

Subprogramação

Programando com funções e procedimentos

**Declarando e chamando de funções e
procedimentos em linguagem C**

Relembrando Algoritmos

- Propriedades:
 - possui um estado inicial;
 - contém uma sequência lógica e finita de ações (comandos), claras e precisas, com fluxo de execução baseado em:
 - sequência;
 - seleção condicional (seleção de ações);
 - iteração (repetição de ações);
 - possui dados de entrada;
 - produz dados de saída corretos;
 - possui estado final previsível;
 - deve ser eficaz.

*Programação
Estruturada*

Programação Estruturada

- Desenvolvimento de algoritmos por fases ou refinamentos sucessivos
- Uso de um número muito limitado de estruturas de controle em casa fase
- Decomposição do algoritmo total em módulos desenvolvidos usando **subprogramas**

Dividir para conquistar!



Subprograma

Função ou Procedimento

- Objetivos principais:
 - evitar **repetição** de sequência de comandos;
 - dividir e estruturar um programa **em partes** fechadas e logicamente coerentes.

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

Dijkstra, 1972

Subprograma

Função ou Procedimento

- O que são?
 - **Sequência de instruções** executadas somente quando **chamadas** por um programa em execução
 - A execução de um programa em C corresponde a execução da função principal **main()**, **que é obrigatória**
 - Realiza **UMA tarefa específica**, muito bem identificada
 - Após sua execução, o fluxo do programa **retorna** ao ponto imediatamente após o da chamada da função (ou subprograma)

Subprograma

Função ou Procedimento

- **Funções pré-definidas**

- Disponibilizadas juntamente com os compiladores
- Incluídas no programa principal através de *headers*
- `#include <math.h>`

- **Funções desenvolvidas pelo programador**

- Um programa pode incluir diversas funções
- A declaração e implementação dessas funções fica a cargo do programador

Funções pré-definidas

- Exemplo usando a função `pow()` da biblioteca `math.h` para calcular a área de um círculo

```
1 // Calcula a área de um círculo, utilizando a função pow.
2 #include <stdio.h> // para uso de funções de entrada e saída
3 #include <math.h> // para o uso de funções aritméticas já disponíveis
4 #define PI 3.14159265359 // não existe esta constante em C
5
6 int main(){
7     float raio, area;
8     printf("Forneca o raio de um círculo: "); // função
9     scanf("%f", &raio); // função
10    area = PI * pow(raio, 2);
11    printf("Área do círculo de raio %6.2f eh %6.2f \n", raio, area);
12    return 0;
13 }
```

Outras funções disponíveis na biblioteca `math.h`

Nome	Exemplo	Descrição
<code>abs</code>	<code>abs(x)</code>	Valor absoluto
<code>ceil</code>	<code>ceil(x)</code>	Arredonda o número real para cima: <code>ceil(3.2)</code> é 4
<code>cos</code>	<code>cos(x)</code>	Cosseno de x (x em radianos)
<code>exp</code>	<code>exp(x)</code>	e elevado à potencia x
<code>fabs</code>	<code>fabs(x)</code>	Valor absoluto de x
<code>floor</code>	<code>floor(x)</code>	Arredonda o número real para baixo: <code>floor(3.9)</code> é 3
<code>log</code>	<code>log(x)</code>	Logaritmo natural de x
<code>pow</code>	<code>pow(x, y)</code>	Calcula x elevado à potência y
<code>sin</code>	<code>sin(x)</code>	Seno de x
<code>sqrt</code>	<code>sqrt(x)</code>	Raiz quadrada de x
<code>tan</code>	<code>tan(x)</code>	Tangente de x

Lista completa em:

<http://www.cplusplus.com/reference/cmath/>

Funções desenvolvidas pelo programador

- Objetivos e vantagens
 - Estrutura lógica mais clara
 - Solução de problemas complexos em partes (**dividir para conquistar**)
 - Maior facilidade de **depuração e teste** (subprogramas podem ser testados separadamente)
 - **Reuso** de trechos de programas que solucionam problemas recorrentes
 - **Evita repetição** de sequência de comandos (e erros decorrentes)

Funções desenvolvidas pelo programador

Como evitar repetição código?

```
1  /* Escrita de numeros inteiros */
2  #include <stdio.h>
3
4  int main(){
5      int i;
6      //apresentacao do cabecalho
7      for (i=0;i<20;i++)
8          printf("*");
9      printf("\n");
10
11     printf("Numeros entre 1 e 5\n");
12
13     for (i=0;i<20;i++)
14         printf("*");
15     printf("\n");
16
17     //escrita dos numeros
18     for (i=1;i<=5;i++)
19         printf("%d\n",i);
20
21     for (i=0;i<20;i++)
22         printf("*");
23     printf("\n");
24
25     return 0;
26 }
```

Problemas associados a repetições de código

- Repetição de trechos de código idênticos
 - Um procedimento fácil e rápido, mas que facilmente tende a **produzir e propagar erros**
 - **Manutenção e alteração** de programas com trechos repetidos é **trabalhosa** e sujeita a erros
 - Com frequência alterações de trechos repetidos não são realizadas em todas as ocorrências

Solução: Usar funções!

