

Estrutura de Dados

Funções e Passagem de Parâmetros

Prof. Luiz Gustavo Almeida Martins

Modularização de Programas

- Característica fundamental da programação estruturada é a **modularização do programa**
 - Divisão do problema/tarefa em **subproblemas menores e mais simples** (**dividir para conquistar**)
- Cada módulo é tratado de **forma individual e independente**
 - Funções devem desempenhar **tarefas completas**
- Motivação para a modularização:
 - Facilitar o tratamento de problemas complexos
 - Permitir a reutilização de módulos

Modularização de Programas

- Um programa pode ser organizado em um ou mais módulos (funções):
 - **Função principal:** ponto de partida do programa (obrigatória)
 - **Funções auxiliares:** fornece serviços para outras funções

Exemplo

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

```
int quadrado (int x) {  
    return (x*x);  
}
```

```
int main () {  
    int nro, x;  
    printf ("Digite um numero: ");  
    scanf ("%d", &nro);  
    x = quadrado (nro);  
    printf ("\nO dobro de %d eh %d.\n", nro, x);  
    return 0;  
}
```

Protótipo da Função

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

```
int quadrado (int x) ; // Protótipo da função
```

```
int main () {  
    // corpo da função principal  
}
```

```
int quadrado (int x) {  
    return (x*x);  
}
```

Protótipo da Função

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

```
→ int dobro (int x); // Protótipo da função
```

```
int main () {  
    // corpo da função principal  
}
```

```
int dobro (int x) {  
    return (x*x);  
}
```

Forma Geral

- **Sintaxe:**

```
tipo nome (lista_parâmetros) {  
    bloco de comandos // corpo da função  
}
```

- **Tipo:** define o tipo de dado retornado pela função
 - void (função sem retorno)
 - tipos primitivos (ex: int, char, float, double)
 - tipos estruturados heterogêneos (struct)
 - Ponteiros (ex: variáveis indexadas)
- **Nome:** identificador da função, ou seja, o nome pelo qual ela deve ser chamada nos demais módulos
- **Lista_parâmetros:** define a quantidade e o tipo dos argumentos de entrada da função
 - O que ela deve receber para realizar sua tarefa

Argumentos de Entrada

- São os dados passados explicitamente para a função e que são necessários para a sua execução
 - Devem ser declarados entre parênteses após o nome da função como uma **lista de variáveis**

- **Exemplos:**

`void menu ()` OU `void menu (void)` // SEM entrada

`int soma (int x, int y)` // CERTO!

`int soma (int x, y)` // ERRADO!

`void exemplo (int x, float y)` // Permite diferentes tipos

- Nas chamadas da função devem ser fornecidos **valores para todos os seus parâmetros de entrada**

Argumentos de Entrada

- Dados informados na sua declaração da função são chamados de **argumentos formais**
 - Ex: *int elevado(int **base**, int **fator**)*
- Dados usados na chamada da função são denominados **argumentos atuais**
 - Ex: *resultado = elevado(**nro**, **exp**);*
- **A quantidade e os tipos de dados** dos **argumentos atuais** devem ser os mesmos dos seus respectivos **argumentos formais**

Retorno da Função

- Existe 2 formas da função retornar o controle para a função que a chamou:
- **Retorno implícito (sem resposta):**
 - Ocorre após a execução do último comando da função
 - Usado em funções do tipo *void*
- **Exemplo:**

```
void monta_linha(int tam) {  
    int y;  
    for (y=0; y<tam; y++)  
        printf("-");  
    printf("\n");  
}
```

Retorno da Função

- **Retorno explícito:**

- Ocorre por meio da instrução: ***return termo;***
- ***Termo*** define o que deve ser retornado
 - Pode ser **constante, variável ou expressão**
 - Deve ser do **mesmo tipo da função**
- Interrompe a execução e retorna algo para a função chamadora

- **Exemplo:**

```
int maior (int x, int y) {  
    if (x > y)  
        return (x); // Uso do parênteses é opcional  
    else  
        return y;  
}
```

Exemplo

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

```
int fatorial(int x) {  
    int i; fat = 1;  
    for (i = 2; i <= x; i++)  
        fat = fat * i;  
    return fat;  
}
```

```
int main() {  
    int n, resp;  
    printf("Digite um numero inteiro positivo: ");  
    scanf("%d", &n);  
    resp = fatorial(n);  
    printf("O fatorial de %d eh %d\n", n, resp);  
    return 0;  
}
```

Exemplo

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

```
int fatorial(int x) {  
    int i; fat = 1;  
    for (i = 2; i <= x; i++)  
        fat = fat * i;  
    return fat;  
}
```

