Fundamentos de Programação

Filipe Francisco

05 de abril de 2018

Loops

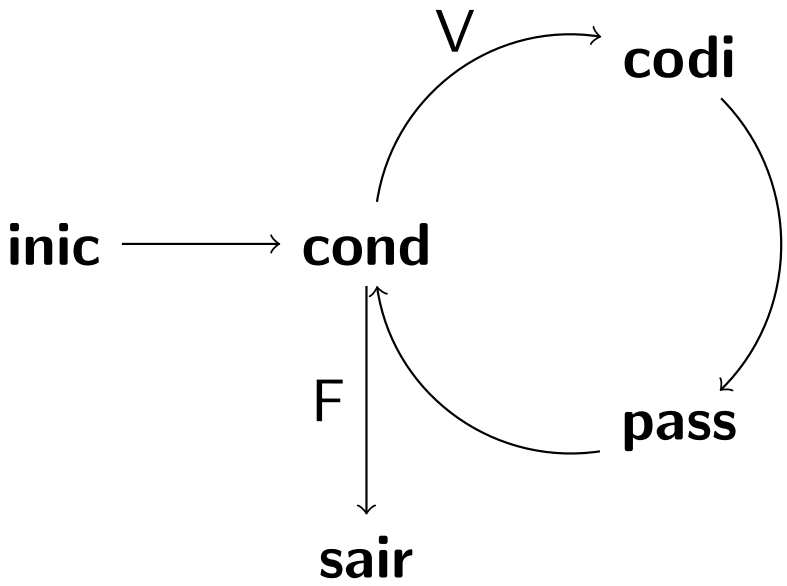
- Inicialização de variáveis:
  - na soma de inteiros, a variável soma deve ser inicializada com 0
  - no cálculo do fatorial, a variável fat deve ser inicializada com 1
- Inicialização em um loop
  - atribuição de valor inicial a uma variável auxiliar (iteradora)
  - ex: i=1
  - obs: não é obrigatório termos uma variável auxiliar
    - em alguns casos podemos trabalhar sem uma variável auxiliar!

Loops

- Condição em um loop:
  - também chamada de condição de parada, é uma condição que deve ser satisfeita para que o loop continue a ser executado
  - por ser uma condição, deve ser uma expressão que retorne verdadeiro ou falso
  - ex: i<=n
  - obs: se a condição for inserida incorretamente, pode-se nunca entrar no loop, ou resultar em loop infinito!
- Passo de um loop:
  - indica a taxa em que a variável auxiliar (iteradora) é aumentada a cada passo do algoritmo
  - ex: i++
  - obs: sem passo, vira loop infinito!

Loops

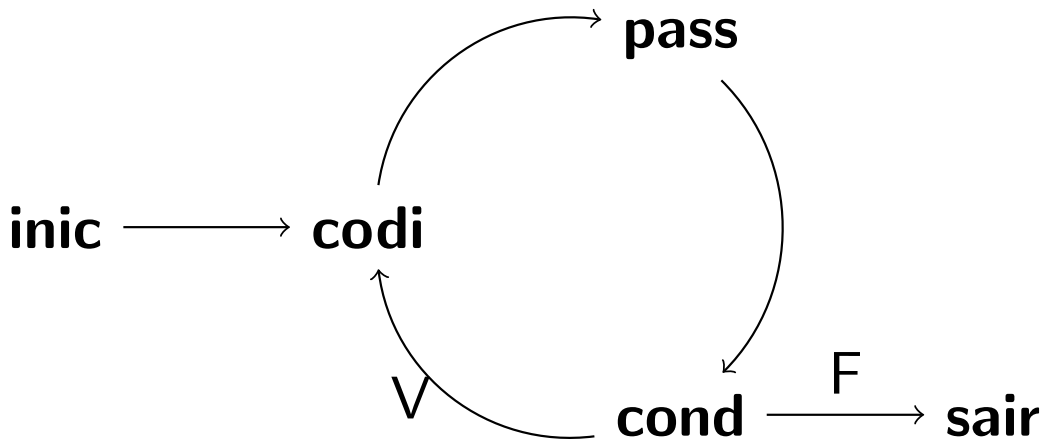
- Esquema da execução de um loop **for** ou **while**:



- Código só é executado após verificar a condição
- Inicialização (se houver) acontece antes de verificar a condição
- Se a condição é satisfeita, executamos o código do loop e fazemos o passo
- Se a condição não é satisfeita, saímos do loop

Loops

- Esquema da execução de um loop **do-while**:



- Código e passos são executados antes de se verificar a condição pela primeira vez
- Inicialização (se houver) acontece antes do código
- Se a condição for satisfeita, executamos o código do loop e fazemos o passo
- Se a condição não é satisfeita, saímos do loop
  - obs: executa o código pelo menos uma vez!

