Aula 13 – Oficina de Programação

Modularização

Profa. Elaine Faria

UFU - 2017

O que é modularização?

- No século XIX, **Henry Ford**, para baratear e massificar a montagem de carros, criou uma base modular.
- Esta base era usada para montar diversos tipos de automóveis.
- Ele batizou de modelo T (Ford Modelo T).

O que é modularização?

O que é modularização?

- Atualmente, encontram-se diversos produtos modulares.
- Isto significa que peças menores podem se encaixar de diversas formas, com o objetivo de formar algo maior.

O que é modularização?

- Até agora, nesta disciplina, foi utilizado apenas o programa principal (main) para executar todas as tarefas.
- No entanto, esta não é a melhor solução para o desenvolvimento de código estruturado.
- A modularização do código visa separar as tarefas em **funções** ou **procedimentos**, cada um deles realizando uma parte do programa.

O que é modularização?

- Para se realizar uma tarefa, muitas vezes, a mesma precisa ser dividida em uma sequência de passos.

- Exemplo: Trocar a lâmpada
  - …
  - Posicionar escada embaixo da lâmpada
  - Subir na escada
  - Desconectar a lâmpada velha
  - Conectar a lâmpada nova
  - …E assim por diante...

O que é modularização?

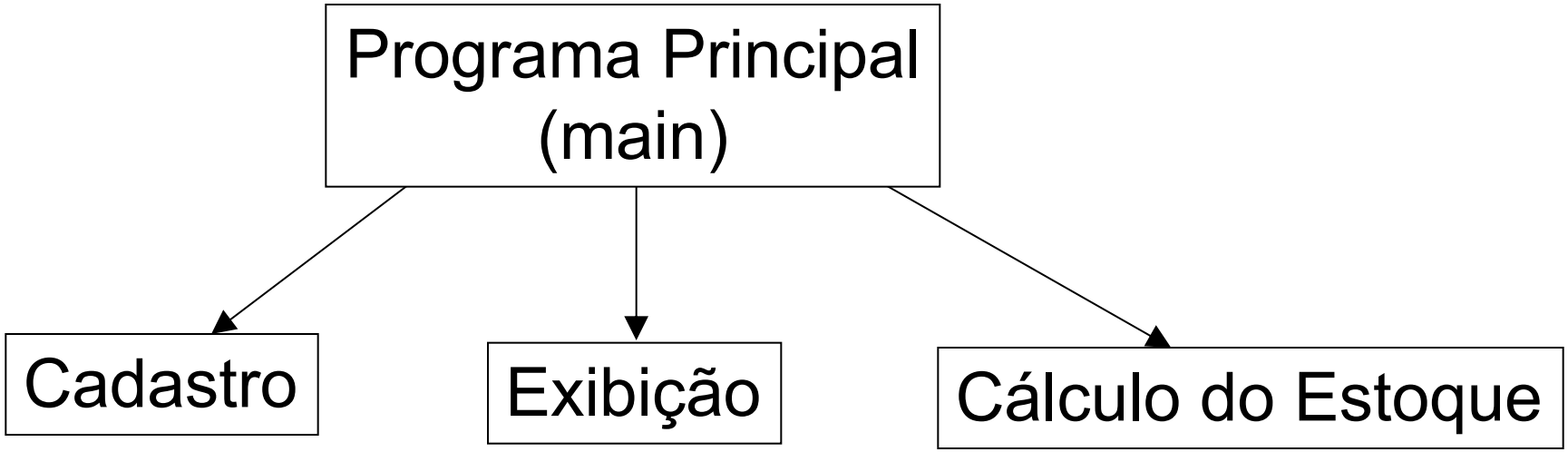
- Conforme a tarefa vai crescendo, e se tornando mais complexa, surgem situações a serem resolvidas, de forma que o **problema** possa ser **solucionado**
- Dentro de um grande problema, temos diversos problemas menores.
- Esses problemas menores afetam a legibilidade (Clareza), fazendo com que o entendimento e/ou manutenção dessa lógica seja difícil.
- Usando a modularização é possível minimizar, ou até mesmo evitar, essa ”confusão”.

O que é modularização?

- Modularizar é quebrar um problema em pequenas partes, sendo cada uma destas, responsável pela realização de uma etapa do problema.

