Descrição das funções do ambiente de desenvolvimento de algoritmos

Ricardo Vargas Dorneles, Delcino Picinin Jr

9 de setembro de 2009

1 Descrição geral do ambiente

O ambiente de algoritmo é composto de duas partes, um editor/executor de algoritmos, e um banco de problemas propostos, com recursos para submissão de algoritmos e teste automático da correção dos mesmos. O editor/executor pode ser utilizado independentemente do banco de problemas, sem a necessidade de o usuário logar-se no ambiente. A figura 1 mostra a tela de entrada do ambiente.

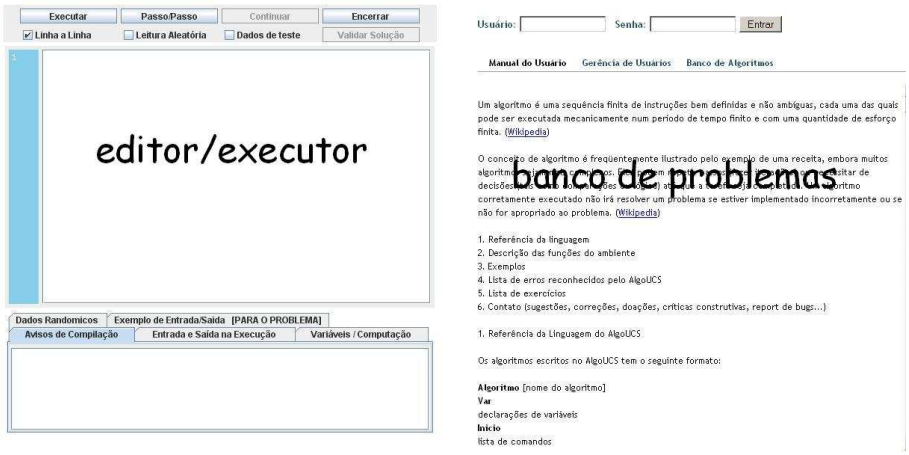

Figura 1: Tela de Entrada do Ambiente

1.1 Edição do algoritmo

O AlgoUCS permite a edição de algoritmos utilizando a linguagem descrita na seção 2, bem como a execução dos algoritmos descritos. O editor é um editor orientado a sintaxe, que atribui cores diferentes aos diferentes elementos do algoritmo, facilitando a visualização e compreensão do mesmo. Por ser uma aplet, ou seja, um programa rodado a partir de um servidor, o ambiente de edição ainda não oferece recursos para salvamento de arquivos podendo-se, entretanto, efetuar copy/paste do algoritmo digitado, utilizando-se o ambiente juntamente com um editor como o Bloco de Notas para salvar os algoritmos desenvolvidos.

1.2 Execução do algoritmo

Após editar o algoritmo, o usuário pode executar o algoritmo. Antes de iniciar a execução é feita uma análise léxica e sintática do algoritmo e se ele contiver algum erro de sintaxe, uma mensagem de erro será exibida na aba Avisos de Compilação, juntamente com o número da linha onde ocorreu o erro. As mensagens de erro são bastante claras, mas uma lista das mensagens de erro e uma explicação de cada uma pode ser encontrada na seção 4 desse documento. Se o algoritmo não tiver erros de sintaxe será disparada a execução do mesmo. Essa execução pode ser feita de três formas (figura 2):

1.2.1 Execução direta

É disparada pelo botão Executar. O algoritmo executa todos os comandos até chegar no fimalgoritmo quando termina a execução do mesmo. Ao executar um comando de leitura, aparecerá uma caixinha solicitando o dado a ser lido, especificando o tipo de dado esperado e a variável que irá receber o dado. A ativação da opção Leitura Aleatória (acima, à direita), fará com que sejam gerados dados aleatórios a cada leitura, ao invés de solicitar o dado do usuário. O resultado da execução dos comandos de escrita é mostrado na aba Saída da Execução.

1.2.2 Ativação de Pontos de Parada (breakpoints)

Ao clicar duas vezes sobre o número de uma linha, à esquerda, a linha fica vermelha. Isso significa que foi definido um ponto de parada (breakpoint)

