Registros

Prof. Angelo Loula
UEFS

Vetores e Matrizes

- Variáveis Compostas Homogêneas (mesmo tipo)

```
int lista[10];
```

lista										
posição	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
valor										

Todos os elementos são *int*

```
int matriz[4][5];
```

matriz					
índices	0	1	2	3	4
0					
1					
2					
3					

Todos os elementos são *int*

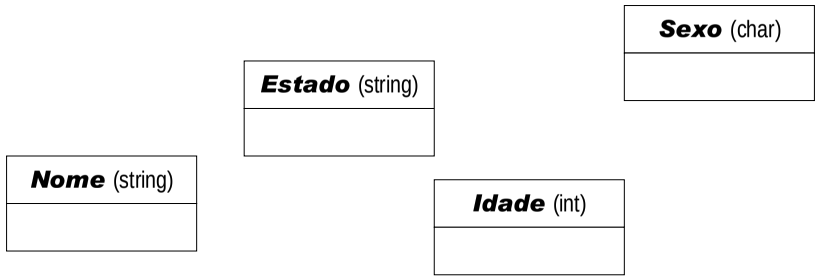
Registros

- Variáveis Compostas Heterogêneas
  - Podem agrupar variáveis de tipos diferentes
  - Váriáveis (campos ou membros) com nomes diferentes mas relacionadas e referenciada por um nome comum
  - Chamadas de registros ou estruturas

Pessoa			
Nome (string)	***Estado*** (string)	***Idade*** (int)	***Sexo*** (char)

Registros

```
char nome[50];
char estado[20];
int idade;
char sexo;
```



Registros

```
tipopessoa pessoa;
...
strcpy(pessoa.nome, "Fulano");
strcpy(pessoa.estado, "Paraná");
pessoa.idade = 37;
pessoa.sexo = 'M';
```

Pessoa			
Nome (string)	***Estado*** (string)	***Idade*** (int)	***Sexo*** (char)
Fulano	*Paraná*	*37*	*M*

Registros

```
typedef struct estruturapessoa{
  char nome[50];
  char estado[20];
  int idade;
  char sexo;
} tipopessoa;
...
tipopessoa pessoa;
```

Pessoa			
Nome (string)	***Estado*** (string)	***Idade*** (int)	***Sexo*** (char)

Registros

```
tipopessoa pessoa1,pessoa2,pessoa3;
...
strcpy(pessoa1.nome, "Fulana da Silva");
strcpy(pessoa2.nome, "Cicrano de Tal");
pessoa1.idade = 30;
pessoa2.idade = pessoa1.idade * 2;
pessoa3 = pessoa1;
scanf("%c", &pessoa1.estado);
printf("%c", pessoa1.estado);
```


Registros

- Como representar uma tabela como esta?

Nome (string)	***Estado*** (string)	***Idade*** (int)	***Sexo*** (char)
Fulano	*Paraná*	*37*	*M*
Cicrano	*Rio de Janeiro*	*15*	*M*
Beltrano	*Acre*	*22*	*M*
Fulana	*Pará*	*25*	*F*
Beltrana	*Ceará*	*76*	*F*

Matriz? Registro? Vetor?

Registros

- Como representar uma tabela como esta?

Pessoas	
posição	***valor***
0	(registro tipopessoa)
1	(registro tipopessoa)
2	(registro tipopessoa)
3	(registro tipopessoa)
4	(registro tipopessoa)

Pessoa[0]			
Nome (string)	***Estado*** (string)	***Idade*** (int)	***Sexo*** (char)
Fulano	*Paraná*	*37*	*M*

Um vetor de registros!

```
tipopessoa pessoas[5];
```



```
typedef struct{
  char nome[50];
  char estado[20];
  int idade;
  char sexo;
} tipopessoa;

int main(){
 tipopessoa pessoas[5]; /*tabela de registros*/

 strcpy(pessoas[1].nome,"Fulano");
 strcpy(pessoas[1].estado,"Paraná");
 pessoas[1].idade=37;
 pessoas[1].sexo='M';
 strcpy(pessoas[2].nome,"Cicrano");
 strcpy(pessoas[2].estado,"Rio de Janeiro");
 pessoas[2].idade=15;
 pessoas[2].sexo='M';
}
```


- Registro dentro de registro

```
typedef struct{
  float x;
  float y;
} tipocoordenada;
typedef struct {
  tipocoordenada p1;
  tipocoordenada p2;
} tiporeta;
...
p1.x = 0;
p1.y = 0;
r1.p1.y = 100;
r1.p1.x = 100.5;

scanf ("%f %f", &p2.x, &p2.y);
scanf ("%f %f", &r1.p2.x, &r1.p2.y);

printf ("(%f, %f)\n", p2.x, p2.y);
printf ("(%f, %f)\n", r1.p2.x, r1.p2.y);
```



```
typedef struct{
   float real,img;{ real + img . i }
} complexo;

complexo SomaComplexo(complexo x, complexo y){
  complexo z;

  z.real = x.real + y.real;
  z.img = x.img + y.img;
  return z;
}

complexo SubtraiComplexo(complexo x, complexo y){
 complexo z;

  z.real = x.real - y.real;
  z.img = x.img - y.img;
  return z;
}
```

- Passagem por valor:

```
void ExibeComplexo(complexo x){
  printf ("(%.2f, %.2f)", x.real, x.img);
}
```

- Passagem por referência:

```
void ExibeComplexo2(complexo *x){
  printf ("(%.2f, %.2f)", x->real, x->img);
  /* x->real equivale a (*x).real */
}
void AlteraCoordenada (complexo *x, float a,
  float b){
  x->real = a;
  x->img = b;
}
```


Enumerados

- Tipo de dado representando conjunto de constantes

```
enum tipocor {
   branco,
   preto,
   cinza,
   azul
};

struct forma {
   enum tipocor cor ;
    int pos_x,pos_y;
} geometria[10];
enum tipocor corjanela;

corjanela=branco;
if(corjanela==branco)
```


Enumerados

- Cada elemento do enumerado tem um inteiro associado, por padrão o primeiro elemento é 0, o segundo é 1, e assim por diante.

```
enum cor {
   branco=0,
   preto=1,
   cinza=2,
   azul=3
};
```


Enumerados

- Mas podemos mudar inteiro associado

```
enum cor {
   branco=10,
   preto=15,
   cinza=40,
   azul=30
};
```

- Enumerados melhoram a manutenção e documentação do código, e permitem limitar possíveis valores de variáveis.