

# REGISTROS

Essa aula teve como base o livro Treinamento em Linguagem C, de Viviane Victorine Mizrahi. Em caso de dúvidas ou aprofundamento consulte-o.

A aula apresenta a seguinte estrutura de conteúdo:

1. Struct;
2. Union;
3. Enum;
4. Typedef;
5. Typedef e struct.

O objetivo dessa aula é conhecer os principais conceitos de struct, union, enum e typedef na linguagem de programação C, e representá-los facilmente em diversos algoritmos para resolver problemas computacionais.

## TEMA 1 – STRUCT

Uma struct pode ser compreendida como um conjunto de variáveis referenciadas pelo mesmo nome, sendo que cada uma delas pode ter um mesmo tipo de dado, ou então vários.

A ideia básica por trás de uma struct é criar uma variável que contenha várias outras variáveis, ou seja, estamos criando uma variável que contém dentro de si outras variáveis.

Na linguagem de programação C podemos declarar tipos de variáveis como:

- Tipos básicos: char, int, float, double;  
☐ Exemplo: int x; float y;
- Tipos compostos homogêneos: array;  
☐ Exemplo: int x[5]; char nome[25].

Além dos tipos de dados mencionados acima, a linguagem de programação C permite a criação dos nossos próprios tipos de variáveis e um desses tipos é a estrutura ou struct.

A struct segue a seguinte sintaxe:

```

struct <nome_da_struct>
{
    <tipo 1> e <variável 1>;
    <tipo 2> e <variável 2>;
    <tipo 3> e <variável 3>;
    ...
    <tipo n> e <variável n>;
}
struct < nome_da_struct > <nome_variavel>;

```

A struct segue a seguinte sintaxe:

Figura 1 – Declaração de uma struct

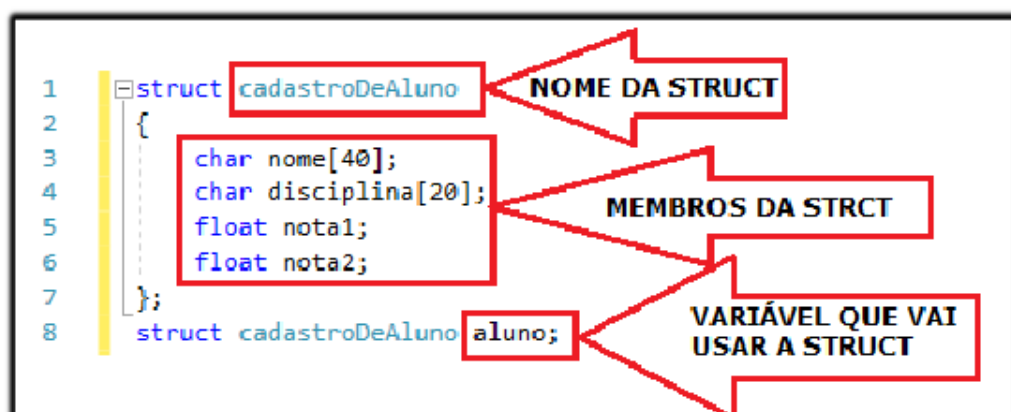
```

1 struct cadastroDeAluno
2 {
3     char nome[40];
4     char disciplina[20];
5     float nota1;
6     float nota2;
7 };
8 struct cadastroDeAluno aluno;

```

A Figura 2 mostra o algoritmo acima de forma detalhada.

Figura 2 – Declaração de uma struct



No exemplo acima temos o nome da struct (cadastroDeAluno), os membros que compõem a estrutura (char nome[40], char disciplina[20], float nota1 e float nota2) e a variável

que vai usar a struct (aluno). Dizemos que a variável aluno é do tipo cadastroDeAluno (struct cadastroDeAluno aluno). A Figura 3 traz um algoritmo com um exemplo de uma struct.

