

Programação Estruturada

Prof. Paulo Henrique Pisani

<http://professor.ufabc.edu.br/~paulo.pisani/>

setembro/2018

Tópicos

- Vetores
- Funções

Revisão

Conversão de tipo

- Conversão entre tipos usando **cast**:

```
#include <stdio.h>
int main() {
    float r1 = 7 / 2;
    float r2 = ((float) 7) / 2;
    float r3 = 7 / ((float) 2);
    float r4 = ((float) 7) / ((float) 2);
    float r5 = (float) 7 / 2;
    float r6 = (float) (7 / 2);

    printf("%.1f %.1f %.1f\n", r1, r2, r3);
    printf("%.1f %.1f %.1f\n", r4, r5, r6);

    return 0;
}
```

Saída

?

Conversão de tipo

- Conversão entre tipos usando **cast**:

```
#include <stdio.h>
int main() {
    float r1 = 7 / 2;
    float r2 = ((float) 7) / 2;
    float r3 = 7 / ((float) 2);
    float r4 = ((float) 7) / ((float) 2);
    float r5 = (float) 7 / 2;
    float r6 = (float) (7 / 2);

    printf("%.1f %.1f %.1f\n", r1, r2, r3);
    printf("%.1f %.1f %.1f\n", r4, r5, r6);

    return 0;
}
```

Type cast tem precedência sobre o operador de divisão.

Saída

```
3.0 3.5 3.5
3.5 3.5 3.0
```

Revisão

- Padrões:
 - A. Programa mínimo
 - B. Acumulador (e.g. soma)
 - C. Valor referência (e.g. menor, maior) - métodos de inicialização
 - D. Leitura de n elementos (n pré-definido)
 - E. Leitura de n elementos (n desconhecido)

Leitura de n elementos (n pré-definido)

```
#include <stdio.h>

int main() {

    int qtd;
    scanf("%d", &qtd);

    int i, numero;
    for (int i = 0; i < qtd; i++) {

        scanf("%d", &numero);

        // Faz alguma coisa
        printf("Leu numero!\n");
    }

    printf("Saiu!\n");

    return 0;
}
```

Leitura de n elementos (n desconhecido)

Lê apenas números
positivos e o zero

```
#include <stdio.h>

int main() {


    int numero;
    scanf("%d", &numero);
    while (numero >= 0) {

        scanf("%d", &numero);

        // Faz alguma coisa
        printf("Leu numero!\n");
    }

    printf("Saiu!\n");

    return 0;
}
```



Vetores

Vetores

- É um conjunto de variáveis do mesmo tipo:
 - Referenciada por um mesmo identificador;
 - Cada elemento é acessado por meio de um índice.
- **Exemplo:** ler a idade de 10 pessoas e contar quantas estão acima da idade média.
 - Uma alternativa seria criar 10 variáveis;
 - Outra (bem melhor), seria criar um vetor/array de comprimento 10.

Vetores

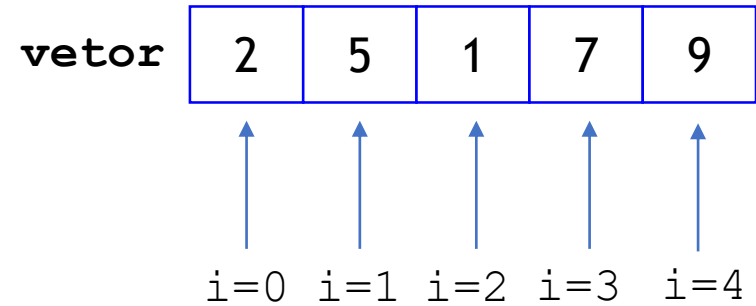
- Declarar vetor:

```
<tipo> <nome>[<tamanho>;
```

Exemplos

```
int idades[10];  
double vetor2[5];  
int valores[3] = {10, 20, 30};
```

Vetores



Índices começam no 0 (zero)

- Acessar valores em um vetor:

```
int vetor[5];  
vetor[4] Acessa elemento de índice 4  
vetor[3] = 7; Atribui valor no elemento de índice 3  
if (vetor[3] == 8) Acessa elemento de índice 3  
    return 0;  
vetor[5] Elemento não existe!
```

- Ler elementos em um vetor:

```
int idade[10];  
int i;  
for (i = 0; i < 10; i++)  
    scanf("%d", &idade[i]);
```

