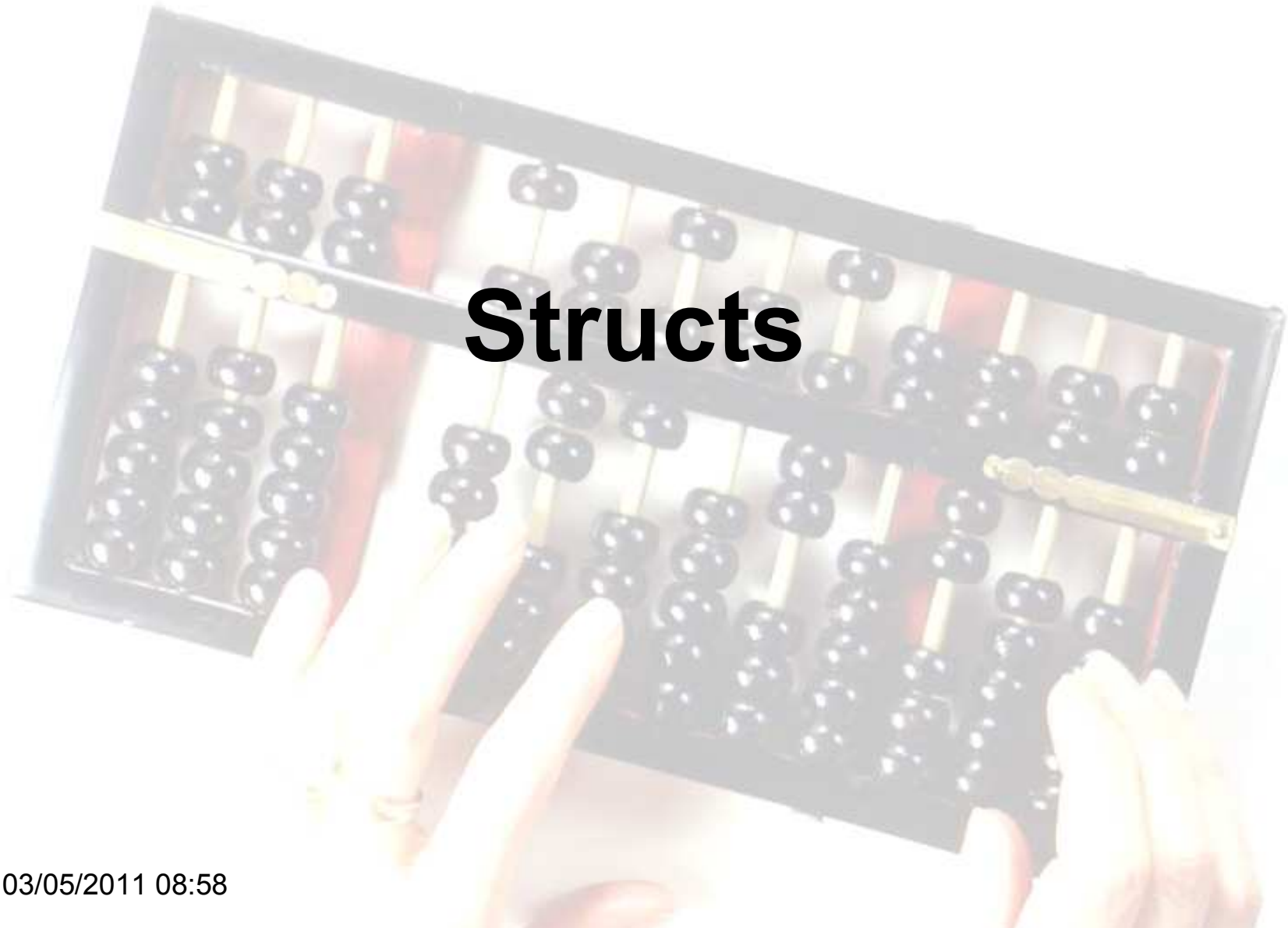
Curso de C

Estruturas: Structs e Unions

Estruturas

Roteiro:

- O que são estruturas compostas
- Declaração
- Atributos



Structs

Structs

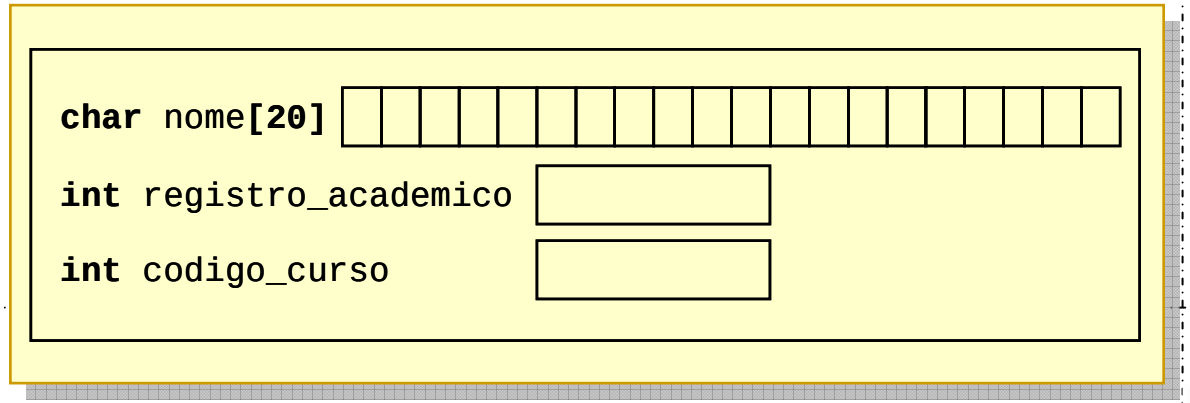
Motivação:

- Agrupar variáveis inter-relacionadas
- Organizar dados complexos
- Reduzir número de variáveis
- Usar um único nome de variável

Structs

Motivação:

- Agrupar variáveis inter-relacionadas
- Organizar dados complexos
- Reduzir número de variáveis
- Usar um único nome de variável



Structs

Declaração

Structs: declaração

Sintaxe:

```
struct {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variavel;
```

Structs: declaração

Sintaxe:

Tipo Estrutura

```
struct {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variavel;
```

Lista de atributos

Aloca e cria variável

Structs: declaração

Sintaxe:

Tipo Estrutura

```
struct {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variavel;
```

Lista de atributos

Aloca e cria variável

- *Declara estrutura*
- *Cria apenas a variável*
- *Só para esta declaração*

>

Structs

Exemplo 1:

```
struct {
    char nome[20];
    int registro_academico;
    int codigo_curso;
} aluno;
```

Structs

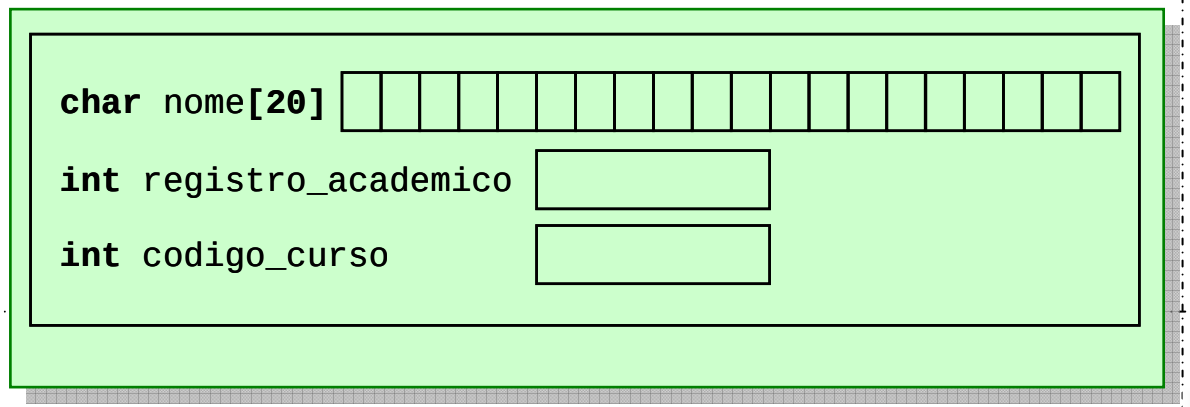
Exemplo 1:

Tipo Estrutura

```
struct {
    char nome[20];
    int registro_academico;
    int codigo_curso;
} aluno;
```

Lista de atributos

Criar variável (reserva memória)



