

Fundamentos de Programação

Registros (*structs*) - parte 1

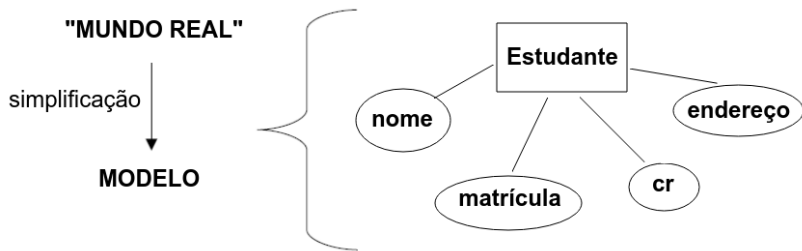
Dainf - UTFPR

Profa. Leyza B. Dorini

Prof. Bogdan T. Nassu

Modelagem de dados

Objetos e dados do “mundo real” tipicamente precisam ser representados computacionalmente por um modelo.



No exemplo acima, a “entidade” Estudante é representada por um conjunto de atributos...

Modelagem de dados

A representação computacional da entidade Estudante poderia ser realizada utilizando os tipos e estruturas de dados que vimos até agora:

```
1  int main(){
2
3      char nome[100];
4      int matricula;
5      float cr;
6      char endereco[500];
7
8      return 0;
9  }
```

Observe que a representação de um estudante específico envolveria a declaração de 4 variáveis, as quais não são automaticamente “agrupadas” (ou seja, é responsabilidade do(a) programador(a) controlar quais variáveis estão associadas a quais estudantes).

Mais um exemplo...

Considere, por exemplo um jogo de luta. Neste caso, diversos conceitos precisam ser modelados, incluindo:

- Para um personagem: velocidade, golpes disponíveis, etc.
- Para um participante: personagem, hit points, posição atual, etc.
- Para a luta: round atual, participantes, tempo, etc.
- Para o jogo: luta atual, lutas concluídas, resultados anteriores, etc.

Como fazer isso?

O próximo passo na nossa escala de abstração é criar novas estruturas de dados! Que tal algo como os tipos `Personagem`, `Participante` e `Luta`?

Registros

Registros (que em C são chamados de structs) permitem representar conceitos de forma mais abstrata.

Registro

Um registro (struct) é um tipo composto por campos, sendo que cada um deles funciona como uma “sub-variável”. Tais campos podem possuir tipos diferentes e quando agrupados fazem sentido em um determinado contexto.

Podemos comparar um registro com uma ficha que possui todos os dados sobre uma determinada entidade, por exemplo:

- Cadastro de alunos (nome, matrícula, CR, endereço, etc.)
- Um sinal de áudio (número de amostras, taxa de amostragem, número de canais, um vetor de amostras, etc.)
- Uma imagem (altura, largura, formato, matriz de pixels, etc.)

Declaração

Primeiro, declaramos a `struct` como um tipo de dados. Depois, declaramos uma ou mais variáveis deste tipo.

Declarando o formato do registro

A primeira parte da criação de um registro é declarar seu formato. Isso é feito da seguinte forma:

```
typedef struct {  
    tipo_1 nome_1;  
    tipo_2 nome_2;  
    tipo_3 nome_3;  
    ...  
    tipo_n nome_n;  
} NomeDoTipoDoRegistro;
```

Atenção à convenção: *PalavrasIniciandoEmMaiusculas* (iniciar com *underline* também é utilizado).

Em um registro, cada campo tem um nome e os tipos de cada um deles **podem ser diferentes** (ao contrário do que ocorre com vetores e matrizes). Vetores/matrizes são homogêneos, registros são heterogêneos.

Declarando o formato do registro

Quase sempre, struct's são definidas fora de qualquer função (ou seja, com escopo global). Entretanto, é possível fazê-lo dentro de uma função, quando o tipo é usado somente naquela função.

Exemplo:

```
#include <stdio.h>
```

```
    /* Em geral, declare o formato de seu registro aqui */
```

```
int main ()
```

```
{
```

```
    /* Construa seu programa aqui */
```

```
    return 0;
```

```
}
```

Exemplo: declarando o formato do registro

O exemplo abaixo declara uma struct!! A partir deste momento, **é definido um novo tipo** chamado Estudante.

```
1  #include<stdio.h>
2  #define N 100
3
4  typedef struct{
5      int mat;
6      char nome[N];
7      float cr;
8  } Estudante;
9
10
11 int main(){
12     //codigo
13     return 0;
14 }
```

Atenção: a `struct` é um tipo, não uma variável!!!

Portanto, lembre-se: para armazenar as informações, é preciso declarar uma variável do tipo definido!

Exemplo: para armazenar informações para dois estudantes utilizando a `struct` definida no slide anterior, seria necessário **declarar duas variáveis do tipo** `Estudante`.

