# IMD0030
# LINGUAGEM DE PROGRAMAÇÃO I

Aula 01 – Apresentação da disciplina e Introdução à Linguagem de Programação C++

(material baseado nas notas de aula do Prof. Silvio Sampaio e Prof. César Rennó-Costa)

IMD INSTITUTO METRÓPOLE DIGITAL

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

# Apresentação

- Professor responsável
  - Kayo Gonçalves e Silva
  - kayo@imd.ufrn.br (use no início do título [IMD0030])
  - Sala A206

- Atendimento extra-classe agendado via email

# Objetivos da disciplina

Capacitar o estudante a utilizar a linguagem de programação C++ para a implementação de programas visando a solução de problemas, aplicando boas práticas de programação.

Operadores de alocação dinâmica • Formas de implementação de TADs (Tipos Abstratos de Dados). • Funções e Recursividade. Tipos de recursão. Recursão x Interação. • Performance, Expressividade. • Introdução a Classes. • Construtores e Destrutores. Tipos compostos. Tipos recursivos. • Gerenciamento de memória. • Modularização de Programas • Depuração e Profiling • Aplicações em estruturas e algoritmos presentes em EDB1

# Competências e habilidades

- **Idealizar de forma algorítmica soluções para problemas**
- **Conhecer e fazer uso de importantes ferramentas** de suporte ao programador
- **Identificar e corrigir problemas** de codificação e execução de programas
- **Dominar o uso dos recursos** básicos da linguagem de programação C++ e sua biblioteca padrão
- **Implementar soluções para problemas** utilizando a linguagem de programação C++

Conteúdos

# Ver no SIGAA

# Metodologia

- **Aulas teóricas** expositivas
- **Aulas práticas** voltadas para a resolução de exercícios de programação e aplicação dos conceitos vistos
- **Laboratórios e Projeto Final** com o objetivo de solucionar problemas por meio de programas implementados na linguagem C++

# Avaliação

**Instrumentos de avaliação**

- Exercícios de programação
- Projeto final de programação
- Avaliações presenciais

# Avaliação

**Unidade 1:** atividade(s) (média simples) e prova
**Unidade 2:** atividade(s) (média simples) e prova
**Unidade 3:** Projeto

**Critérios de avaliação**

- Utilização correta dos conteúdos vistos em aula
- Corretude da execução dos algoritmos implementados, com saída em conformidade com a especificação e as entradas de dados fornecidas
- Aplicação correta de boas práticas de programação (legibilidade, organização e documentação de código)
- Qualidade da escrita

# Avaliação

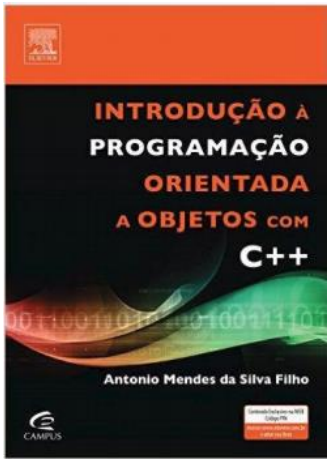
**Rendimento acadêmico**

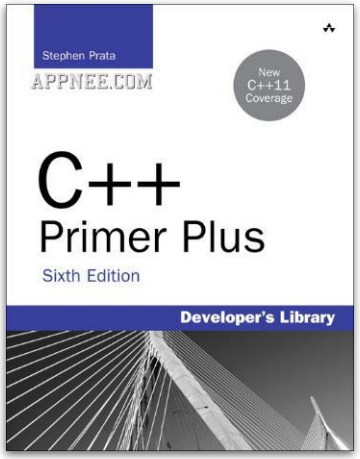
- Não entrega de atividade avaliativa: nota zero na avaliação
  - Não serão aceitos envios atrasados
- Avaliação de reposição
  - Substituição do menor rendimento acadêmico nas unidades (Art. 107 e 110 do Regulamento dos Cursos de Graduação)
  - Avaliação individual realizada no fim do período letivo, **podendo cobrir parte ou todo o conteúdo ministrado**