Structs

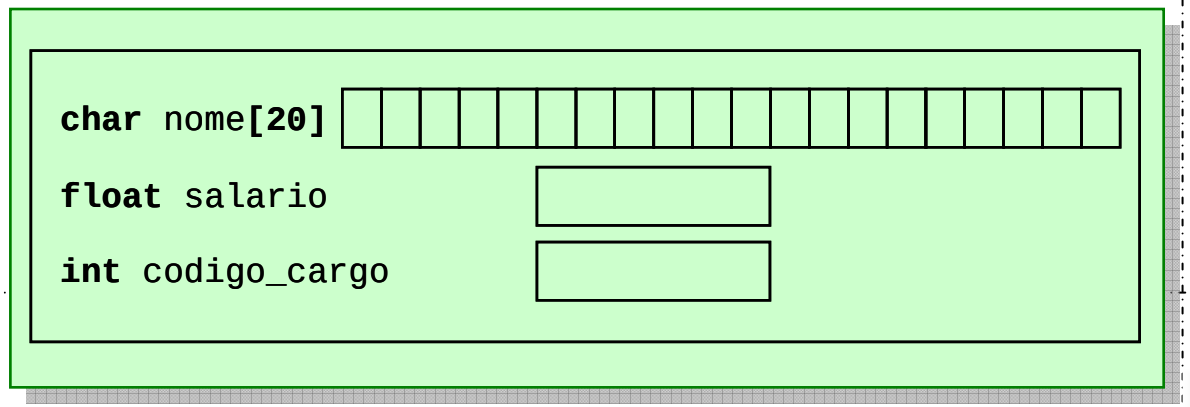
Exemplo 2:

Tipo Estrutura

```
struct {
    char nome[20];
    float salario;
    int codigo_cargo;
} funcionario;
```

Lista de atributos

Cria variável



Structs

Uso dos atributos

Structs: atributos

Acesso a atributos:

Acesso: `funcionario.salario`

Structs: atributos

Acesso a atributos:

Acesso:

```
funcionario.salario
```

Atribuição:

```
funcionario.salario = 1000.0f;
```

Structs: atributos

Acesso a atributos:

Acesso:

```
funcionario.salario
```

Atribuição:

```
funcionario.salario = 1000.0f;
```

Impressão:

```
printf("A variável: %s", funcionario.nome);
```

Structs: atributos

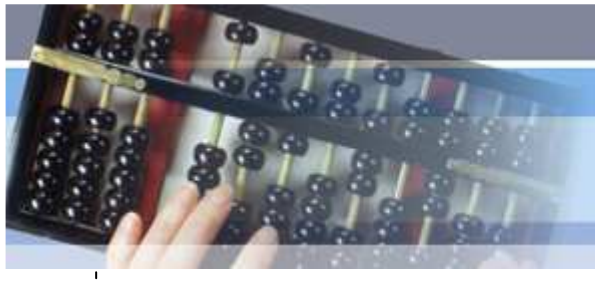
Acesso a atributos:

Acesso:
```
funcionario.salario
```

Atribuição:
```
funcionario.salario = 1000.0f;
```

Impressão:
```
printf("A variável: %s", funcionario.nome);
```

Leitura:
```
scanf("%f", &funcionario.salario);
```



Structs: atributos

Acesso a atributos:

Acesso:
```
funcionario.salario
```

Atribuição:
```
funcionario.salario = 1000.0f;
```

Impressão:
```
printf("A variável: %s", funcionario.nome);
```

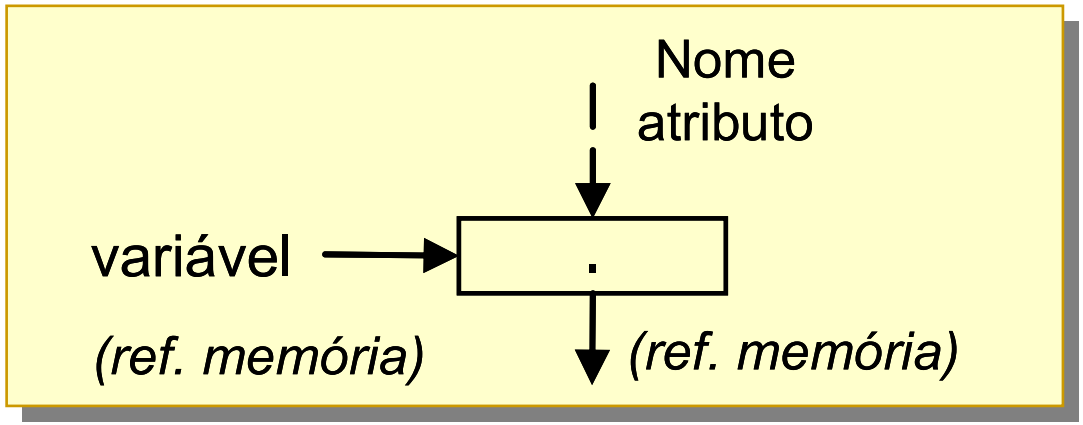
Leitura:
```
scanf("%f", &funcionario.salario);
```

Cópia:
```
funcionarioA = funcionarioB;
```

>

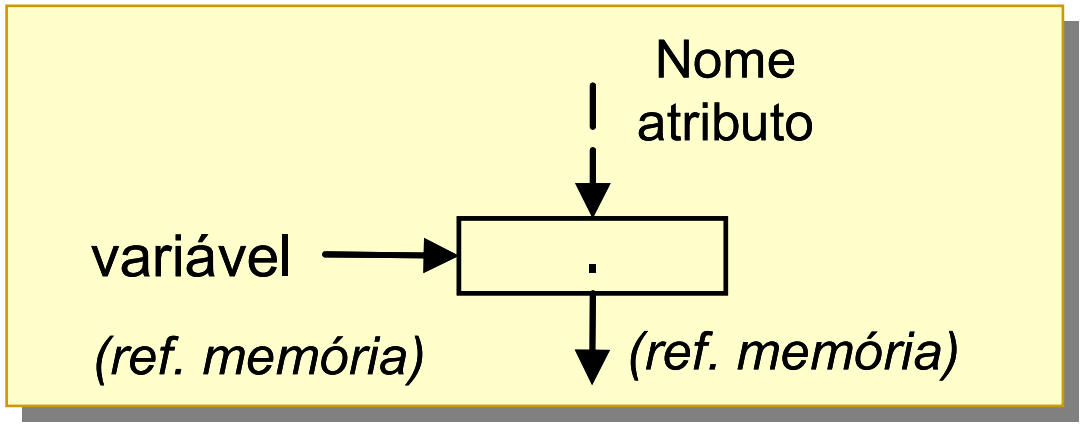
Structs: atributos

Operador de seleção de atributo:

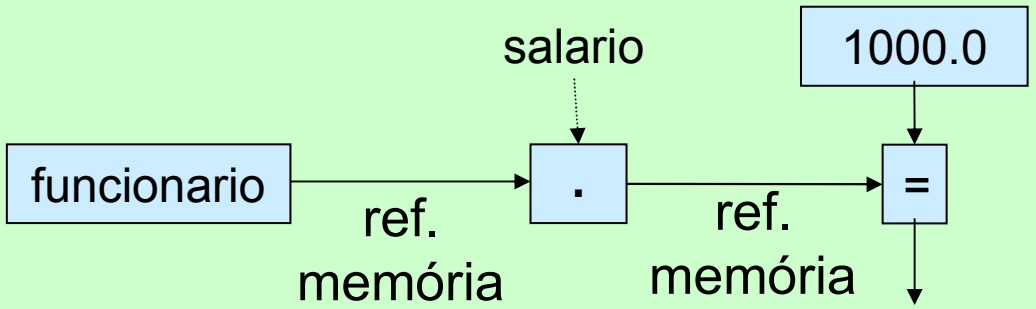


Structs: atributos

Operador de seleção de atributo:




```
funcionario.salario = 1000.0f;
```



>

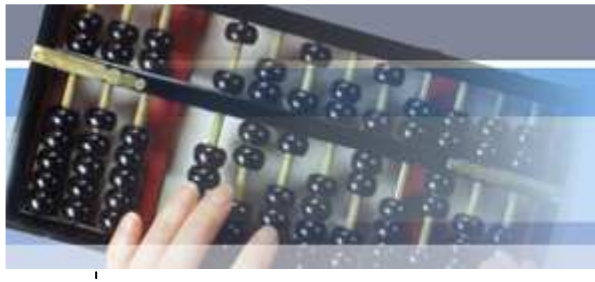
Structs

Declaração com definição de tipo

Structs: definição de tipo

Declaração identificada:

```
struct nome_estrutura {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variavel;
```



Structs: definição de tipo

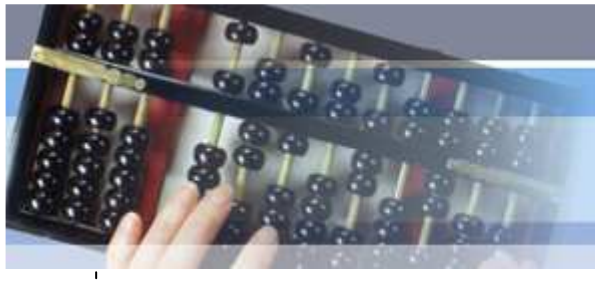
Declaração identificada:

Tipo Estrutura

```
struct nome_estrutura {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variavel;
```

Lista de atributos

Aloca e cria variável



Structs: definição de tipo

Declaração identificada:

