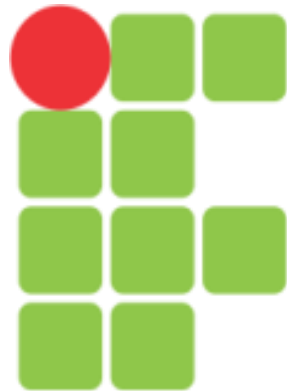




INSTITUTO FEDERAL
CEARÁ



INSTITUTO FEDERAL
CEARÁ



Ernani Andrade Leite

ernani@ifce.edu.br

Classificação dos tipos de dados

- Tipos de Dados Simples
 - Tipos de Dados Inteiros: *int*, *long* e *unsigned int*
 - Tipos de Dados Reais: *float* e *double*
 - Tipos de Dados Caracteres: *char*
- Tipos de Dados Estruturados
 - cadeia de caracteres (*string*), vetores (*array*), registros (*struct*) e arquivos em disco.
- ponteiros (alocação dinâmica de memória)
- Classes e Objetos (Orientação a Objetos)

Tipos de Dados Estruturados

- armazenam diversos itens de uma só vez
- isto significa:
 - em uma mesma estrutura de dados, é possível ter diversas variáveis de tipos de dados simples agrupadas

Lembrando Vetores

- representa um conjunto de valores do mesmo tipo (estrutura homogênea), referenciáveis pelo mesmo nome e individualizados entre si através de sua posição dentro desse conjunto (variáveis indexadas)
 - portanto, os vetores, armazenam “múltiplos” valores do mesmo tipo
-
- para armazenar, ou agrupar, informações “relacionadas” que têm tipos diferentes deve-se utilizar variáveis do tipo registro, ou estrutura (struct)

Registro, ou **Estrutura** (**struct**) :

Considere que seja informado o nome de um aluno e suas quatro notas bimestrais que deverão ser agrupadas em uma mesma estrutura.

Cadastro de Notas Escolares	
Nome.....:	_____
Primeira Nota.:	_____
Segunda Nota.:	_____
Terceira Nota.:	_____
Quarta Nota.:	_____



Layout de um registro de aluno formado pelos campos: **Nome**, **Primeira Nota**, **Segunda Nota**, **Terceira Nota** e **Quarta Nota**.

Definições de registro (struct)

- representa um conjunto de valores logicamente relacionados, que podem ser de tipos diferentes (estrutura heterogênea)
- junção ou composição de tipos em um tipo composto
- conjunto de informações agrupadas e relacionadas entre si
- agrupamento de variáveis, não necessariamente do mesmo tipo, que guardam estreita relação lógica

struct, características básicas

- contém um número fixo de elementos chamados de campos, ou “membros”
- os campos podem ser de tipos diferentes (estrutura heterogênea)
- cada campo tem um nome próprio chamado de “*identificador de campo*”
- campo = unidade de registro

Exemplos de registros / campos

- Registros informações do cadastro
 - Campos informações do registro
-

- os clientes de uma empresa
nome, endereço, e-mail, cpf, ...
-

- os assinantes da lista telefônica
nome, telefone, endereço
-

- os alunos da disciplina de Algoritmos
matrícula, nome, faltas, notas bimestrais, situação
-

All Database Aliases

Databases Dictionary

banco de dados
DBDEMOS

ponteiro de registro (registro atual)- denota o
registro com o qual o usuário está trabalhando

Name	Capital	Continent	Area	Population
Argentina	Buenos Aires	South America	2777815	32300003
Bolivia	La Paz	South America	1098575	7300000
Brazil	Brasilia	South America	8511196	150400000
Canada	Ottawa	North America	9976147	26500000
Chile	Santiago	South America	756943	13200000
Colombia	Bagota	South America	1138907	33000000
Cuba	Havana	North America	114524	10600000
El Salvador	San Salvador	North America	20865	5300000
Ecuador	Quito	South America	11424	2500000
Guyana	Georgetown	South America	1967180	88600000
Jamaica	Kingston	North America	139000	3900000
		South America	406576	4660000
Peru	Lima	South America	1285215	21600000
United States of America	Washington	North America	9363130	249200000
Uruguay	Montevideo	South America	176140	3002000
Venezuela	Caracas	South America	912047	19700000

tabela de dados "**country.db**"
do banco de dados **DBDEMOS**

linhas = registros

colunas = campos

Como declarar **registros** (ou **estruturas**) - **struct**:

Na **linguagem C**, os **registros** (ou **estruturas**) devem ser definidos antes das declarações das variáveis, pois é muito comum ocorrer a necessidade de se declarar uma ou mais variáveis com o tipo de registro definido, conforme a seguinte sintaxe:

```
// definição do tipo registro  
struct <IdentificadorDoRegistro>  
{  
    <lista dos tipos e seus campos (ou membros)>;  
};
```

```
// declaração da variável do tipo registro definido  
struct <IdentificadorDoRegistro> <NomeDaVariável>;
```



onde:

IdentificadorDoRegistro: nome do tipo registro definido

lista dos tipos e seus campos: relação de tipos e campos do registro

NomeDaVariável: nome da variável declarada a partir do tipo registro definido anteriormente

Lista dos tipos e seus **campos** (ou membros):

```
// definição do tipo registro  
struct <IdentificadorDoRegistro>  
{  
    <tipo1> <campo1>;  
    <tipo2> <campo2>;  
    ...  
    <tipoN> <campoN>;  
};
```

```
// declaração da variável do tipo registro definido  
struct <IdentificadorDoRegistro> <NomeDaVariável>;
```

onde:

campo1, campo2, ..., campoN: representam os nomes associados a cada campo do registro;

tipo1, tipo2, ..., tipoN: representam qualquer um dos tipos básicos ou 'tipo anteriormente definido'.

Tomando como exemplo a proposta de se criar um registro denominado **rgAluno**, cujos campos são **nome**, **nota1**, **nota2**, **nota3** e **nota4**, este seria assim declarado em C:

```
// definição do tipo registro rgAluno  
struct rgAluno {  
    char nome[35];  
    float nota1;  
    float nota2;  
    float nota3;  
    float nota4;  
};
```

```
// declaração da variável Aluno declarada a partir  
// do tipo registro rgAluno  
struct rgAluno Aluno;
```

Este exemplo corresponde à definição de um modelo de um registro (**rgAluno**) e à criação de uma área de memória chamada **Aluno**, capaz de conter cinco subdivisões, ou campos: **nome**, **nota1**, **nota2**, **nota3** e **nota4**.

Declaração "compacta" e inicialização de **registros**:

Na sintaxe da linguagem C é permitido declarar e inicializar variáveis do tipo de registro segundo a definição do modelo do registro, conforme mostra o exemplo a seguir:

```
// declaração da variável Aluno declarada a partir  
// do tipo registro rgAluno  
struct rgAluno {  
    char nome[35];  
    float nota1;  
    float nota2;  
    float nota3;  
    float nota4;  
} Aluno = {"Omero Fco Bertol", 7.0, 8.5, 10.0, 7.0};
```

Nota:

Esta forma de declaração justifica, ou explica, a necessidade da colocação do caracter ";" logo após o símbolo sintático "}" que encerra a declaração do tipo registro.

Para utilizar um campo específico do registro, deve-se diferenciar esse campo. Para tal utiliza-se o caractere "." (ponto) para estabelecer a separação do nome da variável registro do nome do campo.

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

para denotar
tipos estruturados
registro (struct)

```
struct rgData dtNascto;
```

// referência aos campos da variável registro

```
dtNascto.dia = 30;  
dtNascto.mes = 11;  
dtNascto.ano = 1965;
```

Usando o registro time `<dos.h>`

definição:

```
struct time {  
    unsigned char ti_min;    // minutos  
    unsigned char ti_hour;  // horas  
    unsigned char ti_hund;  // centésimos de segundos  
    unsigned char ti_sec;   // segundos  
};
```

.....

```
#include <dos.h>
```

```
void mostraHora(int x, int y) {  
    // declarando a variável t do tipo registro time  
    struct time t;  
  
    // pega a hora do sistema operacional  
    gettimeofday(&t);  
    gotoxy(x, y);  
    printf("%d:%d:%d", t.ti_hour, t.ti_min, t.ti_sec);  
}
```

Usando o registro date <dos.h>

definição:

```
struct date {  
    int da_year;        // ano corrente  
    char da_day;        // dia do mês  
    char da_mon;        // mês do ano (1 = Janeiro)  
};
```

.....

```
#include <dos.h>
```

```
void mostraData(int x, int y, int tipo) {  
    char mes[12][3] = {"jan", "fev", "mar", "abr", "mai",  
        "jun", "jul", "ago", "set", "out", "nov", "dez"};  
    struct date d;  
    getdate(&d);  
    gotoxy(x, y);  
    if (tipo == 0)        // formato: 99/99/9999  
        printf("%d/%d/%d", d.da_day, d.da_mon, d.da_year);  
    else                  // formato: 99/xxx/9999  
        printf("%d/%.3s/%d", d.da_day, mes[d.da_mon-1], d.da_year);  
}
```


Operações básicas com registros (struct):

Do mesmo modo que acontece com variáveis simples, também é possível realizar operações de atribuição (=), leitura (scanf) e escrita (printf) com variáveis compostas heterogêneas do tipo registro ou estrutura.

.....