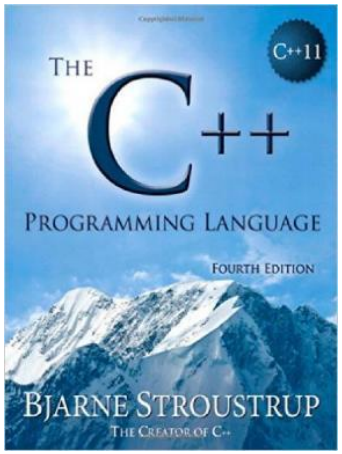
# Bibliografia sugerida

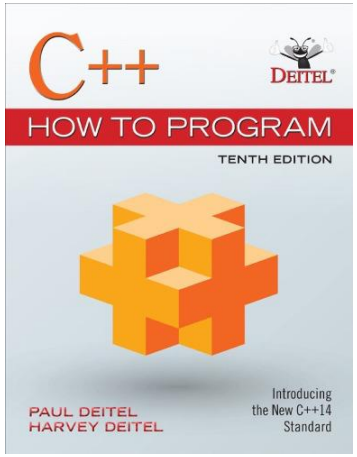
Muitos disponíveis na BCZM

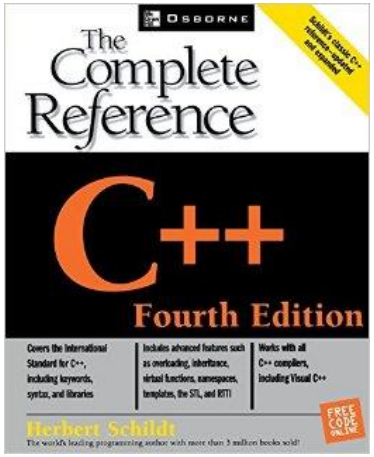
Não tem na BCZM



# Bibliografia sugerida

## Links úteis

- cplusplus.com – The C++ Resources Network: http://www.cplusplus.com/
- cppreference.com: http://en.cppreference.com/w/
- Stack Overflow: http://stackoverflow.com/

# Observações gerais

**Controle de presença**

- Aprovação condicionada à presença mínima de 75%

# Observações gerais

**Sobre plagiarismo**

- O trabalho em cooperação é estimulado, sendo aceitável a discussão de ideias e estratégias

- Não será permitida a utilização de (parte de) códigos-fonte de outros estudantes

- **Trabalhos copiados em todo ou em parte de outros estudantes ou da Internet receberão automaticamente nota zero**

# Recomendações

- **Será aceito** código fonte desenvolvido utilizando recursos de C ou mescla C e C++ (não exagere)

- **Não será adotada qualquer IDE**, privilegiando-se o uso de editores de texto simples e ferramentas em linha de comando

- **Será fortemente cobrada** a implementação de programas sem mensagens de aviso (*warnings*)

# Recomendações

- Programas devem compilar e não apresentar falha de segmentação (*segmentation fault*) na execução

- Programas devem produzir a saída esperada de acordo com a especificação fornecida

# Recomendações

**Será ampla e fortemente estimulada** a aplicação de boas práticas de programação

- Codificação de programas de maneira legível (com indentação de código, nomes consistentes, etc.)
- Documentação adequada na forma de comentários
- Organização de programas complexos na forma de funções modulares e arquivos
- Teste sistemático de programas na forma de casos de teste

# Recomendações

- O conteúdo da disciplina é **incremental**
  - Os conceitos avançados somente podem ser compreendidos quando os básicos forem bem assimilados
- O conteúdo da disciplina é **abrangente** e requer um **esforço importante e permanente**
- Os exercícios propostos devem ser resolvidos para **melhor fixação** dos conceitos apresentados

