

>>> Programação Orientada a Objetos (POO)

... Motivação

Prof: André de Freitas Smaira

```
#include <iostream>
```

```
int main() {  
    std::cout << "Hello World!" << std::endl;  
    return 0;  
}
```

Quantas linhas de código?

```
#include <iostream>
```

```
int main() {  
    std::cout << "Hello World!"  
              << std::endl;  
    return 0;  
}
```

Quantas linhas de código?

>>> Windows

Ano	Versão	Linhas de Código	Linguagens
1985	Windows 1.0	~ 50k	C, Assembly
1987	Windows 2.0	~ 50k	C, Assembly
1990	Windows 3.0	~ 100k	C, Assembly
1995	Windows 95	~ 10M	C, C++, Assembly
2001	Windows XP	~ 45M	C, C++, Assembly
2006	Windows Vista	~ 50M	C, C++, Assembly
2009	Windows 7	~ 50M	C, C++, Assembly
2015	Windows 10	~ 80M	C, C++, Assembly
2021	Windows 11	~ 85M	C, C++, C#, Assembly

>>> Apple

Ano	Versão	Linhas	Linguagens
1984	System 1	$\sim 6k$	Assembly
1991	System 7	$\sim 200k$	Pascal, Assembly
1997	Mac OS 8	$\sim 3M$	C, C++, Pascal, Assembly
1999	Mac OS 9	$\sim 6M$	C, C++, Pascal, Assembly
2001	macOS X 10.0	$\sim 10M$	C, C++, Objective-C, Assembly
2005	macOS X 10.4 Tiger	$\sim 85M$	C, C++, Objective-C
2011	macOS X 10.7 Lion	$\sim 100M$	C, C++, Objective-C
2016	macOS 10.12 Sierra	$\sim 100M$	C, C++, Objective-C, Swift
2020	macOS 11 Big Sur	$\sim 100M$	C, C++, Objective-C, Swift
2021	macOS 12 Monterey	$\sim 100M$	C, C++, Objective-C, Swift
2022	macOS 13 Ventura	$\sim 100M$	C, C++, Objective-C, Swift

>>> Debian

Ano	Versão	Linhas	Linguagens
1993	Debian 0.91	~ 15M	C, C++, Shell, Perl
1996	Debian 1.1 (Buzz)	~ 30M	C, C++, Shell, Perl
1998	Debian 2.1 (Slink)	~ 59M	C, C++, Shell, Perl
2001	Debian 3.0 (Woody)	~ 104M	C, C++, Shell, Perl
2005	Debian 3.1 (Sarge)	~ 215M	C, C++, Shell, Perl
2007	Debian 4.0 (Etch)	~ 283M	C, C++, Shell, Perl
2009	Debian 5.0 (Lenny)	~ 380M	C, C++, Shell, Perl
2011	Debian 6.0 (Squeeze)	~ 475M	C, C++, Shell, Perl
2013	Debian 7.0 (Wheezy)	~ 570M	C, C++, Shell, Perl
2015	Debian 8.0 (Jessie)	~ 665M	C, C++, Shell, Perl
2017	Debian 9.0 (Stretch)	~ 760M	C, C++, Shell, Perl
2019	Debian 10.0 (Buster)	~ 855M	C, C++, Shell, Perl
2021	Debian 11.0 (Bullseye)	~ 950M	C, C++, Shell, Perl

```
>>> Custo
```

Debian 2.2

- * ~ 59M linhas de código
- * 4000 linhas / pessoa / ano
- * 1,9 bilhão de dólares

```
>>> Taxa de erros
```

Após depuração

- * **Comercial**: 5 para cada 1000 linhas
- * **Militar**: 1 para cada 1000 linhas


```
>>> Desenvolvimento de Software
```

Ciclo de Vida

- * **Especificação**: Como vai ser?
- * **Desenvolvimento**
- * **Manutenção**: o que tem de errado?

>>> Especificação

- * o **que** vai ser?
- * em conjunto com o **cliente**
- * resulta em um **contrato**

>>> Desenvolvimento

Baseado na **especificação**

- * Programação
- * Testes
- * Documentação

>>> Programação

- * Implementação
- * Deseja-se
 - * Reduzir **erros**
 - * **Simplificar** a programação
 - * Fornecer **comunicação** entre programadores
- * Idealmente **métodos formais**

>>> Testes

- * **Continuamente** testado
- * Perguntas:
 - * É executado **corretamente**?
 - * Segue os objetivos das **especificações**?
- * **Melhorar** as especificações
- * Fases
 - * **Alfa**: testes internos
 - * **Beta**: testes externos

