"Para quem só sabe usar martelo, todo problema é um prego" (Abraham Maslow).

I/O, Classes e objetos

Extensões dos arquivos

Como são os arquivos em C?

Códigos fonte:

Arquivos de header:

Extensões dos arquivos

Como são os arquivos em C?

- Códigos fonte: .c
- Arquivos de header: .h

Em C++ os arquivos possuem as seguintes extensões.

- Códigos fonte: .cpp
- Arquivos de header: .hpp
  - Ou .h, mas prefira usar .hpp

Olá mundo

```cpp
#include<iostream>

int main(){
   std::cout << "Ola Mundo" << std::endl;

   return 0;
}
```

Olá mundo

Equivalente da biblioteca stdio.h no C.

Modo portável de se incluir um \n.

```cpp
#include<iostream>

int main(){
   std::cout << "Ola Mundo" << std::endl;

   return 0;
}
```

Equivalente a um printf.

Lendo da entrada padrão

Para ler da entrada padrão, basta utilizar *cin*.

biblioteca *iostream*.

Como com scanf, podemos ler várias variáveis ao mesmo tempo.

Exemplo

Lembrete: para compilar, você pode usar o comando
`g++ nomeArquivo.cpp -o nomeBinario`

Entenda, compile e execute o exemplo a seguir.

```
#include<iostream>

int main(){
   int meuInteiro;
   char meuChar;

   std::cout << "Digite um inteiro e um char" << std::endl;
   std::cin >> meuInteiro >> meuChar;
   std::cout << "Voce Digitou " << meuInteiro << " " << meuChar << " " << std::endl;

   return 0;
}
```

Da aula passada

Considerando o exercício solicitado na aula passada.

Quais são os membros da struct?

Da aula passada

Considerando o exercício solicitado na aula passada.

- Quais são os membros da struct?
- O que a struct representa?

Da aula passada

Considerando o exercício solicitado na aula passada.

Quais são os membros da struct?

O que a struct representa?

Por que uma struct, se podíamos ter criado as variáveis individuais para representar idade, cpf, ...?

Da aula passada

Considerando o exercício solicitado na aula passada.

Quais são os membros da struct?

O que a struct representa?

Por que uma struct, se podíamos ter criado as variáveis individuais para representar idade, cpf, ...?

A struct ocupa espaço na memória? O que ocupa espaço na memória?

Criando as primeiras Classes

Uma struct em C é uma forma de agregar os elementos que pertencem a um “tipo de objeto” específico.

Mais legível.

Podemos reutilizar structs prontas (e.g. struct FILE).

Criando as primeiras Classes

Uma struct em C é uma forma de agregar os elementos que pertencem a um “tipo de objeto” específico.

Mais legível.

Podemos reutilizar structs prontas (e.g. struct FILE).

Inicialmente, você pode ver uma **classe** de maneira similar a uma struct.

Mas leva os conceitos de reuso e encapsulamento a outro nível.

Uma **struct** armazena **dados** referentes a determinados objetos.

Uma **classe** possui **dados e comportamentos**.

Criando as primeiras Classes

- Uma classe em C++ é comumente separada em dois componentes.
- Um arquivo .hpp de **header** (cabeçalho), que contém a **definição da classe**.

Criando as primeiras Classes

- Uma classe em C++ é comumente separada em dois componentes.
- Um arquivo .hpp de **header** (cabeçalho), que contém a **definição da classe**.
    - Definimos as variáveis que vão armazenar os dados para essa classe.
        - Chamamos de **dados membro**, ou **variáveis membro**.
            - Em Java, chamamos essas variáveis de atributos.

Criando as primeiras Classes

- Uma classe em C++ é comumente separada em dois componentes.
- Um arquivo .hpp de **header** (cabeçalho), que contém a **definição da classe**.
  - Definimos as variáveis que vão armazenar os dados para essa classe.
    - Chamamos de **dados membro**, ou **variáveis membro**.
      - Em Java, chamamos essas variáveis de atributos.
  - Definimos as funções, mas **não as implementamos**.
    - Damos apenas os **protótipos** (assinaturas) das funções, indicando seus nomes, parâmetros e retorno.
    - Chamamos as funções de uma classe de **funções membro**.
      - Em Java, chamamos de métodos.

