



# LINGUAGENS DE PROGRAMAÇÃO

MÓDULO 03

2023

**DESENVOLVIMENTO DE COMPETÊNCIAS EM MENINAS E  
JOVENS MULHERES PARA ELABORAÇÃO DE PROJETO DE  
FOGUETES EDUCACIONAIS**

**MANUAL: LINGUAGENS DE  
PROGRAMAÇÃO**

Coordenadora: Profa. Mariana Almeida – UFRN

MINISTRANTE: Profa. Alessandra Mendes Pacheco – EAJ/UFRN

## SUMÁRIO

INTRODUÇÃO .....	3
PENSAMENTO COMPUTACIONAL .....	4
ALGORITMOS.....	6
ESTRUTURA GERAL DE UM PROGRAMA DE COMPUTADOR.....	11
A LINGUAGEM DE PROGRAMAÇÃO C++ .....	12
ESCOPO DE UM ALGORITMO .....	15
COMANDO DE SAÍDA: INTERAÇÃO COM O USUÁRIO .....	16
A MEMÓRIA DO COMPUTADOR .....	16
VARIÁVEIS.....	17
TIPOS PRIMITIVOS DE DADOS .....	19
COMANDO DE ENTRADA: INTERAÇÃO COM O USUÁRIO .....	21
OPERADOR DE ATRIBUIÇÃO .....	22
OPERADORES E EXPRESSÕES ARITMÉTICAS.....	22
ATIVIDADES PRÁTICAS.....	24
ESTRUTURAS DE CONTROLE DE FLUXO DE EXECUÇÃO: SELEÇÃO .....	24
ATIVIDADES PRÁTICAS.....	29
ESTRUTURAS DE CONTROLE DE FLUXO DE EXECUÇÃO: REPETIÇÃO.....	29
ATIVIDADE PRÁTICA .....	33
DISCUSSÃO TEÓRICA – O PROBLEMA DO LANÇAMENTO OBLÍQUO.....	34
ATIVIDADE PRÁTICA .....	34
DISCUSSÃO TEÓRICA – SIMULANDO O LANÇAMENTO DE FOGUETES .....	34
ATIVIDADE PRÁTICA .....	35
APRESENTAÇÃO FINAL .....	35

## INTRODUÇÃO

Este manual tem o objetivo de compartilhar conhecimentos sobre o desenvolvimento de programas de computador utilizando linguagens de programação. A partir de conhecimentos sobre pensamento computacional, algoritmos, linguagem de programação, raciocínio lógico e códigos computacionais, busca também prover as alunas de técnicas para tomadas objetivas de decisões e construções de soluções em blocos e sequencialmente orientadas que possam ser executadas pelo computador.

O manual volta-se ainda para a promoção de uma educação melhor para todos, disseminando as linguagens computacionais como ferramentas formativas de profissionais do setor aeroespacial, utilizando o tema para despertar o interesse de estudantes pela Ciência, Tecnologia, Engenharia, Artes e Matemática – STEAM.

O seu objetivo principal é proporcionar um estudo sobre algoritmos e programação de computadores, possibilitando o desenvolvimento do raciocínio lógico aplicado na solução de problemas em nível computacional.

São objetivos específicos:

- Introduzir os conceitos básicos e definições de pensamento computacional, algoritmos e programação de computadores;
- Compreender os conceitos de linguagens de programação de computadores, compiladores e códigos-fonte;
- Compreender os conceitos básicos do desenvolvimento de algoritmos na linguagem de programação C++;
  - Entender o escopo e fluxo de execução de um programa de computador;
  - Conhecer as instruções primitivas de um algoritmo;
  - Conhecer os tipos de dados, operadores e expressões mais

utilizados;

- Desenvolver o raciocínio lógico e abstrato de programação;
- Implementar os algoritmos na linguagem de programação C++;
- Visualizar soluções computacionais para problemas;
- Praticar o processo de desenvolvimento de algoritmos;
- Explorar o desenvolvimento de práticas computacionais na área do setor aeroespacial;
- Colaborar para o desenvolvimento do pensamento computacional na área espacial.

Espera-se que as alunas, a partir do conhecimento adquirido, estejam aptas ao desenvolvimento de programas de computadores visando a resolução dos mais diversos problemas.

## PENSAMENTO COMPUTACIONAL

O pensamento computacional pode ser definido como uma habilidade para resolver problemas e desafios de forma eficiente, assim como um computador o faria. Essa resolução pode ou não envolver equipamentos tecnológicos, mas a sua base é a exploração de forma criativa, crítica e estratégica dos domínios computacionais.

Usar o pensamento computacional é ver um desafio ou problema, refletir sobre ele, separá-lo em partes, resolver cada uma dessas partes da maneira mais lógica e assertiva, para daí sim chegar a um resultado final. Dessa forma, podemos dividir o pensamento computacional em 4 pilares:

**Decomposição:** habilidade de dividir um problema complexo em partes menores, facilitando a solução e permitindo ainda inferir maior

atenção a cada etapa;

**Abstração:** propõe o foco em processos relevantes em vez de priorizar os detalhes, de modo que a solução possa ser válida para outros problemas;

**Reconhecimento de padrão:** identificação de aspectos comuns nos processos, reconhecendo padrões e similaridade e permitindo assim a construção de soluções para problemas comuns;

**Algoritmo:** criação de passos e soluções para alcançar um objetivo específico para qualquer problema, seja de ordem matemática ou não.

### Exemplo de problema

Qual a soma de todos os números entre 1 e 200?  
 $1+2+3+4+5+6+\dots+200=?$

Uma solução tradicional seria:

$$1+2 = 3 \quad 3+4 = 7 \quad 5+6 = 11 \quad 7+8 = 15 \dots$$

Ao final calcula-se a soma os resultados parciais ( $3+7+11+15\dots$ ).

Utilizando o pensamento computacional, teríamos a resolução do problema em cada um dos 4 pilares:

- Decomposição: a soma dos extremos é sempre igual.  
 $1+200 = 201 \quad 2+199 = 201 \quad 3+198 = 201\dots \quad 101+100 = 201$
- Abstração: soma de cada extremo = 201.
- Reconhecimento de padrões: se são 200 números, são 100 somas.
- Algoritmo:

Passo 1: Soma de cada par = 201

Passo 2: Total de pares = 100

Passo 3: Multiplique  $201 \times 100 = 20.100$  (resultado)

## ALGORITMOS

Atualmente a tecnologia e os algoritmos estão presente no dia-a-dia. Originado na matemática, o termo caracteriza um conjunto de etapas que um programa de computador (*software*) precisa realizar para chegar a um resultado. Os programadores que desenvolvem os algoritmos os utilizam como uma forma de dividir problemas maiores em passos menores que podem ser aplicados por computadores na realização de alguma tarefa específica.