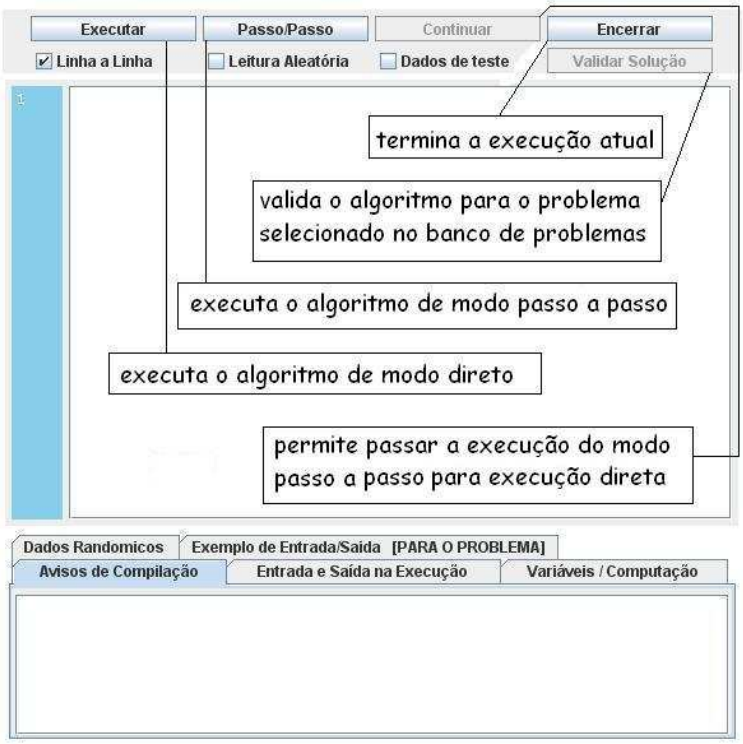

Figura 2: Modos de Execução

nessa linha, e que ao ser executada a linha a execução do algoritmo é interrompida naquela linha. Pode-se retomar a execução do algoritmo passo a passo a partir do botão Passo-Passo ou a execução contínua do algoritmo a partir do botão Continuar.

1.2.3 Execução Passo a passo

É disparada pelo botão Passo/Passo. A cada vez que o botão é ativado, é executada uma linha do algoritmo. Durante a execução do algoritmo pode-se examinar o conteúdo das variávis na aba Variáveis/Computação.

1.2.4 Execução temporizada

É disparada pelo botão Executar com a opção Linha a Linha (canto superior direito) habilitada. Nessa opção, cada linha é executada a um intervalo de meio segundo, e a linha sendo executada a cada instante aparece em uma cor diferenciada. Durante a execução pode-se também examinar o conteúdo das variáveis na aba Variáveis/Computação.

1.3 Entrada de Dados

Ao executar um comando de leitura o algoritmo espera que o usuário digite um valor e esse valor é armazenado na variável definida no comando. Durante a depuração de um algoritmo, o processo de entrada de dados pode ser tedioso, se a quantidade de dados esperados for grande. O AlgoUCS oferece duas alternativas para o usuário não ter que digitar os dados de teste a cada execução, a geração de dados de teste em uma faixa de valores definida pelo usuário, e a definição de um conjunto de dados a serem utilizados durante a execução. A figura 3 mostra as caixinhas de ativação das duas opções.

1.3.1 Geração de dados aleatórios

Nessa opção, quando forem solicitados dados reais ou inteiros, eles serão gerados dentro da faixa definida na aba **Dados Aleatórios**, abaixo da tela de edição. Se os dados lidos forem do tipo literal, os dados serão escolhidos dentre uma lista de 200 literais pré-definidos.

1.3.2 Uso de dados de teste pré-definidos

Essa opção é útil quando se quer fazer diversas execuções com os mesmos dados de teste. Nessa opção, edita-se na aba **Exemplo de Entrada/Saída** quando forem solicitados dados reais ou inteiros, eles serão gerados dentro da faixa definida na aba Dados Aleatórios, abaixo da tela de edição. Se os dados lidos forem do tipo literal, os dados serão escolhidos dentre uma lista de 200 literais pré-definidos. A figura 4 mostra um exemplo de definição de dados de teste para um problema.

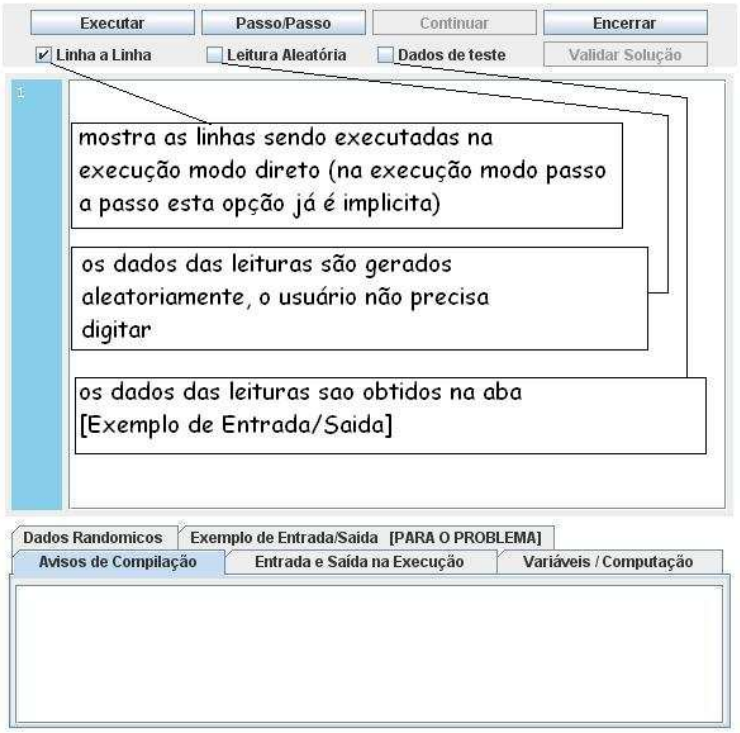

Figura 3: Modos de Entrada de Dados

1.4 Aba de visualização de Variáveis/Computação

Durante e após a execução do algoritmo é possível visualizar todo o histórico das variáveis durante a execução dos últimos 1000 comandos. Cada coluna da tabela representa uma variável, e cada linha representa o conteúdo da variável após a execução do comando descrito na primeira coluna. Na execução de cada comando o conteúdo das variáveis que não foram alteradas pelo comando aparece entre colchetes, permitindo identificar qual variável teve seu conteúdo alterado pelo mesmo. No cabeçalho da tabela aparece o nome da variável e o tipo da mesma (I - Inteiro, R - Real ou Numérico, B - Lógico (do inglês, Boolean) e L - Literal). O conteúdo de variáveis a quem ainda não foi atribuído um valor é representado por um hífem (-).

