



Unidade 4

Modularização de Programa Funções

DEC0012 - Linguagem de Programação I
Curso de Engenharia de Computação
Profa. Andréa Sabedra Bordin

Introdução

- A maioria dos programas resolvem problemas muito maiores do que os programas desenvolvidos até agora.
- A melhor maneira de desenvolver e manter um programa grande é construí-lo a partir de **pequenas partes** ou componentes (**módulos**), sendo cada uma delas mais fácil de manipular que o programa original.
- Essa técnica é chamada **dividir para conquistar**.
- Uma das formas de modularizar é através da criação de **funções**. Uma **função** é um modulo (sequência de código) criado para resolver uma tarefa simples e bem definida.

Chamada de função

As funções são “**chamadas**” ou “**ativadas**” ao longo da execução do programa.

No momento da chamada é especificado:

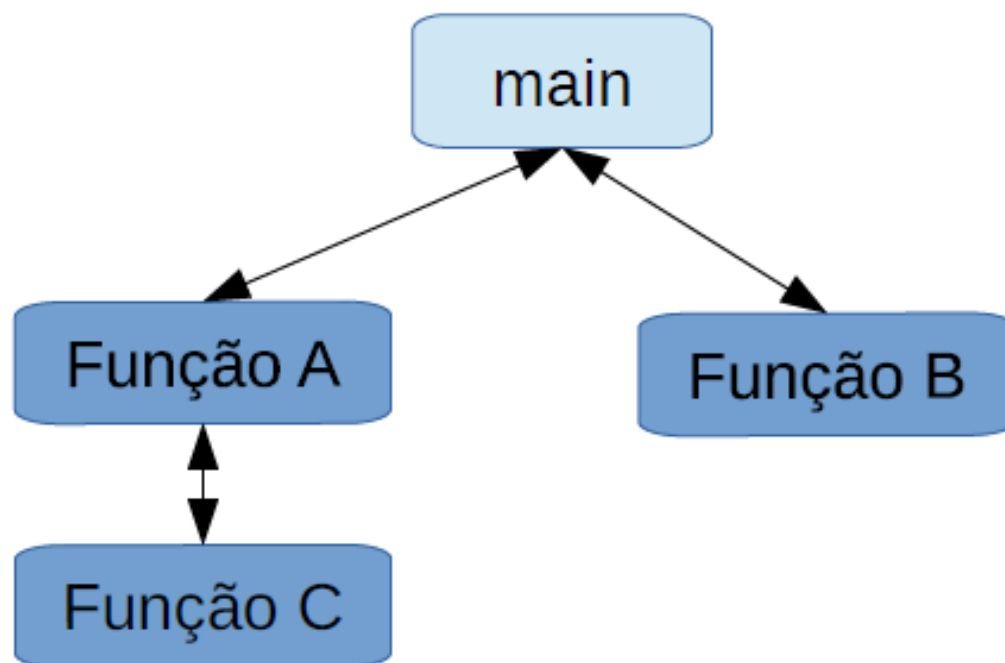
- o **nome da função**
- os **argumentos** – a informação que é necessária para que a função cumpra seu propósito.

Depois da execução a função retorna o resultado para o ponto em qual foi chamada.

Introdução

- Na linguagem C, a função **main()** será a sempre a primeira a ser executada e é a partir dela que serão feitas **chamadas** as outras funções
- As funções que podem ser chamadas são:
 - as funções **criadas pelo programador**;
 - as funções das **bibliotecas** disponíveis.
- As vantagens de uso das funções:
 - reutilização de software;
 - facilidade de desenvolvimento e manutenção.

Chamada de funções



Como criar uma função

Para criar uma função precisamos definir:

- **a declaração da função (ou cabeçalho)**
- **o corpo da função.**

Declaração:


tipo_retornado nome_da_funcao (lista dos parâmetros);

- o **tipo do dado resultado** produzido pela função;
- o **nome da função**;
- a **lista dos parâmetros** com a sequência com que esses parâmetros serão fornecidos.

