

Variáveis e Memória

Revisão

Estudamos, na Introdução, que programas de computador implementam algoritmos, os quais manipulam um conjunto de dados para produzir um resultado. O algoritmo é um conjunto de instruções que são executadas sequencialmente. Em cada uma delas, o algoritmo consulta dados armazenados na memória, realiza algum tipo de processamento sobre estes e, por fim, armazena novamente o resultado na memória.

Este funcionamento dos programas motiva o estudo da memória do computador, na qual é realizado o armazenamento dos dados manipulados pelo algoritmo.

Conceitos

A memória pode ser entendida como uma sequência de células nas quais se pode armazenar um valor correspondente a um dado. Para fins didáticos, consideramos memórias com um número ilimitado de células. Na prática, os computadores atuais costumam apresentar algumas centenas de milhões de células, ou mesmo bilhões de células.

Como as células estão organizadas sequencialmente, atribui-se à elas um número correspondente a sua **posição** na sequência, contando a partir do zero. Este número denomina-se **endereço da célula**. A manipulação dos endereços de memória será estudada mais para o final do curso.

Cada **célula de memória** é caracterizada pelo seu **endereço** e seu **conteúdo**.

Uma célula de memória armazena um único valor (conteúdo), que é uma sequência de bits. Um bit nada mais é que um dos valores 0 ou 1. Essa sequência de bits pode representar um número, um caractere (letra), ou outro objeto, como veremos oportunamente. Eventualmente, quando for necessário armazenar um valor relativamente grande, por exemplo, um número com muitos dígitos, poderá surgir a necessidade de se utilizar mais de uma célula consecutiva da memória para armazenar o dado.

O armazenamento do conteúdo das células é volátil (não permanente). Se ocorrer alguma falha no computador ou cessar a alimentação elétrica, o seu conteúdo é perdido.

Operações sobre a memória

A memória permite dois tipos de operações: **leitura** e **escrita**. Sempre precisamos indicar o endereço da célula sobre qual desejamos realizar a operação.

A primeira delas, a **leitura**, localiza a célula correspondente ao endereço desejado e consulta o valor armazenado nesta célula. Após a leitura, a célula continua com o valor original que estava armazenado antes de se executar esta operação. Note que a leitura retorna o conteúdo de uma cela de memória. Se o dado estiver contido em mais de uma célula, então todas as celas onde está armazenado o dado devem ser consultadas para que se possa recuperar o valor correto do dado.

A **escrita** trabalha com um endereço e um valor. De forma semelhante à leitura, a escrita localiza a célula do endereço desejado, mas em seguida substitui o conteúdo da mesma pelo novo valor. Durante este processo, o conteúdo anterior da célula é perdido de forma irreversível. De forma semelhante à leitura, se o dado ocupar mais de uma célula, então todas as células associadas a ele devem ser adequadamente escritas para que o novo valor do dado seja corretamente armazenado na memória.

Processamento de um programa

Essencialmente, para executar um algoritmo, o computador repete o seguinte ciclo:

- 1) Obtém um ou mais valores da memória, utilizando operações de leitura.
- 2) Realiza algum tipo de processamento sobre os dados consultados na memória. Como consequência, é gerado um novo valor.
- 3) Efetua uma escrita para armazenar o resultado em uma célula da memória.

Um programa nada mais faz que **manipular valores** que estão **armazenados nas células da memória**. O efeito final do programa é dado pelo **conteúdo final** das células de memória que ele manipula.

Exemplo

Vamos relembrar o algoritmo de Euclides que calcula o MDC (máximo divisor comum), apresentado na Introdução. Veja ao lado. No início, temos dois números, que são a entrada do algoritmo. Portanto, estes dois números devem ser armazenados cada um em uma célula de memória. Assumiremos que o primeiro número será armazenado na célula de endereço 1 e o segundo número na célula de endereço 2. A Figura 1 mostra como os valores são armazenados nas células de memória.

No terceiro passo, calculamos o quociente

e o resto da divisão do primeiro número pelo segundo. Como não desejamos perder o valor destes dois números, precisamos escrever o quociente e o resto obtidos em novas células de memória. Neste exemplo, decidimos guardar o quociente e o resto, respectivamente, nas células de endereço 3 e 4.

Memória:	
7	
6	
5	
4	Resto
3	Quociente
2	Número 2
1	Número 1
0	

Figura 1 - Conteúdo das células da memória

- 1) **Leia** um número e escreva na célula 1
- 2) **Leia** um número e escreva na célula 2
- 3) **Divida** o valor da célula 1 pelo valor da célula 2. Guarde o quociente na célula 3 e o resto na célula 4.
- 4) **Se** o valor da célula 4 for 0 (zero), então **mostre** o valor da célula 2 e **PARE**.
- 5) **Escreva** na célula 1 o valor da célula 2.
- 6) **Escreva** na célula 2 o valor da célula 4.
- 7) **Retorne** ao passo 3.

Algoritmo 1 - MDC usando endereços de células de memória

Finalmente, antes de o algoritmo continuar a execução, note que a próxima operação de divisão espera que os números estejam nas células de endereço 1 e 2. Por este motivo, precisamos copiar, respectivamente, o quociente (que está na célula de endereço 3) para a célula de endereço 1 e o resto (que está na célula de endereço 4) para a célula de endereço 2.

Desafio: É possível reescrever este algoritmo de forma a utilizar menos células de memória durante o processamento? Quais seriam as vantagens e desvantagens de reduzir o número de células utilizadas por um algoritmo?

Discussão sobre células de memória

A princípio, qualquer algoritmo poderia ser descrito utilizando-se apenas operações sobre células da memória. Para isso, basta associar cada um dos valores manipulados pelo algoritmo a uma célula de memória. Cada instrução do algoritmo é acompanhada pelos endereços das células onde estão localizados os dados, conforme foi exemplificado no Algoritmo 1.

