>>> Programação Orientada a Objetos (POO)

... Dados Definidos

Prof: André de Freitas Smaira

[~]$ _

>>> Structs

[~]$ _

>>> Estruturas

* Dados podem estar relacionados
* Possivelmente de tipos diferentes
* Queremos indicar a relação entre eles
* Em C++:
    * `struct`
    * `class` (mais tarde).

>>> Data sem struct

```c
int dia_nascimento, mes_nascimento,
ano_nascimento;
int dia_hoje, mes_hoje, ano_hoje;
dia_nascimento = 17;
mes_nascimento = 8;
ano_nascimento = 1996;
dia_hoje = 9;
mes_hoje = 4;
ano_hoje = 2013;

void mostra_data(int dia, int mes, int ano);
```

[~]$ _

>>> Data com struct

```
struct Data {
    int dia, mes, ano;
};

Data nascimento, hoje;
nascimento.dia = 17;
nascimento.mes = 8;
nascimento.ano = 1996;
hoje.dia = 9;
hoje.mes = 4;
hoje.ano = 2013;

void mostra(Data d);
```

Em C++ não precisa do typedef

[~]$ _

>>> Struct

Os membros podem ser de qualquer tipo já definido

```
struct Funcionario {
    char const *nome;
    Data nascimento, ingresso
};

Funcionario maria;
maria.nome = "Maria Silva"
maria.nascimento.dia = 22;
maria.nascimento.mes = 7;
maria.nascimento.ano = 1989
maria.ingresso.dia = 30;
maria.ingresso.mes = 3;
```

[~]$ _

>>> Struct

* Estruturas podem ser inicializadas
* Os valores dos membros são fornecidos na seqüência correta.

```
struct Endereco {
    const char *rua;
    int numero;
};

Endereco destinatario = { "Rua Jose Silva", 1024 };
Endereco remetente{ "Avenida Joao Oliveira", 512 };
```

[~]$ _

>>> Enumerações

[~]$ _

>>> Enumerações

* Pode assumir apenas um de poucos valores
* Nome simbólico para cada um dos valores
* Tipo enum

```
enum Sexo { feminino, masculino };
```

>>> Enumeração

* **enum** pode ser convertido para **int**
* Valores **seqüenciais** e começam de 0
* Podem ser especificados outros valores na definição
* int -> enum: somente conversão explícita

```
enum Avaliacao { bom, regular, ruim };

enum Podium { primeiro = 1, segundo, terceiro };

Avaliacao pontos = bom;
pontos = static_cast<Avaliacao>(pontos + 1);
Podium time = segundo;
int x = time; // x = 2;
```

[~]$ _

>>> Enum class

* **enum -> int**: pode gerar problema para **enum** diferents com mesmo identificador
* Tipo dos valores é sempre **int**
* **Enum class**

```
enum class Cor { vermelho, verde, azul };
enum class Farol { vermelho, amarelo, verde };

Cor c;
Farol f;
c = Cor::vermelho; // OK
c = vermelho; // ERRO: Que vermelho?
c = Farol::vermelho; // ERRO: tipos incompatíveis
int i = Cor::vermelho; // ERRO: Não há conversão
enum class Moeda : char { cara, coroa };
// Variáveis do tipo Moeda vão ocupar um char
```

>>> typedef

Dá nome a um tipo que jpa existe.

```
typedef int Codigo;
typedef double Coord;
typedef double* Vetor;
```