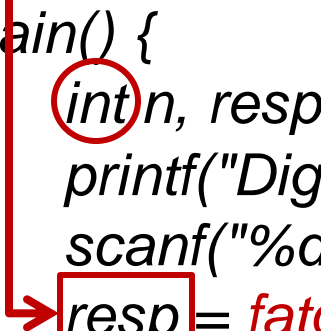
```
int main() {  
    int n, resp;  
    printf("Digite um numero inteiro positivo: ");  
    scanf("%d", &n);  
    resp = fatorial(n);  
    printf("O fatorial de %d eh %d\n", n, resp);  
    return 0;  
}
```

Exemplo

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

```
int fatorial(int x) {  
    int i; fat = 1;  
    for (i = 2; i <= x; i++)  
        fat = fat * i;  
    return fat;  
}
```

```
int main() {  
    int n, resp;  
    printf("Digite um numero inteiro positivo: ");  
    scanf("%d", &n);  
    resp = fatorial(n);  
    printf("O fatorial de %d eh %d\n", n, resp);  
    return 0;  
}
```

A red line with an arrow at the end originates from the boxed 'fat' variable in the 'return fat;' statement of the 'fatorial' function. It extends downwards and then horizontally to the right, pointing to the boxed 'resp' variable in the 'resp = fatorial(n);' statement of the 'main' function, illustrating the return value being passed back to the caller.

Escopo de Validade das Declarações

- **Variáveis locais:**
 - Declaradas no corpo de uma função
 - Variável é válida apenas na função de origem
 - Parâmetros de entrada são tratadas como variáveis locais
- **Variáveis globais:**
 - Declaradas fora do corpo das funções (no início do programa)
 - Variáveis válidas em todo o programa

Exemplo

```
#include <stdio.h>
int a = 33;
void sss() {
    int b = 88;
    printf("sss: a = %d, b = %d\n", a, b);
}

int main() {
    int a = 77, b = 55;
    printf("main1: a = %d, b = %d\n", a, b);
    sss();
    printf("main2: a = %d, b = %d\n", a, b);
    return 0;
}
```


Exemplo

Saída esperada:

main(): a = 77, b = 55

```
#include <stdio.h>
int a = 33;
void sss() {
    int b = 88;
    printf("sss: a = %d, b = %d\n", a, b);
}

int main() {
    int a = 77, b = 55;
    printf("main1: a = %d, b = %d\n", a, b);
    sss();
    printf("main2: a = %d, b = %d\n", a, b);
    return 0;
}
```

Exemplo

Saída esperada:

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

```
int a = 33;
```

```
void sss() {
```

```
    int b = 88;
```

```
    printf("sss: a = %d, b = %d\n", a, b);
```

```
}
```

```
int main() {
```

```
    int a = 77, b = 55;
```

```
    printf("main1: a = %d, b = %d\n", a, b);
```

```
    sss();
```

```
    printf("main2: a = %d, b = %d\n", a, b);
```

```
    return 0;
```

```
}
```

main(): a = 77, b = 55

sss(): a = 33, b = 88

Exemplo

Saída esperada:

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

```
int a = 33;
```

```
void sss() {
```

```
    int b = 88;
```

```
    printf("sss: a = %d, b = %d\n", a, b);
```

```
}
```

```
int main() {
```

```
    int a = 77, b = 55;
```

```
    printf("main1: a = %d, b = %d\n", a, b);
```

```
    sss();
```

```
    printf("main2: a = %d, b = %d\n", a, b);
```

```
    return 0;
```

```
}
```

main(): a = 77, b = 55

sss(): a = 33, b = 88

main(): a = 77, b = 55

Troca de Dados entre Funções

- Existem duas formas de interagir com funções:
 - **Explícita:** quando a troca é feita através de **argumentos de entrada**:
 - **Implícita:** quando a troca de dados é feita por meio de **variáveis globais**.
 - ★ Esta forma funciona de modo similar à passagem de argumentos por referência, pois qualquer alteração realizada durante a execução da função **afeta o conteúdo da variável** para todos os demais módulos

Passagem por Valor

- O argumento formal recebe uma **cópia do conteúdo** do argumento atual
- Ocorre uma **duplicação do dado** na memória
 - Variável local é criada no escopo da função e recebe o mesmo valor do argumento atual
- Mudanças na função chamada **NÃO AFETAM** o valor do argumento atual na função chamadora

Passagem por Valor

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

```
void ff(int a) {  
    a++;  
    printf("Durante ff, a = %d\n", a);  
}
```

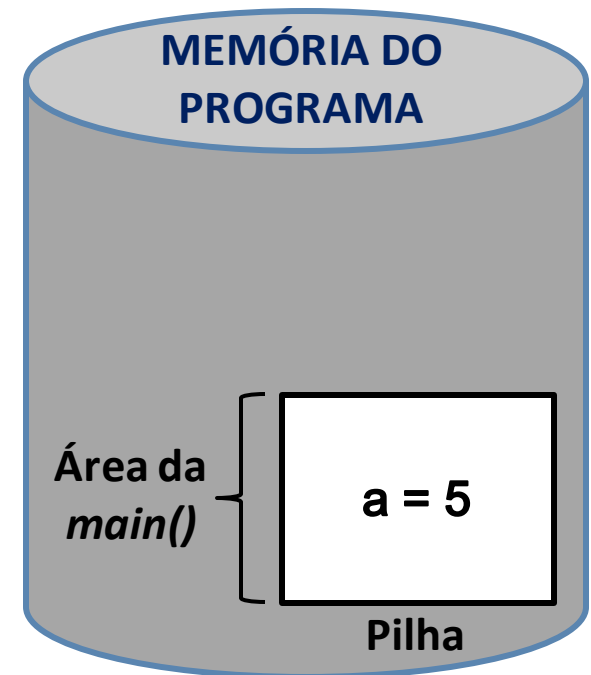
```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```

Passagem por Valor

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