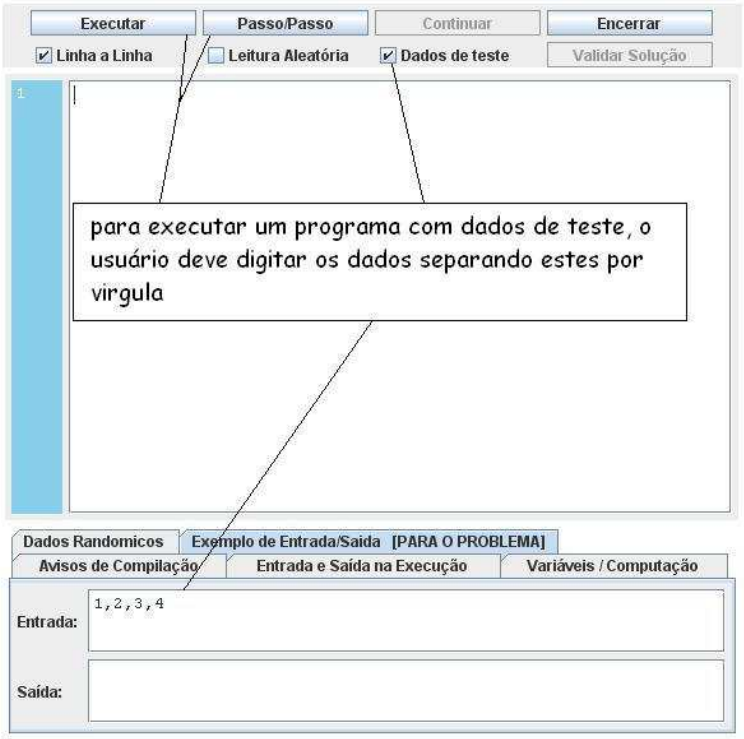

Figura 4: Definição de Dados de Teste

1.5 Erros durante a execução

Durante a execução podem ocorrer erros que, ao serem detectados, causam a interrupção do programa e a emissão de uma mensagem de erro, juntamente com o número da linha onde ocorreu o erro, na aba Avisos de Compilação. Os erros que podem ocorrer em tempo de execução são:

- Divisão por zero
- Extração de raiz negativa
- Índice inválido em vetor
- Variável NNN não inicializada - O conteúdo da variável NNN está sendo utilizado antes de ela receber qualquer conteúdo
- Chegou ao fim da função sem executar Retorna
- Foram efetuadas mais leituras do que o esperado
- Dado de teste não é inteiro

- Maior Randômico Inteiro não é inteiro - Maior valor inteiro, na aba de configuração de números randômicos, não está definido como inteiro
- Menor Randômico Inteiro não é inteiro
- Faixa de valores randômicos inteiros inválida
- Valor digitado não é inteiro

1.6 Uso do ambiente

Para utilizar o banco de problemas e o recurso de verificação automática da solução o usuário deve estar logado no ambiente. Para isso ele deve ser cadastrado no mesmo. Esse cadastramento é feito na aba **Gerência de Usuários**.

Uma vez cadastrado, o usuário tem acesso a um extensivo banco de problemas englobando todo o conteúdo do ensino de algoritmos, para os quais o usuário pode submeter soluções que serão automaticamente testadas pelo ambiente. Esses problemas são disponibilizados em grau crescente de dificuldade, e são divididos em 5 grupos, de acordo com as estruturas necessárias para resolvê-los: problemas que podem ser resolvidos sem a necessidade de comandos condicionais ou de repetição, problemas para os quais é necessário o uso do comando condicional, problemas para os quais é necessário o uso da estrutura de repetição, problemas para cuja solução é necessário o uso de vetores, problemas para cuja solução é necessário o uso de matrizes, problemas para os quais podem ser utilizadas funções e procedimentos, e problemas para cuja solução deve ser utilizada recursividade.

A página principal do portal de algoritmos possui três abas: **Manual do Usuário**, **Gerência de Usuários** e **Banco de Algoritmos**. A aba **Manual do Usuário** contem a descrição da linguagem utilizada no ambiente, com exemplos. A aba **Gerência de Usuários** é utilizada para o cadastramento de usuário e a aba **Banco de Algoritmos** apresenta os problemas cadastrados no ambiente e o gerenciamento das soluções dos usuários a esses problemas.

1.7 Banco de Algoritmos

Para utilizar o banco de problemas o usuário deve estar logado no portal. Ao entrar no banco de problemas o usuário seleciona uma classe de problemas e aparece a lista dos problemas cadastrados na base de dados (figura 5). A letra no início do código do problema indica a categoria do problema (S -

Sequenciais, C - Condicionais, I - Iterativos, V - Vetores, M - Matrizes, F - Funções, R - Recursividade, G - Geral). Para resolver um problema da base de dados o usuário seleciona o problema na lista de problemas. Ao selecionar a aba **problemas**, aparece o enunciado do problema na janela e um conjunto de dados de entrada para teste do problema e a saída correspondente.

