Nota: Este material complementar, disponível em https://prettore.github.io/lectures.html representa uma cópia resumida de conteúdos bibliográficos disponíveis gratuitamente na Internet.

Funções

Introdução

As funções são componentes fundamentais na programação, sendo essenciais para a criação de códigos mais limpos, organizados e reutilizáveis. Elas permitem dividir um programa em blocos menores e mais gerenciáveis, facilitando a manutenção e o desenvolvimento de software. Este material explora os conceitos, características e aplicações de funções em programação, fornecendo uma base sólida para sua utilização em diferentes contextos.

Conceito de Função

Uma função é um bloco de código projetado para realizar uma tarefa específica, que possui um nome associado e que pode ser chamado quando necessário. Ao final da execução da função, o controle retorna para quem a chamou. As funções permitem o reaproveitamento de código, evitando a repetição de instruções e tornando o programa mais eficiente e organizado.
Em termos práticos, uma função pode ser vista como um subprograma para o qual podemos repassar dados de entrada através de parâmetros e receber os resultados através do retorno da função. Essa abordagem

segue o princípio da modularização, que consiste em dividir um programa em partes que serão desenvolvidas separadamente.

Principais Vantagens do Uso de Funções

- Reutilização de código: Elimina a repetição de código (princípio DRY - "Don't Repeat Yourself").
- Organização: Divide o código em blocos menores e mais fáceis de gerenciar.
- Facilidade de manutenção: Mudanças em uma função precisam ser feitas apenas em um lugar, sem afetar todo o programa.
- Abstração: Permite focar no "o que" a função faz, não no "como" ela faz.
- Colaboração: Facilita a divisão de tarefas em um projeto, onde diferentes desenvolvedores podem trabalhar em diferentes funções.
- Testabilidade: Funções podem ser testadas isoladamente antes de serem integradas ao programa principal.

Definição, Declaração e Chamada de Funções

É importante distinguir três momentos diferentes no uso de funções:

- Definição de função: É quando se constrói a função explicitando o seu nome, tipo de retorno, parâmetros e o seu corpo (instruções e variáveis locais).
- Declaração da função (protótipo): É somente uma declaração contendo o nome da função, tipo de retorno e parâmetros. O corpo da função não é explicitado, pois supõe-se que ela foi definida em algum outro lugar.
- Chamada (invocação) da função: Quando se usa a função, chamando-a pelo seu nome e passando os parâmetros que se deseja.

Sintaxe Básica de uma Função

A forma geral da definição de uma função varia de acordo com a linguagem de programação, mas geralmente segue um padrão similar. Abaixo está um exemplo em C:

```c
tipo_de_retorno nome_da_função(tipo_param1 param1, tipo_param2 param2, ...) {
    // Declarações de variáveis locais
    // Instruções
    return valor_de_retorno; // Se a função retornar um valor
}
```

Os principais componentes da sintaxe são:

- Tipo de retorno: Especifica o tipo do valor que a função retorna (int, float, char, etc.) ou void se nenhum valor é retornado.
- Nome da função: Deve ser descritivo, seguindo as convenções de nomenclatura da linguagem.
- Parâmetros: Valores que são passados para a função quando ela é chamada.

- Corpo da função: Bloco de código delimitado por chaves que contém as instruções a serem executadas.
- Instrução return: Usada para retornar um valor ao chamador da função (exceto em funções void).

Parâmetros e Argumentos

Os parâmetros são variáveis que a função recebe para realizar suas operações. Eles são definidos na declaração da função e podem ser de qualquer tipo de dados suportado pela linguagem. Os valores reais passados para esses parâmetros quando a função é chamada são chamados de argumentos.
Existem duas formas principais de passagem de parâmetros:

- Passagem por valor: O valor do argumento é copiado para o parâmetro da função. Alterações no parâmetro dentro da função não afetam o argumento original.
- Passagem por referência: É passada uma referência (ou endereço) do argumento para a função. Alterações no parâmetro dentro da função afetam o argumento original.

Exemplo de função com parâmetros em C:

```c
int somar(int a, int b) {
    return a + b;
}
// Chamada da função
int resultado = somar(5, 3); // resultado = 8
```

Valor de Retorno

Frequentemente, uma função faz algum tipo de processamento ou cálculo e precisa retornar o resultado desse procedimento. O valor de retorno é especificado usando a instrução return, seguida do valor a ser retornado. O tipo desse valor deve corresponder ao tipo de retorno declarado na definição da função.
Uma função pode ter múltiplos pontos de retorno, mas uma vez que um return é executado, a função termina imediatamente, e o controle retorna ao chamador.
Exemplo de função com retorno em C:

```c
int quadrado(int x) {
    return x * x;
}
// Chamada da função
int resultado = quadrado(4); // resultado = 16
```

Funções e Procedimentos

Em algumas linguagens e contextos, faz-se uma distinção entre funções e procedimentos:

- Função: Retorna um valor para o chamador.

- Procedimento: Não retorna valor, apenas executa uma série de instruções.