Tipo Estrutura

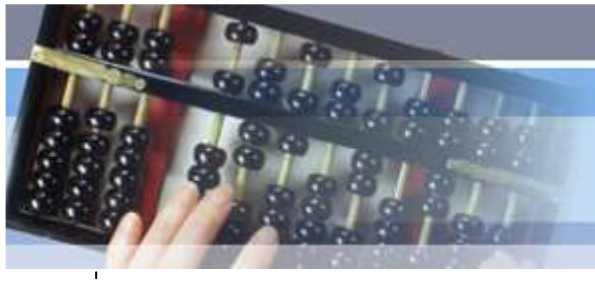
Nome da Estrutura

```
struct nome_estrutura {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variavel;
```

Lista de atributos

Aloca e cria variável

- *Declara estrutura*
- *Memoriza a declaração*
- *Novo tipo de dados*
- *CRIA a variável*



Structs: definição de tipo

Declaração identificada:

Tipo Estrutura

Nome da Estrutura

```
struct nome_estrutura {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
};
```

Lista de atributos

Não aloca nada

- *Declara estrutura*
- *Memoriza a declaração*
- *Novo tipo de dados*
- *NÃO cria a variável*

Structs: definição de tipo

Declaração identificada:

Declaração memorizada pelo compilador

```
struct nome_estrutura {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
};
```

Structs: definição de tipo

Declaração identificada:

Declaração memorizada pelo compilador

```
struct nome_estrutura {
        tipo1 atributo1;
        tipo2 atributo2;
        tipo3 atributo3;
        ...
};
```

Declaração abreviada de variável:

```
struct nome_estrutura variavel
```

Structs: definição de tipo

Declarar e criar variáveis:

```
int main(int argc, char *argv[]) {
      struct TipoAluno {
            char sobrenome[50];
            int registro_academico;
            int codigo_curso;
      };

      struct TipoAluno aluno1;
      struct TipoAluno aluno2;
      struct TipoAluno aluno_selecionado;
...
```


>

Structs: definição de tipo

Ler atributos:

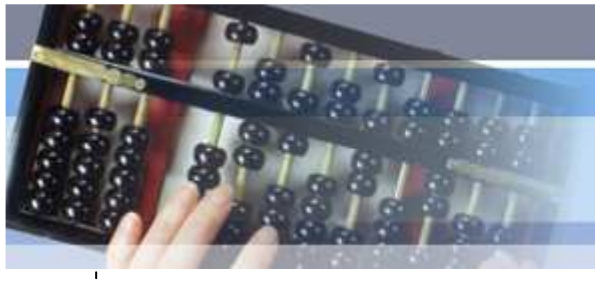
```
printf("Aluno 1: ");
scanf("%s", aluno1.sobrenome);
scanf("%d", &aluno1.registro_academico);
scanf("%d", &aluno1.codigo_curso);

printf("Aluno 2: ");
scanf("%s", aluno2.sobrenome);
scanf("%d", &aluno2.registro_academico);
scanf("%d", &aluno2.codigo_curso);
...
```

Structs: definição de tipo

Comparar atributos:

```
if (aluno1.registro_academico <
     aluno2.registro_academico) {
   aluno_selecionado = aluno1;
} else {
   aluno_selecionado = aluno2;
}
```



Structs: definição de tipo

Imprimir resultado:

```
  printf("Sobrenome: %s \n",
    aluno_selecionado.sobrenome);
  printf("RA:        %d \n",
    aluno_selecionado.registro_academico);
  printf("Curso:     %d \n",
    aluno_selecionado.codigo_curso);
  return 0;
}
```

Structs

Declarações aninhadas

Structs: aninhamento

Organizando o código:

```
struct {
    char nome[50];
    int codigo_curso;
    int dia_matricula;
    int mes_matricula;
    int ano_matricula;
    ...
} aluno;
```

Structs: aninhamento

Organizando o código:

```
struct {
    char nome[50];
    int codigo_curso;
    int dia_matricula;
    int mes_matricula;
    int ano_matricula;
    ...
} aluno;
```

```
struct {
    char nome[50];
    int codigo_curso;
    struct {
        int dia,mes,ano;
    } data_matricula;
    ...
} aluno;
```

Structs: aninhamento

Organizando o código:

```
struct {
   char nome[50];
   int codigo_curso;
   int dia_matricula;
   int mes_matricula;
   int ano_matricula;
   ...
} aluno;
```

```
struct {
   char nome[50];
   int codigo_curso;
   struct {
      int dia,mes,ano;
   } data_matricula;
   ...
} aluno;
```

```
aluno.data_matricula.dia = 2;
aluno.data_matricula.mes = 3;
aluno.data_matricula.ano = 1999;
```

Structs: aninhamento

Reaproveitando estruturas:

```
struct TipoAluno {
        char nome[50];
        int codigo_curso;
        int dia_matricula;
        int mes_matricula;
        int ano_matricula;
        ...
};
```

Structs: aninhamento

Reaproveitando estruturas:

```
struct TipoAluno {
        char nome[50];
        int codigo_curso;
        int dia_matricula;
        int mes_matricula;
        int ano_matricula;
        ...
};
```

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

Structs: aninhamento

Reaproveitando estruturas:

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

```
struct TipoAluno {
        char nome[50];
        int codigo_curso;
        struct TipoData data_matricula;
        ...
};
```

Structs: aninhamento

Reaproveitando estruturas:

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

```
struct TipoAluno {
        char nome[50];
        int codigo_curso;
        struct TipoData data_matricula;
        ...
};
```

```
aluno.data_matricula.dia = 2;
aluno.data_matricula.mes = 3;
aluno.data_matricula.ano = 1999;
```

>

Structs: aninhamento

Reaproveitando estruturas:

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

Structs: aninhamento

Reaproveitando estruturas:

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

```
struct TipoAluno {
        char nome[50];
        struct TipoData data_matricula;
        ...
};
```

Structs: aninhamento

Reaproveitando estruturas:

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

```
struct TipoAluno {
        char nome[50];
        struct TipoData data_matricula;
        ...
};
```

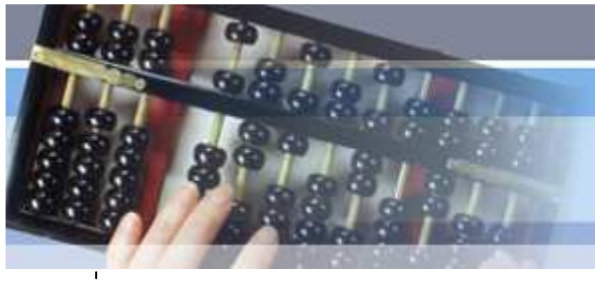
```
struct TipoFuncionario {
        char nome[50];
        struct TipoData data_contratacao;
        ...
};
```

>

Structs: aninhamento

Erro: **declaração cíclica**

```
struct TipoFuncionario {
      char nome[50];
      int codigo_cargo;
      struct TipoData data_contratacao;
      ...
      struct TipoFuncionario supervisor;
};
```



Structs: aninhamento

Erro: **declaração cíclica**

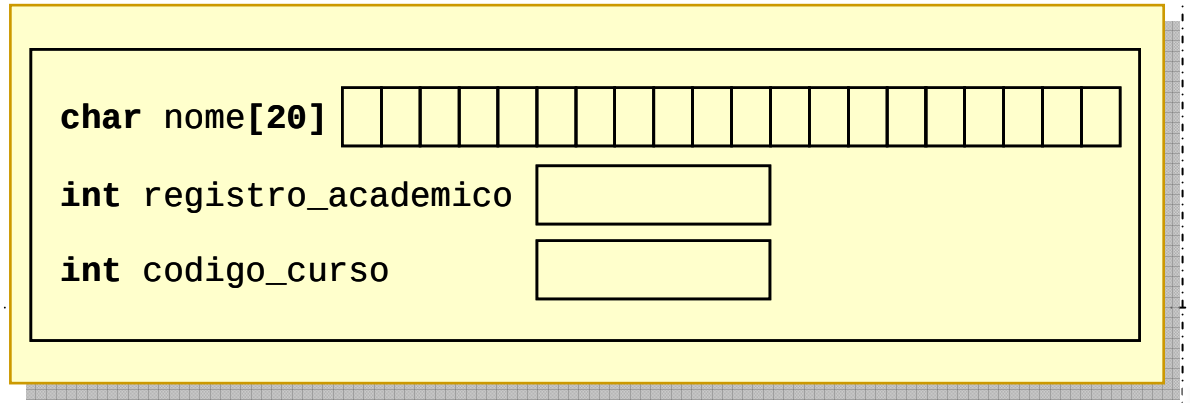
```
struct TipoFuncionario {
        char nome[50];
        int codigo_cargo;
        struct TipoData data_contratacao;
        ...
        struct TipoFuncionario supervisor;
};
```

Erro: Declaração cíclica!