Criando as primeiras Classes

Quais são os possíveis **dados membro** de uma classe Pessoa?

Criando as primeiras Classes

Quais são os possíveis **dados membro** de uma classe Pessoa?

```
char nome[50];
unsigned long cpf;
unsigned char idade;
```

Classe Pessoa

Crie um arquivo **Pessoa.hpp**.

Com **P maiúsculo**.

```
#ifndef PESSOA_HPP
#define PESSOA_HPP
class Pessoa{
    public:
        char nome[50];
        unsigned long cpf;
        unsigned char idade;
};
#endif
```

Nome da Classe começa com **Maiúsculo**.

Ponto e vírgula para indicar o fim da definição da classe.

Guardas

Os **ifndef** são chamados de **guardas.**

Arquivos de cabeçalho podem ser incluídos em diversas partes do programa.

Guardas impedem que o arquivo seja compilado e adicionado ao programa mais de uma vez, gerando conflitos e overhead.

Regra de nome para o guarda: NOME_DA_CLASSE_HPP

Inclua os guardas **apenas** nos .hpp

```
#ifndef PESSOA_HPP
#define PESSOA_HPP
class Pessoa{
   public:
       char nome[50];
       unsigned long cpf;
       unsigned char idade;
};
#endif
```

Guardas

Para mais detalhes, leia a Seção 15.3.3 Include Guards de Bjarne Stroustrup. The C++ Programming Language. 2013.

Os **ifndef** são chamados de **guardas.**

- Arquivos de cabeçalho podem ser incluídos em diversas partes do programa.
- Guardas impedem que o arquivo seja compilado e adicionado ao programa mais de uma vez, gerando conflitos e overhead

Regra de nome para o guarda: NOME_DA_CLASSE_HPP

Inclua os guardas **apenas** nos .hpp

```
#ifndef PESSOA_HPP
#define PESSOA_HPP
class Pessoa{
   public:
       char nome[50];
       unsigned long cpf;
       unsigned char idade;
};
#endif
```

Public

Public é um **modificador de acesso**.

Indica que podemos acessar os membros a partir de qualquer lugar.

Veremos mais sobre isso nas próximas aulas.

```
#ifndef PESSOA_HPP
#define PESSOA_HPP
class Pessoa{
   public:
        char nome[50];
        unsigned long cpf;
        unsigned char idade;
};
#endif
```

Funções membro

Adicionamos os **protótipos** (assinaturas) da funções membro nos .hpp.

Não implementamos no .hpp.

Exceto em alguns casos especiais que veremos no futuro.

```
#ifndef PESSOA_HPP
#define PESSOA_HPP
class Pessoa{
   public:
        bool validarCPF(unsigned long cpfTeste);
        char nome[50];
        unsigned long cpf;
        unsigned char idade;
};
#endif
```

Funções membro

Como em C, não é necessário especificar o nome da variável no protótipo (será descartado pelo compilador), mas é uma **boa prática**.

Adicionamos os **protótipos** (assinaturas) da funções membro nos .hpp.

Não implementamos no .hpp.

Exceto em alguns casos especiais que veremos no futuro.

```
#ifndef PESSOA_HPP
#define PESSOA_HPP
class Pessoa{
   public:
        bool validarCPF(unsigned long cpfTeste);
        char nome[50];
        unsigned long cpf;
        unsigned char idade;
};
#endif
```

Implementação

Arquivo de **implementação**.

Arquivos .cpp

Possui o mesmo nome do arquivo .hpp

Contém, dentre outros, a **implementação das funções membro**.

O arquivo .cpp **deve incluir (via include) o arquivo de cabeçalho.**

Não inclua guardas no cpp, exceto se você tiver um bom motivo para isso.

Implementação

- C++ faz uma separação explícita entre a **interface** e a **implementação**.
- O arquivo .hpp dá a interface.
  - Indica como os objetos são (e.g., dados e funções membro).
- O arquivo .cpp dá a implementação concreta.
  - Implementa as funções membro especificadas no .hpp.
- **Vantagens? Desvantagens?**

Implementação

- C++ faz uma separação explícita entre a **interface** e a **implementação**.
- O arquivo .hpp dá a interface.
  - Indica como os objetos são (e.g., dados e funções membro).
- O arquivo .cpp dá a implementação concreta.
  - Implementa as funções membro especificadas no .hpp.
- **Vantagens.**
  - Os programas dependem apenas dos .hpps. Se você precisar mudar a implementação, precisa apenas alterar o .cpp.
    - Exemplo: um .cpp para x86, e outro para ARM.
      - O programa que usa a classe não precisa saber disso, já que ele conhece só o .hpp.
  - Facilita e torna mais rápida a compilação.
- **Desvantagens.**
  - Trabalhoso.

