Aula 04
(Parte A)

Modularização de Programas

Profa. Carla G. Pelissoni

Grupo ibmec
METROCAMP IBTA

Introdução

- Conforme aumenta a complexidade dos algoritmos, mais extenso se torna um programa (maior o número de linhas de código).
- Ideia básica: dividir um algoritmo complexo em pequenas partes (**subalgoritmos / módulos / funções**).

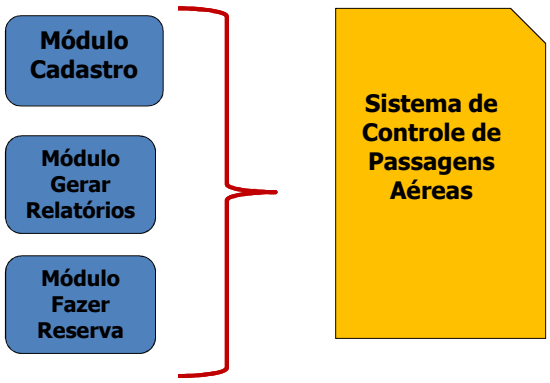
Vantagens

- Sendo um algoritmo modularizado, qualquer alteração numa função não afeta o restante do programa; **somente a função alterada**.
- Isto facilita a depuração, pois é mais simples corrigir o erro de uma função do que de um programa inteiro.
- Evita-se que uma mesma seqüência de comandos tenha que ser rescrita em vários pontos do mesmo programa **(qualquer seqüência de instruções que apareça no programa mais de uma vez é candidata a ser uma função)**.

Vantagens

- Essas vantagens tornam eficiente o trabalho em equipe, já que um problema pode ser dividido entre vários programadores, os quais podem desenvolver e testar suas partes **(seus módulos)** em separado.

Vantagens



Definição

- Uma **módulo / função** é um conjunto de instruções escritas para **cumprir uma tarefa particular** e agrupadas numa unidade com um nome para referenciá-la.
- O código de uma função é **escrito** no programa uma única vez e pode ser **executado** (chamado) muitas vezes no decorrer do programa.

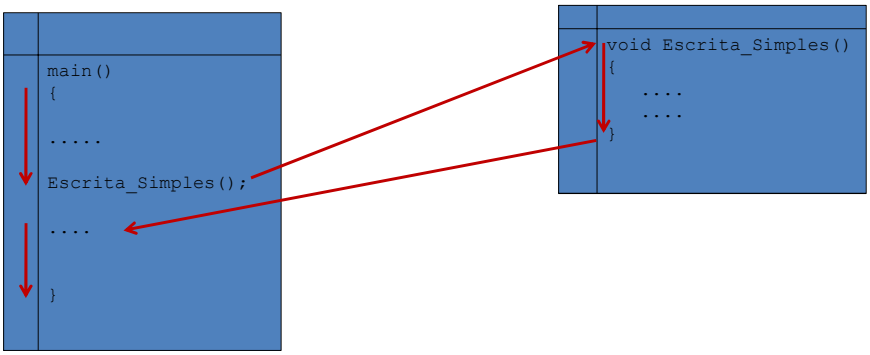
Executando (chamando) uma função

- Um programa em C pode conter uma ou mais funções, das quais uma delas deve ser **main()**.
- A execução do programa sempre começa em **main()** e, quando o controle do programa encontra uma instrução que inclui o nome de uma função, a função é chamada.
- **Chamar uma função** é o meio pelo qual solicitamos que o programa desvie o controle e passe para a função, execute suas instruções e depois volte à instrução seguinte à chamada da função.

Exemplo 1: Define, cria protótipo e chama uma função simples

```
#include <iostream.h>
void Escrita_Simples();// Protótipo da função
main()
{
     cout << "main() chamará a função simples(): \n";
     Escrita_Simples(); // Executando (chamando) a função
     cout << "\nRetorno a funcao main().";
}
// Definição (escrita) da função
void Escrita_Simples()
{
     cout << "Eu sou apenas uma função que escreve uma
     mensagem.\n";
}
```

Exemplo 1



Exemplo 2: Define uma função passando um valor como parâmetro

```
#include <iostream.h>
int Idade(int n);// Protótipo da função
void main()
{
   int id, id_meses;

     //Chama a função Idade() passando um valor constante (18).
     id_meses=Idade (18);
     cout << "\n18 anos em meses eh: " << id_meses;

     cout << "Entre com sua idade: ";
     cin >> id;
     //Chama a função Idade() novamente, agora passando uma variável lida.
     id_meses=Idade(id);
     cout << "Sua idade em meses é: " << id_meses << " meses.\n";
}

// Função Idade(): Calcula a idade em meses
int Idade (int n) {
   int local;
   local = n * 12;
   return local;  // Retorna o valor calculado para a funcao main().
}
```