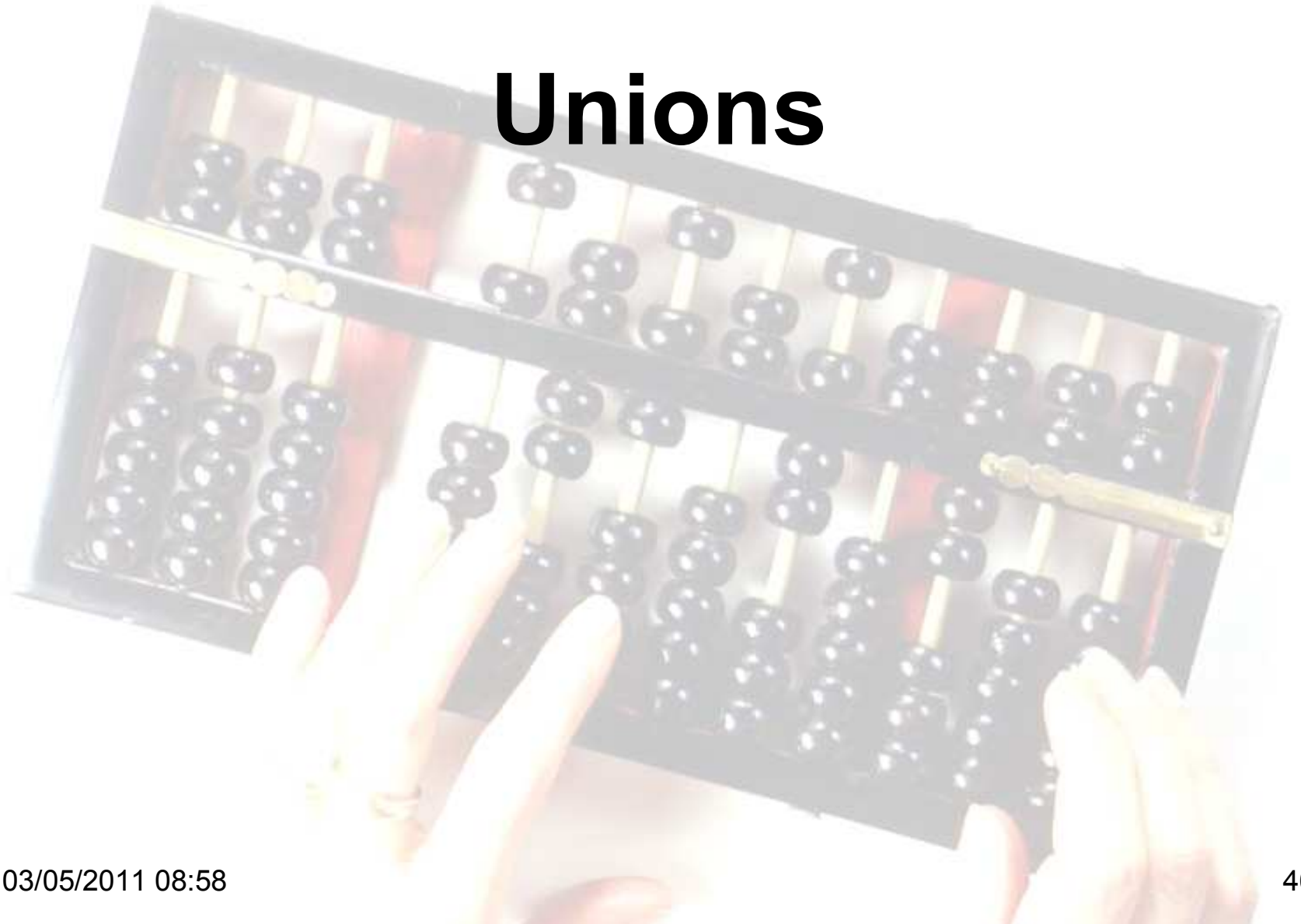
Structs

Conceitos:

- Coleção de variáveis
- Tipos diferentes
- Declaração com um único nome
- Operação simultânea sobre todos os dados
- Acesso por atributo



Unions



Unions

Motivação:

- Mesma idéia que de estruturas
- Economizando memória:
  - Só um dentre vários valores ocupa a memória
  - Programador é responsável por gerenciar qual valor é válido a cada instante

Unions

Motivação:

- Mesma idéia que de estruturas
- Economizando memória:
  - Só um dentre vários valores ocupa a memória
  - Programador é responsável por gerenciar qual valor é válido a cada instante

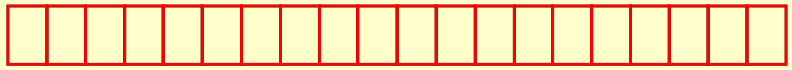
int cod_curso

Unions

Motivação:

- Mesma idéia que de estruturas
- Economizando memória:
  - Só um dentre vários valores ocupa a memória
  - Programador é responsável por gerenciar qual valor é válido a cada instante

char nome[20]

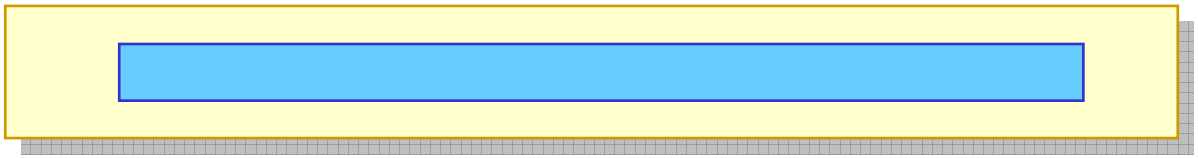


Unions

Motivação:

- Mesma idéia que de estruturas
- Economizando memória:
  - Só um dentre vários valores ocupa a memória
  - Programador é responsável por gerenciar qual valor é válido a cada instante

int reg_academico



Unions

Declaração

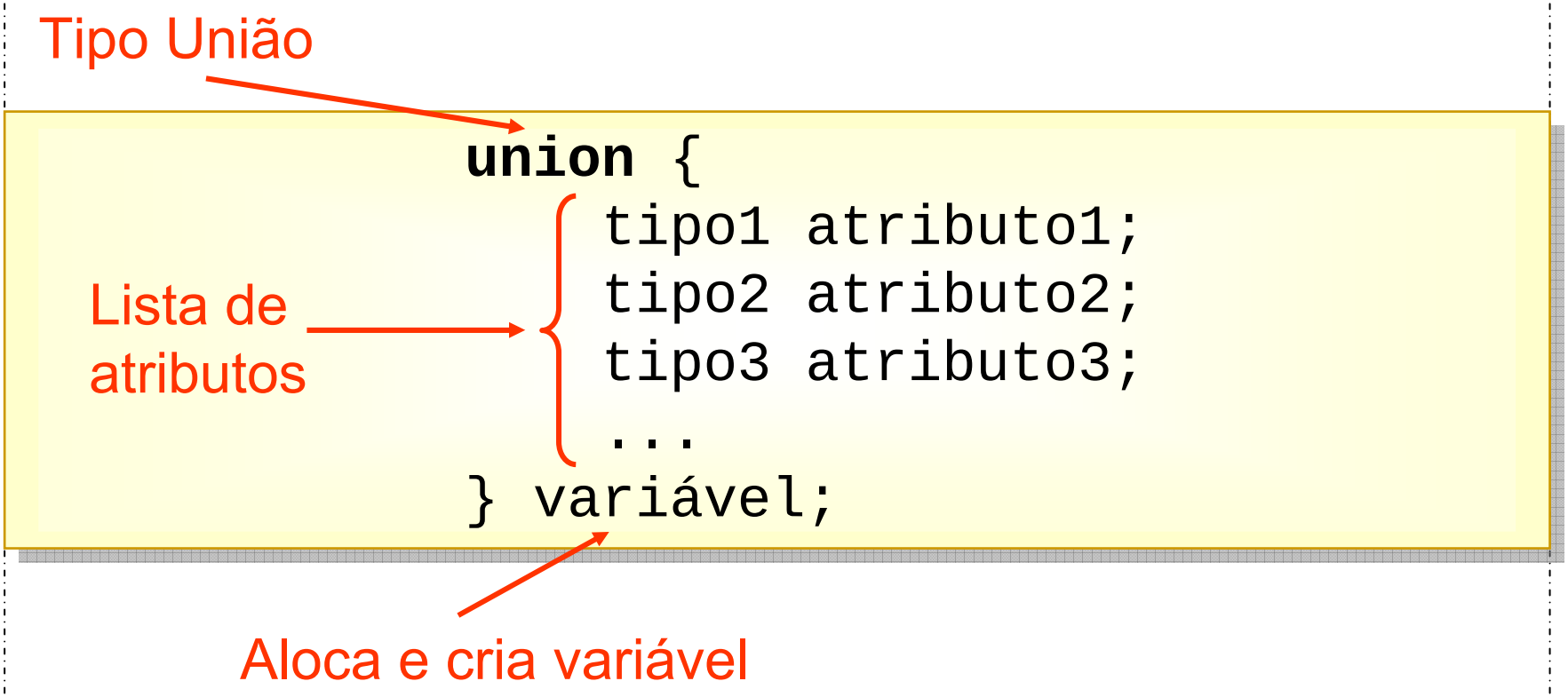
Unions: declaração

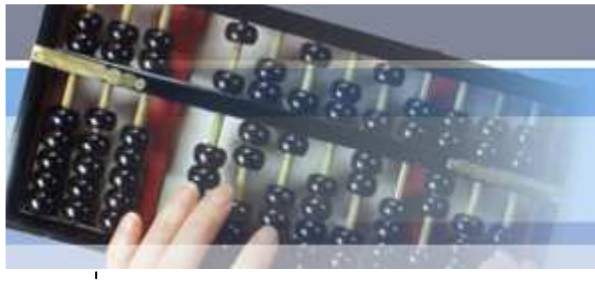
Sintaxe:

```
union {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variável;
```