Figura 3 – Algoritmo com *struct*

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5
6      printf("\n CADATRO DE CLIENTE\n\n");
7
8      struct ficha_do_cliente /*Criando a struct */
9      {
10         char nome[50];
11         char rua[50];
12         int telefone[11];
13         char email[40];
14     }; struct ficha_do_cliente cliente; /*Criando a variável aluno que
15                                         será do tipo struct ficha_de_aluno */
16
17     printf("\n***** Lendo dados do Cliente *****\n\n");
18
19     printf("Digite o nome do Cliente: ");
20     fflush(stdin);
21     fgets(cliente.nome, 50, stdin);
22
23     /*usaremos o comando fgets() para ler strings, no caso o nome do aluno e
24     a disciplina fgets (variável, tamanho da string, entrada) como estamos
25     lendo do teclado a entrada é stdin (entrada padrão),
26     porém em outro caso, a entrada também poderia ser um arquivo */
27
28     printf("Rua: ");
29     fflush(stdin);
30     fgets(cliente.rua, 30, stdin);
31
32     printf("Telefone: ");
33     scanf_s("%d", &cliente.telefone);
34
35     printf("E-mail: ");
36     fflush(stdin);
37     fgets(cliente.email, 30, stdin);
38
39     printf("\n***** Imprimindo os dados da struct *****\n");
40     printf("***** Nome....: %s", cliente.nome);
41     printf("***** Rua.....: %s", cliente.rua);
42     printf("***** Telefone: %d", cliente.telefone);
43     printf("\n***** E-mail...: %s", cliente.email);
44     printf("\n\n");
45     getch();
46     return 0;
47 }
```

Figura 4 – Saída do algoritmo

Figura 4 – Saída do algoritmo

```
CADASTRO DO CLIENTE

***** Cadastro do Cliente *****

Digite o nome do Cliente: SANDRO DE ARAUJO
Rua: CAMPUS GARCEZ
Telefone: 999999999
E-mail: sandro.ar@uninter.com

***** Lendo os dados da struct *****
***** Nome....: SANDRO DE ARAUJO
***** Rua....: CAMPUS GARCEZ
***** Telefone: 6487540
***** E-mail...: sandro.ar@uninter.com

Pressione qualquer tecla para continuar. . .
```

## TEMA 2 – UNION

Com a union pode-se criar variáveis capazes de suportar dados diferentes, alocados no mesmo espaço de memória, em momentos diferentes. Isto é, a union permite que um conjunto de variáveis compartilhem o mesmo espaço na memória

Declara-se uma union de forma muito semelhante à uma struct; estas só se diferem no aspecto da struct ser alocada com espaço suficiente para todos os objetos, e o union só aloca espaço para o maior objeto que o compõe. Esse espaço alocado é suficiente para armazenar o maior dos seus membros.

Portanto, uma union define um conjunto de membros que serão armazenados numa porção compartilhada da memória, isto é, apenas um membro será armazenado de cada vez.

A sintaxe da union é similar à da struct, conforme mostrado a seguir:

```
union <nome_da_union>
{
    <tipo 1> e <variável 1>;
    <tipo 2> e <variável 2>;
    <tipo 3> e <variável 3>;
    ...
    <tipo n> e <variável n>;
};
```

A declaração começa com a palavra union seguida do identificador da união.

A primeira linha da declaração indica ao compilador que este é um novo tipo de dados para ser usado em outras partes do programa. Após as chaves, declara-se as variáveis que vão compor a union.

No exemplo da Figura 5 vamos trabalhar com números, porém não sabemos se os dados serão do tipo int ou float. Portanto, declara-se um novo tipo de dados, que iremos chamar de numeroFlex, capaz de armazenar valores de um dos dois tipos:

Figura 5 – Declaração de uma union

