SI100

Algoritmos e Programação de Computadores I

1º Semestre – 2018

Tópico 9 – Parte 1
Modularização de Programas

Unidade 13

Prof. Guilherme Palermo Coelho

guilherme@ft.unicamp.br

Roteiro

- Introdução e Definições;
- Notação em C;
- A função ***main()***;
- Localização das funções em um programa;
- Exemplo;
- Exercícios;
- Referências.

Introdução e Definições

Introdução

- A **modularização de programas** consiste na utilização de **funções** e **procedimentos** para desempenhar parte das tarefas do programa;
  - Ou seja, consiste em ***dividir*** grandes tarefas de computação em ***tarefas menores***;
- O uso de funções e procedimentos em um programa traz uma série de **vantagens**:
  - Permitir o ***reaproveitamento*** do código já construído:
    - Quantas vezes já usamos a função ***printf()*** neste curso?
  - Evitar a repetição de um mesmo trecho de código várias vezes em um programa:
    - Basta chamar a função que desempenha este trecho de código.

Introdução

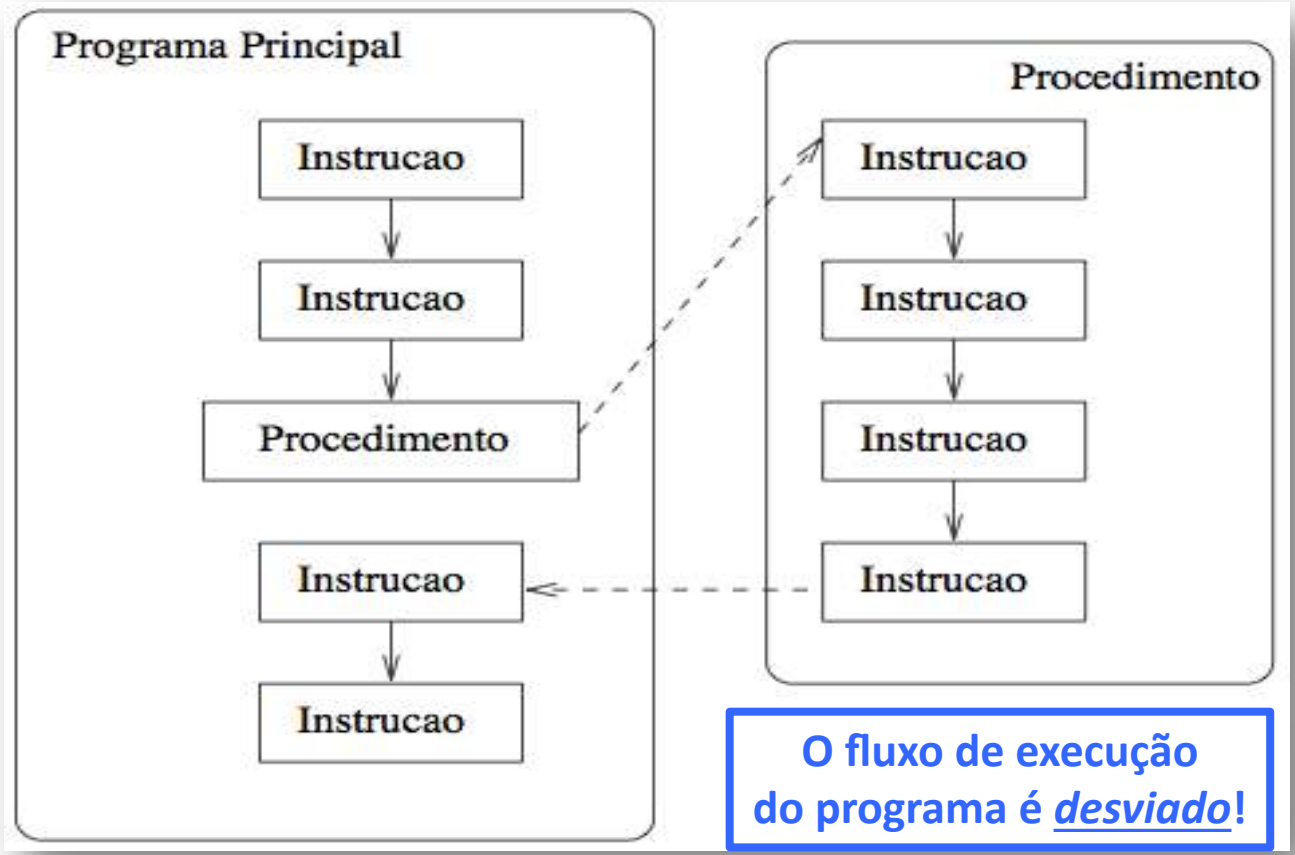
- O uso de funções e procedimentos em um programa traz uma série de vantagens:
  - Permitir a alteração de um trecho de código de forma mais rápida: basta alterar o conteúdo da função ou procedimento;
  - Evitar que blocos de programas fiquem grandes demais e, por consequência, mais difíceis de entender;
  - Facilitar a leitura do código-fonte:
    - Funções podem esconder detalhes de operação de partes do programa → tome novamente como exemplo a função ***printf()***;
  - Separar o programa em partes (blocos) que possam ser logicamente compreendidos de forma isolada;