A função pode não ter parâmetro(s)

Exemplo 1: Função max (retorna o maior de dois números)

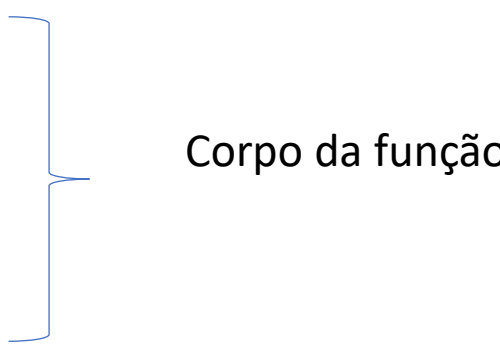
```
1  #include <stdio.h>
2
3  int max(int num1, int num2); // declaração de função
4  //=====
5  int main ()
6  {
7      /* variaveis locais */
8      int a, b;
9      int num_max;
10
11     printf("\n O programa retorna o maior de dois números\n");
12     printf("\n Digite 1o numero: ");
13     scanf("%d",&a);
14     printf("\n Digite 2o numero: ");
15     scanf("%d",&b);
16
17     num_max = max(a, b); /* chamada de função */
18
19     printf( "\n\n O maio número é : %d\n", num_max );
20
21     return 0;
22 }
23 //=====
24 /* a função retora o maximo de dois numeros */
25 int max(int num1, int num2)
26 {
27     /* variaveis locais */
28     int result;
29
30     if (num1 > num2)
31         result = num1;
32     else
33         result = num2;
34
35     return result;
36 }
```



Como criar uma função

O **corpo da função** pode ser definido separadamente da declaração e deve conter a informação sobre variáveis locais e os comandos de processamento propriamente ditos.

```
tipo_retornado nome_da_função (lista dos parâmetros)
{
declarações;
operadores;
return ;
}
```




Corpo da função

The diagram shows a blue bracket on the right side of the code block, grouping the lines from 'declarações;' to 'return ;'. To the right of the bracket is the text 'Corpo da função'.

Todas as variáveis usadas em uma função são consideradas locais – elas existem somente dentro daquela função.

Exemplo 1: Função **max** (retorna o maior de dois números)

```
1  #include <stdio.h>
2
3  int max(int num1, int num2); // declaração de função
4  //=====
5  int main ()
6  {
7      /* variaveis locais */
8      int a, b;
9      int num_max;
10
11     printf("\n O programa retorna o maior de dois números\n");
12     printf("\n Digite 1o numero: ");
13     scanf("%d",&a);
14     printf("\n Digite 2o numero: ");
15     scanf("%d",&b);
16
17     num_max = max(a, b); /* chamada de função */
18
19     printf( "\n\n O maio número é : %d\n", num_max );
20
21     return 0;
22 }
23 //=====
24 /* a função retora o maximo de dois numeros */
25 int max(int num1, int num2)
26 {
27     /* variaveis locais */
28     int result;
29
30     if (num1 > num2)
31         result = num1;
32     else
33         result = num2;
34
35     return result;
36 }
```



Parâmetros

- Uma função pode ou não receber **parâmetros** (valores) que são serão utilizados dentro do corpo da função.
- A lista de parâmetros é uma lista separada por vírgulas, contendo o **tipo** e o **nome dos parâmetros** (valores)
- recebidos pela função, quando ela é chamada.
- Isso deve ser feito junto à definição do corpo da função, mas os mesmos nomes podem aparecer na declaração da função (para ajudar na legibilidade do programa).

<code>int function (int x, int y);</code>	correto
<code>int function (int x, y);</code>	incorreto

Exemplo 1: Função **max** (retorna o maior de dois números)

```
1  #include <stdio.h>
2
3  int max(int num1, int num2); // declaração de função
4  //=====
5  int main ()
6  {
7      /* variaveis locais */
8      int a, b;
9      int num_max;
10
11     printf("\n O programa retorna o maior de dois números\n");
12     printf("\n Digite 1o numero: ");
13     scanf("%d",&a);
14     printf("\n Digite 2o numero: ");
15     scanf("%d",&b);
16
17     num_max = max(a, b); /* chamada de função */
18
19     printf( "\n\n O maio número é : %d\n", num_max );
20
21     return 0;
22 }
23 //=====
24 /* a função retora o maximo de dois numeros */
25 int max(int num1, int num2)
26 {
27     /* variaveis locais */
28     int result;
29
30     if (num1 > num2)
31         result = num1;
32     else
33         result = num2;
34
35     return result;
36 }
```

Argumentos

Argumentos são os valores passados no momento da chamada da função.