### Definições

A palavra “*algoritmo*” surgiu na Idade Média e vem do nome do persa *Muhammad ibn Musa al Khwarizmi*, que foi astrônomo na Casa de Sabedoria do Califado Abássida, em Bagdá. Graças a sua vasta obra, o sistema de numeração indo arábico, que usamos até hoje, se difundiu no Oriente Médio e no Ocidente. O nome “*al Khwarizmi*”, devidamente latinizado, primeiro foi associado ao sistema de numeração e depois ao conceito moderno de algoritmo. A ideia principal contida na palavra refere-se à descrição sistemática da maneira de se realizar alguma tarefa.

Para a Ciência da Computação, o conceito de algoritmo foi formalizado em 1936 por Alan Turing (Máquina de Turing) e Alonzo Church, como sendo “um conjunto não ambíguo e ordenado de passos executáveis que definem um processo finito”.

De forma mais simples, um algoritmo é uma sequência de raciocínios, instruções ou operações que visam alcançar um objetivo específico. Em outras palavras, consiste em uma sequência finita de comandos que, quando executados, realizam tarefas específicas para as quais foram sistematicamente desenvolvidos como meio para se atingir determinado fim.

Pode-se dizer então que as receitas culinárias, os manuais de

instrução e as funções matemáticas são algoritmos. As receitas culinárias, por exemplo, possuem os ingredientes necessários (dados de entrada), as sequências de passos para realizar a receita (processamento ou instruções lógicas) e atingem os resultados (os pratos finalizados).

Um algoritmo, portanto, conta com os dados de entrada (*input*) e saída (*output*) mediados pelas instruções (processamento). Finalmente, todo problema a ser codificado em um algoritmo deve ser dividido em 3 fases:

**Entrada:** são os dados e informações iniciais do problema;

**Processamento:** são os procedimentos utilizados para solucionar o problema;

**Saída:** são os dados e informações resultantes do processamento.



Figura 01: fases de um algoritmo.

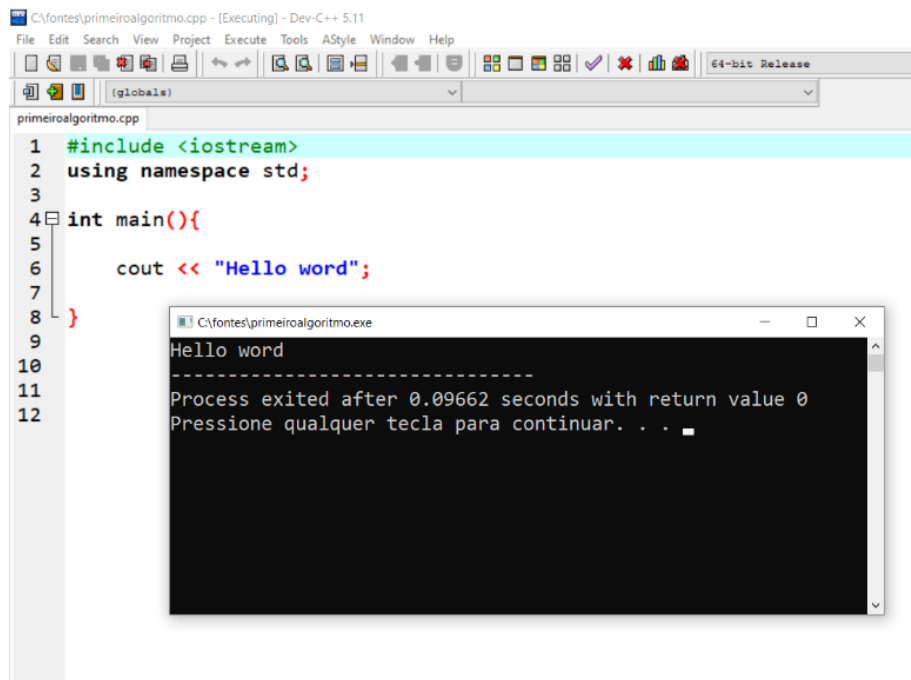
Fonte: de autoria própria

É de grande importância perceber que o resultado que o algoritmo busca prover é o que justifica a sua existência e que uma sequência simples de instruções pode se tornar mais complexa à medida que outros cenários passam a ser considerados.

Para que um programador seja capaz de elaborar programas utilizando uma linguagem de programação (como o C++, por exemplo), ter conhecimento sobre algoritmos é essencial. Saber usar algoritmos é saber criar estratégias para fracionar problemas reais em instruções mais objetivas que podem ser compreendidas e executadas por um computador para a solução de um problema. Em resumo, o algoritmo codificado em uma linguagem de programação resultará em um



programa de computador.



```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5
6     cout << "Hello word";
7
8 }
9
10
11
12
```