Figura 5: Tipos de Problemas

As soluções de cada usuário são armazenadas na base de dados do portal. Para fazer um novo algoritmo, ou alterar um já existente, o usuário deve selecionar a aba **Minhas Soluções**. Ao fazer isso, as soluções que o usuário já fez para aquele problema são listadas. Se o usuário quiser trabalhar em um algoritmo já iniciado, deve selecionar o algoritmo desejado, e a informação se o algoritmo, já foi validado ou não, aparece na janela. Caso o algoritmo já tenha sido validado, também é mostrado o número de operações executadas. Após confirmação do usuário o código do algoritmo selecionado é copiado para a janela de edição e execução e o usuário pode editá-lo.

1.7.1 Contagem do número de operações

Ao executar um algoritmo as operações executadas são contadas e o número de operações efetuadas é apresentado ao final. As operações consideradas nessa contagem são atribuiçoes, leitura de uma variável, escrita de uma variável, e avaliação de um operador de qualquer tipo (aritmético, lógico, relacional ou literal).

1.7.2 Validação do Algoritmo

Após editado, o algoritmo pode ser executado utilizando os recursos descritos na seção 1.2, como execução passo a passo, execução temporizada, entrada de dados pelo usuário ou geração automática de dados de entrada, pontos de parada. Quando o usuário considerar que o algoritmo está correto, ele pode validá-lo com os dados de entrada cadastrados para aquele problema. Para cada problema proposto no ambiente, estão cadastrados um ou mais conjuntos de dados, dos quais apenas o primeiro conjunto é mostrado ao usuário, a título de exemplo. Para verificar se o algoritmo é executado corretamente para todos os conjuntos de dados cadastrados, o usuário deve entrar na aba 'Minhas soluções' e associar o algoritmo que será validado (que está sendo editado, ou será editado) ao problema correspondente. Isso é feito através do botão 'Criar uma nova solução para o problema' (passo 1 na figura 6). Ao ser feito isso, é criada uma nova solução e o número dela aparece na lista de soluções do usuário para o problema selecionado (passo 2 na figura). O usuário deve então selecionar qual a solução da lista que será editada e validada. Ao selecionar a solução, será questionado se deseja carregar a solução selecionada para a janela de edição (mensagem 'Deseja copiar a solução...'). Se a solução já estiver no editor, deve responder 'Não'. Se ele deseja carregar a solução selecionada, deve responder 'Sim'. Após terminar a edição do algoritmo selecionado, o usuário deve ativar o botão *Validar Solução* (passo 4 da figura), acima da janela de edição. Isso faz com que o algoritmo seja executado para cada conjunto de dados de entrada e a saída de cada execução seja comparada com a saída esperada para aquela entrada. Se o algoritmo executar corretamente para todos os conjuntos de dados será exibida a mensagem **Executou corretamente para todas as instâncias** e será exibido o número total de operações para a execução de todas as instâncias, conforme subseção anterior (passo 4 da figura).

Se o algoritmo não executou corretamente para algum dos conjuntos de

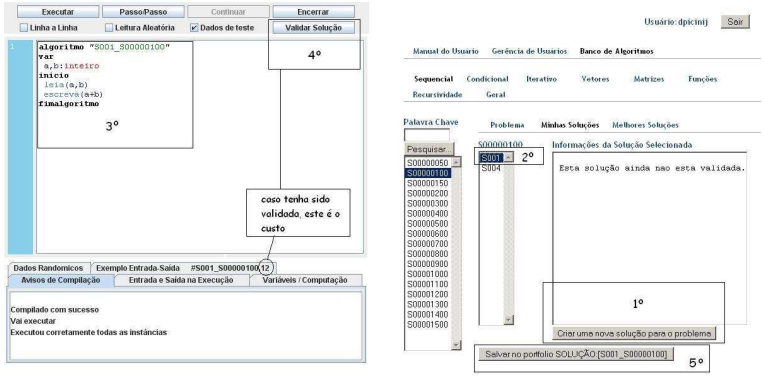

Figura 6: Passos para a validação do algoritmo

dados de entrada, aparecerá uma mensagem de erro. A lista completa de erros de execução é apresentada na seção 1.5.

Se o algoritmo executou corretamente para alguns conjuntos de dados, mas falha ao ser validado, isso pode significar que entre os conjuntos de dados cadastrados há conjuntos que recaem em alguma situação não prevista pelo algoritmo.

1.8 Botão Salvar no portfolio solução YYY

A qualquer momento da edição, ou após a execução, o algoritmo sendo editado pode ser salvo no conjunto de algoritmos do usuário. Assim, não é necessário que o algoritmo esteja correto e validado para ser salvo. Os algoritmos salvos podem ser carregados e editados conforme descrito na seção 1.1.

1.9 Aba Melhores soluções

Para cada problema armazenado é guardado o usuário que submeteu a solução mais eficiente, ou seja, o algoritmo que executou todos os conjuntos de dados de teste com o menor número de operações.

