# ESTRUTURA DE DADOS E ALGORITMOS I

## Modularização – parte I

Profº. Sérgio Roberto Costa Vieira, M.Sc.
Cursos de Computação
2º. Período

# Estrutura de Dados e Algoritmos I

## Modularização

- A modularização se baseia na divisão de um problema em partes, isto é, divide um problema em problemas menores.
- Podem ser chamados de módulos ou funções (sub-rotinas) menores que facilitam a depuração do programa.
- Na linguagem C, todas as ações ou rotinas de um programa ocorrem dentro de funções.
- main() é a função principal de qualquer programa em C e sempre será a primeira função a ser executada.
- É o ponto de partida em linguagem C.

# Estrutura de Dados e Algoritmos I

## Modularização

- Uma sub-rotina é um conjunto de comandos agrupados em um bloco que recebe um nome.
- É através deste, que pode ser ativado ou chamado, podendo ser utilizado diversas vezes em sua execução.
- Basicamente, uma sub-rotina recebe informações, processa e se necessário retorna a informação modificada.
- É um bloco de código que pode ser nomeado ou chamado de dentro de um programa.

# Estrutura de Dados e Algoritmos I

## Modularização

- A linguagem C possui muitas sub-rotinas já implementadas, algumas já usamos bastante.
- Um exemplo são as sub-rotinas básicas de entrada e saída: scanf() e printf()
- O programador não precisa saber o código contido dentro das sub-rotinas ou funções de entrada e saída para utilizá-las.
- Basta saber seu nome e como utilizá-las.

## Por que usar Funções ou Sub-rotinas?

- Para evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa.
- Para permitir o reaproveitamento do código já construído.
- Para tornar mais rápida a alteração de um trecho de código.
- Para que os blocos de programas não fiquem grandes demais, e mais difíceis de entender.
- Para facilitar a leitura do código-fonte.

## Declarando uma Função

- Em linguagem C, a declaração de uma função pelo programador segue esta forma geral:

```
tipo_retornado  nome_função (lista_de_parâmetros) {

  sequência de declarações e comandos;

}
```

- O nome_função é como aquele trecho de código será conhecido dentro do programa, segue as mesmas regras de variáveis.

# Estrutura de Dados e Algoritmos I

## Modularização

### Local de declaração de uma função

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

int Square(int  a){
    return (a * a);
}

int main( ){
    int n1, n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);
    n2 = Square(n1);
    printf("O seu quadrado vale: %d \n", n2);
    system("pause");
    return 0;
}
```

***A sub-rotina ou função pode ser definida ou declarada antes de ser utilizada, ou seja, antes da cláusula main()***

# Estrutura de Dados e Algoritmos I

## Modularização

### Local de declaração de uma função

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

int Square(int  a);

int main( ){
    int n1, n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);
    n2 = Square(n1);
    printf("O seu quadrado vale: %d \n", n2);
    system("pause");
    return 0;
}
int Square(int  a){
    return (a * a);
}
```

*Nesse caso é preciso declarar antes o protótipo da função.*

*Pode-se também declarar uma função depois da cláusula main()*

## Declarando uma Função

- O protótipo de uma função é a declaração de função que omite o corpo, mas especifica o seu nome, tipo de retorno e lista de parâmetros.
- O protótipo de uma função não precisa incluir os nomes das variáveis passadas como parâmetros.
- Apenas os seus tipos já são suficientes:
  - `int Square(int a);`
  - `int Square(int );`

## Funcionamento de uma função

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

int Square(int  a){
    return (a * a);
}

int main( ){
    int n1, n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);
    n2 = Square(n1);
    printf("O seu quadrado vale: %d \n", n2);
    system("pause");
    return 0;
}
```

***O código do programa é executado até encontrar uma chamada de função.***

## Funcionamento de uma função

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

int Square(int  a){
    return (a * a);
}

int main( ){
    int n1, n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);
    n2 = Square(n1);
    printf("O seu quadrado vale: %d \n", n2);
    system("pause");
    return 0;
}
```

