AULA 6

Tipos de Dados definidos pelo Programador

Metas da Aula

1. Entender os conceitos necessários para definir um tipo em linguagem C.
2. Aprender a utilizar o novo tipo de dado definido.
3. Aplicar o uso de definição do próprio tipo de dado em variadas aplicações.
4. Aprender a escrever programas em linguagem C que utilizam tipos de dados definidos pelo programador.

Ao término desta aula, você será capaz de:

1. Definir o próprio tipo de dado em linguagem C.
2. Determinar qual a melhor estrutura se aplica ao tipo de dado necessário ao problema.
3. Escrever programas que utilizam os tipos de dados definidos pelo programador.

6.1 Tipo de dado

Os tipos de dados aprendidos até o momento, como o **int** para armazenar números inteiros, o **float** para armazenar números reais, entre outros, conforme apresentado na tabela 1, são nativos da linguagem C e com limitações inerentes, visto serem tipos primitivos. Em geral a limitação está relacionada ao fato de que, com os tipos primitivos só é possível definir variáveis com dados homogêneos, por exemplo, com uma variável do tipo **int**, só é possível armazenar um número pertencente ao conjunto dos inteiros, com uma variável do tipo **char**, só é possível armazenar um caractere, desta forma, não existe, por exemplo, um tipo que permita armazenar um dado que seja composto por um tipo **int** em conjunto com um **char**.

Da mesma forma, temos outra limitação relacionada aos vetores e matrizes, que como já dito, são estruturas homogêneas, pois, nos exemplos vistos até agora, ao definir um vetor ou uma matriz, o tipo que cada célula recebe é primitivo e igual para todas as células. Ou seja, um vetor do tipo **int**, terá todas as células, ou seja, espaços de armazenamento, do tipo **int**, sendo assim uma estrutura homogênea. Contudo, no mundo real, são comuns os problemas que requerem o armazenamento e processamento de dados pertencentes ao mesmo contexto, mas que, tem natureza heterogênia. Exemplo, os dados de um cliente são algo comum, algo que toda empresa precisa armazenar e processar, agora imagine alguns dados básicos do cliente, como: nome (texto), CPF (conjunto de números inteiros combinado com caracteres especiais), data de nascimento (conjunto de números inteiros combinados com caracteres especiais), endereço (combinação de texto com números).

É fácil perceber que no mundo real frequentemente será necessário manipular dados heterogêneos, assim, as linguagens de programação, em geral, oferecem a possibilidade do programador definir os seus próprios tipos de dados, para que assim, possa confortavelmente resolver situações comuns no mundo real. O objetivo desta aula é explorar os conceitos necessários para dominar esta técnica. A seguir, entenda melhor sobre os registros ou estruturas de dados para avançar no conteúdo.

6.2 Estruturas de dados

Quando o programador necessita definir um tipo de dado heterogêneo, será necessário declarar uma estrutura de dados que é uma coleção de dados de quaisquer tipos, inclusive tipos já previamente definidos pelo programador, desta forma, é possível definir uma estrutura de dados heterogênea (BACKES, 2013, p. 145-146). Os dados que integram a estrutura são denominados de campos (EDELWEISS; LIVI, 2014, p. 294), conforme pode ser visto na figura 20. Note na figura 20, que o nome da estrutura é **funcionário** e os campos pertencentes à esta estrutura são: **cod**, **nome**, **salário**, **depto** e **cargo**. Naturalmente, cada campo tem um tipo específico, por exemplo, o campo **nome** armazena texto, já o campo **cod** armazena números, desta forma, esta estrutura **funcionário** é heterogênea.

Para entender bem o que são estes elementos, a estrutura de dados, o tipo de dado e a variável, basta fazer uma analogia, imagine que a estrutura seria o equivalente à planta de uma casa, e o tipo de dado seria uma planta padrão que seria utilizada para construir várias casas, e a variável definida a partir de uma estrutura ou um tipo de dado, seria a casa, ou seja, a estrutura de dados é a definição ou característica que terá o tipo de dado ou a variável, o tipo de dado é um padrão de estrutura e a variável é a referência para a memória alocada com as características do tipo definido pelo programador.

