

Linguagens de Programação e Compiladores

Programação Modular

José Osvano da Silva, PMP, PSM I

Sumário

- › Programação Modular;
- › Introdução;
- › Subprogramas;
- › Funções;
- › Exercício.

Introdução: Principal Objetivo.

Facilitar a solução de problemas complexos.

“A arte de programar consiste na arte de organizar e dominar a complexidade dos sistemas”

Dijkstra, 1972

Baseada na estratégia: Dividir para conquistar

- › **Divisão de um problema original em subproblemas (módulos);**
- › **mais fáceis de resolver e transformáveis em trechos mais simples;**
- › **com poucos comandos (subprogramas).**

Subprogramas

- › Trechos de código independentes, com estrutura semelhante àquela de programas, mas executados somente quando chamados por outro(s) trecho(s) de código.
- › Devem executar UMA tarefa específica, muito bem identificada (conforme a programação estruturada).
- › Ao ser ativado um subprograma, o fluxo de execução desloca-se do fluxo principal para o subprograma. Concluída a execução do subprograma, o fluxo de execução retorna ao ponto imediatamente após onde ocorreu a chamada do subprograma.

Vantagens do uso de subprogramas:

- ◆ Maior controle sobre a complexidade.
- ◆ Estrutura lógica mais clara.
- ◆ Maior facilidade de depuração e teste, já que subprogramas podem ser testados separadamente.
- ◆ Possibilidade de reutilização de código.

Subprogramas em Linguagem C

Implementados através de FUNÇÕES

Funções

- Funções são segmentos de programa que executam uma determinada tarefa específica.
- Funções (também chamadas de rotinas, ou sub-programas) são a essência da **programação estruturada**.
- Ex: `sqrt()`, `strlen()`, etc.
- O programador também pode escrever suas próprias funções, chamadas de **funções de usuário**, que tem uma estrutura muito semelhante a um programa.

Forma geral da declaração de uma função

```
tipo_da_funcao  nome_da_função (lista_de_parâmetros)
{
    //declarações locais
    //comandos
}
```

- tipo_da_funcao:** o tipo de valor retornado pela função. Se não especificado, por falta a função é considerada como retornando um inteiro.

- nome_da_função:** nome da função conforme as regras do C

- lista_de_parâmetros:** tipo de cada parâmetro seguido de seu identificador, com vírgulas entre cada parâmetro. Mesmo se nenhum parâmetro for utilizado, os parênteses são obrigatórios. Os parâmetros da declaração da função são chamados de parâmetros formais.

Exemplos de cabeçalhos de funções

Ex.:

```
soma_valores (int valor1, int valor2) // por falta é inteira
```

```
void imprime_linhas(int num_lin)
```

```
void apresenta_menu ( )
```

```
float conv_dolar_para_reais(float dolar);
```

Funções void

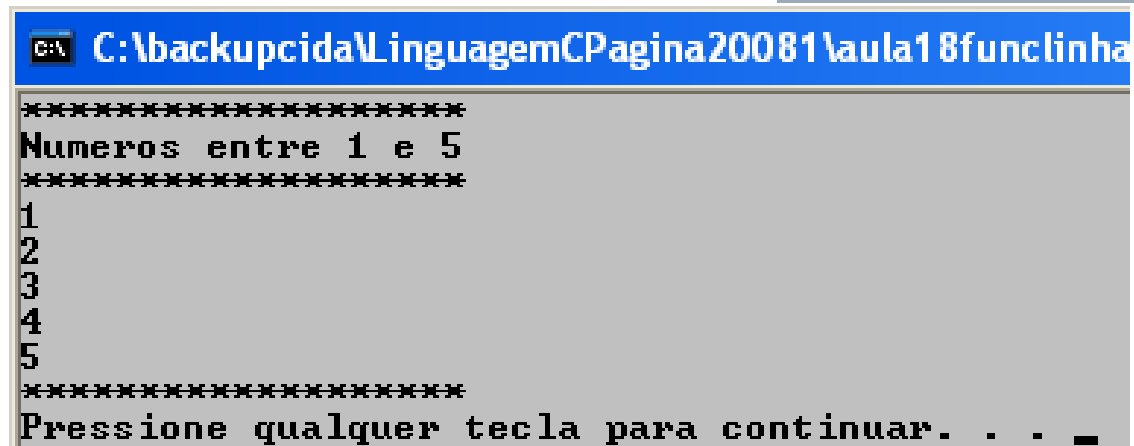
Void é um termo que indica ausência.
Em linguagem C é um tipo de dados.

