Programação de Computadores I

UNDÉCIMA PARTE

CAPÍTULO V

SUBPROGRAMAS
& PONTEIROS

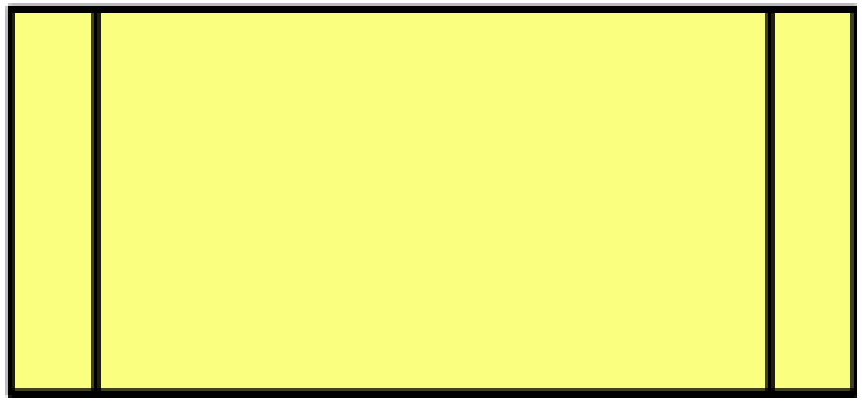
PARTE I –
SUBPROGRAMAS &
FUNÇÕES EM C

Programação de Computadores I

prof. Marco Villaça

SUBPROGRAMAS
DEFINIÇÃO

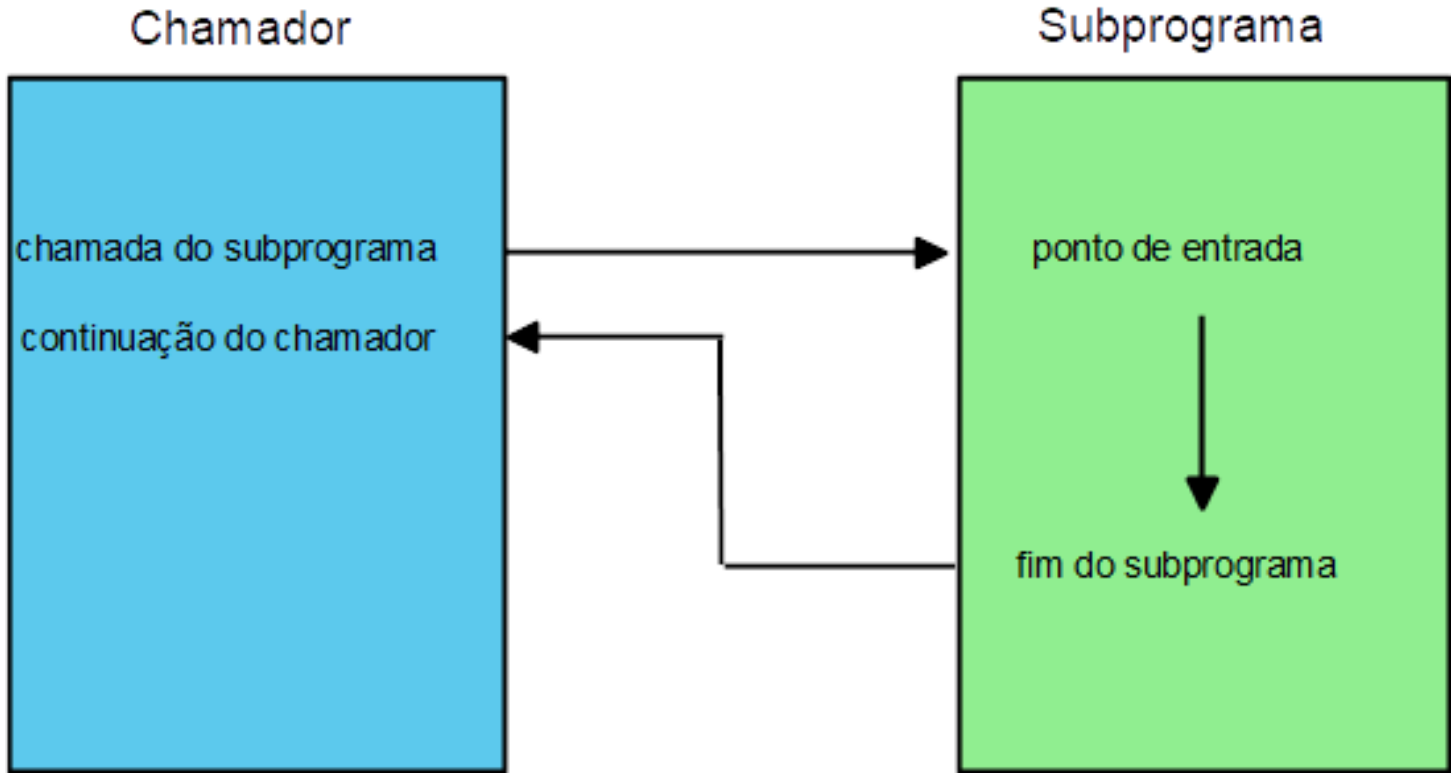
- Um subprograma é um conjunto de instruções desenhadas para cumprir uma tarefa particular;
- Subprogramas dividem grandes tarefas de computação em tarefas menores;
- Permitem que os outros programadores os utilizem em seu programas;
- Evita que o programador tenha que escrever o mesmo código repetidas vezes;
  - O conceito economiza espaço e esforço de desenvolvimento e codificação