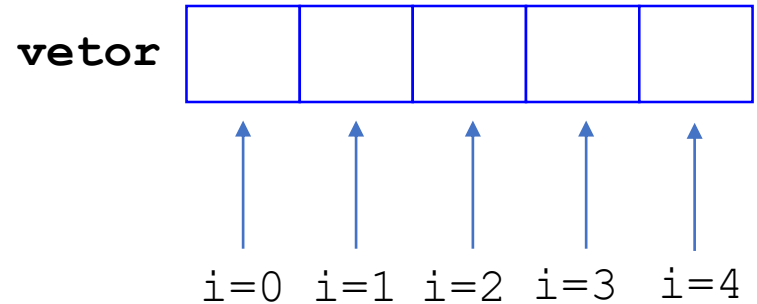
Vetores

- Percorrer um vetor:

```
int vetor[5];
int i;
for (i = 0; i < 5; i++) {
    vetor[i];
}
```

- O que faz este código?

```
int vetor[5];
int i;
for (i = 0; i < 5; i++) {
    vetor[i] = (i+1)*(i+1);
}
```



Exemplo

- Ler a idade de 10 pessoas e contar quantas estão acima da idade média.

```
#include<stdio.h>
```

```
int main() {
```

Declaração do vetor

```
int idade[10];
```

```
int i, soma=0;
```

```
for (i = 0; i < 10; i++) {
```

```
scanf("%d", &idade[i]);
```

```
soma += idade[i];
```

```
}
```

Importante: lembre-se de usar o "&" no scanf!

```
double media = soma / 10.0;
```

Necessário para não realizar divisão inteira.

```
int qtd_acima_media = 0;
```

```
for (i = 0; i < 10; i++)
```

```
if (idade[i] > media)
```

```
qtd_acima_media++;
```

```
printf("Ha %d pessoa(s) acima da media.", qtd_acima_media);
```

```
return 0;
```

```
}
```

```
#include<stdio.h>
```

```
int main() {
```

```
    int idade[10];  
    int i, soma=0;  
    for (i = 0; i < 10; i++) {  
        scanf("%d", &idade[i]);  
        soma += idade[i];  
    }
```

```
    double media = soma / 10.0;
```

```
    int qtd_acima_media = 0;  
    for (i = 0; i < 10; i++)  
        if (idade[i] > media)  
            qtd_acima_media++;
```

```
    printf("Ha %d pessoa(s) acima da media.", qtd_acima_media);
```

```
    return 0;
```

```
}
```

**Atenção com os
limites do vetor!**


```
#include<stdio.h>
```

```
int main() {
```

```
    int idade[10];  
    int i, soma=0;  
    for (i = 0; i <= 10; i++) {  
        scanf("%d", &idade[i]);  
        soma += idade[i];  
    }
```

```
    double media = soma / 10.0;
```

```
    int qtd_acima_media = 0;  
    for (i = 0; i < 20; i++)  
        if (idade[i] > media)  
            qtd_acima_media++;
```

```
    printf("Ha %d pessoa(s) acima da media.", qtd_acima_media);
```

```
    return 0;
```

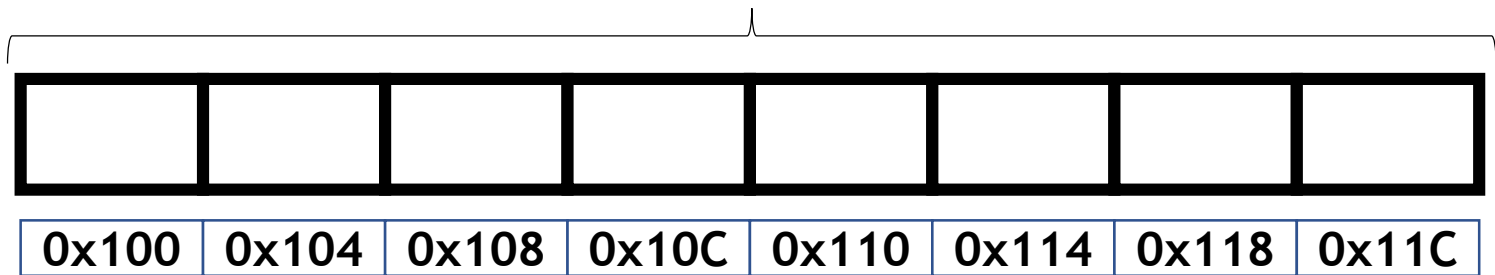
```
}
```

Comportamento
imprevisível!!!



Vetores são armazenados em posições consecutivas na memória!

```
int vetor[8];
```



Endereço de memória do primeiro elemento do vetor.

