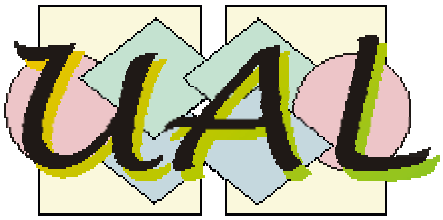


UNESA ALGORITHMIC LANGUAGE

UAL é uma linguagem interpretada para descrição de algoritmos em Português. Tem por objetivo auxiliar o aprendizado do aluno iniciante em programação através da execução e visualização das etapas de um algoritmo.

Com esse interpretador, os alunos poderão implementar seus próprios algoritmos utilizando recursos gráficos, ou seja, a representação lógica desses algoritmos através da animação, para compreender, de forma rápida e dinâmica, como as estruturas por eles desenvolvidas se comportam.

Com o intuito de difundir este novo sistema para a comunidade acadêmica, informamos que a linguagem UAL encontra-se disponibilizada gratuitamente, levando-se em consideração o leftright, ou seja, o sistema possui código aberto, permitindo aos interessados o desenvolvimento de novas funcionalidades, desde que mantida a filosofia inicial do mesmo e mencionadas as autorias.

Desenvolvido por:

Juarez A. Muylaert Filho - jamf@estacio.br
Andréa T. Medeiros - andrea@iprj.uerj.br
Adriana S. Spallanzani - spallanzani@uol.com.br

Versão 2.0
Data: mai/2001

ManUAL

Comandos em UAL

1. Introdução 3
2. Estrutura do UAL 4
3. Palavras-Chave 5
4. Pontuação 6
5. Maiúsculas e Minúsculas 7
6. Tipos de Dados 8
7. Declaração de Variáveis 9
8. Inicialização de Variáveis 10
9. Fluxos de Entrada e Saída 11
10. Precedência de Operadores 12
11. Operadores 13
12. Comandos para Tomada de Decisão 15
13. Comandos de Repetição 16
14. Comandos de Desvio 18
15. Vetores 20
16. Funções 21
17. UALGraph 25

1. Introdução

A grande dificuldade na concepção e no entendimento de algoritmos é o problema do relacionamento dos aspectos ativos e passivos, ou seja, como entender as estruturas dinâmicas das possíveis execuções do algoritmo a partir de sua estrutura estática.

A linguagem UAL visa auxiliar o processo de aprendizagem do aluno iniciante em programação, integrando os aspectos dinâmicos e estáticos em uma só ferramenta, permitindo ao aluno escrever seus programas em "Portugol" - pseudo-linguagem de programação, executando e visualizando, através de gráficos animados, o comportamento ativo do seu algoritmo.

Esta linguagem aborda o mínimo de blocos básicos (de dados e controle) necessários para o aprendizado de algoritmos, os quais encontram-se descritos nos demais itens deste Help.

2. Estrutura do UAL

Todo programa UAL inicia-se com a palavra reservada *prog* seguida do nome do programa, que obedece as normas de nomenclatura de identificadores. A seguir, são feitas as declarações de variáveis, caso existam, conforme o item 7 deste manual e, posteriormente, são inseridos os blocos de comandos, encerrando-se o programa com a palavra-chave *fimprog*.

```
prog <nome do programa>
     <declaração de variáveis> <opcional>
     <seqüência de comandos>
fimprog
```

Um exemplo de programa em UAL é:

```
prog meuPrimeiroPrograma
     string ola;
     ola <- "Olá Mundo!";
     imprima ola;
fimprog
```

Que também pode ser representado da seguinte forma:

```
prog meuPrimeiroPrograma
     imprima "Olá Mundo!";
fimprog
```

3. Palavras-Chave

Palavras-chave são palavras reservadas da linguagem, não podendo as mesmas serem atribuídas a nenhum identificador (nome do programa e das variáveis definidos pelo usuário) pertencente ao programa. Por exemplo, não podemos ter uma variável chamada se.

As palavras-chave em UAL são:

prog fimprog int real logico string se

senao enquanto faca para leia imprima

div raiz sen cos tan exp log pi abs

intreal realint formatar saia continue pare

strtam strconcat strcopia strprim strelem

strcomp strult strnprim strnresto

As regras para nomenclatura dos identificadores encontram-se no item Declaração de Variáveis.

