

Programação de Computadores

# INTRODUÇÃO

# Introdução

- Computadores são equipamentos eletrônicos



# Introdução

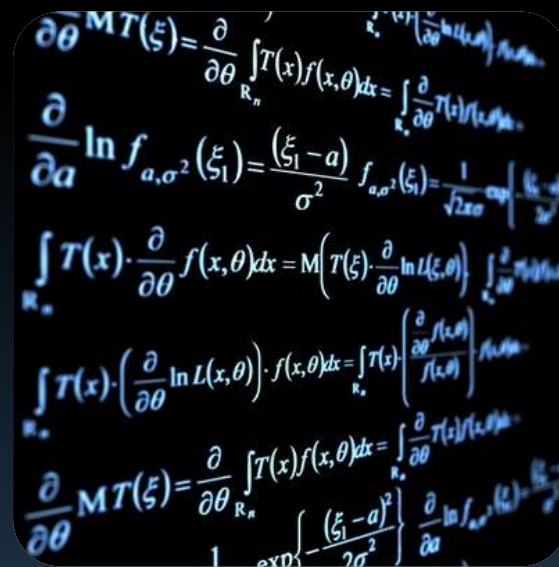
- Computadores são destinados ao processamento dos mais **variados tipos de informações**



# Introdução

- Computadores executam **diversas tarefas**:
  - Solução de problemas matemáticos

- Achar raízes de uma função
- Calcular derivadas e integrais


$$\begin{aligned}\frac{\partial}{\partial \theta} M T(\xi) &= \frac{\partial}{\partial \theta} \int_{\mathbb{R}_n} T(x) f(x, \theta) dx = \int_{\mathbb{R}_n} \frac{\partial}{\partial \theta} T(x) f(x, \theta) dx \\ \frac{\partial}{\partial a} \ln f_{a, \sigma^2}(\xi_1) &= \frac{(\xi_1 - a)}{\sigma^2} f_{a, \sigma^2}(\xi_1) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left\{-\frac{(\xi_1 - a)^2}{2\sigma^2}\right\} \\ \int_{\mathbb{R}_n} T(x) \cdot \frac{\partial}{\partial \theta} f(x, \theta) dx &= M\left(T(\xi) \cdot \frac{\partial}{\partial \theta} \ln L(\xi, \theta)\right) \\ \int_{\mathbb{R}_n} T(x) \cdot \left(\frac{\partial}{\partial \theta} \ln L(x, \theta)\right) \cdot f(x, \theta) dx &= \int_{\mathbb{R}_n} T(x) \cdot \left(\frac{\partial}{\partial \theta} \ln f(x, \theta)\right) \cdot f(x, \theta) dx \\ \frac{\partial}{\partial \theta} M T(\xi) &= \frac{\partial}{\partial \theta} \int_{\mathbb{R}_n} T(x) f(x, \theta) dx = \int_{\mathbb{R}_n} T(x) \cdot \frac{\partial}{\partial \theta} f(x, \theta) dx \\ &= \int_{\mathbb{R}_n} T(x) \cdot \frac{\partial}{\partial \theta} \left[ \frac{1}{\sqrt{2\pi\sigma}} \exp\left\{-\frac{(\xi_1 - a)^2}{2\sigma^2}\right\} \right] \cdot \frac{\partial}{\partial a} \ln f_{a, \sigma^2}(\xi_1) dx\end{aligned}$$

# Introdução

- Computadores executam **diversas tarefas**:
  - Controle de processos industriais



