Programação Estruturada

Prof. Rodrigo Hausen
http://progest.compscinet.org

Vetores (Arrays)

AULA PASSADA - CONDICIONAIS

Exercício 1: Crie a função vlrabs que recebe um double e retorna o seu valor absoluto.

$$|x| = \begin{cases} x \text{ , se } x \geq 0 \\ -x \text{ , se } x < 0 \end{cases}$$

```
double vlrabs(double x) {
   ____________
   ____________
}
```

AULA PASSADA - LAÇOS

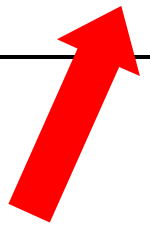
Repetição de código:

```
int i;


for (i = 1; i <= 7; ++i) {
  puts("All the single ladies!");
}
```

AULA PASSADA - LAÇOS

Repetição de código:

```
int i;


for (i = 1; i <= 7; ++i) {
  ____________________________________
}
```

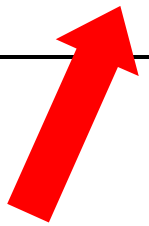


inicialização
*executada **uma vez** logo antes do início do laço*

AULA PASSADA - LAÇOS

Repetição de código:

```
int i;


for (i = 1; i <= 7; ++i) {
    ____________________________
}
```

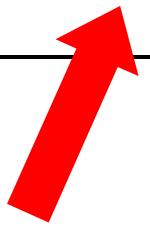


condição de continuação
*verificada **antes** de inciar **cada** iteração*
se a expressão lógica for verdadeira, executa a iteração

AULA PASSADA - LAÇOS

Repetição de código:

```
int i;


for (i = 1; i <= 7; ++i) {

}
```



atualização

*declaração executada **logo após cada** iteração*

LAÇOS DE ITERAÇÃO – **while** E **do...while**

Além do laço **for**, temos duas outras estruturas de iteração em C:

```
while (cond) {
   ______
   ______
   ______
}
```

expressão é verificada ***logo antes do início de cada iteração***

```
do {
   ______
   ______
   ______
} while (cond);
```

LAÇOS DE ITERAÇÃO – **while** E **do...while**

Além do laço **for**, temos duas outras estruturas de iteração em C:

```
while (cond) {
   ______
   ______
   ______
}
```

```
do {
   ______
   ______
   ______
} while (cond);
```

expressão é verificada
após cada iteração

LAÇOS DE ITERAÇÃO – **while** E **do...while**

Exemplo: algoritmo de euclides para o máximo divisor comum de inteiros positivos.

```
unsigned int mdc(unsigned int a,
                 unsigned int b) {
  while (b != 0) {
    int resto;
    resto = a % b;
    a = b;
    b = resto;
  }
  return a;
}
```

LAÇOS DE ITERAÇÃO – **while** E **do...while**

Exercício 2: o método de Newton é um algoritmo numérico que permite encontrar uma solução aproximada para a equação f(x) = 0, com erro de ε no máximo.

Passo 1: escolha x_0 próximo da raiz

Passo 2: faça $x_{n+1} = x_n - \frac{f(x)}{f'(x)}$

Passo 3: se $|x_{n+1} - x_n| < \varepsilon$ pare,
senão volte ao passo 2

Usando este método, crie uma função para calcular a raiz quadrada de um número.

LAÇOS DE ITERAÇÃO – **while** E **do...while**

Seja c o número e $f(x) = x^2 - c$. Logo, $f'(x) = 2x$

Passo 1: escolha $x_0 = c$

Passo 2: faça $x_{n+1} = \frac{x_n}{2} - \frac{c}{2x}$

Usaremos $\varepsilon = 10^{-10}$ no passo 3

LAÇOS DE ITERAÇÃO – **while** E **do...while**

Seja c o número e $f(x) = x^2 - c$. Logo, $f'(x) = 2x$

Passo 1: escolha $x_0 = c$

Passo 2: faça $x_{n+1} = \frac{x_n}{2} - \frac{c}{2x}$

Usaremos $\varepsilon = 10^{-10}$ no passo 3

Use a função vlrabs feita no exerc. 1.