Argumentos e Retorno

- Uma função pode **receber** dados de entrada, utilizados localmente para executar os comandos incluídos na função: estes dados são chamados de **parâmetros ou argumentos**
- Uma função pode também **retornar** um valor, o que chamamos de **retorno de função**

Exemplos de chamada:



```
x = pow(3, 5); //Retorna 243.00 em x
y = cos(0);    //Retorna 1.00 em y
```

Argumentos e Retorno

- Funções podem ser de 4 tipos

	Sem Argumentos	Com Argumentos
Sem Retorno	Não recebem nada e não retornam nada. Ex: Imprimir uma informação fixa na tela	Recebem um ou mais argumentos e não retornam nada. Ex: Imprime na tela um resultado de uma computação
Com Retorno	Não recebem nada e retornam apenas um valor Ex: Ler um valor do teclado e retorna-lo	Recebem um ou mais argumentos e retornam apenas um valor. Ex: Realiza um cálculo a partir dos argumentos e retorna o resultado

Declaração e Chamada de Funções

- Declaração de função **sem retorno e sem argumentos**

Tipo de retorno Nome da função Argumentos

```
void linha(void) {  
    int i;  
    for (i=0; i<20; i++)  
        printf("*");  
    printf("\n");  
}
```

Variável local

Escopo da função

- Chamada da função declarada (a partir do main por exemplo)

Chamada da função pelo nome

```
linha();
```

Em uma função o tipo de retorno **void** indica que a função não tem retorno

Resolvendo o problema anterior sem repetição de código

```
1  /* Escrita de numeros inteiros */
2  #include <stdio.h>
3
4  void linha(void){
5      int i;
6      for (i=0;i<20;i++)
7          printf("*");
8      printf("\n");
9  }
10 int main(){
11     int i;
12     //apresentação do cabeçalho
13     linha();
14     printf("Numeros entre 1 e 5\n");
15     linha();
16
17     //escrita dos numeros
18     for (i=1;i<=5;i++)
19         printf("%d\n",i);
20
21     linha();
22     return 0;
23 }
```

