

# I/O, Classes e objetos

Paulo Ricardo Lisboa de Almeida

# Extensões dos arquivos

- Em C++ os arquivos possuem as seguintes extensões
  - Códigos fonte são .cpp
  - Arquivos de header são .hpp
- Como eram os arquivos em C?

# Extensões dos arquivos

- Em C++ os arquivos possuem as seguintes extensões
  - Códigos fonte são .cpp
  - Arquivos de header são .hpp
- Como eram os arquivos em C?
  - Códigos fonte são .c
  - Arquivos de header são .h

# Olá mundo

- Crie um diretório em um local qualquer
  - Exemplo: em sua home do Linux
    - /home/seuNomeUsuario/nomeDiretorioCriado
- No diretório, crie um arquivo chamado olaMundo.cpp
- Edite o arquivo com um editor de texto qualquer

```
#include<iostream>

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

    return 0;
}
```

# Olá mundo

```
#include<iostream>

int main(){
    std::cout << "Ola Mundo" << std::endl;
    return 0;
}
```

Equivalente da biblioteca stdio.h no C

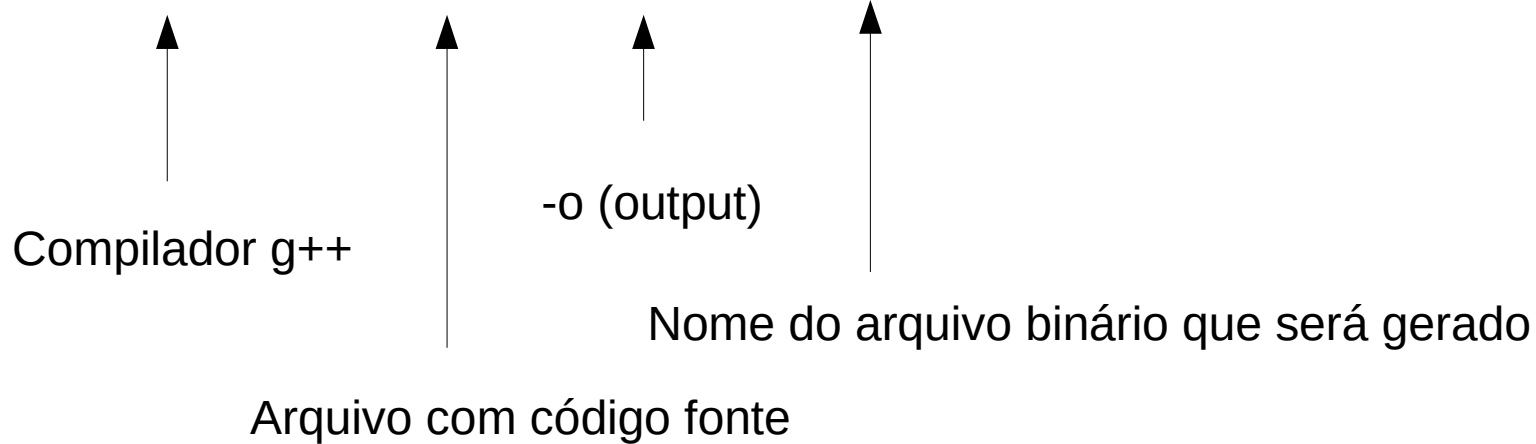
Equivalente a um \n

std::cout é o equivalente a um printf

# Compilando

- Em um terminal, digite

*g++ olaMundo.cpp -o olaMundo*



# Executando

- Caso a compilação ocorra sem problemas, o seu diretório agora vai conter um arquivo chamado olaMundo
- Para executar, digite no terminal
  - ./olaMundo

# Lendo do teclado

- Para ler do teclado, basta utilizar cin (biblioteca iostream)
- Como com scanf, podemos ler várias variáveis ao mesmo tempo do teclado



