

Tipos Compostos de Dados

REGISTROS

Introdução

- As **variáveis** e **constantes** armazenam informações
 - Elas ocupam espaço na memória
 - Possuem um tipo
- Os **tipos básicos** armazenam valores:

Inteiros	{	char	ch	=	'W';
		short	sol	=	25;
		int	num	=	45820;
Ponto-flutuantes	{	float	taxa	=	0.25f;
		double	peso	=	1.729156E5;

Introdução

- Porém, com os tipos básicos não é possível armazenar um **conjunto de informações**
 - Como armazenar o peso de 22 jogadores?

```
float p1 = 80.2;  
float p2 = 70.6;  
float p3 = 65.5;  
...  
float p21 = 85.8;  
float p22 = 91.0;
```

Criar 22 variáveis diferentes não é a melhor solução.

- A solução é usar vetores:

```
float peso[22];
```

Introdução

- Com vetores não é possível armazenar um conjunto de **informações de tipos diferentes**
 - Como armazenar um cadastro completo de 22 jogadores (nome, idade, altura, peso, gols, etc.)?

```
char    nome[22][80];  
unsigned idade[22];  
unsigned altura[22];  
float    peso[22];  
unsigned gols[22];
```

Criar vários vetores
não é a melhor
solução.

- A solução é usar **registros**

Introdução

- O registro agrupa informações, de **tipos possivelmente diferentes**, sob um único identificador

```
jogador
char    nome[20];
int     idade;
float   peso;
double  salario;
unsigned gols;
```

Para solucionar o problema anterior poderíamos criar um vetor do tipo **jogador**

```
jogador equipe[22];
```

- Em C++, um **registro** define um **novo tipo de dado**

Declaração

- Declaração de um registro:

Palavra-chave struct

Nome do registro

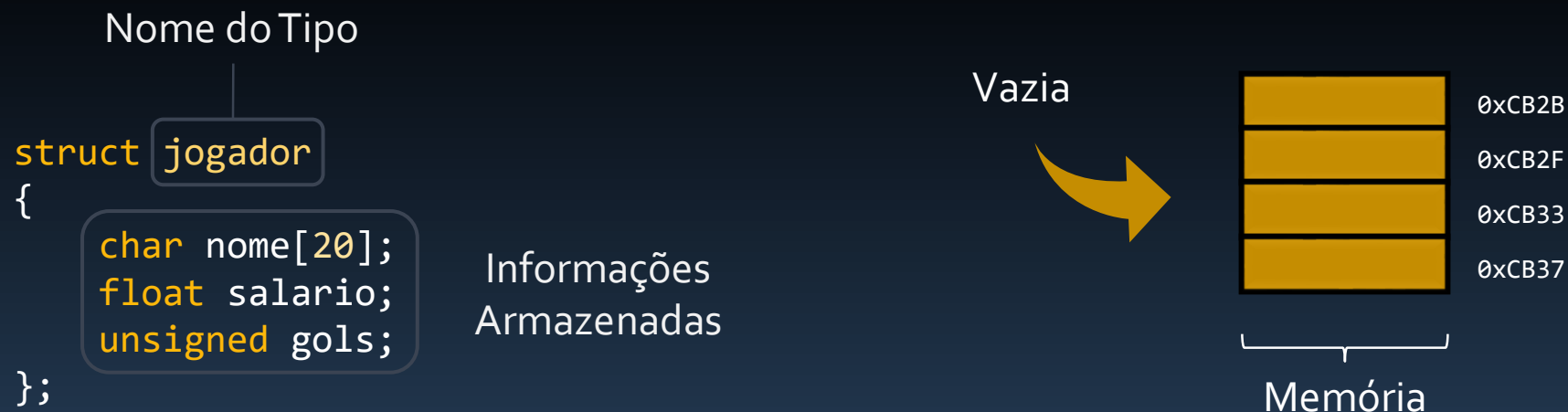
```
struct jogador  
{  
    char nome[20];  
    float salario;  
    unsigned gols;  
};
```

Membros do registro

Finaliza a instrução de declaração

Declaração

- A **declaração do registro** não cria variáveis
 - Não aloca espaço em memória para os dados
 - Apenas define que **tipo de informações** serão armazenadas
 - Define **um nome** para um novo tipo de dado



