# Linguagem de Programação C
# Aula 6

**Professor: Rodrigo Rocha**

# Conteúdo - Funções

- Funções
- Argumentos
- Retorno
- Passagem de parametros
  - Por valor
  - Por referência

# Definição

- Uma função é um bloco de código de programa que pode ser usado diversas vezes em sua execução.
- O uso de funções permite que o programa fique mais legível, mais bem estruturado.
- Um programa em C pode consistir de várias funções colocadas juntas

# Declaração

- *Formato geral:*

*<tipo_de_retorno> <nome_da_função>(lista_de_argumentos) {*
*código_da_função*
*}*

- Com relação ao local de declaração de uma função, ela deve ser definida ou declarada antes de ser utilizada, ou seja, antes da cláusula **main**.

- Caso queira declarar uma função depois da função main, deve declarar a assinatura da função em questão.
  - EX: *<tipo_de_retorno> nome_da_função>(lista_de_argumentos);*

- Mesmo se não houver parâmetros na função, os parênteses ainda são necessários.

# Função – Exemplo
# Definindo a função antes do main

```
#include <stdio.h>

/* Função simples: imprime Ola! */
int mensagem() {
  printf ("Ola!");
  return(0);
}
// Função principal: corpo do programa */
int main() {
  mensagem();
  printf("Eu estou vivo!\n");
  return(0);
}

//Não precisa declarar o protótipo/assinatura
  da função!
```

# Função – Exemplo
# Definindo a função depois do main

```
#include <stdio.h>

int mensagem(); //declaração do
  protótipo/assinatura da função

// Função principal: corpo do programa */
int main() {
  mensagem();
  printf("Eu estou vivo!\n");
  return(0);
}

/* Função simples: imprime Ola! */
int mensagem() {
  printf ("Ola!");
  return(0);
}
```

# Escopo de variável

- Pode-se definir variável dentro de uma função
- Uma variável definida em uma função não é vista por outra, a não ser que seja passada como argumento
- Variáveis globais podem ser lidas e alteradas dentro de uma função
- Na boa técnica de programação procura-se não se utilizar variáveis globais

# Argumentos

- Argumentos são as entradas que a função recebe.
- É através dos argumentos que passamos *parâmetros* para a função.
- Argumentos passados como parâmetros não podem ter seus valores alterados dentro das funções.
- As funções printf() e scanf() são exemplos de funções que recebem argumentos.

# Argumentos - Exemplo

```
#include <stdio.h>

int square (int x) /* Calcula o quadrado de x
  */  {
   printf ("O quadrado e %d",(x*x));
   return(0);
}

int main() {
   int num;
   printf("Entre com um numero: ");
   scanf("%d",&num);
   printf ("\n\n");
   square(num);
   return(0);
}
```

# Retornando valores

- Muitas vezes é necessário fazer com que uma função retorne um valor.
- Pode-se especificar um tipo de retorno indicando-o antes do nome da função.
- Para dizer ao C *o que* vai ser retornado é utilizado a palavra reservada **return()**.
- Toda função tipada tem que ter pelo menos um comando return()
- O valor do retorno tem que ser do mesmo tipo definido na função
- O comando return interrompe o funcionamento da função

# Retornando valores - Exemplo

```
int prod (int x, int y) {
   return (x*y);
}

int main () {
   int saida;
   saida=prod(12, 7);
   printf("A saida e: %d\n",saida);
   return(0);
}
```

# Passagem por valor ou referência

- O C não permite que um argumento passado como parâmetro de uma função seja alterado dentro da mesma
- O valor de cada argumento é copiado para uma variável interna à função e não é retornado ao fim da mesma
- Para alterar o valor de um argumento é necessário passá-lo como endereço de memória (**&**)
- Não é possível alterar o endereço de memória passado como parâmetro, mas pode-se alterar o conteúdo da memória apontado pelo mesmo

# Passagem por valor

- O valor do argumento passado para uma função é copiado em outra variável de escopo válido apenas para a função
- Caso se altera o valor da variável parâmetro não irá refletir no valor original da chamada
- Pode-se passar como parâmetro uma variável, uma constante ou o resultado de uma expressão
- É a forma padrão (default) de passagem de parâmetro

# Passagem por valor - Exemplo

```
Void imprime(int qtd) {
   while (qtd>0) {
      printf("Teste (%i)", qtd);
      qtd=qtd-1;
   }
}

Void main() {
   int qtd=5;
   printf("antes: %i", qtd);
   imprime(qtd);
   printf("depois: %i", qtd);
   imprime(10);
   imprime(qtd + 5);
}
```

# Passagem por referência

- O argumento passado por referência deve ser uma variável e deve-se utilizar o modificador **&** antes da mesma para indicar que está se passando o endereço de memória
- A função que receberá o argumento deverá definir a variável com o modificador * na frente da variável indicando que receberá um endereço
- Dentro a função, a variável passada como parâmetro é um endereço
- Para acessar ou alterar o conteúdo da mesma deverá-se colocar o modificador * antes da variável
- A função scan recebe o endereço da variável idade, efetua a leitura e altera o conteúdo do endereço, e não o endereço

```
scanf("%i", &idade);
```

# Passagem por referência - Exemplo

```
void imprime(int *qtd) {
    *qtd=*qtd-1;
}

void main() {
  int qtd=5;
  printf("antes: %i\n",qtd);
  imprime(&qtd);
  printf("depois: %i\n", qtd);
}
```

# Exercícios

- Dada a sequência: 1 1 2 3 5 8 13 ... (fibonacci).

Implemente uma função para calcular:

- a) Os n primeiros termos da sequencia

Outra para:

- b) A soma dos s primeiros termos, s <= n

Outra para:

- c) O produto dos p últimos termos, p <= n