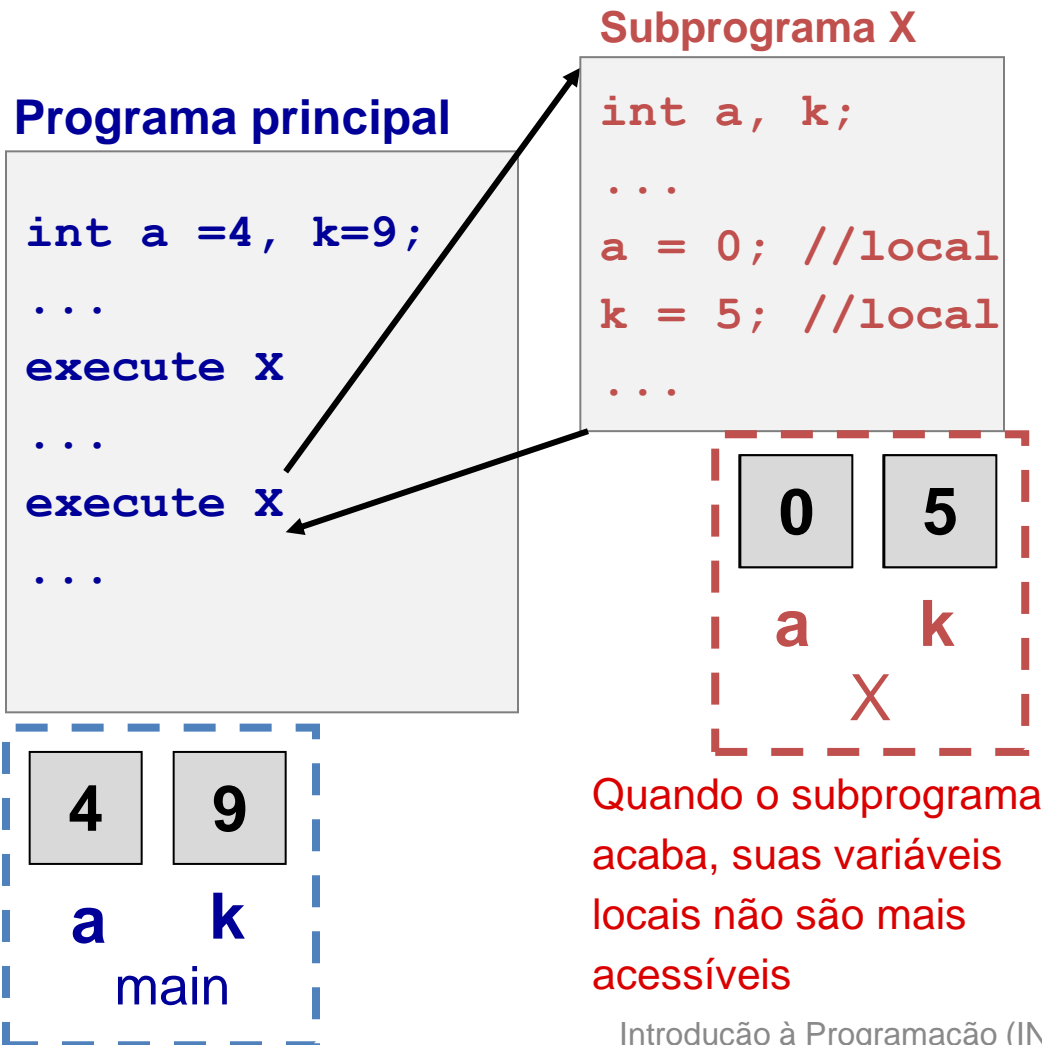
Variáveis locais

Três chamadas a mesma função

Note que o `main()` também é uma função declara e presente em todos os programas em C

```
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
Process returned 0 (0x0) execution ti
Press any key to continue.
```

Variáveis Locais



- Variáveis Locais
 - Uma função (inclusive a main) define somente variáveis locais
 - Existem apenas no **escopo da função** que as declarou
 - Não afetam os valores umas das outras, **mesmo que tenham o mesmo nome!**

Quando o subprograma acaba, suas variáveis locais não são mais acessíveis

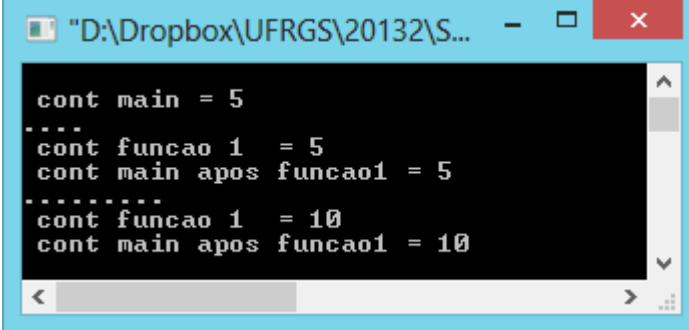
Variáveis

Locais vs Globais

- Locais
 - São declaradas **dentro do escopo** de uma função
 - Só podem ser referenciadas por comandos que estão **dentro do mesmo escopo** no qual elas foram declaradas
 - Não são reconhecidas fora da função onde foram declaradas (programa principal ou outras funções)
 - Existem apenas enquanto o bloco de código em que foram declaradas **está sendo executado**
- Globais
 - São declaradas **fora do escopo** de uma função
 - Por exemplo, logo após os comandos **#include** e **#define** no início do arquivo
 - Podem ser referenciadas dentro do **escopo de qualquer função**
 - Existem durante toda a execução do programa
 - Quando existe uma variável global e uma local **ativas no mesmo escopo**, a variável **local tem prioridade** sobre a global

Variáveis Locais e Globais (exemplo 1)

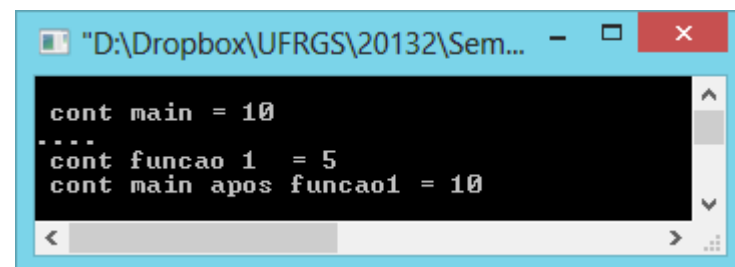
```
1 // Exemplo de programa para ilustrar uma variável global
2 #include <stdio.h>
3 int cont = 5; // variável global
4
5 void funcao1() {
6     int i;
7     for (i = 1; i < cont; i++)
8         printf(".");
9     printf("\n cont funcao 1 = %d", cont);
10 }
11
12 int main() {
13     printf("\n cont main = %d \n", cont);
14     funcao1();
15     printf("\n cont main apos funcao1 = %d \n", cont);
16     cont = 10;
17     funcao1();
18     printf("\n cont main apos funcao1 = %d \n", cont);
19     return 0;
20 }
```



```
cont main = 5
.....
cont funcao 1 = 5
cont main apos funcao1 = 5
.....
cont funcao 1 = 10
cont main apos funcao1 = 10
```

