Notas de aula - 14/03

Structs

Na disciplina Fundamentos de Programação, estudamos estruturas de dados homogêneas: vetores, matrizes, strings. Dizemos que essas estruturas são homogêneas pois todos os dados armazenados nelas são do mesmo tipo. Por exemplo: em um vetor de tamanho 10, declarado com tipo int, todos os 10 elementos serão do tipo int.

As **estruturas heterogêneas**, são estruturas que podem guardar diferentes tipos de dados. Imagine o seguinte problema: preciso escrever um programa para armazenar dados sobre os alunos da UTFPR. Preciso guardar informações como nome do aluno, registro acadêmico (RA) e coeficiente de rendimento. O nome do aluno deve ser uma string. O RA pode ser do tipo int. Já o coeficiente de rendimento, deve ser do tipo float. Como posso criar uma estrutura para armazenar esses dados usando a linguagem C?

Na linguagem C, as estruturas heterogêneas são representadas pelo tipo *struct.* Uma struct tem a seguinte sintaxe:

```
struct nome_da_struct
{
    tipo_do_dado membro_1;
    tipo_do_dado membro_2;
    ...
    tipo_do_dado membro_n;
};
```

A keyword *struct* é usada para indicar que será definido um novo de tipo de dados. Ao lado dessa keyword podemos definir o nome desejado para a struct. Dentro do bloco da struct escrevemos as variáveis desejadas (da mesma maneira que declaramos variáveis na função main). Cada variável é chamada de membro ou campo da struct. Os membros da struct podem ser de qualquer tipo.

Exemplo:

```
struct aluno
{
      char nome[30];
      int RA;
      float coeficiente;
};
```

A declaração de uma estrutura não é uma definição. Não é alocada memória, simplesmente é introduzido um novo tipo de dados. Essa declaração é feita normalmente fora da função main, próxima ao topo do arquivo. Assim, todo o código poderá usar o novo tipo de dados.

```c
#include <stdio.h>

struct aluno
{
	char nome[30];
	int RA;
	float coeficiente;
};


main()
{
	printf("Hello world!");
}
```

Declarando uma variável de um tipo struct

Assim como na declaração de qualquer tipo de variável, é preciso informar o tipo do dado e seu identificador. O tipo do dado no exemplo acima será “struct aluno”:

```c
struct aluno aluno_1;
```

Ao declarar uma variável do tipo “struct aluno” será alocada memória suficiente para armazenar uma string de tamanho 30, um int e um float.

Typedef

Para simplificar os códigos, podemos usar a keyword *typedef*. Ela permite definir um novo nome para um tipo de dados. No exemplo abaixo é definido um novo nome para a struct aluno:

```c
typedef struct aluno Aluno;

struct aluno
{
	char nome[30];
	int RA;
	float coeficiente;
};
```

Assim, cada vez que for preciso declarar uma variável do tipo struct aluno, em vez de usar:

```c
struct aluno aluno_1;
```

basta usar:

```
Aluno aluno_1;
```

O uso de structs deixa o código mais sucinto e menos confuso. Imagine que você precisa guardar as informações de 3 alunos e ainda não conhece structs. Você precisaria fazer algo assim:

```
char nome_1[30], nome_2[30]; nome_3[30];
int RA_1, RA_2, RA_3;
float coeficiente_1, coeficiente_2, coeficiente_3;
```

Você também precisaria tomar cuidado para sempre acessar os dados do aluno certo. Não pegar nome_1, RA_1 e coeficiente_2 por engano, por exemplo. Usando structs, o código fica mais sucinto e os dados de cada aluno ficam “empacotados” em um só lugar, facilitando o acesso:

```
Aluno aluno_1, aluno_2, aluno_3;
```