4. Pontuação

Um sinal de pontuação é um tipo de palavra-chave abreviada. É um símbolo composto por um ou dois caracteres, que tem um significado especial para o interpretador. O conjunto completo de sinais da linguagem inclui os seguintes símbolos:

/* */ # { } ( ) [ ] , ; \n \t

O sinal ";" encerra uma instrução, exceto nos casos em que a mesma possua delimitadores, como no comando enquanto. Esse sinal informa ao interpretador que ocorreu o fim de uma instrução. Desse modo, podemos ter mais de uma instrução na mesma linha ou ter uma única instrução em mais de uma linha.

O uso dos delimitadores será demonstrado mais adiante, nos itens 12 e 13.

Os símbolos #, /* e */ devem ser inseridos para especificar um comentário. Comentários em programas são mensagens que só serão vistas pelo programador, sendo completamente ignoradas pelo interpretador. São utilizados visando dar maior compreensão ao algoritmo.

O símbolo # permite que apenas uma linha seja comentada por vez, como segue o exemplo:

```
# Isto é um comentário
# Autor <nome>
prog meuPrimeiroPrograma
        string ola;     # Declarando variável ola
        ola <- "Olá Mundo!";
        imprima ola;
fimprog
```

Já os símbolos /* e */ delimitam o início e o fim de um comentário, independente do número de linhas que o mesmo possua, como no exemplo abaixo:

```
/* Este é o início de um comentário
   Autor <nome>
   Data: <data>
*/
```

Esses comentários podem ser inseridos em qualquer posição dentro do programa.

O símbolo "\t" efetua uma tabulação em uma string e o "\n" permite uma quebra de linha na saída do programa. No item 9 é dado um exemplo de suas aplicações, juntamente com o comando imprima.

5. Maiúsculas e Minúsculas

A linguagem UAL é sensível a maiúsculas e minúsculas em seus identificadores e palavras reservadas. Exemplo:

ola, Ola, OLA, olA

São variáveis diferentes para o UAL.

String - Não é uma palavra reservada como *string*.

Devido a esta diferença, é recomendável utilizar um padrão para a nomenclatura dos identificadores. Aconselhamos que os identificadores sejam sempre iniciados por letras minúsculas, seguidas de iniciais maiúsculas para nomes compostos.

6. Tipos de Dados

O UAL fornece um conjunto de tipos de dados básico, onde cada um deles serve a um propósito específico, definindo desse modo, o escopo dos valores que as variáveis podem assumir no programa.

Os tipos básicos existentes são:

- representando o conjunto dos números inteiros;
- representando os pontos flutuantes;
- representando um conjunto de um ou mais caracteres
- representando os valores lógicos verdadeiro e falso.

7. Declaração de Variáveis

Todas as variáveis devem ser declaradas antes de serem usadas. Esta declaração deve ser feita no início do programa.

As variáveis possuem uma característica em comum: um tipo de dado associado. Isso significa que, além de escolhermos um nome apropriado para uma variável, devemos também dizer que tipo de informação deve ser armazenada nela.

Uma declaração é definida por um tipo, seguido por uma lista de uma ou mais variáveis daquele tipo, tal como o seguinte exemplo:

int num2; *string* resp, nome;

As variáveis podem ser distribuídas entre declarações livremente. O exemplo acima poderia ser igualmente escrito como:

int num2;
string resp;
string nome;

Os nomes das variáveis podem ser compostos de letras e dígitos, sendo que o primeiro caractere deve ser, obrigatoriamente, uma letra.

Identificadores válidos: count, teste123, Programa.
Identificadores inválidos: 123count, Teste_de_loop, Teste...Novo.

8. Inicialização de Variáveis

Para inicializar uma variável, ou seja, atribuir um valor a ela, deve-se proceder da seguinte forma:

```
var <- 4;
```

onde var é o nome da variável declarada, <- é o símbolo de atribuição e 4 é o valor a ela atribuído.

Quando interpretamos as instruções acima, os nomes das variáveis são substituídos pelos valores associados na memória. É importante observar que o resultado da expressão do lado direito de um comando de atribuição deve ser coerente com o tipo declarado para a variável do lado esquerdo.

Ao longo do programa é possível, através de novas atribuições, alterar os valores dessas variáveis na memória.

9. Fluxos de Entrada e Saída

O UAL permite que um programa receba ou envie dados para o ambiente externo. Isto pode ocorrer com o uso dos comandos *leia* e *imprima*, respectivamente.

O comando *leia* permite a entrada de um único dado, fornecido pelo usuário, através da entrada padrão, que normalmente é o teclado. Ao executarmos esse comando, atribuiremos o dado fornecido a uma variável, conforme demonstra o seguinte exemplo:

```
int x;
leia x;
```

Nesse momento, o programa aguardará a entrada de um valor inteiro que será armazenado na variável x. O usuário deve fornecer esse valor digitando um número e pressionando, em seguida, a tecla <Enter>.

