Compiladores/Projecto de Compiladores/Projecto 2017-2018/Manual de Referência da Linguagem GR8

*From Wiki**3*

< Compiladores | Projecto de Compiladores

AVISOS - Avaliação em Época Normal **[Expand]**

Material de Uso Obrigatório **[Expand]**

Contents

A linguagem GR8 é uma linguagem imperativa não estruturada. Um dos objectivos da linguagem é permitir a fácil leitura, pelo que é fortemente restritiva relativamente ao aspecto e à posição do código nas linhas (indentação).

Neste manual, são apresentadas características básicas da linguagem (tipos de dados, manipulação de nomes); convenções lexicais; estrutura/sintaxe; especificação das funções; semântica das instruções; semântica das expressões; e, finalmente, alguns exemplos.

Tipos de Dados

A linguagem é fracamente tipificada (são efectuadas algumas conversões implícitas). Existem 4 tipos de dados, todos compatíveis com a linguagem C, e com alinhamento em memória sempre a 32 bits:

- Tipos numéricos: os Inteiros, em complemento para dois, ocupam 4 bytes; os reais, em vírgula flutuante, ocupam 8 bytes (IEEE 754).
- As cadeias de caracteres são vectores de caracteres terminados por ASCII NUL (carácter com o valor zero). Variáveis e literais deste tipo só podem ser utilizados em atribuições, impressões, ou como argumentos/retornos de funções.
- Os ponteiros representam endereços de objectos e ocupam 4 bytes. Podem ser objecto de operações aritméticas (deslocamentos) e permitem aceder ao valor apontado.

Os tipos suportados por cada operador e a operação a realizar são indicados na definição das expressões.

Manipulação de Nomes

Os nomes (identificadores) correspondem a variáveis e funções. Nos pontos que se seguem, usa-se o termo entidade para as designar indiscriminadamente, explicitando-se quando a descrição for válida apenas para um dos casos.

Espaço de nomes e visibilidade dos identificadores

O espaço de nomes global é único, pelo que um nome utilizado para designar uma entidade num dado contexto não pode ser utilizado para designar outras (ainda que de natureza diferente).

Os identificadores são visíveis desde a declaração até ao fim do alcance: ficheiro (globais) ou função (locais). A reutilização de identificadores em contextos inferiores encobre declarações em contextos superiores: redeclarações locais podem encobrir as globais até ao fim de uma função. É possível utilizar símbolos globais nos contextos das funções, mas não é possível defini-los (ver símbolos globais).

Validade das variáveis

As entidades globais (declaradas fora de qualquer função), existem durante toda a execução do programa. As variáveis locais a uma função existem apenas durante a sua execução. Os argumentos formais são válidos enquanto a função está activa.

Convenções Lexicais

Para cada grupo de elementos lexicais (tokens), considera-se a maior sequência de caracteres constituindo um elemento válido.

Estrutura de linha (linhas lógicas)

Um programa está dividido em linhas lógicas e uma instrução não pode ocupar mais de uma linha lógica, excepto se a instrução permitir explicitamente mudanças de linha através, por exemplo, da utilização de blocos. Uma linha lógica pode ser constituída por uma ou mais linhas físicas separadas pelo marcador de continuação: **...** (três pontos seguidos).

Linhas físicas

Uma linha física pode ter qualquer comprimento. A terminaçâo de uma linha física depende da convenção utilizada pelo sistema operativo de suporte utilizado. Assim, em UNIX, o carácter utilizado é o ASCII 0x0A (linefeed). Embora sistemas diferentes possam considerar outros caracteres para marcar o final de linha, neste caso, apenas se considera o mencionado acima.