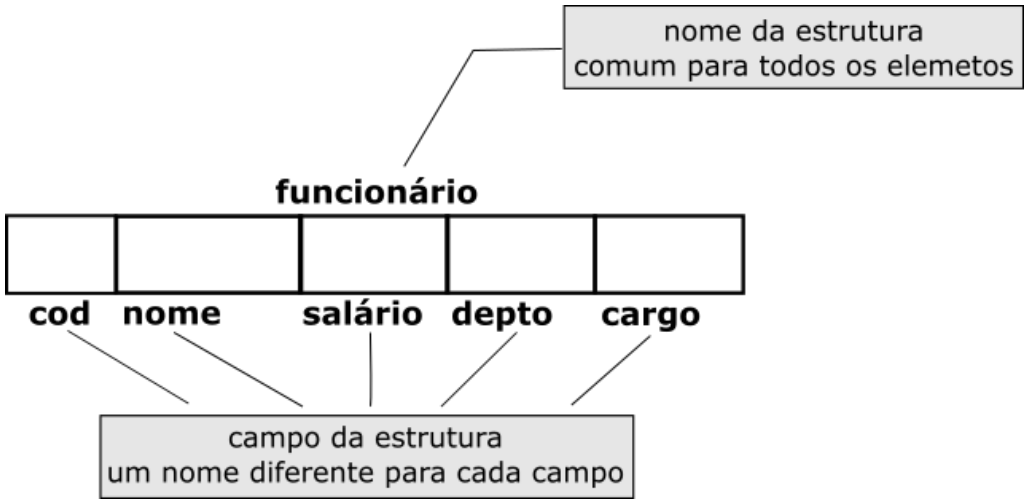

Figura 20 – Exemplo de estrutura

Fonte: Adaptado de (EDELWEISS; LIVI, 2014, p. 294)

6.3 Estruturas de dados em linguagem C

Como já foi mencionado, ao definir uma estrutura, na prática, é como se o programador definisse como será a característica do tipo de dado que ele pretende construir, desta forma, definir a estrutura não é definir o tipo de dado, contudo, a linguagem C nos permite utilizar a estrutura sem definir o tipo. Veja a seguir a sintaxe para definir uma estrutura em C.

```
1 //Sintaxe:
2 struct nome_estrutura {
3     tipo1 campo1;
4     tipo2 campo2;
5     ...
6     tipoN campoN;
7 };
```

Como pode ver na sintaxe, primeiro informa-se a palavra reservada **struct** para indicar o início de uma estrutura, na sequência, define-se o nome da estrutura que deve seguir o mesmo padrão de definição de nomes de variáveis, conforme aula 1. Os campos da estrutura são definidos entre chaves, conforme as linhas 2 e 7 da sintaxe, isso permite definir **N** campos para uma estrutura, no mínimo deve definir-se 1 campo e não há quantidade máxima. Cada campo é definido como se fosse uma variável, assim, informa-se o tipo e na sequência o nome, o tipo pode ser primitivo, conforme a tabela 1 ou um outro tipo definido pelo programador. Veja a seguir um exemplo de estrutura:

```
1 struct funcionario {
2     int cod;
3     char nome[30];
4     float salario;
5     int depto;
6     int cargo;
7 };
```

A estrutura de exemplo foi denominada de **funcionario** e os campos são **cod** do tipo **int**, **nome** do tipo **char** com tamanho 30, **salario** do tipo **float**, **depto** do tipo **int** e **cargo** do tipo **int**. Cada tipo e tamanho definido vai variar de acordo com a necessidade do problema, neste caso, por exemplo, **depto** e **cargo** foram definidos com o tipo inteiro, pois provavelmente pensou-se em armazenar apenas os códigos dessas informações, contudo, se a necessidade é a de armazenar apenas a descrição, então o tipo poderia ser **char**. Observe que neste exemplo foram definidos apenas campos com tipos nativos da linguagem C, mas ainda nesta aula serão apresentados os conceitos para definir campos de outras estruturas ou tipos customizados.