Modularização

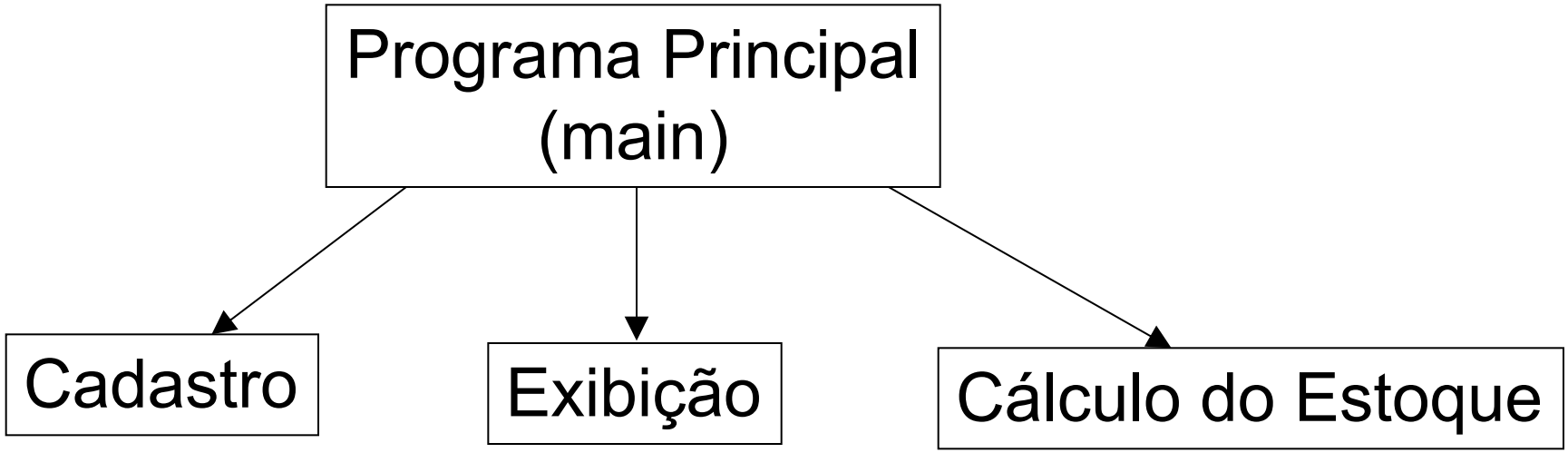
- Suponha que você tenha que fazer um programa que permita cadastrar produtos, exibir produtos e calcular o total do estoque.



- Diferentes módulos podem ser chamados pelo programa principal.

Modularização

- Agora,as funções podem ser chamadas tanto do programa principal quanto de outras funções, a qualquer momento e quantas vezes for necessário



Funções em C

- Na linguagem C os módulos são implementados por **funções**
- O *main* (programa principal) é uma função especial. Ela é sempre a primeira a ser executada.
- Já usamos funções em C mas sem perceber.

Funções em C

- O que faz o código abaixo?

```
int idade;
printf("Digite a sua idade: ");
scanf("%d",&idade);
```

- Quantas funções ele utiliza?

Funções Matemáticas

Função	descrição	Exemplo
ceil(x)	"teto" – arredonda para o menor inteiro não menor que x	ceil(10.3) é 11.0 ceil(-7.8) é -7.0
floor(x)	"piso" – arredonda para o maior inteiro não maior que x	floor(5.3) é 5.0 floor (-4.2) é -5.0
cos(x)	Cosseno de x em radianos	Cos(0.0) é 1.0
sin(x)	Seno de x em radianos	Sin(0.0) é 0
exp(x)	Exponencial e^x	exp(1) é 2.71828
fabs(x)	Valor absoluto de x	fabs(5.3) é 5.3 fabs(-4.2) é 4.2
Log(x)	Logaritmo natural de x	log(2.71828) é 1.0
Log10(x)	Logaritmo na base 10	Log10(10.0) é 1.0

Funções Matemáticas

Função	Descrição	Exemplo
pow(x,y)	X elevado a Y (x^y)	pow(2,7) é 128
sqrt(x)	Raiz quadrada de x (x deve ser um valor não negativo)	sqrt(9.0) é 3.0
fmod(x,y)	Retorna o resto da divisão de x por y em ponto flutuante	fmod(2.6, 1.2) é 0.2
tan(x)	Tangente de x (x em radianos)	tan(0) é)

Para se usar as funções matemáticas deve-se usar a biblioteca <math.h>

Funções em C

- A forma geral de uma função em C é:

```
tipo_da_função nome_da_funcao (lista_de_parametros)
{
        corpo_da_funcao
}
```

