UNISINOS - **UNIVERSIDADE DO VALE DO RIO DOS SINOS**
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS (C6/6) – Curso: Informática

LABORATÓRIO II

Disciplina: Linguagem de Programação PASCAL
Professor responsável: ***Fernando Santos Osório***
Semestre: 2003/1
Horário: 41 e 53

E-mail: ***osorio@exatas.unisinos.br***
Web:
http://www.inf.unisinos.br/~osorio/lab2.html
Xerox : *Pasta 54* – LAB. II (Xerox do C6/6)

TRABALHO PRÁTICO 2003/1 – GRAU B

(Versão 1.0 – 26/05/2003)

Faça um programa para implementar o ***BRASIL – "Brazilian Rules and Actions SImple Language"*** , uma nova e revolucionária linguagem baseada em Regras e Ações que em breve será lançada no mercado internacional pela ACME Brasil. O BRASIL é a linguagem ideal para se desenvolver "Sistemas Especialistas" de última geração. Siga as instruções abaixo que descrevem a proposta desta nova linguagem e o seu modo de funcionamento, e lembre-se "***o sucesso do Brasil depende de você***!".

A linguagem BRASIL possui três componentes básicos:

1. **VARIÁVEIS:** As variáveis são todas lidas de um arquivo texto denominado "vars.txt" e armazenadas em uma estrutura de dados com alocação dinâmica (lista ou árvore) que permita consultar e atualizar o valor destas variáveis. O arquivo texto irá possuir dados em pares de linhas, onde na primeira linha encontra-se o nome da variável (max. de 20 caracteres) e na segunda o seu valor atual (valor é um número real). Na versão Beta0.1 do Brasil apenas variáveis do tipo Real poderão ser usadas. Exemplo do arquivo de entrada:

```
Idade                    False
20.0                     0.0
Salario                  SexoMasc
5000.00                  1.0
True                     Departamento
1.0                      5.0
```

   As funções usualmente executadas sobre as variáveis são:
   Ler arquivo vars (nome_arq) => Lê os dados do arquivo de variáveis e cria a estrutura de dados
   Insere variável (nome, valor) => Insere na estrutura de dados a variável e seu valor atual
   Consulta variável (nome) => Retorna o valor da variável procurada ou "não encontrada"
   Altera variável (nome, valor) => Altera o valor atual da variável. Pode usar as rotinas insere e consulta... se a variável já existe, altera o valor; se não existe então insere com o valor indicado.

2. **AÇÕES:** As ações são executadas como resultado da ativação de uma regra e estão descritas em um arquivo chamado "actions.txt". As ações são descritas da seguinte forma:

```
EXEC <identificador_da_ação>
TYPE <tipo_da_ação>
DATA <dados_usados_na_ação>
NEXT <próximo_comando>
```

- EXEC "Identificador_da_ação" é composto pela letra 'A' seguida de um número que identifica esta ação, por exemplo: EXEC A1, EXEC A12, EXEC A150.
- TYPE "Tipo_da_ação" é um dos seguintes tipos: (4 letras + espaço)

  OUTM = Output message: escreve na tela a mensagem contida no campo DATA da ação.

  OUTV = Output variable: escreve na tela o valor contido em uma variável, cujo nome está indicado no campo DATA da ação. Este comando implica em consultar as variáveis, e caso ela exista, exibir seu conteúdo na tela. Caso a variável não seja encontrada, o BRASIL irá abortar a execução exibindo a seguinte mensagem "Run-Time Error OUTV: Variable not found".

  SETV = Set variable: altera o valor atual de uma variável (já existente), onde o nome e o novo valor desta variável estão indicados no campo DATA (separados por um ponto-e-vírgula). Este comando implica em consultar as variáveis, e caso ela exista, alterar seu conteúdo. Caso a variável não seja encontrada, o BRASIL irá abortar a execução exibindo a seguinte mensagem "Run-Time Error SETV: Variable not found".

  Exemplos: TYPE OUTM, TYPE OUTV, TYPE SETV.