6.4 Variáveis e tipos de dados de estruturas

Após definir a estrutura, pode-se então utiliza-la como referência para definir uma variável ou um tipo de dado. Ambos os casos irão produzir "produtos"distintos, ao declarar uma variável de uma estrutura, é o mesmo que alocar espaço de memória com as características presentes na estrutura, por outro lado, ao definir um tipo de dado de uma estrutura, definiu-se um padrão a ser utilizado a partir daquela estrutura, desta forma, é como se estivesse dizendo: Esta estrutura será utilizada várias vezes! Assim, pode-se adotar como prática, que se a estrutura será utilizada poucas vezes, não há a necessidade de definir um tipo para ela, contudo, isso não é uma regra. Definir uma variável de uma estrutura é muito simples, basta substituir o tipo da variável pela palavra reservada **struct** seguido do nome da estrutura conforme o exemplo a seguir.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct funcionario {
5      int cod;
6      char nome[30];
7      float salario;
8      int depto;
9      int cargo;
10 };
11
12 void main()
13 {
14     struct funcionario func1, func2;
15 }
```

A definição da estrutura foi propositadamente repetida no exemplo, pois assim será mais claro o entendimento, note que o nome da estrutura na linha 4, **funcionario**, é idêntico ao nome da estrutura na definição da variável, na linha 14, isso permitirá à linguagem C determinar qual estrutura é desejada na definição da variável. Note também que foram declaradas duas variáveis para a mesma estrutura, no caso, **func1** e **func2**, o formato de declaração é o mesmo que a declaração de variáveis apresentado na aula 1, assim, basta separar por **","**, quando desejar declarar mais de uma variável para o mesmo tipo.

Após declarada a variável, a forma de uso é similar à de uma variável tradicional, a única diferença é que deve-se indicar o nome da variável e o campo, no qual, deseja-se armazenar o dado. Um ponto importante é que não é necessário que todos os campos da variável sejam preenchidos. O exemplo já apresentado foi complementado para mostrar como atribuir dados à variáveis de estrutura. Veja a seguir:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct funcionario {
    int cod;
    char nome[30];
    float salario;
    int depto;
    int cargo;
};

void main()
{
    struct funcionario func1, func2;

    //atribuindo dados a func1
    func1.cod = 1;
    strcpy(func1.nome, "Joao");
    func1.salario = 1200;
    //imprimindo os dados de func1
    printf("Codigo: %d \n", func1.cod);
    printf("Nome: %s \n", func1.nome);
    printf("Salario: %f \n", func1.salario);

    //atribuindo valores a func2 com o uso do scanf
    printf("Informe o codigo: \n");
    scanf("%d", &func2.cod);
    printf("Informe o nome: \n");
    scanf("%s", &func2.nome);
    printf("Informe o salario: \n");
    scanf("%f", &func2.salario);
    //imprimindo os dados de func2
    printf("Codigo: %d \n", func2.cod);
    printf("Nome: %s \n", func2.nome);
    printf("Salario: %f \n", func2.salario);
}
```

Como pode ver, até a linha 15 o programa é igual ao anterior, agora note a linha 18, veja que é uma atribuição direto no código-fonte, e como já mencionado foi indicado a variável, no caso **func1** e o campo, no caso **cod**, ambos foram associados por um ponto, **"."**, é assim que deve-se indicar o campo presente na variável, veja agora a linha 22, notou? Neste caso, foi impresso o valor do campo **cod** da variável **func1**. Mas, e como seria o uso do **scanf**? Não muda nada, a não ser o fato de que deve ser indicado o campo da variável, veja a linha 28, foi utilizado o **scanf** para ler um valor e armazenar no campo **cod** da variável **func2**. Assim, basicamente, o que muda ao fazer atribuições ou obter valor de variáveis com campos, é o fato de que o campo deve ser informado.