```
void ff(int a) {  
    a++;  
    printf("Durante ff, a = %d\n", a);  
}
```

```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```



Passagem por Valor

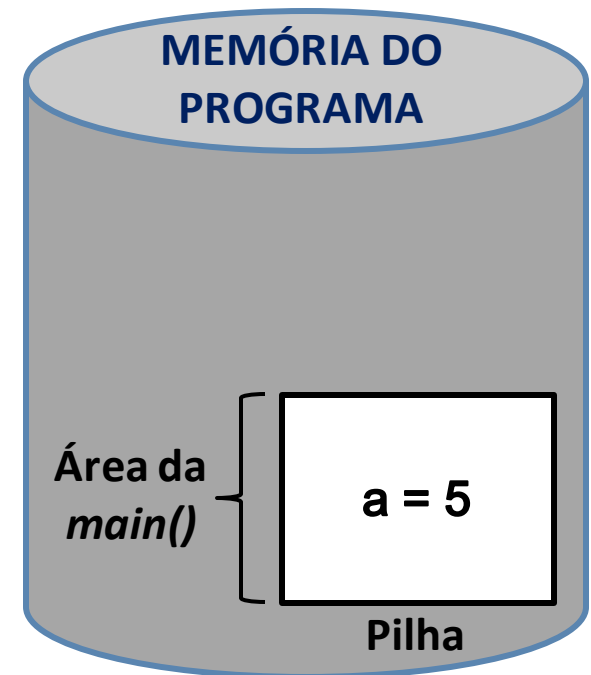
```
#include <stdio.h>
```

```
void ff(int a) {  
    a++;  
    printf("Durante ff, a = %d\n", a);  
}
```

```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```

Saída esperada:

main(): a = 5



Passagem por Valor

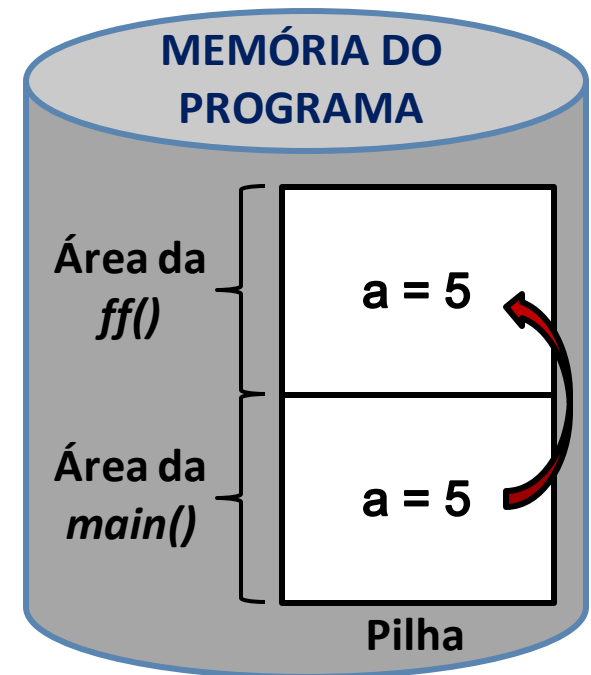
```
#include <stdio.h>
```

```
void ff(int a) {  
    a++;  
    printf("Durante ff, a = %d\n", a);  
}
```

```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```

Saída esperada:

