Tipos de Dados Definidos pelo Usuário

- Nesta aula veremos o conceito de tipos de dados definidos pelo usuário: **registros** e **enumerações**
- Tipos enumerados definem e limitam os valores que uma variável pode assumir
- Um registro permite agrupar várias variáveis (campos) em uma única variável
- Veremos também como definir um sinônimo para um tipo de dados através do comando **typedef**
- Além disso veremos uma breve introdução ao tipo de dado **ponteiro**

José Augusto Baranauskas
Departamento de Física e Matemática – FFCLRP-USP
Sala 228 – Bloco P2 – Fone (16) 602-4439
E-mail: augusto@ffclrp.usp.br
URL: http://fmrp.usp.br/augusto

Enumerações

- Numa enumeração informamos ao compilador quais valores uma determinada variável pode assumir
- Definição:
  - **enum** nome_enumeração {*valor1, valor2, ..., valorN*};
- Declaração de Variáveis
  - nome_enumeração *variável1, variável2, ..., variávelN*;
- Atribuição
  - variávelP = valorK;
  - variávelQ = variávelP;
  - onde
    - variávelP, variávelQ é uma das variáveis *variável1, variável2, ..., variávelN*
    - valorK é um dos valores *valor1, valor2, ..., valorN*

Enumerações: Exemplos

- Definindo novos tipos de dados enumerados:
  - enum semana {seg, ter, qua, qui, sex, sab, dom};
  - enum cor {amarelo, azul, verde, branco};
- Declarando variáveis dos tipos enumerados:
  - semana s,k;
  - cor c1,c2;
- Atribuindo valores:

s = seg;	c1 = amarelo;
k = dom;	c2 = verde;

Exemplo

```
#include <iostream>
using namespace std;
enum semana {seg, ter, qua, qui, sex, sab, dom};
int main()
{ semana d1,d2;

  d1 = seg;
  d2 = sex;
  if(d1 == d2)
    cout << "Dias iguais\n";
  else
    cout << "Dias diferentes\n";
  return 0;
}
```


Exemplo

```
#include <iostream>
using namespace std;
enum semana {seg, ter, qua, qui, sex, sab, dom};
int main()
{ semana d1,d2;

  d1 = seg;
  d2 = sex;
  if(d1 == d2)
    cout << "Dias iguais\n";
  else
    cout << "Dias diferentes\n";
  return 0;
}
```

A definição de um novo tipo de dado através do comando "enum" **não** aloca/ocupa espaço na memória do computador

Exemplo

```
#include <iostream>
using namespace std;
enum semana {seg, ter, qua, qui, sex, sab, dom};
int main()
{ semana d1,d2;

  d1 = seg;
  d2 = sex;
  if(d1 == d2)
    cout << "Dias iguais\n";
  else
    cout << "Dias diferentes\n";
  return 0;
}
```

A declaração de variáveis de um novo tipo de dado (definido através do comando "enum") aloca/ocupa espaço na memória do computador para as variáveis declaradas

Enumerações

- No caso de enumerações, o compilador associa à lista de valores um número inteiro
- Assim, as variáveis enumeradas são do tipo **int**
- Ao primeiro da lista, é associado o número zero, o segundo ao número 1 e assim por diante
- É possível alterar a ordem de associação, atribuindo-se o valor numérico correspondente para cada valor enumerado

Exemplo

```
#include <iostream>
using namespace std;
enum semana {seg, ter, qua, qui, sex, sab, dom};
int main()
{ semana d1,d2;
  d1=seg;
  d2=sex;
  cout << "d1=" << d1
       << ", d2=" << d2 << endl;
  return 0;
}
```

d1=0, d2=4

Exemplo

```
#include <iostream>
using namespace std;
enum  semana {seg=1,ter,qua,qui,sex,sab,dom};
int main()
{ semana d1,d2;
  d1=seg;
  d2=sex;
  cout << "d1=" << d1
       << ", d2=" << d2 << endl;
  return 0;
}
```

d1=1, d2=5

Registros

- Um registro agrupa dados não homogêneos (campos) numa única variável, formando um novo tipo de dado
- Definição:

```
struct nome_registro
{ tipo_1 nome_1;
  tipo_2 nome_2;
  ...
  tipo_n nome_n;
};
```