Loops

- O que é uma variável auxiliar iteradora?
  - é aquela nossa variável *i*!
  - é uma variável que nos auxilia a trabalhar num loop
  - nem sempre é obrigatória, mas como o próprio nome indica, vem para nos ajudar!
- Como utilizar a variável iteradora?
- Resposta: depende do problema!
- Exemplo: somar todos os inteiros de *a* até *b*
  - podemos fazer o *a* aumentar até chegar em *b*, ou *b* diminuir até chegar em *a*
  - porém com a variável auxiliar, não mexemos com *a* e *b*
  - em vez disso, fazemos a variável *i* ir de *a* até *b*

Loops

- Como fazer uma variável auxiliar *i* ir de *a* até *b*?
- Utilizando **for**:

```
for(i=a;i<=b;i++) {
    [código]
}
```

  - obs: não é legal alterar o valor do *i* dentro do código em um **for**
  - o passo já faz esta alteração, portanto é bom evitar!

- Utilizando **while**:

```
i=a;
while(i<=b) {
    [código]
    i++;
}
```

  - no **while**, o passo acontece em algum ponto dentro do código do loop
  - o passo não aparece necessariamente na última linha!

Loops

- As condições não se restringem a coisas do tipo i<=b
- Também podemos ter outros tipos de condições em loops
  - nestes casos, também pode haver uma outra ideia para o passo!
- Exemplos:
  - Soma de vários inteiros:

```
scanf("%d",&n);
while (n!=0) {
    [código]
    scanf("%d",&n);
}
```

  - Conta dígitos:

```
scanf("%d",&n);
while (n!=0) {
    [código]
    n=n/10;
}
```

Loops

- Apesar de não termos visto nenhum caso, é importante dizer que a condição nem sempre é apenas uma comparação
- Em alguns casos, ela também pode ser dada por um conectivo binário!
- Exemplo:
  - vamos ler até *n* valores inteiros e executar um código sobre cada um deles, mas queremos interromper o loop se o valor lido for igual a 0

```
i=1;
while (i<=n && x!=0) {
    scanf("%d",&x);
    [código]
    i++;
}
```

Vetores

- Em alguns dos exercícios vistos, nos deparamos com casos de ler vários valores
- Um dos problemas encontrados ao fazer isto, é que não podíamos armazenar todos estes valores
  - isto não era possível pois uma variável pode armazenar apenas um valor
  - também não podíamos criar várias variáveis, pois não sabemos inicialmente quantas devem ser criadas.
  - ou seja, códigos como o do slide anterior não armazenam vários valores, mas sim escrevem o novo valor na variável ignorando o valor anterior
- Esta restrição de não podermos armazenar nos limitava um pouco
  - tínhamos que trabalhar imediatamente com o valor dado, já que em seguida ele seria perdido
- Hoje, veremos uma maneira de armazenar vários valores para um uso posterior!

Vetores

- Suponha um problema como o "Maior e menor"
  - Dada uma sequência de n números, determinar quem são o maior e o menor elementos da sequência
- Sabemos que vamos ler n elementos
- Como já comentamos, não podemos criar n variáveis inteiras
- Porém, podemos criar uma estrutura que armazene n valores inteiros!
- A este tipo de estrutura, damos o nome de **vetor**

Vetores

- Como funciona um vetor?

- Um vetor segue a ideia de uma sequência de variáveis de um mesmo tipo
  - por exemplo, uma sequência de inteiros, ou de caracteres
- Ao criarmos um vetor, definimos o seu tamanho, ou seja, a quantidade de variáveis desta sequência
  - este tamanho é fixo e não pode ser alterado!
- Após criarmos o vetor, podemos preencher cada uma das suas posições como se fosse uma variável

Vetores

- Exemplo de funcionamento de um vetor:

- Primeiro, lemos a quantidade n de elementos
- Como já sabemos quantos valores inteiros vamos ler, podemos criar um vetor de n posições para armazenar estes valores
- Uma vez criado o vetor, começamos a preenchê-lo com os valores inseridos
- Após inserirmos todos os valores, temos um vetor totalmente preenchido, e podemos trabalhar com estes valores armazenados!