Unions: declaração

Sintaxe:





Unions: declaração

Sintaxe:

Tipo União

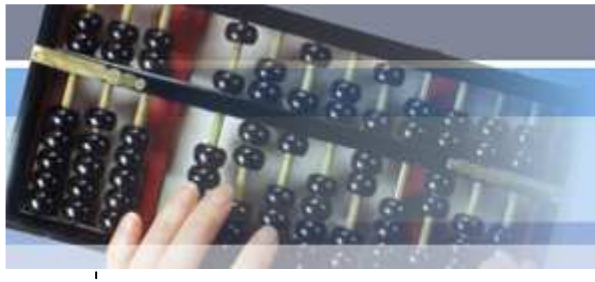
```
union {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variável;
```

Lista de atributos

Aloca e cria variável

- *Declara a union*
- *Cria apenas a variável*
- *Só para esta declaração*

>



Unions: exemplo

Exemplo:

Tipo Union

```
union {
    int inteiro;
    float frac;
} numflexivel;
```

Lista de possibilidades

Cria variável

Unions: exemplo

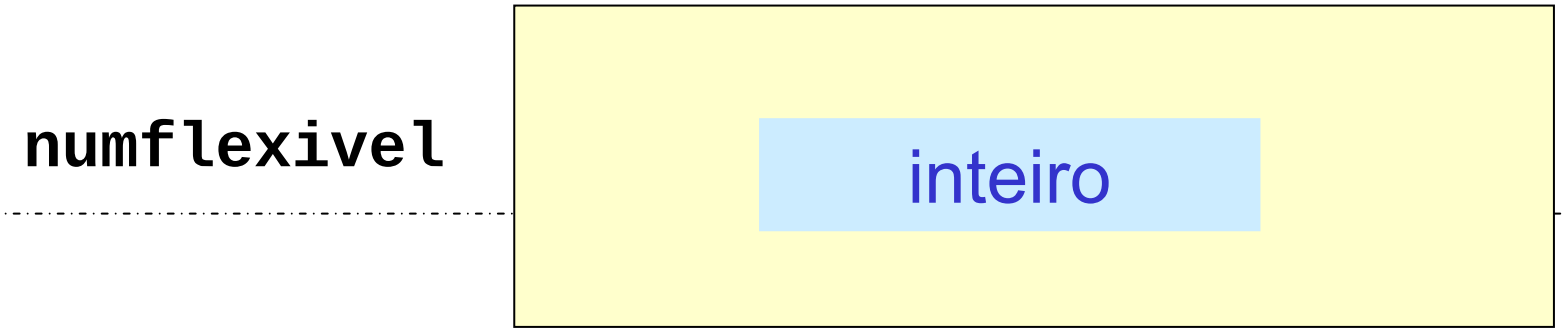
Exemplo:

Tipo Union

```
union {
    int inteiro;
    float frac;
} numflexivel;
```

Lista de possibilidades

Criar variável



Unions: exemplo

Exemplo:

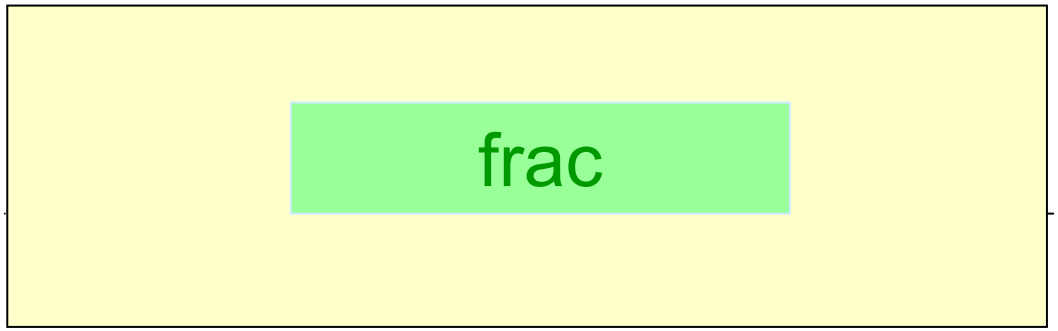
Tipo Union

```
union {
    int inteiro;
    float frac;
} numflexivel;
```

Lista de possibilidades

Criar variável

numflexivel



Unions

Atribuindo

Unions: atribuindo

```
union {
      int inteiro;
      float frac;
} numflexivel;
```

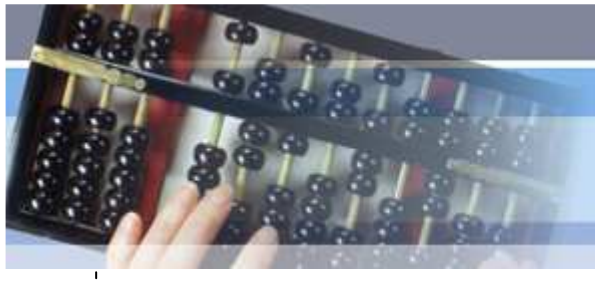
?????

Unions: atribuindo

```
union {
    int inteiro;
    float frac;
} numflexivel;
```

10

```
numflexivel.inteiro = 10;
```



Unions: atribuindo

```
union {
    int inteiro;
    float frac;
} numflexivel;
```

3.1415

```
numflexivel.inteiro = 10;
numflexivel.frac = 3.1415;
```

Unions: atribuindo

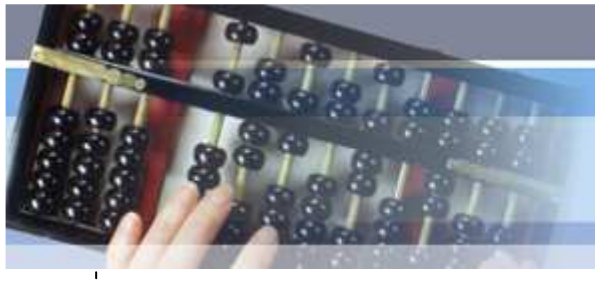
```
union {
    int inteiro;
    float frac;
} numflexivel;
```

```
-50
```

```
numflexivel.inteiro = 10;
numflexivel.frac = 3.1415;
numflexivel.inteiro = -50;
```

Unions

Declaração com definição de tipo



Unions: definição de tipo

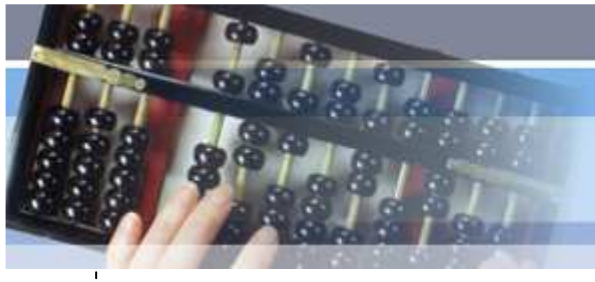
Declaração identificada:

Tipo Union

```
union nome_union {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variável;
```

Lista de possibilidsades

Aloca e cria variável



Unions: definição de tipo

Declaração identificada:

Nome da Union

Tipo Union

```
union nome_union {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variável;
```

Lista de possibilidsades

Aloca e cria variável

Unions: definição de tipo

Declaração identificada:

Tipo Union

Nome da Union

```
union nome_union {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variável;
```