O comando *imprima* permite que sejam apresentados na saída padrão, normalmente o monitor, mensagens e valores obtidos através da execução do programa, como mostra o exemplo abaixo:

```
a <- 400;
imprima "\tO valor de a é = ", a, "\n";
```

Este comando pode vir acompanhado do comando *formatar*, que permite delimitar o número de casas decimais de um algarismo em ponto flutuante, de acordo com a sintaxe abaixo:

```
a <- 400.6794;
imprima "Valor de a = ", formatar(a,2);
```

Cujo resultado será:

```
Valor de a = 400.67
```

Obs.: O comando *formatar* converte um número real para o tipo string, efetuando o truncamento em suas casas decimais.

10. Precedência de Operadores

Todas as operações realizadas no UAL obedecem aos critérios de precedência dos operadores, conforme a relação abaixo:

Primeiro	- funções e parênteses
Segundo	- expressões aritméticas (potência e módulo)
Terceiro	- expressões aritméticas (multiplicação e divisão)
Quarto	- expressões aritméticas (adição e subtração)
Quinto	- operadores relacionais.
Sexto	- operadores lógicos.

Na utilização normal, operadores de mais alta precedência executam as suas operações em expressões antes de operadores de menor precedência. Por exemplo:

```
a = b + c * d
```

Nesse exemplo, primeiro será calculado o produto entre c e d, para depois somar o resultado a b.

Para forçar uma ordem de avaliação diferente, deve-se fazer uso de parênteses, como no exemplo:

```
a = (b + c) * d
```

11. Operadores

O UAL é composto de operadores - símbolos que executam várias operações sobre os seus argumentos. O sinal de mais (+) é um operador. Em uma expressão, ele soma dois valores:

a <- b + c;

As variáveis a, b, c poderiam ser de quaisquer tipos numéricos de dados. Nesta operação, b é acrescentado a c, sendo o resultado atribuído à variável a.

O sinal de atribuição <- nessa expressão é também um operador, com a capacidade única de copiar um valor a sua direita para a variável a sua esquerda.

No UAL, existem operadores aritméticos, relacionais, lógicos e de atribuição, que estão demonstrados abaixo.

Operações aritméticas:

adição (+)
subtração (-)
multiplicação (*)
divisão real (/)
divisão inteira (div)
módulo (%)
potência real (**)
potência inteira (^)

Caso as operações sejam efetuadas com os operandos de um mesmo tipo, o valor retornado seguirá a mesma tipagem, caso contrário, o valor retornado será do tipo real.

Vale ressaltar que as operações aritméticas permitidas aceitam apenas os tipos inteiro e/ou real.

Operadores relacionais:

maior (>)
maior ou igual (> =)
menor (<)
menor ou igual (< =)
igual (= =)
diferente (< >)

Operadores lógicos:

conjunção (&&)
disjunção (||)
negação (!)

Esses dois últimos operadores tem como resultado um valor lógico, do tipo verdadeiro ou falso.

Operadores de atribuição:

atribuição (<-)
incremento (++)
decremento (—)

Os operadores de incremento e decremento são similares à atribuição, realizando operações de adição e subtração, respectivamente. Dessa forma, uma variável é incrementada ou decrementada em uma unidade. Portanto, essa variável deve ser do tipo inteiro.

Em uma atribuição, não podem ser usados operadores de incremento e decremento.

Exemplos:

```
x <- a;                  # x recebe o valor de a.
x—;                      # x é decrementado em 1.
x <- x-1;                # operação idêntica a anterior.
x++;                     # x é incrementado em 1.
x <- x++ ou x <- x—;     # operações não permitidas.
```

12. Comandos para Tomada de Decisão

O comando utilizado para tomada de decisões no UAL é o *se*, que é acompanhado, opcionalmente, pelo *senao*. Sua sintaxe é bastante simples, necessitando acrescentar uma expressão que retorne um tipo lógico após a palavra-chave *se* e uma lista de comandos, caso o *senao* seja utilizado, também haverá uma lista de comandos.

Por exemplo, para exibir o valor de uma variável "conta" apenas se este valor for maior do que 100, pode-se utilizar a seguinte instrução:

```
se (conta > 100)
{
        imprima conta;
}
senao
{
        imprima "Saldo insuficiente.";
}
```

Os parênteses, bem como as chaves, são obrigatórios.

13. Comandos de Repetição

Na linguagem UAL, podemos utilizar os comandos *enquanto*, *para* ou *faca-enquanto*, para aplicar instruções repetidamente.