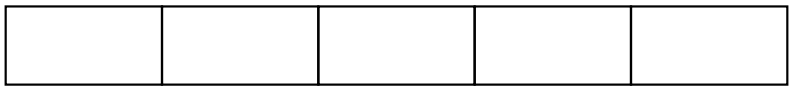
Vetores

- Exemplo de funcionamento de um vetor: problema "Maior e menor"
- Considere que lemos $n = 5$

Vetores

- Exemplo de funcionamento de um vetor: problema "Maior e menor"

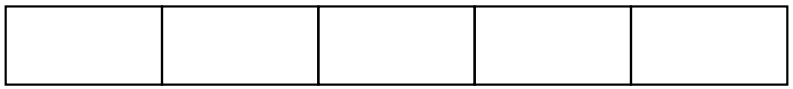
- Considere que lemos $n = 5$
- Criamos um vetor de $n = 5$ posições



Vetores

- Exemplo de funcionamento de um vetor: problema "Maior e menor"

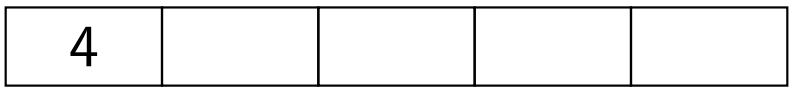
- Considere que lemos $n = 5$
- Criamos um vetor de $n = 5$ posições
- Lemos os valores inseridos



Vetores

- Exemplo de funcionamento de um vetor: problema "Maior e menor"

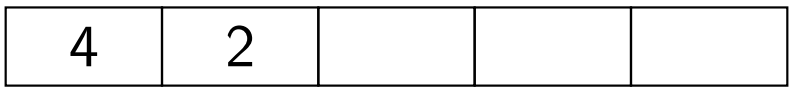
- Considere que lemos $n = 5$
- Criamos um vetor de $n = 5$ posições
- Lemos os valores inseridos
  - suponha que os valores inseridos são, nesta ordem, 4



Vetores

- Exemplo de funcionamento de um vetor: problema "Maior e menor"

- Considere que lemos $n = 5$
- Criamos um vetor de $n = 5$ posições
- Lemos os valores inseridos
  - suponha que os valores inseridos são, nesta ordem, 4, 2



Vetores

- Exemplo de funcionamento de um vetor: problema "Maior e menor"

- Considere que lemos $n = 5$
- Criamos um vetor de $n = 5$ posições
- Lemos os valores inseridos
  - suponha que os valores inseridos são, nesta ordem, 4, 2, 5

4	2	5		

Vetores

- Exemplo de funcionamento de um vetor: problema "Maior e menor"

- Considere que lemos $n = 5$
- Criamos um vetor de $n = 5$ posições
- Lemos os valores inseridos
  - suponha que os valores inseridos são, nesta ordem, 4, 2, 5, -3

4	2	5	-3	

Vetores

- Exemplo de funcionamento de um vetor: problema "Maior e menor"

- Considere que lemos $n = 5$
- Criamos um vetor de $n = 5$ posições
- Lemos os valores inseridos
  - suponha que os valores inseridos são, nesta ordem, 4, 2, 5, -3, 7

4	2	5	-3	7

Vetores

- Exemplo de funcionamento de um vetor: problema "Maior e menor"

- Considere que lemos $n = 5$
- Criamos um vetor de $n = 5$ posições
- Lemos os valores inseridos
  - suponha que os valores inseridos são, nesta ordem, 4, 2, 5, -3, 7
- Após isso, temos um vetor preenchido e podemos trabalhar com ele!

4	2	5	-3	7

Vetores

- Como criamos um vetor?

- Devemos informar o tipo de variável e a quantidade de posições
- Pseudocódigo:
  - ex: inteiro V[n]
  - ex: real S[7]
  - podemos criar um vetor com uma quantidade fixa ou variável de posições, fica a nosso critério
  - obs: um vetor só pode conter uma quantidade **inteira** de posições!
    - ou seja, o valor *n* acima não pode ser de outro tipo senão inteiro!
- C:
  - ex: **int** V[n];
  - ex: **float** S[7];
  - os comentários feitos acima também valem para C
- Importante: se vamos criar um vetor com *n* posições, devemos primeiro ler (ou inicializar) o valor de *n*

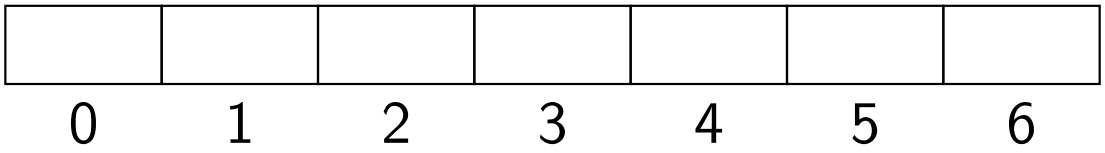
Vetores

- Como acessamos uma posição específica de um vetor?