- Declaração de Variáveis:
  - nome_registro *variável1*, *variável2*, ..., *variávelN*;
- Atribuição
  - variávelQ = variávelP;
  - variávelP.nome_i = valor_i;
  - onde
    - variávelP, variávelQ é uma das variáveis *variável1*, *variável2*, ..., *variávelN*
    - nome_i é um dos nomes (campos) nome_1, nome_2, ..., nome_n
    - valor_i é um dos valores permitidos para o tipo tipo_i

Exemplo: Definição

```
struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil; // estado civil
  float  salario;
};
```

A definição de um novo tipo de dado através do comando "struct" **não** aloca/ocupa espaço na memória do computador

Exemplo: Declaração de Variáveis

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};

int main()
{ funcionario f,g;
  ...
}
```

A declaração de variáveis de um novo tipo de dado (definido através do comando "struct") aloca/ocupa espaço na memória do computador para as variáveis declaradas

Exemplo: Declaração de Variáveis

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};

int main()
{ funcionario f,g;
  ...
}
```

f

matricula	nome	cargo	escolaridade
sexo	**local**	**ecivil**	**salario**

g

matricula	nome	cargo	escolaridade
sexo	**local**	**ecivil**	**salario**

Exemplo: Atribuição de Valores

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};

int main()
{ funcionario f,g;
  f.matricula = 145;

  return 0;
}
```

f

matricula	nome	cargo	escolaridade
145			
sexo	**local**	**ecivil**	**salario**

g

matricula	nome	cargo	escolaridade
sexo	**local**	**ecivil**	**salario**

Exemplo: Atribuição de Valores

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};

int main()
{ funcionario f,g;
  f.matricula = 145;
  f.nome = "Maria Mota";

  return 0;
}
```

f

matricula	nome	cargo	escolaridade
145	Maria Mota		
sexo	**local**	**ecivil**	**salario**

g

matricula	nome	cargo	escolaridade
sexo	**local**	**ecivil**	**salario**

Exemplo: Atribuição de Valores

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};

int main()
{ funcionario f,g;
  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";

  return 0;
}
```

f

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	
sexo	**local**	**ecivil**	**salario**

g

matricula	nome	cargo	escolaridade
sexo	**local**	**ecivil**	**salario**

Exemplo: Atribuição de Valores

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};

int main()
{ funcionario f,g;
  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;

  return 0;
}
```

f

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	3
sexo	**local**	**ecivil**	**salario**

g

matricula	nome	cargo	escolaridade
sexo	**local**	**ecivil**	**salario**

Exemplo: Atribuição de Valores

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};

int main()
{ funcionario f,g;
  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';

  return 0;
}
```

f

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	3
sexo	**local**	**ecivil**	**salario**
f			

g

matricula	nome	cargo	escolaridade
sexo	**local**	**ecivil**	**salario**

Exemplo: Atribuição de Valores

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};
```

```
int main()
{ funcionario f,g;
  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";

  return 0;
}
```

f

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	3
sexo	local	ecivil	salario
f	Vendas		

g

matricula	nome	cargo	escolaridade
sexo	local	ecivil	salario

Exemplo: Atribuição de Valores

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};
```

```
int main()
{ funcionario f,g;
  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";
  f.salario = 5200.00;

  return 0;
}
```

f

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	3
sexo	local	ecivil	salario
f	Vendas		5200.00

g

matricula	nome	cargo	escolaridade
sexo	local	ecivil	salario

Exemplo: Atribuição de Valores

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};
```

```
int main()
{ funcionario f,g;
  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";
  f.salario = 5200.00;
  f.ecivil = 's';
  return 0;
}
```

f

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	3
sexo	local	ecivil	salario
f	Vendas	s	5200.00

g

matricula	nome	cargo	escolaridade
sexo	local	ecivil	salario

Atribuição

- Podemos atribuir duas estruturas que sejam do *mesmo* tipo
- Neste caso, o compilador irá copiar uma estrutura, campo por campo, na outra
- Note que **isto é diferente do que acontece em vetores**, nos quais, para fazer a cópia dos elementos de um vetor em outro, deve-se copiar elemento por elemento de um vetor para o outro vetor