# Cronograma de aulas

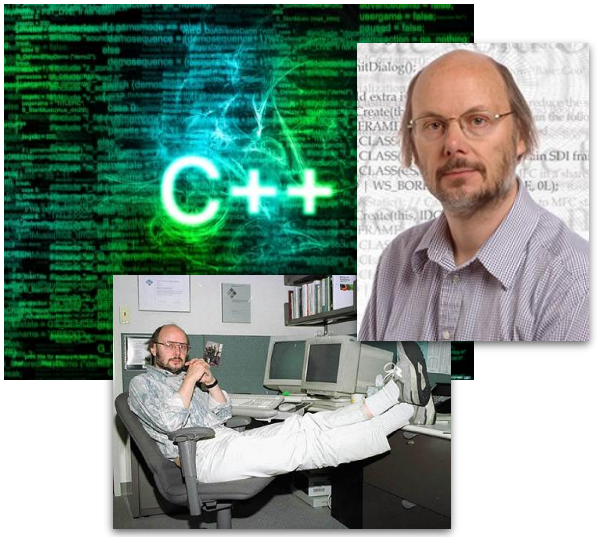
- Ver no SIGAA
- Todo conteúdo será ministrado
- Flexível: depende parcialmente do andamento da turma
  - Uma aula pode ter parte do conteúdo ministrado na próxima aula
  - Uma aula pode ter conteúdo da próxima aula adiantado

# Competências e habilidades


?

# A linguagem de programação C++

- **Linguagem de programação multiparadigma** de propósito geral padronizada pela ISO
- Considerada de médio nível, pois combina características de linguagens de alto e baixo níveis
- Criada por Bjarne Stroustrup no AT&T Bell Labs no início dos anos 1980
- Após a padronização ISO de 1998 e a posterior revisão de 2003, uma nova versão da especificação da linguagem, conhecida como C++11, foi lançada em 2011



# A linguagem de programação C++

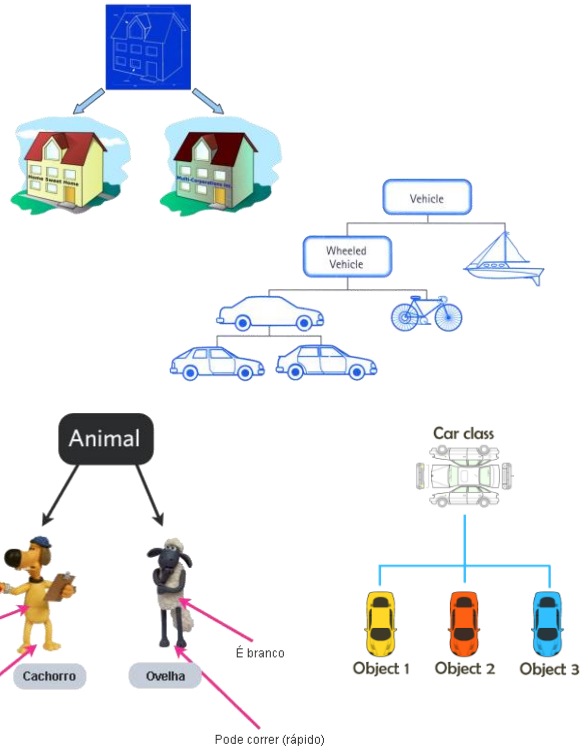
- Principais objetivos
  - Inserir o paradigma de programação orientada a objetos em C
  - **Manter-se simultaneamente próxima da máquina e (da análise) do problema**

- Por quê?
  - A medida que os sistemas de software crescem, também cresce a **complexidade** associada a eles, tornando difícil satisfazer um grande número de requisitos

- O **paradigma de programação orientada a objetos** oferece uma nova forma para tratar essa complexidade
  - Organiza o código em componentes lógicos que facilitam a programação

# Paradigma Orientado a Objetos

- Paradigma de programação que permite aos programadores raciocinar e solucionar problemas em termos de **objetos** diretamente associados às entidades reais
  - Mais próximo da forma como pensamos naturalmente!