Marcador de continuação: junção explícita de linhas físicas

Duas ou mais linhas físicas podem ser juntas numa única linha lógica através da utilização do marcador de continuação: **...** (três pontos seguidos), quando ocorrem imediatamente antes da terminação da linha física (ver exemplo -- neste exemplo, a condição está dividida em várias linhas físicas).

```
if year above 2006 and not year above 2010 and not day below 1 ...
   and ((month equals 1 or month equals 3 or month equals 5 or month equals 7 ...
         or month equals 8 or month equals 10 or month equals 12) and not day above 31 ...
     or (month equals 4 or month equals 6 or month equals 9 or month equals 11) and not day above 36 ...
     or (month equals 8 or month equals 10 or month equals 12) and not day above 31 ...
     or month equals 2 and (year equals 2008 and not day above 29 or not year equals 2008 ...
          and not day above 28)) then
   post "huge day!"
else
   post "not a great dayl"
```

O marcador de continuação pode também ser utilizado em cadeias de caracteres literais, sendo ilegal em qualquer outro contexto. Em particular, não pode ser utilizado para permitir a continuação de tokens (identificadores, palavras chave, delimitadores, operadores, literais inteiros ou reais), nem comentários explicativos.

Caracteres brancos

São considerados caracteres brancos aqueles que, embora servindo para separar os elementos lexicais, não representam nenhum elemento lexical, São considerados caracteres brancos: ASCII 0x20 (espaço) e ASCII 0x09 (tabulação).

Note-se que os caracteres mencionados acima, apesar de brancos. têm significado especial se aparecerem no inicio de uma linha lógica. Analogamente, os caracteres de mudança de linha também tem significado especial (ver linhas brancas e indentação),

Linhas brancas

Uma linha lógica que contenha apenas caracteres brancos e comentários é ignorada, não sendo gerado nenhum elemento lexical nem alterada a indentação.

Indentação

Os caracteres espaço e tabulação horizontal não são considerados brancos no início de uma linha lógica. Neste caso, as suas cadeias determinam o nível de indentação do código.

Os caracteres de tabulação são substituídos (da esquerda para a direita) por 1 (um) a 8 (oito) espaços, de tal forma que o número total de caracteres brancos até ao carácter corrente (e incluindo este) seja o múltiplo de 8 seguinte (regra utilizada pelo UNIX). O número total de espaços brancos até ao primeiro carácter não branco determina o nível de indentação da expressão.

A indentação não pode continuar na linha física seguinte mediante a utilização de uma junção de linhas. Assim, no máximo, o número de espaços brancos até ao primeiro carácter de continuação determina o nível de indentação.

Embora os diversos níveis de indentação possam ser diferentes, o significado pode ser o mesmo (ver exemplo seguinte).

(a) OK: bom estilo

```
define small function max on small a, small b as
  if a above b then
    return a
  else
    return b
```

(b) OK: mau estilo

```
define small function max on small a, small b as
  if a above b then
             return a
  else
   return b
```

Errado!

```
    define small function max on small a, small b as
  if a above b then
              return a
    else
     return b
```

Comentários

Existem dois tipos de comentários, que também funcionam como elementos separadores:

- **explicativos** -- começam com **!!** e acabam no fim da linha física; e
- **operacionais** -- começam com **<<** e terminam com **>>**, podendo estar aninhados.

Se as sequências de início fizerem parte de uma cadeia de caracteres, não iniciam um comentário (ver definição das cadeias de caracteres).

Palavras chave

As seguintes palavras são reservadas, não constituindo identificadores (devem ser escritas exactamente como indicado):

- **small huge news fake initially**
- **use public define procedure function on as do uses for return**
- **plus minus times over modulus not and or assign to cell at above below equals input objects**
- **if then elsif else stop again post tweet sweeping from by**

O identificador **covfefe**, embora não reservado, corresponde à função principal.

Tipos

Os seguintes elementos lexicais designam tipos em declarações (ver gramática): **small** (inteiro), **huge** (real), **news** (cadeia de caracteres).

Os tipos correspondentes a ponteiros são designados pela palavra chave **fake**, pois designam uma indirecção e não o objecto real (ver gramática).

Operadores de expressões

São considerados operadores os elementos lexicais apresentados na definição das expressões.

Delimitadores e terminadores

Os seguintes elementos lexicais são delimitadores/terminadores: **,** (vírgula) e **(** e **)** (delimitadores de expressões).

Identificadores (nomes)

São iniciados por uma letra, seguindo-se 0 (zero) ou mais letras, dígitos, ou **-** (hífen). O comprimento do nome é ilimitado e dois nomes são distintos se houver alteração de maiúscula para minúscula, ou vice-versa, de pelo menos um carácter.