Variáveis Locais e Globais (exemplo 2)

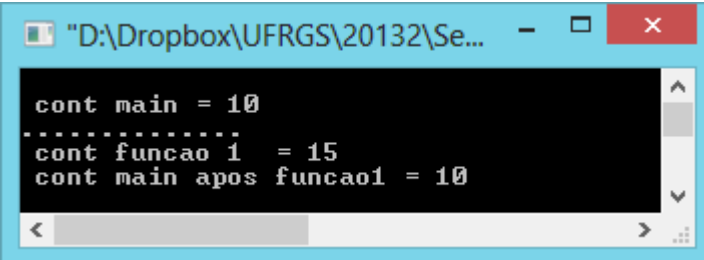
```
1  /* Outro exemplo de programa para ilustrar
2     uma variável global */
3  #include <stdio.h>
4  int cont = 5; // variável global
5
6  void funcao1(){
7      int i;
8      // reconhece a variável global
9      for (i = 1; i < cont; i++)
10         printf(".");
11     printf("\n cont funcao 1 = %d", cont);
12 }
13
14 int main(){
15     /* cont local, do programa principal.
16        Não altera a variável cont global,
17        quando acessada fora do main!! */
18     int cont = 10;
19     printf("\n cont main = %d \n", cont);
20     funcao1();
21     printf("\n cont main apos funcao1 = %d \n", cont);
22     return(0);
23 }
```



```
cont main = 10
....
cont funcao 1 = 5
cont main apos funcao1 = 10
```

Variáveis Locais e Globais (exemplo 3)

```
1 // Mais um exemplo ilustrando variáveis globais
2 #include <stdio.h>
3 int cont = 5; // variável global
4
5 void funcao1() {
6     int cont = 15, i; // cont local da funcao1
7     for (i=1; i < cont; i++)
8         printf(".");
9     printf("\n cont funcao 1 = %d", cont);
10 }
11
12 int main() {
13     // cont local do programa principal
14     int cont = 10;
15     printf("\n cont main = %d \n", cont);
16     funcao1();
17     printf("\n cont main apos funcao1 = %d \n", cont);
18     return(0);
19 }
```



```
cont main = 10
.....
cont funcao 1 = 15
cont main apos funcao1 = 10
```

Variáveis Locais e Globais (exemplo 4)

O que aconteceria?
Programa não compila!

```
1  #include <stdio.h>
2  // cont global foi excluída!!!
3  void funcao1() {
4      int i; // declaração cont local também foi excluída
5      for (i = 1; i < cont; i++)
6          printf(".");
7      printf("\n cont funcao 1 = %d", cont);
8  }
9
10 int main() {
11     // cont local do programa principal = não é global
12     int cont = 10, x;
13     printf("\n cont main = %d \n", cont);
14     funcao1( );
15     printf("\n cont main apos funcao1 = %d \n", cont);
16     return(0);
17 }
```

Ainda sobre variáveis globais e locais...

- Em funções procure utilizar preferencialmente **variáveis locais**
 - Evita confusão ao controlar os escopos
 - Economiza recursos do computador
- Troca de dados entre subprogramas deve ser feita somente através de **argumentos e retorno** de funções!

Evite utilizar variáveis globais!!!!



Declaração de Função com Protótipo

- Se uma chamada de função **aparecer antes da sua declaração** é preciso declarar um **protótipo** no começo do programa