  - DATA "Dados_usados_na_ação" é uma string se o comando for OUTM, é um nome de uma variável se o comando for OUTV, ou, é um par "variável;valor" se o comando for SETV. Os dados neste item não terão mais do que 50 caracteres de comprimento. Exemplos: DATA Mensagem, DATA Idade, DATA Salario;1200.00
- NEXT "Próximo_comando" é composto de uma identificação do comando ("identificador da ação" ou "identificador da regra") ou de um comando nulo ("NULL"). Note que todo o identificador da ação sempre começa pela letra "A" (exemplo: A1, A12, A150) e todo o identificador da regra sempre começa pela letra "R" (exemplo: R1, R3, R12). Caso não exista um comando seguinte, então o próximo comando será a palavra "NULL".
  Exemplo: NEXT A1, NEXT A12, NEXT R3, NEXT NULL.

Exemplos dos diferentes tipos de ações contidas no arquivo "action.txt":

```
EXEC A1
TYPE OUTM
DATA Hello World!
NEXT NULL
EXEC A2
TYPE OUTV
DATA Idade
NEXT A1
```

```
EXEC A3
TYPE SETV
DATA Salario;8000.0
NEXT A1
EXEC A4
TYPE OUTM
DATA Idade maior que 18?
NEXT R2
```

Note que a primeira palavra de cada linha do arquivo de ações ***possui sempre*** 4 caracteres seguidos de um espaço em branco ("EXEC", "TYPE", "DATA", "NEXT") de modo que será simples quebrar os conteúdos de cada linha, separando-os em pedaços. É importante salientar também que não existe uma ordem de organização dos dados no arquivo, sendo que eles constituem apenas uma lista de ações identificadas por um identificador único (identificador da ação). Crie uma estrutura de dados com alocação dinâmica (lista ou árvore) para armazenar as ações, preferencialmente ordenadas pelo "identificador_da_ação". Atenção: note que a execução das ações é um laço do tipo: acha ação, executa, acha próxima ação, executa, repetindo estas operações até que a ação seja nula (não tente encadear as ações pelo conteúdo do campo NEXT).

Sugestão de algumas das funções relativas as ações que poderão serão implementadas:
Ler arquivo actions (nome_arq) => Lê os dados do arquivo de ações e cria a estrutura de dados.
Insere ação (identificador, tipo, dado, proximo) => Insere na estrutura de dados a ação e seus dados complementares.

Executa ação (identificador) => Executa a ação: acha a ação e executa, sendo que caso esta ação possua uma ação seguinte, esta rotina deve retornar o identificador da próxima ação/regra a ser executada, ou caso não exista uma próxima ação/regra, deve retornar "NULL".

3. **REGRAS:** As regras são testadas (condição – [IF]) e como resultado deste teste é executado um comando se esta regra for satisfeita (THEN), ou um outro comando será executado se esta regra não for satisfeita (ELSE). As regras estão descritas em um arquivo chamado "rules.txt". As regras são descritas da seguinte forma:

```
RULE <identificador_da_regra>
[IF] <condição>
THEN <próximo_comando>
ELSE <próximo_comando>
```

- RULE "Identificador_da_regra" é composto pela letra 'R' seguida de um número que identifica esta regra, por exemplo: RULE R1, RULE R3, RULE R12.
- [IF] "Condição" é composto por três elementos separados por um espaço em branco. O primeiro elemento pode ser o nome de uma variável ou um valor. O segundo elemento é composto por um único caracter que define o operador relacional usado: "=" (igual), "<" (menor), ">" (maior) ou "#" (diferente). Por fim, o terceiro elemento também pode ser o nome de uma variável ou um valor. Sendo assim, a "condição" permite comparar um valor/variável em relação a outro valor/variável. Se a condição for verdadeira, então o "THEN" será executado, caso contrário, o "ELSE" será executado.
- THEN ou ELSE "Próximo_comando" é composto de uma identificação do comando ("identificador da ação" ou "identificador da regra") ou de um comando nulo ("NULL"). Note que todo o identificador da ação sempre começa pela letra "A" (exemplo: A1, A12, A150) e todo o identificador da regra sempre começa pela letra "R" (exemplo: R1, R3, R12). Caso não exista um comando seguinte, então o próximo comando será a palavra "NULL".
  Exemplo: THEN A1, ELSE A12, THENR3, ELSE NULL.