Declaração

- Os **membros do registro** são definidos por instruções de declaração de variáveis
 - Pode-se usar **qualquer tipo** para as variáveis (incluindo vetores ou mesmo outro registro já definido)

```
struct data
{
    short dia;
    short mes;
    short ano;
};
```

```
struct horario
{
    short hora;
    short min;
    short seg;
};
```

```
struct evento
{
    data dia;
    horario hora;
    char lugar[40];
};
```


Criação de Variáveis

- Declarado o registro, pode-se **criar variáveis** desse novo tipo:

<pre>struct jogador { char nome[20]; float salario; unsigned gols; };</pre>	<pre>jogador pele; jogador zico; jogador beбето;</pre>
---	--

- Em C é obrigatório manter a palavra-chave struct

```
struct jogador beбето; // struct necessário em C
```

- C++ enfatiza que o registro é um novo tipo

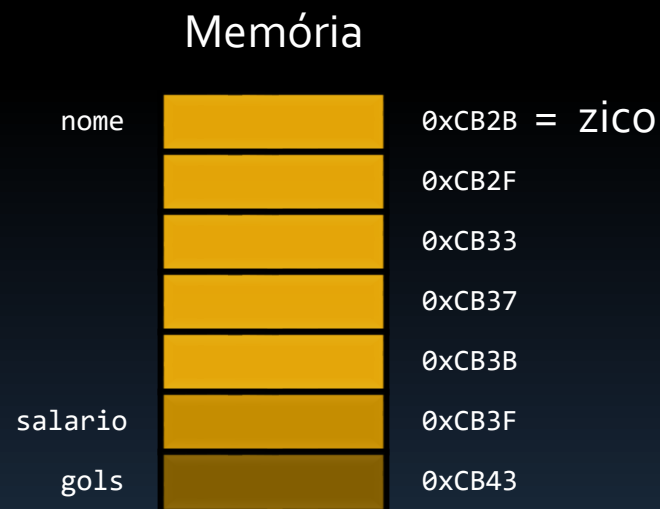
```
jogador beбето; // struct não é necessário em C++
```

Criação de Variáveis

- A variável aloca **espaço em memória**

```
struct jogador  
{  
    char nome[20];  
    float salario;  
    unsigned gols;  
};
```

```
jogador zico;
```



Acesso ao Registro

- Os **campos individuais** de um registro são acessados através do **operador membro (.)**

```
struct jogador
{
    char nome[20];
    float salario;
    unsigned gols;
};

jogador zico;
```

```
zico.gols    = 300;
zico.salario = 40'000;

// atribuição inválida
zico.nome = "Zico";

// use a função strcpy
strcpy(zico.nome, "Zico");
```

Acesso ao Registro

```
#include <iostream>
using namespace std;

struct jogador
{
    char nome[20];
    float salario;
    unsigned gols;
};

int main()
{
    jogador a = {"Bebeto", 200'000, 600};
    jogador b = {"Romário", 300'000, 800};

    cout << "Contratações para o próximo ano: " << a.nome << " e " << b.nome << "!\n";
    cout << "Preço da aquisição: R$" << a.salario + b.salario << "!\n";
}
```

Acesso ao Registro

- Saída do programa:

Contratações para o próximo ano: Bebeto e Romário!
Preço da aquisição: R\$500000!

- Cada **membro é tratado como uma variável** do tipo definido na declaração do registro

```
struct jogador          jogador bebeto;
{
    char nome[20];       bebeto          // tipo jogador
    float salario;      bebeto.nome     // tipo char [20]
    unsigned gols;       bebeto.salario  // tipo float
                        bebeto.gols      // tipo unsigned int
};                       bebeto.nome[0]  // tipo char
```

Definição de Tipo

- Tipos definidos através de registros **se comportam de forma semelhante aos tipos básicos** da linguagem C++
 - Registros podem ser passados **como argumentos de funções**

```
void exibir(jogador);
```

```
int main()
{
    jogador bebeto = {"Bebeto", 200'000, 600};
    exibir(bebeto);
    ...
}
```

Definição de Tipo

- Tipos definidos através de registros **se comportam de forma semelhante aos tipos básicos** da linguagem C++
 - Um registro pode ser um tipo de **retorno de uma função**

```
jogador ler();
```

```
int main()  
{  
    jogador novato;  
    novato = ler();  
    ...  
}
```

