

# Fundamentos de Programação

Filipe Francisco

22 de março de 2018

Alguns problemas nos algoritmos esta semana:

- Lendo caracteres:
  - `"scanf("%d%d%c",&a,&b,&op);`
  - problema: separador (espaço) entre `%d` e `%c`!
    - se você não insere um espaço entre eles, o `scanf` lerá o caractere logo em seguida!
    - ou seja, o `scanf` lerá um `enter` (`'\n'`), ou `espaço` (`'\0'`)
  - este espaço representa uma separação entre os termos
  - não era importante para inteiros, mas é para caracteres!
- Imprimindo resultado de uma operação:
  - `"printf("%d",a/b);"`
  - o computador primeiro efetua a divisão, e só então efetua a divisão
  - portanto, neste caso também temos que verificar (**antes** do `printf`) se  $b \neq 0$ !

Alguns problemas nos algoritmos esta semana:

- Trabalhando com caracteres:
  - "if(var=='+') {...}"
  - utilizar aspas simples!
- Operadores:
  - operador de igualdade: ==
    - um = é atribuição!
  - operador AND: &&
    - um & é o AND bit a bit!
  - operador OR: ||
    - um | é o OR bit a bit!

- Condição a ser verificada em condicionais:
  - utilizar parênteses quando houver vários operadores binários!
  - indica a prioridade (precedência) caso hajam operadores diferentes!
  - obs: é opcional caso os conectivos sejam todos iguais
    - ideal: utilizar um par de parênteses para cada conectivo binário
    - ex: "if(a==b && a==c) {...}"
    - ex: "if((a==b && a!=c) || (a!=b && a==c)) {...}"
    - ex: "if(a==b && a==c && a==d) {...}"
- Indentação de código!
  - maneira de ressaltar a estrutura do código
  - fácil identificação do aninhamento das estruturas
    - aninhamento: uma estrutura dentro de outra
    - ex: um if dentro de outro if
  - facilita a leitura (e a escrita também)
  - facilita localizar erros (especialmente falta de chaves - { } )

- Código não indentado:



```
1  #include<stdio.h>
2  int main() {
3      int a,b;
4      char c;
5
6      scanf("%d %d %c",&a,&b,&c);
7
8      switch(c) {
9          case '+': printf("%d",a+b); break;
10         case '-': printf("%d",a-b); break;
11         case '*': printf("%d",a*b); break;
12         case '/': if(b==0) printf("invalida");
13         else printf("%d",a/b); break;
14         default: printf("invalida");
15     }
16
17     return 0;
18 }
```

- Código indentado:

```
1  #include<stdio.h>
2  int main() {
3      int a,b;
4      char c;
5
6      scanf("%d %d %c",&a,&b,&c);
7
8      switch(c) {
9          case '+': printf("%d",a+b); break;
10         case '-': printf("%d",a-b); break;
11         case '*': printf("%d",a*b); break;
12         case '/': if(b==0) printf("invalida");
13                    else printf("%d",a/b); break;
14         default: printf("invalida");
15     }
16
17     return 0;
18 }
```

- Mas você provavelmente não vai precisar fazer isso...

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] == $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```







- Até aqui, só trabalhamos com variáveis, operações e condicionais
- Até aqui, só trabalhamos com quantidades fixas
  - ex: receber **três** valores inteiros
  - ex: calcular **uma** operação entre dois números
  - ex: imprimir **um único** resultado
- Já vimos que podemos fazer muita coisa, mas ainda temos limitações
  - o fatorial de um número é dado por uma sequência de multiplicações
  - como calcular o fatorial com os conhecimentos vistos até agora?
- Hoje, veremos um novo tipo de estrutura para resolver esse problema
- Nossa ideia é de repetir passos, portanto precisamos de algo que nos permita fazer isso
- Para isso, temos as estruturas de repetição!

