

Programação 2

Revisão: Programação básica

Rivera

PARTE 2

Funções

- Comando para definição de função:

```
tipo nome_função (parâmetros... )
```

```
{  
    corpo da função  
    retorna valor  
}
```

```
void nome_função (parâmetros... )
```

```
{  
    corpo da função  
}
```

```
#include <stdio.h>  
int fat (int n);  
int main (void) {  
    int r, n = 5;  
    r = fat ( n );  
    printf ("Fat (%d )= %d \n", n, r);  
    return 0;  
}
```

Funções podem ser definidas por meio da declaração de seu tipo e nome, com corpo (comandos a serem executados dentro da função) e um valor a ser retornado, exceto nas funções de tipo void, que não retornam valor algum

```

int defineDados (int vn [ ] )
{
    int i, a, n;
    printf (" \n Digite n: ");
    scanf ("%d", &n );
    for (i = 1; i <= n; i++)
    {
        printf (" Entre o val %d-o: ", i);
        scanf ("%d", &a);
        vn[i-1] = a;
    }
    return ( n );
}

```

// -----(programa principal)-----

```

void main (void)
{
    int n, vn[100];
    n = defineDados(vn);
    mostraDados(n, vn);
    ordenaPar(n, vn);
    mostraDados(n, vn);
}

```

```

Void defineDados (int *n, int vn [ ] )
{
    int i, a;
    printf (" \n Digite n: ");
    scanf ("%d", n );
    for (i = 1; i <= *n; i++)
    {
        printf (" Entre o val %d-o: ", i);
        scanf ("%d", &a);
        vn[i-1] = a;
    }
}

```

// -----(programa principal)-----

```

void main (void)
{
    int n, vn[100];
    defineDados(&n, vn);
    mostraDados(n, vn);
    ordenaPar(n, vn);
    mostraDados(n, vn);
}

```

O programa a esquerda faz uso de uma função do tipo Int, retornando o valor de N, enquanto o programa à direita utiliza apenas funções do tipo void, que por sua vez, não retornam nenhum valor.

```
/* programa fatorial de um número */
```

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

```
int fat (int n);
```

```
int main (void)
```

```
{    int n, r;  
    printf("Digite um número nao negativo:");  
    scanf("%d", &n);  
    fat(n);  
    return 0;  
}
```

```
/* função para calcular o valor do fatorial */
```

```
int fat (int n)
```

```
{    int i, f = 1;  
    for (i = 1; i <= n; i++)  
        f *= i;  
    printf ("Fatorial = %f", f);  
}
```

```
/* programa fatorial de um número */
```

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

```
/* função para calcular o valor do fatorial */
```

```
int fat (int n)
```

```
{    int i, f = 1;  
    for (i = 1; i <= n; i++)  
        f *= i;  
    printf ("Fatorial = %f", f);  
}
```

```
int main (void)
```