# Introdução

- Computadores executam **diversas tarefas**:
  - Execução remota de cirurgias





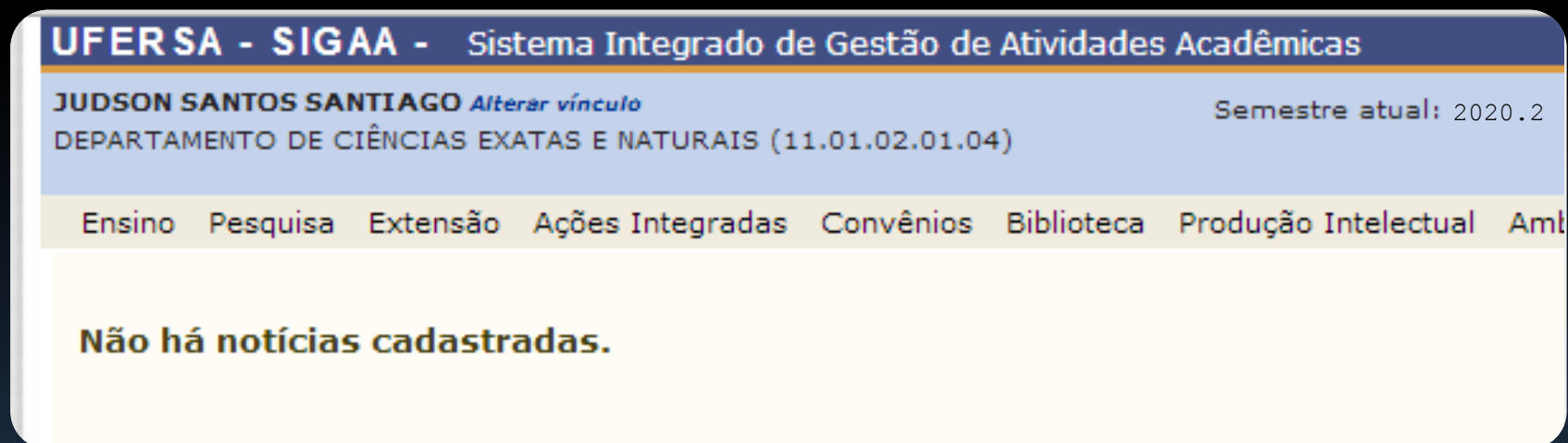
# Introdução

- Computadores executam **diversas tarefas**:
  - Previsão de condições climáticas



# Introdução

- Computadores executam **diversas tarefas**:
  - Controle acadêmico de uma universidade



The screenshot displays the UFERSA - SIGAA - Sistema Integrado de Gestão de Atividades Acadêmicas interface. At the top, the header reads "UFERSA - SIGAA - Sistema Integrado de Gestão de Atividades Acadêmicas". Below this, the user profile for "JUDSON SANTOS SANTIAGO" is shown, with a link to "Alterar vínculo" and the current semester "Semestre atual: 2020.2". The department is listed as "DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS (11.01.02.01.04)". A navigation bar includes links for "Ensino", "Pesquisa", "Extensão", "Ações Integradas", "Convênios", "Biblioteca", "Produção Intelectual", and "Ambiente". The main content area displays the message "Não há notícias cadastradas."



# Introdução

- Computadores executam **diversas tarefas**:
  - Jogos e simulações



# Introdução

- Computadores executam **diversas tarefas**:
  - Controle de robôs



# Introdução

- Como um mesmo dispositivo eletrônico executa **trabalhos de natureza tão diversas?**
  - Um telefone faz e recebe chamadas de voz
  - Um aparelho de DVD reproduz filmes
  - Um microondas cozinha alimentos
- Ao contrário da maioria dos dispositivos eletrônicos, **o computador pode ser programado** para executar uma tarefa qualquer

# Programação

- Como programar um computador?



Criando um programa

"Um **programa** é uma sequência de instruções que, ao serem executadas pelo computador, realizam uma determinada tarefa."

# Programação

- Um programa é como uma **receita de bolo**

## Ingredientes

- 1 xícara de açúcar
- 1 colher de fermento em pó
- ½ xícara de chocolate em pó
- 2 xícaras de farinha de trigo
- 1 xícara de leite
- 5 ovos

Dados

## Preparo

- Bata as claras em neve bem firme
- Junte as gemas e acrescente o açúcar
- Despeje o leite sem parar de bater
- Adicione farinha, chocolate e fermento
- Despeje em uma fôrma redonda untada
- Leve para assar em forno por 40 minutos

Instruções

# Programação

- Um **programa** em uma linguagem de programação
  - **Dados:** os valores 10 e 20
  - **Instruções:** soma, atribuição e exibição

```
#include <iostream>
using namespace std;

int main()
{
    int a = 10, b = 20;
    int c = a + b;
    cout << "A soma dos valores: " << c << endl;
    return 0;
}
```

Alterando os dados,  
ou as instruções,  
produzimos um  
resultado diferente.