Literais

São notações para valores constantes de alguns tipos da linguagem (não confundir com constantes, i.e., identificadores que designam elementos cujo valor não pode ser alterado durante a execução do programa).

Inteiros

Um literal inteiro é um número não negativo. Números negativos são construídos pela aplicação do operador de negação unária (**minus**) a um literal positivo.

Literais inteiros decimais são constituídos por sequências de 1 (um) ou mais dígitos de **0** a **9**, em que o primeiro digito não é 0 (zero), excepto no caso do número 0 (zero). Neste caso, é composto apenas pelo dígito o (zero) (em qualquer base).

Literais inteiros em base 7 começam sempre pelo dígito 0 (zero), sendo seguidos de um ou mais dígitos de **0** a **6** (note-se que **09** é um literal invalido em base 7). Exemplo: **006**.

Se não for possível representar um literal Inteiro na máquina, devido a um overflow, deverá ser gerado um erro lexical.

Reais em vírgula flutuante

Os literais reais positivos são expressos tal como em C (base 10).

Não existem literais negativos (números negativos resultam da operação unária **minus**).

Um literal sem **.** (ponto decimal) nem parte exponencial é do tipo inteiro.

Exemplos: **3.14**, **1E3** = 1000 (número inteiro representado em virgula flutuante). **12.34e-24** = 12.34×10^{-24} (notação cientifica).

Cadeias de caracteres

As cadeias de caracteres são delimitadas por aspas (**"**) e podem conter quaisquer caracteres, excepto ASCII NUL (0x00) e ASCII LF (0x0A). Nas cadeias, os delimitadores de comentários não têm significado especial. Se for escrito um literal que contenha **~0**, então a cadela termina nessa posição. Exemplo: **ab~0xy** tem o mesmo significado que **ab**.

É possível designar caracteres por sequências especiais (iniciadas por **~**), especialmente úteis quando não existe representação gráfica directa. As sequências especiais correspondem aos caracteres ASCII HT, LF e CR (**~t**, **~n** e **~r**, respectivamente), aspa (**~"**). til (**~~**), ou a quaisquer outros especificados através de 1 a 3 dígitos em base 7, designando valores de 8 bits (e.g., **~013** ou apenas **~13** se o carácter seguinte não representar um digito em base 7). Exemplo: **xy~013z** tem o mesmo significado que **xy~13z** e que **xy~nz**.

Elementos lexicais distintos que representem duas ou mais cadeias consecutivas são representadas na linguagem como uma única cadeia que resulta da concatenação. Exemplo: **"ab"␣"cd"** é o mesmo que **"abcd"**.

Ponteiros

O único literal admissível para ponteiros corresponde ao ponteiro nulo e é indicado pela palavra reservada **null**.

Gramática

A gramática da linguagem está resumida abaixo. Considerou-se que os elementos em tipo fixo são literais, que os parênteses curvos agrupam elementos, que elementos alternativos são separados por uma barra vertical, que elementos opcionais estão entre parênteses rectos, que os elementos que se repetem zero ou mais vezes estão entre ⟨ e ⟩. Alguns elementos usados na gramática também são elementos da linguagem descrita se representados em tipo fixo (e.g., parênteses).

ficheiro	→	⟨ *declaração* ⟩
declaração	→	*variável* | *função*
variável	→	[*qualificador*] *tipo identificador* [**(** **initially** *expressão* **)**]
função	→	[*qualificador*] (*tipo* **function** | **procedure**) *identificador* [**uses** *variáveis*]
	→	**define** [*qualificador*] (*tipo* **function** | **procedure**) *identificador* [**on** *variáveis*] **as** *bloco*