Output:

```
Hello word
-----
Process exited after 0.09662 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 02: exemplo de algoritmo codificado na linguagem de programação C++.

Fonte: de autoria própria

## Características

As principais características de um algoritmo são:

**Finitude:** faz-se necessário que o algoritmo seja finito, tendo em vista a finalidade pela qual foi desenvolvido;

**Objetividade:** uma vez que o algoritmo descreve ações sequências bem definidas a serem executadas, é de extrema importância que o mesmo seja elaborado de forma clara e livre de ambiguidades, ou seja, que não deixe margem para mais de uma interpretação;

**Possuir pelo menos um resultado:** sendo função do algoritmo executar determinada tarefa a fim de atingir um objetivo específico, faz-se é necessário que o algoritmo gere pelo menos um resultado ou saída.

## Formas de Representação

Considerando que um algoritmo possui passos e objetivos próprios, existem formas bem definidas de representação a partir das quais o mesmo deve ser concebido. São três as formas representações principais:

**Descrição narrativa:** consiste na descrição dos passos a serem seguidos em linguagem natural e de forma narrativa após a análise clara da problemática a ser solucionada.

Exemplo: Trocar uma lâmpada

- 1: Pegar uma escada
- 2: Posicionar a escada embaixo da lâmpada
- 3: Buscar uma lâmpada nova
- 4: Subir na escada com a lâmpada nova
- 5: Retirar a lâmpada velha
- 6: Colocar a lâmpada nova
- 7: Descer da escada

**Fluxograma:** consiste na representação gráfica elaborada em forma de diagrama que descreve cada ação do algoritmo que deverá ser executada para atingir o objetivo inicialmente proposto.

Exemplo: Cálculo da média de duas notas

**Entrada:** duas notas ( $N1$  e  $N2$ )

**Processamento:** calcular a média aritmética  $(N1+N2)/2$  e verificar se a média é maior ou igual a 7.

**Saída:** “Aprovado” (se a condição for verdadeira) ou “Reprovado” (se a condição for falsa).

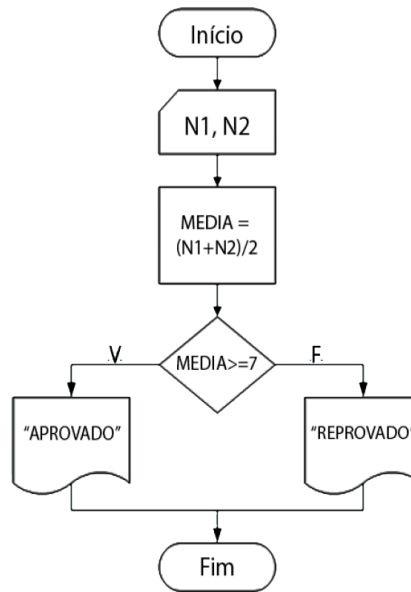


Figura 03: exemplo de fluxograma.

Fonte: de autoria própria

**Pseudocódigo:** consiste na escrita das ações que compõem o algoritmo em linguagem não computacional, ou seja, é o código que será previamente escrito para depois ser transcrito em uma linguagem de programação. Esta forma de representação é rica em detalhes e utiliza uma linguagem estruturada assemelhando-se a um programa escrito na linguagem de programação.

Exemplo: algoritmo Média

```

Algoritmo Media;
  Var N1, N2, MEDIA: real;
Início
  Leia (N1, N2);
  MEDIA ← (N1 + N2) / 2;
  Se MEDIA >= 7 então
    Escreva "Aprovado"
  Senão
    Escreva "Reprovado";
  Fim_se
Fim
  
```

## **ESTRUTURA GERAL DE UM PROGRAMA DE COMPUTADOR**

Como já foi visto, a um procedimento constituído de um conjunto de instruções bem definidas, executáveis e com o objetivo de resolver um problema, chama-se Algoritmo. Um algoritmo escrito em linguagem natural passa a ser chamado de programa de computador depois de convertido para uma linguagem aceita por um computador real. Um programa de computador é um conjunto de instruções que devem ser seguidas e executadas por um mecanismo, seja ele um computador ou um aparato eletromecânico.

Todos os trabalhos executados pelo computador são feitos a partir da utilização de programas. Um computador sem programas pode ser comparado a uma televisão sem transmissão. Porém, um computador só entende linguagem binária (ou linguagem de máquina) e, de uma forma simplificada, todos os demais caracteres (letras, números, símbolos) são convertidos em códigos construídos por conjuntos de zeros e uns.

Para diminuir a complexidade de se trabalhar diretamente com códigos binários, foram construídos códigos operacionais mais fáceis de serem manipulados pelo homem e deu-se ao próprio computador o trabalho de traduzi-los tanto para a linguagem binária, única que ele entende, quanto para os símbolos normais da linguagem humana, quando precisasse informar as respostas do processamento realizado.

A partir da criação das linguagens de montagem, que utilizam abreviações de palavras inglesas indicando a função que se deseja do computador, passar de abreviações à utilização de palavras humanas completas foi só uma questão de tempo. As linguagens de alto nível são as linguagens de programação que utilizam palavras completas e estruturas de expressão semelhantes ao linguajar humano, além de

outros recursos importantes, para informar ao computador quais funções ele deverá realizar. Por questões históricas relacionadas ao desenvolvimento dos computadores, estas linguagens geralmente utilizam termos em inglês.

Após escrito em uma linguagem de programação, o computador deverá converter os comandos dados em linguagem de alto nível para linguagem de máquina. Esta tarefa de conversão é feita por um programa especial de computador, chamado Compilador, que recebe as instruções em linguagem de alto nível e dá como saída outro programa constituído de instruções binárias. Ao programa original, em linguagem de alto nível, dá-se o nome de *código fonte* e ao resultado, em linguagem de máquina, de *executável* ou *aplicativo*.

## A LINGUAGEM DE PROGRAMAÇÃO C++

A linguagem de programação C++ é uma linguagem de programação compilada de uso geral. Desde os anos 1990 é uma das linguagens comerciais mais populares, sendo bastante usada também na academia por seu grande desempenho e base de utilizadores.

Um programa em C++ consiste em um ou mais arquivos. Um arquivo é uma porção de texto contendo código fonte em C++ e comandos do pré-processador. A extensão dos nomes dos arquivos fonte em C++, normalmente, é ".cpp".

Existem cinco espécies de símbolos em C++: identificadores, palavras-chave, literais, operadores e outros separadores. Espaços em branco, tabulações horizontais e verticais, novas linhas e comentários são ignorados, exceto pelo fato de servirem para separar símbolos. Porém, algum espaço em branco é necessário para separar identificadores, palavras-chave e constantes que de outro modo ficariam adjacentes.

Considerando sua estrutura, um programa em linguagem C++ é

uma coleção de variáveis, definições e chamadas de função, onde uma função é um conjunto de instruções com um nome e que desempenham uma ou mais ações. Quando o programa começa, ele executa o código de inicialização e chama uma função especial *main()*, onde é colocado o código primário para o programa. Um programa C++ mínimo consiste em:

```
main() {  
    // comandos que farão parte do algoritmo  
}
```

Este programa define a função *main*, que não possui argumentos e não faz nada. As chaves, { e }, são usadas para expressar agrupamentos em C++; no exemplo anterior, estas indicam o início e o fim do corpo da função (vazia) *main*. Cada programa em C++ deve ter uma função *main*.

Tipicamente, um programa produz alguma saída. O exemplo abaixo mostra um programa que escreve "Hello, World!" na tela do computador.