```
1 union numeroFlex {  
2     int num1;  
3     float num2;  
4 };
```

Ao acessar uma variável de tipo union, precisamos também indicar qual a variável que desejamos acessar. A sintaxe é a mesma que aquela para o acesso de membros em uma struct. Conforme exemplo na Figura 6 que vai declarar e atribuir um valor inteiro à variável num:

Figura 6 – Declaração e atribuição de uma union

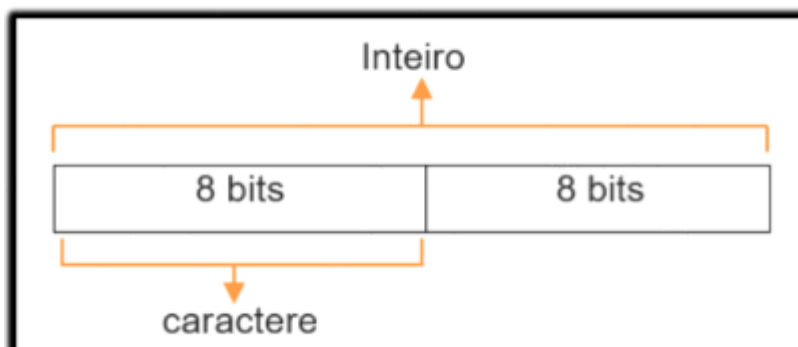
```
1 union numeroFlex num;  
2 num.num1 = 10;
```

Para atribuir um valor real à variável num usou-se a seguinte instrução:

```
num.num2 = 4.0;
```

A particularidade da union está no fato de que as variáveis num.num1 e num.num2 vão compartilhar a mesma posição na memória, e a atribuição de um valor fracionário sobrescreve o valor inteiro e vice-versa. Portanto, não há uma forma de “consultar” uma union e sobre qual tipo ela está armazenando em um dado momento. Tampouco existe uma forma para o programa decidir automaticamente a que variável da união ele deve atribuir um valor. Este controle fica por conta do programador. A Figura 7 ilustra o compartilhamento de espaço na memória. 7

Figura 7 – Compartilhamento da memória entre variáveis



A Figura 7 representa que se aloca a quantidade de armazenamento ocupada pelo maior membro da union. Com isso, a memória economiza espaço de armazenamento.

No próximo exemplo, na Figura 8, vamos trabalhar com um inteiro e um caractere. Para esse algoritmo declara-se um novo tipo de dado que iremos chamar de totalFlex, capaz de armazenar valores de um dos dois tipos (inteiro e caractere).

Figura 8 – Union com um inteiro e um caractere

```

1 union totalFlex {
2     int numero;
3     char letra;
4 };

```

Uma union torna o código fonte um pouco confuso e, portanto, deve ser utilizada em momentos em que as variáveis não sejam executadas no mesmo momento.

Para um melhor entendimento vejamos o algoritmo em que ele imprime o tamanho de uma union e struct, ambas com os mesmos tipos de variáveis (Figura 9).

Figura 9 – Compartilhamento da memória entre variáveis

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 union ex_union {
5
6     int inteiro_1;
7     char caractere_1;
8     float decimal_1;
9
10 };
11
12 struct ex_struct {
13
14     int inteiro_2;
15     char caractere_2;
16     float decimal_2;
17
18 };
19
20 int main()
21 {
22     printf("Tamanho da Union: %d\n", sizeof(union ex_union));
23     printf("Tamanho da Struct: %d\n", sizeof(struct ex_struct));
24
25     system("pause");
26     return 0;
27 }

```

A Figura 10 mostra a saída do algoritmo acima após a sua execução:

Figura 10 – Compartilhamento da memória entre variáveis

```
Tamanho da Union: 4
Tamanho da Struct: 12

Process returned 0 (0x0)   execution time : 0.115 s
Press any key to continue.
_
```

No exemplo da Figura 10 podemos verificar que a union ocupou um terço do espaço ocupado pela struct com o método de compartilhamento de espaço.

## TEMA 3 – ENUM

Enum ou enumeração, é um tipo de dado definido pelo usuário, com o uso de uma lista de identificadores. Os identificadores podem ser vistos como uma lista de constantes, onde cada constante tem um nome significativo.

A enum segue a seguinte sintaxe:

```
enum <nome_da_enum>{lista_de_identificadores}
```

Assim como em um vetor, uma estrutura enum também começa com o valor zero (0). Para adequar uma enum em um conjunto de constantes que referencie os meses do ano instanciamos o valor um (1) ao mês de janeiro, conforme o exemplo mostrado na Figura 11.