SUBPROGRAMAS
SÍMBOLO PARA FLUXOGRAMAS

SUBPROGRAMAS

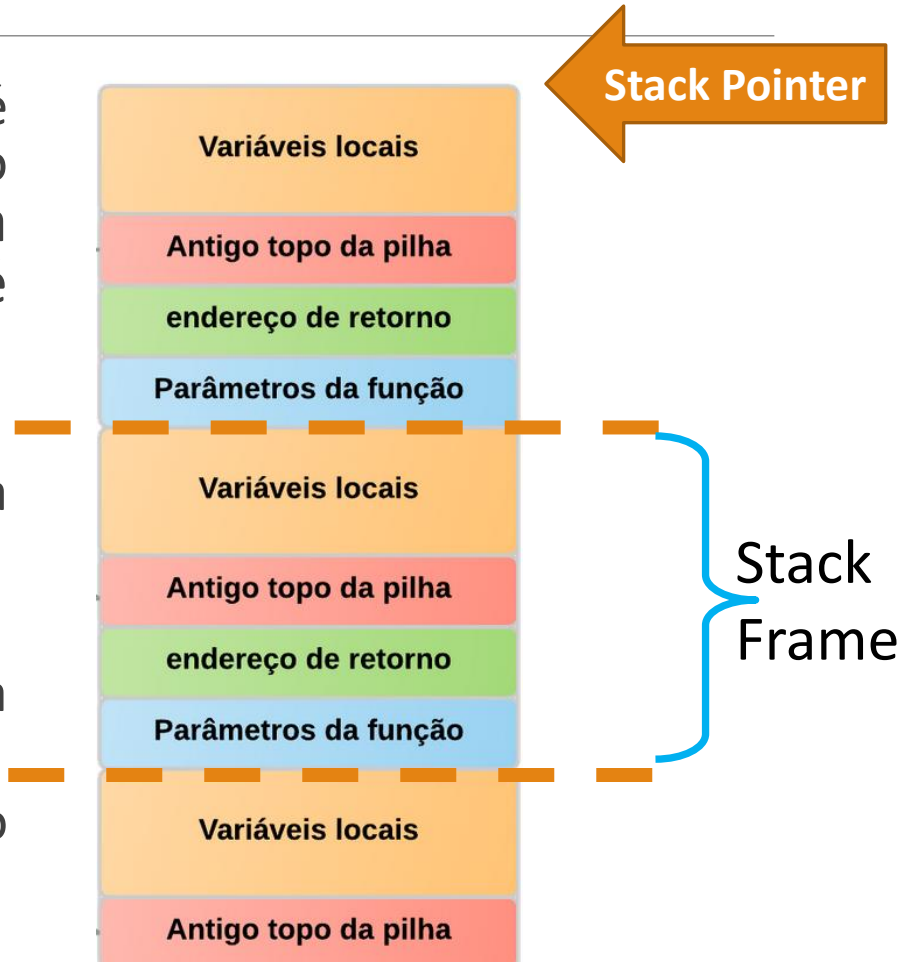
- Cada subprograma tem um único ponto de entrada
- A unidade de programa chamadora é suspensa durante a execução do subprograma chamado
- O controle sempre retorna para o chamador quando a execução do subprograma termina



SUBPROGRAMAS

Pilha de funções

- Quando uma função é chamada, um bloco de memória é empilhado no topo da pilha de funções. Cada bloco empilhado é chamado de *Stack frame.* Ao término da execução da função, esse bloco é desempilhado/desalocado.
- Dentro de cada bloco tem-se
  - As variáveis para os parâmetros passados para a função
  - O endereço de retorno para a função anterior
  - um ponteiro chamado *frame pointer* que representa onde o topo da pilha estava anteriormente
  - uma área para variáveis criadas dentro da função (variáveis locais).



