Algoritmo e Estrutura de Dados I

Módulo 1 - Revisão de Fundamentos de Programação

Profª. Elisa de Cássia Silva Rodrigues

Prof. Pedro Henrique Del Bianco Hokama

Profª. Vanessa Cristina Oliveira de Souza

Sumário

- Introdução.
- Algoritmo.
- Programação estruturada.
- Estruturas sequenciais.
- Estruturas condicionais.
- Estruturas de repetição.
- Funções.
- Ponteiros.

Introdução

- Etapas para desenvolvimento de um programa:
  - **Análise:**
    - ★ Estudar o enunciado do problema.
    - ★ Definir dados de entrada.
    - ★ Definir o processamento.
    - ★ Definir os dados de saída.

  - **Algoritmo:**
    - ★ Sequência de instruções que realizam uma tarefa específica.
    - ★ Instruções simples, objetivas e não ambíguas.
    - ★ Note que um problema pode ser resolvido por vários algoritmos.
  - **Codificação:**
    - ★ Transformar um algoritmo em códigos de uma linguagem de programação.

Algoritmo

- O que é necessário para construir um algoritmo?
  - Compreender o problema.
  - Destacar pontos importantes e objetos que o compõem.
  - Definir os dados de entrada, processamento e saída.
  - Escolher um tipo de algoritmo.
  - Construir o algoritmo.
  - Testar o algoritmo realizando simulações.
- Tipos de algoritmos
  - Descrição narrativa: utiliza linguagem natural.
  - Fluxograma: utiliza símbolos gráficos predefinidos.
  - Pseudocódigo: utiliza regras predefinidas.

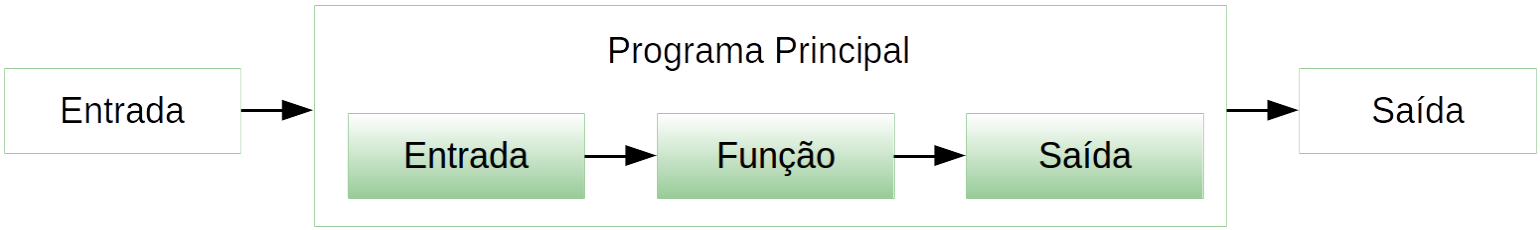
Vídeo - Algoritmos e Fluxogramas: https://www.youtube.com/watch?v=Fhp2rYQpNAc

Algoritmo

- **Problema:** Somar dois números.
  - Dados de entrada: primeiro número (n1) e segundo número (n2)
  - Processamento: somar os números ($s = n1 + n2$).
  - Dados de saída: resultado da soma (s).
- **Exemplo:**
  - Variável $n1 = 5$
  - Variável $n2 = 4$
  - Variável $s = n1 + n2 = 9$

Programação Estruturada

- Um problema pode ser dividido em problemas menores mais fáceis de resolver (funções).



- Todo processamento pode ser realizado pelo uso de três estruturas:
  - Sequencial.
  - Condicional.
  - Repetição.

Estruturas Sequenciais (Linguagem C)

- Declaração de variáveis:

Algoritmo

```
DECLARE x NUMÉRICO
```

Programa em C

```
int x;
float x;
double x;
```

- Comando de atribuição:

Algoritmo

```
x ← 1
```

Programa em C

```
x = 1;
```

Estruturas Sequenciais (Linguagem C)

- Leitura de dados (para variáveis inteiras):

Algoritmo

```
LEIA x
LEIA x, y, z
```

Programa em C

```
scanf ("%d", &x);
scanf ("%d %d %d", &x, &y, &z);
```

- Escrita de dados (para variáveis inteiras):

Algoritmo

```
ESCREVA "Mensagem"
ESCREVA x
ESCREVA "Valor = ", x
```

Programa em C

```
printf("Mensagem");
printf("%d", x);
printf("Valor = %d", x);
```

Estruturas Condicionais (Linguagem C)

SE

```
if (condição)
{
    comando;
    ⋮
    comando;
}
```

SE ... SENÃO

```
if (condição)
{
    comandos;
}
else
{
    comandos;
}
```

Estruturas Condicionais (Linguagem C)

SE ... SENÃO SE

```
if (variável == valor1)
{
    comandos;
}
else if (variável == valor2)
{
    comandos;
else
{
    comandos;
}
```

ESCOLHA ... CASO

```
switch (variável)
{
    case valor1:
        comandos;
        break;

    case valor2:
        comandos;
        break;

    default:
        comandos;
}
```

Estruturas de Repetição (Linguagem C)

ENQUANTO

```
while (condição)
{
    comando;
    ⋮
    comando;
}
```

FAÇA ... ENQUANTO

```
do
{
    comando;
    ⋮
    comando;
} while (condição);
```

PARA

```
for (inicialização; condição; passo)
{
    comando;
    ⋮
    comando;
}
```

Exemplos (Linguagem C)

Programa para imprimir os números pares inteiros de 1 a n.