VETORES ou ARRAYS

Declaração de um vetor (array) em C:

```
tipo nome[tamanho];
```

Exemplos:

```
int numeros[100];

double notas[40];

char nomeDoMes[10];
```

Obs.: padrão C99 permite definir vetores com tamanho dependente de variável

```
int n = 40;
double notas[n];
```

VETORES

Acesso a elemento:

- nomeDoArray[posicao]

Obs.: em C, a posição do primeiro elemento de um array é sempre **0** (zero)

Inicialização com atribuição:

- int numeros[] = { 7, 9, 5, 2, 1 };

VETORES

Exercício 3: implemente a função **somaint** que recebe um vetor de inteiros e seu comprimento *n* e retorna a soma dos números.

Exercício 4: implemente a função **prodEscalar** para calcular o produto escalar entre dois vetores de *n* elementos.

VETORES

Exercício 5: implemente a função **strlen** que recebe uma “string” (array de char) e retorna o seu comprimento.

- O último caractere de uma string é sempre o caractere nulo '\0'
- O caractere nulo não deve ser contado

Exercício 6: implemente a função **ePalindromo**, que recebe uma string e determina se ela é um palíndromo (idêntica quando lida ao contrário):

- “anilina” é palíndromo
- “amar” não é palíndromo

VETORES

Exemplo: ordenação por “borbulhamento” (*bubble sort*)

Ordena um vetor *v* de forma crescente.

> ***enquanto*** *existir par de elementos v[i] , v[i+1] tais que v[i] > v[i+1]*
>
> *troca v[i] com v[i+1]*

Menores elementos vão se deslocando lentamente (“borbulhando”) para o início do vetor.

Exercício 7: implementar bubble sort para vetor de int.

PARA CASA

0) implemente a função
void inverteStr(**char** s[], **int** i, **int** j)
que inverte a parte da string entre os caracteres s[i] e s[j] (inclusos)

1) implemente a função
void invertePalavras(**char** s[]) que inverte as letras das palavras de uma string, sem alterar a ordem das palavras

Ex.: resultado no **cling**

```
[cling]$ char str[] = "bom dia amigos";
[cling]$ invertePalavras(str);
[cling]$ str
  (char [15]) "mob aid sogima"
```

PARA CASA

2) implemente a função
void reversaoPalavras(**char** s[])
que inverte a ordem de palavras de uma string, mantendo a ordem de letras em cada palavra.

Ex.: resultado no **cling**

```
[cling]$ char str[] = "bom dia amigos";
[cling]$ reversaoPalavras(str);
[cling]$ str
  (char [15]) "amigos dia bom"
```

PARA CASA

3) um polinômio pode ser representado apenas pelos seus coeficientes. Por exemplo, o vetor abaixo:

```
double p = { 3.0, 5.2, 0, 2.1 };
```

pode ser usado para representar o polinômio na variável *x*:

$$2.1x^3 + 5.2x + 3.0$$

Implemente a função

```
void imprimePolinomio(double p[],
     int n, char varname[])
```

que imprime na tela o polinômio na variável *varname*.

PARA CASA

A função deve imprimir na forma simplificada, eliminando termos iguais a zero e constantes multiplicativas iguais a 1. Para testar no **cling**:

```
double p[] = {-0.5, 1.0, 0.0, 5.1, 0.0};
imprimePolinomio(p, 5, "x");
```

Imprimirá: 5.1 x^3 + x - 0.5

```
double q[] = {1.0, -1.0, 0.0, 0.0, 1.0};
imprimePolinomio(q, 5, "y");
```

Imprimirá: y^4 – y + 1.0

PARA CASA

```
double r[] = {-2.5, 0.0, -8.1};
imprimePolinomio(r, 3, "z");
```

Imprimirá: -8.1 z^2 - 2.5

```
double s[] = {0.0, 0.0, 0.0, 0.0};
imprimePolinomio(s, 4, "w");
```

Imprimirá: 0.0