Implementação

Crie um arquivo Pessoa.cpp

```cpp
#include "Pessoa.hpp"

bool Pessoa::validarCPF(unsigned long cpfTeste){
   unsigned int somatorioValidaUltimo;
   unsigned int modulo;
   unsigned int somatorioValidaPenultimo{0};
   unsigned int ultimo{(unsigned int)(cpfTeste%10)};
   cpfTeste = cpfTeste/10;
   unsigned int penultimo{(unsigned int)(cpfTeste%10)};
   cpfTeste = cpfTeste/10;

   somatorioValidaUltimo = penultimo*2;
   for(int i{2}; i <= 11; i++){
       modulo = cpfTeste%10;
       cpfTeste = cpfTeste/10;
       somatorioValidaPenultimo += modulo*i;
       somatorioValidaUltimo += modulo*(i+1);
   }
   modulo = somatorioValidaPenultimo%11;
   if(modulo < 2){
       if(!penultimo) return false;//cpf invalido
   }else{
       if(penultimo != 11 - modulo) return false;//cpf invalido
   }
   modulo = somatorioValidaUltimo%11;
   if(modulo < 2){
       if(!ultimo) return false;//cpf invalido
   }else{
       if(ultimo != 11-modulo) return false;//cpf invalido
   }
   return true;//cpf valido
}
```

Implementação

Inclua o Header de Pessoa.

```
#include "Pessoa.hpp"

bool Pessoa::validarCPF(unsigned long cpfTeste){
    unsigned int somatorioValidaUltimo;
    unsigned int modulo;
    unsigned int somatorioValidaPenultimo{0};
    unsigned int ultimo{(unsigned int)(cpfTeste%10)};
    cpfTeste = cpfTeste/10;
    unsigned int penultimo{(unsigned int)(cpfTeste%10)};
    cpfTeste = cpfTeste/10;

    somatorioValidaUltimo = penultimo*2;
    for(int i{2}; i <= 11; i++){
        modulo = cpfTeste%10;
        cpfTeste = cpfTeste/10;
        somatorioValidaPenultimo += modulo*i;
        somatorioValidaUltimo += modulo*(i+1);
    }
    modulo = somatorioValidaPenultimo%11;
    if(modulo < 2){
        if(!penultimo) return false;//cpf invalido
    }else{
        if(penultimo != 11 - modulo) return false;//cpf invalido
    }
    modulo = somatorioValidaUltimo%11;
    if(modulo < 2){
        if(!ultimo) return false;//cpf invalido
    }else{
        if(ultimo != 11-modulo) return false;//cpf invalido
    }
    return true;//cpf valido
}
```

Implementação

Implemente as funções membro.
É **obrigatório** que antes do protótipo você coloque **NomeClasse::** para indicar que a função sendo implementada pertence ao escopo da NomeClasse (nesse caso, Pessoa).

O operador :: é o operador de resolução de escopo em C++.

```cpp
#include "Pessoa.hpp"

bool Pessoa::validarCPF(unsigned long cpfTeste){
    unsigned int somatorioValidaUltimo;
    unsigned int modulo;
    unsigned int somatorioValidaPenultimo{0};
    unsigned int ultimo{(unsigned int)(cpfTeste%10)};
    cpfTeste = cpfTeste/10;
    unsigned int penultimo{(unsigned int)(cpfTeste%10)};
    cpfTeste = cpfTeste/10;

    somatorioValidaUltimo = penultimo*2;
    for(int i{2}; i <= 11; i++){
        modulo = cpfTeste%10;
        cpfTeste = cpfTeste/10;
        somatorioValidaPenultimo += modulo*i;
        somatorioValidaUltimo += modulo*(i+1);
    }
    modulo = somatorioValidaPenultimo%11;
    if(modulo < 2){
        if(!penultimo) return false;//cpf invalido
    }else{
        if(penultimo != 11 - modulo) return false;//cpf invalido
    }
    modulo = somatorioValidaUltimo%11;
    if(modulo < 2){
        if(!ultimo) return false;//cpf invalido
    }else{
        if(ultimo != 11-modulo) return false;//cpf invalido
    }
    return true;//cpf valido
}
```

Implementação

Para inicializar uma variável criada, use `{}`.

