caso o registrador indicado por `x` não contenha o valor zero.
- O código-fonte nesta linguagem deve ser salva com extensão `.ed1`.

Exemplo de código válido

```
10 mov a 5
20 mov b 7
30 add a b
40 inc a
50 dec b
60 jnz b 30
70 mov c a
80 sub c 2
90 out c
```

O código acima exibe em tela o número 38.
Explicação:

Linha	Instrução	Descrição
10	mov a 5	O valor 5 é copiado no registrador A.
20	mov b 7	O valor 7 é copiado no registrador B.
30	add a b	O valor de B (7) é adicionado a A (5), fazendo com que o valor de A agora seja 12.
40	inc a	O registrador A é incrementado, passando de 12 para 13.
50	dec b	O registrador B é decrementado, passando de 7 para 6.
60	jnz b 30	A instrução jnz verifica o valor de B. Como B é diferente de zero, o programa volta para a linha 30. O ciclo entre as linhas 30 e 60 continua até que B seja decrementado a zero. Quando B for zero, o programa segue para a linha 70.
70	mov c a	O valor final de A (40) é copiado para o registrador C.
80	sub c 2	O valor 2 é subtraído do registrador C, fazendo com o que o valor de C agora seja 38.
90	out c	O valor do registrador C (38) é exibido em tela.

Exemplo de código inválido

```
10 mov a 5
10 mov b 3
20 mov b x
30 add c 10
40 inc 10
50 jnz a linha20
60 mul a b c
70 jnz a 90
80 out 9999
```

Explicação dos problemas encontrados no exemplo de código inválido:

Linha	Instrução	Descrição
`10`	`mov a 5`	Essa instrução está correta.
`10`	`mov b 3`	Observe que a instrução começa com a linha 10, que já foi definida na instrução anterior.
`20`	`mov b x`	O registrador `X` é inválido porque ainda não possui valor definido e, portanto, não pode ser usado como leitura.
`30`	`add c 10`	O registrador `C` é inválido porque ainda não possui valor definido e, portanto, não pode ser usado como leitura.
`40`	`inc 10`	A instrução `inc` só pode ser aplicada a registradores.
`50`	`jnz a linha20`	O segundo argumento de `jnz` deve ser um número de linha para o qual o código deve saltar. Neste caso, `linha20` não é um número e nem um registrador válido.
`60`	`mul a b c`	A instrução `mul` aceita apenas dois argumentos.
`70`	`jnz a 90`	A linha 90 não existe no código, portanto, o salto condicional é inválido.
`80`	`out 9999`	A instrução `out` só pode ser aplicada a registradores.

Para instruções inválidas, o interpretador deve mostrar uma mensagem de erro coerente.

Arquivo .ed1 de teste

A equipe desenvolvedora do programa deve criar seus próprios arquivos `.ed1` de teste, com a finalidade de:

- Demonstrar o uso correto de todas as instruções da linguagem Assembly simplificada.
- Demonstrar todos os cenários em que as instruções do Assembly simplificado são usadas de maneira incorreta.

A quantidade de arquivos `.ed1` de teste fica a critério da equipe, sendo que os exemplos apresentados neste documento não podem ser usados *ipsis litteris*.

Exemplo de execução do programa

Observação: Conteúdo com **texto azul** representa o que foi digitado pela pessoa usuária do programa.

```
> TESTE
Erro: comando inválido.
> LOAD c:\tmp\teste1.ed1
Arquivo 'c:\tmp\teste1.ed1' carregado com sucesso.
> LIST
10 mov a 5
20 mov b 2
30 inc a
40 dec b
50 jnz b 30
60 mov x a
70 sub x 2
80 out x
> RUN
5
> INS 71 out a
Linha inserida:
71 out a
> LIST
10 mov a 5
20 mov b 2
30 inc a
40 dec b
50 jnz b 30
60 mov x a
70 sub x 2
71 out a
80 out x
> RUN
7
5
> DEL 1
Erro: linha 1 inexistente.
> DEL 71
Linha removida:
71 out a
> INS 5 mov j 30
Linha inserida:
5 mov j 30
> INS 50 jnz b j
Linha:
50 jnz b 30
Atualizada para:
50 jnz b j
```

```
> LIST
5 mov j 30
10 mov a 5
20 mov b 2
30 inc a
40 dec b
50 jnz b j
60 mov x a
70 sub x 2
80 out x
> RUN
5
> DEL 25 50
Linhas removidas:
30 inc a
40 dec b
50 jnz b j
> LIST
5 mov j 30
10 mov a 5
20 mov b 2
60 mov x a
70 sub x 2
80 out x
> DEL 60 10
Erro: intervalo inválido de linhas.
> INS -10 out x
Erro: linha -10 inválida.
> SAVE
Arquivo 'c:\tmp\teste1.ed1' salvo com sucesso.
> LOAD d:\facul 2024\trabalhos\teste.ed1
Erro ao abrir o arquivo 'd:\facul 2024\trabalhos\teste.ed1'.
> LIST
5 mov j 30
10 mov a 5
20 mov b 2
60 mov x a
70 sub x 2
80 out x
> SAVE
Arquivo 'c:\tmp\teste1.ed1' salvo com sucesso.
> DEL 5
Linha removida:
5 mov j 30
> LOAD c:\tmp\teste1.ed1
Arquivo atual ('c:\tmp\teste1.ed1') contém alterações não salvas.
Deseja salvar? (S/N)
> S
Arquivo 'c:\tmp\teste1.ed1' salvo com sucesso.
```

```
Arquivo 'c:\tmp\teste1.ed1' carregado com sucesso.
> LIST
10 mov a 5
20 mov b 2
60 mov x a
70 sub x 2
80 out x
> RUN
3
> DEL 20
Linha removida:
20 mov b 2
> DEL 60
Linha removida:
60 mov x a
> RUN
Erro: registrador X inválido.
Linha: 70 sub x 2
Erro: registrador X inválido.
Linha: 80 out x
> EXIT
Arquivo atual ('c:\tmp\teste1.ed1') contém alterações não salvas.
Deseja salvar? (S/N)
> N
Fim.
```