Exemplo 3

```
#include <iostream.h>
// Define uma função com um valor de retorno
// Converte kilogramas em gramas
float Kilogramas (float peso);  // Protótipo da função
main(){
   float kilo, grama;
   cout << "Entre com o peso em kilogramas: ";
   cin >> kilo;
   grama = Kilogramas (kilo);
   cout << kilo << " kilogramas são ";
   cout << grama << " gramas. \n";
}
float Kilogramas (float peso){
   float total;
   total = 1000 * peso;
   return total;
}
```

Variáveis Locais e Parâmetros

- No **Exemplo 3** as variáveis ***kilo*** e ***grama*** são **variáveis locais** à função **main()**.
- Já ***total*** é uma **variável local** à função **Kilogramas()**.
- O que isso significa?
- As variáveis locais só existem dentro de suas respectivas funções. Ou seja, se você tentar usar a variável ***kilo*** dentro da função **Kilogramas()** irá gerar um erro de compilação.
- Terminada a função, as variáveis locais "desaparecem" da memória do computador (seus valores deixam de existir).

Exemplo 3

```
#include <iostream.h>
// Define uma função com um valor de retorno
// Converte kilogramas em gramas
float Kilogramas (float peso);        // Protótipo da função
main(){
      float kilo, grama;
      cout << "Entre com o peso em kilogramas: ";
      cin >> kilo;
      grama = Kilogramas (kilo);
      cout << kilo << " kilogramas são ";
      cout << grama << " gramas. \n";
}
float Kilogramas (float peso){
      float total;
      total = 1000 * peso;
      return total;
}
```

Variável local main()

Parâmetro

Variável local Kilogramas()

Grupo ibmec METROCAMP IBTA

Variáveis Locais e Parâmetros

- **Parâmetros:** São variáveis através das quais são passadas informações às funções para que elas possam realizar cálculos, verificações, etc.
- No **Exemplo 3**, o parâmetro ***peso*** recebe o valor em kilos da função main() que será transformado em gramas pela função ***Kilogramas***.
- A seguir, no **Exemplo 4**, função **quadrado (float x)** recebe um único parâmetro (x).
- **Este parâmetro funciona como uma variável local à função, não existindo fora dela**.

Grupo ibmec METROCAMP IBTA

Exemplo 4

```
main()
{
   float x, resultado;
   cout << "Valor: ";
   cin >> x;
   resultado = quadrado(x);
   cout << "Quadrado de " << x << " = " <<
   resultado;
}
float quadrado(float x)
{
   float quad;
   quad = x*x;
   return quad;
}
```

Variáveis Locais e Parâmetros

- Observe que a variável local ***float x*** declarada dentro da função ***main()*** é diferente do parâmetro declarado na função `float` **`quadrado`** ***(float x)***.
- Apesar de terem o mesmo nome são duas variáveis diferentes.
- A relação entre elas acontece quando a função ***quadrado (x)*** é chamada na função **main()**. Nesse momento é passada uma cópia do valor **variável local x** (da função **main**) para o **parâmetro x** da função **`quadrado`**.

Exemplo 5

```
...
main()
{
   float x=4, resultado;
   resultado = quadrado(x);
   cout << "Quadrado de " << x << " = " << resultado;
   cout << "Valor atual de x = " << x;
}

float quadrado (float x)
{
   x = x * x;
   return x;
}
```

Variáveis Locais e Parâmetros

- A variável **x** da função main() **é diferente** do parâmetro **x** da função quadrado().
- O valor da **variável local x** (4) é copiado para o **parâmetro x** da função quadrado().
- Quando a atribuição **x = x*x** ocorre, apenas o valor do **parâmetro x** é modificado. A **variável x** declarada na função main(), ainda tem o valor 4.
- Esta função não altera os valores de seus parâmetros ⇨ **passagem de parâmetro por valor.**

Variáveis Locais e Parâmetros – Exemplo de Teste de Mesa

Função main		Função quadrado
x	**resultado**	**x**
4	-------	4
4	-------	16
4	16	-------

Saídas:

```
Quadrado de 4 = 16
Valor atual de x = 4
```


Passagem de Parâmetros por Valor

- É feita uma cópia do valor passado como parâmetro e as alterações são feitas nesta cópia, mantendo o valor original inalterado.
- O que ocorre dentro da função não tem efeito algum sobre a variável usada na chamada da função.

Exercícios

Exercício 1: Fazer o teste de mesa e responder quais são as saídas.

```
int Func_A(int a, int b);
main()
{
   int a, b, resu;
   a = 10;
   b = 25;
   resu = Func_A(a, b);
   cout << "\nResultado = " << resu;
}

int Func_A(int a, int b)
{
   int s;
   a = 2*a;
   b = 2*b;
   s = a + b;
   return s;
}
```

Exercício 2: Fazer o teste de mesa e responder quais são as saídas.

```
void Func_A(int a, int b);
main()
{
   int a, b;
   a = 20;
   b = 4;
   Func_A(a, b);
   cout << "\nResultados = " << a;
   cout << "\nResultados = " << b;
}

void Func_A(int a, int b)
{
   int s;
   a = 2*a;
   b = 2*b;
   a = a + b;
}
```