Observe que cada elemento tem 4 bytes (tamanho de um int = sizeof(int))

Este vetor ocupa $8 * 4 = 32$ bytes
Ou seja, tamanho * sizeof(<tipo>)

Tamanhos dos tipos de dados (sizeof)

- Para saber quantos bytes um tipo de dados ocupa, usamos **sizeof** (o retorno é do tipo **long int**).

```
#include<stdio.h>

int main() {
    int a;
    long int b;
    long long int c;
    float d;
    double e;
    char f;

    printf("%ld %ld %ld %ld %ld %ld",
           sizeof(a),
           sizeof(b),
           sizeof(c),
           sizeof(d),
           sizeof(e),
           sizeof(f));
    return 0;
}
```

Algoritmos importantes

- **Imprimir vetor:** cada elemento em uma linha, todos os elementos na mesma linha;
- **Inserção de elementos em vetor:** início, meio e final (lista estática);
- **Buscar elemento;**
- **Remoção de elementos de vetor:** início/meio e final (lista estática).

Imprimir vetor

```
#include <stdio.h>

int main() {

    int lista[5] = {20, 30, 50, 10, 40};

    printf("\n");
    int i;
    for (i = 0; i < 5; i++)
        printf("%d ", lista[i]);
    printf("\n");

    return 0;
}
```

Imprimir vetor

```
#include <stdio.h>

int main() {

    int lista[5] = {20, 30, 50, 10, 40};

    printf("\n");
    int i;
    for (i = 0; i < 5; i++)
        printf("%d ", lista[i]);
    printf("\n");

    return 0;
}
```

**E para imprimir na
ordem inversa?**

Imprimir vetor (ordem inversa)

```
#include <stdio.h>

int main() {

    int lista[5] = {20, 30, 50, 10, 40};

    printf("\n");
    int i;
    for (i = 4; i >= 0; i++)
        printf("%d ", lista[i]);
    printf("\n");

    return 0;
}
```

Inserir elemento no **final** de um vetor

```
#include <stdio.h>


int main() {

    int lista[5];
    int i;
    for (i = 0; i < 5; i++) {
        int numero;
        scanf("%d", &numero);
        lista[i] = numero;
    }

    printf("\n");
    for (i = 0; i < 5; i++)
        printf("%d ", lista[i]);
    printf("\n");

    return 0;
}
```

Usamos um índice para inserir no final



Inserir elemento no **final** de um vetor

```
#include <stdio.h>


int main() {

    int lista[5];
    int i = 0;
    while (i < 5) {
        int numero;
        scanf("%d", &numero);
        lista[i] = numero;
        i++;
    }

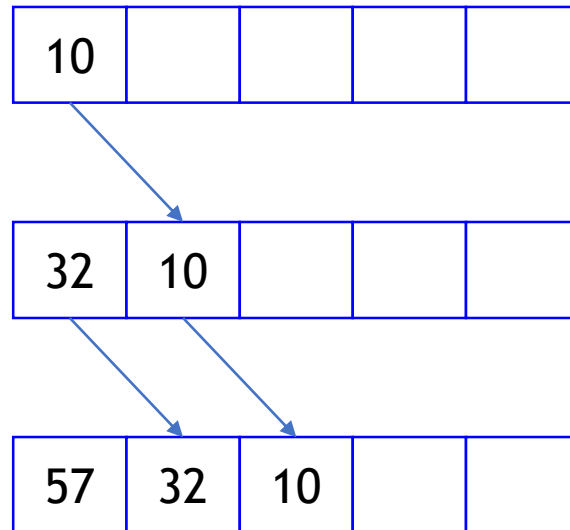
    printf("\n");
    for (i = 0; i < 5; i++)
        printf("%d ", lista[i]);
    printf("\n");

    return 0;
}
```