Exemplo: Atribuição de Variáveis

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};
```

```
int main()
{ funcionario f,g;
  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";
  f.salario = 5200.00;
  f.ecivil = 's';

  return 0;
}
```

f

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	3
sexo	local	ecivil	salario
f	Vendas	s	5200.00

g

matricula	nome	cargo	escolaridade
sexo	local	ecivil	salario

Exemplo: Atribuição de Variáveis

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};
```

```
int main()
{ funcionario f,g;
  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";
  f.salario = 5200.00;
  f.ecivil = 's';

  // copiar registro f para g
  g = f;
  return 0;
}
```

f

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	3
sexo	local	ecivil	salario
f	Vendas	s	5200.00

g

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	3
sexo	local	ecivil	salario
f	Vendas	s	5200.00

Passagem de Parâmetros

- Um registro em C++ é considerado como um tipo elementar de dados, por exemplo, do tipo **int** ou do tipo **float**
- Assim, é necessário informar ao compilador qual o tipo de passagem de parâmetros desejado: por variável ou por valor
- A sintaxe de passar registros como parâmetros é a mesma que para tipos elementares de dados

Exemplo: Passagem de Parâmetros

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};
// passagem por valor
void p1(funcionario x)
{ x.salario=1000;
}
// passagem por variável
void p2(funcionario &x)
{ x.salario=1000;
}
```

```
int main()
{ funcionario f,g;
  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";
  f.ecivil = 's';
  f.salario = 5200.00;

  // copiar registro f para g
  g = f;
  p1(g);
  cout << "f.salario="   << f.salario
       << ", g.salario=" << g.salario;
  p2(g);
  cout << "f.salario="   << f.salario
       << ", g.salario=" << g.salario;