A INSTRUÇÃO *enquanto*

O comando *enquanto* tem sintaxe semelhante a do comando *se*, bastando inserir uma expressão que retorne um tipo lógico após a palavra-chave *enquanto* e uma lista de comandos entre chaves, como mostra o exemplo:

```
i <- 0;
enquanto ( i < 10)
{
        leia var;
        i++;
}
```

Nesse comando, a expressão relacional é avaliada antes da execução da instrução ou bloco de comandos, ou seja, caso essa expressão seja falsa no início da execução do loop, os comandos não serão executados.

A INSTRUÇÃO *para*

Quando um programa pode calcular previamente o número de vezes que um bloco de instruções deve ser executado, uma instrução *para* é normalmente, a melhor escolha para a construção da estrutura de repetição. A instrução para possui os seguintes elementos:

A palavra-chave *para*;
Uma expressão composta por três partes, entre parênteses;
Uma instrução ou bloco de comandos entre chaves.

O comando *para* combina os três elementos acima, tendo como formato geral:

```
para (i<-0; i < 10; i++)
{
        leia var;
}
```

Dentro dos parênteses, após a palavra-chave *para*, há três argumentos que controlam a ação do loop. No primeiro argumento, há uma instrução que é executada uma única vez, antes do início do loop, atribuindo um valor a uma variável. A inicialização e a atribuição de valor que compõem o primeiro e o terceiro argumento do comando *para*, respectivamente, fazem referência a uma mesma variável. O segundo argumento é uma expressão relacional que a instrução *para* testa no princípio do loop. O último argumento é executado ao final do loop, após a execução da instrução ou bloco de comandos, que devem estar entre chaves. Esse argumento pode ser uma atribuição, um incremento ou decremento.

A INSTRUÇÃO *faca-enquanto*

É um tipo de instrução *enquanto* invertida, ou seja, ela executa uma ou mais ações enquanto uma expressão for verdadeira. Exemplo:

```
i <- 0;
faca
{
        leia var;
        i++;
}
enquanto (i < 10)
```

Seu funcionamento se diferencia do comando *enquanto* pelo fato da expressão relacional que controla o loop estar ao final da instrução, e não no seu início. Com isso, a instrução ou o bloco de comandos é executado pelo menos uma vez, pois o programa não avalia a expressão de controle até que tenha executado essas instruções.

Um loop *faca-enquanto* requer ambas as palavras-chaves: *faca*, no inicio, e *enquanto*, no final. As instruções a serem executadas são colocadas entre o *faca* e o *enquanto*, como demonstrado no exemplo.

14. Comandos de Desvio

O UAL tem três comandos que realizam desvios incondicionais: *continue*, *pare* e *saia*. Destes, você pode usar o *saia* em qualquer lugar em seu programa. Os comandos *pare* e *continue* devem ser utilizados em comandos de repetição.

A INSTRUÇÃO *saia*

O comando *saia* permite a saída de um programa. Este comando provoca uma terminação imediata do programa inteiro, forçando um retorno ao sistema operacional.

A forma geral do comando *saia* é:

saia;

A INSTRUÇÃO *pare*

O comando *pare* serve para forçar a terminação imediata de um laço, isto é, o laço é interrompido evitando o teste condicional. A seguir, são executados os comandos que sucedem o laço.

```
b <- 3;
para (i<-o;i<10;i++) {
        a <- b * i;
        imprima " a = ", a;
        se (a > 10) {
                pare;
        }
}
```

A saída do programa será:

a = 0 a = 3 a = 6 a = 9 a = 12

A INSTRUÇÃO *continue*

O comando continue trabalha de uma forma parecida com a do comando *pare*. Porém, em vez de forçar a terminação do laço, ele força que ocorra a próxima iteração, não executando nenhum código intermediário.

```
b <- "Este teste remove os espaços em branco";
para (i<-0; i<strtam(b); i++) {
        se (strelem(b,i) == " ") {
            continue;
        } senao {
                imprima strelem(b,i);
        }
}
```

A saída do programa será:

```
Estetesteremoveosespaçosembranco
```

15. Vetores

Os vetores reúnem várias informações de um mesmo tipo em uma única estrutura. Para isso, é necessário declarar, além do nome do vetor, o seu tamanho. Devemos informar antecipadamente quantos elementos o vetor irá armazenar dentro dele, ou seja, seu tamanho é estático (não varia com a execução do programa).