```
/* Programa Hello, World! */  
# include <iostream>  
int main() {    //função principal  
    cout << "Hello, World! \n";  
}
```

A linha *#include <iostream>* instrui o compilador a incluir as declarações das facilidades de um fluxo de entrada e saída padrão, encontradas em *<iostream>*. Sem esta declaração, a expressão *cout << "Hello, World! \n"* não faria sentido. O operador *<<* ("colocar em") escreve o seu segundo argumento no primeiro. Neste caso, a string *"Hello, World! \n"* é escrita no fluxo de saída padrão *cout*. Uma *string* é uma sequência de caracteres entre aspas duplas. As aspas servem para o compilador entender o texto delimitado, isto é, para que ele não

processe os caracteres circundados como se fossem instruções de programação ou como outros comandos. Em uma *string*, o caractere "\" seguido por outro caractere denota um único caractere especial; neste caso "\n" é o caractere de nova linha, então é escrito "Hello, World!" seguido de uma troca de linha. As duas barras "//" indicam um comentário no código fonte que não será processado pelo compilador. O valor inteiro retornado pela função main, se houver algum, é o valor de retorno do programa ao "sistema". Se nada é retornado, "o sistema" irá receber um valor randômico.

Após ser escrito em linguagem C++, o código fonte precisa ser compilado para que seja gerado o arquivo executável em linguagem de máquina. Se estiver livre de erros, o código fonte compilado (executável) poderá ser executado pelo computador como um aplicativo.

### **O AMBIENTE DE DESENVOLVIMENTO DEV C++**

A primeira coisa a fazer antes de desenvolver programas usando a linguagem C++ é instalar um compilador. Pode-se, adicionalmente, instalar um ambiente de desenvolvimento. O ambiente de programação Dev C++, que será utilizado neste curso e que está disponível para download em <http://www.bloodshed.net/devcpp.html>, é um ambiente gratuito e livre, o que significa que toda e qualquer pessoa pode fazer o download e instalá-lo em sua máquina livremente. O compilador utilizado pelo Dev C++ é o gcc, compilador padrão que já está integrado ao ambiente. Além disso, o Dev C++ é multiplataforma, o que significa que pode ser utilizado tanto em computadores com sistema operacional Windows quanto com sistema operacional Linux (basta fazer o download da versão correta (existe uma para cada sistema operacional)).

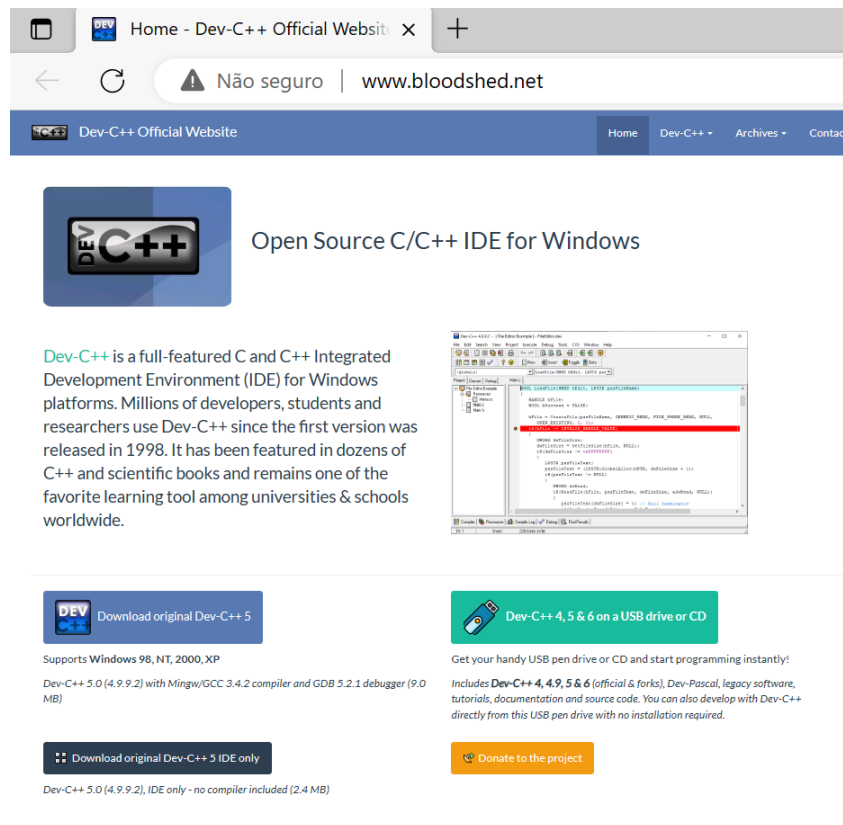


Figura 04: O ambiente de programação Dev C++.

Fonte: de autoria própria

Uma outra opção é utilizar um compilador disponível de forma online, como o que pode ser encontrado no site <https://replit.com/>.

## ESCOPO DE UM ALGORITMO

O escopo de um algoritmo é um contexto delimitante aos quais valores e expressões estão associados. As linguagens de programação têm diversos tipos de escopos. O tipo de escopo vai determinar quais tipos de entidades este pode conter e como estas são afetadas. Normalmente o escopo é utilizado para definir o grau de ocultação da informação, isto é, a visibilidade e acessibilidade às variáveis e aos dados em diferentes partes do algoritmo. Escopos podem conter declarações ou definições de identificadores, instruções e/ou expressões, que definem um algoritmo executável ou parte dele. A supracitada função *main* do C++, por exemplo, tem o seu escopo definido pelas chaves `{ }`.



## COMANDO DE SAÍDA: INTERAÇÃO COM O USUÁRIO

Os programas de computador precisam interagir com o usuário e os comandos de saída são utilizados para, nesta interação, fornecer informações a respeito do que está acontecendo ou requerer que alguma ação específica seja executada. Para que uma mensagem possa ser exibida na tela utiliza-se o comando *escreva* ou, em C++, *cout*.

*Em pseudocódigo*