Podemos utilizar a estrutura para definir **"um padrão"**, neste caso, definir um tipo de dado a partir da estrutura, para fazer isso em linguagem C, utilizamos o operador **typedef**, que permite definir tipos com base em outros tipos. Veja a seguir a sintaxe:

```c
//Sintaxe:
typedef tipo_existente novo_nome;
```

Com o **typedef** pode-se por exemplo criar um tipo básico apenas mudando o nome, por exemplo, suponha que você, por algum motivo gostaria de indicar os tipos das variáveis com nomes em português, ao invés das abreviações em inglês nativas do C, então você poderia fazer algo como a seguir:

```
1 typedef int inteiro;
2 typedef float real;
3 typedef char caractere;
```

Notou no exemplo? Foram criados os tipos **inteiro**, **real** e **caractere** a partir dos tipos já existentes. Ao fazer isso, você pode utilizar os novos tipos definidos da mesma forma que aprendeu na aula 1. Mas, o nosso interesse não é esse, queremos utilizar o **typedef** para criar tipos heterogêneos. Desta forma, podemos ter um tipo de dado, padrão, definido para utilizar em nosso programa. Para definir funciona da mesma forma, substituindo o tipo pela palavra reservada **struct** acompanhado do nome da estrutura. Veja o exemplo a seguir.

```
1 struct funcionario {
2     int cod;
3     char nome[30];
4     float salario;
5     int depto;
6     int cargo;
7 };
8
9 typedef struct funcionario Funcionario;
```

Entre as linhas 1 e 7 foi definida a estrutura, cujo nome é **funcionario**, conforme a linha 1, na linha 9 foi definido o novo tipo, cujo nome é **Funcionario**, notou a diferença? Isso mesmo, como a linguagem C é *case sensitive*, utilizou deste fato para utilizar o mesmo nome da **struct** no **typedef**, alternando apenas a primeira letra para maiúsculo. Contudo, poderia ter sido utilizado qualquer nome para o **typedef**, desde que, siga as mesmas regras da definição do nome de variáveis.

Após definir um **typedef**, para utilizar basta seguir a mesma sintaxe na definição de variáveis e para utilizar a variável, utiliza-se o mesmo padrão adotado para variável de **struct**, para atribuição e obtenção de valor. Veja a seguir o exemplo adaptado.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  struct funcionario {
6      int cod;
7      char nome[30];
8      float salario;
9      int depto;
10     int cargo;
11 };
12
13 typedef struct funcionario Funcionario;
14
15 void main()
16 {
17     Funcionario func1, func2;
18
```