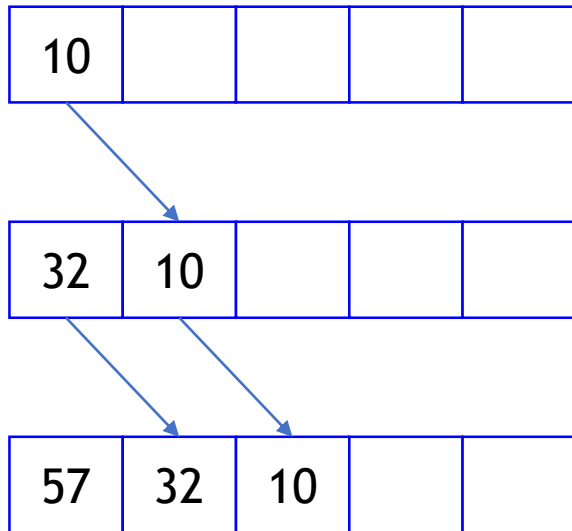
Usamos um índice para inserir no final



Inserir elemento no **início** de um vetor



Inserir elemento no início de um vetor



Deslocamento
dos elementos a
frente

```
#include <stdio.h>

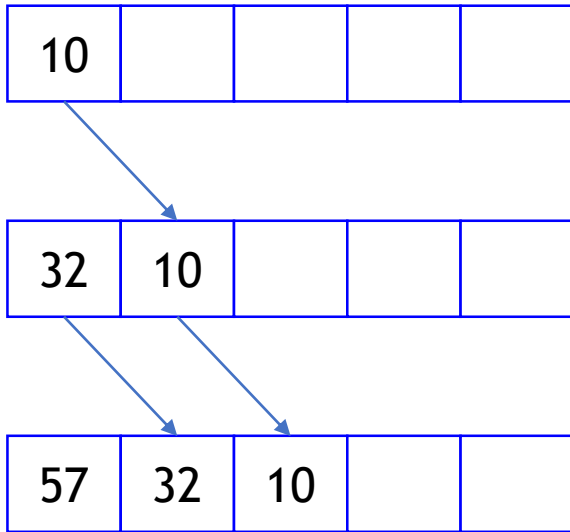
int main() {
    int lista[5];
    int i;
    for (i = 0; i < 5; i++) {
        int numero;
        scanf("%d", &numero);

        int k;
        for (k = 5-1; k > 0; k--)
            lista[k] = lista[k-1];
        lista[0] = numero;
    }

    printf("\n");
    for (i = 0; i < 5; i++)
        printf("%d ", lista[i]);
    printf("\n");

    return 0;
}
```

Inserir elemento no **início** de um vetor



Podemos melhorar esse programa?

```
#include <stdio.h>
```

```
int main() {  
    int lista[5];  
    int i;  
    for (i = 0; i < 5; i++) {  
        int numero;  
        scanf("%d", &numero);
```

Deslocamento
dos elementos a
frente

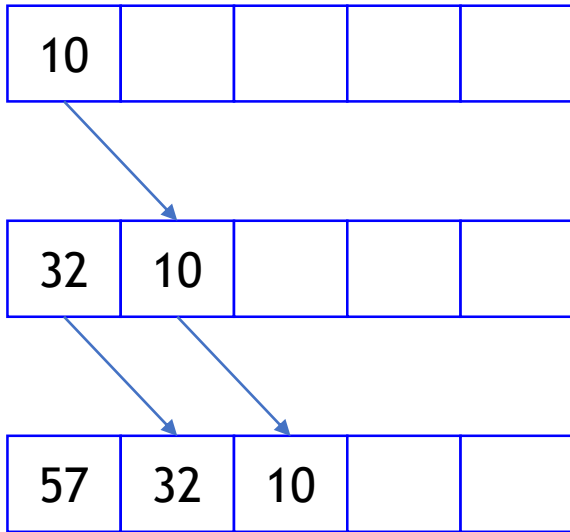
```
int k;  
for (k = 5-1; k > 0; k--)  
    lista[k] = lista[k-1];  
lista[0] = numero;
```

```
}  
  
printf("\n");  
for (i = 0; i < 5; i++)  
    printf("%d ", lista[i]);  
printf("\n");
```

```
return 0;
```

```
}
```

Inserir elemento no **início** de um vetor



```
#include <stdio.h>
```

```
int main() {  
    int lista[5];  
    int i;  
    for (i = 0; i < 5; i++) {  
        int numero;  
        scanf("%d", &numero);
```

Deslocamento
dos elementos a
frente

```
int k;  
for (k = i; k > 0; k--)  
    lista[k] = lista[k-1];  
lista[0] = numero;
```

```
}  
  
printf("\n");  
for (i = 0; i < 5; i++)  
    printf("%d ", lista[i]);  
printf("\n");
```

```
return 0;
```


```
}
```

**Podemos melhorar
esse programa?**

E se quisermos
trabalhar com um
vetor de 10
elementos?

```
#include <stdio.h>
```

```
int main() {  
    int lista[5];  
    int i;  
    for (i = 0; i < 5; i++) {  
        int numero;  
        scanf("%d", &numero);  
  
        int k;  
        for (k = i; k > 0; k--)  
            lista[k] = lista[k-1];  
        lista[0] = numero;  
    }  
  
    printf("\n");  
    for (i = 0; i < 5; i++)  
        printf("%d ", lista[i]);  
    printf("\n");  
  
    return 0;  
}
```




**E se quisermos
trabalhar com um
vetor de 10
elementos?**

Precisamos mudar o valor “5”
(que está “hardcoded”) em todo
o código! Há como evitar isso?

```
#include <stdio.h>
```

```
int main() {  
    int lista[5];  
    int i;  
    for (i = 0; i < 5; i++) {  
        int numero;  
        scanf("%d", &numero);  
  
        int k;  
        for (k = i; k > 0; k--)  
            lista[k] = lista[k-1];  
        lista[0] = numero;  
    }  
  
    printf("\n");  
    for (i = 0; i < 5; i++)  
        printf("%d ", lista[i]);  
    printf("\n");  
  
    return 0;  
}
```