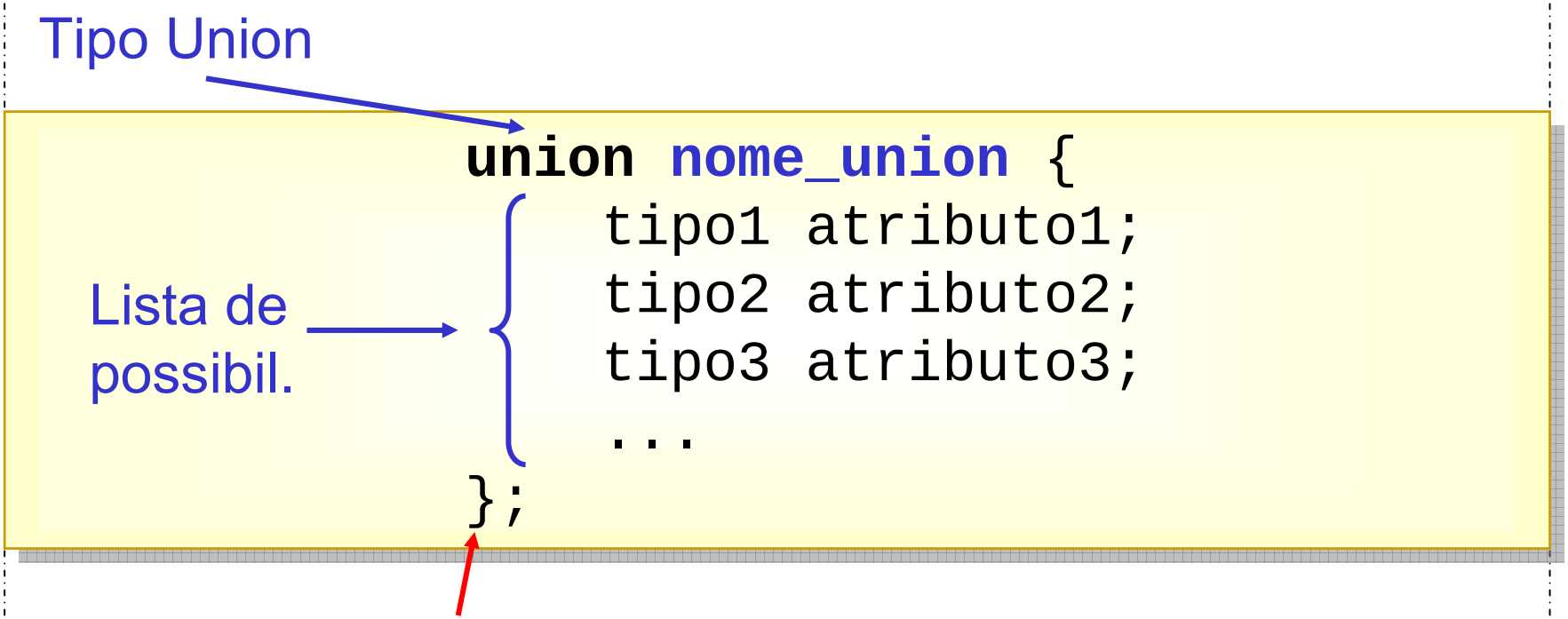
Lista de possibilidsades

Aloca e cria variável

- *Declara a union*
- *Memoriza a declaração*
- *Novo tipo de dados*
- *Cria a variável*

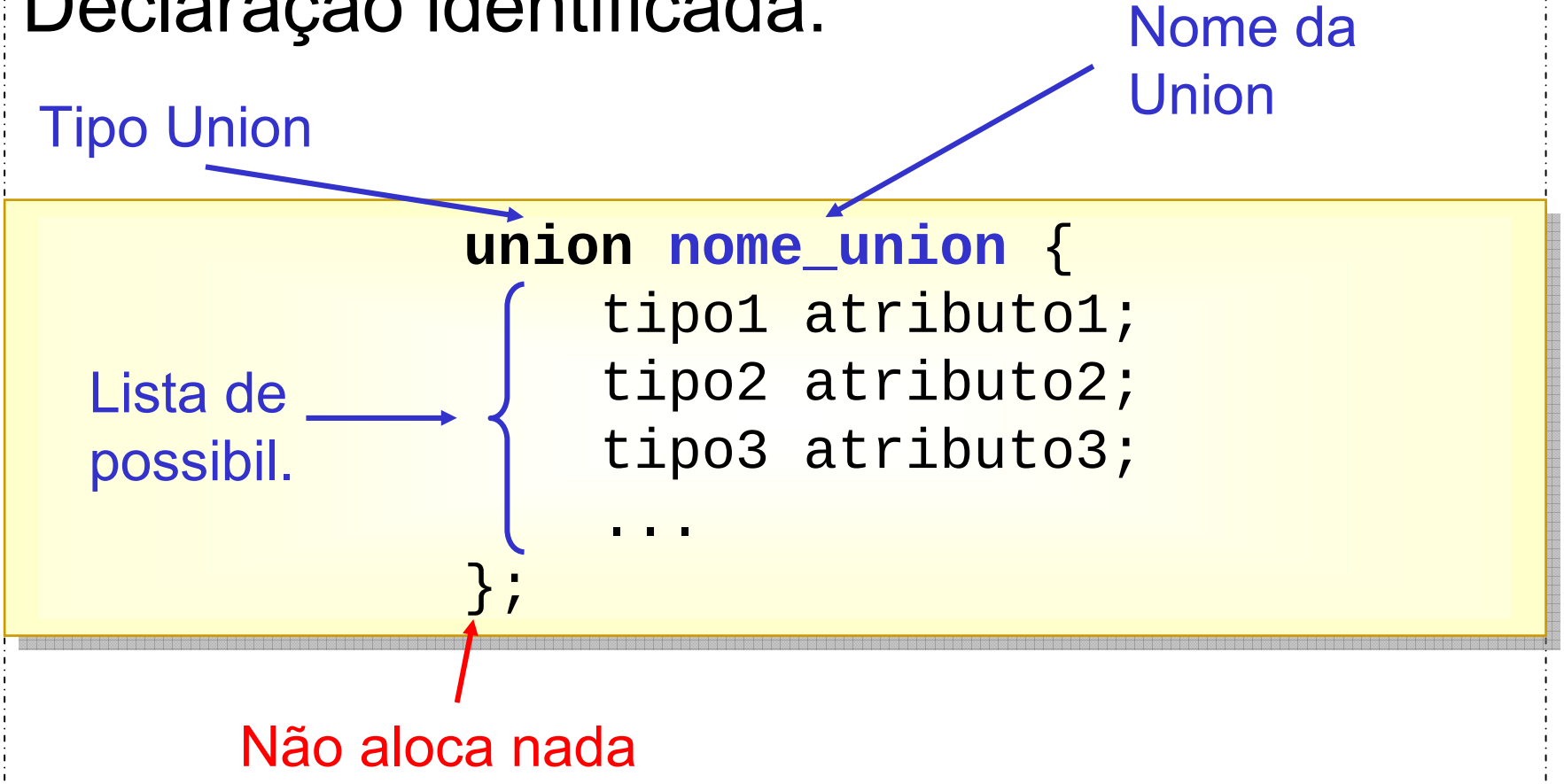
Unions: definição de tipo

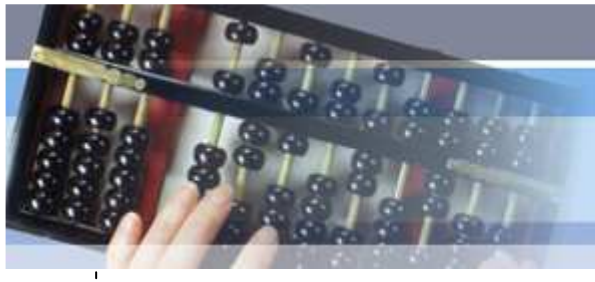
Declaração identificada:



Unions: definição de tipo

Declaração identificada:





Unions: definição de tipo

Declaração identificada:

Tipo Union

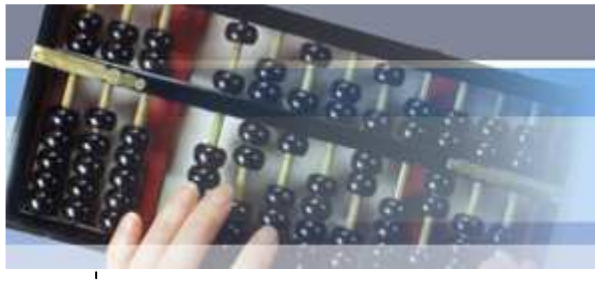
Nome da Union

```
union nome_union {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
};
```

Lista de possibil.