main(): a = 5



Passagem por Valor

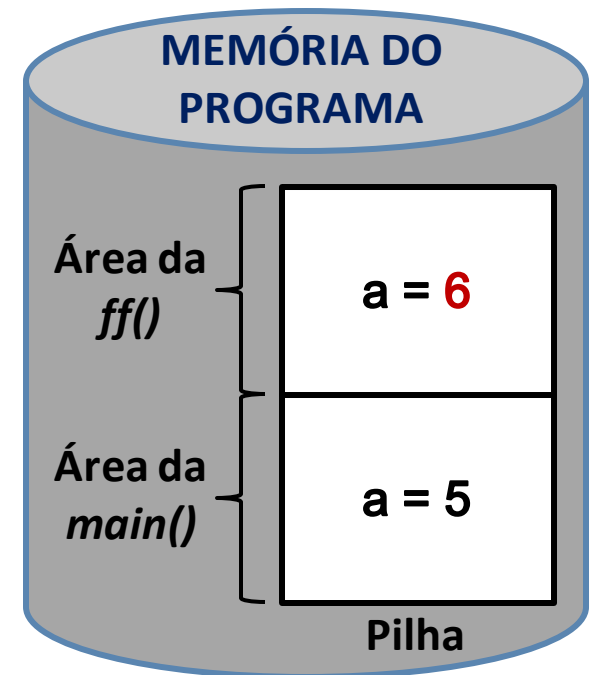
```
#include <stdio.h>
```

```
void ff(int a) {  
    a++;  
    printf("Durante ff, a = %d\n", a);  
}
```

```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```

Saída esperada:

main(): a = 5



Passagem por Valor

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

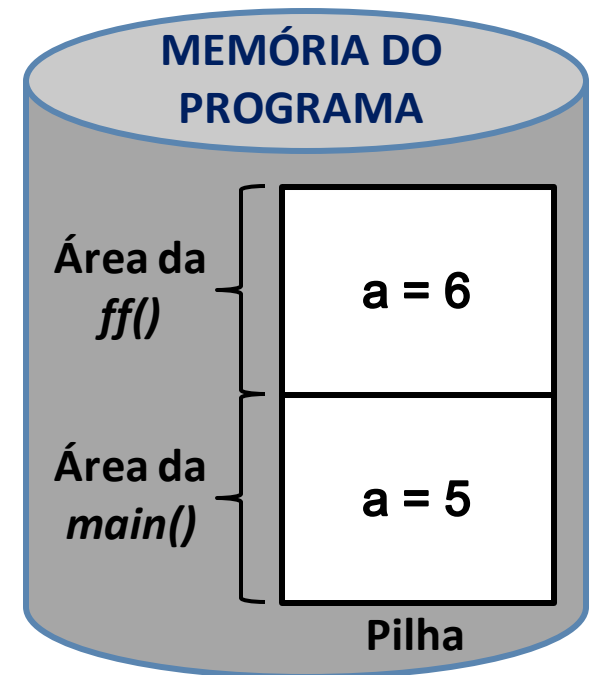
```
void ff(int a) {  
    a++;  
    printf("Durante ff, a = %d\n", a);  
}
```

```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```

Saída esperada:

main(): a = 5

ff(): a = 6



Passagem por Valor

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

```
void ff(int a) {  
    a++;  
    printf("Durante ff, a = %d\n", a);  
}
```

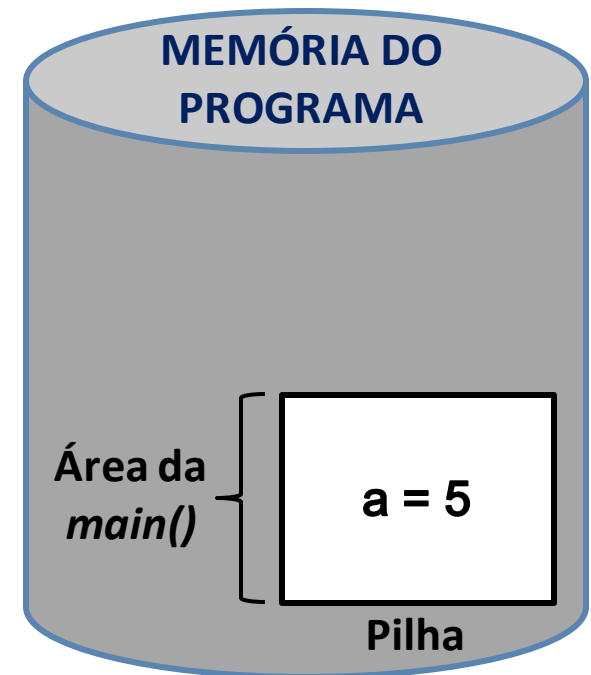
```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```

Saída esperada:

main(): a = 5

ff(): a = 6

main(): a = 5



Passagem por Referência

- O argumento formal recebe o **endereço da memória** onde está o argumento atual, não o seu conteúdo
 - Envolve o uso de ponteiros
 - Endereço é passado por valor (**CÓPIA**)
- Mudanças na função chamada **AFETAM** o valor do argumento atual na função chamadora
 - Alterações são realizadas na posição da memória que contém o dado original
- Variáveis indexadas são sempre passadas por referência
 - **Nome da variável (sem o índice) = endereço do 1º elemento**

Passagem por Referência

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