```
escreva("Oi mundo!");
```

*Em C++*

```
cout << "Oi Mundo!";
```

```
// Programa Hello World!  
# include <iostream>  
using namespace std;  
int main() {  
    cout << "Oi mundo!";  
}
```

Vale observar que a mensagem a ser apresentada na tela é colocada entre aspas duplas. Além disso, os caracteres especiais “\n” e “\t” podem ser utilizados dentro das aspas para, respectivamente, pular uma linha em branco ou adicionar um espaço de tabulação.

## A MEMÓRIA DO COMPUTADOR

A memória de um computador (RAM – *Random Access Memory*) é o local em que são guardadas as informações necessárias para execução dos programas. A memória pode ser entendida como uma sequência de células nas quais se podem armazenar dados (conteúdo). Como as células estão ordenadas, atribui-se a cada uma delas um

número (endereço) correspondente a sua posição na sequência. Assim, cada célula de memória é caracterizada pelo seu endereço e seu conteúdo.

Esta memória é volátil (uma vez desligado o computador, as informações são perdidas). Os algoritmos utilizam esta memória para salvar informações durante a sua execução. O computador, por sua vez, manipula as informações contidas em sua memória através da leitura e da escrita. A leitura localiza a célula correspondente ao endereço desejado e consulta o valor armazenado (o valor não é alterado). Já a escrita localiza a célula correspondente ao endereço desejado e substitui o seu conteúdo pelo novo valor (o conteúdo anterior é perdido de forma irreversível).

Essencialmente, para executar um algoritmo o computador repete o seguinte ciclo: 1) Obtém um ou mais valores da memória, utilizando operações de leitura. 2) Realiza algum tipo de processamento sobre os dados consultados na memória e, como consequência, gera novos valores. 3) Efetua uma escrita para armazenar os novos valores (resultados) em células da memória. Assim, um algoritmo nada mais faz do que manipular valores que estão armazenados nas células da memória. O efeito final do programa é dado pelo conteúdo final das células de memória que ele manipula.

Em resumo, a memória é o conjunto de locais para armazenar os dados. Estes dados podem ser armazenados na memória por meio das variáveis.

## **VARIÁVEIS**

A variável é uma abstração da célula de memória. Neste caso, o programador, ao invés de utilizar os endereços físicos da memória para identificar as células que usa, tem à sua disposição rótulos simbólicos com nomes intuitivos escolhidos por ele mesmo. Esses rótulos são os

nomes que foram definidos para cada variável. Dessa forma, sempre que for necessário referir-se a algum dado armazenado na memória, o programador deverá utilizar o nome associado à variável que o contém. O compilador realiza automaticamente a transformação dos nomes das variáveis em seus respectivos endereços de memória.

Além disso, logo antes do programa iniciar sua execução, o computador pode escolher livremente quais células físicas da memória, desde que não estejam sendo utilizadas por nenhum outro programa, serão associadas a cada um dos nomes atribuídos às variáveis. Assim, logo antes de executar, o computador escolhe estas células e as associa aos nomes das variáveis do programa que vai entrar em execução e bloqueia o uso dessas células por qualquer outro programa que venha a ser iniciado enquanto o programa em questão estiver sendo executado. Quando esse programa terminar, as células associadas às variáveis são liberadas pelo computador e ficam livres para serem utilizadas por outros programas.

Portanto, a célula de memória, quando identificada por um rótulo (nomeada), será uma variável. Assim, ao invés de pensar na memória como uma sequência longa de células, pode-se compreendê-la como sendo um repositório que contém as variáveis associadas ao programa. Cada variável, por sua vez, é caracterizada pelo seu nome e conteúdo.

Em resumo, a variável é o local onde um dado específico é guardado. Através do nome da variável, o seu dado pode ser acessado (leitura) e modificado (escrita) pelo algoritmo. O dado consiste em um valor de um tipo específico que é armazenado em uma variável.

Toda variável possui NOME, TIPO e CONTEÚDO. As regras para nomes de variáveis mudam de uma linguagem para outra. Normalmente não podem começar por números, não podem ter letras que não pertençam ao alfabeto inglês, não podem ter espaços, não podem ser palavras reservadas da linguagem de programação e não podem possuir caracteres especiais (exceto “\_”).

As variáveis devem ser declaradas (criadas) antes de serem utilizadas e ao declarar uma variável, o computador reserva um espaço na memória para ela. Além disso, cada tipo de variável ocupa um tamanho diferente na memória.

*Exemplo de declaração de variável: Tipo nome\_da\_variável;*

São características de qualquer variável:

- O seu nome é único em todo o algoritmo. A variável “NOME”, por exemplo, é diferente da variável “nome”.
- O conteúdo da variável deve ser do mesmo tipo usado na declaração da variável.
- O uso de uma variável (leitura) em uma expressão representa o seu conteúdo naquele momento.
- Na atribuição, o conteúdo da variável é substituído por outro (escrita).

Uma constante é uma “variável especial” pois também terá reservado um espaço de memória para o seu dado. Porém, uma constante armazenará um valor ÚNICO, que NÃO mudará durante o algoritmo.

## TIPOS PRIMITIVOS DE DADOS

Um tipo de dado consiste em algo do mundo real que pode ser representado computacionalmente. Por exemplo, os números que pertencem ao conjunto dos números inteiros, os números que pertencem ao conjunto dos números reais, as letras, os caracteres especiais. As linguagens de programação implementam formas de representar e manipular esses dados, que podem ser classificados em dois grandes grupos: os tipos de dados primitivos e os tipos de dados não primitivos. Os tipos de dados primitivos são os tipos básicos que

devem ser implementados por todas as linguagens de programação, como os números reais, inteiros, booleanos, caracteres e strings. Os tipos de dados não primitivos costumam ser estruturas de dados mais complexas do que os tipos de dados primitivos e não serão abordados neste curso.

Para otimizar a utilização da memória, cada variável armazena apenas um tipo de dado que deve ser previamente especificado quando da declaração. A variável “nome”, por exemplo, deveria armazenar textos (caracteres). A variável “idade”, por sua vez, deveria armazenar apenas números inteiros (sem casa decimal). Já a variável “salário” deveria armazenar números reais (com casa decimal).

De modo geral são 4 os tipos de dados primitivos:

- Inteiro: Representa valores numéricos negativos ou positivos sem casa decimal, ou seja, valores inteiros.
- Real: Representa valores numéricos negativos ou positivos com casa decimal, ou seja, valores reais. Também são chamados de ponto flutuante.
- Lógico: Representa valores booleanos, assumindo apenas dois estados, *verdadeiro* ou *falso*. Pode ser representado apenas um bit (1 ou 0).
- Caracter: Representa uma sequência de um ou mais caracteres alfanuméricos (pode ser qualquer símbolo, letra ou número), desde de que esteja entre “ ” (aspas duplas).

Na linguagem C++ os tipos primitivos são: *int* (inteiro), *float* (real), *boolean* (lógico) e *char* (um único caracter) ou *string* (cadeia de caracteres). Algumas linguagens de programação subdividem esses tipos primitivos de acordo com o espaço necessário para os valores daquela variável.

*Exemplo de declaração de variável inteira em C++: int idade;*

## COMANDO DE ENTRADA: INTERAÇÃO COM O USUÁRIO

Para que o usuário interaja com o algoritmo, faz-se necessário que o programa leia o dado digitado pelo usuário e armazene-o em algum lugar para posterior processamento. Assim, da mesma forma que após a leitura uma informação é armazenada no cérebro, o computador armazenará o dado lido na memória utilizando uma variável. Para que um dado possa ser lido a partir da digitação do usuário, utiliza-se o comando *leia* já informando em qual variável aquele dado será armazenado. Em C++ o referido comando é o *cin*. Vale ressaltar que o dado deverá ser obrigatoriamente do mesmo tipo declarado para a variável que o armazenará.

*Em pseudocódigo*