Exemplo: `int valor{0};`

Introduzido no C++11 para prover:

- Inicialização uniforme.
- Evitar problemas de *most vexing parse.*
- Avisar sobre *narrowing conversions.*
  - Conversão implícita de um tipo maior para um menor.

```
#include "Pessoa.hpp"

bool Pessoa::validarCPF(unsigned long cpfTeste){
   unsigned int somatorioValidaUltimo;
   unsigned int modulo;
   unsigned int somatorioValidaPenultimo{0};
   unsigned int ultimo{(unsigned int)(cpfTeste%10)};
   cpfTeste = cpfTeste/10;
   unsigned int penultimo{(unsigned int)(cpfTeste%10)};
   cpfTeste = cpfTeste/10;

   somatorioValidaUltimo = penultimo*2;
   for(int i{2}; i <= 11; i++){
       modulo = cpfTeste%10;
       cpfTeste = cpfTeste/10;
       somatorioValidaPenultimo += modulo*i;
       somatorioValidaUltimo += modulo*(i+1);
   }
   modulo = somatorioValidaPenultimo%11;
   if(modulo < 2){
       if(!penultimo) return false;//cpf invalido
   }else{
       if(penultimo != 11 - modulo) return false;//cpf invalido
   }
   modulo = somatorioValidaUltimo%11;
   if(modulo < 2){
       if(!ultimo) return false;//cpf invalido
   }else{
       if(ultimo != 11-modulo) return false;//cpf invalido
   }
   return true;//cpf valido
}
```

Implementação

Para inicializar uma variável criada, use `{}`.

Exemplo: `int valor{0};`

Introduzido no C++11 para prover:

- Inicialização uniforme.
- Evitar problemas de *most vexing parse.*
- Avisar sobre *narrowing conversions.*
  - Conversão implícita de um tipo maior para um menor.

```
#include "Pessoa.hpp"

bool Pessoa::validarCPF(unsigned long cpfTeste){
   unsigned int somatorioValidaUltimo;
   unsigned int modulo;
   unsigned int somatorioValidaPenultimo{0};
   unsigned int ultimo{(unsigned int)(cpfTeste%10)};
   cpfTeste = cpfTeste/10;
   unsigned int penultimo{(unsigned int)(cpfTeste%10)};
   cpfTeste = cpfTeste/10;

   somatorioValidaUltimo = penultimo*2;
   for(int i{2}; i <= 11; i++){
       modulo = cpfTeste%10;
       cpfTeste = cpfTeste/10;
       somatorioValidaPenultimo += modulo*i;
       somatorioValidaUltimo += modulo*(i+1);
   }
   modulo = somatorioValidaPenultimo%11;
   if(modulo < 2){
       if(!penultimo) return false;//cpf invalido
   }else{
       if(penultimo != 11 - modulo) return false;//cpf invalido
   }
   modulo = somatorioValidaUltimo%11;
   if(modulo < 2){
       if(!ultimo) return false;//cpf invalido
   }else{
       if(ultimo != 11-modulo) return false;//cpf invalido
   }
   return true;//cpf valido
}
```

Instanciando

Acabamos de criar nossa primeira classe!

- Pode ser importada e utilizada em qualquer programa C++.
- O **reuso** para "não reinventar a roda" é a **regra de ouro** na orientação a objetos.

Ao criar uma classe, criamos apenas uma estrutura que não foi alocada na memória.

- Assim como uma struct em C não é alocada na memória, a menos que você crie variáveis do tipo dessa struct.

Instanciando

Ao alocar memória para uma variável do tipo da classe que criamos, estamos **instanciando um objeto dessa classe.**

Uma variável da classe X que foi instanciada na memória é chamada de **objeto da classe X**, ou **objeto X**.