```
19      //atribuindo dados a func1
20      func1.cod = 1;
21      strcpy(func1.nome, "Joao");
22      func1.salario = 1200;
23      //imprimindo os dados de func1
24      printf("Codigo: %d \n", func1.cod);
25      printf("Nome: %s \n", func1.nome);
26      printf("Salario: %f \n", func1.salario);
27
28      //atribuindo valores a func2 com o uso do scanf
29      printf("Informe o codigo: \n");
30      scanf("%d", &func2.cod);
31      printf("Informe o nome: \n");
32      scanf("%s", &func2.nome);
33      printf("Informe o salario: \n");
34      scanf("%f", &func2.salario);
35      //imprimindo os dados de func2
36      printf("Codigo: %d \n", func2.cod);
37      printf("Nome: %s \n", func2.nome);
38      printf("Salario: %f \n", func2.salario);
39  }
```

Como pode ver no exemplo, entre as linhas 5 e 11 foi definida a estrutura **funcionario**, na linha 13 foi definido o tipo de dado **Funcionario** para a estrutura, na linha 17, as variáveis **func1** e **func2** foram declaradas com o novo tipo criado, **Funcionario**, note que neste caso, como é um tipo, não deve-se utilizar a palavra **struct**, nem mesmo **typedef**, contudo o nome do tipo criado deve ser exatamente igual para que o compilador consiga determinar quem é o novo tipo. Após declarar as duas variáveis, as linhas seguintes são exatamente iguais às do exemplo anterior, pois o formato de uso é idêntico.

6.5 Vetores de estruturas e tipos de dados

Ao utilizar as variáveis, mesmo sendo do tipo estrutura, há a limitação na quantidade de elementos a armazenar, exemplo, se declarar uma variável do tipo estrutura de **funcionario**, será possível armazenar vários dados do funcionário, mas apenas para 1 funcionário. Se é necessário armazenar mais de um funcionário, então é preciso utilizar vetores. Os vetores combinados com estrutura de dados heterogênea trazem muitos benefícios, pois ao definir um vetor de uma estrutura ou um tipo de dado, tiramos a limitação dele relacionada ao fato de que é uma estrutura homogênea e o mesmo assume as características de uma matriz heterogênea. Se ainda não consegue perceber isso com clareza, veja a figura 21.

Note que no exemplo apresentado na figura, primeiro foi definida a estrutura com nome **funcionario**, depois foi definido vetor **vetFunc** do tipo **struct funcionario** e o resultado do vetor é apresentado na parte inferior da figura 21, note que cada célula do vetor possui uma estrutura do tipo **funcionario**, que por sua vez comporta áreas distintas de armazenamento com tipos distintos, sendo assim, heterogênea, desta forma, o vetor passa a ter as características de uma matriz heterogênea.

Para declarar um vetor do tipo de uma estrutura basta seguir a mesma regra de declaração de vetores considerando que, ao invés de indicar um tipo nativo do C, indique o tipo definido pelo programador ou a estrutura, exemplo, ao invés de indicar **int**, indique **struct funcionario**. Veja a seguir um exemplo.

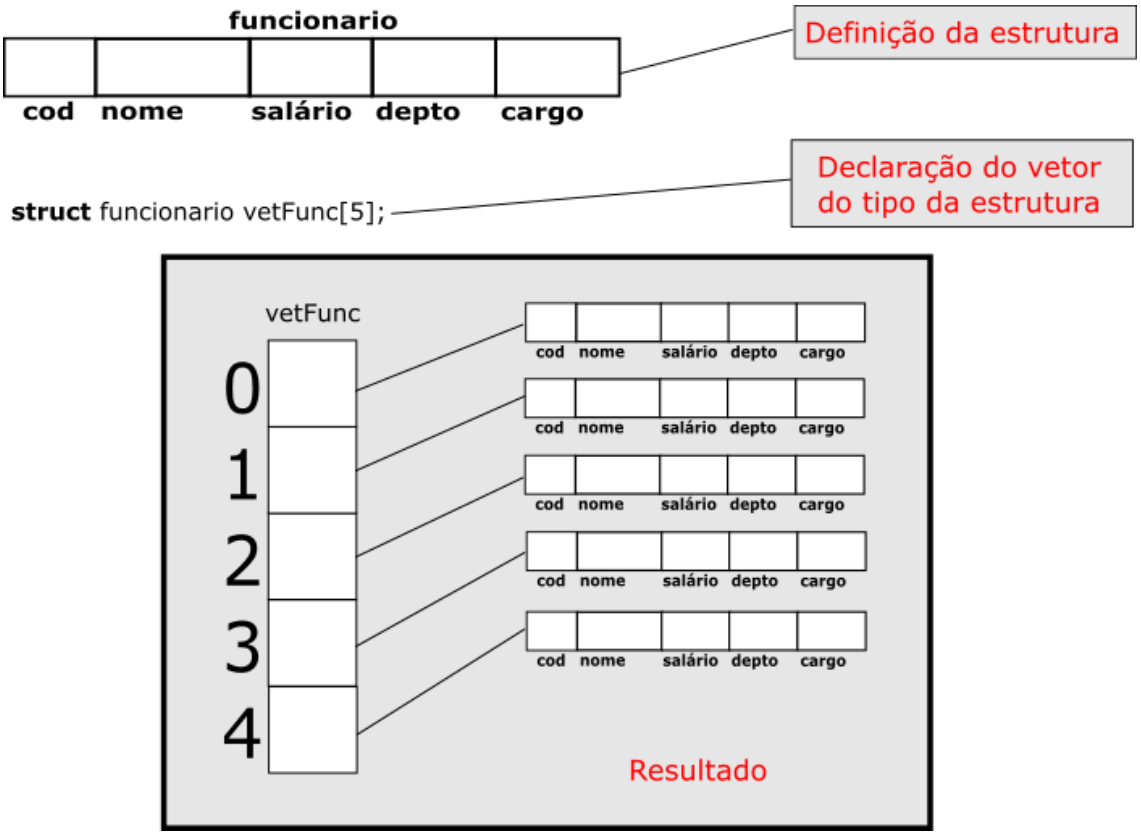

Figura 21 – Exemplo de vetor de estrutura

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct funcionario {
5      int cod;
6      char nome[30];
7      float salario;
8      int depto;
9      int cargo;
10 };
11
12 typedef struct funcionario Funcionario;
13
14 void main()
15 {
16     struct funcionario func1[5];
17     Funcionario func2[10];
18 }
```