- Estruturas de repetição!
- Também chamadas de laços ou loops
- São utilizados para executar repetidamente um bloco de comandos a partir de uma certa condição
  - a condição nos lembra do if!
- Existem três estruturas principais de repetição:
  - para (**for**)
  - enquanto (**while**)
  - repita (**do-while**)

# Repetição

- Cada estrutura de repetição segue um mesmo esquema, mas de maneiras diferentes
  - cada um tem seu próprio cabeçalho
- Estas estruturas tem basicamente três componentes:
  - inicialização: valor inicial, a partir de que valor o loop deve executar
    - normalmente utiliza-se uma variável auxiliar
  - condição: é verificada para decidir se executamos uma iteração do loop
    - iteração: cada vez que os passos do loop são executados
  - passo (ou iteração): indica "em que velocidade" o valor inicial vai crescendo, ou a que medida o loop vai caminhando, a cada passo do loop
    - sem passo, seu algoritmo entra em loop infinito!
- A partir do valor inicial, o loop for é executado várias vezes, até que a condição não seja mais satisfeita
- A cada iteração, alteramos o valor inicial utilizando o passo

# Repetição: for/para

- Seu cabeçalho engloba inicialização, condição e passo
- Esquema básico de um loop for:

**para** *i* **de** *val1* **até** *val2* **passo** *p* **faça** { [código] }

- *i* é uma variável auxiliar do loop
  - chamamos *i* de variável iteradora
- *val1* é o valor inicial
- *val2* é a condição
  - é verificada antes mesmo da primeira iteração!
  - pode ser um valor numérico (*i* vai até este valor)
  - pode ser um booleano (executa até o booleano ficar falso)
- *p* é o passo
  - o valor de *i* é aumentado em *p* a cada iteração do loop
- Pergunta: como calcular o fatorial de um número utilizando for?

# Repetição: for/para

---

## Algoritmo: Fatorial

---

```
1 inicio  
2   inteiro n, i, fat=1  
3   leia n  
4   para i de 1 até n passo 1 faça  
5     |   fat = fat * i  
6   fim  
7   imprima fat  
8 fim
```

---

Podemos fazer multiplicações de 1 até *n* (crescente)!

# Repetição: for/para

---

## Algoritmo: Fatorial

---

```
1 inicio  
2   inteiro n, i, fat=1  
3   leia n  
4   para i de n até 1 passo -1 faça  
5     |    $fat = fat * i$   
6   fim  
7   imprima fat  
8 fim
```

---

Podemos fazer multiplicações de n até 1 (decrescente)!

---

## Algoritmo: Fatorial

---

```
1 inicio  
2   inteiro n, i, fat=1  
3   leia n  
4   para i de n até 2 passo -1 faça  
5     |    $fat = fat * i$   
6   fim  
7   imprima fat  
8 fim
```

---

Podemos fazer multiplicações de n até 2 (decrecente e eliminando uma multiplicação por 1)!

# Repetição: while/enquanto

- Seu cabeçalho só contém a condição!
- Esquema básico de um loop while:

**enquanto** *cond* **faça** { [código] }

- *cond* é a condição a ser verificada
  - é verificada antes mesmo da primeira iteração!
- normalmente, a condição é uma expressão
  - ex:  $i \leq n$
  - ex:  $a < n \ \&\& \ b < n$
- porém, também pode ser uma variável booleana (ou no caso do C, um inteiro)
  - lembrando: 0 é falso, e qualquer outro valor é tratado como verdadeiro!
  - boa prática: trabalhar com 0 e 1!
- Onde ficam inicialização e passo no while?



# Repetição: while/enquanto

- No while, a inicialização deve vir antes do loop!
  - antes do loop, devemos atribuir um valor à variável iteradora
- No while, o passo deve vir dentro do loop
  - devemos alterar o valor da variável iteradora dentro do loop
    - lembrando: se não alteramos, podemos ter loop infinito!
  - em boa parte das vezes, o passo estará na última linha do loop
    - mas saibam que isso não acontece sempre, ok!?
- Pergunta: como calcular o fatorial de um número utilizando while?