Main

Vamos utilizar o main para criar alguns objetos de teste.

Crie um arquivo chamado **main.cpp**

O main não é uma classe, então não possui um .hpp

```cpp
#include <iostream>
#include "Pessoa.hpp"

int main(){
   Pessoa p1;
   std::cin >> p1.nome;
   std::cout << p1.nome << std::endl;
   return 0;
}
```

Main

p1 é um **objeto da classe Pessoa**.

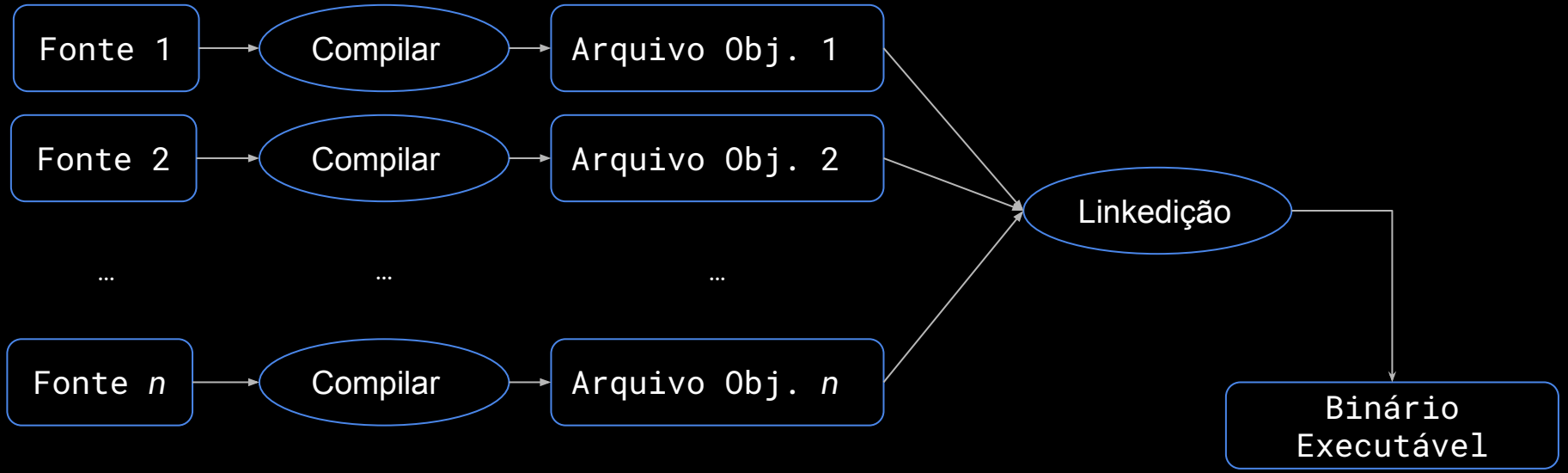
Para acessar os membros do objeto, utilizamos o operador . (ponto).

Da mesma forma que fazemos com structs em C.

```cpp
#include <iostream>
#include "Pessoa.hpp"

int main(){
   Pessoa p1;
   std::cin >> p1.nome;
   std::cout << p1.nome << std::endl;
   return 0;
}
```

Compilando

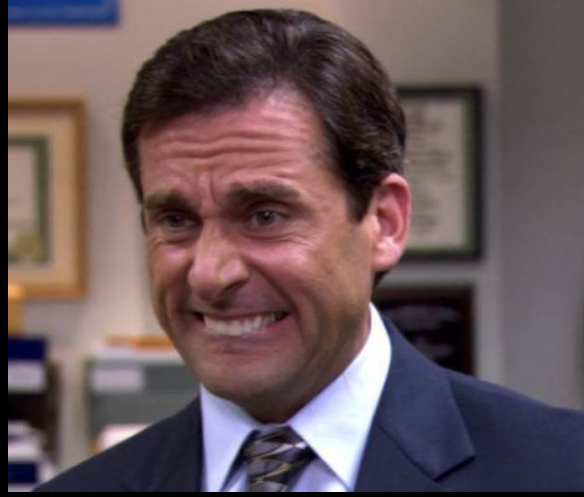

Fonte 1
Compilar
Arquivo Obj. 1
Fonte 2
Compilar
Arquivo Obj. 2
...
...
...
Fonte n
Compilar
Arquivo Obj. n
Linkedição
Binário
Executável

Compilando

Compilar sem uma IDE agora pode se tornar um terror.