Protótipo
da função

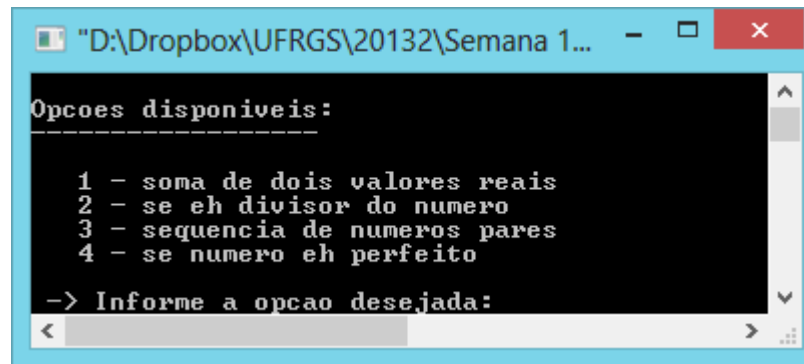
Chamada
da função

Declaração
da função

```
1  #include <stdio.h>
2
3  void linha(void); // protótipo
4
5  int main() { // função principal
6      /* ... */
7      linha( );
8      /* ... */
9  }
10
11  void linha(void) { // declaração da função
12      // preenche linha com 19 asteriscos
13      int i; // variável local
14      for (i=1; i<20; i++)
15          printf("*");
16      printf("\n");
17  }
```

Sugestão Exercício

- Escreva uma função *void*, de nome **menu_de_opcoes**, que gere a tela abaixo, sem incluir a leitura da opção informada:



```
"D:\Dropbox\UFRGS\20132\Semana 1..."  
Opcoes disponiveis:  
-----  
1 - soma de dois valores reais  
2 - se eh divisor do numero  
3 - sequencia de numeros pares  
4 - se numero eh perfeito  
-> Informe a opcao desejada:
```

- No programa principal, inclua a chamada desta função e depois faça o tratamento da entrada do usuário

Argumentos e Retorno

Vimos até agora funções sem argumentos e sem retorno...

	Sem Argumentos	Com Argumentos
Sem Retorno	<p>Não recebem nada e não retornam nada.</p> <p>Ex: Imprimir uma informação fixa na tela</p>	<p>Recebem um ou mais argumentos e não retornam nada.</p> <p>Ex: Imprime na tela um resultado de uma computação</p>
Com Retorno	<p>Não recebem nada e retornam apenas um valor</p> <p>Ex: Ler um valor do teclado e retorna-lo</p>	<p>Recebem um ou mais argumentos e retornam apenas um valor.</p> <p>Ex: Realiza um cálculo a partir dos argumentos e retorna o resultado</p>

Argumentos e Retorno

Agora veremos como passar argumentos para funções

	Sem Argumentos	Com Argumentos
Sem Retorno	Não recebem nada e não retornam nada. Ex: Imprimir uma informação fixa na tela	Recebem um ou mais argumentos e não retornam nada. Ex: Imprime na tela um resultado de uma computação
Com Retorno	Não recebem nada e retornam apenas um valor Ex: Ler um valor do teclado e retorna-lo	Recebem um ou mais argumentos e retornam apenas um valor. Ex: Realiza um cálculo a partir dos argumentos e retorna o resultado

Declaração de funções com argumentos e sem retorno

- Declaração de função **sem retorno e com argumentos**

Tipo de retorno

Nome da função

Argumentos

```
void linha(int n) {  
    int i;  
    for (i=0; i<n; i++)  
        printf("*");  
    printf("\n");  
}
```

Nome do argumento

Tipo do argumento

- Chamada da função declarada (a partir do main por exemplo)

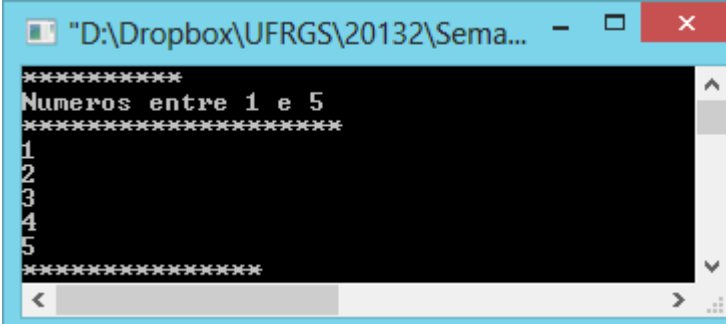
Chamada da função pelo nome

```
linha(20);
```

O valor 20 será atribuído
ao argumento n

Estendendo o exemplo anterior

```
1  /* Escrita de numeros inteiros */
2  #include <stdio.h>
3
4  void linha(int n){
5      int i;
6      for (i=0;i<n;i++)
7          printf("*");
8      printf("\n");
9  }
10 int main(){
11     int i;
12     //apresentacao do cabecalho
13     linha(10);
14     printf("Numeros entre 1 e 5\n");
15     linha(20);
16
17     //escrita dos numeros
18     for (i=1;i<=5;i++)
19         printf("%d\n",i);
20
21     linha(15);
22     return 0;
23 }
```