Definições

- **Procedimentos**: são estruturas que agrupam um conjunto de comandos que são executados quando o procedimento é chamado.
  - São ***subprogramas*** que são executados quando o programa principal invoca o procedimento.
- **Funções**: são semelhantes aos procedimentos, exceto que uma função **sempre** retorna um valor:
  - **Exemplo:** função que calcula o fatorial em um programa.
    - Esta função contém as instruções necessárias para calcular o fatorial de um número ***n***;
    - O número ***n*** é recebido como ***parâmetro*** da função;
    - A função retorna o valor de ***n*!**

Definições

- Chamada de um procedimento (ou função) em um programa:



Notação em C

Notação em C

- A linguagem C só permite a definição de ***funções***.
- A forma geral de uma função em C é:

```
<tipo_da_funcao> NomeDaFuncao(<lista_de_parametros>)
{
        instrucoes;
}
```

- Caso o tipo da função não seja fornecido, C supõe que a função é do tipo inteiro (**int**);
- É possível simular o comportamento de um ***procedimento*** em C definindo o tipo da função como **void** (tipo nulo):
  - Neste caso, a função não precisa retornar nenhum valor

Notação em C

- **Exemplo 1**: procedimento em C.

```
Programa Principal!
 Ola!!!
Posso chamar de novo:
 Ola!!!
```

```c
#include <stdio.h>

void Ola() //Declaração da função: "(" ")" obrigatórios
{
        printf("\n Ola!!! \n"); //Corpo da função
}

int main() { // Programa principal
        printf("Programa Principal!");
        Ola(); //Chamada da função

        printf("Posso chamar de novo:");
        Ola(); //Atenção: "(" ")" são obrigatórios!

        return 0;
}
```

Notação em C

- **Exemplo 2**: Função soma sem parâmetros.

```
#include <stdio.h>

int soma1010() //Declaração da função
{
        int A;
        A = 10+10;
        return A; //Indica o valor a ser retornado
                  //A função é encerrada no return
}

int main() {
        int res = soma1010() + soma1010();
        printf("Resultado = %d\n", res);

        return 0;
}
```

Resultado = 40

Notação em C

- O comando ***return*** indica o valor que uma função deve retornar;
- Os argumentos de ***return*** podem ser:
  - Constantes:
    - **Ex.**: return 1; // para uma função do tipo int.
    - **Ex.**: return (3.14); // para uma função do tipo float.
  - Variáveis:
    - **Ex.**: return A; // função do mesmo tipo de A.
  - Expressões:
    - **Ex.**:return (i+(A-4)*3); // função do mesmo tipo retornado
      // pela expressão.

Notação em C

- **Exemplo 3**: Função soma que recebe parâmetros.

```
#include <stdio.h>

int soma(int X, int Y)//Parâmetros sempre entre "(" ")"
{
        return (X+Y);
}

int main() {
        printf("Resultado = %d\n", soma(15, 25));

        return 0;
}
```

```
Resultado = 40
```

Notação em C

- O uso de parâmetros torna as funções mais ***flexíveis***;
  - Podem ser aplicadas a diferentes valores de dados.
- A ***lista de parâmetros*** de uma função deve estar entre "(" ")" e logo após o nome da função;
- Cada parâmetro deve ser **precedido** de seu tipo (*int*, *float*, ...);
- Caso haja mais de um parâmetro, eles devem ser separados por vírgula;
- **Exemplos**:
  - float divide(int a, int b) {...}
  - float sen(float x) {...}
  - char maiuscula(char letra) {...}

A FUNÇÃO *MAIN*()

A função *main*()

- Como vocês devem ter percebido, o programa principal em C também é uma ***função*** → função ***main*()**;
- Como toda função em C, ela também deve ter um ***tipo*** associado e, no final, **retornar** um valor;
  - A partir do padrão ANSI C 99: ***main*** deve ser do tipo ***int***.
  - Alguns compiladores aceitam ***main*** como sendo ***void***.

```
int main() // Declaração da função - Programa Principal {
        ...
        return 0;  // Retorno de um valor
}
```

A função *main*()

- A função ***main*()** também pode ser definida com parâmetros;
  - Dois parâmetros: um ***int*** e um vetor de ***strings***.
    - O primeiro indica o número de *strings* que foram passadas para ***main()***.
  - A passagem das *strings* se dá através da chamada ao programa na linha de comando.

```
#include <stdio.h>

int main (int argc, char *argv[]) {
        int i;

        for (i=0; i < argc; i++)
                puts(argv[i]);
        return 0;
}
```

```
./tmain teste1 teste2
./tmain
teste1
teste2
```

Note que argv[0] corresponde ao comando de chamada ao programa

LOCALIZAÇÃO DAS FUNÇÕES EM UM PROGRAMA

Localização das funções

- A princípio, devemos adotar a seguinte regra:

  "Toda função deve ser declarada antes de ser usada".