Em C, um procedimento é implementado como uma função com tipo de retorno void:

```c
void imprimirMensagem() {
    printf("Olá, mundo!
");
    // Não há instrução return com valor
}
```

Protótipos de Funções

Em linguagens como C e C++, é comum declarar protótipos de funções no início do programa ou em arquivos de cabeçalho. Um protótipo informa ao compilador sobre a existência da função, seu tipo de retorno e seus parâmetros, antes que a função seja efetivamente definida ou chamada.

Exemplo de protótipo em C:

```c
// Protótipo
int somar(int a, int b);
int main() {
    int resultado = somar(5, 3);
    printf("Resultado: %d
", resultado);
    return 0;
}
// Definição da função
int somar(int a, int b) {
    return a + b;
}
```

Escopo de Variáveis em Funções

O escopo de uma variável refere-se à região do programa onde a variável é visível e pode ser acessada. Em relação às funções, podemos ter:

- Variáveis locais: São declaradas dentro de uma função e só podem ser acessadas dentro dela. Elas são criadas quando a função é chamada e destruídas quando a função termina.
- Variáveis globais: São declaradas fora de qualquer função e podem ser acessadas por qualquer função no programa.
- Parâmetros: São variáveis locais especiais que recebem os valores passados para a função.

Exemplo de escopo de variáveis em C:

```c
#include <stdio.h>
// Variável global
int global = 10;
void funcao() {
    // Variável local
    int local = 5;
    printf("Global: %d, Local: %d
", global, local);
    // local só é acessível dentro desta função
}
int main() {
    funcao();
    printf("Global: %d
", global);
    // printf("Local: %d
", local); // Erro: local não é acessível aqui
    return 0;
}
```

Recursividade

Uma função recursiva é aquela que chama a si mesma diretamente ou indiretamente. A recursividade é uma técnica poderosa para resolver problemas que podem ser divididos em subproblemas menores e similares.
Toda função recursiva deve ter uma condição de parada (caso base) para evitar recursão infinita. A recursividade pode ser uma alternativa elegante a loops iterativos em muitos casos.

Exemplo de função recursiva para calcular o fatorial:

```c
int fatorial(int n) {
    // Caso base
    if (n <= 1) {
        return 1;
    }
    // Caso recursivo
    else {
        return n * fatorial(n - 1);
    }
}
```

Funções Anônimas e Arrow Functions

Em linguagens modernas como JavaScript, Python e Java, existem conceitos como funções anônimas (lambdas) e arrow functions, que permitem definir funções de forma mais concisa, especialmente para uso como callbacks ou em operações de curta duração.
Exemplo de arrow function em JavaScript:
```javascript
// Função tradicional
function somar(a, b) {
    return a + b;
}
// Arrow function equivalente
const somarArrow = (a, b) => a + b;
```

Funções de Biblioteca vs. Funções Definidas pelo Usuário

- Funções de biblioteca: São funções pré-definidas fornecidas pela linguagem de programação ou por bibliotecas externas. Exemplos incluem printf(), scanf() em C, ou Math.sqrt() em JavaScript.
- Funções definidas pelo usuário: São funções criadas pelo programador para atender a necessidades específicas do programa.

Ambos os tipos de funções são essenciais para o desenvolvimento de software eficiente e bem estruturado.

Boas Práticas no Uso de Funções

- Nomes descritivos: Use nomes que descrevam claramente o que a função faz.
- Funções pequenas e focadas: Cada função deve realizar uma única tarefa bem definida.
- Documentação: Documente o propósito da função, seus parâmetros e valor de retorno.
- Evite efeitos colaterais: Funções não devem modificar variáveis globais ou parâmetros inesperadamente.
- Limite o número de parâmetros: Muitos parâmetros podem indicar que a função está fazendo muitas coisas.
- Consistência nos tipos de retorno: Mantenha consistência nos valores retornados em diferentes caminhos de execução.
- Teste unitário: Teste cada função isoladamente para garantir seu correto funcionamento.

Aplicações de Funções em Programação

As funções são utilizadas em praticamente todos os aspectos da programação, incluindo:

- Manipulação de dados: Processamento, transformação e validação de dados.
- Interface com o usuário: Captura de entrada, exibição de saída, menus interativos.
- Algoritmos: Implementação de algoritmos de ordenação, busca, etc.

- Acesso a recursos: Operações de arquivo, banco de dados, rede.
- Modularização: Divisão de programas complexos em componentes gerenciáveis.
- Reutilização de código: Bibliotecas e frameworks.

Conclusão

As funções são componentes fundamentais na programação que permitem a modularização, reutilização de código e organização lógica dos programas. Elas são essenciais para o desenvolvimento de software eficiente, manutenível e escalável.
Ao dominar o conceito de funções, seus parâmetros, valores de retorno e escopo, o programador adquire uma ferramenta poderosa para resolver problemas complexos de forma elegante e estruturada. A capacidade de dividir um problema em subproblemas menores e resolvê-los através de funções bem definidas é uma habilidade crucial para qualquer desenvolvedor de software.