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int x, n;
    scanf ("%d", &n);
    x = 1;
    while(x <= n) {
        if((x % 2) == 0) {
            printf ("%d ", x);
        }
        x = x + 1;
    }
    return 0;
}
```

Exemplos (Linguagem C)

Programa para imprimir os números pares inteiros de 1 a n.

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int x, n;
    scanf ("%d", &n);
    for(x = 1; x <= n; x++)
    {
        if((x % 2) == 0)
        {
            printf ("%d ", x);
        }
    }
    return 0;
}
```

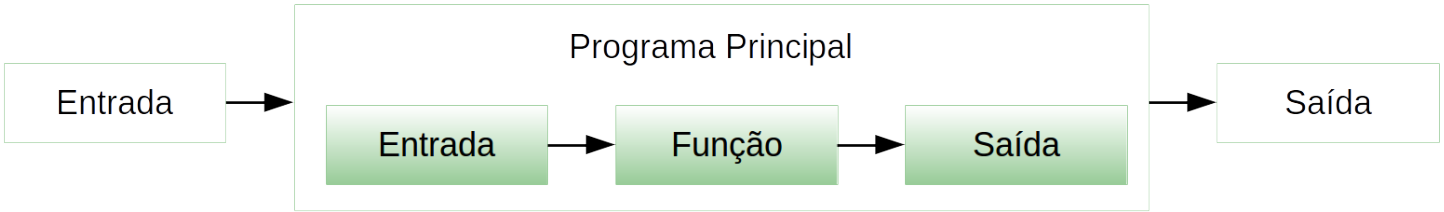
Exemplos (Linguagem C)

Programa para imprimir os números pares inteiros de 1 a n.

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int x, n;
    scanf ("%d", &n);
    for(x = 2; x <= n; x+=2)
    {
        printf ("%d ", x);
    }
    return 0;
}
```

Funções

- Blocos de instruções que realizam tarefas específicas podendo ser executados quantas vezes for necessário.
- Um problema pode ser subdividido em pequenas tarefas.



- Os programas tendem a ficar menores e mais organizados.
- O uso de funções permite a realização de desvios na execução do código (quando a função é chamada).

Funções

- Uma função pode retornar valores ou não. Exemplo:
  - `void soma(); // sem retorno`
  - `int soma(); // retorna valor inteiro`
- Uma função pode receber parâmetros ou não. Exemplo:
  - `void soma(); // sem parâmetros`
  - `void soma(int x, int y); // parâmetros do tipo inteiro`

Exemplo de Função

Programa para imprimir a soma de dois números inteiros.

```c
#include <stdio.h>
#include <stdlib.h>
int main() { // programa principal
    int x, y;
    scanf ("%d %d", &x, &y);
    soma(x,y); // chamada da função soma
    return 0;
}
void soma(int x, int y) { // função soma
    int s; // variável local
    s = x + y;
    printf ("%d", s);
}
```

Funções

- Variáveis globais:
  - São declaradas fora do escopo das funções.
  - O valor de uma variável global pode ser utilizado por qualquer função do programa.
  - ***Desvantagem:*** *torna difícil a manuntenção do programa.*
- Variáveis locais:
  - São declaradas dentro do escopo de uma determinada função.
  - Para outra função utilizar o valor de uma variável local, é necessário passar esse valor por parâmetro para a função em questão.
  - Se a função for apenas utilizar o valor recebido, sem alterá-lo, temos:
    - ★ Passagem de parâmetros por valor.
  - Se a função for alterar o valor recebido, temos:
    - ★ Passagem de parâmetros por referência.

Funções

- Exemplo de passagem de parâmetros por valor:
  - As variáveis `x` e `y` não podem ser alteradas dentro da função `soma`.
  - **Chamada da função:** `s = soma(x, y);`
  - **Protótipo da função:** `int soma(int a, int b);`
- Exemplo de passagem de parâmetros por referência:
  - A variável `s` pode ser alterada dentro da função `soma`.
  - **Chamada da função:** `soma(&s, x, y);`
  - **Protótipo da função:** `void soma(int *s, int a, int b);`

Vídeo - C++: https://www.youtube.com/watch?v=ErMKBh1pobg&index=7&list=PLrKBFf87Cy9CNZpzi3poq8BFWc0h4f0vL

Ponteiros

- Armazenamento e manipulação de valores de endereços de memória são permitidos pela linguagem C.
- Para cada tipo de dado existente, há um *tipo ponteiro* que armazena um endereço de memória que contém aquele tipo de dado.
  - **Exemplo:**

```
int a; // armazena um dado do tipo inteiro (2 bytes)
int *p; // armazena o endereço de memória que contém um inteiro
```

  - Para atribuir valores ao ponteiro p, temos duas formas:
    - ★ Operador unário & ("endereço de"): p = &a;
    - ★ Operador unário * ("conteúdo de"): *p = 10;

Vídeo - Pointer Fun C: https://www.youtube.com/watch?v=mnXkiAKbUPg
Vídeo - Pointers in C++: https://www.youtube.com/watch?v=Eyt46xFUB-8

Exemplos com Ponteiros

- Passagem de parâmetros por referência:
  https://repl.it/community/classrooms/205600/assignments/5694402
- Vetores e matrizes:
  https://repl.it/community/classrooms/205600/assignments/5694398

Referências Bibliográficas

1. ASCÊNCIO, A. F. G.; CAMPOS, E. A. V. ***Fundamentos da Programação de Computadores***. 2012.
2. BACKES, A. ***Estrutura de dados descomplicada em linguagem C***. 2016.