UNIVERSIDADE DO ESTADO DE MINAS GERAIS
UEMG

MODULARIZAÇÃO

Kamila Rios
21/03/2015

Modularização

Muitas vezes um problema grande e/ou complexo pode ser resolvido mais facilmente se for dividido em pequenas partes (subproblemas).

Modularização

Muitas vezes um problema grande e/ ou complexo pode ser resolvido mais facilmente se for dividido em pequenas partes.

Isso é o que chamamos de **Programação Modular ou**

Modularização

Modularização

Muitas vezes um problema grande e/ou complexo pode ser resolvido mais facilmente se for dividido em pequenas partes.

Isso é o que chamamos de **Programação Modular ou Modularização**

Modularização significa desenvolver algoritmos em **módulos!**

Modularização

Muitas vezes um problema grande e/ ou complexo pode ser resolvido mais facilmente se for dividido em pequenas partes.

Isso é o que chamamos de **Programação Modular ou** M**odularização**

Modularização significa desenvolver algoritmos em **módulos!**

A implementação desses Módulos é feita através de **Subprogramas**

Modularização

Muitas vezes um problema grande e/ou complexo pode ser resolvido mais facilmente se for dividido em pequenas partes.

Isso é o que chamamos de **Programação Modular ou Modularização**

Modularização significa desenvolver algoritmos em **módulos!**

A implementação desses Módulos é feita através de **Subprogramas**

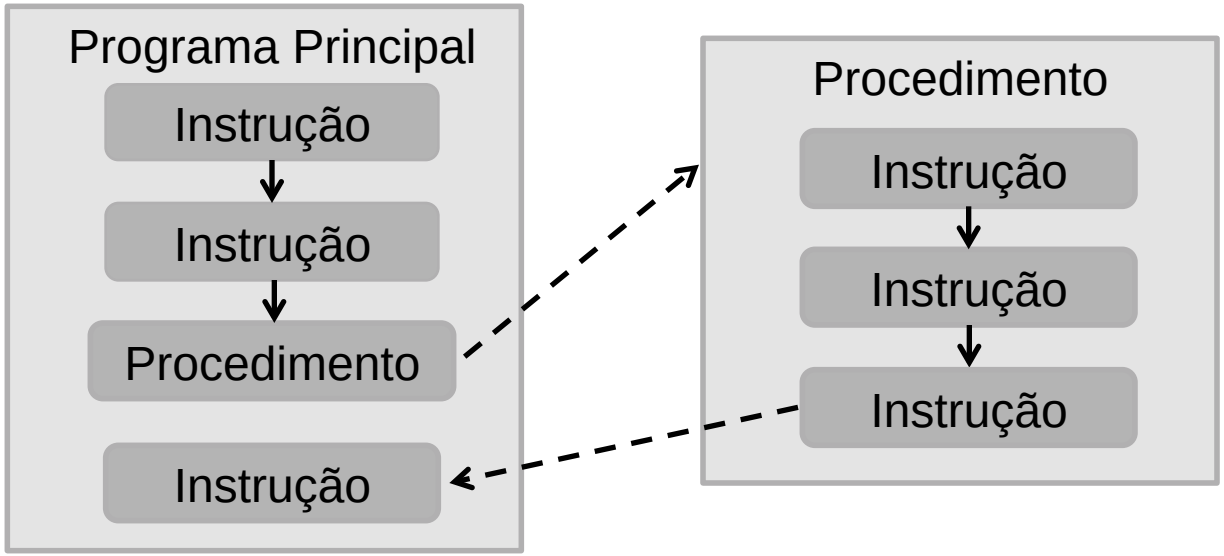
Os subprogramas são **funções**

Modularização - Vantagens

- Divide um problema em subproblemas (subprogramas);
- Módulos são mais simples de serem construídos;
- Evita repetição de um mesmo trecho de código;
- Facilita o entendimento, teste e a detecção de erros;
- Módulos menores facilita a manutenção;
- Divide o desenvolvimento entre vários programadores;
- Permite “reutilização” de trechos de programas.