Esta forma complicada introduz um nível desnecessário de complexidade referente à tecnologia de armazenamento do computador, mas não enriquece a descrição do algoritmo. Em particular, essa descrição só vai funcionar quando esse algoritmo, ao executar, puder dispor exatamente das células de memória cujos endereços referencia em seu código. E, se na próxima execução desse algoritmo essas células não estiverem todas disponíveis, i.e., estiverem sendo utilizadas por algum outro programa? Durante o desenvolvimento de um programa de computador, é praticamente impossível prever quais posições de memória estarão disponíveis no início da execução do programa. Além disso, existem outros programas em execução simultânea, e eles poderiam potencialmente acessar as mesmas células de memória e gerar conflitos.

O uso de endereços fixos de memória também torna-se impraticável para algoritmos mais longos, pois será muito difícil para o programador associar manualmente cada dado a uma célula de memória. Muito menos ainda para uma equipe de programadores. Erros comuns ocorrem quando um programador acidentalmente confunde o significado de duas células de memória. Além disso, introduzir modificações em um algoritmo ou realizar alguma manutenção no código torna-se uma tarefa muito penosa.

Ademais, nem todos os dados podem ser acomodados em uma única célula de memória. Por exemplo, números muito grandes, podem exigir duas ou mais células. Neste caso, o programador precisa lidar manualmente com estas situações excepcionais.

Por este motivo, as linguagens de programação criaram o conceito de **variáveis** para abstrair a complexidade do endereçamento das células de memória. Lembre-se que a motivação de utilizar linguagens de programação foi encontrar uma *forma simples* de representar algoritmos e que não dependa da forma de funcionamento de cada computador.

Variáveis

A variável é uma abstração da célula de memória.

Imagine agora que o programador, ao invés de utilizar os endereços físicos da memória para identificar as células que usa, tenha à sua disposição **rótulos simbólicos** com nomes intuitivos. Ao invés de utilizar o endereço (posição da célula dentro da memória), o valor em questão é identificado através desse rótulo (ou nome) simbólico, escolhido com auxílio do bom senso do programador. Sempre que for necessário referir-se a algum dado, o programador utiliza o rótulo associado à variável que contém o mesmo. O compilador realiza automaticamente a transformação dos rótulos em endereços de memória, sem que seja necessário nos preocuparmos com isso.

Além disso, logo antes do programa iniciar sua execução, o computador pode escolher livremente quais células físicas da memória serão associadas a cada um dos nomes simbólicos. Assim, logo antes de executar, o computador escolhe células que não estejam sendo utilizadas por nenhum outro programa, associa estas células aos nomes simbólicos do programa que vai entrar em execução e bloqueia o uso dessas células por qualquer outro programa que venha a ser iniciado enquanto o programa em questão estiver em andamento. Quando esse programa terminar, as células associadas aos nomes simbólicos que usou são liberadas pelo computador, e ficam livres para serem utilizadas por outros programas.

Uma célula de memória, quando identificada por um rótulo, será denominada uma **variável**. Ao invés de pensar na memória como uma sequência longa de células, o podemos entender a memória como um repositório que contém as variáveis associadas ao programa.

Cada **variável** é caracterizada pelo seu **rótulo** e seu **conteúdo**.

Exemplo

Vamos rever o algoritmo de Euclides que calcula o MDC (máximo divisor comum), apresentado na Introdução, desta vez utilizando variáveis. Ele começa lendo dois números, que são a entrada do algoritmo. Portanto, estes dois números devem ser armazenados em duas variáveis, que decidimos chamar de A e B. Não é mais necessário nos preocuparmos com a posição exata que esses dados vão ocupar na memória. A Figura 2 mostra a representação abstrata da memória.

No terceiro passo, o algoritmo calcula o quociente e o resto da divisão dos dois números (identificados pelas variáveis A e B). Como não desejamos perder o valor destes dois números, precisamos escrever o quociente e o resto em novas variáveis, que foram chamadas, respectivamente, de Q e R. Finalmente, antes de o algoritmo repetir o ciclo,

note que a próxima divisão espera que os dois números sobre os quais vai operar estejam nas variáveis A e B. Por este motivo, precisamos copiar, respectivamente, o quociente (que está na variável Q) para a variável A e o resto (que está na variável R) para a variável B.

- 1) **Leia** um número e armazene na **variável A**.
- 2) **Leia** um número e armazene na **variável B**.
- 3) **Divida** o valor da variável A pelo valor da variável B. Guarde o quociente na **variável Q** e o resto na **variável R**.
- 4) **Se** o valor da variável R for 0 (zero), então **mostre** o valor da variável B e **PARE**.
- 5) **Copie** o conteúdo da variável B para a variável A.
- 6) **Copie** o conteúdo da variável R para a variável B.
- 7) **Retorne** ao passo 3.

Algoritmo 2 - MDC usando variáveis

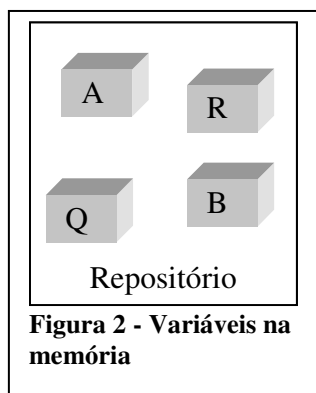


Figura 2 - Variáveis na memória

Desafio: É possível reescrever este algoritmo de forma a utilizar menos variáveis. Como? Quais seriam as vantagens e desvantagens de reduzir o número de variáveis? Compare a importância e a dificuldade da redução do número de células de memória com a redução do número de variáveis.