Exemplo 1: Função **max** (retorna o maior de dois números)

```
1  #include <stdio.h>
2
3  int max(int num1, int num2); // declaração de função
4  //=====
5  int main ()
6  {
7      /* variáveis locais */
8      int a, b;
9      int num_max;
10
11     printf("\n O programa retorna o maior de dois números\n");
12     printf("\n Digite 1o numero: ");
13     scanf("%d",&a);
14     printf("\n Digite 2o numero: ");
15     scanf("%d",&b);
16
17     num_max = max(a, b); /* chamada de função */
18
19     printf( "\n\n O maio número é : %d\n", num_max );
20
21     return 0;
22 }
23 //=====
24 /* a função retora o maximo de dois numeros */
25 int max(int num1, int num2)
26 {
27     /* variáveis locais */
28     int result;
29
30     if (num1 > num2)
31         result = num1;
32     else
33         result = num2;
34
35     return result;
36 }
```

Argumentos



Tipos de retorno de uma função

- Quando o **tipo de retorno** da função não é declarado, automaticamente assume-se que a função retornará um valor do tipo **int**.
- Se a função não deve retornar nenhum dado, o tipo de retorno deve ser declarado como **void**.

Exemplo:

function (int);	recebe um argumento do tipo int retorna um valor do tipo int
int function (int);	recebe um argumento do tipo int retorna um valor do tipo int
void function (int);	recebe um argumento do tipo int não retorna nenhum tipo de dados
void function();	não recebe nenhum argumento não retorna nenhum tipo de dados

Tipos de retorno de uma função

Há diferentes maneiras de retornar o controle para o ponto no qual uma função foi chamada, usando o operador:

return expressao;

Retorna o valor de expressão para a função que realizou a chamada, por exemplo:

return (a+b);

return a+1;

return;

Exemplo 2: Função square

```
1  #include <stdio.h>
2
3  int square(int num); // declaração de função
4  void printl();
5  //=====
6  int main ()
7  {
8      /* variaveis locais */
9      int i;
10
11     printf("\n 0 programa retorna n * n, [0..10] \n\n");
12
13     for(i = 0; i<=10; i++)
14     {
15         printf("%i elevado ao quadrado = %i", i, square(i));
16         printl();
17     }
18
19     return 0;
20 }
21 //=====
22 /* a função retora o num x num */
23 int square(int num)
24 {
25     return num * num;
26 }
27 //-----
28 /* a função desenha uma linha */
29 void printl()
30 {
31     printf("\n ----- \n\n");
32     return;
33 }
```

**Retorno da
função**

Funções das bibliotecas padrão

C/C++ oferece várias funções nas próprias bibliotecas padrão.

Para usar as funções disponíveis deve ser incluído no programa o nome da biblioteca específica.

Algumas das funções matemáticas disponíveis em **math.h**:

Função	Descrição	Exemplo
ceil (x)	arredonda o valor de x “pra cima”	ceil (9.2) é 10.0 ceil(-9.7) é -9.0
cos (x)	cosseno de x (x em radianos)	cos (0.0) é 1.0
exp (x)	função exponencial e	exp (1.0) é 2.71727
fabs (x)	valor absoluto de x	fabs(-7.76) é 7.76
floor (x)	arredonda x “pra baixo”	floor (9 .2) é 9 .0 floor(-9.7) é -10.0
fmod(x, y)	resto de x/y como número de ponto flutuante	fmod(13.0, 2.0) é 1 fmod(13.0, 2.1) é 0.4
log (x)	logaritmo natural de x (base e)	log (2.717272) é 1.0
pow(x, y)	x elevado à potência de y	pow(2, 7) é 128
sin (x)	seno de x (x em radianos)	sin (0.0) é 0
sqrt(x)	raiz quadrada de x	sqrt(900.0) é 30.0
tan (x)	tangente de x (x em radianos)	tan (0.0) é 0

Passagem de parâmetros por valor e por referência

- Na linguagem C, quando uma função é chamada, os parâmetros formais da função **copiam** os valores dos argumentos que são passados para a função.
- Isto quer dizer que **não são alterados** os valores das variáveis originais que foram passados para a função.
- Este tipo de chamada de função é denominada “**chamada por valor**”.
-

```
include <stdio.h>

float sqr (float num);

void main () {
    float num,sq;
    printf ("Entre com um numero: ");
    scanf ("%f",&num);
    sq = sqr(num);
    printf ("\n O numero original é: %f \n",num);
    printf ("O seu quadrado vale: %f \n",sq);
}

//retorna o quadrado de um número
float sqr (float num) {
    num=num*num;
    return num;
}
```

No exemplo acima o argumento num da função sqr() sofre alterações dentro da função, mas a variável num da função main() permanece inalterada: é uma chamada por valor.