Buscar elemento



```
#include <stdio.h>

int main() {

    int lista[5] = {20, 30, 50, 10, 40};

    int numero;
    scanf("%d", &numero);

    int i, encontrou=0;
    for (i = 0; i < 5; i++) {
        if (lista[i] == numero)
            encontrou = 1;
    }

    if (encontrou)
        printf("Encontrou!\n");
    else
        printf("NAO Encontrou!\n");

    return 0;
}
```


Buscar elemento



```
#include <stdio.h>

int main() {

    int lista[5] = {20, 30, 50, 10, 40};

    int numero;
    scanf("%d", &numero);

    int i, encontrou=0;
    for (i = 0; i < 5; i++) {
        if (lista[i] == numero)
            encontrou = 1;
    }

    if (encontrou)
        printf("Encontrou!\n");
    else
        printf("NAO Encontrou!\n");

    return 0;
}
```

Podemos melhorar
esse programa?

Buscar elemento

```
int i, encontrado=0;
for (i = 0; i < 5; i++) {
    if (lista[i] == numero) {
        encontrado = 1;
        break;
    }
}
```

```
int i, encontrado=0;
for (i = 0; i < 5; i++) {
    if (lista[i] == numero)
        encontrado = 1;
}
```

```
int i, encontrado=0;
for (i = 0; i < 5 && !encontrado; i++) {
    if (lista[i] == numero)
        encontrado = 1;
}
```

Funções

Funções

- Usadas para **modularizar** o código;
- Uma função executa uma operação e retorna um valor;
- Estrutura de uma função em C:

```
tipo_retorno nome_funcao(Lista de parâmetros) {  
    ...  
}
```

Funções

Estilo de código: nomes de funções são escritos em letras minúsculas e cada palavra é separado por “_”.

- Exemplo

```
int calcula_quadrado(int num) {  
    return num * num;  
}
```

Usado para retornar o valor da função.

Em C, apenas um valor pode ser retornado.

Chamando uma função

```
#include<stdio.h>
```

```
int calcula_quadrado(int num) {  
    return num * num;  
}
```

```
int main() {
```

```
    int n1;  
    scanf("%d", &n1);
```

```
    int quad = calcula_quadrado(n1);
```

```
    printf("%d\n", quad);
```


```
    return 0;
```

```
}
```



Procedimentos

- Um procedimento é uma função que não retorna um valor;
- Usamos **void** para indicar isso.



```
void mostra_quadrado(int num) {  
    printf("%d\n", num * num);  
}
```

Declaração de funções

- Declaramos as funções antes de suas chamadas:
 - Por exemplo, o código da função `calcula_quadrado` está antes da função `main`.

```
#include<stdio.h>

int calcula_quadrado(int num) {
    return num * num;
}

int main() {
    int n1;
    scanf("%d", &n1);
    int quad = calcula_quadrado(n1);
    printf("%d\n", quad);
    return 0;
}
```


Declaração de funções

- Mas podemos declarar as funções em qualquer ordem se declararmos seus **protótipos** no início:

```
#include<stdio.h>
```

```
int calcula_quadrado(int num);
```



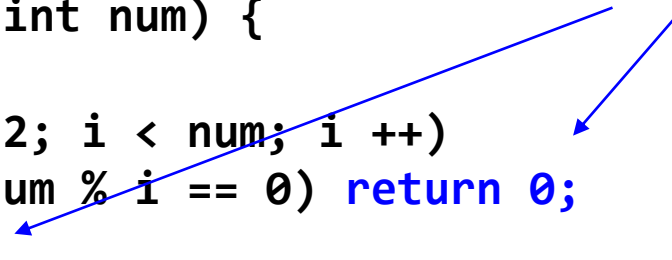
```
int main() {  
    int n1;  
    scanf("%d", &n1);  
    int quad = calcula_quadrado(n1);  
    printf("%d\n", quad);  
    return 0;  
}
```

```
int calcula_quadrado(int num) {  
    return num * num;  
}
```

```
#include<stdio.h>
```

- **return** interrompe a função e retorna o valor.

```
int eh_primo(int num) {  
    int i;  
    for (i = 2; i < num; i ++)  
        if (num % i == 0) return 0;  
    return 1;  
}  
  
int main() {  
  
    int numero;  
    scanf("%d", &numero);  
  
    if (eh_primo(numero))  
        printf("%d eh primo.\n", numero);  
    else  
        printf("%d NAO eh primo.\n", numero);  
  
    return 0;  
}
```



```
#include<stdio.h>
```

```
int eh_primo(int num) {  
    int i;  
    for (i = 2; i < num; i++)  
        if (num % i == 0) return 0;  
    return 1;  
}
```

```
int main() {  
  
    int numero;  
    scanf("%d", &numero);  
  
    if (eh_primo(numero))  
        printf("%d eh primo.\n", numero);  
    else  
        printf("%d NAO eh primo.\n", numero);  
  
    return 0;  
}
```

Podemos otimizar esse algoritmo de verificação de primo?