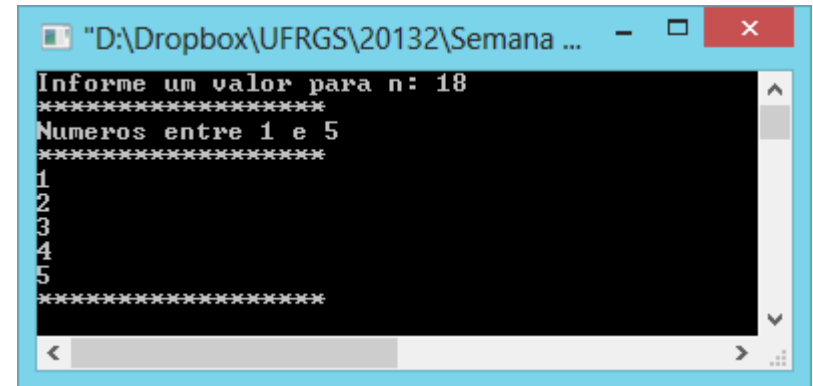


```
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
```

Três chamadas a mesma função com argumentos diferentes vão gerar resultados diferentes

Mesmo exemplo, agora o usuário informa o valor de n

```
1  /* Escrita de numeros inteiros */
2  #include <stdio.h>
3
4  void linha(int n){
5      int i;
6      for (i=0;i<n;i++)
7          printf("*");
8      printf("\n");
9  }
10 int main(){
11     int i, n;
12     printf("Informe um valor para n: ");
13     scanf("%d", &n);
14     //apresentacao do cabecalho
15     linha(n);
16     printf("Numeros entre 1 e 5\n");
17     linha(n);
18
19     //escrita dos numeros
20     for (i=1;i<=5;i++)
21         printf("%d\n",i);
22
23     linha(n);
24     return 0;
25 }
```



```
"D:\Dropbox\UFRGS\20132\Semana ... - [X]
Informe um valor para n: 18
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
```

Note que **n** no **main** é uma variável local enquanto que **n** na função **linha** é um parâmetro. Apesar de terem o mesmo nome são duas variáveis em escopos diferentes. Isto é, **alterar o valor de n dentro da função linha não altera o valor de n no main.**

Mais uma extensão da função linha

- Passando dois argumentos para a função

Tipo de retorno

Nome da função

Argumentos

```
void linha(int n, char c) {  
    int i;  
    for (i=0; i<n; i++)  
        printf("%c", c);  
    printf("\n");  
}
```

Dois argumentos
de tipos diferentes

- Chamada da função declarada (a partir do main por exemplo)

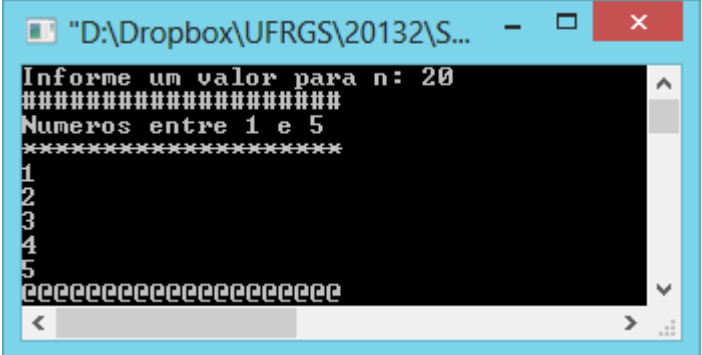
Chamada da função pelo nome

```
linha(20, '#');
```

Os argumentos são passados considerando os seus tipos e na mesma ordem que foram declarados

Exemplo completo com dois argumentos

```
1  /* Escrita de numeros inteiros */
2  #include <stdio.h>
3
4  void linha(int n, char c){
5      int i;
6      for (i=0;i<n;i++)
7          printf("%c", c);
8      printf("\n");
9  }
10 int main(){
11     int i, n;
12     printf("Informe um valor para n: ");
13     scanf("%d", &n);
14     //apresentacao do cabecalho
15     linha(n, '#');
16     printf("Numeros entre 1 e 5\n");
17     linha(n, '*');
18
19     //escrita dos numeros
20     for (i=1;i<=5;i++)
21         printf("%d\n",i);
22
23     linha(n, '@');
24     return 0;
25 }
```