***O programa é interrompido temporariamente, e o fluxo do programa passa para a função chamada.***

## Funcionamento de uma função

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

int Square(int  a){
    return (a * a);
}

int main( ){
    int n1, n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);
    n2 = Square(n1);
    printf("O seu quadrado vale: %d \n", n2);
    system("pause");
    return 0;
}
```

***Se houver parâmetros na função, os valores da chamada da função são copiados para os parâmetros no código da função .***

## Funcionamento de uma função

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

int Square(int  a){
    return (a * a);
}

int main( ){
    int n1, n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);
    n2 = Square(n1);
    printf("O seu quadrado vale: %d \n", n2);
    system("pause");
    return 0;
}
```

*Os comandos da função são executados .*

*Quando a função termina (seus comandos acabaram ou o comando return foi encontrado) o programa volta ao ponto em que foi interrompido.*

## Funcionamento de uma função

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

int Square(int  a){
    return (a * a);
}

int main( ){
    int n1, n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);
    n2 = Square(n1);
    printf("O seu quadrado vale: %d \n", n2);
    system("pause");
    return 0;
}
```

***Se houver um comando return, o valor dele será copiado para a variável que foi escolhida para receber o retorno da função.***

# Estrutura de Dados e Algoritmos I

## Modularização - Exemplo

### Funcionamento de uma função

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

float celsius(float  fahr){
    float c;
    c = (fahr – 32.0) *5.0 / 9.0;
    return  c;
}

int main( ){
    float c, f;
    printf(“Digite a temperatura em graus Fahrenheit: ”);
    scanf(“%f”, &f);
    c = celsius( f );
    printf(“Celsius: %.2f \n”, c);
    system(“pause”);
    return 0;
}
```

# Estrutura de Dados e Algoritmos I

## Modularização - Exemplo

### Funcionamento de uma função

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

float celsius(float  fahr);

int main( ){
     float c, f;
      printf("Digite a temperatura em graus Fahrenheit: ");
      scanf("%f", &f);
      c = celsius( f );
      printf("Celsius: %.2f \n", c);
      system("pause");
      return 0;
}
float celsius(float  fahr){
     float c;
      c = (fahr – 32.0) *5.0 / 9.0;
      return  c;
}
```

*O protótipo da função.*

# Estrutura de Dados e Algoritmos I

## Modularização - Exemplo

### Funcionamento de uma função

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

float celsius(float   );

int main( ){
     float c, f;
      printf("Digite a temperatura em graus Fahrenheit: ");
      scanf("%f", &f);
      c = celsius( f );
      printf("Celsius: %.2f \n", c);
      system("pause");
      return 0;
}
float celsius(float  fahr){
     float c;
      c = (fahr – 32.0) *5.0 / 9.0;
      return  c;
}
```

***O protótipo da função. Sem o nome da variável.***

## Parâmetros de uma Função

- Os parâmetros são o que o programador utiliza para passar informações de um trecho de código para dentro da função.
- Os parâmetros são uma lista de variáveis, separadas por vírgula, em que são especificadas o tipo e o nome de cada variável passada.
- Na declaração de parâmetros de uma função é necessário especificar o tipo para cada variável.

# Estrutura de Dados e Algoritmos I

## Modularização - Exemplo

### Parâmetros de uma função

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

int calculoSoma(int  x, int y){

      return  x + y;
}

int main( ){
     int a, b, soma;
      printf("Digite um número para A: ");
      scanf("%d", &a);
      printf("Digite um número para B: ");
      scanf("%d", &b);
      soma = calculoSoma( a, b );
      printf("A soma %c: %d \n", 130, soma);
      system("pause");
      return 0;
}
```

## Parâmetros de uma Função

- Declaração correta:

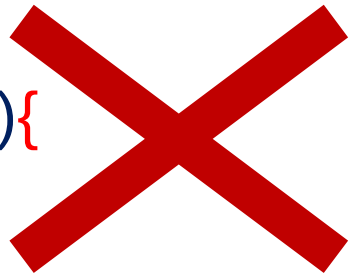
```
int calculoSoma(int  x, int y){

    return  x + y;
}
```

- Declaração incorreta:

```
int calculoSoma(int  x, y){

    return  x + y;
}
```



## Corpo da Função

- Pode-se dizer que o corpo de uma função é a sua alma. É no corpo de uma função que se define a tarefa que ela vai realizar quando for chamada.
- Basicamente, é formado por:
  - Sequência de declarações: variáveis, constantes, arrays etc.
  - Sequência de comandos: comandos condicionais, de repetição, chamada de outras funções etc.
- Todo programa possui ao menos uma função: a função main().

## Corpo da Função

```
int main( ){
    sequência de declarações e comandos
    return 0;
}
```

- É no corpo da função que as entradas (parâmetros ou argumentos) são processadas, as saídas são geradas ou outras ações são feitas.

## Corpo da Função

- Dessa forma, tudo o que temos dentro de uma função main() pode ser feito em uma função desenvolvida pelo programador.

- Uma função é construída com intuito de realizar uma tarefa específica e bem definida.

Por exemplo, uma função para calcular o fatorial de um número qualquer.

## Corpo da função

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

int main( ){
    int x, i, f = 1;
    printf("Digite um número inteiro positivo: ");
    scanf("%d", &x);
    for(i = 1; i <= x; i++){
            f = f * i ;
    }
    printf("O fatorial de %d %c: %d \n", x,130, f);
    system("pause");
    return 0;
}
```

***Corpo da função main() sem uso de sub - rotina.***

# Estrutura de Dados e Algoritmos I

## Modularização - Fatorial

### Corpo da função

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

int fatorial(int  n){
      int  i, f = 1;
      for(i = 1; i <= n; i++){
              f = f * i ;   }
      return f ;
}

int main( ){
     int x, fat;
      printf("Digite um número inteiro positivo: ");
      scanf("%d", &x);
      fat = fatorial( x );
      printf("O fatorial de %d %c: %d \n", x,130, fat);
      system("pause");
      return 0;
}
```

***Corpo da função main() com uso de sub - rotina.***

**De modo geral, evita-se fazer operações de leitura e escrita dentro de uma função.**

## Corpo da Função

- Uma função deve conter apenas o trecho de código responsável por fazer aquilo que é objetivo da função.
- Isso não impede que operações de leitura e escrita sejam utilizadas dentro dela.
- Elas só não devem ser usadas quando os valores podem ser passados para a função por meio de parâmetros.
- A seguir, apresenta-se um exemplo com um menu de opções para o usuário, que tem de escolher entre uma delas.

# Estrutura de Dados e Algoritmos I

## Modularização – Menu de Opções

### Corpo da função

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

int menu( ){
      int  i ;
     do{
          printf(“Escolha uma opção: \n”);
          printf(“(1) Opcao 1 \n (2) Opcao 2 \n (3) Opcao 3 \n”);
          scanf(“%d”, &i);
     }while((i < 1) || (i >3));
   return  i ;
}

int main( ){
     int op = menu();
     printf(“Voce escolheu a Opcao %d. \n”, op);
     system(“pause”);
     return 0;
}
```

***Exemplo de sub – rotina com leitura e escrita.***

## Retorno da Função

- O retorno da função é a maneira como uma função devolve o resultado (se ele existir) da sua execução para quem a chamou.

- Uma função pode retornar qualquer tipo válido na linguagem C:

  Básicos: int, char, float, double, **void** e ponteiros.

  Definidos pelo Programador: struct, array (indiretamente) etc.

## Retorno da Função (sem retorno de valor)

- O tipo void é conhecido como tipo vazio.
- Uma função declarada com o tipo void vai apenas executar um conjunto de comando e não devolverá nenhum valor para quem a chamar.

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

void  imprime( int  n ){
     int  i ;
     for( i = 1; i <= n; i++)
          printf("Linha %d  \n", i);
}

int  main( ){
     imprime(5) ;

system("pause");
return 0;
}
```

**Basta colocar no código onde a função será chamada o nome da função.**

## Retorno da Função (com retorno de valor)

- Se a função não for do tipo void, ela deverá retornar um valor.
- O comando return é utilizado para retornar esse valor para o programa.

  **return** expressão;

- O valor de retorno tem de ser compatível com o tipo de retorno declarado para a função.
- Consiste em: qualquer constante, variável ou expressão aritmética.

## Retorno da Função (com retorno de valor)

- Essa expressão pode até mesmo ser uma outra função, como a função sqrt().