  return 0;
}
```

```
f.salario=5200.00, g.salario=5200.00
f.salario=5200.00, g.salario=1000.00
```

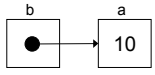

Ponteiros

- **Ponteiros** são variáveis que contêm endereços de memória como valores
- Normalmente, uma variável contém diretamente um valor específico
- Um ponteiro, por outro lado, contém um endereço de uma variável que contém um valor específico
  - uma variável referencia **diretamente** um valor
  - um ponteiro referencia **indiretamente** um valor

Ponteiros

- Referência direta a uma variável
  - "**a**" referencia diretamente uma variável cujo valor é 10

- Referência indireta a uma variável
  - "b" referencia indiretamente uma variável cujo valor é 10



Ponteiros

- Ponteiros, como quaisquer outras variáveis devem ser declarados antes do uso
- A declaração
  - int *b, a;
- Declara
  - a variável **a** do tipo **int**
  - a variável **b** do tipo **int *** (isto é, um ponteiro para um inteiro) e é lido como "**b** é um ponteiro para **int**"
  - O * apenas se aplica à declaração de **b**

Ponteiros

- Cada variável sendo declarada como um ponteiro deve ser precedida por um asterisco
  - float *xptr, *yptr;
  - indica que **xptr** e **yptr** são ambos ponteiros para valores **float**
- Quando * é usado desta maneira na **declaração**, ele indica que a variável sendo declarada é um ponteiro
- Ponteiros podem ser declarados para apontar para qualquer tipo de dados (primitivos ou definidos pelo usuário)

Ponteiros

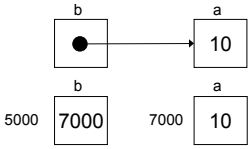
- Como toda variável, ponteiros devem ser inicializados quando declarados ou através de um comando de atribuição
- Um ponteiro pode ser inicializado com **0**, **NULL** ou um endereço
- Um ponteiro com **0** ou **NULL** aponta para nada (algumas vezes, dizemos que o ponteiro está aterrado, ou que aponta para o terra)
- **NULL** é uma constante simbólica definida no arquivo de cabeçalho (header) **<iostream>**
- Inicializar um ponteiro com **NULL** é equivalente a inicializar um ponteiro com **0**
- O valor **0** é o único inteiro que pode ser atribuído diretamente a um ponteiro sem necessitar conversão para tipo ponteiro primeiro

Ponteiros: Operadores

- **&**: operador de endereço = operador unário que retorna o endereço do seu operando
  - int a=10;
  - int *b;

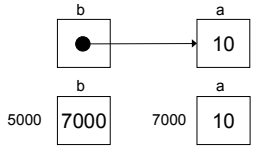
Ponteiros: Operadores

- **&**: operador de endereço = operador unário que retorna o endereço do seu operando
  - int a=10;
  - int *b;
- O comando
  - b = &a;
- Atribui o endereço da variável **a** para o ponteiro **b**
- Dizemos nesse caso que "**b** aponta para **a**"



Ponteiros: Operadores

- ***: operador de indireção ou derreferenciação = operador unário que retorna um sinônimo, ou apelido do seu operando
  - int a=10;
  - int *b;
  - b = &a;
- O comando
  - cout << *b << endl;
- Escreve o valor da variável apontada por **b**, ou seja, escreve o valor da variável **a** da mesma forma que o comando
  - cout << a << endl;



Exemplo 1

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  *b = 11;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```

"a" é uma variável inteira. "b" aponta para um inteiro, ou seja, "b" é um ponteiro para um inteiro

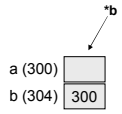
Exemplo 1

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  *b = 11;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```

a (300)
b (304)

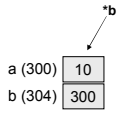
Exemplo 1

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  *b = 11;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```



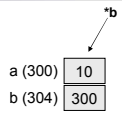
Exemplo 1

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  *b = 11;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```



Exemplo 1

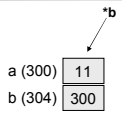
```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  *b = 11;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```



&a=300 a=10 &b=304 b=300 *b=10

Exemplo 1

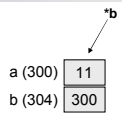
```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  *b = 11;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```



&a=300 a=10 &b=304 b=300 *b=10

Exemplo 1

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  *b = 11;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```



&a=300 a=10 &b=304 b=300 *b=10
&a=300 a=11 &b=304 b=300 *b=11

Exemplo 2

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  (*b)++;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```

Exemplo 2

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  (*b)++;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```

a (300)
b (304)

Exemplo 2

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  (*b)++;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```

*b
a (300)
b (304) 300

Exemplo 2

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  (*b)++;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```

*b
a (300) 10
b (304) 300

Exemplo 2

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  (*b)++;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```

*b
a (300) 10
b (304) 300

&a=300 a=10 &b=304 b=300 *b=10

Exemplo 2

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  (*b)++;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```

*b
a (300) 11
b (304) 300

&a=300 a=10 &b=304 b=300 *b=10

Exemplo 2

```
#include <iostream>
using namespace std;
int main()
{ int a,*b;