# Repetição: while/enquanto

---

## Algoritmo: Fatorial

---

```
1 inicio  
2   inteiro n, i, fat=1  
3   leia n  
4    $i = 1$   
5   enquanto  $i \leq n$  faça  
6      $fat = fat * i$   
7      $i = i + 1$   
8   fim  
9   imprima fat  
0 fim
```

---

Podemos fazer multiplicações de 1 até  $n$  (crescente)!

# Repetição: while/enquanto

---

## Algoritmo: Fatorial

---

```
1 inicio  
2   inteiro n, i, fat=1  
3   leia n  
4    $i = n$   
5   enquanto  $i \geq 1$  faça  
6      $fat = fat * i$   
7      $i = i - 1$   
8   fim  
9   imprima fat  
0 fim
```

---

Podemos fazer multiplicações de  $n$  até 1 (decrescente)!

# Repetição: while/enquanto

---

## Algoritmo: Fatorial

---

```
1 inicio  
2   inteiro n, i, fat=1  
3   leia n  
4    $i = n$   
5   enquanto  $i > 1$  faça  
6      $fat = fat * i$   
7      $i = i - 1$   
8   fim  
9   imprima fat  
0 fim
```

---

Podemos fazer multiplicações de  $n$  até 2 (decrecente e eliminando uma multiplicação por 1)!

# Repetição: do-while/repita

- Lembra o while em alguns pontos
- Seu cabeçalho também só contém a condição!
- Esquema básico de um loop do-while:

**repita** { [código] } **enquanto** cond

- *cond* é a condição a ser verificada
  - **IMPORTANTE**: só é verificada após executar a primeira iteração!
- Inicialização e passo são semelhantes ao while
  - inicialização: antes do loop
  - passo: dentro do loop
- Pergunta: como calcular o fatorial de um número utilizando do-while?

# Repetição: do-while/repita

---

## Algoritmo: Fatorial

---

```
1 inicio  
2   inteiro n, i, fat=1  
3   leia n  
4    $i = 1$   
5   repita  
6      $fat = fat * i$   
7      $i = i + 1$   
8   enquanto  $i \leq n$   
9   imprima fat  
0 fim
```

---

Podemos fazer multiplicações de 1 até  $n$  (crescente)!

# Repetição: do-while/repita

---

## Algoritmo: Fatorial

---

```
1 inicio  
2   inteiro n, i, fat=1  
3   leia n  
4    $i = n$   
5   repita  
6      $fat = fat * i$   
7      $i = i - 1$   
8   enquanto  $i \geq 1$   
9   imprima fat  
0 fim
```

---

Podemos fazer multiplicações de  $n$  até 1 (decrescente)!

# Repetição: do-while/repita

- Problema: o loop do-while executa o código uma vez antes de verificar a condição
- Pergunta: e se  $n = 0$ ?
  - sabemos que  $0! = 1$
- Note que todos os códigos até aqui resolvem este caso
- Porém, o segundo código para o do-while fará uma multiplicação de fat por 0, logo o resultado será zero!
- O loop do-while pode gerar problemas devido ao seu funcionamento
  - devido ao fato de executar o código uma vez antes de verificar a condição
- Por este motivo, acaba-se utilizando bem mais os loops for e while
- Apesar disso, saibam que ele ainda tem utilidade!



# Repetição: for/para em C

- Esquema do loop for em C:

**for**(ini; cond; pas) { [código] }

- ini: inicialização
- cond: condição
- pas: passo
- exemplo: **for**(i=1;i<=n;i++) { [código] }
  - lembrando: i é a variável iteradora!

# Repetição: while/enquanto em C

- Esquema do loop while em C:

**while**(cond) { [código] }

- cond: condição
  - relembrando: inicialização está antes do while, e passo está dentro do código!
- exemplo: **while**(i<=n) { [código] }

# Repetição: do-while/repita em C

- Esquema do loop do-while em C:

**do** { [código] } **while**(cond)

- cond: condição
  - lembrando: inicialização está antes do while, e passo está dentro do código!
- exemplo: **do** { [código] } **while**(i<=n)