# Programação

- Como criar um programa?
  - O computador não reconhece a **linguagem natural** utilizada na receita de bolo
- O computador só executa instruções em **linguagem de máquina**

```
01010111101011010101011101101101
01010101010111101110101011011010
11011101010101101010101010101010
10101010101010101010101101111011
00110111011101101110111011101110
11101111011110000001111000011111
11010101010101010100000001111111
11010101011111110101010001101110
```

Conjunto de Instruções de 32 bits

# Programação

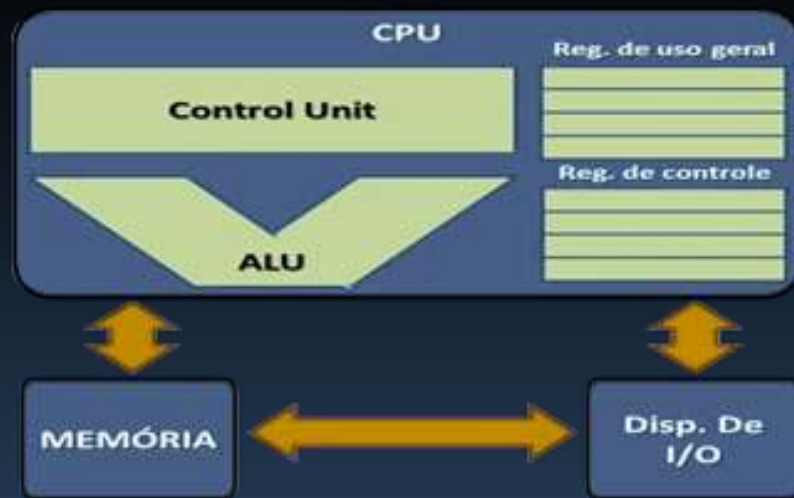
- Ninguém escreve programas em linguagem de máquina
  - A primeira linguagem utilizada para construir programas foi a **linguagem Assembly**

```
{calcula a soma 10 + 20}  
asm  
    mov eax,10  
    add eax,20  
end;
```

```
{A rotina a seguir imprime uma palavra}  
procedure imprime_nts; assembler;  
asm  
    push ax  
    push si  
    jmp @primeiro_char  
@imprime:  
    call imprime_char  
    inc si {aponta para o próximo caractere}  
@primeiro_char:  
    mov al,[si]  
    cmp al,0 {string acabou?}  
    jne @imprime {se não, imprime o caractere}  
    pop si  
    pop ax  
end;
```

# Programação

- **Assembly** é uma linguagem de **baixo nível**
  - Manipula as informações no nível de registradores da CPU e endereços de memória
  - Depende da **arquitetura da máquina**: x86 (CISC), PowerPC (RISC)



# Programação

- Linguagens de baixo nível **não são adequadas** para a programação de **grandes sistemas**
- Para isso utilizam-se **linguagens de alto nível**
  - Por exemplo: C, C++, Objective-C, C#, Java, Python
  - As instruções em linguagem de alto nível são **traduzidas**

```
cout << "Bem vindo a programação com C++";
```



```
01010111101011010101011101101101  
01010101010111101110101011011010
```



# Linguagens

- As primeiras linguagens de alto nível são hoje classificadas como **linguagens não-estruturadas**
  - Não há um mecanismo para agrupar instruções
  - Conjunto muito extenso de instruções
  - Muito difícil acompanhar a execução
  - Caracterizadas pelo uso da instrução GOTO
  - Exemplo: BASIC, FORTRAN

# Linguagens

- Exemplo de Programa em BASIC:

```
10 REM RESOLVE EQUACAO DO SEGUNDO GRAU
20 READ A,B,C
25 IF A=0 THEN GOTO 410
30 LET D=B*B-4*A*C
40 IF D<0 THEN GOTO 430
50 PRINT "SOLUCAO"
60 IF D=0 THEN GOTO 100
70 PRINT "PRIMEIRA SOLUCAO",(-B+SQR(D))/(2*A)
80 PRINT "SEGUNDA SOLUCAO",(-B-SQR(D))/(2*A)
90 GOTO 20
100 PRINT "SOLUCAO UNICA",(-B)/(2*A)
200 GOTO 20
410 PRINT "A DEVE SER DIFERENTE DE ZERO"
420 GOTO 20
430 PRINT "NAO HA SOLUCOES REAIS"
440 GOTO 20
490 DATA 10,20,1241,123,22,-1
500 END
```