Otimizando a verificação de primo

```
int eh_primo(int num) {  
    if (num == 2) return 1;  
    if (num % 2 == 0) return 0;  
    int i;  
    for (i = 3; i <= sqrt(num); i += 2)  
        if (num % i == 0) return 0;  
    return 1;  
}
```



Requer:
#include <math.h>

Passagem de parâmetros

- Em C, todo parâmetro de função é passado **por valor**;
- Para passar um argumento **por referência**, precisamos passar o valor do endereço de memória (ponteiro) - veremos isso em outras aulas...

Parâmetros e argumentos são a mesma coisa?

Escopo de variáveis

- O escopo de uma variável determina de onde ela pode ser acessada;
- Em C, existem dois escopos principais:
 - Variáveis **locais**: acessíveis apenas localmente (pela função);
 - Variáveis **globais**: acessíveis a partir de qualquer parte do código.

```
#include<stdio.h>
```

```
double resultado; ←
```

```
void calcula_quadrado(double num) {  
    resultado = num * num;  
}
```

```
double calcula_soma(double n1, double n2) {  
    double r; ←  
    r = n1 + n2;  
    return r;  
}
```

```
int main() {  
    int a = 2, b = 3; ←  
    resultado = calcula_soma(a, b);  
    printf("%.2lf\n", resultado);  
    calcula_quadrado(resultado);  
    printf("%.2lf\n", resultado);  
    calcula_quadrado(resultado);  
    printf("%.2lf\n", resultado);  
  
    return 0;  
}
```

Variável global

A variável resultado pode ser acessada a partir de qualquer função!

Variável local da função
calcula_soma

Variáveis locais da
função main

Variáveis locais são acessíveis apenas da função onde foram declaradas.

```
#include<stdio.h>
```

```
double resultado;
```

```
void calcula_quadrado(double num) {  
    resultado = num * num;  
}
```

```
double calcula_soma(double n1, double n2) {  
    double r;  
    r = n1 + n2;  
    return r;  
}
```

```
int main() {  
    int a = 2, b = 3;  
    resultado = calcula_soma(a, b);  
    printf("%.2lf\n", resultado);  
    calcula_quadrado(resultado);  
    printf("%.2lf\n", resultado);  
    calcula_quadrado(resultado);  
    printf("%.2lf\n", resultado);  
  
    return 0;  
}
```

O que será impresso?


```
#include<stdio.h>
```

```
double resultado;
```

```
void calcula_quadrado(double num) {  
    resultado = num * num;  
}
```

```
double calcula_soma(double n1, double n2) {  
    double r;  
    r = n1 + n2;  
    return r;  
}
```

```
int main() {  
    int a = 2, b = 3;  
    resultado = calcula_soma(a, b);  
    printf("%.2lf\n", resultado);  
    calcula_quadrado(resultado);  
    printf("%.2lf\n", resultado);  
    calcula_quadrado(resultado);  
    printf("%.2lf\n", resultado);  
  
    return 0;  
}
```

O que será impresso?

```
5.00  
25.00  
625.00
```

Veja que o uso de variáveis globais dificulta a leitura do código (e pode levar a erros de programação).

Portanto, evite variáveis globais ao máximo!

```
#include<stdio.h>
```

```
double g_resultado;
```

```
void calcula_quadrado(double num) {  
    g_resultado = num * num;  
}
```

```
double calcula_soma(double n1, double n2) {  
    double r;  
    r = n1 + n2;  
    return r;  
}
```

```
int main() {  
    int a = 2, b = 3;  
    g_resultado = calcula_soma(a, b);  
    printf("%.2lf\n", g_resultado);  
    calcula_quadrado(g_resultado);  
    printf("%.2lf\n", g_resultado);  
    calcula_quadrado(g_resultado);  
    printf("%.2lf\n", g_resultado);  
  
    return 0;  
}
```

Quando declarar uma variável global, use o prefixo "g_"

Escopo de variáveis

```
#include<stdio.h>

void funcao_teste(int param1) {
    int a = param1;
    if (param1 > 0) {
        int b = 0;
        int i;
        for (i = 0; i < 10; i++) {
            int c = i * i;
            b += c;
            printf("%d %d %d", a, b, c);
        }
        printf("%d %d %d", a, b, c);
    }
    printf("%d %d %d", a, b, c);
}

int main() {
    funcao_teste(507);

    return 0;
}
```

O que será impresso?