  b = &a;
  a = 10;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  (*b)++;
  cout << "&a=" << &a << " a=" << a << " &b=" << &b
       << " b=" << b << " *b=" << *b << endl;
  return 0;
}
```

*b
a (300) 11
b (304) 300

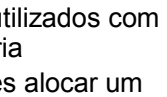
&a=300 a=10 &b=304 b=300 *b=10
&a=300 a=11 &b=304 b=300 *b=11

Alocação Dinâmica de Memória

- Ponteiros normalmente são utilizados com alocação dinâmica de memória
- Para tanto, é necessário antes alocar um novo espaço na memória antes de utilizar, liberando-o ao término do uso
- Em C++ a alocação é efetuada através do operador **new** e a liberação através de **delete**
- Sintaxe (p é um ponteiro do tipo T):
  - p = **new** T;
  - **delete** p;

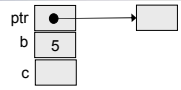
Exemplo 3

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5,c;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  c = *ptr + b;
  cout << *ptr << "+" << b
       << "=" << c << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```



Exemplo 3

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5,c;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  c = *ptr + b;
  cout << *ptr << "+" << b
       << "=" << c << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```

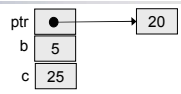


Exemplo 3

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5,c;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  c = *ptr + b;
  cout << *ptr << "+" << b
       << "=" << c << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```


Exemplo 3

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5,c;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  c = *ptr + b;
  cout << *ptr << "+" << b
       << "=" << c << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```



Exemplo 3

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5,c;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  c = *ptr + b;
  cout << *ptr << "+" << b
       << "=" << c << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```


20+5=25

Exemplo 3

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5,c;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  c = *ptr + b;
  cout << *ptr << "+" << b
       << "=" << c << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```

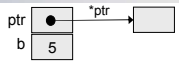
20+5=25

Exemplo 4

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  ptr = new int;
  *ptr = 50;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```

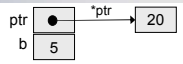
Exemplo 4

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  ptr = new int;
  *ptr = 50;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```



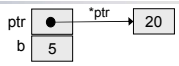
Exemplo 4

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  ptr = new int;
  *ptr = 50;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```



Exemplo 4

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  ptr = new int;
  *ptr = 50;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```

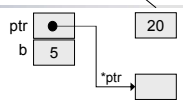


20+5=25

Exemplo 4

Note que essa posição de memória ficou "perdida" pois não há mais como chegar até ela (foi perdido o ponteiro para ela)

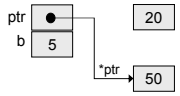
```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  ptr = new int;
  *ptr = 50;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```



20+5=25

Exemplo 4

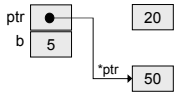
```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  ptr = new int;
  *ptr = 50;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```



20+5=25

Exemplo 4

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  ptr = new int;
  *ptr = 50;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```



20+5=25
50+5=55

Exemplo 4

```
#include <iostream>
using namespace std;
int main()
{ int *ptr,b=5;

  // alocar novo inteiro apontado por ptr
  ptr = new int;
  *ptr = 20;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  ptr = new int;
  *ptr = 50;
  cout << *ptr << "+" << b
       << "=" << b + *ptr << endl;
  delete ptr; // liberar inteiro alocado
  return 0;
}
```

20+5=25
50+5=55

Alocação Dinâmica de Memória

- Podemos alocar dinamicamente estruturas definidas pelo usuário, por exemplo, registros
- Um campo x de um registro apontados pelo ponteiro p pode ser acessado usando as notações equivalentes:
  - `(*p).x`
  - `p->x`

Exemplo 5

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};
```

	matricula	nome	cargo	escolaridade
f	145	Maria Mota	Vendedora	3
	sexo	local	ecivil	salario
	f	Vendas	s	5200.00

g ●

```
int main()
{ funcionario f,*g;

  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";
  f.salario = 5200.00;
  f.ecivil = 's';

  g = new funcionario;
  *g = f;
  cout << "f.salario="<<f.salario
       << ", (*g).salario="
       << (*g).salario
       << ", g->salario="
       << g->salario << endl;
  delete g;

  return 0;
}
```


Exemplo 5

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};
```

	matricula	nome	cargo	escolaridade
f	145	Maria Mota	Vendedora	3
	sexo	local	ecivil	salario
	f	Vendas	s	5200.00

g ● *g

matricula	nome	cargo	escolaridade
sexo	local	ecivil	salario

```
int main()
{ funcionario f,*g;