- Nos exemplos visto até agora, todas as funções foram ***definidas*** antes da função ***main*()**;
  - Na **definição** de uma função está implícita a sua **declaração**.
- Alguns programadores preferem que o programa principal esteja no ***início*** do arquivo-fonte:
  - C permite que se ***declare uma função antes de defini-la***;
  - Isto é feito através do ***protótipo*** da função:
    - O protótipo da função nada mais é do que o trecho de código que especifica o **nome e os parâmetros da função**.

Localização das funções

- **Exemplo 4**: protótipo de funções.

```
#include <stdio.h>

int soma(int X, int Y); //Protótipo da função

int main() { // Programa principal
        printf("Resultado = %d\n", soma(15, 25));

        return 0;
}

int soma(int X, int Y)//Definição da função
{
        return (X+Y);
}
```

```
Resultado = 40
```

UM ÚLTIMO EXEMPLO

Exemplo

- **Exemplo 5**: cálculo da combinação de ***n*** eventos em conjuntos de ***s*** eventos, ***s*** ≤ ***p***:

$$C_s^m = \binom{n}{s} = \frac{n!}{s! \cdot (n-s)!}$$

```
#include <stdio.h>

double fatorial(int x); //Protótipo da função
int combinacao(int a, int b); //Protótipo da função

double fatorial(int x)//Definição da função
{
        double fat = 1.0;
        int i;
        for (i=x; i>1; i--)
                fat = fat*i;
        return fat;
} //continua
```

Exemplo

```
//continuação
int combinacao(int a, int b)//Definição da função
{
        return fatorial(a)/(fatorial(b) * fatorial(a-b));
}

int main() { //Programa principal
        int p, s;

        printf("Digite os valores de p e s: ");
        scanf("%d %d",&p, &s);

        if ((p >= 0) && (s >= 0) && (s <= p))
           printf("Combinação = %d\n", combinacao(p,s));
        else
           printf("Valores inválidos");

        return 0;
}
```

EXERCÍCIOS

Exercícios

1. Escreva um programa que receba o valor do raio de uma esfera e retorne o seu volume. O cálculo do volume da esfera deve ser feito em uma **função** chamada **volume_esfera** definida ***antes*** da função *main*().

2. Reescreva o programa 1 de forma que a definição da função **volume_esfera** seja feita ***após*** a definição da função *main*().

3. Escreva uma função **media** que receba duas notas (tipo *float*) e um caractere. Caso este caractere seja 'A', a função deve retornar a ***média aritmética*** das duas notas. Caso seja 'B', a ***média geométrica*** (raiz quadrada do produto das duas notas) deve ser retornada. Escreva também um programa que leia os valores das duas notas e do caractere, chame a função **media** e imprima o resultado na tela. A função **media** deve ser definida ***após*** a função *main*().

Exercícios

4. Escreva um procedimento que receba o peso e a altura de uma pessoa, obtenha seu IMC (peso dividido pela altura ao quadrado) e imprima na tela a classificação desta pessoa conforme a tabela a seguir. O IMC deve ser calculado por uma outra função que também receberá o peso e a altura da pessoa. Caberá ao programa principal ler os valores de peso e altura e chamar as funções adequadamente.

Condição	IMC
Abaixo do Peso	IMC < 18,5
Peso Normal	18,5 <= IMC <= 25,0
Acima do Peso	25,0 < IMC <= 30,0
Obeso	IMC > 30,0

Exercícios

5. Escreva um programa que simule o funcionamento de uma calculadora com cinco operações (+, -, *, / e ^). Este programa deverá ler os operandos e o operador (no formato **numero operador numero**) e exibir o resultado da operação. O operador ^ corresponde ao operador “elevado a” (potência). Cada operador deve ser implementado em uma **função própria**, definida **após** o *main*().

Exercícios

6. Escreva um procedimento que receba o **número total de dias** de um mês e o **dia da semana** (código numérico) em que começa tal mês e imprima o calendário para este mês, como ilustrado abaixo. O programa principal deve ler estes dois valores.

Exemplo:

- Número total de dias: 31
- Início do mês: terça-feira (código 2).

```
|-----------------------------|
| dom seg ter qua qui sex sab |
|  --  --  01  02  03  04  05 |
|  06  07  08  09  10  11  12 |
|  13  14  15  16  17  18  19 |
|  20  21  22  23  24  25  26 |
|  27  28  29  30  31  --  -- |
|-----------------------------|
```

Exercícios

- **Observação**:
  - Os seguintes exercícios devem ser entregues via SuSy.
    - Exercício 1;
    - Exercício 3;
    - Exercício 5.
  - Veja os enunciados atualizados no site do sistema:
    - https://susy.ic.unicamp.br:9999/si100a (Turma A);
    - https://susy.ic.unicamp.br:9999/si100b (Turma B).

Referências

Referências

- MIZRAHI, V. V., *Treinamento em Linguagem C – Curso Completo – Módulo 1*. 2a Edição, Pearson Makron Books, 2005.
- ASCENCIO, A. F. G. & DE CAMPOS, E. A. V., *Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++*. Pearson Prentice Hall, 2003.