Tarefa de Laboratório 08 - Dicionários e Conjuntos

Interpretador de comandos

Certamente você já deve ter ouvido falar dos interpretadores de comandos. Um interpretador de comandos nada mais é do que um programa de computador responsável por ler comandos do usuário e executar ações baseadas nesses comandos. Um grande exemplo desse tipo de programa é o terminal do Python que é responsável por processar instruções da linguagem.

Sua tarefa nesse laboratório será criar um interpretador de comandos capaz de manusear variáveis e computar expressões aritméticas e expressões lógicas. As especificações do interpretador que você irá implementar estão descritas a seguir.

Especificação dos comandos

Seu interpretador de comandos deve ser capaz de executar dois tipos de comandos:

1. **Comando de atribuição**:
   - Sintaxe: `X = EXPRESSÃO`
   - Descrição: esse tipo de comando é composto pelo **nome de uma variável** (`X`), seguido de um **operador de atribuição** (`=`) e de uma **expressão a ser calculada** (**`EXPRESSÃO`**). Seu programa deve calcular o valor da `EXPRESSÃO` e atribuir o resultado à variável `X` silenciosamente, isto é, sem imprimir qualquer informação na tela. O valor armazenado em `X` poderá ser referenciado em comandos futuros.
   - Exemplos:
     - `soma = 2`
     - `total = 3 + 4 + soma - 2`
2. **Cálculo de expressão sem atribuição**:
   - Sintaxe: `EXPRESSÃO`
   - Descrição: esse tipo de comando é composto somente por uma expressão. Seu programa deverá calcular o valor da `EXPRESSÃO` e imprimir o resultado na tela.
   - Exemplos:
     - `soma`
     - `soma - 10 + 2`
     - `1 + 4 - 5`

Uma `EXPRESSÃO` pode ser de três tipos diferentes:

1. **Expressão aritmética**:
   - Sintaxe para uma expressão aritmética com *n* operandos: $X_1\ op_1\ X_2\ op_2\ \ldots\ op_{n-1}\ X_n$, onde:
     - X_i é um **operando** que pode ser uma **constante** ou uma **variável** (*veja no final dessa seção o que será considerado como constante e variável*),
     - op_i é um **operador aritmético** que pode ser `-` (subtração) ou `+` (adição).
   - Exemplos:
     - `3`
     - `soma`
     - `total - 4`
     - `3 + 29 - 4443 + soma - 2`
2. **Expressão booleana simples**:
   - Sintaxe: $Exp_1\ op\ Exp_2$, onde:
     - Exp_i é uma **expressão aritmética**,
     - op é um dos seguintes **operadores de comparação**: `==` (igual), `!=` (diferente), `<` (menor), `<=` (menor ou igual), `>` (maior), `>=` (maior ou igual).
   - Descrição: esse tipo de expressão é composto por uma **expressão aritmética** (`Exp1`) seguida de um **operador de comparação** (`op`) e de outra **expressão aritmética** (`Exp2`). Seu programa deve, **primeiro**, computar o valor das duas expressões aritméticas e, **depois**, calcular o resultado da comparação dos dois valores.
   - Resultado: o valor de uma expressão booleana sempre será **1** quando o resultado for *Verdadeiro* e **0** quando o resultado for *Falso*.
   - Exemplos:
     - `2 == 3`
     - `soma > 93255123 - 199`
     - `3 + 29 - 4443 + soma - 2 <= 9300 + 2`
3. **Expressão booleana composta**:
   - Sintaxe para uma expressão composta com *n* sub-expressões booleanas: $Exp_1\ op_1\ Exp_2\ op_2\ \ldots\ op_{n-1}\ Exp_n$, onde:
     - Exp_i é uma **expressão booleana simples**,
     - op_i é um **operador lógico** que pode ser `AND` ou `OR`.
   - Descrição: esse tipo de expressão é composto por várias **expressões booleanas simples** (`Expi`) intercaladas por **operadores lógicos** (`opi`). Seu programa deve, **primeiro**, computar o valor das expressões booleanas simples e, **depois**, calcular o resultado das comparações lógicas.
   - Resultado: o valor de uma expressão booleana composta sempre será **1** quando o resultado for *Verdadeiro* e **0** quando o resultado for *Falso*.
   - Exemplos:
     - `2 <= 3 OR 4 > 5`
     - `soma == total AND 3 < soma`
     - `3 + 29 + soma - 2 <= 9300 + 2 AND 432 >= soma - 1 OR 3 != 2`

Um `operando` de uma expressão pode ser uma constante ou uma variável:

1. **Constante**: para esse lab, uma constante será sempre um número inteiro positivo (sem sinal).
   - Exemplos:
     - `0`
     - `912384`
2. **Variável**: o nome de uma variável será uma string composta somente por letras ou dígitos, sendo que o primeiro caractere do nome da variável sempre deve ser uma letra.
   - Exemplos:
     - `minhaVariavel`
     - `var2`

Ordem de avaliação das expressões

A fim de simplificar esta tarefa, vamos considerar que todos os operadores lógicos (`AND` e `OR`) possuem a mesma ordem de precedência, ou seja, as comparações lógicas devem ser calculadas da esquerda para a direita, sem se importar com o tipo de operador que está sendo utilizado. De maneira semelhante, todos os operadores aritméticos (`+` e `-`) possuem a mesma prioridade.