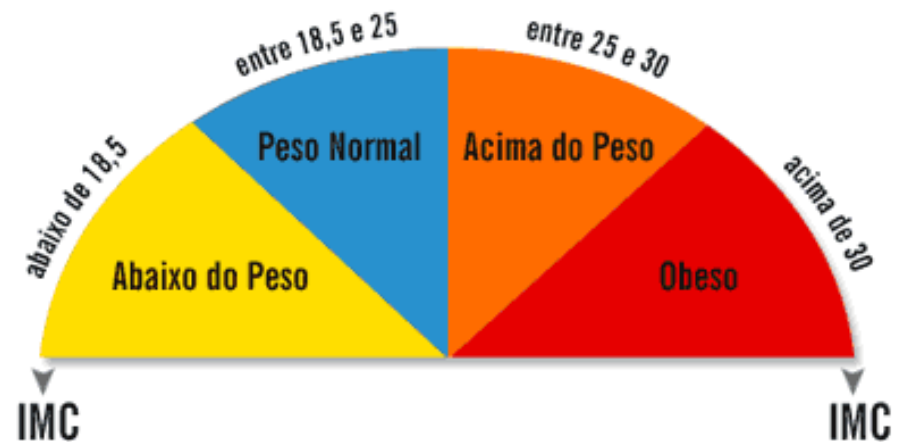
```
"D:\Dropbox\UFRGS\20132\S... - [X]
Informe um valor para n: 20
#####
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
```

O caractere também poderia ser um valor informado pelo usuário

Um outro exemplo

- Fazer uma função para calcular o Índice de Massa Corpórea (IMC)
 - Receber dois argumentos:
 - o peso em quilos (Kg);
 - a altura em metros (m);

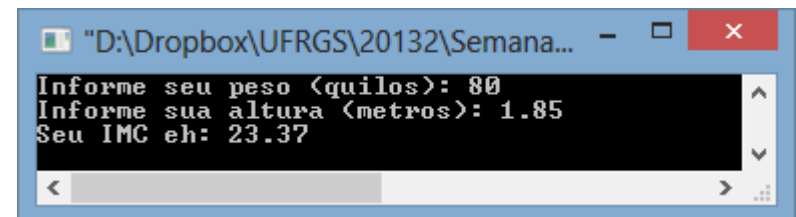
$$- IMC = \frac{peso}{altura^2}$$



Índice de Massa Corpórea (IMC)

```
1  /* Escrita de numeros inteiros */
2  #include <stdio.h>
3  #include <math.h>
4  void imc(float peso, float altura){
5      float imc;
6      imc = peso / pow(altura, 2);
7      printf("%.2f\n", imc);
8  }
9  int main(){
10     float p, a;
11     printf("Informe seu peso (quilos): ");
12     scanf("%f", &p);
13     printf("Informe sua altura (metros): ");
14     scanf("%f", &a);
15     printf("Seu IMC eh: ");
16     imc(p, a);
17     return 0;
18 }
```

Note que usamos uma chamada de função pré-definida dentro de uma função escrita pelo programador



The screenshot shows a terminal window with the following text:

```
"D:\Dropbox\UFRGS\20132\Semana..."
Informe seu peso <quilos>: 80
Informe sua altura <metros>: 1.85
Seu IMC eh: 23.37
```

Comentários sobre a solução para o cálculo do IMC

- Veja que a função `imc` recebe como entrada **dois argumentos do tipo `float`**
- O IMC é calculado e impresso como um tipo `float` (`%f`)
- A variável `imc` utilizada na função é **local**, sendo assim não é acessível na função `main!!!`

Questão: como passar um valor calculado em uma função de volta para quem a chamou?

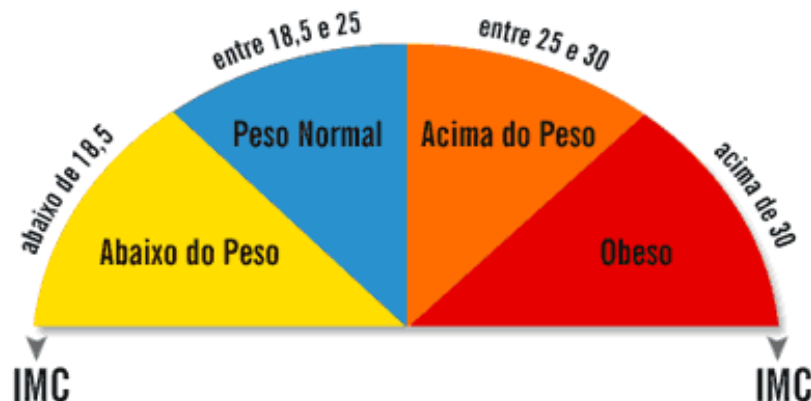
Argumentos e Retorno

Funções com retorno com ou sem argumentos

	Sem Argumentos	Com Argumentos
Sem Retorno	Não recebem nada e não retornam nada. Ex: Imprimir uma informação fixa na tela	Recebem um ou mais argumentos e não retornam nada. Ex: Imprime na tela um resultado de uma computação
Com Retorno	Não recebem nada e retornam apenas um valor Ex: Ler um valor do teclado e retorna-lo	Recebem um ou mais argumentos e retornam apenas um valor. Ex: Realiza um cálculo a partir dos argumentos e retorna o resultado