No exemplo abaixo, declaramos um vetor do tipo inteiro com 10 posições.

int x[10];

Cada elemento de um vetor é tratado como se fosse uma variável simples. Para referência a um elemento do vetor, utiliza-se o nome do vetor seguido da identificação do elemento (índice) entre colchetes. Esse índice deve ser do tipo inteiro. Exemplo:

x[0] <- 0;

x[9] <- 9;

No UAL, o primeiro elemento de um vetor sempre é representado pelo índice 0, ou seja, se um vetor contiver 10 posições, seus índices irão variar de 0 a 9.

16. Funções

As funções em UAL estão divididas em Funções Matemáticas, Funções Para Manipulação de Strings e Funções de Conversão de Tipos, conforme detalhamento abaixo:

Funções Matemáticas

Englobam as seguintes funções: *raiz, sen, cos, tan, pi, exp, log, abs.*

raiz
Retorna a raiz quadrada de um número, em real. Se você chamá-la com argumento negativo, ocorrerá um erro de domínio.

Exemplo: imprima raiz(49);
Resultado: 7.0

sen
Retorna o seno de um número, em radianos.

Exemplo: imprima sen(0);
Resultado: 0.0

cos
Retorna o coseno de um número, em radianos.

Exemplo: imprima cos(0);
Resultado: 1.0

tan
Retorna a tangente de um número, em radianos.

Exemplo: imprima tan(0);
Resultado: 0.0

pi
Retorna a constante pi, em real

Exemplo: imprima pi;
Resultado: 3,141592653589793

exp
Retorna o exponencial de um número, em real.

Exemplo: imprima exp(0);
Resultado: 1.0

log

Retorna o logaritmo de um número, em real. Se você chamá-la com argumento negativo, ocorrerá um erro de domínio.

Exemplo: imprima log(1);
Resultado: 0.0

abs
Retorna o valor absoluto de um número.

Exemplo: imprima abs(-5);
Resultado: 5

Funções para Manipulação de strings

Incluem as seguintes funções: *strtam, strconcat, strcopia, strprim, strresto, strelem, strult, strnprim, strnresto.*

strtam
Retorna o tamanho de uma string.

Exemplo: imprima strtam("string ");
Resultado: 6

strconcat
Concatena duas strings.

Exemplo: imprima strconcat("string","!");
Resultado: string!

strtcopia
Copia o conteúdo de uma string.

Exemplo: imprima strcopia("string");
Resultado: string

strprim
Retorna o primeiro elemento de uma string.

Exemplo: imprima strprim("string");
Resultado: s

strresto
Retorna todos os elementos de uma string, exceto o primeiro.

Exemplo: imprima strresto("string");
Resultado: tring

strelem
Retorna o enésimo elemento de uma string.

Exemplo: imprima strelem("string", 3);
Resultado: r

strcomp
Compara duas strings, retornado "maior", "menor" ou "igual".

Exemplo: imprima strcomp("string", "String");
Resultado: maior

strult
Retorna o último elemento de uma string.

Exemplo: imprima strult("string");
Resultado: g

strnprim
Retorna os n primeiros elementos de uma string.

Exemplo: imprima strnprim("string", 3);
Resultado: str

strnresto
Retorna os elementos de uma string após os n primeiros.

Exemplo: imprima strelem("string", 2);
Resultado: ring

Funções de Conversão de Tipos

Existem duas funções em UAL que podem ser utilizadas para converter tipos inteiro em real ou vice-versa. São elas: intreal, realint.

intreal
Converte um número inteiro em real.

Exemplo: imprima intreal(20);
Resultado: 20.0

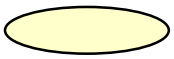
realint
Converte um número real em inteiro, arredondando-o.

Exemplo: imprima intreal(20.8);
Resultado: 21

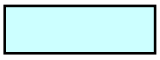
17. UALGraph

O Módulo UALGraph permite a execução dos programas fontes feitos em UAL em um ambiente gráfico, ou seja, é possível visualizar a representação lógica desses algoritmos através da animação, para compreender, de forma rápida e dinâmica, como as estruturas desenvolvidas se comportam.

Os símbolos que representam a estrutura do programa seguem os seguintes padrões:



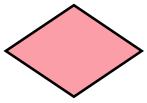
Delimita o início e fim de um programa



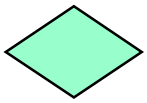
Atribuição

Incremento

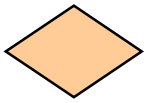
Decremento



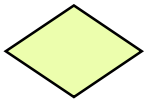
Enquanto



Para



Se-Senão



Faça-Enquanto

Leia

Imprima

Saia

Continue

Pare