variáveis	→	*variável* ⟨ **,** *variável* ⟩
tipo	→	**small** [⟨ **small** ⟩ **fake**] | **huge** [⟨ **huge** ⟩ **fake**] | [⟨ **fake** ⟩] **news**
bloco	→	⟨ *declaração* ⟩ ⟨ *instrução* ⟩
instrução	→	*expressão* | *atribuição* | **tweet** *expressão* | **post** *expressão*
	→	**again** [*literal-inteiro*] | **stop** [*literal-inteiro*] | **return** [*expressão*]
	→	*instrução-condicional* | *instrução-de-iteração* | *bloco*
atribuição	→	**assign** *expressão* **to** *lvalue*
instrução-condicional	→	**if** *expressão* **then** *bloco* ⟨ **elsif** *expressão* **then** *bloco* ⟩ [**else** *bloco*]
instrução-de-iteração	→	**sweeping** *lvalue* **from** *expressão* **to** *expressão* [**by** *expressão*] **do** *bloco*
expressões	→	*expressão* ⟨ **,** *expressão* ⟩

Tipos, identificadores. literais e definição de expressões

Algumas definições foram omitidas da gramática: tipos de dados. identificador (ver identificadores), literal (ver literais); expressão (ver expressões),

Quanto a tipos de dados, **small** designa valores inteiros, **huge** designa valores reais, **news** designa cadeias de caracteres. Os ponteiros são tipos compostos por um dos tipos básicos e a palavra **fake**.

Left-values

Os *left-values* são posições de memória que podem ser modificadas (excepto onde proibido pelo tipo de dados). Os elementos de uma expressão que podem ser utilizados como left-values encontram-se individualmente identificados na semântica das expressões.

Ficheiros

Um ficheiro é designado por principal se contiver a função principal (a que inicia o programa).

Declaração de variáveis

Uma declaração de variável indica sempre um tipo de dados e um identificador.

Exemplos:

- Inteiro: **small i**
- Real: **huge r**
- Cadeia de caracteres: **news s**
- Ponteiro para inteiro: **small take p1** (equivalente a **int*** em C)
- Ponteiro para real: **huge fake p2** (equivalente a **double*** em C)
- Ponteiro para cadeia de caracteres: **fake news p3** (equivalente a **char**** em C)
- Ponteiro para ponteiro para inteiro: **small small fake p4** (equivalente a **int**** em C)

Símbolos globais

Por omissão, os símbolos são privados a um módulo. não podendo ser importados por outros módulos.

A palavra chave **public** permite declarar um identificador como público, tornando-o acessível a partir de outros módulos.

A palavra chave **use** permite declarar num módulo variáveis definidas noutros módulos. As variáveis não podem ser inicializadas numa declaração importada. Funções definidas noutros módulos podem ser declaradas num módulo omitindo o corpo da função.

Exemplos:

- Declarar variável privada ao módulo: **huge pl (initially 22.0 over 7)**
- Declarar variável pública: **public huge pi (initially 22 over 7.0)**
- Usar definição externa: **use huge pi**

Inicialização

Quando existe, é uma expressão que segue a palavra **initially** entre parênteses: inteira, real, ponteiro. Entidades reais podem ser inicializadas por expressões inteiras (conversão implícita). A expressão de inicialização deve ser um literal se a variável for global.

As cadeias de caracteres são (possivelmente) inicializadas com uma lista não nula de valores sem separadores.

Exemplos:

- Inteiro (literal): **small i (initially 3)**
- Inteiro (expressão): **small i (initially j plus 1)**
- Real (literal): **huge r (initlally 3.2)**
- Real (expressão): **huge r (initially i minus 2.5 plus use 3 for i)**
- Cadeia de caracteres (literal): **news s (initially "olá")**
- Cadeia de caracteres (literais): **news s (initially "olá" "mãe")**
- Ponteiro (literal): **huge huge huge fake p (initially null)**
- Ponteiro (expressão): **small fake p (initially q plus 1)**

Funções

Uma função permite agrupar um conjunto de instruções num corpo, executado com base num conjunto de parâmetros (os argumentos formais). quando é invocada a partir de uma expressão.

Declaração

As funções são sempre designadas por identificadores constantes precedidos do tipo de dados devolvido pela função. Se a função não devolver um valor. usa-se a palavra reservada **procedure** para o indicar.