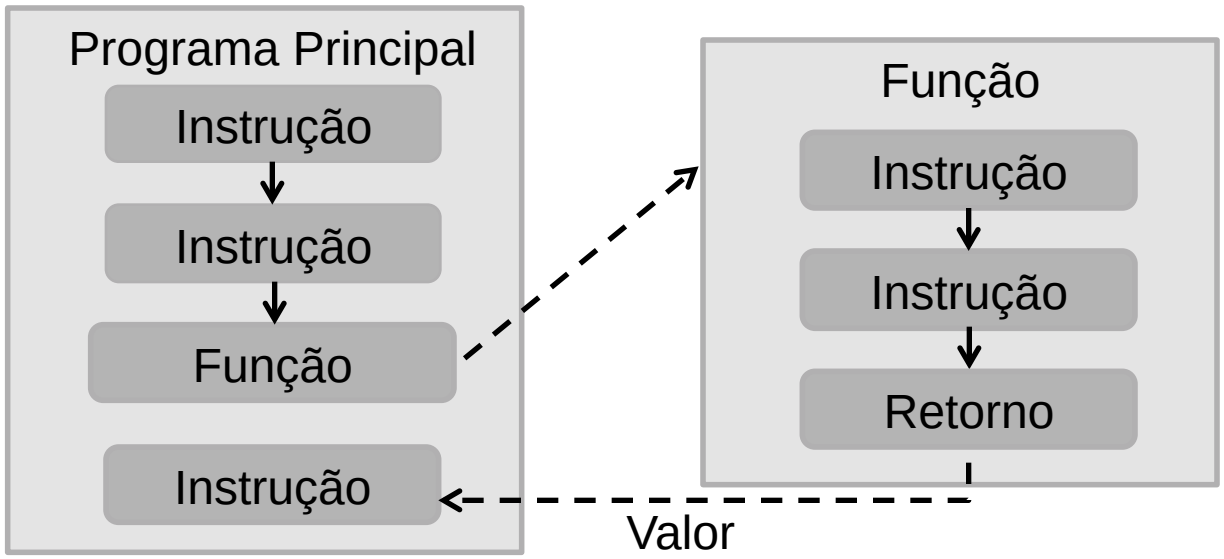
Tipos de Módulos

- Procedimentos:
  - São módulos que quando chamamos causam desvio no fluxo de execução. As tarefas do procedimento são realizadas e, ao término da execução, o fluxo retorna ao programa principal, e a instrução subsequente à chamada do procedimento é executada;
  - Não retorna valores ao programa principal ou ao seu “chamador”.
  - void main () {...}



Tipos de Módulos

- Funções:
  - São módulos que quando chamamos causam desvio no fluxo de execução. As tarefas da função são executadas e, ao término da execução, a função retorna um valor ao programa principal;
  - int / float/ double/ char main () {...return? ;}



Modularização - Funções

- As **FUNÇÕES:**
  - ✓Trecho de código de um programa projetado para cumprir uma tarefa específica;
  - ✓Possuem “código independente”;
  - ✓Pode ser especificada no mesmo arquivo do programa principal ou em um arquivo à parte que constitua uma biblioteca a ser incluída no programa principal;

- O uso da função envolve três passos:
  - **Declaração** (protótipo) – antes do main( )
  - **Definição** (código da função) – depois do main( )
  - **Ativação** (chamada): dentro do main( ) ou de uma outra função

Funções – Definição (Protótipo)

- Sintaxe da Declaração ou Prototipação da Função:

  tipo de retorno da função* nome_da_função *(lista_de_parâmetros);

(**tipo** parametro1, **tipo** parametro2,..., **tipo** parametroN

  - ***tipo*** é o tipo da informação retornada da função; se a função não retornar nada, seu tipo deve ser *void*;
  - ***parâmetros***: lista de tipos (e variáveis) que serão passados como argumentos para a função; pode ser vazio.

  tipo de retorno da função* nome_da_função *(void);

- Exemplo em linguagem C:

  int soma (float a, float b);
  int main (void) ou void main ();

Para que serve o protótipo?