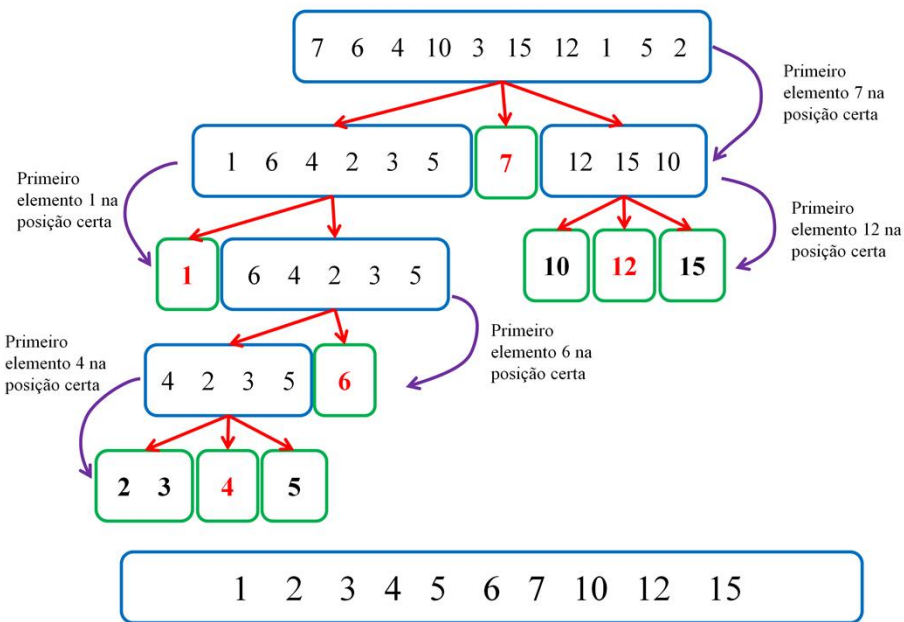
```
{    int n, r;  
    printf("Digite um número nao negativo:");  
    scanf("%d", &n);  
    fat(n);  
    return 0;  
}
```

Os programas utilizam da mesma lógica de algoritmo, porém com ordenação diferente das funções: O programa à esquerda possui uma função que lê o número dado pelo usuário, e depois esse valor é lido dentro de outra função. Já o programa à direita utiliza uma função que define o cálculo de fatorial, para depois essa função ser chamada dentro de outra função posteriormente definida, que lê o número dado pelo usuário, e o utiliza como parâmetro para a função de cálculo

Exercícios

- 1) Dado um vetor v de n números inteiros. Definir um algoritmo (programa) para recolocar os elementos de v de forma que o primeiro número ($v[0]$) apareça em alguma posição j do vetor de forma que $v[0] \dots v[j-1] < v[j] \leq v[j+1] \dots v[n-1]$
- 2) Dados dois vetores va e vb de números inteiros ordenados em forma crescente de n e m elementos respectivamente. Definir um terceiro vetor vc ordenado, de forma eficiente, combinando elementos de va e vb .
- 3) Dado um vetor v de n números inteiros ordenados em forma crescente. Deseja-se buscar o número de ocorrências de um número a usando busca binária e sequência parcial.
- 4) Usando o algoritmo em (1) escrever um algoritmo para a ordenação dos vetores parciais $v[0] \dots v[j-1]$ e $v[j+1] \dots v[n-1]$.

Estrutura binária (recursiva)



Esse programa é um algoritmo de ordenação de vetores, que sequencialmente lê o vetor, e posiciona seu primeiro elemento na sua posição correta crescente, criando outros vetores, "encaixando-os" de maneira ordenada, e depois unindo novamente todos os elementos em um único vetor

Algoritmo básico

Ordenar (ai, bi, v)

iniciar: a, b, pivô

colocar pivô na posição certa b

ordenar (a1, b-1, v)

ordenar (b+1, b1, v)

Fim.

Ordenar (ai, bi, v)

iniciar: a, b, pivô

Enquanto $a < b$, fazer

 caminhar $a++$ até obstáculo maior

 caminhar $b--$ até obstáculo menor

Se ($a < b$)

 trocar $v[a]$ com $v[b]$

Fenq.

colocar pivô na posição b // posicao certa

ordenar (a1, b-1, v)

ordenar (b+1, b1, v)

Fim.

Aqui é apresentado a ideia inicial do algoritmo de ordenação anterior, fazendo uso de um pivô e estruturas de repetição.

Pilha de Execução

Pilha: comunicação entre funções

c	'x'	112 - variável c no endereço 112 com valor 'x'
b	43.5	108 - variável b no endereço 108 com valor 43.5
a	7	104 - variável a no endereço 104 com valor 7

A pilha de execução armazena as variáveis, seus respectivos valores e endereços. A pilha funciona de maneira imperativamente ordenada, de forma que os dados são, conforme a execução é feita, colocados da base ao topo da pilha, e devem ser retirados na ordem contrária.