Definição de Tipo

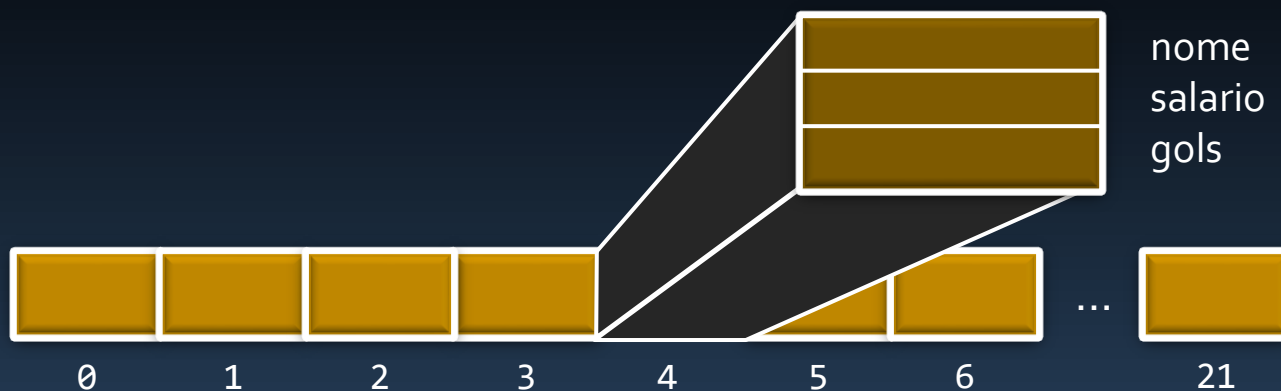
- Tipos definidos através de registros **se comportam de forma semelhante aos tipos básicos** da linguagem C++
 - Um registro pode ser atribuído a outro de mesmo tipo usando o operador de **atribuição**

```
int main()
{
    jogador beбето = {"Bebeto", 200'000, 600};
    jogador romario;
    romario = beбето;
    ...
}
```


Vetores de Registros

- Pelo registro ser semelhante a um tipo básico, podemos criar **vetores de registros**

```
struct jogador          jogador equipe[22];          // cria vetor de 22 jogadores
{
    char nome[20];       cin >> equipe[0].nome;       // lê nome, salário e
    float salario;      cin >> equipe[0].salario;     // gols do primeiro
    unsigned gols;      cin >> equipe[0].gols;       // jogador do time
};
```



Vetores de Registros

```
#include <iostream>
using namespace std;

struct jogador
{
    char nome[20];
    float salario;
    unsigned gols;
};

int main()
{
    jogador equipe[22] =
    {
        {"Bebeto", 200'000, 600},
        {"Romário", 300'000, 800}
    };

    cout << "Contrações para o próximo ano: " << equipe[0].nome << " e " << equipe[1].nome << "!\n";
    cout << "Preço da aquisição: R$" << equipe[0].salario + equipe[1].salario << "!\n";
}
```

Vetores de Registros

- Saída do programa:

Contrações para o próximo ano: Bebeto e Romário!
Preço da aquisição: R\$500000!

- Como equipe é um **vetor de jogador**, equipe[0] é um jogador

```
struct jogador          jogador potiguar[22];
{
    char nome[20];       potiguar           // tipo jogador[22]
    float salario;      potiguar.nome        // inválido
    unsigned gols;      potiguar[0].nome     // tipo char[20]
};                      potiguar[0].nome[0]  // tipo char
                       potiguar[0].salario   // tipo float
```

Tipos Sem Nome

- Pode-se combinar a declaração com a criação de variáveis:

```
struct jogador
{
    char nome[20];
    float salario;
}
maradona, zidane; // criar uma variável é opcional
```

- Pode-se também criar um registro sem nome

```
struct
{
    char nome[20];
    float salario;
}
pele; // criar uma variável é obrigatório
```

Tipos Sem Nome

```
#include <iostream>
using namespace std;

int main()
{
    struct
    {
        int x;
        int y;
    }
    ponto;

    cout << "Entre com as coordenadas do ponto: \n";
    cin >> ponto.x;
    cin >> ponto.y;

    cout << "O ponto se encontra na posição ("
        << ponto.x
        << ", "
        << ponto.y
        << ")\n";
}
```

Resumo

- **Registros** são tipos compostos de dados
 - Agrupam um **conjunto de informações**
 - Aceitam **diferentes tipos de dados**
 - Permitem a criação de **novos tipos**
 - Funcionam como um **tipo básico** de dado para:
 - Passagem de parâmetros
 - Retorno de valores
 - Atribuição de valor