# Linguagens

- A segunda geração de linguagens são conhecidas como **linguagens estruturadas**
  - Foi introduzido o conceito de sub-rotina
    - As instruções podem ser agrupadas
    - Uma sub-rotinas pode chamar outra
  - Muito mais fácil acompanhar a execução
  - Possibilita o reuso de código
  - Exemplo: Pascal, C

# Linguagens

- Exemplo de Programa em Pascal:

```
program Maximum;
var
    a, b, ret : integer;

function max(num1, num2: integer): integer;
var
    result: integer;
begin
    if (num1 > num2)
    then result := num1;
    else result := num2;
    max := result;
end;

begin
    a := 100; b := 200;
    ret := max(a, b);
    writeln('Max value is : ', ret);
end.
```

# Linguagens

- A terceira geração de linguagens são as **linguagens orientadas a objeto**
  - É a mais usada nos dias de hoje
  - Atrela as sub-rotinas a um conjunto de dados
  - Expande ainda mais as opções de reuso de código
  - Trata os problemas de forma mais intuitiva
  - Facilita o tratamento de erros
  - Exemplo: C++, C#, Java, Python

# Linguagens

- Exemplo de Programa em C++:

```
// controla ações da bolsa de valores
#include <iostream>

class Acoes
{
private:
    char empresa[40];
    int quantidade;
    double valor;

public:
    void adquirir(const char * companhia, int quant, double preco);
    void comprar(int quant, double preco);
    void vender(int quant, double preco);
    void atualizar(double preco);
    void mostrar();
};
```

# Linguagens

- O que é necessário saber para programar?
  - Conhecer a linguagem:
    - Representação de dados (int, float, char, etc.)
    - Entrada e saída de dados (printf, cout, writeln, etc.)
    - Processar dados (+, -, \*, >, <, <=, &&, ||, !, etc.)
    - Desvio e repetição (if, switch, for, while, etc.)
    - Declarar e chamar funções (bibliotecas ou criadas)
    - Manipulação de arquivos (texto e binário)
  - Usar um editor de código e compilador
  - Saber lógica de programação

# Linguagens

- Aprenderemos a programar usando a **linguagem C++**
- Existem diversos compiladores para C++:
  - **Microsoft Visual C++**
  - Clang
  - GNU g++
  - Intel C++
- Compilador é diferente de IDE
  - O **Visual Studio** utiliza o compilador **Visual C++**