As funções que recebam argumentos devem indicá-los no cabeçalho. Funções sem argumentos definem um cabeçalho vazio. Não é possível aplicar os qualificadores de exportação/importação **public** ou **use** (ver símbolos globais) às declarações dos argumentos de uma função. Não é possível especificar valores por omissão para os argumentos de uma função.

A declaração de uma função sem corpo é utilizada para caracterizar um identificador exterior ou para efectuar declarações antecipadas (utilizadas para pré-declarar funções que sejam usadas antes de ser definidas. por exemplo. entre duas funções mutuamente recursivas). Caso a declaração tenha corpo, define-se uma nova função.

Invocação

A função só pode ser invocada através de um identificador que refira uma função previamente declarada ou definida.

Se existirem argumentos, na invocação da função, o identificador é precedido pela lista de expressões entre as palavras chave **use** e **for**. Esta lista é uma sequência, não vazia, de expressões separadas por vírgulas. O número e tipo de parâmetros actuais devem ser iguais ao número e tipo dos parâmetros formais da função invocada (note-se que conversões implícitas, de inteiros em reais, são possíveis). A ordem dos parâmetros actuais deverá ser a mesma dos argumentos formais da função a ser invocada. Se a função não tiver argumentos, então é invocada através da palavra chave **do**.

Exemplos:

- Função com argumentos: **use n plus 1 for factorlal** (equivalente à chamada **factorial(n+1)** em C).
- Função sem argumentos: **do argc** (equivalente à chamada **argc()** em C).

De acordo com a convenção Cdecl, a função chamadora coloca os argumentos na pilha e é responsável pela sua remoção, após o retorno da chamada. Assim, os parâmetros actuais devem ser colocados na pilha pela ordem inversa da sua declaração (i.e., são avaliados da direita para a esquerda antes da Invocação da função e o resultado passado por cópia/valor). O endereço de retorno é colocado no topo da pilha pela chamada à função.

Corpo

O corpo de uma função consiste num bloco que contém declarações (opcionais) seguidas de instruções (opcionais). Não é possível aplicar os qualificadores de exportação (**public**) ou de importação (**use**) (ver símbolos globais) dentro do corpo de uma função.

Uma instrução **return** causa a interrupção da função. O valor devolvido por uma função, através de expressão usada como argumento da instrução **return**, deve ser do tipo declarado no cabeçalho da função.

É um erro especificar um valor de retorno se a função for declarada como não retornando um valor (indicada como **procedure**).

Qualquer bloco (usado, por exemplo, numa instrução condicional ou de iteração) pode definir variáveis.

Função principal e execução de programas

Um programa inicia-se com a invocação da função **covfefe** (sem argumentos). Os argumentos com que o programa foi chamado podem ser obtidos através de funções **argc** (devolve o número de argumentos); **argv** (devolve o n-ésimo argumento como uma cadeia de caracteres, com n>0); e **envp** (devolve a n-ésima variável de ambiente como uma cadeia de caracteres, com n>0).

```
public small function argc
public news function argv uses small n
public news function envp uses small n
public small function covfefe
```

O valor de retorno da função principal é devolvido ao ambiente que invocou o programa. Este valor de retorno segue as seguintes regras (sistema operativo): 0 (zero), execução sem erros; 1 (um), argumentos inválidos (em número ou valor); 2 (dois), erro de execução. Os valores superiores a 128 indicam que o programa terminou com um sinal. Em geral, para correcto funcionamento, os programas devem devolver 0 (zero) se a execução foi bem sucedida e um valor diferente de 0 (zero) em caso de erro.

A biblioteca de run-time (RTS) contém informação sobre outras funções de suporte disponíveis. incluindo chamadas ao sistema (ver também o manual da RTS).

Instruções

Excepto quando indicado, as instruções são executadas em sequência.

Blocos

Cada bloco tem uma zona de declarações de variáveis locais (facultativa), seguida por uma zona com instruções (possivelmente vazia). Não é possível declarar ou definir funções dentro de blocos.