>>> Documentação

- * Documentação **externa**: Manual do usuário
 - * funcionalidades
 - * uso
- * Documentação **interna**: comentários e documentos descritivos
 - * compreensão do código

>>> Manutenção

- * Programação após entrega
- * Correção de erros
- * Novas versões

>>> Erros

- * Implementação
- * Compreensão
- * Especificação

>>> Versões

- * Necessidade de novas funcionalidades
- * Mudanças externas (leis, técnicas, etc)
- * Novas tecnologias

```
>>> História
```

>>> Espaço Restrito

- * Primeiros computadores: **alguns milhares** de palavras
- * Logo: programas **pequenos e simples**

>>> Linguagem de máquina

LD R1, 1001

LD R2, 1002

ADD R1, R2

ST 1000, R1

HALT

>>> Linguagem de Alto Nível

- * Operações **mais complexas**

- * Esquecer a **arquitetura** e se concentrar no **problema**

>>> Equipes

- * Programador: **poucos milhares** de linhas de código depurado por ano
- * Precisa-se de **equipes**
- * Trabalho precisa ser **distribuído**
- * **Interação** entre programadores
- * **Testes** de partes
- * **Técnicas** precisas

>>> Técnica e Linguagem

- * Técnica refletida na linguagem
- * Se técnica não condiz com a linguagem
 - * Implementação fica difícil

>>> Orientação a Objetos

- * **Organiza** o código
- * **Isola detalhes** de implementação
- * Facilita **reaproveitamento** de código
- * Facilita desenvolvimento de códigos **extensíveis**

>>> Programação estruturada

- * **Substituição de código** é dificultada se houver múltiplos pontos de entrada
- * **Ideal**: cada bloco possui apenas um ponto de entrada e um ponto de saída

>>> Múltiplas entradas

```
if (a > b) goto la;
```

```
tmp = a;
```

```
a = b;
```

```
b = tmp;
```

```
while (a != b) {
```

```
    if (b > a) {
```

```
        tmp = a;
```

```
        a = b;
```

```
        b = tmp;
```

```
    }
```

```
la:
```

```
    a -= b;
```

```
}
```

>>> Múltiplas saídas

```
i = 0;
while (i < n) {
    if (dados[i] == chave) goto fim;
    i++;
}
dados[i] = chave;
cont[i] = 0;
n++;
fim:
cont[i]++;
return i;
```

>>> Estruturas Padrão

- * Condicional (**if**)
- * Seleção (**switch**)
- * Repetição (**while, for, do**)
- * Rotinas (**funções**)

>>> Rotinas

- * Corpo da **função** pode ser totalmente substituído
- * Interação com o exterior:
 - * Parâmetros
 - * Retorno
 - * ~~Variáveis globais~~ (evitar)

>>> Estrutura de Dados

- * Dados possuem **relação** entre si
- * **Relações** devem ser expressas na **linguagem**
- * \Rightarrow técnicas de **estruturação de dados**

>>> Estrutura de Dados - Exemplo

- * Precisamos de uma **lista de inteiros**
- * Precisamos guardar os números e sua quantidade
- * Como?

- * **Vetor** de números

- * **Variável** com a quantidade

```
int lista[100];  
int n;  
n = 0;  
/* outras operações */  
lista[n] = 3;  
n++;
```

>>> Problemas

- * `lista` e `n` são independentes no código

- * Alguma outra forma?

- * `struct`

```
struct Lista {  
    int lista[100];  
    int n;  
} umaLista;  
  
umaLista.n = 0;  
/* outras operações */  
umaLista.lista[umaLista.n] = 3;  
umaLista.n++;
```


>>> Vantagens

- * **lista** e **n** são relacionados pelo código (**variável única**)
- * Temos um **tipo** que pode ser reutilizado
- * Se tivermos mais de uma lista, **não temos chance de confundir** as relações

```
>>> Orientação a Objetos
```

>>> Vantagens

- * Estrutura de Dados associada a operações
- * Exemplos:
 - * Inserção
 - * Busca

>>> Estrutura de Dados - Exemplo

```
struct Lista {
    int lista[100];
    int n;
};

bool busca(Lista *lst, int chave) {
    int i = 0;
    while (i < lst->n && lst->lista[i] != chave)
        i++;
    return !(i == lst->n);
}

void insere(Lista *lst, int valor) {
    lst->lista[lst->n] = valor;
    lst->n++;
}
```

>>> Problema

- * No **código**, há separação entre **estrutura** e **operações**
- * Isso está relacionado a **Tipos Abstratos de Dados** (TAD)

>>> Próxima Aula

Introdução à Programação Orientada a Objetos

(começando por Tipos Abstratos de Dados)

>>> Referências

* Apostila e Aulas do Gonzalo Travieso (IFSC/USP)