- Através do protótipo, usado no início do programa, é possível que o compilador verifique se existem erros nos tipos de dados entre os argumentos usados para chamar uma função e a definição de seus parâmetros;

- Verifica também se a quantidade de argumentos é igual a quantidade de parâmetros, caso contrário causará erros na compilação/execução do programa.

Funções – Definição (Código da Função)

- Sintaxe da Definição:

tipo de retorno da função nome_da_função (lista_de_parâmetros)

{

declaração de variáveis;

comandos; ***//corpo da função***

return (variável de retorno);

}

  - a primeira linha é igual à declaração;
  - o ***return*** serve para indicar o valor a ser retornado, se for o caso, e pode aparecer em qualquer ponto da função, não só no final.

Funções – Definição (Código da Função)

- Exemplo em linguagem C:

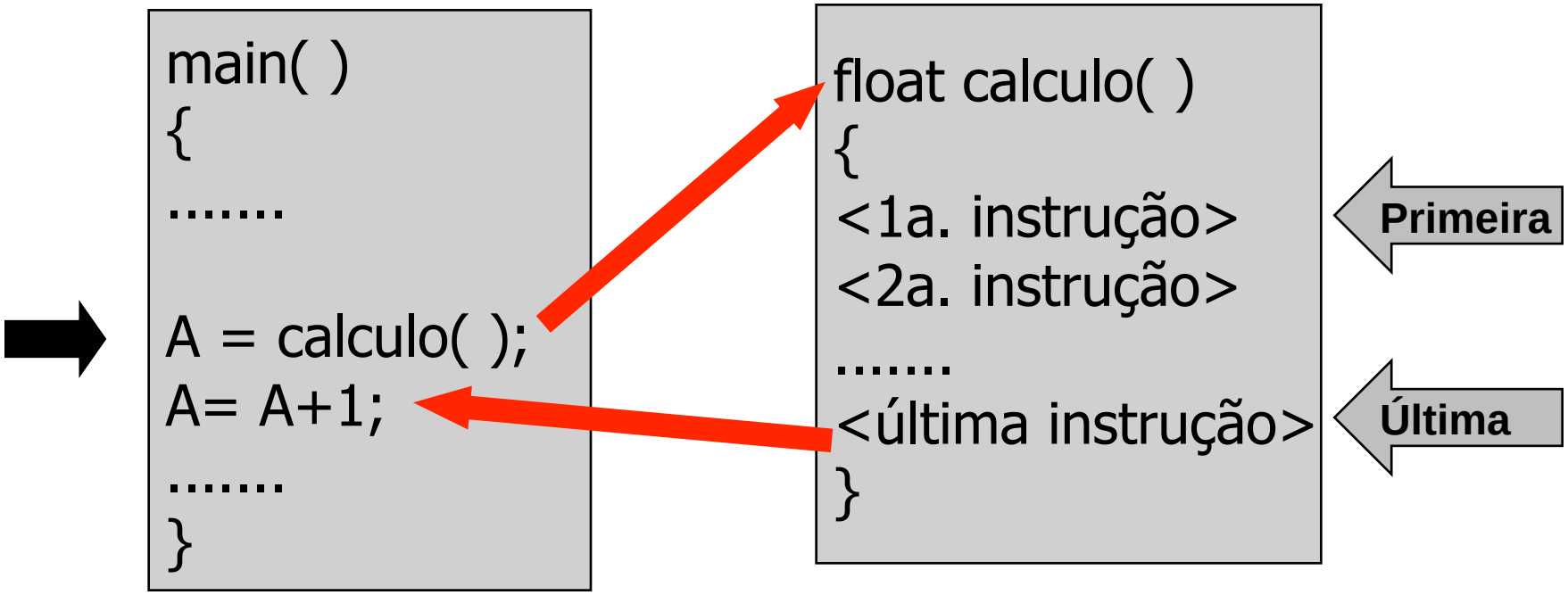
```
float soma (float a, float b){
float c;
  c= a +b;
return c;
}
```

