3. Linguagem de Programação C

Prof. Renato Tinós

Departamento de Computação e Matemática
(FFCLRP/USP)

Principais Tópicos

3.7. Funções

3.7.1. Introdução
3.7.2. Uso de Funções
3.7.3. Passagem de Parâmetros
3.7.4. Funcionamento
3.7.5. Exercícios

3.7.1. Introdução

- **Função ou Procedimento ou Sub-Rotina**
  - Agrupa um conjunto de comandos e associa a ele um nome
    - O uso deste nome é uma chamada da função
  - Após sua execução, programa volta ao ponto do programa situado imediatamente após a chamada
    - A volta ao programa que chamou a função é chamada de retorno

3.7.1. Introdução

- **A chamada de uma função pode passar informações (argumentos) para o processamento da função**

- **Argumentos = lista de expressões**
  - Lista pode ser vazia
  - Lista aparece entre parênteses após o nome da função
    - Ex. *int Soma(int x, int y) {*

3.7.1. Introdução

- **Retornando resultados de funções**
  - No seu retorno, uma função pode retornar resultados ao programa que a chamou
    - Exemplo: *return (resultado);*
      - O valor da variável local *resultado* é passado de volta como o valor da função
    - Valores de qualquer tipo podem ser retornados
      - Funções: retornam valores
      - Procedimentos: não retornam valores
        - » Exemplo: *void function (int x)*

3.7.1. Introdução

- **Comando return**
  - Encerra a execução da função
    - No caso de procedimentos, é opcional
  - Procedimentos
    - return;
  - Funções
    - return (expressão);
    - Expressão não precisa estar entre parênteses

3.7.2. Uso de Funções

- **Definições e protótipos de funções**
  - Funções são definidas de acordo com a seguinte sintaxe:

```
tipo nome (lista de parâmetros)
{
        corpo de função
}
```

3.7.2. Uso de Funções

- **Definições e protótipos de funções**
  - Tipo de resultado
    - Quando a função é um procedimento (ou seja, não retorna nenhum valor), usa-se a palavra chave *void*
    - Procedimento não retorna valor
  - Lista de parâmetros
    - Funcionam como variáveis locais com valores iniciais
    - Quando função não recebe parâmetros, a lista de parâmetros é substituída pela palavra *void*

3.7.2. Uso de Funções

- **Protótipos**
  - Declaração
    - Antes de usar uma função em C, é aconselhável declará-la especificando seu protótipo
      - Tem a mesma forma que a função, só que substitui o corpo por um (;)

3.7.2. Uso de Funções

- **Funcionamento de uma chamada:**
  - Cada expressão na lista de argumentos é avaliada
  - O valor da expressão é convertido, se necessário, para o tipo de parâmetro formal
    - Este tipo é atribuído ao parâmetro formal correspondente no início do corpo da função
  - O corpo da função é executado

3.7.2. Uso de Funções

- **Funcionamento de uma chamada:**
  - Se um comando *return* é executado, o controle é passado de volta para o trecho que chamou a função
    - Se um comando *return* inclui uma expressão, o valor da expressão é convertido, se necessário, pelo tipo do valor que a função retorna
      - O valor então é retornado para o trecho que chamou a função
      - Se um comando *return* não inclui uma expressão nenhum valor é retornado ao trecho que chamou a função

3.7.2. Uso de Funções

- **Funcionamento de uma chamada:**
  - Se não existir um comando *return*, o controle é passado de volta para o trecho que chamou a função após o corpo da função ser executado

3.7.2. Uso de Funções

```
...
float media (float prova_1, float prova_2, float trabalho) {
      float final;

      if (trabalho > 5){
            final = 0.3*trabalho + 0.7*(prova_1+prova_2)/2 ;
      }
      else{
            final = 0.5*trabalho + 0.5*(prova_1+prova_2)/2 ;
     }
     return final;
}

void main(void) {
      ...
      for (i=0;i<nro_alunos;i++) {
            media_aluno[i]=media(prova_1[i],prova_2[i],trabalho[i]);
     }
     ...
}
```

3.7.3. Passagem de Informações

- **Ocorre através da declaração de argumentos dentro da declaração**
  - Declaração de argumentos informa o número e tipo dos argumentos
  - Conjunto de argumentos forma uma lista de declarações de variáveis separadas por vírgulas
    - *Ex.: double bolsa, double notas, int idade*

3.7.3. Passagem de Informações

- **Passagem de informações por valores**
  - Argumentos são passados como valores
    - Quando chamada, a função recebe o valor da variável
      - A passagem por valor significa que a função não pode mudar seu valor
    - Os argumentos deixam de existir após a execução do método

3.7.3. Passagem de Informações

- **Variáveis locais e globais**
  - A variável local só existe durante a vida da função
  - Mesmo nome para variáveis locais em diferentes funções significam diferentes variáveis
  - No entanto, se a variável for global, ela vale para toda a função
    - Não se aconselha a utilização excessiva de variáveis globais
      - Torna mais difícil manutenção do programa
      - Torna mais difícil a busca por erros nos programas