Exemplos dos diferentes tipos de regras contidas no arquivo "rules.txt":

```
RULE R1                          RULE R3
[IF] SexoMasc = True             [IF] Idade # 30
THEN A4                          THEN NULL
ELSE R3                          ELSE R4
RULE R2                          RULE R4
[IF] Idade > 18                  [IF] Salario < 7000.00
THEN A2                          THEN A3
ELSE NULL                        ELSE NULL
```

- Note que a primeira palavra de cada linha do arquivo de regras ***possui sempre*** 4 caracteres seguidos de um espaço em branco ("RULE", "[IF]", "THEN", "ELSE") de modo que será simples quebrar os conteúdos de cada linha, separando-os em pedaços.
- Crie uma estrutura de dados com alocação dinâmica (árvore) para armazenar as regras. É importante salientar também que EXISTE uma ordem de organização dos dados no arquivo, pois cada regra é inserida junto ao THEN ou ELSE caso o nodo possua uma ligação com esta nova regra. Caso a regra que está sendo inserida possuir um identificador da regra que ainda não existe indicado em alguma parte da árvore, então esta regra irá se tornar um novo nodo raiz. Sendo assim, poderemos terer várias árvores, com diversos nodos raiz. Para manter uma referência para estes nodos raiz, crie uma lista de ponteiros para raiz, conforme a figura apresentada a seguir. Exemplo: no arquivo "rules.txt" descrito acima, a regra R3 é um nodo filho da regra R1, e a regra R4 é um nodo filho de R3. As regras R1 e R2 são nodos raiz.

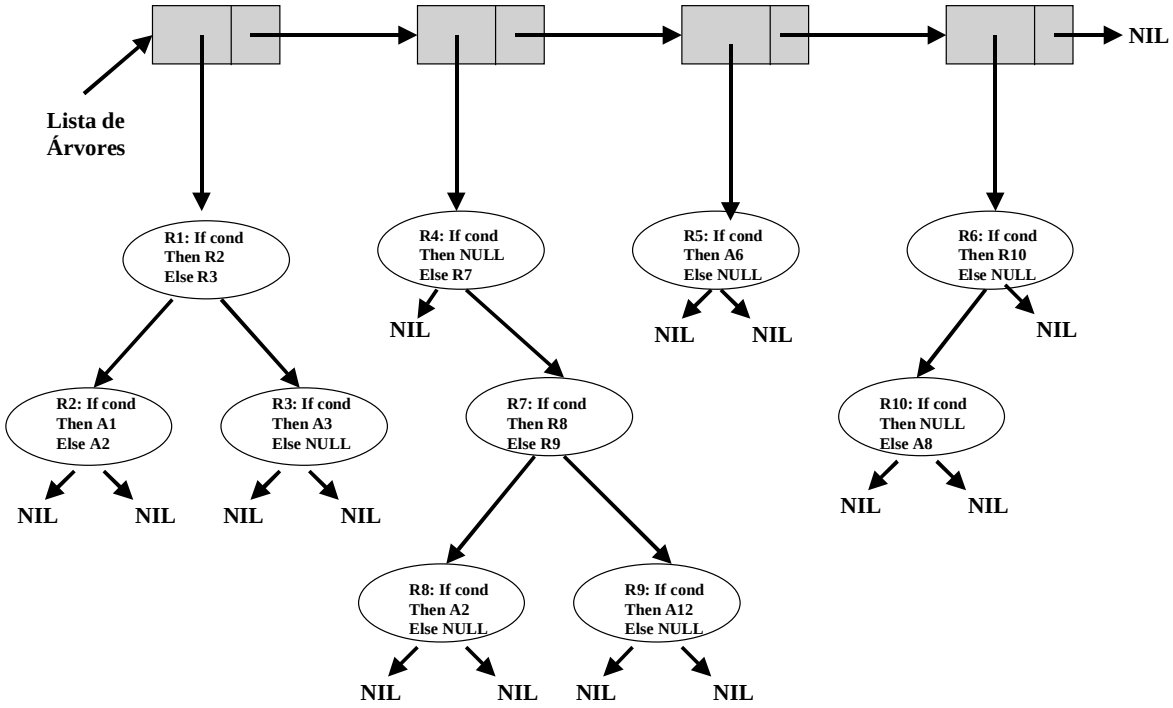

Figura – Exemplo de uma possível estrutura de dados usada para representar as regras

- Atenção: note que a execução das regras é um laço do tipo: executa regra (raiz), acha próximo comando, executa comando, repetindo estas operações até que o comando seja nulo. Este laço de execução deve ser realizado para cada um dos nodos raiz.