# Por que C++?



C++

## Desempenho/\$

**Energia:** importante em todas as escalas – embarcado, mobile, desktop, datacenter

**Tamanho:** a quantidade de transistores é limitada pelo tamanho do dispositivo e pela tecnologia

**Experiências:** experiências interativas melhores em hardware menor – cada ciclo conta



## Desempenho / W



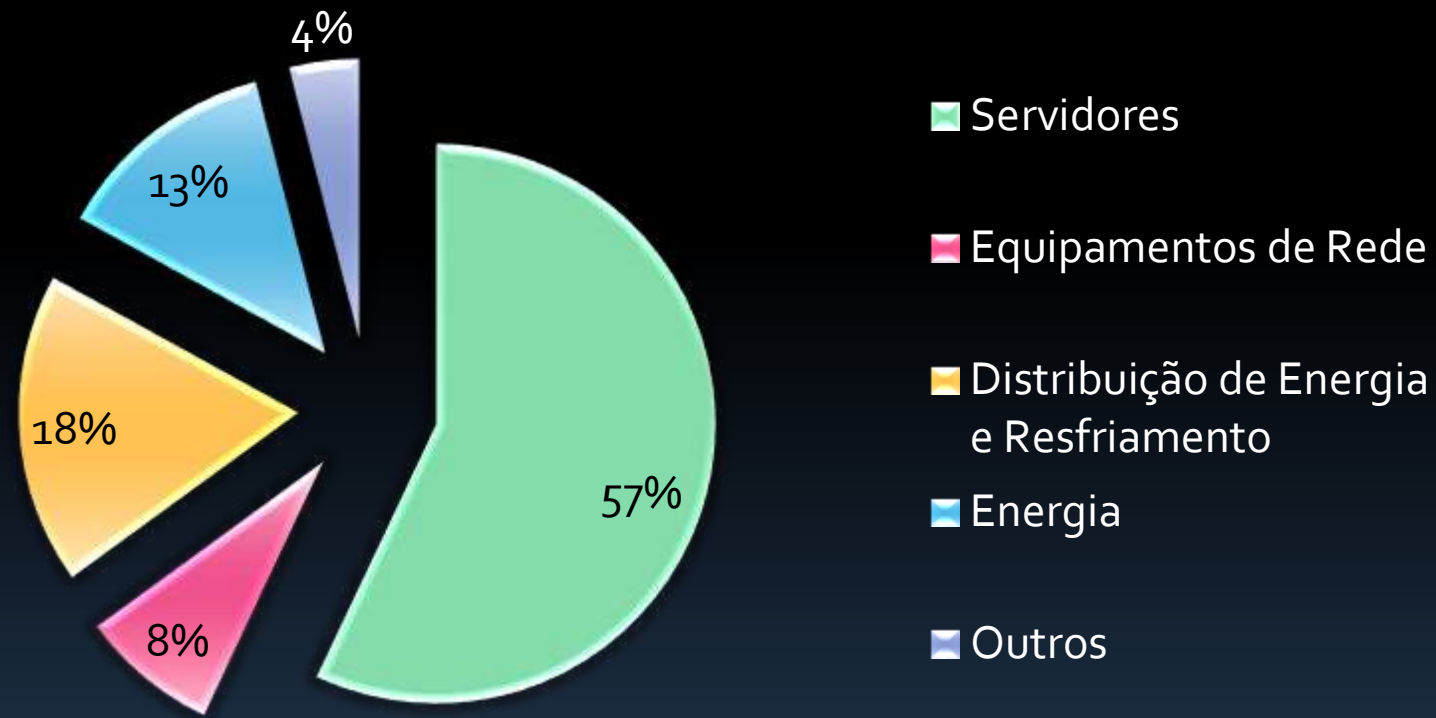
## Desempenho / T



## Desempenho / C

# Datacenter

**Custo Mensal**



88% do custo está relacionado ao desempenho dos programas

# Smartphones

Que linguagens são suportadas para o desenvolvimento em smartphones?



APIs	iPhone	Android	Windows Phone
Versão 1	-	Java	.NET
Versão 2+	Objective-C, C & C++	Java, C & C++	.NET, C++

# Eficiência de C++

## Loop Recognition in C++/Java/Go/Scala

Robert Hundt  
Google  
1600 Amphitheatre Parkway  
Mountain View, CA, 94043  
rhundt@google.com

Benchmark	Time [sec]	Factor
C++ Opt	23	1.0x
C++ Dbg	197	8.6x
Java 64-bit	134	5.8x
Java 32-bit	290	12.6x
Java 32-bit GC*	106	4.6x
Java 32-bit SPEC GC	89	3.7x
Scala	82	3.6x
Scala low-level*	67	2.9x
Scala low-level GC*	58	2.5x
Go 6g	161	7.0x
Go Pro*	126	5.5x

# O mundo é construído em C/C++

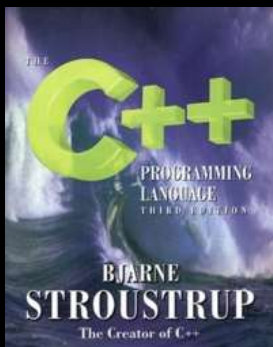
Windows HP-UX Windows UI KDE MS Office Corel Office  
MacOS Gnome MacOS UI OpenOffice Adobe Systems  
Blackberry ChromeOS Oracle MySQL  
Solaris Linux SAP DB IBM DB2 SQL Server  
iPad OS Symbian Google Internet Explorer  
Firefox Safari IBM Informix Apache  
iPhone iPod Opera Chrome Photoshop Apple iPod Software  
Visual C++ Visual Basic DirectX Games The GIMP Winamp Nero  
Visual C# gcc OGRE 3D OpenGL Windows Media Player  
PHP Perl TomTom Garmin Paypal Facebook Amazon

# Histórico



1979 - 1989

Pesquisa:  
C com classes,  
C++ ARM



1989 - 1999

Tendência:  
investimento  
pesado em  
compiladores  
e ferramentas  
(ISO C++98)



1999 - 2009

Linguagens  
focadas em  
produtividade

Java/C#

Pergunta:  
elas conseguem  
resolver tudo?