- Suponha que desejamos acessar uma posição *i*, ou uma posição 3, do vetor
- Pseudocódigo:
  - para acessar o valor da posição *i* do vetor V, fazemos x=V[i]
  - para atribuir o valor 10 à posição 3 do vetor S, fazemos S[3]=10
- C:
  - para acessar o valor da posição *i* do vetor V, fazemos x=V[i];
  - para atribuir o valor 10 à posição 3 do vetor S, fazemos S[3]=10;
    - sim, é igual ao pseudocódigo, só que com ponto e vírgula :)
- Cada uma das posições de um vetor pode ser indicada por um índice
  - um índice é um valor que indica uma posição
  - ex: os valores *i* e 3 acima são índices

Vetores

- Importante: os índices das posições de um vetor de n posições não são $1, 2, \ldots, n$
- Um vetor de n posições trabalhará com índices $0, 1, \ldots, n-1$
  - por exemplo, se temos um vetor com 5 posições, os índices deste vetor variam de 0 a 4
- A imagem abaixo ilustra um exemplo de vetor com 7 posições
  - os números abaixo de cada posição representam os índices



Vetores

- Como preenchemos um vetor?

- Considere um vetor V de inteiros com *n* posições
- Para preencher este vetor por completo, temos que ler *n* valores
- Para isto, precisaremos de um loop!
- O loop **for** tem uma bela sintaxe para ler um vetor!
- Lembrando: as posições vão de 0 a n-1, logo devemos fazer nosso loop cobrir apenas estes índices!

Vetores

- Como preenchemos um vetor?

- Pseudocódigo:

  para *i* **de** 0 **a** n-1 **passo** 1 **faça**
  leia V[i]
  - ao final deste loop, nosso vetor V estará totalmente preenchido

- C:

```
for(i=0;i<n;i++) {
    scanf("%d",&V[i]);
}
```

  - ao final deste loop, nosso vetor V estará totalmente preenchido
  - obs: se o vetor for de algum outro tipo de variável que não seja inteiro, basta trocar o "%d"

Vetores

- Como preenchemos um vetor?

- Em C, também podemos preencher um vetor no momento da sua criação
- Por exemplo, suponha que criamos um vetor de 5 posições
- Para inserirmos os valores 1,2,3,4,5, podemos fazer:

  int V[5] = {1,2,3,4,5};

  - os elementos devem estar entre chaves e separados por vírgulas
  - obs: se tentarmos inserir mais elementos do que a capacidade do vetor, os elementos adicionais serão descartados
    - ou seja, se tentarmos inserir {1,2,3,4,5,6,7} num vetor de 5 posições, somente os valores {1,2,3,4,5} (os 5 primeiros) serão utilizados

Vetores

- Como preenchemos um vetor?

- Também podemos inicializar apenas parte do vetor
- Para inserirmos apenas três valores, podemos fazer:

  int V[5] = {1,2,3};
  - obs: as posições que não tiverem valor serão inicializadas com 0
  - ou seja, nosso vetor será [1,2,3,0,0]
- Também podemos inicializar o vetor com zeros
- Para isso, fazemos:

  int V[5]={0};

  ou simplesmente

  int V[5]={};
- Problema: não podemos inicializar um vetor de tamanho variável desta forma :(
  - podemos fazer V[5]={1,2,3,4,5}, mas não podemos fazer V[n]={1,2,3,4,5} mesmo se n for igual a 5
  - temos de utilizar um loop de 0 a n-1 para preenchê-lo

Vetores: exemplo

- Monte um algoritmo que verifica se um dado valor pertence a uma sequência

- Seu algoritmo deve receber:
  - a quantidade n de valores a serem lidos
  - os n valores da sequência
  - o valor x a ser buscado dentre os valores da sequência
- Seu algoritmo deve retornar:
  - "sim", se x pertence à sequência
  - "nao", se x não pertence à sequência

- Variação: altere seu método para contar a quantidade de vezes que x aparece na sequência