  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";
  f.salario = 5200.00;
  f.ecivil = 's';

  g = new funcionario;
  *g = f;
  cout << "f.salario="<<f.salario
       << ", (*g).salario="
       << (*g).salario
       << ", g->salario="
       << g->salario << endl;
  delete g;

  return 0;
}
```

Exemplo 5

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};

int main()
{ funcionario f,*g;

  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";
  f.salario = 5200.00;
  f.ecivil = 's';

  g = new funcionario;
  *g = f;
  cout << "f.salario="<<f.salario
       << ", (*g).salario="
       << (*g).salario
       << ", g->salario="
       << g->salario << endl;
  delete g;

  return 0;
}
```

f	matricula	nome	cargo	escolaridade
	145	Maria Mota	Vendedora	3
	sexo	local	ecivil	salario
	f	Vendas	s	5200.00

g → *g

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	3
sexo	local	ecivil	salario
f	Vendas	s	5200.00

Exemplo 5

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};

int main()
{ funcionario f,*g;

  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";
  f.salario = 5200.00;
  f.ecivil = 's';

  g = new funcionario;
  *g = f;
  cout << "f.salario="<<f.salario
       << ", (*g).salario="
       << (*g).salario
       << ", g->salario="
       << g->salario << endl;
  delete g;

  return 0;
}
```

f	matricula	nome	cargo	escolaridade
	145	Maria Mota	Vendedora	3
	sexo	local	ecivil	salario
	f	Vendas	s	5200.00

g → *g

matricula	nome	cargo	escolaridade
145	Maria Mota	Vendedora	3
sexo	local	ecivil	salario
f	Vendas	s	5200.00

f.salario=5200, (*g).salario=5200, g->salario=5200

Exemplo 5

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

struct funcionario
{ int    matricula;
  string nome;
  string cargo;
  int    escolaridade;
  char   sexo;
  string local;
  char   ecivil;
  float  salario;
};

int main()
{ funcionario f,*g;

  f.matricula = 145;
  f.nome = "Maria Mota";
  f.cargo = "Vendedora";
  f.escolaridade = 3;
  f.sexo = 'f';
  f.local = "Vendas";
  f.salario = 5200.00;
  f.ecivil = 's';

  g = new funcionario;
  *g = f;
  cout << "f.salario="<<f.salario
       << ", (*g).salario="
       << (*g).salario
       << ", g->salario="
       << g->salario << endl;
  delete g;

  return 0;
}
```

f	matricula	nome	cargo	escolaridade
	145	Maria Mota	Vendedora	3
	sexo	local	ecivil	salario
	f	Vendas	s	5200.00

g

f.salario=5200, (*g).salario=5200, g->salario=5200.00

Definição de Novos Tipos de Dados

- O comando **typedef** fornece um mecanismo para criar **sinônimos** (ou apelidos) para tipos de dados previamente definidos
- Criar um novo nome utilizando **typedef** não cria um novo tipo de dados; **typedef** apenas cria um novo nome que pode ser utilizado como um sinônimo
- Definição:
  - **typedef** tipo_existente tipo_novo;

Exemplos

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

typedef int inteiro;
const inteiro M=50;
typedef inteiro vetor[M];

int main()
{ inteiro i;
  vetor x,y;

  for(i=0;i<M;i++)
    cin >> x[i] >> y[i];
  for(i=0;i<M;i++)
    cout << x[i] << "\t"
         << y[i] << endl;
  return 0;
}
```

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

typedef int *pint;
// pint é um ponteiro para int

int main()
{ int i;
  pint iptr1;
  int *iptr2;

  iptr1 = &i;
  iptr2 = &i;
  i = 5;
  cout << "i=" << i
       << " *iptr1=" << *iptr1
       << " *iptr2=" << *iptr2
       << endl;
  return 0;
}
```


Resumo

- Nesta aula vimos como é possível definir novos tipos de dados, ou seja, tipos de dados que são definidos pelo usuário
- Uma vez definido um novo tipo, ele tem comportamento *similar* aos tipos de dados primitivos (que são definidos pela linguagem), sendo possível declarar variáveis, ler, escrever, atribuir valores, etc às novas variáveis
- Um registro, que agrupa vários campos em uma única estrutura, juntamente com ponteiros é muito utilizado para definir estruturas de dados mais complexas, como listas lineares, árvores, grafos, etc
- Além disso, registros são muito utilizados quando se trabalha com arquivos, para armazenar e recuperar informações em disco