```
leia(idade);
```

*Em C++*

```
cin >> idade;
```

```
// Programa que lê a idade de uma pessoa e a escreve
# include <iostream>
using namespace std;
int main(){
    int idade;
    cout << "Digite a idade de uma pessoa: ";
    cin >> idade;
    cout << "A idade digitada foi: " << idade;
}
```

Observação: após armazenado na variável pelo comando *leia* (*cin*), o dado poderá ser acessado e informado ao usuário através do comando de saída *escreva* (*cout*) conforme exemplo acima.

## OPERADOR DE ATRIBUIÇÃO

O operador de atribuição é o tipo mais simples de operador da programação de computadores. Basicamente, ele permite modificar os valores de uma variável. Representado na maioria das linguagens de programação pelo símbolo de igualdade =, o operador de atribuição tem a finalidade de colocar um valor dentro de uma variável.

A depender da posição esquerda/direita de uma variável, o operador = atribui para ela um papel ativo ou passivo. Considere para os dois exemplos abaixo que duas variáveis *int*, *primeirValor* e *segundoValor*, foram previamente declaradas.

Exemplo 1:

```
primeiroValor = 50;
```

Exemplo 2:

```
segundoValor = primeiroValor;
```

No exemplo 1, a variável *primeiroValor* está colocada na posição esquerda do operador de atribuição =, de tal forma que a variável possui um papel passivo. Ou seja, ela será modificada pelo que está no lado direito (o valor 50). Dessa forma, o valor 50 será o dado armazenado na variável *primeirValor*.

No exemplo 2, a variável *primeiroValor* está colocada na posição direita do operador de atribuição =, de tal forma que, agora, ela possui um papel ativo. Ou seja, ela modifica a variável *segundoValor*. Neste caso uma cópia do valor contido na variável “*primeiroValor*” (o valor 50, por exemplo) é atribuída à variável “*segundoValor*”.

Em síntese: o que está do lado direito do operador de atribuição (=) modifica o que está do seu lado esquerdo.

## OPERADORES E EXPRESSÕES ARITMÉTICAS

Os operadores aritméticos executam operações matemáticas, como adição e subtração, a partir dos seus operandos. Existem dois tipos de operadores aritméticos: unários e binários. Os operadores unários executam ações com um único operando. Operadores binários executam ações com dois operandos.

São operadores aritméticos unários: `++` (incremento) e `-` (decremento). Estes operadores podem ser anteriores ou posteriores ao operando.

São operadores aritméticos binários: `+` (adição), `-` (subtração), `*` (multiplicação), `/` (divisão), `**` (potência) e `%` (módulo). Estes operadores mantêm os operandos um à esquerda e outro à direita.

Exemplos de operações: `int x = 2 + 3; x++; x = x - 4;`

As expressões aritméticas normalmente são avaliadas da esquerda para a direita. A ordem em que as expressões são avaliadas é determinada pela precedência dos operadores utilizados. A precedência padrão de C++ é a seguinte:

- unário
- potência
- multiplicação, divisão e módulo
- adição e subtração

Se uma expressão contiver dois ou mais operadores com a mesma precedência, o operador à esquerda será avaliado primeiro. Por exemplo, `10 / 2 * 5` será avaliado como `(10 / 2)` e o resultado será multiplicado por 5.

Quando uma operação de precedência inferior precisar ser processada primeiro, deve ser colocada entre parênteses. Por exemplo, `30 / 2 + 8`. Normalmente, isso é avaliado como 30 dividido por 2 depois 8 somado ao resultado. Se desejar dividir por `2 + 8`, isto deveria ser escrito como `30 / (2 + 8)`. Os parênteses podem ser aninhados em



expressões. As expressões entre parênteses mais internas são avaliadas primeiro.

## ATIVIDADES PRÁTICAS

### *Reconhecendo algoritmos*

Sugerimos o acesso ao link <https://compute-it.toxicode.fr/> e a execução dos passos solicitados em cada nível para melhor compreensão dos conceitos (siga no máximo de níveis que conseguir!).

Sugerimos ainda assistir aos vídeos “Como ensinar linguagem de programação para uma criança”, disponível no link <https://www.youtube.com/watch?v=pdhqwbUWf4U>, e “Pensamento Computacional Desplugado”, disponível no link <https://www.youtube.com/watch?v=Bxg8QC93joo>.

### *Construindo programas de computador que apresentem telas iniciais no console e interajam com o usuário*

Elabore um algoritmo que solicite o nome do usuário e apresente uma tela de boas vindas individualizada.

## ESTRUTURAS DE CONTROLE DE FLUXO DE EXECUÇÃO: SELEÇÃO

As estruturas de controle de fluxo de execução de seleção, ou condicionais, possibilitam a escolha de um grupo de ações e estruturas a serem executadas quando determinadas condições forem ou não satisfeitas. Em outras palavras, as estruturas condicionais permitem que um programa execute diferentes comandos de acordo com as condições estabelecidas, visando possibilitar a verificação de uma condição e alteração do fluxo de execução do algoritmo. Assim, é possível definir uma ação específica para diferentes cenários e obter exatamente o resultado esperado durante a execução de um

programa.

Em geral, o funcionamento das estruturas condicionais depende apenas de um algoritmo simples que utiliza a instrução condicional “se” (ou “if”, em C++). Na instrução condicional é preciso declarar a condição que será analisada e os blocos de comando que o programa executará em cada cenário possível. E é justamente de acordo com o número de possibilidades que as estruturas condicionais são classificadas em simples e compostas.

### **Estruturas Condicionais Simples**

As condicionais simples são aquelas nas quais é preciso declarar apenas o que será executado caso a condição definida seja satisfeita (resultado do teste verdadeiro). Isso quer dizer que, se o retorno da condicional for falso, sua execução é apenas encerrada e o algoritmo prossegue para interpretar as linhas de código presentes após o final da estrutura.

#### **Em pseudocódigo**