# Exemplo

- 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;  
}
```

# Criando as primeiras classes

- Quais são os membros da struct?
- O que a struct representa?
- Por que criamos uma struct, se podíamos ter criado as variáveis individualmente (no main por exemplo)?
- A struct ocupa espaço na memória? O que realmente ocupa espaço na memória?

# Criando as primeiras classes

- A struct em C é uma forma de agregar os elementos que pertencem a um “objeto” específico
  - Tornamos as coisas mais legíveis
  - Podemos reutilizar structs prontas
- A princípio uma classe é similar a uma struct
  - Mas leva os conceitos de reuso e encapsulamento a outro nível

# Criando as primeiras classes

- Uma classe em C++ é comumente separada em dois componentes
- Um arquivo .hpp que é o header (cabeçalho)
  - O header contém a **definição** da classe
  - Definimos as variáveis que constituem 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

- Para criar uma classe pessoa, como fizemos com a struct pessoa, quais são as variáveis membro?

# Criando as primeiras classes

- Para criar uma classe pessoa, como fizemos com a struct pessoa, quais são as variáveis membro?
  - char nome[50];
  - unsigned long cpf;
  - unsigned char idade;

# Pessoa.hpp

- Crie um arquivo Pessoa.hpp
  - Com o P maiúsculo
- Adicione no arquivo

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

};

#endif
```

Nome da classe.  
Começa com maiúsculo.

# Pessoa.hpp

- Os ifndef são chamados de guardas
  - Os arquivos de cabeçalho podem ser incluídos em diversas partes do programa
  - Os guardas impedem que o arquivo seja compilado e adicionado ao programa mais de uma vez, gerando conflitos
- Regra de nome para o guarda
  - NOMEDOCABEÇALHO\_H
    - Ou
  - NOMEDOCABEÇALHO\_HPP
- Inclua os guardas apenas nos .hpp

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

#endif
```



# Pessoa.hpp

- *Public* indica que podemos acessar os membros a partir de qualquer lugar
- Veremos mais sobre isso nas próximas aulas

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

#endif
```

# Arquivo de implementação

- Os arquivos com a implementação são os .cpp
  - Contém, dentre outros, a implementação das funções membro
- O arquivo com a implementação deve ter o **mesmo nome do arquivo de cabeçalho**, mas com a extensão .cpp
- O arquivo .cpp deve incluir (via include) o arquivo de cabeçalho
- Não inclua guardas, a não ser que você tenha um bom motivo para isso

# Pessoa.cpp

- Crie o arquivo Pessoa.cpp
- Como não temos funções membro para implementar, a única coisa presente no arquivo será a inclusão de Pessoa.hpp
- Adicione em Pessoa.cpp

```
#include "Pessoa.hpp"
```

# Classes

- Acabamos de criar nossa primeira classe!
- Ela pode ser importada e utilizada em qualquer programa em C++
  - O reuso para “não reinventarmos 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

# Classes versus objetos

- Quando alocamos memória para uma “variável do tipo da classe que criamos, estamos **instanciando a classe**
- Uma “variável” da classe X que foi instanciada na memória é chamada de **objeto** da classe X, ou objeto X

# Criando objetos

- 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

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

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

    return 0;
}
```

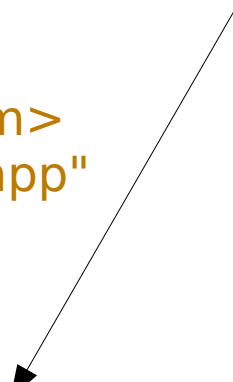
# Criando objetos

- p1 é um objeto da classe Pessoa
- Para acessar os membros do objeto, utilizamos o operador . (**ponto**)

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

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

    return 0;
}
```



# Compilando

- Compilar sem uma IDE agora pode ser tornar um terror
- Temos que compilar vários arquivos, e fazer a linkedição
  - Lembre-se que cada classe possui seu .hpp e .cpp, e tudo deve ser linkado no final
- Para resolver isso, vamos utilizar o make
  - Ferramenta para realizar o build automaticamente por linha de comando



# Make

- Crie um arquivo chamado **makefile** no mesmo diretório dos arquivos fonte
  - Exemplo disponível no Moodle
- Adicione o seguinte

```
parametrosCompilacao=-Wall #-Wshadow
```

```
all: aula2
```

```
aula2: main.o Pessoa.o
```

```
g++ -o aula2 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 *.o aula2
```

# Makefile

- Feito o makefile
- 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 o makefile, pois você precisará editá-lo nas próximas aulas para compilar os próximos programas

# Para a próxima aula

- 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.