No exemplo, veja que entre as linhas 4 e 10 foi definida a estrutura **funcionario**, na linha 12 foi definido o tipo **Funcionario**, na linha 16 foi declarado um vetor **func1** com tamanho 5 do tipo **struct funcionario** e na linha 17 foi declarado outro vetor **func2** com tamanho 10 e tipo **Funcionario**. Veja que é possível declarar o vetor tanto da estrutura quanto do novo tipo. A forma de uso da estrutura com vetor segue o mesmo padrão

adotado para as variáveis, contudo, deve-se indicar o índice. Veja a seguir o exemplo.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct funcionario {
5      int cod;
6      char nome[30];
7      float salario;
8      int depto;
9      int cargo;
10 };
11
12 typedef struct funcionario Funcionario;
13
14 void main()
15 {
16     Funcionario func[10];
17     int i;
18     for (i=0; i<10; i++) {
19         printf("Informe o codigo: \n");
20         scanf("%d", &func[i].cod);
21         printf("Informe o nome: \n");
22         scanf("%s", &func[i].nome);
23     }
24
25     for (i=0; i<10; i++) {
26         printf("Codigo: %d \n", func[i].cod);
27         printf("Nome: %s \n", func[i].nome);
28     }
29 }
```

Na linha 16 foi declarado o vetor de tamanho 10 e tipo **Funcionario**, entre as linhas 18 e 23 foi realizado um laço com **for** para ler os valores no vetor, para simplificar foram preenchidos apenas dois campos, o **cod** na linha 20 e o nome na linha 22, note que para atribuir o valor foi necessário indicar o o nome do vetor seguido do índice e do campo. Entre as linhas 25 e 28 foi realizado outro laço **for**, neste caso para imprimir os dados lidos, novamente foi necessário indicar o nome do vetor, seguido do índice do campo a ser impresso, conforme pode ser visto nas linhas 26 e 27.

6.6 Estruturas aninhadas

No início da aula foi mencionado que é possível declarar na estrutura campos que são tipo de outras estruturas previamente definidas, quando um estrutura possui entre seus campos tipos pertencentes a outras estruturas, damos o nome de **estruturas aninhadas** (BACKES, 2013, p. 152). Veja a seguir um exemplo.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct departamento {
5      int cod;
6      char descricao[30];
7  };
```

```
8
9  struct cargo {
10     int cod;
11     char descricao[30];
12 };
13
14 struct funcionario {
15     int cod;
16     char nome[30];
17     float salario;
18     struct departamento depto;
19     struct cargo cargo;
20 };
21
22 typedef struct funcionario Funcionario;
23
24 void main()
25 {
26
27 }
```