Para referenciar todas as informações contidas no registro

```
struct rgAluno {  
    char nome[35];  
    float nota1;  
    float nota2;  
};
```

```
struct rgAluno Aluno, NewAluno;
```

```
// Neste caso o acesso ao registro é feito genericamente,  
// ou seja, todos os campos do registro são envolvidos  
// na operação de atribuição.
```

```
    NewAluno = Aluno;
```

Lendo valores para os campos do registro:

```
struct rgAluno {  
    char nome[35];  
    float nota1;  
    float nota2;  
};
```

```
struct rgAluno Aluno;
```

.....

```
void main() {  
    printf("Nome do Aluno: ");  
    gets(Aluno.nome);  
  
    printf("Nota 1o. Bimestre: ");  
    scanf("%f", &Aluno.nota1);  
  
    printf("Nota 2o. Bimestre: ");  
    scanf("%f", &Aluno.nota2);  
}
```

Escrevendo os valores dos campos do registro:

```
struct rgAluno {  
    char nome[35];  
    float nota1;  
    float nota2;  
};
```

```
struct rgAluno Aluno;
```

.....

```
void main() {
```

```
    ...
```

```
    printf("Nome do Aluno: %s\n", Aluno.nome);
```

```
    printf("Nota 1o. Bimestre: %5.2f\n", Aluno.nota1);
```

```
    printf("Nota 2o. Bimestre: %5.2f\n", Aluno.nota2);
```

```
}
```

Campos do Registro declarados como vetor:

Cadastro de Notas Escolares

Nome.: _____

	0	1	2	3
Nota.:				

```
struct rgAluno {  
    char nome[35];  
    float nota[4];  
};  
struct rgAluno Aluno;
```

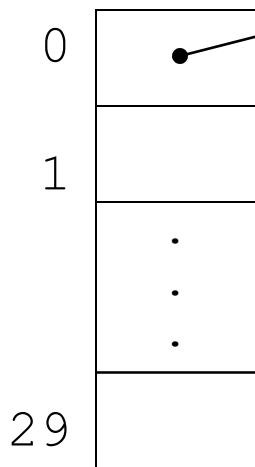
Para manipular um campo do registro do tipo vetor deve-se obedecer às manipulações próprias de cada estrutura de dados, ou seja:

```
Aluno.nota[1] = 7.5;    scanf("%f", &Aluno.nota[3]);
```

Vetores de registros:

Para controlar as notas de 30 alunos, numerados de 0 até 29 seqüencialmente, basta criar um vetor no qual cada posição é um elemento do tipo registro **rgAluno**.

Aluno[?]



Cadastro de Notas Escolares

Nome.: _____

0 1 2 3

Nota.: _____

--	--	--	--

```
const n = 30;
```

```
struct rgAluno {
```

```
    char nome[35];
```

```
    float nota[4];
```

```
};
```


```
struct rgAluno Aluno[n];
```

```
gets(Aluno[5].nome);
```

```
scanf("%f", Aluno[i].nota[1]);
```

Campos de registro que são do tipo registro:

```
struct rgData {  
    int dia;  
    int mes;  
    int ano;  
};  
  
struct rgPessoa {  
    char nome[35];  
    char sexo;  
    struct rgData dtNascto;  
};
```



```
struct rgPessoa Pessoa;
```

// para cada nível da estrutura utiliza-se "um"
// o caracter . (ponto)

```
strcpy(Pessoa.nome, "Estruturas de Dados");  
Pessoa.sexo = 'M';  
Pessoa.dtNascto.dia = 30;  
Pessoa.dtNascto.mes = 11;  
Pessoa.dtNascto.ano = 1965;
```

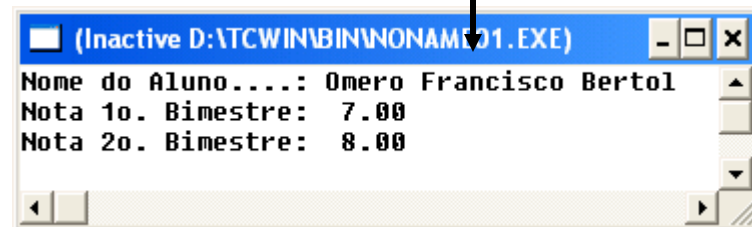
Passando um registro a uma **Função:**

```
struct rgAluno {  
    char nome[35];  
    float nota1;  
    float nota2;  
  
} Aluno = {"Omero Francisco Bertol", 7.0, 8.0};
```

ImprimeAluno(Aluno);

.....

```
void ImprimeAluno(struct rgAluno ficha) {  
  
    printf("Nome do Aluno....: %s\n", ficha.nome);  
    printf("Nota 1o. Bimestre: %5.2f\n", ficha.nota1);  
    printf("Nota 2o. Bimestre: %5.2f\n", ficha.nota2);  
}
```



Referências

- Estrutura de Dados Fundamentais: conceitos e aplicações.
 - Silvio do Lago Pereira.
 - 2^a ed. - São Paulo: Érica, 1996.
- Instituto de Computação da UNICAMP
 - Flávio K. Miyazawa & Tomasz Kowaltowski
- Material adaptado do Prof.: Omero Francisco Bertol - UTFPR.