```
void ff(int *a) {  
    (*a)++;  
    printf("Durante ff, a = %d\n", *a);  
}
```

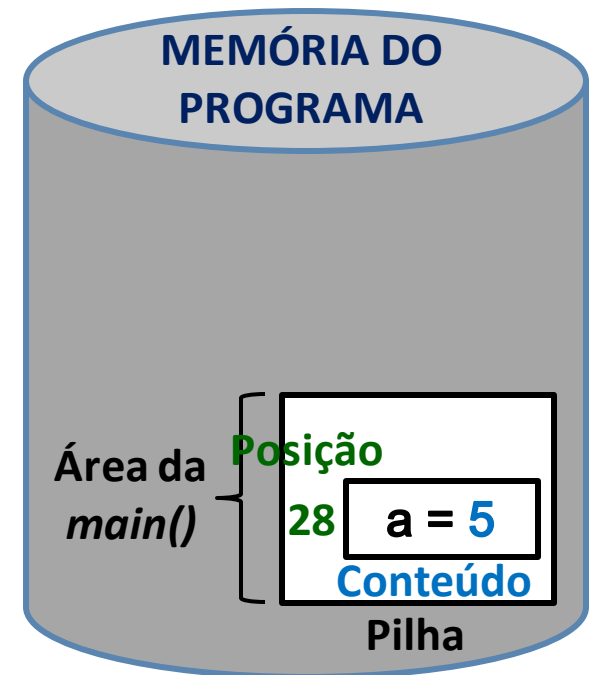
```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(&a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```

Passagem por Referência

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

```
void ff(int *a) {  
    (*a)++;  
    printf("Durante ff, a = %d\n", *a);  
}
```

```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(&a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```



Passagem por Referência

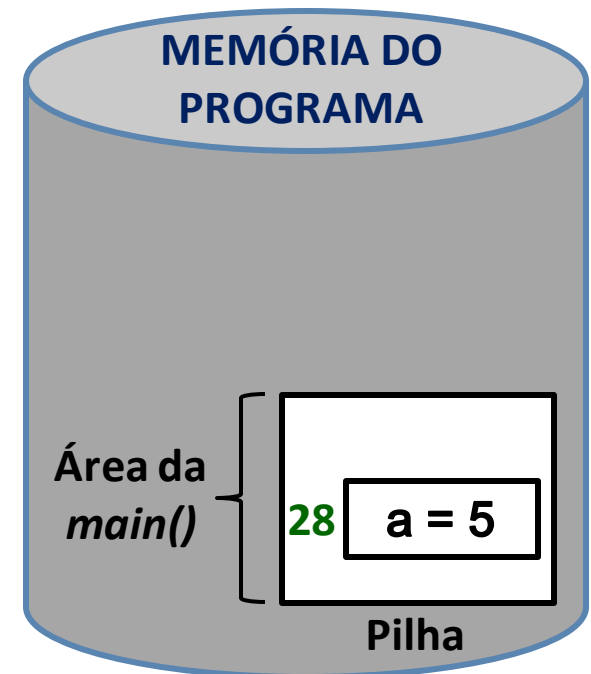
```
#include <stdio.h>
```

```
void ff(int *a) {  
    (*a)++;  
    printf("Durante ff, a = %d\n", *a);  
}
```

```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(&a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```

Saída esperada:

main(): a = 5



Passagem por Referência

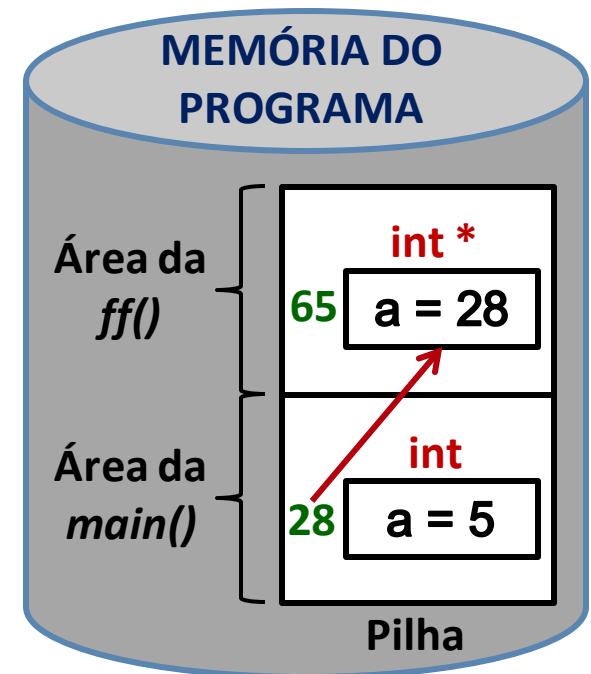
```
#include <stdio.h>
```

```
void ff(int *a) {  
    (*a)++;  
    printf("Durante ff, a = %d\n", *a);  
}
```

```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(&a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```

Saída esperada:

main(): a = 5



Passagem por Referência

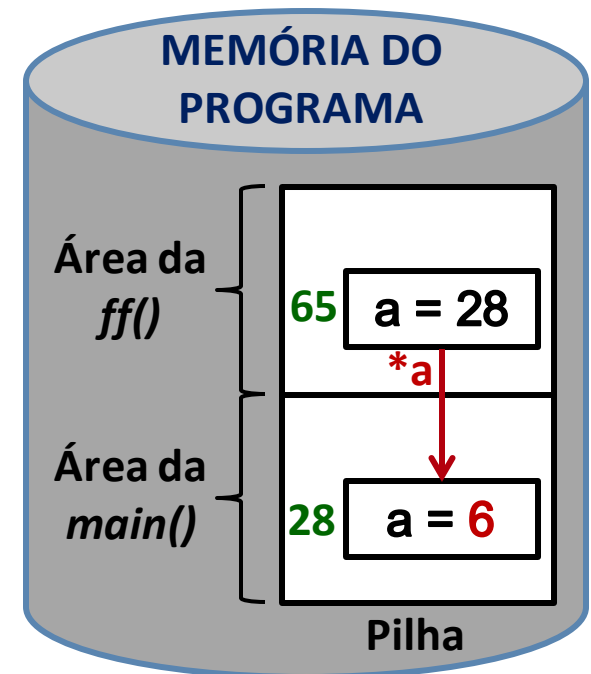
```
#include <stdio.h>
```

