UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Fundamentos de Programação

CP41F

Declaração de funções. Funções sem e com retorno. Parâmetros de funções. Ponteiros para funções.

Aula 13
Prof. Daniel Cavalcanti Jeronymo

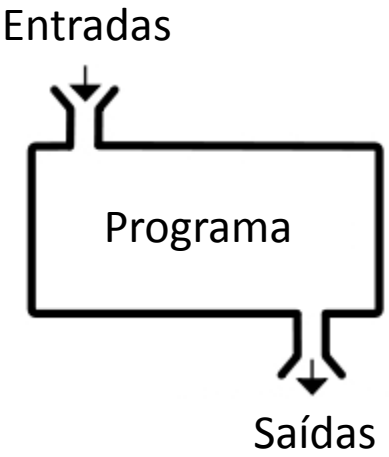
Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia de Computação – 1º Período
2016.1

Plano de Aula

- Declaração de funções.
- Funções sem e com retorno.
- Parâmetros de funções.
- Parâmetros variáveis.
- Ponteiros para funções.

Declaração de funções

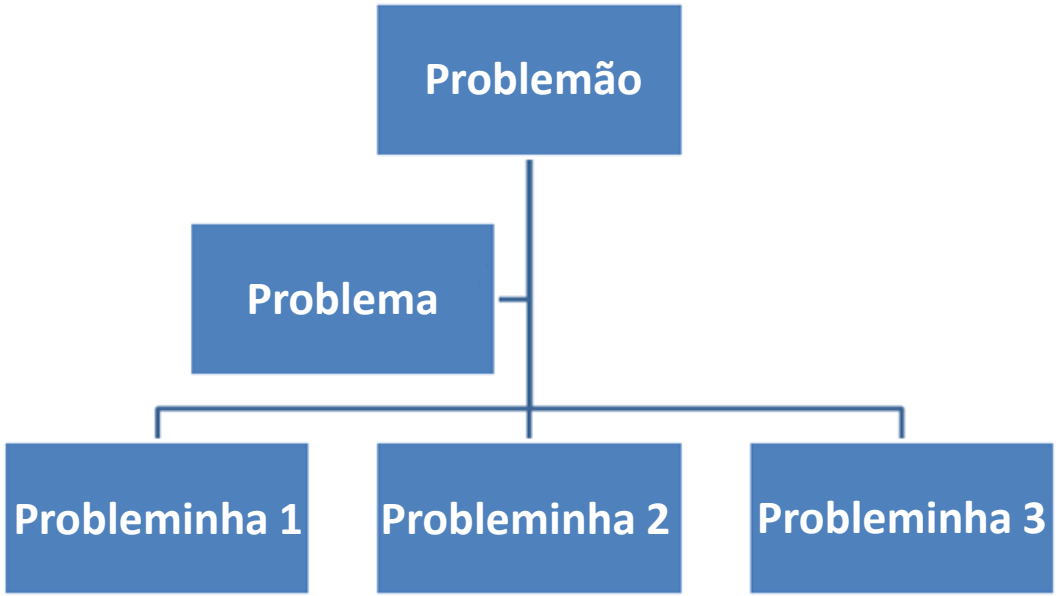
- Etapas no projeto de um programa



1. Definir o problema
2. Determinar requisitos / especificações
3. Definir entradas e saídas
4. Desenvolver a lógica do programa utilizando pseudo-código e/ou um diagrama de fluxo
5. Desenvolver o código – utilizar comentários!
6. Testar o programa – considerar todas as entradas possíveis
7. No caso de bugs, identificar o problema e dependendo do caso voltar para uma das etapas 1, 2, 3, 4 ou 5