3.7.3. Passagem de Informações

- **Passagem de informações por referência**
  - Uso de ponteiros
  - Possibilita alterar as variáveis na função original
    - Exemplo

```
...
void func (int *a) {
    ...
}

main() {
     int a;
     ...
            func(&a);
    ...
}
```

3.7.3. Passagem de Informações

- **Passagem de vetores como parâmetros**
  - Em C, vetores podem ser passados como parâmetros
    - Tamanho do vetor não precisa ser informado na declaração da função
      - Passar junto uma variável com o tamanho
    - Funciona como passagem por referência
      - Alteração em um valor do parâmetro, muda automaticamente o valor correspondente no argumento

3.7.3. Passagem de Informações

```
...
double media (double ind_notas[], int n){
        int i;
        double total = 0.0;

        for (i = 0; i < n; i++){
                total += ind_notas[i];
        }
        return (total / n);
}
...
```

3.7.3. Passagem de Informações

- **Relacionamento entre ponteiros e vetores**
  - Para o compilador, não existe diferença entre vetor e ponteiro quando eles são passados como parâmetros
  - Dentro do computador, declarações *vetor[ ]* e **vetor* são equivalentes
  - Declaração de parâmetros deve refletir o seu uso

3.7.3. Passagem de Informações

- **Se uma função recebe um vetor de uma dimensão, o parâmetro formal pode ser declarado de três formas:**

```
func (int *x){
    .
    .
    .
}
```

```
func (int x[10]){
    .
    .
    .
}
```

```
func (int x[]){
    .
    .
    .
}
```

3.7.3. Passagem de Informações

- **Os três métodos de declaração têm o mesmo resultado**
  - Cada um deles avisa ao compilador que um apontador para inteiro será recebido
    - Primeira declaração usa um ponteiro
    - Segunda declaração usa uma declaração padrão de vetor
    - Terceira declaração avisa que um vetor do tipo *int* de tamanho indefinido será recebido

3.7.3. Passagem de Informações

- **Uma declaração do tipo**

  func (int val[32]){

  - Teria o mesmo funcionamento
  - Compilador gera código instruindo *func()* a receber um ponteiro
    - Não cria um vetor com 32 elementos

3.7.3. Passagem de Informações

```
...
int Soma (int vetor[], int n) {
        int i, soma;

        soma = 0;
        for (i = 0; i < n; i++)
                soma += vetor[i];
        return (soma);
}

main(){
        int tam, v[100];
        ...
        sum = Soma ( v ,  tam );
        ...
```

```
int Soma (int *vetor, int n) {
```

3.7.4. Funcionamento

- **Mecanismo do processo de chamada de função**

1. Valor dos argumentos é calculado pelo programa que está chamando a função
2. Sistema cria nova espaço para todas as variáveis locais da função (estrutura de pilha)
3. Valor de cada argumento é copiado na variável parâmetro correspondente na ordem em que aparecem
   3.1 Realiza conversões de tipo necessárias

3.7.4. Funcionamento

- **Mecanismo do processo de chamada de função**

4. Comandos do corpo da função são executados até:
   4.1 Encontrar comando *return*
   4.2 Não existirem mais comandos para serem executados
5. O valor da expressão *return*, se ele existe, é avaliado e retornado como valor da função
6. Pilha criada é liberada
7. Programa que chamou continua sua execução

3.7.4. Funcionamento

- **Projeto top-down**
  - Procedimentos e funções permitem dividir um programa em pedaços menores
    - Facilita sua leitura
  - É chamado de processo de decomposição
    - Estratégia de programação fundamental
    - Encontrar a decomposição certa não é fácil
      - Requer experiência

3.7.4. Funcionamento

- **Projeto *top-down***
  - Melhor estratégia para escrever programas é começar com o programa principal
    - Pensar no programa como um todo
    - Identificar as principais partes da tarefa completa
      - Maiores pedaços são candidatos a funções
      - Mesmos estas funções podem ser decompostas em funções menores
      - Continuar até cada pedaço ser simples o suficiente para ser resolvido por si só

3.7.5. Exercícios

Exercício 3.7.1. Escrever um programa C, sem utilizar funções, que

a) Leia três conjuntos de *n* números reais digitados pelo usuário (*n* pode ser diferente para cada conjunto);

b) Imprima a média e o desvio padrão de cada um dos três conjuntos;

c) Imprima a média e o desvio padrão das médias calculadas no item b).

Exercício 3.7.2. Repetir o exercício anterior utilizando funções.

3.7.5. Exercícios

Exercício 3.7.3. Escrever um programa C, utilizando funções, que

a) Leia *n* números inteiros digitados pelo usuário;
b) Imprima o máximo e o mínimo valor digitado.