Referências

- 📖 Livros e Apostilas
  - CORMEN, T. H. Introduction to Algorithms. MIT Press.
  - GOODRICH, M. Data Structures and Algorithms in Python.
  - Tenenbaum, A. M. Estruturas de Dados e Algoritmos em C
  - P. Feofiloff. Algoritmos em Linguagem C. Campus-Elsevier, 1a. edição, 2009 H. M. Deitel, P. J. Deitel. C - Como Programar, 6a. edição, Pearson Education, 2011.
  - B. W. Kernighan, D. M. Ritchie. The C Programming Language, 2a. edição, Prentice-Hall, 1988 [Tradução: C - A Linguagem de Programação. Editora Campus, 1989].
  - J. L. Szwarcfiter, L. Markenzon. Estruturas de Dados e seus Algoritmos, 3a. edição, Editora LTC, 2010.
  - W. Celes, R. Cerqueira, J.L. Rangel. Introdução a Estruturas de Dados, 1a. edição, Editora Campus, 2004.
  - N. Ziviani. Projeto de Algoritmos com Implementações em Pascal e C, 3a. edição, Editora Cengage Learning, 2011.
  - T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos - Teoria e Prática, 3a. edição, Editora Campus, 2012.
  - R. Sedgewick, K. Wayne. Algorithms, 4a. edição, Addison -Wesley, 2011.
  - A. Kelley, I. Pohl. A Book on C, 4a. edição, Addison Wesley, 1998.
- 🌐 Recursos Online
  - <Direto ao Ponto 44> As funções e os procedimentos - https://www.dio.me/articles/direto-ao-ponto-44-as-funcoes-e-os-procedimentos
  - Funções na Programação: O Que São e Como Criá-las de Forma Eficiente - https://www.dio.me/articles/funcoes-na-programacao-o-que-sao-e-como-cria-las-de-forma-eficiente-f3ed466dbbab
  - Funções - Programação 1 - Engenharia - https://wiki.sj.ifsc.edu.br/index.php/Fun%C3%A7%C3%B5es_-_Programa%C3%A7%C3%A3o_1_-_Engenharia

    - Aula - Procedimentos e Funções - https://gabrielbueno072.github.io/rea-aed/aula_func.html
    - W3Schools - JavaScript Functions
    - MDN Web Docs - Functions in JavaScript
    - GeeksforGeeks - Functions in C
- 📽️ Vídeos e Cursos
    - Curso em Vídeo - Funções em Programação
    - Programação de Computadores - Funções e Procedimentos - UFOP
    - Khan Academy - Introdução à Programação: Funções

Anexo - Resumo Estruturado/Mapa Mental Gerado por IA

1. Conceito de Função

- Bloco de código nomeado para executar tarefa específica
- Reutilização e modularização do código
- Entrada: parâmetros → Saída: valor de retorno

---

2. Vantagens do Uso

- **Reutilização de código** (princípio DRY)
- **Organização** (blocos menores)
- **Facilidade de manutenção**
- **Abstração** (foco no "o que", não no "como")
- **Colaboração** (divisão de tarefas)
- **Testabilidade** (testes isolados)

---

3. Definição, Declaração e Chamada

- **Definição**: nome, tipo de retorno, parâmetros e corpo
- **Declaração (protótipo)**: assinatura sem corpo (C/C++)
- **Chamada**: invocação com argumentos

---

4. Sintaxe Básica

- Tipo de retorno
- Nome da função
- Parâmetros
- Corpo (instruções)
- `return` (quando aplicável)

---

5. Parâmetros e Argumentos

- **Passagem por valor** → cópia, não altera original
- **Passagem por referência** → altera original

---

6. Valor de Retorno

- Retorna resultado ao chamador
- Finaliza execução da função

---

7. Funções vs Procedimentos

- **Função** → retorna valor
- **Procedimento** → não retorna valor (`void`)

---

8. Protótipos

- Declaração prévia (ex.: em C/C++)
- Permite uso antes da definição

---

9. Escopo de Variáveis

- **Local** (dentro da função)
- **Global** (acessível em todo programa)
- **Parâmetros** (variáveis locais especiais)

---

10. Recursividade

- Função que chama a si mesma
- **Necessário caso base** para evitar loop infinito
- Usada em problemas divisíveis em subproblemas

---

11. Funções Anônimas e Arrow Functions

- Definidas sem nome (lambdas, arrow functions)
- Uso em callbacks, funções rápidas e concisas

---

12. Funções de Biblioteca vs. Definidas pelo Usuário

- **Biblioteca**: já fornecidas (ex.: `printf`, `Math.sqrt`)
- **Usuário**: criadas conforme necessidade

---

13. Boas Práticas

- Nomes descritivos
- Funções pequenas e coesas
- Documentação
- Evitar efeitos colaterais
- Poucos parâmetros

- Consistência nos retornos
- Testes unitários

---

14. Aplicações

- Manipulação de dados
- Interface com usuário
- Algoritmos (busca, ordenação, etc.)
- Acesso a arquivos, banco de dados, rede
- Modularização e frameworks