SUBPROGRAMAS
Tipos

- Procedimentos:
  - Conjunto de instruções parametrizadas que definem uma determinada computação
  - Não retornam valores
- Funções:
  - Similar aos procedimentos, porém geralmente modelam funções matemáticas
  - Retorna um valor para o chamador

SUBPROGRAMAS

- Tanto em C, como em outras linguagens, um código executável é criado a partir de um e somente um programa principal, o qual pode invocar subprogramas.
- Em C, já utilizamos várias funções:
  - Em C, foram usadas funções das bibliotecas-padrão:

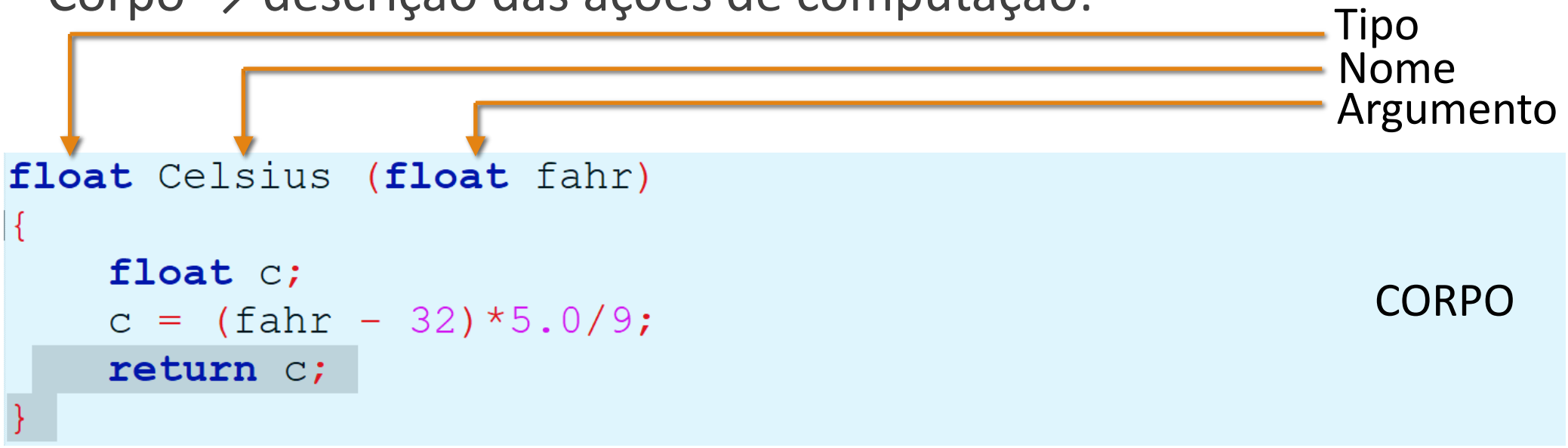
```
printf, scanf, strcmp, pow, ...
```

SUBPROGRAMAS

- Subprograma:
  - Cabeçalho → tipo + nome + argumentos formais

+

  - Corpo → descrição das ações de computação:



FUNÇÃO SIMPLES EM C

```
#include <stdio.h>
 float Celsius(float);                /* Protótipo ou declaração da função */
 int main() {
      float c, f;
      printf("Digite a temperatura em graus Fahrenheit: ");
      scanf("%f", &f);
      c = Celsius(f);                 /* Chamada da função */
      printf("Celsius = %.2f\n", c);
      return 0;  }
 /* Definição da função */
 float Celsius(float fahr) {
      float c;
      c = (fahr -32.0) * 5.0/9.0;
      return c; }
```

ARGUMENTO: VALOR PASSADO PARA A FUNÇÃO

RETORNO: VALOR DEVOLVIDO PELA FUNÇÃO

FUNÇÕES EM C

- Definição

```
tipo_retorno  Nome_funcao(lista de argumentos)
{
   declaracao de variaveis;
   comandos;
   return(dado_a_retornar);
}
```

- Dessa forma, em C, só há **um** retorno
  - ✓ É possível ter mais de um, se usarmos ponteiros (veremos adiante)

PROTÓTIPOS DE FUNÇÕES EM C

- Uma função não pode ser chamada sem antes ter sido declarada, **assim se a função main( ) usar funções definidas depois dela**, é necessário definir um protótipo
- A declaração de uma função é dita protótipo da função
  - ✓ Instrução geralmente alocada no início do programa
  - ✓ Estabelece o tipo da função e os argumentos que ela recebe
- O **protótipo da função** permite que o compilador verifique a sintaxe de chamada à função

```
float Celsius(float);
```

- Essa declaração informa que a função de nome Celsius( ) é do tipo float e recebe como argumento um valor float.