```
se (condição 1)  
    bloco para condição 1 verdadeira;  
fim_se
```

#### **Em C++**

```
if (condição 1) {  
    bloco para condição 1 verdadeira;  
}
```

### **Estruturas Condicionais Compostas**

As condicionais compostas, por sua vez, permitem que sejam programados um comportamento para quando o resultado da condição analisada for verdadeiro e outro diferente para quando o

resultado for falso. Essa segunda possibilidade é representada pela instrução “senão” (ou “else”, em C++) que é declarado após o fechamento do primeiro caso.

#### *Em pseudocódigo*

```
se (condição 1)
    bloco para condição 1 verdadeira;
senão
    bloco para condição 1 falsa;
fim_se
```

#### *Em C++*

```
if (condição 1) {
    bloco para condição 1 verdadeira;
} else {
    bloco para condição 1 falsa;
}
```

Exemplo de algoritmo que lê a idade de uma pessoa, verifica e escreve se pode ou não dirigir utilizando uma condicional composta:

```
// Programa que lê a idade de uma pessoa, verifica
// e escreve se ela pode ou não dirigir
# include <iostream>
using namespace std;
int main(){
    int idade;
    cout << "Digite a idade de uma pessoa: ";
    cin >> idade;
    if (idade >= 18) {
        cout << "Pode dirigir";
    }
```

```
    } else {  
        cout << "Ainda não pode dirigir";  
    }  
}
```

### **Estruturas Condicionais Aninhadas**

Se a intenção for contar com mais de dois possíveis retornos, podem ser utilizadas estruturas condicionais encadeadas ou aninhadas. Dessa forma, podem ser testadas tantas condições quantas forem necessárias ao programa, ampliando bastante o poder dessa importante instrução. O encadeamento acontece apenas declarando uma nova condicional logo após a outra.

Exemplo de algoritmo que lê a idade de uma pessoa, verifica e escreve se ela pode votar e se ela pode ou não dirigir utilizando uma condicional composta aninhada:

```
// Programa que lê a idade de uma pessoa, verifica  
// e escreve se ela pode ou não dirigir  
# include <iostream>  
using namespace std;  
int main(){  
    int idade;  
    cout << "Digite a idade de uma pessoa: ";  
    cin >> idade;  
    if (idade >= 16) {  
        cout << "Pode votar";  
        if (idade >= 18) {  
            cout << "Pode dirigir";  
        }  
    } else {  
        cout << "Ainda não pode votar nem dirigir";  
    }  
}
```

## Estruturas de Seleção Múltipla

A estrutura de seleção múltipla é uma solução elegante quanto se tem várias estruturas condicionais aninhadas. A proposta da estrutura de seleção múltipla é permitir que o programa execute direto o bloco de código desejado, dependendo do valor de uma variável de verificação. Esta estrutura é representada pela instrução “escolha-caso” (ou “switch-case”, em C++).

### Em pseudocódigo

```
escolha (variável)

    caso <valor1> faça: "instruções a serem executadas";
    caso <valor2> faça: "instruções a serem executadas";
    caso <valor3> faça: "instruções a serem executadas";
    outro caso: "instruções a serem executadas";

fim_escolha;
```

### Em C++

```
switch (variável) {

    case <valor1> faça: "instruções a serem executadas"; break;
    case <valor2> faça: "instruções a serem executadas"; break;
    case <valor3> faça: "instruções a serem executadas"; break;
    default: "instruções a serem executadas";

}
```

O “outro caso” é uma parte opcional da instrução que, caso exista, será executada quando nenhum dos casos for executado.

## ATIVIDADES PRÁTICAS

### *Desenvolvendo algoritmos*

- Elabore um programa de computador que acesse e altere os dados armazenados na memória utilizando expressões aritméticas, lógicas e relacionais.
- Elabore uma calculadora com as operações de soma, subtração, multiplicação e divisão utilizando a estrutura condicional e outra solução utilizando a estrutura de seleção múltipla.

## ESTRUTURAS DE CONTROLE DE FLUXO DE EXECUÇÃO: REPETIÇÃO

Também chamadas de laços de repetição, as estruturas de repetição permitem executar mais de uma vez um mesmo bloco de instruções. Trata-se de uma forma de executar blocos de comandos somente sob determinadas condições, mas com a opção de repetir o mesmo bloco quantas vezes for necessário. Em outras palavras, uma estrutura de repetição permite que o mesmo conjunto de instruções seja executado mais de uma vez de acordo com uma condição ou com um contador.

As estruturas de repetição são úteis, por exemplo, para repetir uma série de operações semelhantes que são executadas para todos os elementos de uma lista ou de uma tabela de dados, ou simplesmente para repetir um mesmo processamento até que uma certa condição seja satisfeita.

Existem 3 estruturas de repetição básicas: repetição com teste no início, repetição com teste no fim e repetição contada.

### *Estrutura de repetição com teste no início*

A estrutura de repetição com teste no início, considerada a mais simples, repete um bloco de instruções internas enquanto uma condição permanecer verdadeira. Caso a condição seja falsa, as instruções internas não serão mais executadas e o programa seguirá para as instruções fora do bloco de repetição. A repetição é controlada por uma condição que verifica alguma variável. Porém, para que a repetição funcione corretamente, é importante que essa variável sofra alteração durante a repetição. Esta estrutura é representada pela instrução “enquanto” (ou “while”, em C++).

#### *Em pseudocódigo*

*enquanto (condição)*

*bloco interno de instruções que serão repetidas;*

*fim\_enquanto;*

#### *Em C++*

*while (condição) {*

*bloco interno de instruções que serão repetidas;*

*}*

Exemplo de algoritmo que lê um número inteiro, calcula e escreve seus 5 sucessores:

```
// Programa que lê um número, calcula seus 5 sucessores
// e os escreve
# include <iostream>
using namespace std;
int main(){
    int numero, cont = 1;
    cout << "Digite um número inteiro: ";
    cin >> numero;
    cout << "Sucessores: ";
```

```
while (cont <= 5) {  
    cout << (numero + cont) << " ";  
    cont++;  
}  
}
```

### **Estrutura de repetição com teste no fim**

A estrutura de repetição com teste no fim é muito semelhante à anterior, porém com uma diferença crucial: a condição é verificada pela primeira vez somente após executar o bloco interno de instruções uma vez. Em outras palavras, nessa estrutura os comandos são repetidos pelo menos uma vez, já que a condição se encontra no final. Além disso, como nesta estrutura o bloco de instruções internas vem antes da condição, caso a variável condicional for alterada dentro do bloco de comandos, isso afetará a validação da condição já no primeiro teste. Esta estrutura é representada pela instrução “repita” (ou “do\_while”, em C++).

#### **Em pseudocódigo**

*repita*

*bloco interno de instruções que serão repetidas;*

*até (condição);*

#### **Em C++**

*do{*

*bloco interno de instruções que serão repetidas;*

*} while (condição);*

Exemplo de algoritmo que lê um número inteiro, calcula e escreve seus 5 sucessores:



```
// Programa que lê um número, calcula seus 5 sucessores
// e os escreve
# include <iostream>
using namespace std;
int main(){
    int numero, cont = 1;
    cout << "Digite um número inteiro: ";
    cin >> numero;
    cout << "Sucessores: ";
    do{
        cout << (numero + cont) << " ";
        cont++;
    } while (cont <= 5);
}
```

### **Estrutura de repetição contada**

Essa estrutura de repetição é utilizada quando se sabe o número de vezes que um trecho do algoritmo deverá ser repetido. É passada uma situação inicial, uma condição e uma ação a ser executada a cada repetição. Após inicializar a variável, a estrutura a utiliza para controlar a quantidade de vezes que o conjunto de instruções será executado. Ao final do conjunto de instruções, a variável é alterada e testada para verificar se a próxima repetição será executada. Esta estrutura é representada pela instrução “para” (ou “for”, em C++).

#### **Em pseudocódigo**

*para (variável) de (valor inicial) até (valor final) passo  
(incremento/decremento) faça*

*bloco interno de instruções que serão repetidas;*

*fim\_para;*

#### **Em C++**

```
for(variável=valor inicial; condição; passo) {  
    bloco interno de instruções que serão repetidas;  
}
```

Exemplo de algoritmo que lê um número inteiro, calcula e escreve seus 5 sucessores:

```
// Programa que lê um número, calcula seus 5 sucessores  
// e os escreve  
# include <iostream>  
using namespace std;  
int main(){  
    int numero, cont;  
    cout << "Digite um número inteiro: ";  
    cin >> numero;  
    cout << "Sucessores: ";  
    for(cont=1; cont<=5; cont++){  
        cout << (numero + cont) << " ";  
    }  
}
```

## ATIVIDADE PRÁTICA

### Desenvolvendo algoritmos

- Elabore um programa de computador que apresente um menu de opções ao usuário e execute as operações referentes à opção escolhida por ele;
- Elabore um programa de computador que simule um jogo computacional.

## DISCUSSÃO TEÓRICA – O PROBLEMA DO LANÇAMENTO OBLÍQUO: SIMULANDO O LANÇAMENTO DE FOGUETES

Como se dá o lançamento oblíquo e quais são as variáveis envolvidas?

Quais são as variáveis e como são feitos os cálculos da trajetória de um foguete a partir do seu lançamento?

Qual seria a concepção da solução algorítmica para o cálculo do lançamento oblíquo?

Qual seria a concepção da solução algorítmica para o problema de simulação de lançamento de foguetes?

O lançamento oblíquo (vídeos):

<https://www.youtube.com/watch?v=BzfUqkKHb1Q>

[https://www.youtube.com/watch?v=Zp7-BvEL7\\_4](https://www.youtube.com/watch?v=Zp7-BvEL7_4)

Simulador de lançamento oblíquo:

[https://phet.colorado.edu/sims/html/projectile-motion/latest/projectile-motion\\_all.html?locale=pt\\_BR](https://phet.colorado.edu/sims/html/projectile-motion/latest/projectile-motion_all.html?locale=pt_BR).

## ATIVIDADE PRÁTICA

### *Desenvolvendo projeto*

- Elabore um programa de computador interativo para a leitura e cálculo de variáveis envolvidas no lançamento oblíquo a fim de simular o lançamento de um foguete.

## DISCUSSÃO TEÓRICA – OpenRocket

Como projetar um foguete utilizando a plataforma OpenRocket? Quais são as variáveis envolvidas? Como utilizar a plataforma? A partir do projeto do foguete, como simular o seu lançamento?

Download da plataforma: <https://openrocket.info/>

Tutorial de uso: <https://www.youtube.com/watch?v=CfT25FJbSuo>

## **ATIVIDADE PRÁTICA**

### ***Desenvolvendo projeto***

- Utilizando a plataforma OpenRocket, desenvolva o projeto de um foguete e realize a simulação do seu lançamento.

## **APRESENTAÇÃO FINAL**

Entrega das atividades e produtos desenvolvidos durante o ciclo de aprendizagem deste curso.

## **REFERÊNCIAS**

ALENCAR FILHO, E. Iniciação à lógica matemática. Ed. Nobel, 2002.

SOUZA, M. F.; GOMES, M. M.; SOARES, M. V.; CONCILIO, R. Algoritmos e lógica de programação. Ed. Cengage, 2019.

PINTO, W. S. Introdução ao desenvolvimento de algoritmos e estrutura de dados. Ed. Érica, 1991.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. Algoritmos: lógica para desenvolvimento de programação de computadores. Ed. Erica. 2018.

CORMEN, T., LEISERSON, C. E., RIVEST, R. L.; STEIN, C. Algoritmos - Teoria e Prática. Ed. LTC, 2012.

ASCENCIO, A. F. G.; CAMPOS, E. A. V. Fundamentos da programação de computadores. Ed. Pearson, 2012.

DROZDEK, A. Estrutura de dados e algoritmos em C++. Ed. Cengage Learning, 2ª Edição, 2016.

DEITEL, H; HARVEY, P. C++ Como Programar. Ed. Pearson, 5ª edição, 2006.