Veja que interessante, entre as linhas 4 e 7 foi declarada a estrutura **departamento**, entre as linhas 9 e 12 foi declarada a estrutura **cargo**, entre as linhas 14 e 20 foi declarada a estrutura **funcionario**, mas note que o campo **depto** na linha 18, não é mais do tipo **int**, agora ele é do tipo **struct departamento**, da mesma forma foi feito com o campo **cargo**, na linha 19, desta forma, temos um aninhamento de estruturas. Pode-se utilizar também o tipo (**typedef**) ao invés de **struct**. A seguir o mesmo exemplo adaptado para **typedef**.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct departamento {
5      int cod;
6      char descricao[30];
7  } Departamento;
8
9  typedef struct cargo {
10     int cod;
11     char descricao[30];
12 } Cargo;
13
14 typedef struct funcionario {
15     int cod;
16     char nome[30];
17     float salario;
18     Departamento depto;
19     Cargo cargo;
20 } Funcionario;
21
22 void main()
23 {
24
25 }
```

Dois pontos importantes são apresentados neste exemplo, primeiro note que, o **typedef** foi declarado em conjunto com o **struct**, tanto para a estrutura **departamento**, quanto para **cargo** e também para a estrutura **funcionario**, isso é útil apenas para resumir a codificação, mas não influi no desempenho da aplicação. O segundo ponto é a declaração dos campos **depto** e **cargo**, note que estes campos agora não são mais do tipo estrutura, mas dos tipos definidos **Departamento** e **Cargo**, respectivamente.

Desta forma, pode-se declarar campos de outros tipos definidos dentro de uma estrutura que também pode ser um tipo definido, além disso, não há uma limitação para o número de níveis na declaração aninhada, contudo, não seria prático incluir muitos níveis, pois isso irá complicar o entendimento para manutenção do código-fonte.

6.7 Resumo da Aula

Esta aula trouxe conceitos importantes sobre a declaração de tipos definidos pelo usuário com o uso de **struct** e **typedef**, destacou-se a importância desses tipos nos problemas computacionais tendo em vista que a grande maioria dos problemas requerem o uso de estruturas heterogêneas, o que não é possível com as variáveis e vetores declarados a partir dos tipos nativos da linguagem C.

Ao longo da aula foram apresentados os conceitos necessários para declarar e utilizar as estruturas de dados, em C **struct** e os tipos customizados, em C **typedef**. Ambos os recursos podem ser aplicados tanto com variáveis, bem como com vetores, o que traz um excelente benefício, pois o uso com vetores, permite obter uma estrutura similar à uma matriz heterogênea, o que não é possível com a declaração de uma matriz com tipos básicos.

6.8 Exercícios da Aula

Parte dos exercícios desta lista foram Adaptados de Backes (2013, p. 161-162).

1. Implemente um programa em C que leia o **nome**, a **idade** e o **endereço** de uma pessoa e armazene esses dados em uma estrutura. Em seguida, imprima na tela os dados da estrutura lida.

2. Crie uma estrutura para representar as coordenadas de um **ponto** no plano (posições X e Y). Em seguida, declare e leia do teclado um ponto e exiba a distância dele até a origem das coordenadas, isto é, posição (0, 0). Para realizar o cálculo, utilize a fórmula a seguir [1]:

$$d = \sqrt{(X_B - X_A)^2 + (Y_B - Y_A)^2} \tag{6.1}$$

   Em que:

   - d = distância entre os pontos A e B
   - X = coordenada X em um ponto
   - Y = coordenada Y em um ponto

3. Crie uma estrutura para representar as coordenadas de um **ponto** no plano (posições X e Y). Em seguida, declare e leia do teclado dois pontos e exiba a distância entre eles, considere a mesma fórmula do exercício anterior.