Programa `escreveint` versão 1:

com linha de asteriscos produzida por trecho que se repete no código.

```
//Escrita de numeros inteiros
#include<stdio.h>
#include <stdlib.h>
main( )
{
    int i;
    system("color70")
    //apresentacao do cabecalho
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
    printf("Numeros entre 1 e 5\n");
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
    //escrita dos numeros
    for (i=1;i<=5;i++)
        printf("%d\n",i);
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
    system("pause");
}
```

Execução



```
C:\backupcida\LinguagemCPagina20081\aula18func\linha
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
Pressione qualquer tecla para continuar. . . _
```

A repetição de trechos de código idênticos em um programa pode ser um procedimento fácil e rápido, mas facilmente tende a produzir erros.

Tanto a manutenção quanto a alteração de programas com trechos repetidos tende a ser mais trabalhosa e sujeita a erros.

Com frequência alterações de trechos iguais que se repetem não são realizadas em todas as ocorrências do trecho ou são realizadas de forma incompleta em alguma ocorrência, com resultados bastante danosos.

A solução para esta questão são os **subprogramas**.

A seguir uma versão do programa **escreveint** onde as linhas de asterisco são produzidas pela função **apresente_linha**.

Programa **escreveint** versão 2:

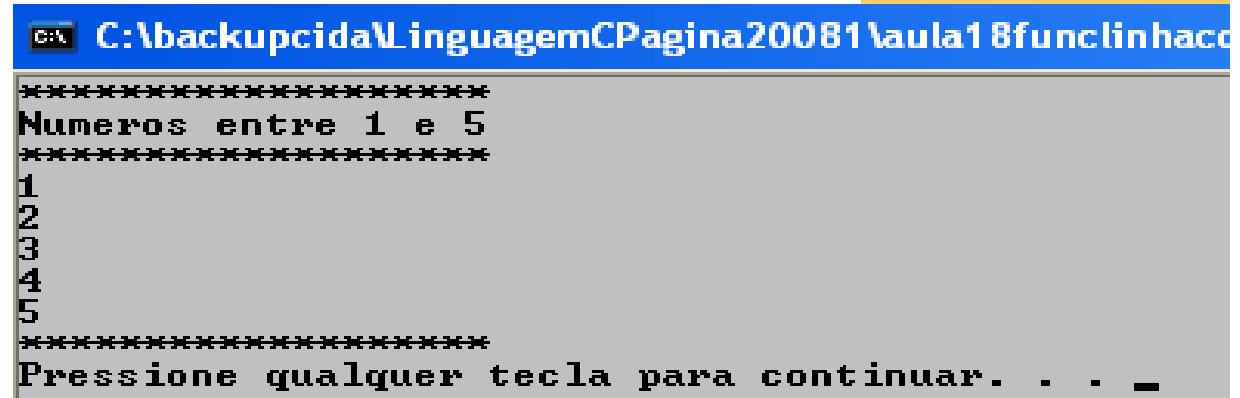
com uma função para apresentar linhas de asteriscos.

```
//escrita de numeros inteiros
#include<stdio.h>
#include <stdlib.h>
void apresente_linha(void);
main( )
{
    int i;
    system("color 70");
    //apresentacao do cabecalho
    apresente_linha( );
    printf("Numeros entre 1 e 5\n");
    apresente_linha( );
    // Escrita dos numeros
    for (i=1;i<=5;i++)
        printf("%d\n",i);
    apresente_linha( );
    system("pause");
}

void apresente_linha (void)
{
    int i;
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}
```

Chamadas à função **apresente_linha** substituem trechos repetidos

Execução



```
C:\backupcida\LinguagemCPagina20081\aula18func\linhaco
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
Pressione qualquer tecla para continuar. . . _
```

Chamadas à função `apresente_linha` substituem trechos repetidos

```
//escrita de numeros inteiros
```

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

```
#include <stdlib h>
```

```
void apresente_linha(void);
```

```
main( )
```

```
{
```

```
    int i;
```

```
    system("color 70");
```

```
    //apresentacao de cabecalho
```

```
    apresente_linha( );
```

```
    printf("Numeros entre 1 e 5\n");
```

```
    apresente_linha( );
```

```
    // Escrita dos numeros
```

```
    for (i=1;i<=5;i++)
```

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

```
    apresente_linha( );
```

```
    system("pause");
```

```
}
```

```
void apresente_linha (void)
```

```
{
```

```
    int i;
```

```
    for (i=1;i<20;i++)
```

```
        printf("*");
```

```
    printf("\n");
```

```
}
```

Protótipo da função `apresente_linha`

Chamadas da função `apresente_linha`

Execução

```
C:\backupcida\LinguagemCPagina20081\aula18funcinhaco
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
Pressione qualquer tecla para continuar. . . _
```

Declaração da função
`apresente_linha`:
escreve uma linha de asteriscos.