Veja um exemplo abaixo de como uma expressão booleana composta pode ser calculada seguindo a ordem de precedência:

1. Vamos considerar que a seguinte expressão deve ser calculada: `1 + 2 - 3 <= 2 OR 10 + 20 >= 5 + 2 AND 1 == 3`
2. Primeiramente, temos que resolver as expressões aritméticas para poder calcular as expressões booleanas simples. Se resolvermos todas de uma vez, a expressão inicial se torna: `0 <= 2 OR 30 >= 7 AND 1 == 3`
3. Se resolvermos, agora, todas as expressões booleanas simples de uma vez, obtemos: `1 OR 1 AND 0`
4. Como os operadores `AND` e `OR` têm a mesma ordem de precedência, devemos resolver a expressão da esquerda para a direita, ou seja, primeiro resolvemos o termo `1 OR 1` que resulta em `1`. Assim a expressão se torna: `1 AND 0`
5. Por último resolvemos a última operação lógica (`1 AND 0`) que resulta em `0`.

- Note que se tivéssemos calculado primeiro o valor da operação `AND` na expressão `1 OR 1 AND 0`, o resultado final seria `1 OR 0 = 1`, o que é incorreto para essa tarefa.

Descrição da entrada

A entrada de um caso de teste é composta por diversas linhas, onde cada linha possui um comando que deve ser processado seguindo as especificações descritas. Sempre haverá um único espaço em branco entre cada operando e operador.

Veja um exemplo de entrada abaixo (Ex1):

```
1
1 + 2
soma = 1 + 2 + 3
soma + 4
verificacao = soma <= 20 AND 9 + 2 > 20 OR 1 <= 3
2 + 3
verificacao
```

Outro exemplo (Ex2):

```
10
20 - var1 - var2
var1 = 10
var2 = 20
20 - var1 - var2
10 = 30 + 40 + var3
23A = 90
```

Descrição da saída

Para cada comando do tipo `Cálculo de expressão sem atribuição`, seu programa deverá imprimir o resultado da expressão em uma linha separada.

Além disso, o interpretador deve ser capaz de detectar e imprimir dois tipos de erro:

- Caso a expressão de algum comando referencie uma variável que ainda não foi declarada através de um comando de atribuição, seu programa deverá imprimir a seguinte mensagem de erro: "*Erro de referencia: a variavel {nome_variavel} nao foi definida.*", onde `{nome_variavel}` é o nome da primeira variável da expressão que não foi definida.
- Caso o nome da **variável que está sendo inicializada** em um **comando de atribuição** não siga a sintaxe especificada, seu programa deverá imprimir a mensagem de erro "*Erro de sintaxe: {nome_variavel} nao e um nome permitido para uma variavel.*", onde `{nome_variavel}` é o nome da variável que está sendo inicializada e possui sintaxe errada.
- No momento em que o primeiro erro for detectado, a análise do comando deve ser encerrada e o erro impresso em uma linha separada.

Após seu programa processar o último comando, imprima "*Encerrando... Bye-bye.*"

Veja a saída correspondente ao primeiro exemplo de entrada (Ex1):

```
1
3
10
5
1
Encerrando... Bye-bye.
```

Saída do segundo exemplo (Ex2):

```
10
Erro de referencia: a variavel var1 nao foi definida.
-10
Erro de sintaxe: 10 nao e um nome permitido para uma variavel.
Erro de sintaxe: 23A nao e um nome permitido para uma variavel.
Encerrando... Bye-bye.
```

Dicas para esta tarefa

Como você já deve ter notado, para esse lab nós não temos uma linha da entrada especificando o número de comandos que devem ser lidos, nem sequer uma palavra-chave para sinalizar que o último comando foi dado.

Como o run.codes armazena os dados da entrada em um arquivo e redireciona a entrada padrão do seu programa para esse arquivo, uma forma de detectar que não há mais comandos para serem lidos é verificar se alcançamos o EOF (end-of-file, em português, final de arquivo) do arquivo de entrada.

Para fazer isso, podemos utilizar do fato de o comando `input()` lançar uma exceção do tipo `EOFError` sempre que está no final do arquivo e tenta ler algum dado. Observe, abaixo, como podemos implementar um código que lê uma nova linha até que o EOF seja detectado:

```
while True:
    try:
        linha = input()
        print("Entrada lida:", linha)
    except EOFError:
        print("Alcancamos o EOF.")
        break
```

Para simular um final de arquivo no terminal, podemos pressionar as teclas `Ctrl+D` no **Linux** ou as teclas `Ctrl+Z` no **Windows**. Veja um exemplo da execução do código de exemplo no terminal do Linux:

```
$ python3 codigo.py
>>> Linha 1
Entrada lida: Linha 1
>>> Linha 2
Entrada lida: Linha 2
>>> (Ctrl+D)
Alcancamos o EOF.
$
```

Obs.: o caractere "$" é do próprio terminal do Linux e os caracteres ">>>" foram colocados para sinalizar a entrada digitada pelo usuário.