Passagem de parâmetros por valor e por referência

- Outro tipo de passagem de parâmetros para uma função ocorre quando alterações nos parâmetros formais, dentro da função, alteram os valores das variáveis originais que foram passados para a função.
- Este tipo de chamada de função tem o nome de "**chamada por referência**".
- Este nome vem do fato de que, neste tipo de chamada, não se passa para a função os valores das variáveis, mas sim suas referências (a função usa as referências para alterar os valores das variáveis fora da função).

Passagem de Vetor e Matriz como Parâmetro

- Para passar um vetor ou matriz como parâmetro, basta declarar o parâmetro **da mesma forma** que a matriz/vetor foi declarada.
- Por definição da linguagem C, um vetor **é sempre passado por referência**, logo, qualquer alteração em seus elementos, altera a variável usada como argumento na chamada da rotina.

```
#include <stdio.h>

void imprimeVet (float vet[3])
{
    int i;
    for (i=0; i< 3; i++)
    {
        vet[i]=vet[i]+0.5;          7.7
        printf("%0.1f", vet[i]);  9.4
    }                               8.2
}

int main(){
    float notas[3]= {7.2,8.9,7.7};

    imprimeVet(notas); // Passa o vetor notas

    printf("Nota da posição 0 do vetor: %f ", notas[0]);
    printf("Nota da posição 1 do vetor: %f ", notas[1]);
    printf("Nota da posição 2 do vetor: %f ", notas[2]);

    return 0;
}
```

7.700000
9.400000
8.200000

Funções e escopo de variáveis

- Chama-se **escopo de variável** o conjunto de regras que determinam a utilização de uma variável em um programa.
- As variáveis são divididas, quanto ao escopo, em três tipos: **variáveis locais, parâmetros formais e variáveis globais.**
- **Variáveis locais**
 - São aquelas declaradas dentro do **bloco** de uma função.
 - *Um bloco começa quando abre-se uma chave e termina quando fecha-se a chave.*
 - Não podem ser usadas ou modificadas por outras funções.
 - Somente existem enquanto a função onde foi declarada estiver sendo executada.

Funções e escopo de variáveis

Parâmetros formais

- Os parâmetros formais de uma função também são variáveis locais da função.

Variáveis Globais

- São declaradas fora de todos os blocos de funções.
- São acessíveis em qualquer parte do programa, ou seja, podem ser usadas e modificadas por todas as outras funções.
- Existem durante toda a execução do programa.


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

```
#include <conio.h>
```

```
//declaração de variáveis globais
```

```
//----- Função main()-----
```

```
int main(void) {
```

```
//declaração das variáveis locais da main()
```

```
return(0);
```

```
}
```

```
//-----Função definida pelo programador-----
```

```
void funcao1(variáveis locais de parâmetros) {
```

```
// declaração das variáveis locais da função1
```

```
return;
```

```
}
```

```

void main () {
int a,x,y;
for (...) {
    float a,b,c;
    ...
}
...
}

func1 (...)
{
int abc,x;
...
}

func2 (...)
{
int abc;
...
}

```

Aqui temos três funções. As variáveis locais de cada uma delas não irão interferir com as variáveis locais de outras funções. Assim, a variável **abc** de **func1()** não tem nada a ver (e pode ser tratada independentemente) com a variável **abc** de **func2()**. A variável **x** de **func1()** é também completamente independente da variável **x** de **main()**. As variáveis **a**, **b** e **c** são locais ao bloco **for**. Isto quer dizer que só são conhecidas dentro deste bloco **for** e são desconhecidas no resto da função **main()**. Quando usarmos a variável **a** dentro do bloco **for** estaremos usando a variável **a** local ao **for** e não a variável **a** da função **main()**.