```
return sqrt( x );
```

- O retorno pode ser feito até no uso da função printf().
- Isso é possível por que a função irá retornar apenas um valor, e o comando printf recebe esse valor a ser impresso.

## Retorno da função

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

float  potencia( int  b, int  e ){

   return  pow(b, e) ;
}

int main( ){
     int base, exp;
     printf("Digite a base: ");
     scanf("%d", &base);
     printf("Digite o  expoente: ");
     scanf("%d", &exp);
     printf( "O resultado da potencia %c: %.2f", 130,potencia(base, exp));
     system("pause");
     return 0;
}
```

***Será impresso o valor retornado da função.***

## Retorno da Função (com retorno de valor)

- Uma função pode ter mais de uma declaração return.
- O uso de vários comandos return é útil quando o retorno da função está relacionado a determinada condição dentro dela.
- No entanto, esse comando também é usado para terminar a execução de uma função.
- Na maioria dos casos, o ideal é reescrever a função para que ela use somente um comando return.

# Estrutura de Dados e Algoritmos I

## Modularização – Vários Retornos

### Retorno da função

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

int  maior( int  x, int  y ){
     if (x > y)
        return  x ;
     else
        return  y ;
}

int main( ){
     int  A, B;
     printf("Digite o valor de A: ");
      scanf("%d", &A);
      printf("Digite o  valor de B: ");
      scanf("%d", &B);
printf( "O maior %c: %d", 130,maior(A, B));
      system("pause");
      return 0;
}
```

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

int  maior( int  x, int  y ){
     int  z ;
     if (x > y)
          z =  x ;
     else
          z =  y ;
     return  z ;
}

int main( ){
     int  A, B;
     printf("Digite o valor de A: ");
     scanf("%d", &A);
     printf("Digite o  valor de B: ");
     scanf("%d", &B);
    printf( "O maior %c: %d", 130,maior(A, B));
     system("pause");
     return 0;
}
```

reescreva a função para que ela use somente um comando return

# Estrutura de Dados e Algoritmos I

## Modularização – Vários Retornos

### Retorno da função

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

int  maior( int  x, int  y ){
     if (x > y)
        return  x ;
     else
        return  y ;

     printf("Fim da função \n");
}

int main( ){
     int  A, B;
     printf("Digite o valor de A: ");
      scanf("%d", &A);
      printf("Digite o  valor de B: ");
      scanf("%d", &B);
      printf( "O maior %c: %d", 130,maior(A, B));
      system("pause");
      return 0;   }
```

***Trecho de código ignorado, pois o comando return encerra a execução da função.***

# Estrutura de Dados e Algoritmos I

## Modularização - Exercícios

1. Faça um programa contendo uma sub-rotina que receba dois números positivos por parâmetros e retorne a soma dos N números inteiros existentes entre eles:

2. Faça uma programa contendo uma sub-rotina que receba três números inteiros *a*, *b* e *c*, sendo *a* maior que 1. A sub-rotina deverá somar todos os inteiros entre *b* e *c* que sejam divisíveis por *a* (inclusive *b* e *c*) e retornar o resultado para ser impresso:

# Estrutura de Dados e Algoritmos I

## Modularização - Exercícios

3. Faça uma programa que leia dois números inteiros *a* e *b*, sendo que o usuário deve escolher qual operação matemática deseja calcular através de uma sub-rotina. Dependendo da escolha do usuário, para cada opção deve chamar uma outra sub-rotina para realizar o cálculo da operação, deve ser feita uma sub-rotina para cada operação:

   1 – Adição

   2 – Subtração

   3 – Multiplicação

   4 – Divisão

   **Lembre-se que não tem divisão por zero.*

# ESTRUTURA DE DADOS E ALGORITMOS I

## Modularização – parte I

Profº. Sérgio Roberto Costa Vieira, M.Sc.
Cursos de Computação
2º. Período