- A programação orientada a objetos serve de **elo** entre os problemas existentes e as soluções computacionais
  - Grande importância na solução de problemas complexos

# Comparativo entre C, C++ e Java

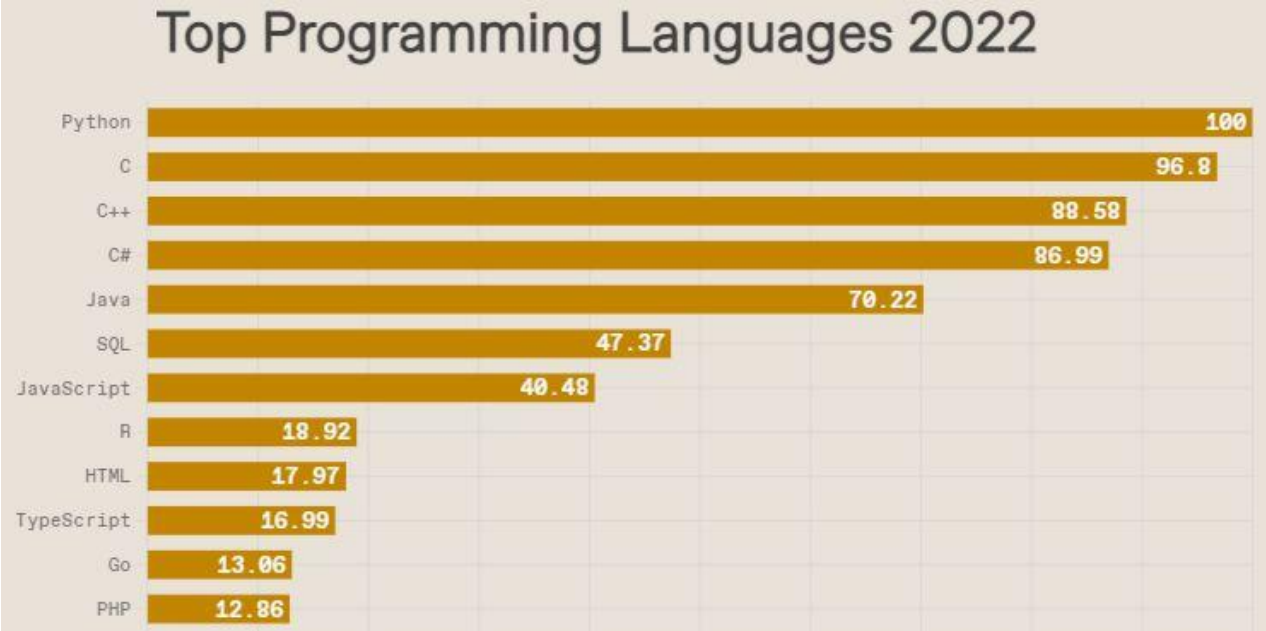
- C, C++ e Java estão entre as **linguagens de programação mais populares**
  - C segue o paradigma procedural, no qual as soluções são baseadas na decomposição de **tarefas** distintas

- Java segue o paradigma orientado a objetos
  - Soluções baseadas na decomposição de **objetos** distintos

- C++ é híbrida (ou multiparadigma), permitindo seguir os paradigmas procedural e orientado a objetos
  - Partes da solução baseadas em **tarefas** distintas e outras partes em **objetos** distintos

# Top Programming Languages 2022

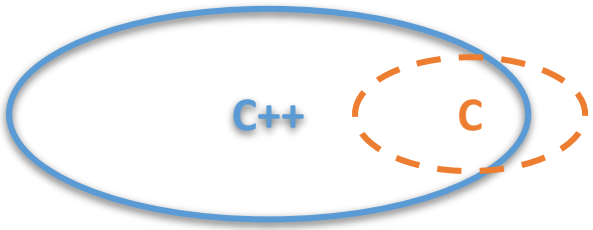
IEEE SPECTRUM



# Top Programming Languages 2023

# Comparativo entre C e C++

- **C++ é uma extensão da linguagem C**
  - Quase toda instrução correta em C é correta em C++




  - Tem todas as vantagens de C, além de permitir abstração de dados e manipulação de objetos

- Ela também é bastante usada na academia devido ao seu **excelente desempenho e uma grande base de usuários**