2 Referência da Linguagem do AlgoUCS

Os algoritmos escritos no AlgoUCS tem o seguinte formato:

```
Algoritmo "nome do algoritmo"
Var
declarações de variáveis
Inicio
lista de comandos
fimalgoritmo
```

onde o nome do algoritmo é opcional, mas quando colocado deve estar entre aspas (ex: Algoritmo "teste") A seção de declarações de variáveis inicia obrigatoriamente por Var e pode conter diversas declarações de variáveis. Uma descrição completa dos tipos de variáveis e do formato das declarações pode ser encontrado na seção 2.1. Os comandos podem ser quaisquer comandos dentre os comandos descritos na seção 2.3. O AlgoUCS não diferencia entre letras maiúsculas e minúsculas. Assim, ele não diferencia entre as palavras Var, var ou VAR. O AlgoUCS não é muito restrito em relação a quebras de linha, podendo reconhecer mais de um comando por linha. Entretanto, para melhor clareza dos algoritmos desenvolvidos, recomenda-se colocar um comando por linha. Da mesma forma, o AlgoUCS não leva em consideração a identação, ou os espaços em branco no início da linha, mas recomenda-se usá-los de forma a deixar bem claro a estrutura do programa ou, em outras palavras, quais comandos e estruturas estão "dentro"de outros comandos. O AlgoUCS não reconhece caracteres acentuados e o caracter cedilha. Assim, Inicio é grafado sem acento, e faca é grafado sem cedilha.

Comentários Ao encontrar os caracteres // , o AlgoUCS ignora todo o resto da linha, permitindo o uso de comentário ao algoritmo.

Ex:

```
Algoritmo [nome do algoritmo]
Var
   a, b, soma : inteiro
Inicio
   escreva("Entre com um valor:")  // aqui é lido um valor
                                   //(isto é um comentário)
```

```
   leia(a)
   escreva("Entre com outro valor:")
   leia(b)
   soma<-a+b
   escreva("A soma é ",soma)
fimalgoritmo
```

2.1 Tipos de variáveis

O AlgoUCS permite o uso de variáveis numéricas, lógicas e literais. O AlgoUCS reconhece três tipos de variáveis numéricas:

- inteiro - permite valores inteiros, positivos ou negativos, na faixa de -32555 a 32555
- real - valores reais, na faixa de -4.000.000 a 4.000.000, com uma precisão de 7 dígitos
- numerico - é o mesmo tipo de variável do real, reconhecido no AlgoUCS apenas para compatibilidade com o conteúdo visto em aula

Os tipos não numéricos reconhecidos no AlgoUCS são dois:

- logico - armazena um valor lógico, verdadeiro ou falso
- literal - armazena um valor literal, ou seja, uma sequência de caracteres

Além dos tipos simples de variáveis, o AlgoUCS suporta o uso de vetores e matrizes de duas dimensões. A sintaxe de declaração de matrizes é a seguinte:

vetor[*indice_inicial*..*indice_final*] **de** *tipo*

e para matrizes de duas dimensões:

vetor[*indice_inicial1*..*indice_final1*,*indice_inicial2*..*indice_final2*] **de** *tipo*

onde o tipo pode ser inteiro, real, numérico, lógico ou literal

```
ex:   v : vetor [1..10] de inteiro
      mat : vetor [1..10,1.10] de real
```

O bloco de declarações de variáveis é formado por uma ou mais declarações, sendo que cada declaração pode conter uma ou mais variáveis do mesmo tipo. Ex:

```
Var
   a , b : inteiro
   c : real
   v , w : vetor [1..10] de inteiro
   teste : logico
   nome : literal
```

2.2 Constantes

O AlgoUCS permite o uso de constantes numéricas, lógicas e literais. Constantes numéricas são quaisquer valores inteiros ou reais, positivos ou negativos. Nos valores reais, o ponto decimal é utilizado como separador da parte inteira e da parte fracionária (ex. 3.1415). Constantes lógicas são as constantes verdadeiro e falso. Constante literal é qualquer sequência de caracteres delimitados por aspas. Ex: "Joao da Silva"

2.3 Comandos

2.3.1 Comando de atribuição

Tem o formato

```
variável <- expressão
```

onde variável pode ser uma variável simples, numérica, lógica ou literal, ou um elemento de um vetor ou um elemento de uma matriz. Caso seja um elemento de um vetor ou uma matriz, deve ser especificada a posição do elemento que receberá o resultado da expressão.

A expressão atribuída pode ser uma constante, variável ou expressão resultante da aplicação de operadores. Expressões serão vistas em mais detalhes na seção 2.3.2.

```
ex:     a <- 3
        b <- a + b
        teste <- verdadeiro
        a[3] <- 4 // o elemento da posição 3 do vetor a
                  // recebe o valor 4
```

2.3.2 Expressões

Podem ser aritméticas, lógicas ou literais. Consistem da aplicação de operadores sobre operandos resultando em valores. Os operadores podem ser:

Aritméticos : são aplicados a operandos numéricos e resultam em valores numéricos. Os operadores aritméticos suportados pelo AlgoUCS são mostrados na tabela 1.

