Registros – *Structs*

UFOP
Universidade Federal de Ouro Preto

CSI030 – PROGRAMAÇÃO DE COMPUTADORES I

DECSI

Registros

- Até o momento foram abordadas apenas estruturas de dados **homogêneas**:
  - Vetores, matrizes e strings.
- Com tipos de dados primitivos:
  - inteiro, real, caracter, ...

Registros

- Por exemplo, podemos armazenar as três notas de uma turma com 40 pessoas numa matriz:
  - **real** notas[40][3];
- Para o nome, poderíamos criar outra matriz:
  - **caracter** nomes[40][100];

Registros

- E para os demais dados, como matrícula, curso, gênero, dentre outros?
  - Criar uma matriz para cada um deles ficaria inviável de se manipular.
  - Ao adicionar ou excluir uma pessoa, todas as variáveis precisariam ser acessadas e atualizadas.
- E se pudéssemos agrupar esses dados?

Registros

- Os dados podem combinados em variáveis compostas.
- Essas variáveis são **heterogêneas**.
- Elas são conhecidas como registros ou estruturas (em C, são definidas como **structs**).

Registros

- Um **registro** (**struct**) é a maneira de se agrupar variáveis, sejam elas de mesmo tipo ou de tipos diferentes.
- As variáveis unidas em um registro se relacionam de modo a criar um contexto maior.
- **Exemplos** de uso de registros:
  - Registro de Alunos: armazena nome, matrícula, médias, faltas.
  - Registro de Pacientes: armazena nome, endereço, convênio, histórico hospitalar.
  - Registro de funcionários: armazena nome, endereço, cargo, salário.

Declaração do registro

- Um registro é declarado da seguinte maneira:

```
registro nome_tipo_registro {
    tipo_1 variavel_1;
    tipo_2 variavel_2;
    ...
    tipo_n variavel_n;
}
```

- Uma variável deste registro deve ser declarada da seguinte maneira:

```
registro nome_tipo_registro variavel_registro;
```

Registros

- Um registro define um **novo tipo de dados** com nome **nome_tipo_registro** que possui os campos **variavel_i.**

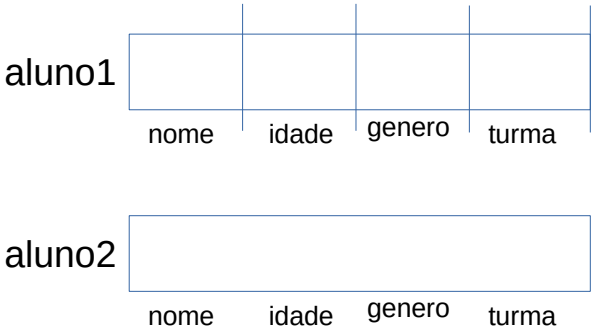
- O campo **variavel_i** é uma variável do tipo **tipo_i** e será acessível a partir de uma variável do novo tipo **nome_tipo_registro**.

- **variavel_registro** é uma variável do tipo **nome_tipo_registro** e possui, internamente, os campos **variavel_i** do tipo declarado.

- A **declaração de um registro** pode ser feita dentro de uma sub-rotina ou fora dela.

Exemplo de utilização

```
// Definição da estrutura
registro Aluno {
    caracter nome[50];
    inteiro idade;
    caracter genero;
    inteiro turma;
}

inicio
    // Declaração de variáveis
    registro Aluno aluno1, aluno2;
fim
```



Inicialização de registros

- Na inicialização de registros, os campos obedecem à ordem de declaração:

```
registro contaBancaria {
    inteiro numero;
    caracter idCorrentista[15];
    real saldo;
}

registro contaBancaria conta = {1,"MG1234567",100.0};
```

Acesso aos campos do registro

- Os campos de um registro podem ser acessados individualmente a partir de variáveis do tipo do registro da seguinte maneira:

  - **variavel_registro.variavel_i**

- Cada campo **variavel_registro.variavel_i** se comporta como uma variável do **tipo** do campo, ou seja, **tipo_i**.

- Isso significa que todas as operações válidas para variáveis do tipo **tipo_i** são válidas para o campo acessado por **variavel_registro.variavel_i**.

Atribuição de registros

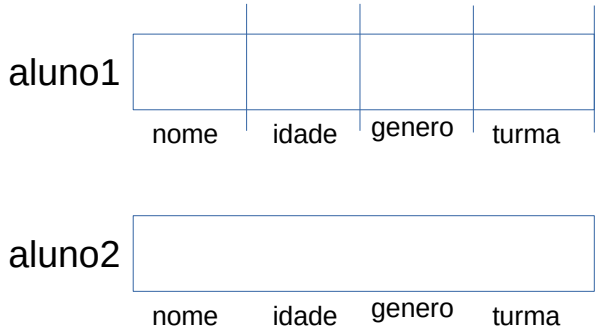
- Pode-se copiar o conteúdo de uma estrutura para outra utilizando uma atribuição simples.

```
registro ponto {int x; int y;};

inicio
    registro ponto p = {1,2}, q;
     q.x = 3; q.y = 4;
     q = p;
     q.x = 5;
fim
```

p		q	
x	y	x	y
1	2		
1	2	**3**	**4**
1	2	**1**	**2**
1	2	**5**	2

Exemplo de utilização

```
// Definição da estrutura
registro Aluno {
    caracter nome[50];
    inteiro idade;
    caracter genero;
    inteiro turma;
};

inicio
    // Declaração de variáveis
    registro Aluno aluno1, aluno2;
     aluno1.nome = “João”
     aluno1.idade = 20
     aluno1.genero = “M”
     aluno1.turma = 31
fim
```