Cabeçalho da função `apresente_linha`

`void apresente_linha (void)`



Indica que a função não retorna valor no seu nome.



Indica que a função não tem parâmetros.

A função `apresente_linha` realiza sua tarefa sem receber nenhum valor do mundo externo à função, via parâmetros, e sem retornar nenhum valor no seu nome.

Seu tipo é `void` e seus parâmetros são `void`.

Execução de uma função

Em tempo de execução, ao ser encontrada uma chamada de uma função, a execução é desviada para o trecho de código da função.

A função é ativada e os itens locais à função são criados (os parâmetros por valor e os itens declarados internamente à função, como variáveis e constantes).

A função é executada até que seu término seja atingido.

Concluída a execução da função, todos os elementos locais à função que foram criados no momento de sua ativação são destruídos e a execução retorna ao fluxo principal, ao ponto imediatamente seguinte àquele onde ocorreu a chamada da função.

Variáveis locais

Os parâmetros que aparecem no cabeçalho das funções e as variáveis e constantes declaradas internamente a funções são locais à função.

Na função `apresente_linha`, o `i` é uma variável local a essa função.

```
void apresente_linha (void)
{
    int i;
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}
```

Variáveis e constantes locais:

Importante:

Recomenda-se fazer todas as declarações de uma função no seu início.

As variáveis e constantes declaradas em uma função são locais à função porque:

- só podem ser referenciadas por comandos que estão dentro da função em que foram declaradas;
- existem apenas enquanto a função em que foram declaradas está sendo executada. São criadas quando a função é ativada e são destruídas quando a função encerra.

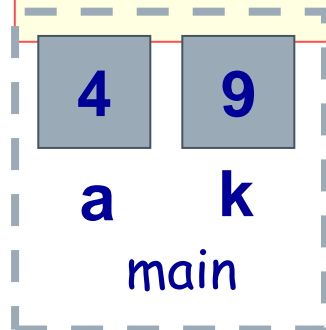
Criação e destruição de variáveis locais a uma função:

Programa principal

```
int a =4, k=9;
```

```
...  
execute x  
...
```

```
...  
execute x  
...
```



Subprograma X

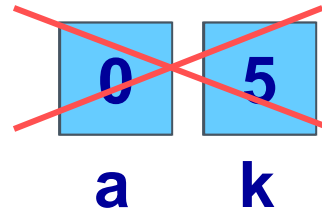
```
int a, k
```

```
...
```

```
a = 0; // local
```

```
k = 5; // local
```

```
...
```



Variáveis locais :

- uma função (inclusive a main) tem acesso somente às variáveis locais
- não altera valor de variáveis de outras funções

Criação e destruição de variáveis locais a uma função:

Programa principal

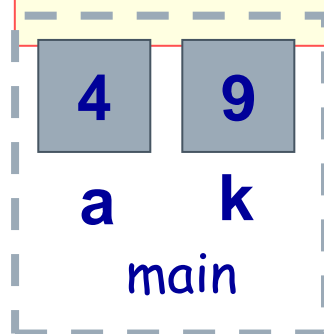
```
int a =4, k=9;
```

```
...  
execute x
```

```
...
```

```
...  
execute x
```

```
...
```



Subprograma X

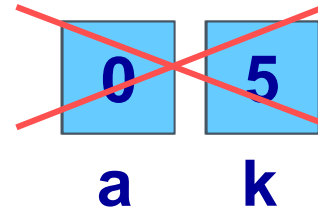
```
int a, k
```

```
...
```

```
a = 0; // local
```

```
k = 5; // local
```

```
...
```



ATENÇÃO:

Uma função (inclusive a *main*) tem acesso somente às suas variáveis locais.

Função *main*

```
main( )
{
    int i;

    //apresentacao do cabecalho

    apresente_linha( );

    printf("Numeros entre 1 e 5\n");

    apresente_linha( );

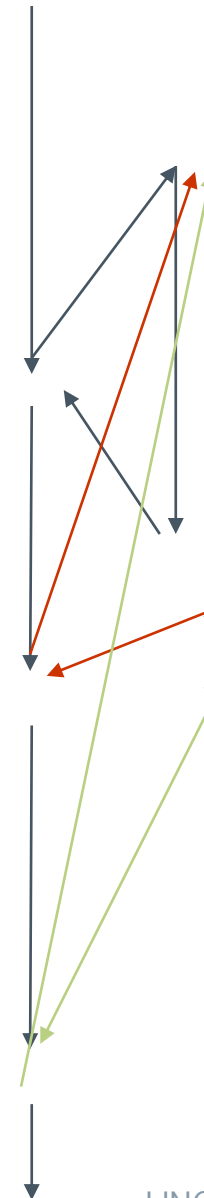
    //escrita dos numeros
    for (i=1;i<=5;i++)
        printf("%d\n",i);

    apresente_linha ( );
    system("pause");
}
```

Execução de *main* com chamadas à função *apresente_linha*

Função *apresente_linha*

```
void apresente_linha (void)
{
    int i;
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}
```



IMPORTANTE:

O *i* é local a *apresente_linha*. A cada nova execução de *apresente_linha* um novo *i* é criado e, ao final, destruído.