PROTÓTIPOS DE FUNÇÕES EM C

- Há duas formas de declarar funções em C
- A mais usada é chamada de protótipo externo
  - ✓ É escrita antes de qualquer função
  - ✓ A declaração é feita uma única vez e é visível para todas as funções do programa
- A outra forma é denominada de protótipo local
  - ✓ É escrita no corpo de todas as funções que a chamam, antes da sua chamada

FUNÇÕES SEM ARGUMENTOS EM C

- Função pode ter qualquer número de argumentos, inclusive zero

```
ret=Imprime_menu(); //chamada da função
```

- No caso de zero argumentos, usa-se a palavra-chave `void` para indicar uma lista de argumentos vazia

```
int Imprime_menu(void); //protótipo
```

PROCEDIMENTOS EM C
Funções void

- `void` também é usado para indicar que a função não tem retorno
- `void` é um dos tipos básicos em C
  - ✓ Indica tamanho zero

```
void Imprime_resposta(int resposta)
{
if (resposta == 0)
{
     printf("Resposta inválida\n");
     return;
}
printf("A resposta é %d\n", resposta);
}
```

EXEMPLO 1

- Fazer uma função para cálculo da área de um círculo e montar um programa exemplo

Exemplo 2

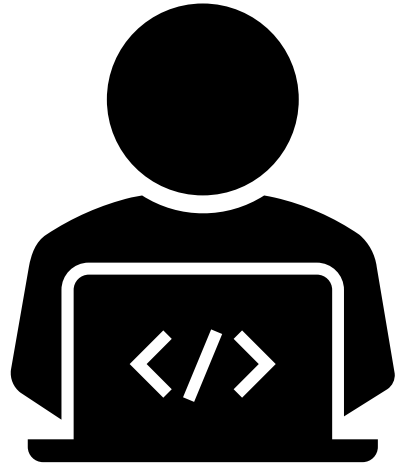
Fazer uma função para cálculo da resistência equivalente da associação de 2 resistores (série ou paralelo) e montar um programa exemplo.

- A função deve receber os valores dos resistores mais um caractere que será s para série e p para paralelo

Exercício 1

Fazer uma função para calcular a média aritmética de 3 valores

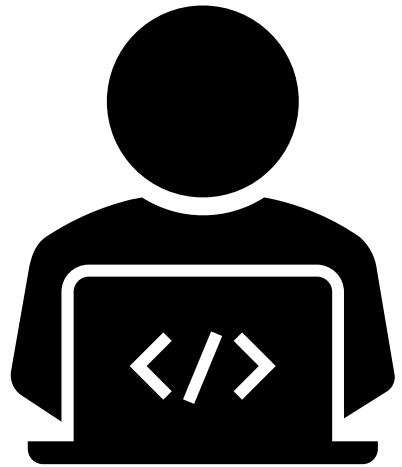
Montar um programa exemplo para pedir ao usuário os valores, repassá-los para a função e depois apresentar o resultado devolvido pela função.



Exercício 2

Elaborar uma função que, chamada a partir da rotina principal, imprima quantos dias tem um mês lido no programa principal em função de um valor recebido. Montar um programa exemplo.

Por exemplo se receber mês = 1, a função imprime “o mês tem 31 dias”.



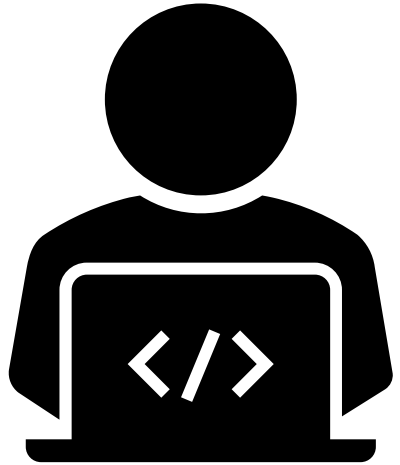
Exercício 3

Elaborar uma função que, chamada a partir da rotina principal, calcule o imc de uma pessoa.

O imc calculado é devolvido para o programa principal que repassa o valor para outra função, que recebendo o imc imprime uma informação:

- Se imc <18,5, imprime - Abaixo do peso
- Se 18,5 ≤ imc < 25, imprime – Peso normal
- Se 25 ≤ imc < 30, imprime – Sobrepeso
- Se imc ≥ 30, imprime – Obeso

Montar um programa exemplo.



Bibliografia e crédito das figuras

OUALLINE, S. Practical C Programming. 3a ed. O'Reilly, 1997.

SEBESTA, R. Conceitos de Linguagens de Programação. 5a ed. Porto Alegre: Bookman, 2003.

http://help.scilab.org/docs/6.1.0/pt_BR/index.html