Funções em C – Tipo da Função

```
tipo_da_função nome_da_funcao (lista_de_parametros)
{
      corpo_da_funcao
}
```

- **Tipo da função:** refere-se ao tipo de resposta que a função devolve (int, float, char, void, etc).
- Se nenhum tipo for especificado, a linguagem C assume que o retorno será do tipo int (inteiro).
- Quando não é necessário um retorno, usa-se **void**

Funções em C – Nome da Função

```
tipo_da_função nome_da_funcao (lista_de_parametros)
{
      corpo_da_funcao
}
```

- **Nome da função:** segue as regras de nomenclatura do C. Não podendo possuir espaços, caracteres especiais ou acentos. Também não podem ser palavras reservadas da linguagem.

- É **recomendável** que o nome da função seja o mais explicativo possível sobre o seu funcionamento. Ex:
  - float divideDoisNumerosInteiros() {…}
  - void AlertaUsuario() { ... }

Funções em C – Lista de Parâmetros

```
tipo_da_função nome_da_funcao (lista_de_parametros)
{
        corpo_da_funcao
}
```

- **Lista de parâmetros:** Relação de nomes de variáveis e seus tipos, para entrada de dados na função.
- Esse é um mecanismo usado para transmitir informações para uma função

Funções em C – Exemplo 1

- Vamos criar uma função que mostre na tela uma saudação ao usuário e, em seguida, vamos chamar esta função, a partir da função principal.

```
# include <stdio.h>
void saudacao()
{
    printf("Seja bem vindo!");
}

int main( )
{
    saudacao();
    return 0;
}
```

Funções em C – Exemplo 1

- Onde está a definição e chamada da função saudacao?

```
# include <stdio.h>
void saudacao()
{
    printf("Seja bem vindo!");
}

int main( )
{
    saudacao();   // Chamada da função saudacao
    return 0;
}
```

// Definição da função saudacao

Funções em C – Exemplo 2

```
# include <stdio.h>
int soma(int f1, int f2)
{
    int s;
    s = f1 + f2;
    return(s);
}

int main( )
{
    int resultado;
    resultado = soma(2, 4);
    printf("\nSoma: %d", resultado);
    return 0;
}
```

Funções em C – Exemplo 2

```
# include <stdio.h>
int soma(int f1, int f2)
{
    int s;
    s = f1 + f2;
    return(s);
}

int main( )
{
    int resultado;
    resultado = soma(2, 4);
    printf("\nSoma: %d", resultado);
    return 0;
}
```

// Definição da função soma

// Chamada função soma

Definição x Chamada de Funções

- Qual a diferença entre a implementação da **definição** de uma função e da **chamada** de uma função?
- Na definição de uma função, o ponto-e-vírgula não pode ser usado.

```
void saudacao( )
{
    // corpo da função
}
```

- A chamada a uma função deve se finalizada por ponto-e-vírgula.

```
saudacao( );
```

Escopo de variáveis

- Cada função só enxerga e pode alterar as variáveis que são criadas dentro delas.
- Uma variável criada dentro do main não poderá ser usada por uma outra função
- Variáveis criadas dentro de cada função são chamadas de **variáveis locais.**
- As variáveis locais existem apenas durante a execução do bloco de código onde estão declaradas.

Entenda o programa abaixo e identifique as variáveis locais.

```
int quadrado(){
   int num, quad;
   scanf("%d", &num);
   quad = num * num;
   return (quad);
}

int main(){
     int result;
     printf("Digite o numero: ");
     result = quadrado();
     printf("Quadrado do numero: %d", result);
     return 0;
}
```

Entenda o programa abaixo e identifique as variáveis locais.

```
int quadrado(){
   int num, quad;
   scanf("%d", &num);
   quad = num * num;
   return (quad);
}

int main(){
     int result;
     printf("Digite o numero: ");
     result = quadrado();
     printf("Quadrado do numero: %d", result);
     return 0;
}
```

Variáveis locais.

Variável local

Por que o programa abaixo não funciona?

```
#include <stdio.h>
int quadrado(){
   int num, quad;
   scanf("%d", &num);
   quad = num * num;
   return (quad);
}
int main(){
     printf("Digite o numero: ");
     quad = quadrado();
     printf("Quadrado do numero: %d", quad);
     return 0;
}
```

Como resolver o problema?