Acessando os membros de uma struct

O acesso dos membros de uma struct se dá através do operador ponto (.). Considere o exemplo da struct aluno:

```
main()
{
	Aluno aluno_1;
	aluno_1.coeficiente = 0.925;
	aluno_1.RA = 123;
	strcpy(aluno_1.nome, "Maria");

	printf("Nome: %s\n", aluno_1.nome);
	printf("RA: %d\n", aluno_1.RA);
	printf("Coeficiente: %.4f\n", aluno_1.coeficiente);
}
```

ou ainda, fazendo a leitura dos dados:

```
main()
{
	Aluno aluno_1;
	printf("Nome: ");
	gets(aluno_1.nome);
	printf("RA: ");
	scanf("%d", &aluno_1.RA);
	printf("Coeficiente: ");
	scanf("%f", &aluno_1.coeficiente);

	printf("\nNome: %s\n", aluno_1.nome);
	printf("RA: %d\n", aluno_1.RA);
	printf("Coeficiente: %.4f\n", aluno_1.coeficiente);
}
```

Inicialização de uma struct

Assim como qualquer outra variável do tipo, int, float, char, etc., é possível inicializar uma struct no momento da declaração. Essa inicialização é parecida com a inicialização de um vetor. Por exemplo:

```
main()
{
	Aluno aluno_1 = {"Maria", 123, 0.925};

	printf("\nNome: %s\n", aluno_1.nome);
	printf("RA: %d\n", aluno_1.RA);
	printf("Coeficiente: %.4f\n", aluno_1.coeficiente);
}
```

Lembrando que neste caso, os valores precisam estar na mesma ordem de declaração dos membros da estrutura.

Atribuição entre structs

É possível atribuir uma struct diretamente em outra somente quando as duas structs forem do mesmo tipo. Cada campo da primeira struct será copiado em seu campo correspondente na segunda struct.

```
main()
{
	Aluno aluno_1, aluno_2;

	strcpy(aluno_1.nome, "Maria");
	aluno_1.RA = 1;
	aluno_1.coeficiente = 0.925;

	aluno_2 = aluno_1;

	printf("Nome: %s\n", aluno_2.nome);
	printf("RA: %d\n", aluno_2.RA);
	printf("Coeficiente: %.4f\n", aluno_2.coeficiente);
}
```

Considere agora o exemplo abaixo:

```
typedef struct aluno Aluno;
typedef struct estudante Estudante;

struct aluno
{
	char nome[30];
	int RA;
	float coeficiente;
};

struct estudante
{
	char nome[30];
	int RA;
	float coeficiente;
};

main()
{
	Aluno aluno_1;
	Estudante aluno_2;

	strcpy(aluno_1.nome, "Maria");
	aluno_1.RA = 1;
	aluno_1.coeficiente = 0.925;

	aluno_2 = aluno_1;

	printf("Nome: %s\n", aluno_2.nome);
```

```
	printf("RA: %d\n", aluno_2.RA);
	printf("Coeficiente: %.4f\n", aluno_2.coeficiente);
}
```

Um erro será mostrado:
error: incompatible types when assigning to type ‘Estudante {aka struct estudante}’ from type ‘Aluno {aka struct aluno}’|

Apesar de as estruturas aluno e estudante serem iguais internamente, elas são tipos de dados diferentes e não é possível fazer essa atribuição.

Arrays de struct

É possível criar arrays de struct, assim como é feito em qualquer outro tipo de dados. Ao percorrer o array de struct, primeiro deve-se acessar a posição desejada através do índice e depois acessar o campo desejado usando o operador ponto:

```
main()
{
	int i, n = 5;
	Aluno alunos[n];

	for(i = 0; i < n; i++)
	{
		printf("Nome: ");
		gets(alunos[i].nome);
		printf("RA: ");
		scanf("%d", &alunos[i].RA);
		printf("Coeficiente: ");
		scanf("%f", &alunos[i].coeficiente);
	}

	for(i = 0; i < n; i++)
	{
		printf("\nNome: %s\n", alunos[i].nome);
		printf("RA: %d\n", alunos[i].RA);
		printf("Coeficiente: %.4f\n", alunos[i].coeficiente);
	}
}
```