Escopo de variáveis

```
#include<stdio.h>

void funcao_teste(int param1) {
    int a = param1;
    if (param1 > 0) {
        int b = 0;
        int i;
        for (i = 0; i < 10; i++) {
            int c = i * i;
            b += c;
            printf("%d %d %d", a, b, c);
        }
        printf("%d %d %d", a, b, c);
    }
    printf("%d %d %d", a, b, c);
}

int main() {
    funcao_teste(507);

    return 0;
}
```

Erro de compilação!

Escopo de variáveis

```
#include<stdio.h>

void funcao_teste(int param1) {
    int a = param1;
    if (param1 > 0) {
        int b = 0;
        int i;
        for (i = 0; i < 10; i++) {
            int c = i * i;
            b += c;
            printf("%d %d %d", a, b, c);
        }
        printf("%d %d %d", a, b, c);
    }
    printf("%d %d %d", a, b, c);
}

int main() {
    funcao_teste(507);

    return 0;
}
```

Variável c existe apenas aqui!

Escopo de variáveis

```
#include<stdio.h>

void funcao_teste(int param1) {
    int a = param1;
    if (param1 > 0) {
        int b = 0;
        int i;
        for (i = 0; i < 10; i++) {
            int c = i * i;
            b += c;
            printf("%d %d %d", a, b, c);
        }
        printf("%d %d %d", a, b, c);
    }
    printf("%d %d %d", a, b, c);
}

int main() {
    funcao_teste(507);

    return 0;
}
```

Variável b existe apenas aqui!

Escopo de variáveis

```
#include<stdio.h>
```

```
void funcao_teste(int param1) {
```

```
    int a = param1;
    if (param1 > 0) {
        int b = 0;
        int i;
        for (i = 0; i < 10; i++) {
            int c = i * i;
            b += c;
            printf("%d %d %d", a, b, c);
        }
        printf("%d %d %d", a, b, c);
    }
    printf("%d %d %d", a, b, c);
}
```

Variável a existe apenas aqui!

```
int main() {
    funcao_teste(507);
```

```
    return 0;
```

```
}
```

main é uma função também!

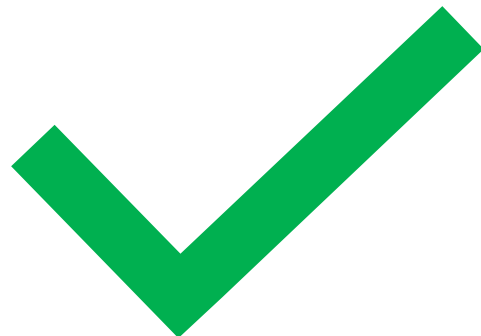
- A função main é chamada quando o programa é iniciado;
- A função retorna um inteiro:
 - Retorna 0 quando o programa termina como esperado;
 - Retorna outro valor em caso de erro!

```
int main() {  
    ...  
    return 0;  
}
```


Indentação

- Cuidado ao indentar vários blocos de código!!!

```
void funcao_teste(int param1) {  
→ int a = param1;  
→ if (param1 > 0) {  
→ int b = 0;  
→ int i;  
→ for (i = 0; i < 10; i++) {  
→ int c = i * i;  
→ b += c;  
→ printf("%d %d %d", a, b, c);  
→ }  
→ printf("%d %d %d", a, b, c);  
→ }  
→ printf("%d %d %d", a, b, c);  
}
```



Indentação

- **Cuidado ao indentar vários blocos de código!!!**

```
void funcao_teste(int param1) {  
    int a = param1;  
    if (param1 > 0) {  
        int b = 0;  
        int i;  
        for (i = 0; i < 10; i++) {  
            int c = i * i;  
            b += c;  
            printf("%d %d %d", a, b, c);  
        }  
        printf("%d %d %d", a, b, c);  
    }  
    printf("%d %d %d", a, b, c);  
}
```




Indentação

- Cuidado ao indentar vários blocos de código!!!

```
void funcao_teste(int param1) {  
    int a = param1;  
    if (param1 > 0) {  
        int b = 0;  
        int i;  
        for (i = 0; i < 10; i++) {  
            int c = i * i;  
            b += c;  
            printf("%d %d %d", a, b, c);  
        }  
        printf("%d %d %d", a, b, c);  
    }  
    printf("%d %d %d", a, b, c);  
}
```

Este printf será chamado se `param1 <= 0`?



Indentação

- Cuidado ao indentar vários blocos de código!!!

```
void funcao_teste(int param1) {
    int a = param1;
    if (param1 > 0) {
        int b = 0;
        int i;
        for (i = 0; i < 10; i++) {
            int c = i * i;
            b += c;
            printf("%d %d %d", a, b, c);
        }
        printf("%d %d %d", a, b, c);
    }
    printf("%d %d %d", a, b, c);
}
```

Esta chave aqui dá a impressão que o segundo `printf` está fora do bloco `if`!