```
void ff(int *a) {  
    (*a)++;  
    printf("Durante ff, a = %d\n", *a);  
}
```

```
int main() {  
    int a = 5;  
    printf("Antes de ff, a = %d\n", a);  
    ff(&a); // chamada da função  
    printf("Depois de ff, a = %d\n", a);  
    return 0;  
}
```

Saída esperada:

main(): a = 5



Passagem por Referência

```
#include <stdio.h>

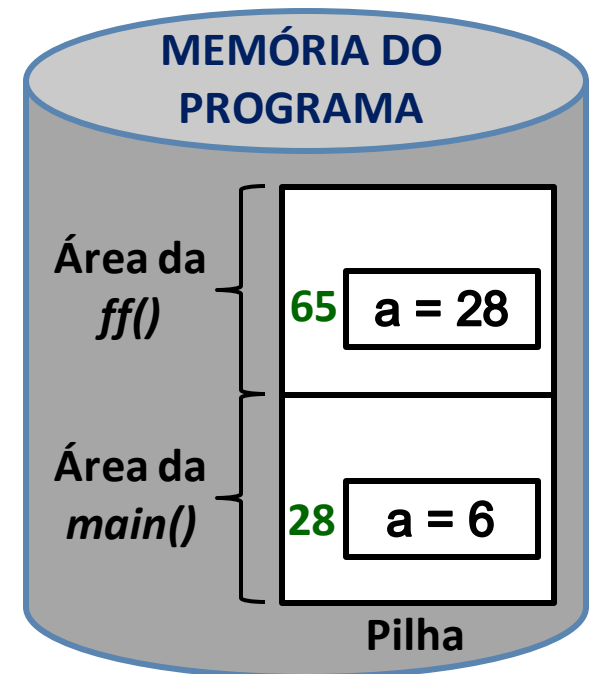
void ff(int *a) {
    (*a)++;
    printf("Durante ff, a = %d\n", *a);
}

int main() {
    int a = 5;
    printf("Antes de ff, a = %d\n", a);
    ff(&a); // chamada da função
    printf("Depois de ff, a = %d\n", a);
    return 0;
}
```

Saída esperada:

main(): a = 5

ff(): a = 6



Passagem por Referência

```
#include <stdio.h>

void ff(int *a) {
    (*a)++;
    printf("Durante ff, a = %d\n", *a);
}

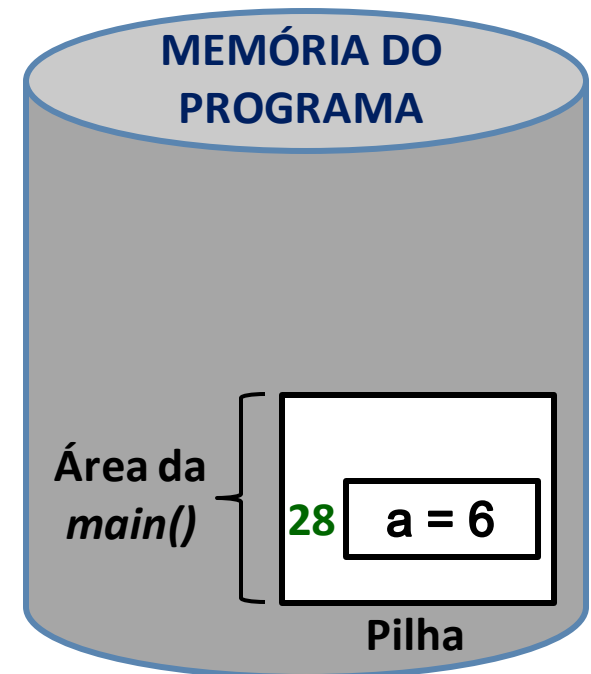
int main() {
    int a = 5;
    printf("Antes de ff, a = %d\n", a);
    ff(&a); // chamada da função
    printf("Depois de ff, a = %d\n", a);
    return 0;
}
```

Saída esperada:

main(): a = 5

ff(): a = 6

main(): a = 6



Variáveis Indexadas

- A **passagem** de uma variável indexada é **por referência**
 - Alterações **AFETAM** os elementos do array original
- O **retorno** de uma variável indexada criada na função deve ser feita **por ponteiro**
 - Estrutura deve ser **alocada dinamicamente** para não ser apagada ao final da função


- **Ex:**