A visibilidade das variáveis e limitada ao bloco em que foram declaradas. As entidades declaradas podem ser directamente utilizadas em sub-blocos ou passadas como argumentos para funções chamadas dentro do bloco. Caso os identificadores usados para definir as variáveis locais já estejam a ser utilizados para definir outras entidades ao alcance do bloco, o novo identificador passa a referir uma nova entidade definida no bloco até que ele termine (a entidade previamente definida continua a existir, mas não pode ser directamente referida pelo seu nome). Esta regra é também válida relativamente a argumentos de funções (ver corpo das funções).

Instrução condicional: if-then-elsif-else

Esta instrução tem comportamento semelhante ao da instrução **if-else** em C. As partes correspondentes a **elsif** comportam-se como encadeamentos de instruções condicionais na parte **else** de um **if-else** à la C.

Instrução de iteração: sweeping-from-to-by-do

Esta instrução tem comportamento idêntico ao da instrução **for** em C. O ciclo decorre entre os valores indicados por **from** e **to** (inclusivamente), sendo o valor actual incrementado, ao fim de cada iteração, pela quantidade indicada em **by** (ou 1, caso esta parte seja omitida). Para cada valor, o bloco indicado em **do** é executado.

Instrução de terminação: stop

Indicada pela palavra reservada **stop** (quando existe, é a última instrução do seu bloco): termina o n-ésimo ciclo mais interior em que a instrução se encontrar (quando o argumento é omitido, assume-se n=1), tal como a instrução **break** em C. Este instrução só pode existir dentro de um ciclo.

Instrução de continuação: again

Indicada pela palavra reservada **again** (quando existe, é a última instrução do seu bloco): reinicia o n-ésimo ciclo mais Interior em que a instrução se encontrar (quando o argumento é omitido, assume-se n=1). tal como a instrução **continue** em C. Esta instrução só pode existir dentro de um ciclo.

Instrução de retorno: return

A instrução **return**, se existir, é a última instrução do seu bloco. Ver comportamento na descrição do corpo de uma função.

Expressões como instruções

As expressões utilizadas como instruções são avaliadas, mesmo que não produzam efeitos secundários.

Instrução de atribuição: assign-to

0 valor da expressão é guardado na posição indicada pelo *left-value*. Podem ser atribuídos valores inteiros a *left-values* reais (conversão automática). Nos outros casos, ambos os tipos têm de concordar.

Operações de Impressão

Existem duas instruções de impressão: **tweet** (impressão sem mudança de linha) e **post** (impressão com mudança de linha). Valores numéricos (inteiros ou reais) são impressos em decimal. As cadeias de caracteres são impressas na codificação nativa. Ponteiros não podem ser impressos.

Expressões

Uma expressão é uma representação algébrica de uma quantidade: todas as expressões têm um tipo e devolvem um valor.

Existem expressões primitivas e expressões que resultam da avaliação de operadores.

A tabela seguinte apresenta as precedências relativas dos operadores: é a mesma para operadores na mesma linha, sendo as linhas seguintes de menor prioridade que as anteriores. A maioria dos operadores segue a semântica da linguagem C (excepto onde explicitamente indicado). Tal como em C, os valores lógicos são 0 (zero) (valor falso), e diferente de zero (valor verdadeiro).

Todos os tipos têm de concordar exactamente. As únicas conversões possíveis são de small para huge (neste sentido, i.e., não há conversão de huge para small).

Tipo de Expressão	**Operadores**	**Associatividade**	**Operandos**	**Semântica**
primária	**( )**	não associativos	-	parênteses curvos
unária	**plus minus ? objects**	não associativos	-	identidade e simétrico, indicação de posição, reserva de memória
multiplicativa	**times over modulus**	da esquerda para a direita	inteiros, reais	C (**modulus** é apenas para inteiros)
aditiva	**plus minus**	da esquerda para a direita	inteiros, reais, ponteiros	C: se envolverem ponteiros, calculam: (i) deslocamentos, i.e., um dos operandos deve ser do tipo ponteiro e o outro do tipo inteiro; (ii) diferenças de ponteiros, i.e., apenas quando se aplica o operador **-** a dois ponteiros do mesmo tipo (o resultado é o número de objectos do tipo apontado entre eles). Se a memória não for contígua, o resultado é indefinido.

comparativa	**below** **above**	da esquerda para a direita	inteiros, reais	C
igualdade	**equals**	da esquerda para a direita	inteiros, reais, ponteiros	C
"não" lógico	**not**	não associativo	inteiros	C
"e" lógico	**and**	da esquerda para a direita	inteiros	C: o 2º argumento só é avaliado se o 1º não for falso.
"ou" lógico	**or**	da esquerda para a direita	inteiros	C: o 2º argumento só é avaliado se o 1º não for verdadeiro.