Funções – Definição (Código da Função)

- **Comando *Return*:**
  - ✓Atribui o valor de uma expressão qualquer à função, retornando este resultado para o trecho do programa que chamou a função;
  - ✓Causa uma saída imediata da função na qual ele se encontra, fazendo com que a execução retorne para o ponto do programa que chamou a função;
  - ✓Pode aparecer mais de uma vez na função (sendo que apenas um será executado a cada ativação/chamada do módulo).

Funções – Ativação (Chamada)

- ✓ A ativação faz com que o controle seja transferido para o trecho chamado (primeira instrução) e executa até o fim do trecho (última instrução);
- ✓ Ao final da função, o controle volta para instrução seguinte à chamada.



Exemplos de especificação dos módulos

```
# include <stdio.h>
void funcao ()
{
printf (“Olá mundo dos módulos!\n”);
}

int  main ()
{
printf (“Programa Principal!\n”);
funcao ();
return 0;
}
```

```
# include <stdio.h>
void funcao ()
int  main ()
{
printf (“Programa Principal!\n”);
funcao ();
return 0;
}
void funcao ()
{
printf (“Olá mundo dos módulos!\n”);
}
```

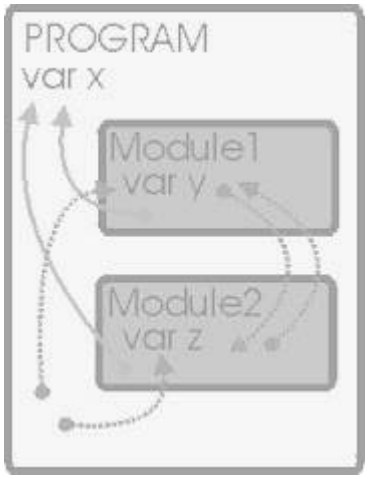
Exemplo de Função

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

int soma1010 ()
{
    int resultado;
    resultado = 10+10;
    return resultado;
}
int main()
{
    int eq;
    eq = 2*soma1010()+ 3*soma1010()+5;
    printf("%d", eq);
    return 0;
}
```

UEMG UNIVERSIDADE DO ESTADO DE MINAS GERAIS

ESCOPO DE VARIÁVEIS



Kamila Rios
21/03/2015

Escopo de variáveis

- Significa a **visibilidade** de uma variável perante os diversos subprogramas integrantes do programa (ou algoritmo);

- O escopo de uma variável é definido pelas regiões (blocos) onde a variável pode ser utilizada;

- Variáveis declaradas no mesmo escopo (bloco) precisam ter nomes diferentes, mas nomes podem ser "reaproveitados" em outros escopos.

Escopo de variáveis

- **Variável global:** declarada no início do algoritmo ou programa (fora das funções e da função principal) e é visível em todos os blocos de código.

- **Variável local:** declarada dentro de um escopo (função principal ou funções específicas) e somente visível dentro do mesmo.

Variáveis Globais

- Declaradas fora das funções;
- Podem ser usadas em qualquer função;
- Devem ficar no início do arquivo, antes da declaração da função main( );
- São globais àquela unidade de programa.

Variáveis Locais

- Espaço de memória é alocado na entrada da execução da função e liberado na saída;
- Podem ser declaradas dentro de qualquer bloco de código (variáveis locais ao bloco);
- Só podem ser usadas pela função à qual pertencem;
- São variáveis automáticas: o valor é perdido quando a função termina → não guardam o valor anterior.

Exercícios

1- Faça um algoritmo usando função para calcular Fatorial;

2-Faça um algoritmo usando função para calcular Potência;

3- Faça um algoritmo usando procedimento para calcular Fatorial;

4-Faça um algoritmo usando procedimento para calcular Potência.