Operação	Op.	Prior.	Assoc.	Tipo de valor retornado	Obs
Troca de Sinal	-	1	$\Leftarrow$	o mesmo do operando	o único operador de apenas um operando
Potenciação	^	2	$\Leftarrow$	se algum operando é real, retorna real, caso contrário retorna inteiro	
Multiplicação	$*$	3	$\Rightarrow$	idem à potenciação	
Divisão	/	3	$\Rightarrow$	real	
Quociente da divisão inteira	\	3	$\Rightarrow$	inteiro	só é aplicável a operandos inteiros
Resto da divisão inteira	%	3	$\Rightarrow$	inteiro	só é aplicável a operandos inteiros
Soma	$+$	4	$\Rightarrow$	idem à potenciação	
Subtração	-	4	$\Rightarrow$	idem à potenciação	

Tabela 1: Operadores numéricos

Observações:

1. Operadores da mesma prioridade são avaliados pela sua associatividade, normalmente da esquerda para a direita, com exceção da potenciação e troca de sinal, que são avaliadas da direita para a esquerda.

2. Parênteses podem ser utilizados para alterar a ordem de avaliação dos operadores.

2.3.3 Expressões literais

O único operador que pode ser aplicado a variáveis e constantes literais é o operador de concatenação de literais, representado pelo operador +.

```
Ex:   Var nome:literal
        ...
        nome<-"João "+"da "+"Silva"
```

2.3.4 Expressões relacionais

Verificam a relação entre os dois operandos (igual, diferente, maior...), resultando em um valor lógico. Os operadores relacionais suportados pelo AlgoUCS são mostrados na tabela 2.

Obs: Nas comparações envolvendo literais, é considerada a ordem lexicográfica, ou seja, qual dos operandos vem antes no dicionário.

Operador	Símbolo	operandos suportados
Igual	=	todos
Diferente	<>	todos
Maior	>	numéricos e literais
Menor	<	numéricos e literais
Maior ou Igual	>=	numéricos e literais
Menor ou Igual	<=	numéricos e literais

Tabela 2: Operadores relacionais

Operador	Símbolo	Resultado
E lógico	E	resulta verdadeiro se ambos os operandos são verdadeiros, caso contrário, resulta falso
OU lógico	OU	resulta verdadeiro se pelo menos um dos operandos é verdadeiro, caso contrário, resulta falso

Tabela 3: Operadores lógicos

2.3.5 Expressões lógicas

Combinam expressões lógicas retornando um valor lógico. Os operadores lógicos suportados pelo AlgoUCS são mostrados na tabela 3.

Obs: O operador E tem uma prioridade maior do que o operador OU, ou seja, em uma expressão envolvendo ambos operadores, os operadores E são avaliados antes dos operadores OU

Precedência entre os operadores Quando uma expressão possui operadores aritméticos, relacionais e lógicos, em primeiro lugar são avaliados os operadores aritméticos, em seguida os relacionais e por fim os lógicos.

2.3.6 Funções pré-definidas

Além de variáveis, constantes e operadores, expressões podem conter chamadas a funções pré-definidas. Uma função recebe um ou mais valores (chamados "parâmetros"ou "argumentos"), efetua cálculos sobre esses valores e retorna um valor como resultado desse processamento. Exemplos

Nome	Retorno
Abs(valor)	Retorna o módulo (valor absoluto) do valor recebido. O tipo de valor retornado é o mesmo do valor recebido (inteiro ou real)
Raizq(valor)	Retorna a raiz quadrada do valor recebido. O tipo de valor retornado é real
Sen(x)	Retorna o seno do ângulo recebido em radianos. Tanto o valor recebido quando o valor retornado são reais
Cos(x)	Retorna o cosseno do ângulo recebido em radianos. Tanto o valor recebido quando o valor retornado são reais
Int(x)	Retorna o valor de x (real) truncado para um inteiro

Tabela 4: Funções pré-definidas

conhecidos de funções da matemática são seno, cosseno e raiz. A tabela 4 mostra as funções pré-definidas suportadas pelo AlgoUCS:

O valor passado para a função pode ser uma variável, constante ou uma expressão envolvendo operadores. A chamada de uma função pode ser colocada em qualquer lugar de uma expressão onde iria uma variável ou uma constante. Ex:

```
var1 <- sen(30)

maior <- (a+b+abs(a-b))/2

val <- abs(-3)   // a variável val receberá o
                 // módulo de (-3), ou seja, 3
```

2.4 Comando de Leitura

É utilizado para o algoritmo receber dados do teclado e armazenar em variáveis. Ao executar uma leitura o algoritmo fica esperando o usuário digitar um valor, armazena esse valor na variável especificada e continua a execução. Seu formato é:

leia (lista de variáveis)

Obs: É possível ler vários valores em um mesmo comando de leitura. Nesse caso as variáveis que irão receber os valores são separadas por vírgulas.

Ex: leia (a , b , c)

2.5 Comando de Escrita

É utilizado para o algoritmo enviar mensagens para a tela do computador. O formato do comando é:

escreva (lista de expressões)

ou

escreval (lista de expressões)

As expressões que compõem a lista são separadas por vírgulas e podem ser expressões aritméticas, lógicas ou literais. O comando escreva, após escrever as expressões de sua lista, mantém o cursos na mesma linha, de modo que as escritas seguintes serão efetuadas na mesma linha. O comando escreval, ao terminar sua execução, posiciona o cursor na linha seguinte, de forma que a próxima execução de um escrita ocorrerá na linha seguinte.