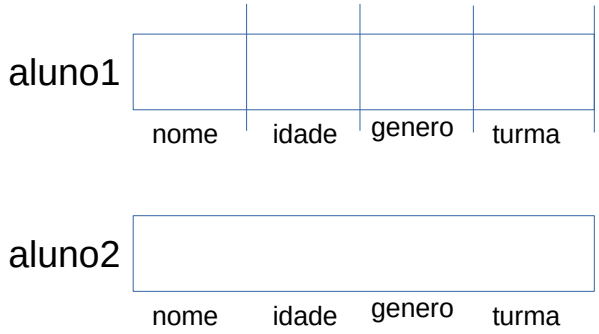


Declaração do registro em C - Struct

```
struct nome_tipo_registro {
    tipo_1 variavel_1;
    tipo_2 variavel_2;
    ...
    tipo_n variavel_n;
};
```

Exemplo de utilização

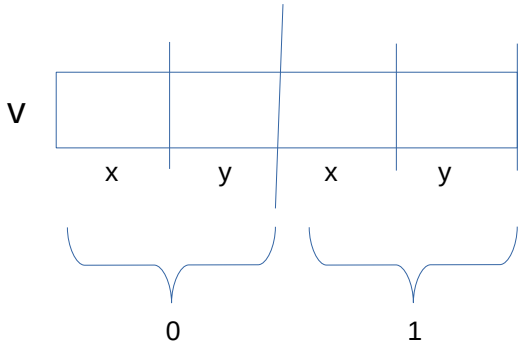
```
// Definição da estrutura
struct Aluno {
    char nome[50];
    int idade;
    char genero;
    int turma;
};

int main(void){
    // Declaração de variáveis
    struct Aluno aluno1, aluno2;
     aluno1.nome = “João”
     aluno1.idade = 20
     aluno1.genero = ‘M’
     aluno1.turma = 31
}
```



Vetores de registros

- **Vetores de registros** podem ser criados da mesma maneira que se criam vetores de tipos primitivos (inteiro**, real, caracter**).
- É necessário definir a estrutura antes de declarar o vetor.

```
registro ponto {inteiro x; inteiro y;};

inicio
    registro ponto v[2];
    v[0].x = 3; v[0].y = 4;
    v[1].x = 5; v[1].y = 6;
fim
```



Registros como parâmetros de sub-rotinas

- Pode-se passar um registro como parâmetro de sub-rotinas.

```
registro ponto {inteiro x; inteiro y;};

 imprimePonto( registro ponto p ){
    imprima(p.x, p.y);
}

inicio
    registro ponto p = {1,2};
    imprimePonto(p);
fim
```

Sinônimos de tipos

```
struct ponto {int x; int y;};

typedef ponto Ponto;

Ponto r, s;
```

Sinônimos de tipos

- Outro exemplo:

```
struct contabancaria {
int numero;
char idCorrentista[15];
float saldo;
};
typedef struct contabancaria CB;

CB conta1, conta2;
```

Sinônimos de tipos

- Podemos ainda associar uma definição de **struct** a um novo tipo de dados, evitando a repetição da palavra **struct** pelo código:

```
typedef struct contabancaria {
int numero;
char idCorrentista[15];
float saldo;
} CB;

CB conta1, conta2;
```

UFOP
Universidade Federal de Ouro Preto

Exercícios

DECSI

Exercício 1

Crie uma estrutura para representar um grupo de alunos. Cada registro deve possuir o campo nome e três notas. Solicite que o usuário preencha as informações e, em seguida, imprima todos os dados dos respectivos alunos.

```
typedef struct aluno{
   char nome [40];
   double nota[3];
} Aluno;

void leVetorAluno(Aluno temp[], int tam){
   int i, j;
   for (i = 0; i < tam; ++i){
       printf("Informe o nome do aluno: ");
       scanf("%s",temp[i].nome);
       for (j = 0; j < 3; ++j){
           printf("Informe a nota %d: ",j+1);
           scanf("%lf",&temp[i].nota[j]);
       }   }  }
```

```
void imprimeVetorAluno(Aluno temp[], int tam){
   int i, j;
   for (i = 0; i < tam; ++i){
       printf("Aluno %d:", i+1);
       printf("\tNome: %s\n",temp[i].nome);
       for (j = 0; j < 3; ++j){
           printf("\tNota %d = %.2lf\n",
                  j+1, temp[i].nota[j]);
       }   }}
int main(){

   Aluno alunos[2];
   leVetorAluno(alunos, 2);
   imprimeVetorAluno(alunos, 2)
   return 0;
}
```

Exercício 2

Crie uma estrutura ponto contendo os campos x e y, coordenadas de um ponto dadas por valores reais. Declare as variáveis p1 e p2 (dois pontos), preencha os campos para cada variável, via teclado, e imprima a distancia entre p1 e p2.

Nota: A distancia entre dois pontos é dada pela fórmula seguinte: d = √((x2 - x1)2 + (y2 - y1)2).

Considere, p1(x1 , y1) e p2(x2 , y2).

```
typedef struct ponto {
    float x,y;
}Ponto;

main ()
{
    Ponto p1,p2;
    float distancia;
    printf ("\nDigite (X,Y) de P1.\n");
    scanf ("%f %f",&p1.x,&p1.y);
    printf ("\nDigite (X,Y) de P2.\n");
    scanf ("%f %f",&p2.x,&p2.y);
    distancia=sqrt(pow(p2.x-p1.x,2)+pow(p2.y-p1.y,2));
    printf ("\nA distância entre P1 e P2 e:%.2f.\n\n",distancia);
    return 0;
}
```