Por que o programa abaixo não funciona?

```
#include <stdio.h>
int quadrado(){
   int num, quad;
   scanf("%d", &num);
   quad = num * num;
   return (quad);
}
int main(){
     printf("Digite o numero: ");
     quad = quadrado();
     printf("Quadrado do numero: %d", quad);
     return 0;
}
```

Como resolver o problema?

Escopo de variáveis

- Soluções:

  1) Usar variáveis globais

  2) Declarar a variável também na função main (solução do slide 26)

  3) Passar a variável quad como parâmetro da função quadrado. (será visto na disciplina de ED)

Escopo de variáveis

- É possível declarar variáveis acessíveis a todas as funções. Para isto tem que ser criadas fora das funções, após os includes.

- Estas variáveis, acessíveis a partir de todas as funções, são chamadas **variáveis globais.**

Solução

```
#include <stdio.h>
int quad;
void quadrado(){
   int num;
   scanf("%d", &num);
   quad = num * num;
}
int main(){
     printf("Digite o numero: ");
     quadrado();
     printf("Quadrado do numero: %d", quad);
     return 0
}
```

Variável global

Como quad é uma variável global a função quadrado pode ser do tipo void.

Variáveis Globais

- Variáveis globais são declaradas fora de todas as funções do programa

- Elas são conhecidas e podem ser alteradas por todas as funções do programa
  - Quando uma função tem uma variável local com o mesmo nome de uma variável global a função dará preferência à variável local.

- **Evite variáveis globais!**

Exemplo do uso de funções com variáveis globais

```c
#include <stdio.h>

float valor1, valor2;

float calculaMedia() {
   return (valor1+valor2)/2;
}

int main(void) {
   float media;
   printf("Digite o valor 1 ");
   scanf("%f",&valor1);
   printf("Digite o valor 2");
   scanf("%f",&valor2);

   media = calculaMedia();

   printf("Media = %.1f",media);

   return 0;
}
```

Tipo de retorno da função main

```
#include <stdio.h>

void imprimirMensagem()
{
    printf("Aprendendo funcao em C");
}


int main()
{
    imprimirMensagem();
    return 0;
}
```

Comando return

- O comando return tem dois usos importantes:

  1) Devolver um valor e retornar para a função que o chamou

  2) Pode ser usado sem os parênteses para causar uma saída imediata da função em que se encontra.

- Limitações do return:
  - O comando return pode retornar somente **um único valor**

Por que a função imprimirMsg() não tem o comando return?

```
#include <stdio.h>
void imprimirMsg()
{
        printf("Seja Bem vindo!");
}

int main()
{
        imprimirMsg();
        return 0;
}
```

```c
#include <stdio.h>

void saudacoes(int hora)
{
   if(hora > 6 && hora < 12 )
   {
       printf("Bom dia!!!");
       return;
    }
   if( hora >= 12 && hora < 18)
   {
       printf("Bom tarde!!!");
       return;
    }

   printf("Sem saudacoes!");
}
```

O return causa a saída imediata da função em que se encontra

```c
main()
{
    printf("Digite a hora: ");
    scanf("%d", &hora);
    saudacoes(hora);
}
```

Funções em C

- Passagem de parâmetros (argumentos)

```
#include <stdio.h>

float soma(float x, float y, float z)
{
    float soma;
    soma = x + y + z;

    return (soma);
}

int main()
{
    float result = soma(2.5, 7.5, 3.0);
    printf("Soma: %f", result);
    return 0;
}
```

Funções em C

- Passagem de parâmetros (argumentos)

```
#include <stdio.h>
void verificaConta(char tipo, float saldo)
{
   if(tipo == 'P' && saldo > 10000)
      printf("Cliente Especial");
   else
      printf("Cliente Normal");
}
int main()
{
  char t = 'P';
  float s = 15000;
  verificaConta(t, s);
  return 0;
}
```

Passagem de Parâmetros

- Na linguagem C, os parâmetros de uma função são sempre passados por valor, ou seja, uma cópia do valor do parâmetro é feita e passada para a função.

- Mesmo que esse valor mude dentro da função, nada acontece com o valor fora da função

Passagem de Parâmetro por Valor

- Na passagem de parâmetro por valor a função chamada tem uma cópia dos valores que são passados como argumentos.
- Dentro da função chamada são criadas outras variáveis temporárias para armazenar estes valores.