Não aloca nada

- *Declara a union*
- *Memoriza a declaração*
- *Novo tipo de dados*
- *NÃO cria a variável*



Unions: definição de tipo

Declaração identificada:

Declaração memorizada pelo compilador

```
union nome_union {
        tipo1 atributo1;
        tipo2 atributo2;
        tipo3 atributo3;
        ...
};
```

Unions: definição de tipo

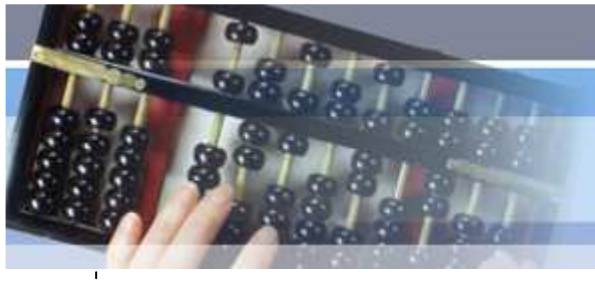
Declaração identificada:

Declaração memorizada pelo compilador

```
union nome_union {
        tipo1 atributo1;
        tipo2 atributo2;
        tipo3 atributo3;
        ...
};
```

Declaração abreviada de variável:

```
union nome_union variável
```



Exemplo

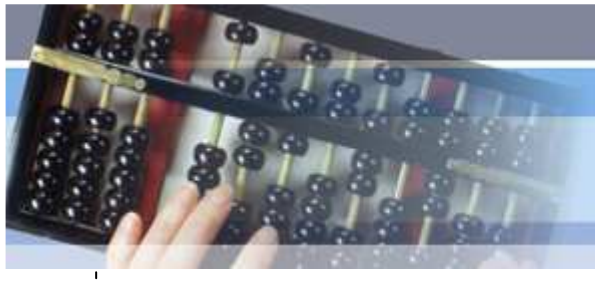
```
struct TipoData {
      int dia;
      int mes;
      int ano;
};


 struct TipoEndereco {
       char local[50];
       int numero;
       char cep[10];
 };
```

Exemplo

```
struct TipoPessoa {
        char nome[50];
        struct TipoData data_nascimento;
        struct TipoEndereco endereco_residencia;
        union {

        } dados_especificos;
} pessoa;
```



Exemplo

```
struct TipoPessoa {
        char nome[50];
        struct TipoData data_nascimento;
        struct TipoEndereco endereco_residencia;
        union {
                struct {
                        struct TipoData data_matricula;
                        int codigo_curso;
                } aluno;









        } dados_especificos;
} pessoa;
```

Exemplo

```
struct TipoPessoa {
        char nome[50];
        struct TipoData data_nascimento;
        struct TipoEndereco endereco_residencia;
        union {
                struct {
                        struct TipoData data_matricula;
                        int codigo_curso;
                } aluno;
                struct {
                        int codigo_contrato;
                        struct TipoData data_contratacao;
                        int codigo_cargo;
                } funcionario;



        } dados_especificos;
} pessoa;
```

Exemplo

```
struct TipoPessoa {
        char nome[50];
        struct TipoData data_nascimento;
        struct TipoEndereco endereco_residencia;
        union {
                struct {
                        struct TipoData data_matricula;
                        int codigo_curso;
                } aluno;
                struct {
                        int codigo_contrato;
                        struct TipoData data_contratacao;
                        int codigo_cargo;
                } funcionario;
                struct {
                        char area_pesquisa[40];
                        char nome_departamento[40];
                } professor;
        } dados_especificos;
} pessoa;
```

Structs e Unions

Fim do Capítulo

Curso de C

Nomeando Tipos

Nomeando Tipos

Typedef

Typedef

Typedef: Definir novos nomes para tipos existentes

Typedef

Typedef: Definir novos nomes para tipos existentes

Declaração:

```
typedef tipo_original sinonimo;
```

Typedef

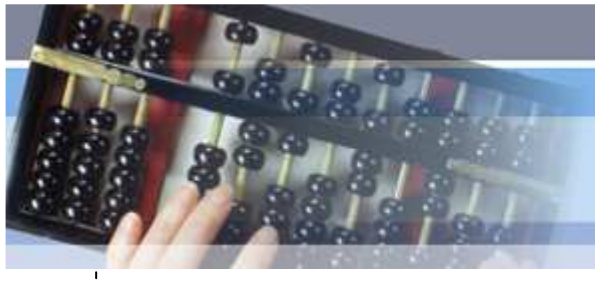
Typedef: Definir novos nomes para tipos existentes

Declaração:

```
typedef tipo_original sinonimo;
```

Qualquer tipo C

Declaração já vista pelo compilador



Typedef

Typedef: Definir novos nomes para tipos existentes

Declaração:

```
typedef tipo_original sinonimo;
```

Qualquer tipo C

Declaração já vista pelo compilador

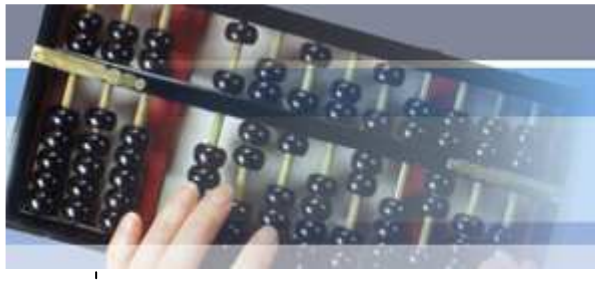
Novo nome do tipo

Typedef

Exemplo: Sinônimos para nomes de tipos

Definir sinônimo para **`int`**:

```
typedef int inteiro;
```



Typedef

Exemplo: Sinonimos para nomes de tipos

Definir sinonimo para **`int`**:

```
typedef int inteiro;
```

Declarar variável com o sinonimo:

```
inteiro numero;
```

Declaração equivalente:

```
int numero;
```

Typedef

Exemplo: Abreviar nomes de tipos

Definir sinonimo para **`unsigned long long int`**:

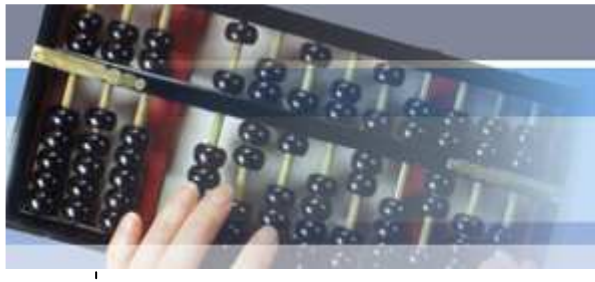
```
typedef unsigned long long int superint;
```

Declarar variável com o sinonimo:

```
superint numero;
```

Declaração equivalente:

```
unsigned long long int numero;
```

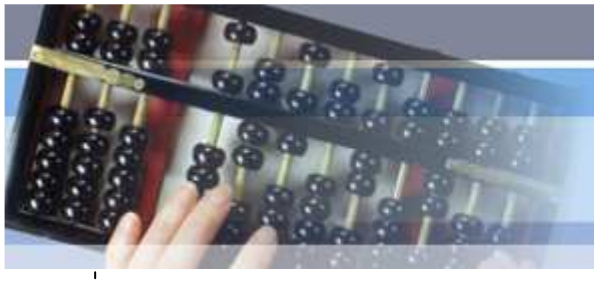


Typedef

Exemplo: Ocultar detalhes da estrutura

Definição de número complexo:

```
struc TipoComplexo {
    double real;
    double imaginario;
}
```



Typedef

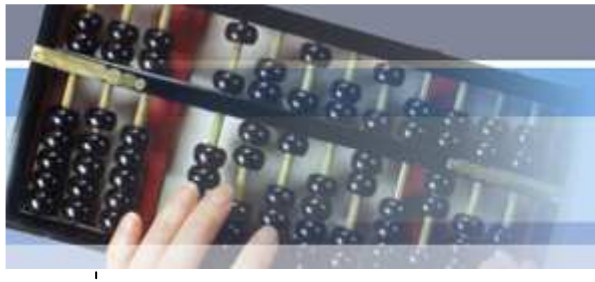
Exemplo: Ocultar detalhes da estrutura

Definição de número complexo:

```
struc TipoComplexo {
    double real;
    double imaginario;
}
```

Declarar variável de número complexo:

```
struct TipoComplexo numero;
```



Typedef

Exemplo: Ocultar detalhes da estrutura

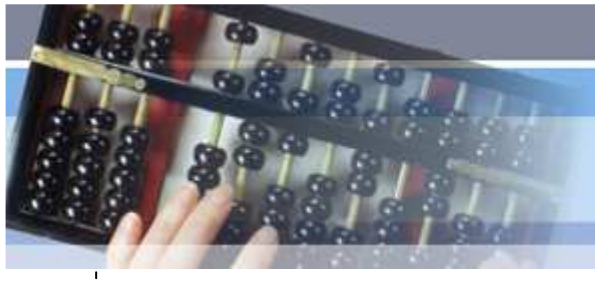
Definição de número complexo:

```
struc TipoComplexo {
    double real;
    double imaginario;
}
```

Declarar variável de número complexo:

```
struct TipoComplexo numero;
```

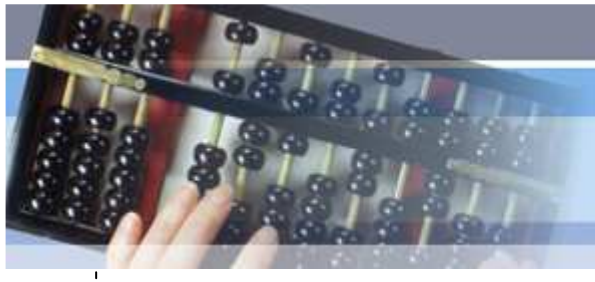
Não desejamos saber que **`TipoComplexo`** é estrutura



Typedef

Exemplo: Ocultar detalhes da estrutura

```
struct TipoComplexo {
    double real;
    double imaginario;
}
```