Compilar vários arquivos, e fazer a linkedição.

Lembre-se que cada classe possui seu .hpp e .cpp, e tudo deve ser linkado no final.



Make

Vamos utilizar o make.

Ferramenta para realizar o build automaticamente por linha de comando.

IDEs como o Eclipse podem criar o makefile automaticamente para você.

Makefile

Existe um arquivo makefile disponível como exemplo no site da disciplina.

Coloque no mesmo diretório onde está o código fonte do programa.

Na linha de comando, digite `make`, e seu programa será compilado.

Digite `make clean` para apagar todos os arquivos compilados e temporários, deixando apenas os fontes.

Estude sobre o make, pois você precisará editá-lo nas próximas aulas para compilar os próximos programas.

Makefile

```
parametrosCompilacao=-Wall #-Wshadow
nomePrograma=aula

all: $(nomePrograma)

$(nomePrograma): main.o Pessoa.o
	g++ -o $(nomePrograma) main.o Pessoa.o $(parametrosCompilacao)

main.o: main.cpp
	g++ -c main.cpp $(parametrosCompilacao)

Pessoa.o: Pessoa.hpp Pessoa.cpp
	g++ -c Pessoa.cpp $(parametrosCompilacao)

clean:
	rm -f *.o *.gch $(nomePrograma)
```

Um main chamando a função membro

```
#include<iostream>

#include"Pessoa.hpp"

int main(){
   Pessoa p1;
   int idade;
   std::cout << "Nome: ";
   std::cin >> p1.nome;
   std::cout << "Idade: ";
   std::cin >> idade;
   p1.idade = idade;
   bool valido = false;
   while(!valido){
       std::cout << "CPF: ";
       std::cin >> p1.cpf;
       valido = p1.validarCPF(p1.cpf);
   }
   std::cout << "Dados da pessoa: " << p1.nome << "\t"
     << (unsigned short int)p1.idade << "\t" << p1.cpf << std::endl;

   return 0;
}
```

Até o momento...

Até o momento transformamos o exemplo do CPF de C para C++ usando uma classe simples.

Deixamos as coisas mais organizadas.

Mas parece que complicamos muito para um benefício pequeno.

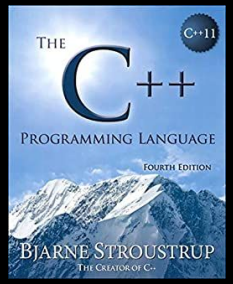
Nas próximas aulas veremos como deixar as operações das classes mais transparentes.

Exercícios

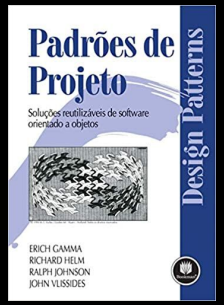
1. Crie mais objetos do tipo pessoa no main e realize testes.
2. Pesquise em detalhes sobre #pragma once.
3. Estude o make passado em aula, e leia sobre o uso do make para compilar programas C e C++.
4. Crie uma classe que representa um retângulo. Crie objetos dessa classe no main e realize testes.

Referências

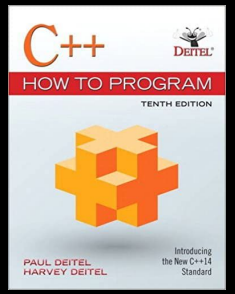
Bjarne Stroustrup. The C++ Programming Language. Addison-Wesley, 2013.



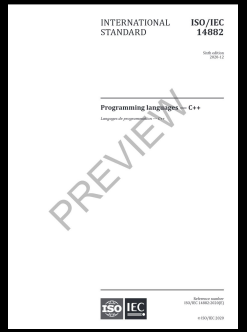
Gamma, E. Padrões de Projetos: Soluções Reutilizáveis. Bookman. 2009.



Deitel, H. M., Deitel, P. J. C++: como programar. 5a ed. Pearson Prentice Hall. 2006.



ISO/IEC 14882:2020 Programming languages - C++: www.iso.org/obp/ui/#iso:std:iso-iec:14882:ed-6:v1:en



Licença