2009 - 2019

Código nativo  
está de volta  
com o  
retorno do  
Rei:

Desempenho

\$  
Watt  
Transistor  
Ciclo



# História da Linguagem C++

- No início de 1970 **Dennis Ritchie** do Bell Labs trabalhava em um projeto para desenvolver o Unix
- Ele precisava de uma linguagem que produzisse programas:
  - Compactos
  - Rápidos
  - Que controlasse o hardware eficientemente

# História da Linguagem C++

- Na época usava-se **Assembly** para atingir estes objetivos
  - Assembly é uma linguagem de **baixo nível**
  - Específica a uma família de processadores
- Ritchie decidiu construir uma **linguagem de alto nível**
  - Eficiente, portátil e com acesso direto ao hardware
  - Nasceu **a linguagem C**
- C foi a linguagem dominante nos **anos 80**

# História da Linguagem C++

- C++ foi desenvolvido no início dos anos 80 por **Bjarne Stroustrup**, também do Bell Labs

“C++ foi desenvolvido para que meus amigos e eu não tivéssemos que programar em Assembly, C ou outra linguagem moderna. Seu propósito era **escrever programas de forma mais fácil e agradável** para o programador.”

– Bjarne Stroustrup

# História da Linguagem C++

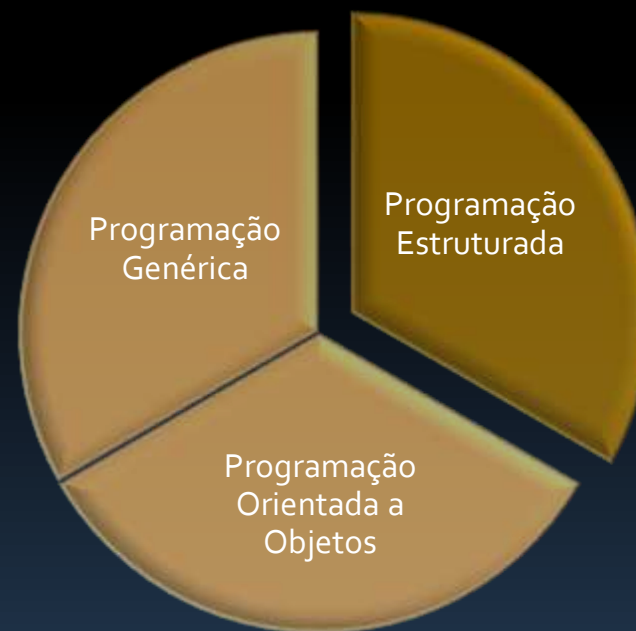
- Stroustrup estava mais preocupado em **fazer C++ ser útil** que seguir alguma filosofia em particular
- Ele decidiu criar **C++ baseado em C** devido:
  - Ser uma linguagem pequena
  - Adequação a programação de sistemas grandes
  - Grande disponibilidade
  - Ligações com o sistema operacional Unix

# História da Linguagem C++

- A linguagem **Simula67** inspirou Stroustrup a introduzir orientação a objetos em C++
- Stroustrup acrescentou também **programação genérica** sem modificar significativamente a base da linguagem C
- Hoje **C++ é um superconjunto de C:**
  - Qualquer programa C válido é um programa C++
  - Programas C++ podem usar as bibliotecas C

# História da Linguagem C++

- C++ é uma linguagem que une **três filosofias de programação**:
  - A programação estruturada (origem em C)
  - A programação orientada à objetos
  - A programação genérica
- Iremos nos restringir a **programação estruturada** de C++



# História da Linguagem C++

- A grande popularidade de C++ levou a linguagem a **várias plataformas e sistemas operacionais**
  - Portabilidade e padronização se tornaram um problema
- Em 1990 a ANSI e a ISO criaram um comitê conjunto para **padronizar a linguagem C++**
  - C++20 é o padrão atual, o C++23 está em desenvolvimento
- Mais informações: <http://www.stroustrup.com/>

# Resumo

- O **computador** é uma máquina que pode ser programada
  - Um **programa** é uma seqüência de instruções
- Um programa criado em linguagem de alto nível precisa ser **traduzido para linguagem de máquina** por um compilador
- Faremos programas na linguagem **C++** com o ambiente integrado **Microsoft Visual Studio** usando o **compilador Visual C++**