Lembre-se: assim que definimos a `struct` como um novo tipo, podemos criar variáveis, vetores, matrizes e tudo mais!

Declarando uma variável do tipo definido

A próxima etapa é declarar uma variável do tipo definido pela **struct**. Confira o exemplo:

```
1  #define N 100
2
3  typedef struct {
4      int mat;
5      char nome[N];
6      float cr;
7  } Estudante;
8
9  int main(){
10     Estudante fulano;
11     Estudante ciclano, beltrano;
12
13     // agora podemos usar a variavel fulano, a qual
14     // é do tipo Estudante
15
16     return 0;
17 }
```

Estudante é um **tipo**.

fulano, ciclano e beltrano
são **instâncias** de Estudante.

Acessando os campos de um registro

Utilizando os campos de um registro

- Os campos de um determinado registro são acessados individualmente como se fossem variáveis “normais”. Para tal, é utilizada a seguinte estrutura:

`nome_da_variavel.nome_do_campo`

- Para o registro do tipo Estudante declarado anteriormente, utilizaríamos

`fulano.mat`

para acessar o campo `mat` do registro `fulano` (note que usamos o nome da variável e não o nome dado ao formato do registro).

- Podemos colocar o campo de um registro em qualquer lugar onde colocaríamos uma variável.

Lendo os campos de um registro

A leitura dos campos de um registro a partir do teclado deve ser feita campo a campo, como se fossem variáveis independentes.

```
1  int main(){
2
3      Estudante x;
4
5      printf("Digite a matricula: ");
6      scanf("%d", &x.mat);
7
8      printf("Digite o cr: ");
9      scanf("%f", &x.cr);
10
11     printf("Digite o nome: ");
12     scanf("%s", x.nome);
13
14     return 0;
15 }
```

Escrevendo os campos de um registro

A escrita na tela do valor dos campos de um registro deve ser feita campo a campo, como se fossem variáveis independentes.

```
1  int main(){
2
3      Estudante x;
4
5      printf("Digite a matricula e o CR: ");
6      scanf("%d %f", &x.mat, &x.cr);
7
8      printf("A matricula eh %d e o cr eh %f", x.mat, x.cr);
9
10     return 0;
11 }
```


Copiando registros

Copiando registros

A cópia de um registro pode ser feita como se fosse a cópia de uma variável “normal”, ou seja, `registro1 = registro2`

```
1 int main(){
2
3     Estudante x1, x2;
4
5     printf("Digite a matricula, o nome e o CR: ");
6     scanf("%d %s %f", &x1.mat, x1.nome, &x1.cr);
7
8     x2 = x1;
9
10
11     printf("Dados de %s copiados.", x2.nome);
12     return 0;
13 }
```

como está lendo uma string,
observe que não tem o &

copia o conteúdo de todos os campos de
x1 para x2 (inclusive o vetor)

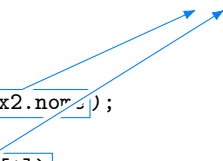
Exemplo: copiando registros cujos campos são vetores

```
1  typedef struct{
2      int mat;
3      char nome[N];
4      float notas[NPROVAS];
5  }Estudante;
6
7  void main(){
8      Estudante x1, x2;
9      int i;
10
11     x1.mat = 10;
12     strcpy(x1.nome, "Foolano de Tal");
13     x1.notas[0] = 75.5;
14     x1.notas[1] = 65.3;
15
16     x2 = x1;
17     printf("%d %s\n", x2.mat, x2.nome);
18     for(i=0; i<NPROVAS; i++)
19         printf("%.2f ", x2.notas[i]);
20 }
```

acessa cada índice do vetor notas



x2 passa a ter o
mesmo conteúdo que x1
em todos os campos




Struct's aninhadas

Struct's aninhadas

Pode-se também declarar uma struct como um dos campos de outra struct. Exemplo:

```
1  typedef struct {
2      int dia;
3      int mes;
4      int ano;
5  } Data;
6
7  typedef struct {
8      int mat;
9      char nome[N];
10     float cr;
11     Data nasc;
12 }Estudante;
13
```

Observe que um dos campos
é do tipo Data, o qual foi definido
anteriormente por uma struct !



Exemplo: Struct's aninhadas

Considerando as struct's definidas no slide anterior:

```
1  int main()
2  {
3      Estudante x;
4
5      x.mat = 109182;
6      strcpy(x.nome, "Fulano de Tal");
7      x.cr = 8.8;
8      x.nasc.dia = 20;
9      x.nasc.mes = 09;
10     x.nasc.ano = 1999;
11
12     return 0;
13 }
```

Observe a forma de acesso !



E o que fazer com as structs criadas?

Lembre-se: a struct é um tipo, não uma variável!!!

Portanto, lembre-se: podemos usar a struct definida para criar variáveis/*arrays*, representar entradas e retornos de funções, etc. Iremos discutir algumas dessas coisas no próximo material!