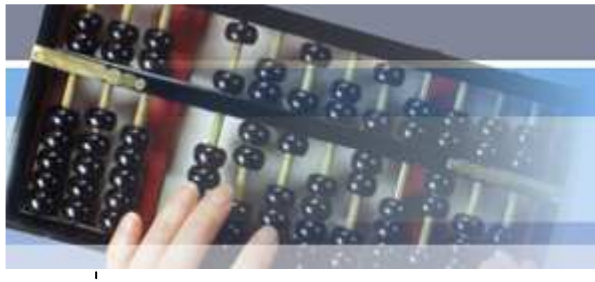
Typedef

Exemplo: Ocultar detalhes da estrutura

```
struct TipoComplexo {
    double real;
    double imaginario;
}
```

Definir sinonimo que esconde **`struct TipoComplexo`**:

```
typedef struct TipoComplexo complexo;
```



Typedef

Exemplo: Ocultar detalhes da estrutura

```
struct TipoComplexo {
    double real;
    double imaginario;
}
```

Definir sinonimo que esconde **`struct TipoComplexo`**:

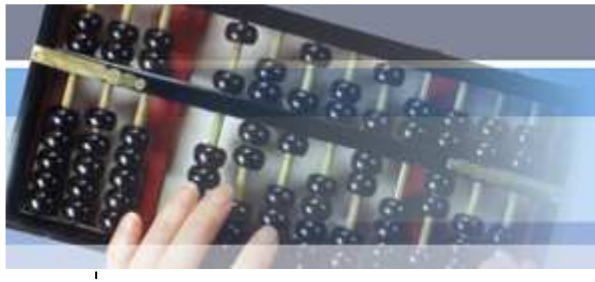
```
typedef struct TipoComplexo complexo;
```

Declarar variável:

```
complexo numero;
```

Curso de C

Enumerações



Enumerações

Objetivos:

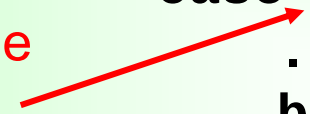
Aprender a representar um conjunto de opções através de identificadores

Enumerações

```
char forma_pagamento;

switch (forma_pagamento) {
   case 'd': // dinheiro
        ...
        break;
   case 'c': // cartão
        ...
        break;
   case 'v': // vale
        ...
        break;
}
```

Convenção de significados



Enumerações

```
char forma_pagamento;

switch (forma_pagamento) {
   case 'd': // dinheiro
        ...
        break;
   case 'c': // cartão
        ...
        break;
   case 'v': // vale
        ...
        break;
}
```

Convenção de significados

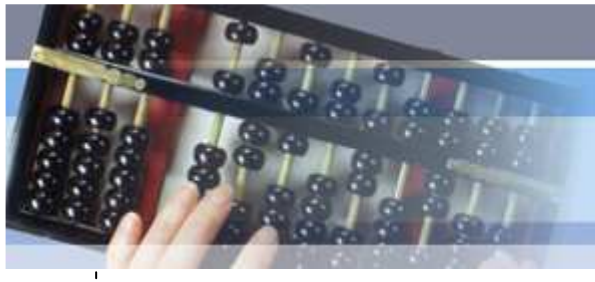
Problemas:

- *Criar a convenção*
- *Manter a convenção*
- *Adicionar significados*
- *Alterações na convenção*

Enumerações

Conceitos:

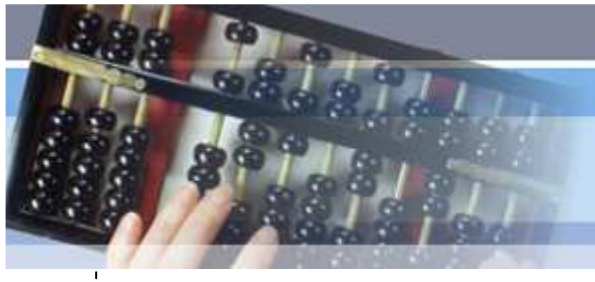
- Enumeração: conjunto de nomes
- Associa internamente cada nome a um número
- Valor da variável tipo enumeração é sempre apenas uma das opções



Enumeações

Declaração:

```
enum nome_enumeracao {
    opcao1,
    opcao2,
    ...
} variavel;
```



Enumeações

Declaração:

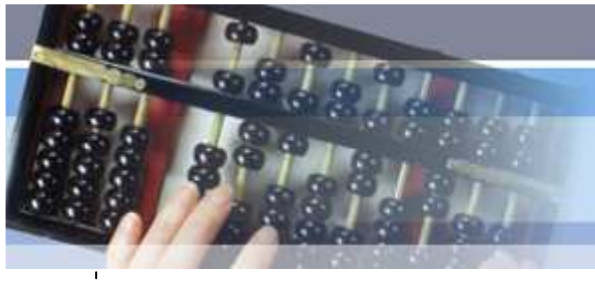
Tipo Enumeração

Nome da enumeração

```
enum nome_enumeracao {
    opcao1,
    opcao2,
    ...
} variavel;
```

Lista de opções

Cria variável



Enumeações

Declaração:

Nome da enumeração

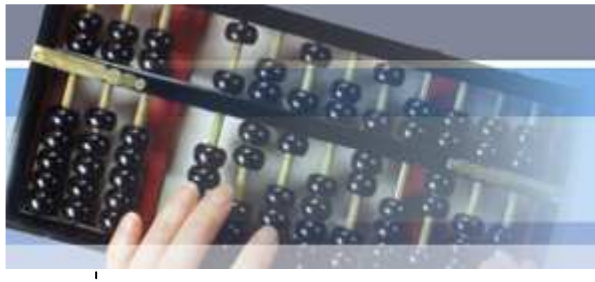
Tipo Enumeração

```
enum nome_enumeracao {
    opcao1,
    opcao2,
    ...
} variavel;
```

Lista de opções

Cria variável

- *Declara enumeração*
- *Memoriza a declaração*
- *Novo tipo de dados*
- *Cria a variável*



Enumerações

Uso de enumerações:

Declaração:

```
enum {dinheiro0, cheque1, vale_refeicao2,
      cartao3} forma_pagamento;
```

Enumerações

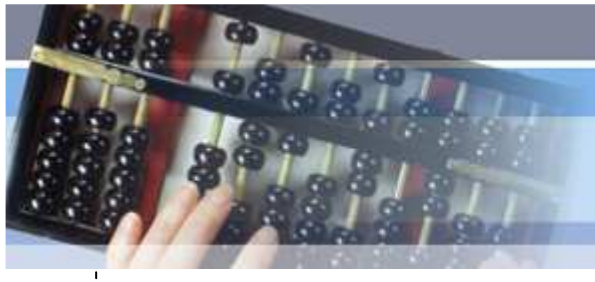
Uso de enumerações:

Declaração:

```
enum {dinheiro[0], cheque[1], vale_refeicao[2],
      cartao[3] } forma_pagamento;
```

Seleção:

```
switch (forma_pagamento) {
        case dinheiro: ...
               break;
        case cheque:  ...
               break;
        case vale_refeicao: ...
               break;
        case cartao: ...
               break;       }
```



Enumerações

Declaração:

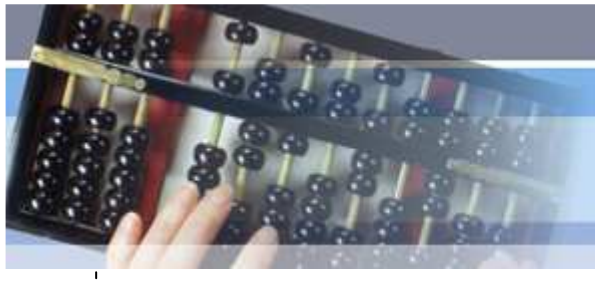
Tipo Enumeração

Nome da enumeração

```
enum nome_enumeracao {
    opcao1 = valor1,
    opcao2 = valor2,
    ...
} variavel;
```

Lista de opções

Cria variável



Enumerações

Declaração:

Tipo Enumeração

Nome da enumeração

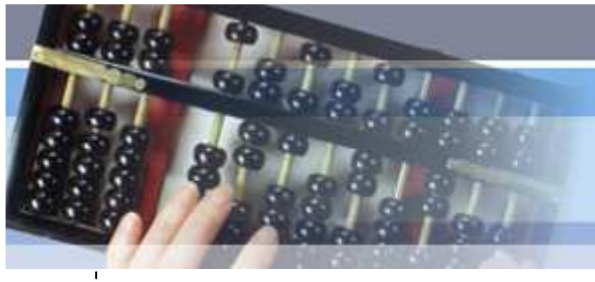
```
enum nome_enumeracao {
    opcao1 = valor1,
    opcao2 = valor2,
    ...
} variavel;
```

Lista de opções

Cria variável

Associação manual entre nomes e valores

>



Enumerações

Uso de enumerações:

Declaração:

```
enum {
        dinheiro = ‘d’,
        cheque = ‘c’,
        vale_refeicao = ‘v’,
        cartao = ‘k’
} forma_pagamento;
```

Enumerações

Fim de capítulo