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>

int fat ( int n );

Int main ( void )
{   int n = 5;
    int r;
    r = fat ( n );
    printf ("Fatorial de %d = %d \n", n, r);
    return 0;
}

int fat ( int n)
{   int f = 1;
    while (n != 0)
    {
        f *= n;
        n--;
    }
    return f;
}
```

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */  
#include <stdio.h>
```

```
int fat ( int n );
```

→

```
Int main ( void )  
{  
    int n = 5;  
    int r;  
    r = fat ( n );  
    printf ("Fatorial de %d = %d \n", n, r);  
    return 0;  
}
```

```
int fat ( int n )  
{  
    int f = 1;  
    while ( n != 0 )  
    {  
        f *= n;  
        n--;  
    }  
    return f;  
}
```

1: Início do programa: pilha vazia



```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
```

```
int fat ( int n );
```

```
Int main ( void )
```

```
{ int n = 5;
  int r;
  r = fat ( n );
  printf ("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



```
int fat ( int n)
{ int f = 1;
  while (n != 0)
  {
    f *= n;
    n--;
  }
  return f;
}
```

2: Declaração das variáveis: n, r

r	-
n	5
>	main

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
```

```
int fat ( int n );
```

```
Int main ( void )
{
    int n = 5;
    int r;
    r = fat ( n );
    printf ("Fatorial de %d = %d \n", n, r);
    return 0;
}
```

→

```
int fat ( int n)
{
    int f = 1;
    while (n != 0)
    {
        f *= n;
        n--;
    }
    return f;
}
```


3: chamada da função: cópia param.

n	5
>f	fat
r	-
n	5
>m	main

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
```

```
int fat ( int n );
```

```
Int main ( void )
{
    int n = 5;
    int r;
    r = fat ( n );
    printf ("Fatorial de %d = %d \n", n, r);
    return 0;
}
```



```
int fat ( int n)
{
    int f = 1;
    while (n != 0)
    {
        f *= n;
        n--;
    }
    return f;
}
```

4: Declara variável local: f

f	1.0
n	5
>f	fat
r	-
n	5
>m	main

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
```

```
int fat ( int n );
```

```
Int main ( void )
{
    int n = 5;
    int r;
    r = fat ( n );
    printf ("Fatorial de %d = %d \n", n, r);
    return 0;
}
```

```
int fat ( int n)
{
    int f = 1;
    while (n != 0)
    {
        f *= n;
        n--;
    }
    return f;
}
```



5: Final do laço

f	120.0	
n	0	
>f	fat	
r	-	108
n	5	104
>m	main	100

```

/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>

int fat ( int n );

Int main ( void )
{
    int n = 5;
    int r;
    r = fat ( n );
    → printf ("Fatorial de %d = %d \n", n, r);
    return 0;
}

int fat ( int n)
{
    int f = 1;
    while (n != 0)
    {
        f *= n;
        n--;
    }
    return f;
}

```

6: Retorno da função: desempilha

r	120.0
n	5
>m	main

Essa sequência de slides mostra o funcionamento da pilha durante a execução de um programa, da maneira explicada anteriormente.

Funções Recursivas

- Tipos de recursão
 - ♦ Direta
 - Função A chama a ela própria
 - ♦ Indireta
 - Função A chama função B que, por sua vez, chama A
- Comportamento
 - ♦ Uso de ambiente local para cada chamada
 - ♦ Independência das variáveis locais em cada chamada
 - Como chamadas entre funções diferentes

A recursão consiste na ideia de quando uma função faz uso de si mesma em sua execução, como um "looping", podendo, por exemplo, durante a primeira execução, calcular um valor, e passar esse valor como novo parâmetro para a segunda execução dessa função, possibilitando um novo cálculo a partir do parâmetro atualizado. Essa técnica é utilizada amplamente na programação, especialmente no uso de linguagens funcionais.