Indentação

- Cuidado ao indentar vários blocos de código!!!

```
void funcao_teste(int param1) {  
    int a = param1;  
    if (param1 > 0) {  
        int b = 0;  
        int i;  
        for (i = 0; i < 10; i++) {  
            int c = i * i;  
            b += c;  
            printf("%d %d %d", a, b, c);  
        }  
        printf("%d %d %d", a, b, c);  
    }  
    printf("%d %d %d", a, b, c);  
}
```



Função para calcular PI

- Escreva uma função que calcule PI usando a Série de Gregory:

$$\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

Passando vetores
como argumento

Variáveis são passadas por valor

```
#include <stdio.h>

void muda_valor(int parametro) {
    parametro = 507;

    printf("%d\n", parametro);
}

int main() {

    int n = 1000;

    muda_valor(n);

    printf("%d\n", n);

    return 0;
}
```

Qual a saída
desse programa?

Variáveis são passadas por valor

```
#include <stdio.h>

void muda_valor(int parametro) {
    parametro = 507;

    printf("%d\n", parametro);
}

int main() {

    int n = 1000;

    muda_valor(n);

    printf("%d\n", n);

    return 0;
}
```

Qual a saída
desse programa?

Ok, variáveis são
passadas por valor!

507
1000

```
#include <stdio.h>

void muda_valor(int vetor[]) {
    vetor[0] = 90;

    printf("%d\n", vetor[0]);
}

int main() {

    int v[3] = {200, 500, 300};

    muda_valor(v);

    printf("%d %d %d\n", v[0], v[1], v[2]);

    return 0;
}
```

Qual a saída
desse programa?

Mas vetores são passados **por referência!**

```
#include <stdio.h>

void muda_valor(int vetor[]) {
    vetor[0] = 90;

    printf("%d\n", vetor[0]);
}

int main() {

    int v[3] = {200, 500, 300};

    muda_valor(v);

    printf("%d %d %d\n", v[0], v[1], v[2]);

    return 0;
}
```

Qual a saída
desse programa?

90

90 500 300

Mas por-que é assim?

Variáveis

```
int matricula = 123;
```

- O identificador de uma variável é usado para acessar seu valor;

```
printf("%d\n", matricula);
```

- O endereço de memória da variável é acessado com o operador address-of **&**


```
printf("%p\n", &matricula);
```

Vetores

```
int vetor[3] = {20, 500, 7};
```

- O identificador de um vetor representa o **endereço do primeiro elemento!**

```
printf("%p\n", vetor);  
printf("%p\n", &vetor[0]);
```



Retorna o mesmo valor
nos dois casos!

Exemplo

- Criar uma função que troca todos os números primos de um vetor pelo número -1.

Exercícios

1. Implemente os algoritmos de inserção (que vimos no início da aula) em forma de função;
2. Implemente uma função para remover o elemento de índice l de um vetor. Essa função deve deslocar (para a esquerda) os elementos que estão a frente no vetor.

Referências

- Slides do Prof. Fabrício Olivetti:
 - <http://folivetti.github.io/courses/ProgramacaoEstruturada/>
- Slide do Prof. Monael Pinheiro Ribeiro:
 - <https://sites.google.com/site/aed2018q1/>
- CELES, W.; CERQUEIRA, R.; RANGEL, J. L. Introdução a Estruturas de Dados. Elsevier/Campus, 2004.

Bibliografia básica

- PINHEIRO, F. A. C. Elementos de programação em C. Porto Alegre, RS: Bookman, 2012.
- FORBELLONE, A. L. V.; EBERSPACHER, H. F. Lógica de programação: a construção de algoritmos e estruturas de dados. 3ª edição. São Paulo, SP: Prentice Hall, 2005.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: teoria e prática. 2ª edição. Rio de Janeiro, RJ: Campus, 2002.

Bibliografia complementar

- AGUILAR, L. J. Programação em C++: algoritmos, estruturas de dados e objetos. São Paulo, SP: McGraw-Hill, 2008.
- DROZDEK, A. Estrutura de dados e algoritmos em C++. São Paulo, SP: Cengage Learning, 2009.
- KNUTH D. E. The art of computer programming. Upper Saddle River, USA: Addison- Wesley, 2005.
- SEDGEWICK, R. Algorithms in C++: parts 1-4: fundamentals, data structures, sorting, searching. Reading, USA: Addison-Wesley, 1998.
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de dados e seus algoritmos. 3a edição. Rio de Janeiro, RJ: LTC, 1994.
- TEWNENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. Estruturas de dados usando C. São Paulo, SP: Pearson Makron Books, 1995.