# Comparativo entre C e C++

| Linguagem C | Linguagem C++ |
|---|---|
| Paradigma procedural | Multiparadigma (procedural e orientado a objetos) |
| Inteiro como valor booleano | Tipo `bool` |
| Variáveis devem ser declaradas no início de um bloco | Variáveis podem ser declaradas em qualquer parte de um bloco |
| `stdio.h` define canais de entrada e saída (`printf` e `scanf`) | `iostream` define canais de entrada e saída (`std::cout` e `std::cin`) |
| *String* como vetor de caracteres | Classe `std::string` |
| *Casts* simples | Novos tipos de *cast* |
| Desprovida de suporte a estruturas genéricas | Suporta estruturas de código parametrizadas ou genéricas (*templates*) |

# Comparativo entre C e C++

| Linguagem C | Linguagem C++ |
|---|---|
| Duas funções não podem ter o mesmo nome | Duas funções não podem ter o mesmo protótipo |
| Parâmetros de funções somente podem ser passados por valor | Parâmetros de funções também podem ser passados por referência |
| Argumentos são sempre necessários nas chamadas de funções | Valores padrão podem ser definidos para os argumentos |
| Operadores de baixo nível para alocação e liberação dinâmica de memória (`malloc` e `free`) | Operadores de alto nível para alocação e liberação dinâmica de memória (`new` e `delete`) |
| Desprovida de mecanismo para manipulação de exceções | Dispõe de mecanismo para manipulação de exceções |

# Componentes da linguagem C++

- A inclusão de cabeçalhos com a diretiva `#include` diz ao compilador para inserir um outro arquivo no código fonte
  - Em C++, ela **não necessita mais da extensão do arquivo (.h)**
  - Na biblioteca padrão de C++, `iostream` substitui `stdio.h` de C

- Comentários iniciam com `//` e terminam no fim da linha

Exemplo de código em linguagem C

```c
#include <stdio.h>

int main(void)
{
    /* Comentário no estilo
       de C */
    return 0;
}
```

Exemplo de código em linguagem C++

```c
#include <iostream>

int main(void)
{
    // Comentário no estilo de C++
    /* Comentário no estilo de C
       também é aceito em C++ */
    return 0;
}
```

# Componentes da linguagem C++

- Comando de fluxo de saída padrão
  - `std::cout` substitui `printf`, eliminando os identificadores `%`
  - `<<` é um operador de inserção que direciona o valor a ser impresso para o dispositivo de saída

- O manipulador `std::endl` substitui o caractere `'\n'`

Exemplo de código em linguagem C

```c
#include <stdio.h>

int main(void)
{
    int x = 10;
    printf("Iniciando...\n");
    printf("%i, %f\n", x, 20.5f);
    return 0;
}
```

Exemplo de código em linguagem C++

```cpp
#include <iostream>

int main(void)
{
    int x = 10;
    std::cout << "Iniciando..." << std::endl;
    std::cout << x << ", " << 20.5f << std::endl;
    return 0;
}
```

# Componentes da linguagem C++

- Comando de fluxo de entrada padrão
  - `std::cin` substitui `scanf`, no qual identificadores `%` e operador de endereçamento `&` não são mais necessários
  - `>>` é um operador de extração que recebe um valor digitado pelo usuário através do dispositivo de entrada

Exemplo de código em linguagem C

```c
#include <stdio.h>

int main(void)
{
    int x;
    float y;
    scanf("%i %f", &x, &y);
    return 0;
}
```

Exemplo de código em linguagem C++

```cpp
#include <iostream>

int main(void)
{
    int x;
    float y;
    std::cin >> x >> y;
    return 0;
}
```

# Compilando tudo

- Para compilar todos os arquivos e gerar o nosso primeiro programa em C++, utilizaremos o compilador g++

- Processo de compilação:
  `g++ -Wall -pedantic teste.cpp main.cpp -o programa`

  - Note que apenas os arquivos de corpo (`.cpp`) são passados para o compilador

  - Como resultado da compilação, será gerado o arquivo executável de nome `programa`

  - Os parâmetros `-Wall -pedantic` são aqui usados para indicar ao compilador que qualquer tipo de mensagem de aviso (*warning*) deve ser interpretada como um erro, devendo o programador corrigir o código que dá origem ao aviso

- Execução: basta executar o arquivo de nome `programa`