4. Cria uma estrutura chamada **retângulo**. Essa estrutura deverá conter o ponto superior esquerdo e o ponto inferior direito do retângulo. Cada ponto é definido por uma estrutura **Ponto**, a qual contém as posições X e Y. Faça um programa que declare e leia uma estrutura **retângulo** e exiba a área e o comprimento da diagonal e o perímetro desse retângulo.

5. Usando a estrutura **retângulo** do exercício anterior, faça um programa que declare e leia uma estrutura **retângulo** e um **ponto**, e informe se esse ponto está ou não dentro do retângulo.

6. Crie uma estrutura representando um aluno de uma disciplina. Essa estrutura deve conter o número de matrícula do aluno, seu nome e as notas de três provas. Defina também um tipo para esta estrutura. Agora, escreva um programa que leia os dados de cinco alunos e os armazena nessa estrutura. Em seguida, exiba o nome e as notas do aluno que possui a maior média geral dentre os cinco.

7. Crie uma estrutura representando uma hora. Essa estrutura deve conter os campos hora, minuto e segundo. Agora, escreva um programa que leia um vetor de cinco posições dessa estrutura e imprima a maior hora.

8. Crie uma estrutura capaz de armazenar o nome e a data de nascimento de uma pessoa. Faça uso de estruturas aninhadas e definição de novo tipo de dado. Agora, escreva um programa que leia os dados de seis pessoas. Calcule e exiba os nomes da pessoa mais nova e da mais velha.

[1] Dica: Se tiver dificuldade com os cálculos de potência e raiz, consulte a aula 7 que trata deste assunto

9. Crie uma estrutura representando um atleta. Essa estrutura deve conter o nome do atleta, seu esporte, idade e altura. Agora, escreva um programa que leia os dados de cinco atletas. Calcule e exiba os nomes do atleta mais alto e do mais velho.

10. Usando a estrutura "atleta"do exercício anterior, escreva um programa que leia os dados de cinco atletas e os exiba por ordem de idade, do mais velho para o mais novo.

11. Escreva um programa que contenha uma estrutura representando uma data válida. Essa estrutura deve conter os campos dia, mês e ano. Em seguida, leia duas datas e armazene nessa estrutura. Calcule e exiba o número de dias que decorrem entre as duas datas.

12. Astolfolov Oliveirescu é técnico de um time da série C do poderoso campeonato de futebol profissional da Albânia. Ele deseja manter os dados dos seus jogadores guardados de forma minuciosa. Ajude-o fazendo um programa para armazenar os seguintes dados de cada jogador: nº da camisa, peso (kg), altura (m) e a posição em que joga (atacante, defensor ou meio campista). Lembre-se que o time tem 22 jogadores, entre reservas e titulares. Leia os dados e depois gere um relatório no vídeo, devidamente tabulado/formatado.

13. Um clube social com 37 associados deseja que você faça um programa para armazenar os dados cadastrais desses associados. Os dados são: nome, dia, mês e ano de nascimento, valor da mensalidade e quantidade de dependentes. O programa deverá ler os dados e imprimir depois na tela. Deverá também informar o associado (ou os associados) com o maior número de dependentes.

14. Crie um programa que tenha uma estrutura para armazenar o nome, a idade e número da carteira de sócio de 50 associados de um clube. Crie também uma estrutura, dentro desta anterior, chamada **dados** que contenha o endereço, telefone e data de nascimento.

15. Crie um programa com uma estrutura para simular uma agenda de telefone celular, com até 100 registros. Nessa agenda deve constar o nome, sobrenome, número de telefone móvel, número de telefone fixo e e-mail. O programa deverá fazer a leitura e, após isso, mostrar os dados na tela.