Exemplo 1: Parâmetro por Valor

```
#include <stdio.h>
int quadrado(int x)
{
    int q = x * x;
    return (q);
}
int main()
{
   int num, result;
   printf("Digite um numero: ");
   scanf("%d", &num);
   result = quadrado(num);
   printf("\nResultado: %d", result);
   return 0;
}
```

Exemplo 2: Parâmetro por Valor

```
#include <stdio.h>
float media(float a, float b)
{
    return ((a+b)/2);
}

int main()
{
   float x, y, med;
   printf("Digite dois valores:");
   scanf("%f %f", &x, &y);
   med = media(x,y);
   printf("A media eh %f:”,med);
   return 0;
}
```

Exercício 1

- Construa um programa C que possua uma função MODULO, ou seja, que converta qualquer número digitado em positivo. Por exemplo, se for digitado o número -3, a função deve retornar 3, mas se for digitado 2 a função deve retornar 2. Utilizar passagem de parâmetro por valor.

Exercício 1 - Resposta

```
int main()
{
      int valor, result;
      printf("\nDigite um numero inteiro: ");
      scanf("%d", &valor);

     result = modulo(valor);

      printf("\nResultado: %d", result);

      return 0;
}
```

```
int modulo(int num)
{
      int mod;
      if(num < 0)
        mod =num * (-1);
      else
          mod = num;
      return mod;
}
```

Exercício 2

- Faça um programa que crie variáveis para modelar um “ponto” em coordenadas (x,y).

→ O programa deverá pedir para o usuário entrar com as coordenadas dos dois pontos

→ Construir uma função que calcule a distância euclidiana entre os dois pontos e mostre esta distância na tela onde (x_1,y_1) são as coordenadas do ponto 1 e (x_2,y_2) são as coordenadas do ponto 2.

$$Dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- * raiz = **sqrt(**numero**)** é a função para a raiz quadrada

Passagem de Parâmetro por Referência

- Quando se quer que o valor da variável mude dentro da função, usa-se passagem de parâmetros por referência.
- Neste tipo de chamada, não se passa para a função o valor da variável, mas a sua referência (seu endereço na memória).
- Utilizando o endereço da variável, qualquer alteração que a variável sofra dentro da função será refletida fora da função.
  - Ex: função scanf()

Arrays como parâmetros

- Para utilizar arrays como parâmetros de funções alguns cuidados simples são necessários
- Arrays são sempre passados por referência para uma função
  - A passagem de arrays por referência evita a cópia desnecessária de grandes quantidades de dados para outras áreas de memória durante a chamada da função, o que afetaria o desempenho do programa

Arrays como parâmetros

- É necessário declarar um segundo parâmetro (em geral uma variável inteira) para passar para a função o tamanho do array separadamente.
  - Quando passamos um array por parâmetro, independente do seu tipo, o que é de fato passado é o endereço do primeiro elemento do array

Ex: `void imprime (int m[5], int n);`

Structs como parâmetros

- Podemos passar uma struct por parâmetro ou por referência
- Temos duas possibilidades
  - Passar por parâmetro toda a struct
  - Passar por parâmetro apenas um campo específico da struct
- Em ambos os casos é possível fazer a passagem por valor ou referência

Funções Recursivas

- Na linguagem C, uma função pode chamar outra função.
  - A função main() pode chamar qualquer função, seja ela da biblioteca da linguagem (como a função printf()) ou definida pelo programador (função imprime()).
- Uma função também pode chamar a si própria
  - A qual chamamos de função recursiva.

Funções Recursivas

- A recursão também é chamada de definição circular. Ela ocorre quando algo é definido em termos de si mesmo.
- Um exemplo clássico de função que usa recursão é o cálculo do fatorial de um número:
  - 3! = 3 * 2!
  - 4! = 4 * 3!
  - n! = n * (n - 1)!

Funções Recursivas

```
int fatorial (int n){
     if (n==0)
          return 1;
     else
          return n * fatorial (n-1);
}
```

Com recursão

```
int fatorial (int n){
     int i, f=1;
     if (n==0)
          return 1;
     else
          for (i = 1;i <=n;i++)
               f = f*i;
          return f;
}
```

Sem recursão

Funções Recursivas

- Em geral, formulações recursivas de algoritmos são frequentemente consideradas "mais enxutas" ou "mais elegantes" do que formulações iterativas.
- Porém, algoritmos recursivos tendem a necessitar de mais espaço do que algoritmos iterativos.

Referências

- Material do prof. André Backes
  www.facom.ufu.br/~backes