Ex: escreva ("A soma de ", a , "e ", b, "vale ", a+b)

resultará em *A soma de 3 e 4 vale 7* se as variáveis a e b valerem, respectivamente, 3 e 4

2.6 Comando Condicional (comando Se)

É utilizado quando um ou mais comandos devem ser executados somente se alguma condição é verdadeira. Seu formato é:

```
se condição
   entao
       lista de comandos
fimse
```

onde condição é qualquer expressão lógica, ou seja, expressão que resulte em verdadeiro ou falso. Ao executar o comando se, a condição é avaliada. Se ela é verdadeira, os comandos da lista são executados e a execução do algoritmo prossegue após o fimse. Se a condição é falsa, os comandos da lista não são executados e a execução prossegue após o fimse. Como em qualquer outro comando que permita uma lista de comandos dentro dele, os comandos da lista podem ser quaisquer comandos, incluindo outro comando Se, atribuições, leituras, escritas ou comandos de repetição a serem vistos na próxima seção.

ex:

```
se a > b
   entao escreva ("a é maior")
fimse
```

Uma segunda forma do comando se é:

```
 se condição
    entao
        lista 1
    senao
        lista 2
 fimse
```

Nessa forma do comando, a condição é avaliada e se ela for verdadeira os comandos da primeira lista são executados. Se ela for falsa, os comandos da segunda lista são executados. Em ambos os casos, após a execução da lista de comandos, execução prossegue no comando após o fimse.

```
se a > b
   entao escreva ("a é maior")
   senao escreva("b é maior")
fimse
```

2.7 Comandos de Repetição

O AlgoUCS reconhece três comandos de repetição, o comando Repita, o comando Enquanto e o comando Para.

2.7.1 Comando Repita

É utilizado para repetir um ou mais comandos até que uma condição seja verdadeira. Seu formato é:

```
Repita
    lista de comandos
ate condição
```

Ao entrar no Repita, a lista de comandos é executada e, após, a condição é avaliada. Se a condição é false, os comandos da lista são novamente executados e a condição é novamente avaliada. Isso se repete até que a condição seja verdadeira.

ex:

```
i<-1
Repita
   escreval(i)
   i<-i+1
ate i>5
```

2.7.2 Comando Enquanto

É utilizado para repetir um ou mais comandos enquanto uma condição for verdadeira. Seu formato é:

```
Enquanto condição faca
   lista de comandos
fimenquanto
```

Ao entrar no Enquanto, a condição é avaliada. Se ela for verdadeira a lista de comandos é executada. Isso se repete até que a condição seja falsa, quando o comando Enquanto é encerrado e a execução continua no comando imediatamente após o fimenquanto.

A diferença principal entre os comandos Repita e Enquanto é que no Repita a condição é avaliada somente após a primeira execução dos comandos, de forma que a lista de comandos sempre é executada ao menos uma vez. No comando Enquanto, a condição é avaliada antes de executar a lista de comandos e se a condição for falsa a lista de comandos não é executada nenhuma vez. Outra diferença importante é que a condição no Repita é uma condição de saída, ou seja, quando a condição é verdadeira, termina a execução do Repita, enquanto que a condição do Enquanto é uma condição para que ele continue sendo executado. Assim, para um repita e um enquanto equivalentes, que façam a mesma coisa, a condição de um é exatamente a condição contrária do outro.

2.7.3 Comando Para

O comando para é utilizado quando se sabe o número de vezes que a lista de comandos será repetida. Ele embute dentro de si a inicialização e o incremento da variável de controle, e o teste de saída da repetição. Seu formato é:

```
Para variável_de_controle de valor_inicial ate valor_final faca
   Lista de comandos
fimpara
```

Ao entrar no Para a variável de controle recebe o valor da expressão de valor inicial. Se for menor que o valor da expressão de valor final, a lista de comandos é executada e a variável de controle é automaticamente incrementada de 1. Isso (a execução dos comandos e o incremento da variável de controle) se repete até que o valor da variável de controle seja MAIOR que o valor da expressão de valor final.

2.8 Funções e Procedimentos

Além das funções pré-definidas na linguagem, O AlgoUCS permite o uso de funções e procedimentos criadas pelo programador. As declarações de funções e procedimentos devem ir no início do algoritmo, no bloco Var, junto às declarações de variáveis. A declaração de uma função tem o seguinte formato:

```
Funcao nome (lista de parâmetros formais):tipo do valor de retorno
var
declarações de variáveis locais
inicio
comandos
Fimfuncao
```

onde *lista de parâmetros formais* é uma lista de declarações dos parâmetros, separadas por ponto-e-vírgula. Cada declaração tem o mesmo formato de uma declaração normal de variáveis

¡lista de nomes¿:¡tipo¿

Ex:

(a,b,c:inteiro; d:real)

A passagem de parâmetros pode ser por valor (por cópia) ou por referência (por endereço). No caso de passagem por referência, a declaração deve iniciar pela palavra Var.

Ex: (a,b:inteiro;var c:real)

Nessa declaração, são passados 2 parâmetros inteiros por valor e um parâmetro inteiro por referência.