Funções de tipo void:

São ativadas como se fossem comandos.

Não ocorrem dentro de expressões.

Correspondem aos procedimentos de outras linguagens (Pascal, etc.).

Funções com tipo não void e com parâmetros

`sqrt`: função pré-definida

- › A seguir um programa que extrai a raiz quadrada de um número indeterminado de valores informados.
- › Para extrair a raiz quadrada dos valores é usada a função pré-definida `sqrt`, da biblioteca `math.h`.

sqrt: função pré-definida (cont.)

//extrai a raiz quadrada de valores informados

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

```
#include <stdlib.h>
```

```
#include <math.h>
```

```
main ( )
```

```
{
```

```
    int seguir;
```

```
    double valor;
```

```
    do
```

```
    {
```

```
        printf("\nValor para extrair raiz: ");
```

```
        scanf("%lf", &valor);
```

```
        printf ("\nRaiz quadrada de %6.2lf = %6.2lf\n", valor, sqrt(valor));
```

```
        printf("\nMais um valor, digite 1, para parar, digite 0: ");
```

```
        scanf("%d", &seguir);
```

```
    }
```

```
    while (seguir);
```

```
    system("pause");
```

```
}
```

sqrt: função pré-definida (cont.)

A função **sqrt** é do tipo **double**, isso significa que quando ela é chamada, no lugar de sua chamada retorna um valor **double**.

Para executar essa função é necessário fornecer um parâmetro, o valor para o qual se deseja que a raiz quadrada seja calculada.

No exemplo, está armazenado na variável **valor**.

`calc_produto`: função definida pelo usuário

- › A seguir um programa que calcula o produto de um número indeterminado de pares de valores informados.
- › Para calcular os produtos é usada a função definida pelo usuário `calc_produto`.

produto: função definida pelo usuário (cont.)

//calcula produtos de pares de valores informados

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

```
#include <stdlib.h>
```

```
int calc_produto(int, int);
```

```
main ( )
```

```
{
```

```
    int seguir;
```

```
    int oper1, oper2, produto;
```

```
    do
```

```
    {
```

```
        printf("\nOperando 1: ");
```

```
        scanf("%d", &oper1);
```

```
        printf("\nOperando 2: ");
```

```
        scanf("%d", &oper2);
```

```
        printf ("\nProduto = %d\n", calc_produto(oper1, oper2));
```

```
        printf("\nPara continuar, digite 1, para parar, digite 0: ");
```

```
        scanf("%d", &seguir);
```

```
    }
```

```
    while (seguir);
```

```
    system("pause");
```

```
}
```

```
int calc_produto(int valor1, int valor2)
```

```
{
```

```
    return valor1 * valor2;
```

```
}
```

produto: função definida pelo usuário (cont.)

A função `calc_produto` é do tipo `int`, isso significa que quando ela é chamada, no lugar de sua chamada retorna um valor `int`.

Para executar essa função é necessário fornecer dois parâmetros, os dois valores para cálculo do produto, `oper1` e `oper2`.

Comando `return()`: retorno de valor e fim lógico da função

O comando `return` **atribui valor** a função.
Ao ser executado, **encerra a execução** da função.

Ao ser executado o `return` na função `calc_produto`, um valor é atribuído à função e ela encerra sua execução.

No ponto onde ocorreu a chamada de `calc_produto`, um valor passa a estar disponível para processamento.

Comando `return()`: retorno de valor da função (cont.)

Se uma função é declarada com tipo diferente de void (int, char, float, etc.) significa que ela pretende explorar a possibilidade de retorno de um valor em seu nome, e então pode ser usada em expressões.

Uma função que retorna um valor em seu nome deve conter pelo menos uma ocorrência do comando `return`, uma vez que é pela execução de um comando `return` que um valor é atribuído ao nome de uma função.

Quando uma função encerra sua execução?

Uma função encerra sua execução quando:

- o fim do seu código é atingido;

ou

- um comando *return* é encontrado e executado.

Dúvidas



José Osvano da Silva
joseosvano@unipac.br