Instalação do Visualg

Execute o aplicativo na pasta: MEUS Exercícios

O código do Visualg pode ser convertido para a Linguagem Pascal, para isso, acesse a aba “Código” no menu do programa.

O Visualg permite apenas **um comando por linha**: não há necessidade de *tokens* separadores de estruturas, como o ponto-e-vírgula ou chaves

As **palavras-chave** do Visualg foram implementadas **sem acentos**, cedilha, etc

O Visualg também **não distingue maiúsculas e minúsculas** no reconhecimento de palavras-chave e nomes de variáveis

O Visualg prevê quatro tipos de dados: **inteiro**, **real**, **cadeia de caracteres** e **lógico**

A **atribuição de valores** a variáveis é feita com o operador **<-**

MOD ou **%**	Operador de módulo (isto é, r**esto da divisão inteira)**. Por exemplo, 8 MOD 3 = 2.

+	Operador de **concatenação de *strings***

A execução de uma expressão lógica obedece como prioridade a **ordem** dos operadores **NOT, AND e OR**

O Visualg implementa as **3 estruturas de repetição**:

- o laço contado **para**...ate...faca. Nessa iteração, é possível determinar o número exato de repetições.
- e os laços condicionados **enquanto**...faca. Sua declaração de condicional é feita no seu início, oposto ao comando Repita.
- e **repita**...ate. Por sua condição estar localizada no final ***(até)***, sua ação será executada pelo menos uma vez! antes da avaliação da condição

Subprograma é um programa que **auxilia o programa principal** através da realização de uma determinada subtarefa. Também costuma **receber os nomes de *sub-rotina, procedimento, método* ou *módulo***. Os subprogramas **são chamados** dentro do corpo do programa principal **como se fossem *comandos***. Após seu término, a execução continua a partir do ponto onde foi chamado. O **procedimento não retorna** nenhum **valor**

Os Subprogramas são descritos após a declaração das variáveis e antes do corpo do programa principal.

Cada subprograma, além de ter acesso às variáveis do programa que o chamou (são as **variáveis** de escopo ***globais***), pode ter suas próprias variáveis (são as **variáveis** de escopo ***locais***), que existem apenas durante sua chamada.

Ao se chamar um subprograma, também é possível passar-lhe determinadas informações que recebem o nome de ***parâmetros***: **valores** que, na linha de chamada, ficam **entre os parênteses** e que estão separados por vírgulas. Algumas linguagens não aceitam que o nome dos parâmetros sejam iguais aos nomes das variáveis.

Os procedimentos e funções podem ter passagem de parâmetros **por valor ou referência.**

A presença (opcional) da **palavra-chave var indica** passagem de parâmetros **por referência**; caso **contrário**, a passagem será **por valor**.

Na **passagem por referência** o subprograma não recebe apenas um valor, mas sim o **endereço** (da memória) **de uma variável global**. Portanto, qualquer modificação que for realizada no conteúdo deste parâmetro afetará também a variável global que está associada a ele.
Sendo assim, use passagem por referência quando quiser alterar também as variáveis globais.

Há um caso particular de subprograma que recebe o nome de ***função***. Uma função, além de executar uma determinada tarefa, **retorna um valor** para quem a chamou, que é o resultado da sua execução.

Como uma boa prática de programação, é recomendável não utilizar comandos de entrada ou saída **dentro do bloco de subprogramas**; reserve-os para o corpo principal do programa.

As linguagens que só possuem métodos (como Java) retornam “nada” (void) quando se deseja apenas um procedimento, ou retornam algo - tornando-se funções.

Use funções para retornar valores para dentro de uma variável. Um dos grandes benefícios dessas rotinas é não precisar copiar o código todas as vezes que precisar executar aquela operação, além de deixar a leitura do código mais intuitiva e mais curta. Use procedimentos quando não precisar retornar um valor, como exemplo: enviar e-mails promocionais ou criar uma rotina.

A função **numpcarac( )** converte inteiros para string

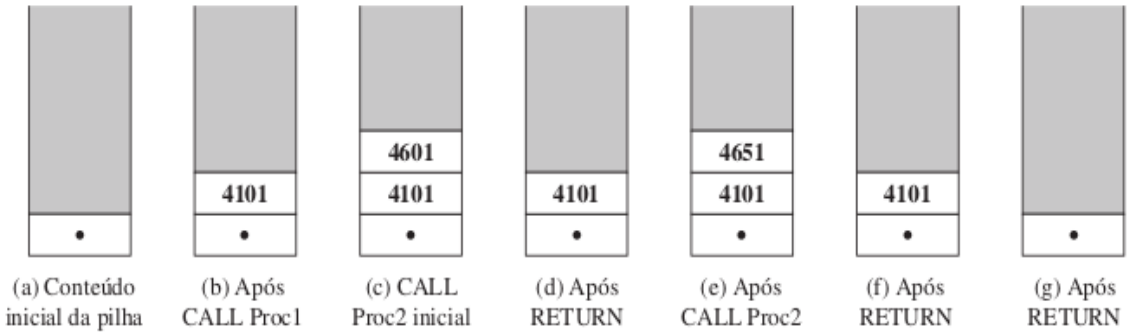
Aninhamento é uma sub-rotina encapsulada noutra, ou seja, um subprograma também pode ter seus próprios subprogramas. É uma técnica de encapsulamento (não confundir com encapsulamento POO), útil para dividir tarefas procedimentais em subtarefas que fazem sentido somente localmente.

Os **vetores** são variáveis compostas homogêneas unidimensionais que contêm vários espaços para guardar valores. Índice é a posição do valor dentro do vetor, sendo identificado por colchetes [ ].

Matrizes são Variáveis Compostas Homogêneas Multidimensionais, ou seja, com mais de uma dimensão

Iteração ou Recursividade?

A iteração e recursão são técnicas que permitem que um bloco de instruções seja executado repetidamente, porém, ambas possuem abordagens diferentes.
Há algoritmos de ordenação de dados, inteligência artificial e ainda outros que não compensa abordá-los iterativamente a não ser por recursão. E da mesma forma existe o oposto em alguns outros algoritmos como no cálculo de Fibonacci onde é melhor evitar a recursão ou isso pode gerar um estouro de memória *stack* (famoso ***stack overflow***, que significa que a pilha que foi mencionada para armazenamento de procedimentos atingiu o seu limite). Cada chamada recursiva necessita da criação de uma nova sub-rotina em memória (que pode conter endereços de parâmetros e endereço de retorno), pois os valores dos parâmetros não podem ser perdidos e a recursão deve funcionar adequadamente para cada chamada como se fosse uma nova função totalmente à parte da que está em execução.



Empilhamento em memória de procedimentos que são chamados

Em resumo:

Iteração repete uma tarefa até que ela seja completa, como exemplo: comando ***(for).***
Prós: é mais rápida em muitos casos, uma vez que, em sua forma mais simples e genérica, ela consiste na reexecução de um conjunto de instruções
Contras: pode ser bem trabalhoso e demorado de programar.

Recursão quebra essa tarefa em tarefas menores até que haja uma solução, como exemplo: comando ***(if)***. A recursão permite solucionar muitos problemas, exigindo menos esforço de codificação e lógica, através da divisão do mesmo em pequenas partes. Recursão significa que uma sub-rotina (método, procedimento ou função) está chamando ela própria. A ideia de recursividade está ligada ao conceito de “dividir para conquistar”.
Prós: confere ao código maior legibilidade, exigindo menos esforço de codificação e lógica, tornando mais simples sua compreensão.
Contras: gasta mais recurso de processamento.