Declaração de funções

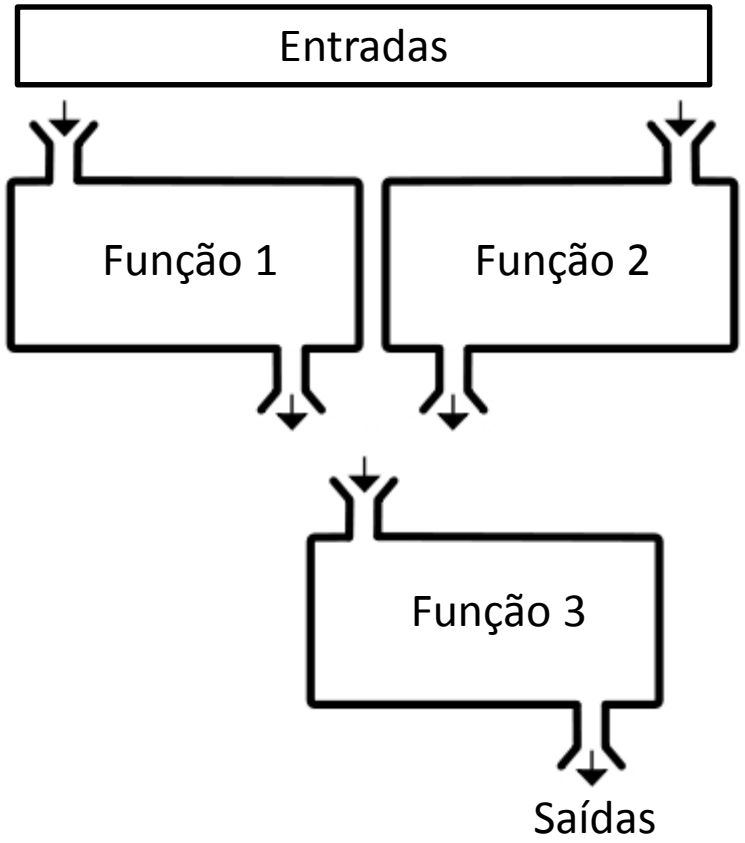
- Desenvolver a lógica e código
  - O que fazer se o problema for muito grande?
- Dividir o problema em subproblemas
  - Decomposição (*decomposition* ou *factoring*)



Outra boa prática: conjuntos reduzidos de testes!
Primeiro teste seu programa para entradas cujas saídas possam ser verificadas manualmente, depois teste para entradas generalizadas.

Declaração de funções

- Quebrar um problema em pedaços pequenos
  - Pedaços menores são conhecidos como módulos, sub-rotinas, procedimentos ou **funções**
  - Porque?
    - Ajuda a gerenciar complexidade
      - Blocos menores de código
      - Leitura e compreensão facilitadas
    - Encoraja reutilização de código
      - No mesmo programa ou códigos externos
    - Permite desenvolvimento independente de código
    - Fornece uma camada de abstração



Declaração de funções

- Funções são o bloco fundamental de um programa em C
  - Já utilizamos funções pré-definidas anteriormente:
  - main
  - printf, scanf, strlen, etc

- Como declarar e definir funções?

Declaração de funções

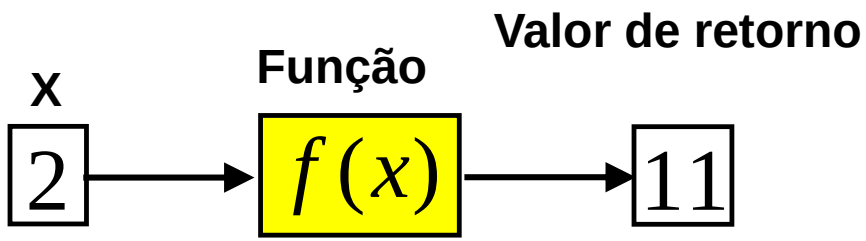
- Etimologia - função
  - Herança das origens matemáticas da programação

$$f(x) \overset{\Delta}{=} x^2 + 2x + 3$$

Quanto é f(2)?

$$f(2) \Rightarrow (2)^2 + 2(2) + 3 \Rightarrow 4 + 4 + 3 \Rightarrow 11$$

$$\therefore f(2) \text{ é } 11$$



- Uma função em C não precisa retornar valores
- Uma função em C não precisa alterar o estado interno do programa (exemplo: printf)
  - Funções vs Procedimentos

Declaração de funções

- Sintaxe para definição de funções

```
tipo_retorno nome_função(tipo1 var1, tipo2 var2)
{
   declarações;
}
```

- A declaração da função contém:
  - Tipo de retorno
  - Nome da função
  - Argumentos (ou parâmetros)

Declaração de funções

- Exemplo – função para multiplicação
  - Declaração da função
    - Similar a declaração de uma variável
      - Diz ao compilador que a função será definida adiante
      - Também chamada de protótipo
      - Atenção !!!
  - Definição da função
    - Ver slide anterior
    - SEM ponto-e-vírgula
  - Retorno da função
    - O comando return termina a execução da função atual
    - Controle retorna a função chamadora
    - se `return expressão;`
      - valor da **expressão** é retornado como valor da chamada da função
      - Apenas um valor pode ser retornado
  - Chamada da função
    - **main()** é a 'função chamadora'
    - **produto()** é a 'função chamada'
    - Transferência de controle para o código da função
    - Código da função é executado

```
#include <stdio.h>
/* protótipo da função */
double produto(double x, double y);

int main()
{
  double var1 = 3.0, var2 = 5.0;
  double ans;
  ans = produto(var1, var2);
  printf("var1 = %.2f\n"
         "var2 = %.2f\n",var1,var2);
  printf("var1*var2 = %g\n", ans);
  return 0;
}
/* definição da função */
double produto(double x, double y)
 {
   double result;
   result = x * y;
   return result;
 }
```