[~]$ _

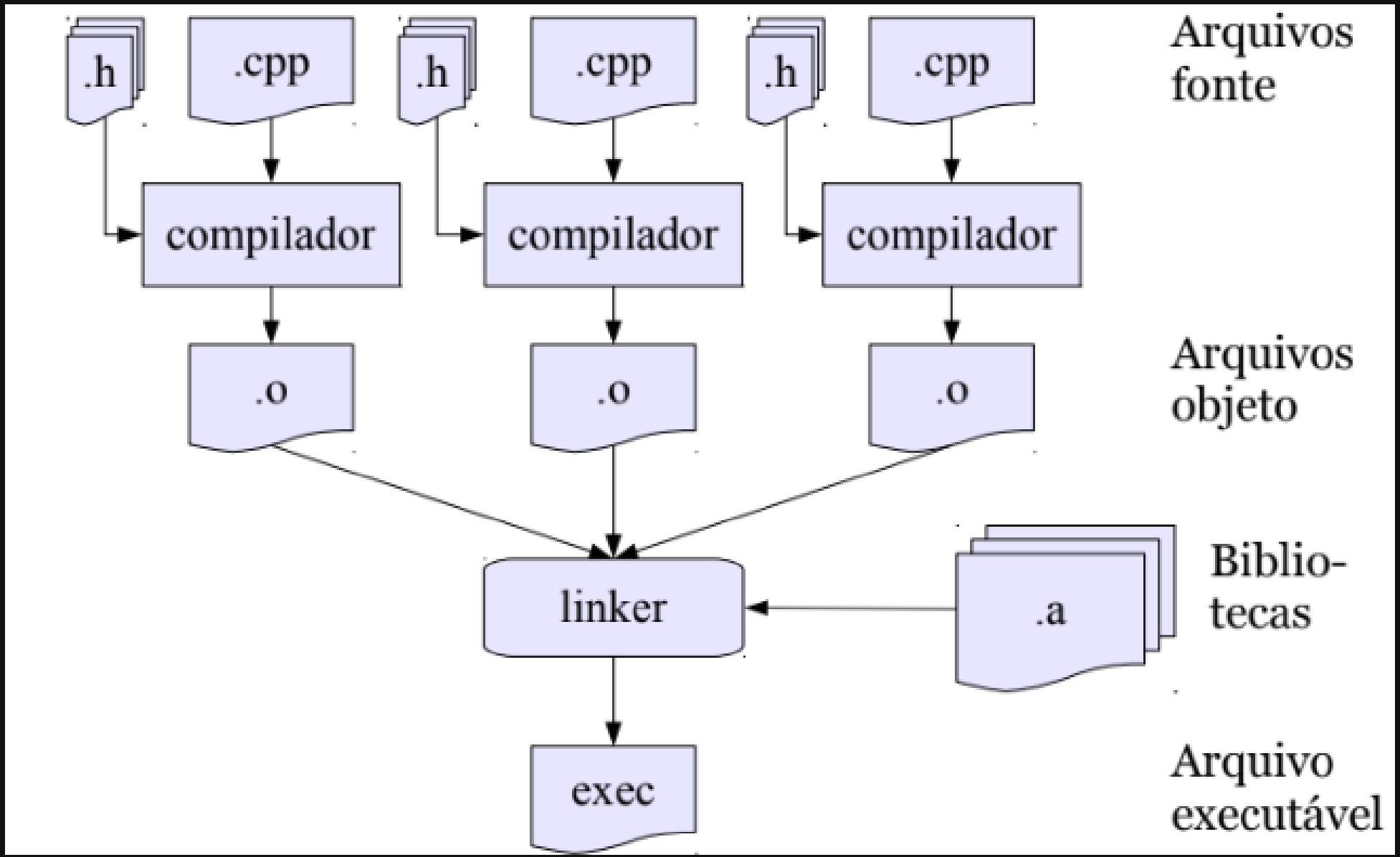
>>> Modularização

[~]$ _

>>> Modularização

* Novos tipos de dados geram a necessidade de bibliotecas
* Compilação separada

>>> Modularização



[~]$ _

>>> Compilação Separada

* **Declarações** no cabeçalho (**.h** ou **.hpp**)
* **Definições** na implementação (**.cpp**)
* Incluir (**#include**) em:
  * Arquivos que usam as declarações
  * Arquivos de implementação
* Arquivos **.h** e **.hpp** devem ser protegidos contra inclusão múltipla.

>>> Compilação Separada

```c
#ifndef _QUEUE_H
#define _QUEUE_H

#define TAMANHO 100

typedef struct {
    int n;
    int fila[TAMANHO];
} FilaE;

FilaE *fila_init();
bool fila_enqueue(FilaE *fila, int item);
int fila_dequeue(FilaE *fila);
int fila_front(FilaE *fila);
bool fila_empty(FilaE *fila);
int fila_size(FilaE *fila);

#endif
```

[~]$ _

>>> Valgrins

* Já foram apresentados ao Valgrind?

```
#include<stdio.h>
int main() {
    int *v = new int[10];
    v[100] = 10;
    printf("%d\n", v[10]);
    return 0;
}

//valgrind --leak-check=full --track-origins=yes -s ./a.out
```

[~]$ _

>>> Proxima aula

Classes

[~]$ _