Retomando o exemplo do IMC

- Fazer uma função para calcular o Índice de Massa Corpórea (IMC)
 - Receber dois argumentos:
 - o peso em quilos (Kg);
 - a altura em metros (m);
 - $IMC = \frac{peso}{altura^2}$
 - **Retornar o valor do IMC**
- No main, apresentar a mensagem apropriada de acordo com o valor retornado



Declaração de funções com retorno, com ou sem argumentos

- Passando dois argumentos para a função

Tipo de retorno Nome da função Argumentos

```
float imc(float peso, float altura) {  
    float imc;  
    imc = peso / pow(altura, 2);  
    return imc;  
}
```

O tipo de retorno
indica o tipo de valor
retornado pela função

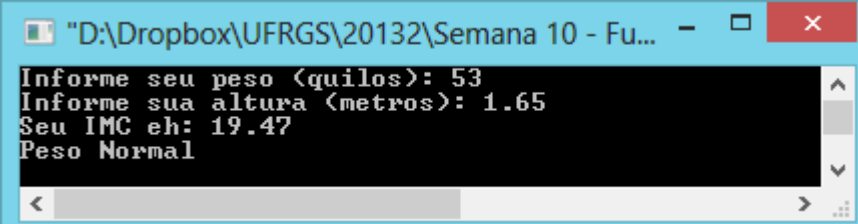
- Chamada da função declarada (a partir do main por exemplo)

```
float i;  
i = imc(65.0, 1.6);
```

Uma variável local do mesmo
tipo pode receber o valor
retornado pela função no main

Exemplo do IMC completo

```
1  /* Escrita de numeros inteiros */  
2  #include <stdio.h>  
3  #include <math.h>  
4  float imc(float peso, float altura){  
5      float imc;  
6      imc = peso / pow(altura, 2);  
7      return imc;  
8  }  
9  int main(){  
10     float p, a, i;  
11     printf("Informe seu peso (quilos): ");  
12     scanf("%f", &p);  
13     printf("Informe sua altura (metros): ");  
14     scanf("%f", &a);  
15     i = imc(p, a);  
16     printf("Seu IMC eh: %.2f\n", i);  
17     if (i < 18.5f){  
18         printf("Abaixo do Peso\n");  
19     }else if(i <= 25.0f){  
20         printf("Peso Normal\n");  
21     }else if(i <= 30.0f){  
22         printf("Acima do Peso\n");  
23     }else{  
24         printf("Obeso\n");  
25     }  
26     return 0;  
27 }
```



```
D:\Dropbox\UFRGS\20132\Semana 10 - Fu...  
Informe seu peso (quilos): 53  
Informe sua altura (metros): 1.65  
Seu IMC eh: 19.47  
Peso Normal
```

Observações Sobre Funções em C

- Todo programa em C tem pelo menos uma função: a função **main**
- Um argumento passado para uma função funciona da mesma forma que uma variável local à função
- Os nomes dos argumentos na utilização na declaração de uma função são independentes dos nomes das variáveis usadas para chamar a função
- A quantidade e tipo dos argumentos usados para chamar uma função deve ser igual ao número e tipo dos argumentos na declaração da função

Observações Sobre Funções em C

- Qualquer tipo de dado pode ser tipo de um argumento de uma função
- Qualquer **expressão** válida em C pode ser argumento para uma função
- Funções podem chamar outras funções
 - Ao final da execução de uma função o controle retorna a função que chamou
 - Uma função pode chamar ela mesma, chamamos isso de recursividade
- Uma função com retorno pode ser utilizada para atribuir valor a uma variável local ou diretamente em uma expressão

```
i = imc(65.0, 172);
```

```
x = 6 * pow(2, 4);
```


Dicas sobre funções

- Lendo declarações de funções pré-definidas
 - Identificar, tipo de retorno e argumentos
 - Não importa saber como internamente a função é implementada
- Tente ler algumas das documentações das bibliotecas que usamos

stdio.h - <http://www.cplusplus.com/reference/cstdio/>

stdlib - <http://www.cplusplus.com/reference/cstdlib/>

math.h <http://www.cplusplus.com/reference/cmath/>