Sugestão de algumas das funções relativas as regras que poderão serão implementadas:
Ler arquivo rules (nome_arq) => Lê os dados do arquivo de regras e cria a estrutura de dados.
Insere regra (identificador, condição, então, senão) => Insere na estrutura de dados a regra e seus dados complementares.
Executa regra (identificador) => Executa a regra: acha a regra e executa, sendo que caso esta regra possua um comando seguinte, esta rotina deve retornar o identificador da próxima ação/regra a ser executada, ou caso não exista uma próxima ação/regra, deve retornar "NULL".
Testa condição (condição) => Quebra a condição em 3 partes, para a primeira parte, se for um número (começa por um dígito ou um sinal de negativo) então armazena seu valor em V1, se for um nome de variável (começa por uma letra) então consulta esta variável e armazena o seu valor em V1. Repete a mesma operação para a terceira parte da condição, armazenando um valor em V2. Compara V2 com V2 através do operador relacional especificado na segunda parte da condição (igual, maior, menor ou diferente) e retorna o resultado: true ou false.

*** Utilização do Programa e Interação com o Usuário:**

O programa deve inicialmente solicitar o nome do arquivo de regras (rules.txt), do arquivo de ações (actions.txt) e do arquivo de variáveis (vars.txt). Estes três arquivos devem ser lidos para a memória e o programa deve ser executado (procura quais nodos raiz são satisfeitos), gerando dois tipos de saídas: na tela devem aparecer as mensagens e valores exibidos pelos comandos EXEC do tipo OUTM e OUTV, e deve ser gravado um arquivo de saída, denominado de "trace.txt", contendo um acompanhamento dos comandos que foram executados.

Exemplo de arquivo de saída: "trace.txt"

```
R1: SexoMasc 1.0 = 1.0  Exec: A4
A4: Idade maior que 18? Exec: R2
R2: Idade 20.0 > 18.0 Exec: A2
A2: Idade 20.0 Exec A1
A1: Hello World! Exec: NULL
```

- Comandos do tipo Regra devem aparecer descritos no arquivo pela variável o conteúdo da variável, o operador relacional (=,<,>,#) e o outro variável/valor testado, seguindo-se de qual foi o comando executado.
- Comandos do tipo Ação devem aparecer descritos no arquivo:
  i) se for do tipo OUTM que escreve uma mensagem na tela, a própria mensagem deve ser gravada no arquivo;
  ii) se for do tipo OUTV que escreve o valor de uma variável na tela, o nome da variável e o seu valor devem ser gravados no arquivo;
  iii) se for do tipo SETV que atribui um valor a uma variável, o nome da variável, o seu valor antigo e o seu valor novo devem ser gravados no arquivo (nesta ordem).
  Após estes dados deve seguir-se o comando que foi executado na seqüência (Next).

*** Consistência e Verificação de Erros:**

- O programa de interpretação e execução da linguagem BRASIL deve ter um mínimo de consistência, detectando erros do tipo: variável não encontrada, ação não encontrada, regra não encontrada. Entretanto, nós assumiremos um "sistema bem comportado", ou seja, assumimos que o usuário não tem a intenção de fornecer dados inválidos ao sistema, sendo obrigação deste prever principalmente as situações de erro ligadas ao próprio sistema e erros "acidentais", que irão resultar em uma mensagem de aviso ao usuário e na interrupção da execução do programa ("Load Error" = Erro de carga do programa; "Run-Time Error" = Erro de execução do programa).

  - Lista de ***erros que tem que ser detectados***:

  - Variável não encontrada (não declarada);
  - Ação não definida (tentativa de executar uma ação com identificador não especificado);
  - Regra não definida (tentativa de executar uma regra que não foi especificada);
  - Ação mal especificada (linha no arq. que não começa por: EXEC, TYPE, DATA, NEXT);
  - Tipo de ação inválido (type com valor diferentes dos aceitos: OUTM, OUTV, SETV);
  - Regra mal especificada (linha no arq. que não começa por: RULE, [IF], THEN, ELSE);
  - Operador relacional inválido (nas condições são aceitos apenas os operadores: =, >, <, #);

*** Observações Finais:**

Para testar o seu programa, exemplos dos arquivos da linguagem BRASIL estarão disponíveis na Internet no seguinte endereço: http://www.inf.unisinos.br/~osorio/lab2/tpgb/brasil/

Caso você não tenha notado... a linguagem BRASIL é um RBS (*Rule Based System*).

BOM TRABALHO!