```
int * novo_vetor (int B[10]) {  
    int i, *C;  
    C = (int *) malloc(10*sizeof(int));  
    for (i = 0; i < 10; i++)  
        C[i] = B[i] + 4;  
    return C; }
```

Variáveis Indexadas

- A **passagem** de uma variável indexada é **por referência**
 - Alterações **AFETAM** os elementos do array original
- O **retorno** de uma variável indexada criada na função deve ser feita **por ponteiro**
 - Estrutura deve ser **alocada dinamicamente** para não ser apagada ao final da função

• **Ex:**

```
int * novo_vetor(int B[10]) {  
    int i, *C;  
    C = (int *) malloc(10*sizeof(int));  
    for (i = 0; i < 10; i++)  
        C[i] = B[i] + 4;  
    return C; }  

```

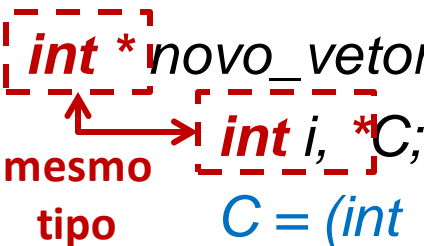
Outras formas:

```
int B[]  
int *B
```

Variáveis Indexadas

- A **passagem** de uma variável indexada é **por referência**
 - Alterações **AFETAM** os elementos do array original
- O **retorno** de uma variável indexada criada na função deve ser feita **por ponteiro**
 - Estrutura deve ser **alocada dinamicamente** para não ser apagada ao final da função

• **Ex:**

```
int * novo_vetor (int B[10]) {  
    int i, *C;  
    C = (int *) malloc(10*sizeof(int));  
    for (i = 0; i < 10; i++)  
        C[i] = B[i] + 4;  
    return C; }  
mesmo  
tipo
```

Variáveis Estruturadas

- Estruturas podem ser passadas por valor ou por referência
 - Por referência costuma ser melhor (**ocupa menos memória**)
- Campos da estrutura também podem ser passados como parâmetros
 - Parâmetro de entrada será do tipo de dado do campo

Exemplo: Passagem por Valor

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

```
struct matriz {  
    int nlin;  
    int ncol;  
    int elem[10][10];  
};
```

```
typedef struct matriz matriz;
```

```
matriz NovaMatriz(matriz B) {  
    int i,j;  
    matriz C;  
    for (i = 0; i < B.nlin; i++)  
        for (j = 0; j < B.ncol; j++)  
            C.elem[i][j] = B.elem[i][j] + 4;  
    C.nlin = B.nlin;  
    C.ncol = B.ncol;  
    return C;  
}
```

Exemplo: Passagem por Valor

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

```
struct matriz {  
    int nlin;  
    int ncol;  
    int elem[10][10];  
};  
typedef struct matriz matriz;
```

**Definição da
estrutura**

```
matriz NovaMatriz(matriz B) {  
    int i,j;  
    matriz C;  
    for (i = 0; i < B.nlin; i++)  
        for (j = 0; j < B.ncol; j++)  
            C.elem[i][j] = B.elem[i][j] + 4;  
    C.nlin = B.nlin;  
    C.ncol = B.ncol;  
    return C;  
}
```

Exemplo: Passagem por Valor

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

```
struct matriz {  
    int nlin;  
    int ncol;  
    int elem[10][10];  
};  
typedef struct matriz matriz;
```

Definição da
estrutura

```
matriz NovaMatriz(matriz B) {  
    int i,j;  
    matriz C;  
    for (i = 0; i < B.nlin; i++)  
        for (j = 0; j < B.ncol; j++)  
            C.elem[i][j] = B.elem[i][j] + 4;  
    C.nlin = B.nlin;  
    C.ncol = B.ncol;  
    return C;  
}
```

Declaração

Exemplo: Passagem por Valor

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

```
struct matriz {  
    int nlin;  
    int ncol;  
    int elem[10][10];  
};  
typedef struct matriz matriz;
```

Definição da
estrutura

```
matriz NovaMatriz(matriz B) {  
    int i,j;  
    matriz C;  
    for (i = 0; i < B.nlin; i++)  
        for (j = 0; j < B.ncol; j++)  
            C.elem[i][j] = B.elem[i][j] + 4;  
    C.nlin = B.nlin;  
    C.ncol = B.ncol;  
    return C;  
}
```

Passagem por
valor

Acesso **DIRETO**
aos campos
(operador [.])

Exemplo: Passagem por Referência

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

```
struct matriz {  
    int nlin;  
    int ncol;  
    int elem[10][10];  
};  
typedef struct matriz matriz;
```