Desenvolvimento

Grupo

A atividade deve ser realizada em **grupo de no mínimo 2 pessoas e no máximo 4 pessoas**.

Um único aluno do grupo deverá publicar o trabalho no Moodle.

Código:

- A solução deve ser implementada em linguagem Java, seguindo as especificações apresentadas neste documento.
- Fica proibido o uso de estruturas de dados fornecidas pela linguagem Java (restrição inclui uso de `Stack`, `Map`, `Hashtable`, `Vector`, `LinkedList`, etc). Projetos que usarem tais estruturas serão desconsiderados (zerados).
- Inclua a identificação do grupo (nome completo e RA de cada integrante) no início de cada arquivo de código, como comentário.
- Inclua todas as referências (livros, artigos, sites, vídeos, entre outros) consultadas para solucionar a atividade como comentário no arquivo `.java` que contém a `main()`.
- Caso ChatGPT e similares seja usado, incluir como comentário no arquivo `.java` que contém a `main()` o(s) link(s) de compartilhamento das interações com o serviço.

Vídeo de apresentação

- Vídeo no Youtube com apresentação do programa, código e sua execução (duração de 5 a 7 minutos). Todos os integrantes devem participar e aparecer no vídeo.
- Inclua o link do vídeo no início do arquivo de código principal (main), como comentário.

Entrega

Compacte o código-fonte (somente arquivos `*.java`) no **formato** `zip`.

No mesmo arquivo `zip`, inclua os arquivos `*.ed1` de teste que o grupo criou para testar e validar a implementação do programa.

Atenção: O arquivo `zip` não deve conter arquivos intermediários e/ou pastas geradas pelo compilador/IDE (ex. arquivos `*.class`, etc.).

Prazo de entrega: via link do Moodle até 18/11/2025 23:59.

Critérios de avaliação

A nota da atividade é calculada de acordo com os critérios da tabela a seguir.

Item avaliado	Pontuação máxima
Implementação do nó da lista e da lista encadeada.	até 0, 5 ponto
Comando **LOAD <ARQUIVO.ED1>**.	até 0,5 ponto
Comando **LIST**.	até 0,5 ponto
Comando **RUN**, incluindo execução (interpretação) do código Assembly.	até 2,5 pontos
Comando **INS <LINHA> <INSTRUÇÃO>**.	até 0,5 ponto
Comando **DEL <LINHA>**.	até 0,5 ponto
Comando **DEL <LINHA_I> <LINHA_F>**.	até 0,5 ponto
Comando **SAVE**.	até 0,5 ponto
Comando **SAVE <ARQUIVO.ED1>**.	até 0,5 ponto
Comando **EXIT**.	até 0,5 ponto
Validação das entradas da pessoa usuária do programa e arquivos `.ed1` de teste.	até 1,0 ponto
Vídeo com a explicação sobre a solução do problema, apresentação do código e sua execução (de 5 a 7 min).	até 2,0 pontos

Tabela 1 - Critérios de avaliação.

A tabela a seguir contém critérios de avaliação que podem **reduzir** a nota final da atividade.

Item indesejável	Redução de nota
O projeto é cópia de outro projeto.	Projeto é zerado
O projeto usa estruturas de dados fornecidas pela linguagem Java.	Projeto é zerado
Há erros de compilação e/ou o programa trava durante a execução.[1]	-1,0 ponto
Não há identificação do grupo no código-fonte. Não há indicação de referências no código-fonte. Arquivos enviados em formatos incorretos. Arquivos e/ou pastas intermediárias que são criadas no processo de compilação foram enviadas junto com o código-fonte.	-1,0 ponto

Tabela 2 - Critérios de avaliação (redução de nota).

[1] Sobre erros de compilação: considere apenas erros. Não há problema se o projeto tiver *warnings* (embora *warnings* podem avisar sobre possíveis travamentos em tempo de execução, como loop infinito, divisão por zero, etc.).

O código-fonte será compilado com o compilador **javac** (21.0.2) na plataforma Windows da seguinte forma:

```
> javac *.java -encoding utf8
```

O código compilado será executado com **java** (21.0.2) na plataforma Windows da seguinte forma:

```
> java <Classe>
```

Sendo que **<Classe>** deve ser substituído pelo nome da classe que contém o método **public static void main(String[] args)**.