A recursão pode ser dada de forma direta, onde a função chama ela mesma, ou indireta, onde existe uma função intermediária para isso. As variáveis são locais para cada execução da função, ou seja, cada função é executada de maneira independente, mantendo apenas os valores repassados como parâmetros.

Funções Recursivas

Exemplo: definição recursiva de fatorial

```
/* Função recursiva para cálculo do fatorial */  
int fat ( int n )  
{  
    if ( n == 0 )  
        return 1;  
    else  
        return n * fat ( n - 1 );  
}
```

Nesse algoritmo, o valor de n é passado por uma estrutura condicional, e caso não possua o valor adequado, o valor é atualizado e passado novamente para a função como parâmetro

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



f	-
n	5
>f	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



f	-
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



	1
	0
	fat(0)
	-
	1
	fat(1)
	-
	2
	fat(2)
	-
	3
	fat(3)
f	-
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



	1
	1
	fat(1)
	-
	2
	fat(2)
	-
	3
	fat(3)
f	-
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



	2
	2
	fat(2)
	-
	3
	fat(3)
f	-
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



	6
	3
	fat(3)
f	-
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



f	24
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```




f	120
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



r	120
n	5
>m	main

Demonstração do comportamento da pilha de execução durante a utilização de recursão. A pilha acumula todos os valores que a função adquire, até que seja limpa no fim da recursão

Variáveis Globais

- Variável global
 - ♦ Declarada fora do corpo das funções
 - Visível por todas as funções subsequentes
 - ♦ Não é armazenada na pilha de execução
 - Não deixa de existir quando a execução de uma função termina
 - Existe enquanto o programa estiver executando
 - ♦ Utilização de variáveis globais
 - Deve ser feito com critério
 - Pode-se criar um alto grau de interdependência entre as funções
 - Dificulta o entendimento e o reuso do código

variáveis globais podem ser utilizadas em todo o escopo do programa, assumindo o mesmo valor de forma universal, entretanto seu uso acarreta consequências e devem ser utilizadas com cautela.

Variáveis Globais

```
#include <stdio.h>

int s, p; /* variáveis globais */

void somaprod (int a, int b)
{
    s = a + b;
    p = a * b;
}

int main (void)
{
    int x, y;
    scanf("%d %d", &x, &y);
    somaprod(x,y);
    printf("Soma = %d produto = %d\n", s, p);
    return 0;
}
```

O programa acima faz uso de variáveis globais, que são utilizadas dentro da função de cálculo, e depois acessadas para impressão, dentro do escopo da função main

Variáveis Estáticas

- Variável estática
 - ♦ Declarada no corpo de uma função
 - Visível apenas dentro da função
 - ♦ Não é armazenada na pilha de execução
 - Armazenada em uma área de memória estática
 - Continua existindo antes e depois de a função ser executada
 - ♦ Utilização de variáveis estáticas
 - Quando for necessário recuperar o valor de uma variável atribuída na última vez que a função foi executada

Uma variável estática é uma variável de escopo limitado, como uma variável declarada dentro de uma função (ou seja, não global), porém permanente, e não armazenada na pilha de execução, ou seja, uma função pode fazer uso de uma variável estática sempre que for executada mais de uma vez, e é necessário a manipulação do valor da execução anterior.

Variáveis Estáticas

Exemplo

```
void imprime ( float a )
{
    static int n = 1;
    printf ( " %f ", a);
    if ( ( n % 5 ) == 0 ) printf( " \n " );
    n++;
}
```

Exemplo de um algoritmo usando uma variável estática, que é atualizada a cada execução, e utilizada pela execução posterior.

Exercícios

- Projetar o algoritmo e implementar em C:
 1. Cálculo do valor de $\pi = 4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + \dots)$
 2. Geração de n pontos (pares real) no plano, calcular centroide, e transladar origem dos ponto para esse centroide.