```
matriz NovaMatriz(matriz *B){  
    int i,j;  
    matriz C;  
    for (i = 0; i < B->nlin; i++)  
        for (j = 0; j < B->ncol; j++)  
            C.elem[i][j] = B->elem[i][j] + 4;  
    C.nlin = B->nlin;  
    C.ncol = B->ncol;  
    return C;  
}
```

Passagem por
referência

Acesso indireto
aos campos
(operador [->])

Exemplo: Passagem por Referência

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

```
struct matriz {  
    int nlin;  
    int ncol;  
    int elem[10][10];  
};
```

```
typedef struct matriz matriz;
```

```
matriz NovaMatriz(matriz *B){  
    int i,j;  
    matriz C;  
    for (i = 0; i < B->nlin; i++)  
        for (j = 0; j < B->ncol; j++)  
            C.elem[i][j] = B->elem[i][j] + 4;  
    C.nlin = B->nlin;  
    C.ncol = B->ncol;  
    return C;  
}
```

B->nlin é equivalente a ***(*B).nlin***

**Passagem por
referência**

Argumentos da Função Principal

- Usados para passar argumentos ao programa
 - Chamada do programa por linha de comando
 - **Sintaxe:**

tipo_retorno main (int argc , char *argv[])

- **argc (argument counter)**: indica a quantidade de palavras digitadas na linha de comando
 - Variável inteira
 - Corresponde à quantidade de elementos presentes no **argv**
- **argv (argument value)**: contém as palavras (*strings*) digitadas na linha de comando
 - Vetor de *strings* (ou matriz de caracteres)
 - O nome do programa também é contado (1º elemento: **argv[0]**)

Funções Auxiliares

- Argumentos recebidos pela *main()* são sempre *strings*
- Argumentos numéricos necessitam de conversão
 - **Conversão manual:** trabalhosa e exige conhecimento sobre as formas de representação dos tipos de dados
 - **Conversão através de funções:** mais fácil e simples
- **Funções de conversão de *strings*:**
 - ***atoi()***: alfanumérico para inteiro
 - ***atof()***: alfanumérico para float
 - Ambas estão presentes na biblioteca ***stdlib.h***
- **Exemplos de uso:**
 - int idade = atoi(argv[1]);*
 - float salario = atof(argv[1]);*

Exemplo

```
// Programa data.c
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[]) {
    int mes;
    char *nm_mes[] = {"Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho",
        "Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro"};

    // if usado para testar se o nro de parametros fornecidos está correto:
    if(argc == 4) {
        mes = atoi(argv[2]); // Converte uma string em um inteiro

        if (mes < 1 || mes > 12) // Testa se o mês é válido
            printf("O mês deveria estar dentro do intervalo 1 e 12.");
        else
            printf("\n%s de %s de %s", argv[1], nm_mes[mes-1], argv[3]);
    }
    else
        printf("Número de parâmetros inválido.");
}
```

Exemplo

```
// Programa data.c
#include <stdio.h>
#include <stdlib.h>
```

```
void main(int argc, char *argv[]) {
    int mes;
    char *nm_mes[] = {"Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho",
        "Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro"};

    // if usado para testar se o nro de parametros fornecidos está correto:
    if(argc == 4) {
        mes = atoi(argv[2]); // Converte uma string em um inteiro

        if (mes < 1 || mes > 12) // Testa se o mês é válido
            printf("O mês deveria estar dentro do intervalo 1 e 12.");
        else
            printf("\n%s de %s de %s", argv[1], nm_mes[mes-1], argv[3]);
    }
    else
        printf("Número de parâmetros inválido.");
}
```

Linha de comando:

data 19 04 1999

Resultado:

19 de Abril de 1999

Exercícios

1. Faça uma função que recebe como parâmetro dois números inteiros e retorne o primeiro elevado ao segundo.
2. Faça uma função para verificar se um número é positivo ou negativo. Sendo que o valor de retorno será 1 se positivo, -1 se negativo ou 0 se o número for igual a 0.
3. Faça uma função para verificar se um número dado como entrada é um quadrado perfeito. Um quadrado perfeito é um número inteiro não negativo que pode ser expresso como o quadrado de outro número inteiro. Ex: 1, 4, 9...
4. Faça uma função que receba dois vetores de 10 posições e retorne um terceiro vetor contendo o resultado do produto dos vetores de entrada.

Referências

- *Coelho, Paulo R. S. L., Linguagem C: Funções, material didático da disciplina de Introdução a Programação, UFU.*
- *Backes, André, Linguagem C Descomplicada, portal de vídeo-aulas, <https://programacaodescomplicada.wordpress.com/>, acessado em 09/03/2016.*
- *Martins, Luiz G. A., Apostila de Linguagem C: conceitos básicos, material didático das disciplinas de introdução a programação, UFU.*
- *Celes, W., Cerqueira, R. e Rangel, J. L. Introdução a estruturas de dados. Ed. Campus Elsevier, 2004.*