Structs como argumentos de função

Assim como em qualquer outro tipo de dados, as structs podem ser passadas como parâmetros para funções.

```
void imprime_dados_aluno(Aluno a)
{
	printf("Nome: %s\n", a.nome);
	printf("RA: %d\n", a.RA);
	printf("Coeficiente: %.4f\n", a.coeficiente);
}
```

e a chamada da função:

```
main()
{
	int i, n = 5;
	Aluno alunos[n];

	for(i = 0; i < n; i++)
	{
		printf("Nome: ");
		gets(alunos[i].nome);
		printf("RA: ");
		scanf("%d", &alunos[i].RA);
		printf("Coeficiente: ");
		scanf("%f", &alunos[i].coeficiente);
	}

	for(i = 0; i < n; i++)
	{
		imprime_dados_aluno(alunos[i]);
	}
}
```

Structs como valor de retorno

Assim como em qualquer outro tipo de dados, é possível ter funções retornando uma struct.

```
Aluno cadastra_aluno()
{
	struct aluno a;
	printf("Nome: ");
	gets(a.nome);
	printf("RA: ");
	scanf("%d", &a.RA);
	printf("Coeficiente: ");
	scanf("%f", &a.coeficiente);
	return a;
}
```

e a chamada de função:

```
main()
{
	int i, n = 1;
	struct aluno alunos[n];

	for(i = 0; i < n; i++)
	{
		alunos[i] = cadastra_aluno();
	}

	for(i = 0; i < n; i++)
	{
		imprime_dados_aluno(alunos[i]);
	}
}
```

Exemplo de função que recebe e retorna uma struct:

```
Aluno atualiza_coeficiente_aluno(Aluno a, float coeficiente)
{
	a.coeficiente = coeficiente;
	return a;
}
```

e a chamada da função:

```
main()
{
	Aluno aluno_1;
	strcpy(aluno_1.nome, "Maria");
	aluno_1.RA = 1;
	aluno_1.coeficiente = 0.925;
	imprime_dados_aluno(aluno_1);

	aluno_1 = atualiza_coeficiente_aluno(aluno_1, 0.890);
	imprime_dados_aluno(aluno_1);
}
```

Aninhamento de structs

Como definido anteriormente, os membros de uma struct podem ser de qualquer tipo, portanto é possível ter uma struct como membro de outra. Considere o exemplo:

```c
#include <stdio.h>

typedef struct ponto Ponto;
typedef struct segmento_reta Segmento;

struct ponto
{
	int x;
	int y;
};

struct segmento_reta
{
	Ponto ponto_ini;
	Ponto ponto_fin;
};

main()
{
	Ponto ponto_a, ponto_b;
	Segmento reta;

	ponto_a.x = 4;
	ponto_a.y = 2;
	ponto_b.x = 6;
	ponto_b.y = 2;

	reta.ponto_ini = ponto_a;
	reta.ponto_fin = ponto_b;

	printf("Inicio da reta: %d, %d\n", reta.ponto_ini.x,
reta.ponto_ini.y);
	printf("Final da reta: %d, %d\n", reta.ponto_fin.x,
reta.ponto_fin.y);
}
```

Outro exemplo:

```c
#include <stdio.h>

typedef struct data Data;
typedef struct pessoa Pessoa;

struct data
{
	int dia;
	int mes;
	int ano;
};

struct pessoa
{
	char nome[30];
	struct data data_nascimento;
}

main()
{
	Pessoa contato_1;
	int idade;

	strcpy(contato_1.nome, "Maria");
	contato_1.data_nascimento.dia = 1;
	contato_1.data_nascimento.mes = 1;
	contato_1.data_nascimento.ano = 1990;

	idade = 2019 - contato_1.data_nascimento.ano;

	printf("Aniversario da Maria: %d/%d. Ela vai fazer %d anos.",
	contato_1.data_nascimento.dia, contato_1.data_nascimento.mes,
idade);

}
```

Links interessantes sobre structs

- Video aulas professor André Backes: https://www.youtube.com/watch?v=MatsUCe5uZw
- Notas de aula Profª. Carmem Hara e Prof. Wagner Zola: http://www.inf.ufpr.br/nicolui/Docs/Livros/C/ArmandoDelgado/notas-27_Estruturas.html