Expressões primitivas

As expressões literais e a invocação de funções foram definidas acima.

Identificadores

Um identificador é uma expressão se tiver sido declarado. Um identificador pode denotar uma variável ou uma constante.

Um identificador é o caso mais simples de um *left-value*, ou seja, uma entidade que pode ser utilizada como zona de escrita numa atribuição.

Leitura

A operação de leitura de um valor inteiro ou real pode ser efectuado pela expressão **input**, que devolve o valor lido, de acordo com o tipo esperado (inteiro ou real). Caso se use como argumento dos operadores de impressão (**tweet** ou **post**), deve ser lido um inteiro.

Exemplos: **assign input to a** (leitura para **a**), **use input for f** (leitura para argumento de função), **post input** (leitura e impressão).

Parênteses curvos

Uma expressão entre parênteses curvos tem o valor da expressão sem os parênteses e permite alterar a prioridade dos operadores. Uma expressão entre parênteses não pode ser utilizada como *left-value* (ver também a expressão de indexação).

Expressões resultantes de avaliação de operadores

Indexação

A indexação devolve o valor de uma posição de memória indicada por um ponteiro. Consiste na indicação de uma célula de memória indexável a partir de uma base de indexação (ponteiro). O resultado de uma indexação é um left-value.

A indexação tem precedência superior aos operadores aritméticos.

Exemplo (acesso à posição **a+1** na região de memória indicada por **p**): **cell a plus 1 at p** (equivalente a **p[a+1]** em C).

Identidade e simétrico

Os operadores identidade (**plus**) e simétrico (**minus**) aplicam-se a inteiros e reais. Têm o mesmo significado que os operadores análogos em C (**+** e **-** unários).

Reserva de memória

A expressão reserva de memória, indicada pelo sufixo **objects**, devolve o ponteiro que aponta para a zona de memória, na pilha da função actual, contendo espaço suficiente para o número de objectos indicados pelo seu argumento inteiro.

Exemplo (reserva vector com 5 reais. apontados por **p**): **huge fake p (initially 5 objects)**

Expressão de indicação de poslção

O operador sufixo **?** aplica-se a *left-values*, retornando o endereço correspondente.

Exemplo (indica o endereço de **a**): **a?**

Exemplos e Testes

Os exemplos abaixo não são exaustivos e não ilustram todos os aspectos da linguagem.

Estão ainda disponíveis outros pacotes de testes.

O seguinte exemplo ilustra um programa com dois módulos: um que define a função **factorlal** e outro que define a função **covfefe** (função principal).

Definição da função **factorial** no flcheiro **factorial.gr8**:

```
define public small function factorial on small n as
  if n above 1 then
    return n times use n minus 1 for factorial
  else
    return 1
```

Exemplo da utilização da função **factorial** no ficheiro **main.gr8**:

```
!! external builtin functions
public small function argc
public news function argv uses small number
public small function atoi uses news piece

!! external user functions
public small function factorial uses small number

!! the main function
define public small function covfefe as
  small value (initially 1)
  post "Teste para a função factorial"
  if do argc equals 2 then
    news flash (initially use 1 for argv)
    assign use flash for atoi to value
  tweet value
  tweet "! is "
  post use value for factorial
  return 0
```

Como compilar:

```
gr8 --target asm factorial.gr8
gr8 --target asm main.gr8
yasm -felf32 factorial.asm
yasm -felf32 main.asm
ld -melf_i386 -o main factorial.o main.o -lrts
```

Omissões e Erros

Casos omissos e erros serão corrigidos em futuras versões do manual de referência.

Categories: **Projecto de Compiladores** **Compiladores** **Ensino**