Obs: Não é permitida a passagem de vetores ou matrizes por referência.

O *tipo de valor de retorno* da função pode ser inteiro, real e lógico.

O comando utilizado para retornar um valor de uma função é o comando **retorne** *valor*

A função a seguir recebe um valor inteiro n e retorna o fatorial de n:

```
Funcao fat(n:inteiro):inteiro
var i,f:inteiro
inicio
   f<-1
   para i de 1 ate n faca
      f<-f*i
   fimpara
   retorne f
fimfuncao
```

O AlgoUCS permite também a definição de procedimentos. O formato da declaração de um procedimento é:

```
Procedimento <nome> (<lista de parâmetros formais>)
var
<declarações de variáveis locais>
inicio
<comandos>
Fimprocedimento
```

Obs: O AlgoUCS suporta a recursividade, tanto nas funções quanto nos procedimentos.

2.9 Erros de compilação

Erros de compilação:

- A variavel YYY não esta sendo usada - A variável foi declarada e não está sendo utilizada em nenhuma expressão. É possível até que ela esteja recebendo algum valor em alguma atribuição ou leitura, mas esse valor não está sendo utilizado
- Ambos operandos devem ser literais - O operador de concatenação de literais está sendo aplicado a um operando literal e um não literal
- Chamada de função no lugar de procedimento - Chamada de função só pode ser feita dentro de uma expressão
- Comando 'retorna' fora de função
- Comparação entre literal e não-literal
- Enviados parâmetros a mais do que declarado
- Enviados parâmetros a menos do que declarado - Chamada de função ou procedimento tem mais parâmetros do que o declarado na função ou procedimento
- ERRO no reconhecimento do tipo - tipo declarado não é vetor, inteiro, real, lógico, numérico ou literal
- Esperado dois pontos
- Esperava 'entao'
- Esperado identificador na declaração de parâmetro
- Esperava a palavra 'ate' no comando para
- Esperava a palavra De
- Esperava a palavra 'de' no comando para
- Esperava a palavra 'faca' no comando para
- Esperava a palavra 'fimpara' no comando para
- Esperava abre colchetes
- Esperava constante inteira
- Esperava fecha colchetes
- Esperava Fimse
- Esperava identificador no comando para
- Esperava início de comando
- Esperava novo identificador - Encontrou algo diferente de um identificador após a vírgula, em uma lista de declarações,
- Esperava o tipo
- Esperava ponto ponto
- Esperava Senao ou Fimse
- Esperava tipo dos elementos do vetor
- Esperava vírgula ou dois pontos

- Esperava sinal de atribuicao - Comando inicia por identificador, o que implica que é uma atribuição
- Expressão no Se não é expressão lógica - A expressão no comando Se deve ser uma expressão lógica, ou seja que resulta em *verdadeiro* ou *falso*
- Faltou Fimalgoritmo
- Faltou palavra Inicio
- Faltou Fimfuncao
- Faltou Fimprocedimento
- Faltou a palavra Algoritmo
- Faltou o 'ate':
- Faltou o 'faca'
- Faltou o 'FimEnquanto'
- Faltou o '('
- Faltou o ')'
- Função YYY não declarada
- Funcao YYY só tem um parâmetro - Foi feita uma chamada de uma função com apenas um parâmetro, passando mais de um parâmetro
- Função INT exige parâmetro real - A função INT, para converter real para inteiro, exige parâmetro real
- Funcao YYY já declarada
- Leitura de variavel lógica - A leitura só é permitida para variáveis numéricas ou literais
- Não é permitido declarar funcão dentro de função ou procedimento
- Não é permitido declarar Procedimento dentro de função ou procedimento
- Operador logico exige operandos logicos
- Parâmetro YYY já declarado
- Parâmetro por referência exige variável
- Passagem de vetor deve ser por referência
- Posição inicial do vetor é maior que final
- Potenciação exige operandos numéricos - foi aplicado o operador de potenciação a operandos lógicos ou literais
- Procedimento YYY não declarado
- Procedimento [YYY] já declarado
- Tipo da expresão de índice deve ser inteiro
- Tipo de parâmetro enviado incompatível com declarado

- Tipo da expressão incompatível na atribuição - Tipo da variável que recebe e da expressão atribuída são incompatíveis
- Tipo de expressao do valor final deve ser inteiro
- Tipo de expressao do valor inicial deve ser inteiro
- Tipo de retorno incompatível com declaração
- Unica operacao permitida com literais é "+- Algum outro operador está sendo aplicado a literais
- Variavel [YYY] já declarada
- Variavel YYY deve ser inteira - Variável de controle do Para deve ser inteira
- Variavel YYY nao declarada - Está sendo utilizada uma variável que não foi declarada
- Variavel YYY é vetor - A variável YYY está sendo usada como uma variável escalar comum, sem a especificação da posição utilizada
- Variavel YYY não é vetor - A variável YYY está sendo utilizada como um elemento de vetor, com especificação de posição, mas é uma variável escalar
- Vetor YYY tem duas dimensões - Está sendo especificado apenas um índice de um elemento de matriz
- Vetor YYY tem somente uma dimensão - Estão sendo especificados dois índices (linha, coluna) de um vetor unidimensional