Declaração de funções

- Declaração
  - Introduz um **identificador** e seu **tipo**
  - Necessário para que o compilador aceite referências ao **identificador**
  - Variáveis globais extern, protótipos de funções ou estruturas

- Definição
  - Instancia, isto é, aloca armazenamento ou implementa, o **identificador**
  - Necessário para que o ligador conecte as entidades aos **identificadores**

Funções sem e com retorno

- Funções podem retornar um valor:
  - double produto(double x, double y);

- Como fazer funções sem retorno? Use **void** como tipo de retorno

```
void imprimir_feliz()
{
   printf(":)");
   return;
}
```

Parâmetros de funções

- Passagem de parâmetros por três tipos
  - **Por valor**
  - **Por referência**
  - **Por listas de argumentos variáveis**

Parâmetros de funções

- Passagem de parâmetros **por valor**
  - double produto(double x, double y);

- O **valor** da variável é **copiado** para memória, nesse caso na pilha, e passado para a função

Parâmetros de funções

- Passagem de parâmetros **por referência**

```
void inc(int *j)
{
   (*j)++;
}
```

- Em C não existe passagem de parâmetros por referência
- A funcionalidade é simulada com passagem por endereço
- O **valor do ponteiro** da variável é **copiado** para memória, nesse caso na pilha, e passado para a função
- A função então **dereferência** o ponteiro para ler ou escrever no parâmetro

Parâmetros de funções - variáveis

- Passagem de parâmetros **por listas de argumentos variáveis**

```
void func(char a, int b, ...);
```

- Funções variádicas aceitam um número variável de argumentos
- Exemplo: printf
- Exigências do ANSI C
  - Pelo menos um argumento antes de "..."
  - O argumento não pode ser register
  - O argumento deve ter tipo auto-promovido, isto é, promoções não podem mudar seu tipo, portanto não pode ser vetor, função, char, float ou short int.

Parâmetros de funções - variáveis

- Passagem de parâmetros **por listas de argumentos variáveis**

```
void func(char a, int b, ...);
```

- Tipo e macros:
  - va_list
  - va_start
  - va_arg
  - va_copy
  - va_end
- Exercício: Faça uma função que receba um número variável parâmetros inteiros e calcule a média

Ponteiros para funções

- Código ocupa um espaço na memória, assim como variáveis
- Portanto, funções em C podem ser endereçadas igual variáveis!

- Sintaxe:
  - int (*nome_ponteiro)(tipo1 var1, tipo2 var2)

Ponteiros para funções

- Exemplo: construir uma calculadora com quatro operações
  - Adição
  - Subtração
  - Multiplicação
  - Divisão

Ponteiros para funções

- Passo 1: Construir as funções desejadas

```
double produto(double x, double y)
{
  return x * y;
}

double soma(double x, double y)
{
  return x + y;
}

double subtracao(double x, double y)
{
  return x - y;
}

double divisao(double x, double y)
{
  return x / y;
}
```

Ponteiros para funções

- Passo 2: Abstrair a operação da escolha utilizando ponteiro pras funções

```
int main()
{
   double x=10.0, y=5.0;
   char escolha = '/';
   double (*operacao)(double x, double y) = NULL;

   /*scanf("%lf", &x);
   scanf("%lf", &x);
   scanf("%c", &c);*/

   switch(escolha)
   {
      case '+': operacao = soma; break;
      case '-': operacao = subtracao; break;
      case '/': operacao = divisao; break;
      case '*': operacao = produto; break;
   }

   printf("Operacao: %.2f %c %.2f\n", x, escolha, y);
   printf("Resultado: %.2f\n", operacao(x,y));

   return 0;
}
```

Ponteiros para funções

- Exemplos de aplicações:
  - *Callbacks*
  - Simular classes e objetos em C
  - Tabelas de saltos (*jump tables* ou *branch tables*)
  - Padronização de funções (exemplos: qsort e bsearch)
  - Design Patterns
  - Camadas de abstração (exemplo da calculadora)
  - Etc (muito mais!)