Figura 11 – Declaração e atribuição de uma enum

```
1 enum meses {
2     janeiro = 1, fevereiro, marco, abril, maio, junho, julho,
3     agosto, setembro, outubro, novembro, dezembro
4 };
```

A Figura 12 apresenta um exemplo com os dias da semana. Para esse exemplo não inicializamos nenhum membro da enum.

Figura 12 – Compartilhamento da memória entre variáveis

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  enum semana { domingo, segunda, terca, quarta, quinta, sexta, sabado };
4
5  int main() {
6
7      enum semana x; // Declaração de uma variável enum
8      x = quarta; //x recebe o valor 3
9      printf("Valor de quarta = %d\n\n", x); //imprime o identificador quarta
10
11      system("pause");
12      return 0;
13  }

```

A Figura 13 mostra a saída do algoritmo acima após a sua execução:

Figura 13 – Compartilhamento da memória entre variáveis

```

Valor de quarta = 3
Pressione qualquer tecla para continuar. . .

```

O identificador quarta apresentou o valor três (3). Para resolver esse problema devemos definir no enum semana uma enumeração em que é atribuído um identificador para a primeira constante com valor um (1); a enum, portanto, vai incrementar com o valor um (1) de forma sequencial os demais identificadores até o fim da lista. Com isso, a lista terá os valores de um a sete para os dias da semana.

Para melhor entendimento vejamos um algoritmo, na Figura 14, que vai executar o mesmo exemplo com dois identificadores instanciados.

Figura 14 – Operações com identificadores



```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  enum semana {
5      domingo = 1, segunda, terca, quarta, quinta = 8, sexta, sabado
6  };
7
8  int main() {
9
10     enum semana a, b, c, d, x, y, z, i;
11
12     //atribuição de um identificador para uma variável
13     a = domingo;
14     b = segunda;
15     c = terca;
16     d = quarta;
17     x = quinta;
18     y = sexta;
19     z = sabado;
20
21     i = y + z; //soma de dois identificadores
22
23     printf("Identificador de domingo = %d\n", a);
24     printf("Identificador de segunda = %d\n", b);
25     printf("Identificador de terca = %d\n", c);
26     printf("Identificador de quarta = %d\n", d);
27
28     printf("\nREINICIA OS IDENTIFICADORES\n\n");
29
30     printf("Identificador de quinta = %d\n", x);
31     printf("Identificador de sexta = %d\n", y);
32     printf("Identificador de sabado = %d\n", z);
33
34     printf("\nSoma dos Identificadores sexta e sabado = %d\n\n", i);
35
36     //comparação com identificador, se for true executa a instrução
37     if (b == segunda)
38         printf("Identificador b = a segunda\n\n");
39
40     system("pause");
41     return 0;
42 }

```

A Figura 15 mostra a saída do algoritmo acima após a sua execução.

Figura 15 – Operações com identificadores

```

Identificador de domingo = 1
Identificador de segunda = 2
Identificador de terca = 3
Identificador de quarta = 4

REINICIA OS IDENTIFICADORES

Identificador de quinta = 8
Identificador de sexta = 9
Identificador de sabado = 10

Soma dos Identificadores sexta e sabado = 19

Variavel b = segunda

Pressione qualquer tecla para continuar. . . █

```

O exemplo da Figura 9 – o identificador quinta – também foi instanciado. Repare ainda que na Figura 10 o identificador sexta seguiu a sequência, apresentando o valor nove (9).

Podemos também atribuir valores da tabela ASCII para enumeração, conforme mostrado no algoritmo da Figura 16.

Figura 16 – Escapes com identificadores

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  enum escapes { backspace = '\b', nova_linha = '\n', tabulacao_h = '\t' };
5
6
7  int main()
8  {
9      enum escapes esc = nova_linha;
10     printf("Testando %c%c%c de %c%c%c", esc, esc, esc, esc, esc, esc);
11
12     esc = tabulacao_h;
13     printf("Testando%cTestando%cTestando\n\n", esc, esc);
14
15     system("pause");
16     return 0;
17 }

```

A Figura 17 mostra a saída do algoritmo acima após a sua execução:

Figura 17 – Escapes com identificadores

```
Testando  
  
de  
  
Testando      Testando      Testando  
Pressione qualquer tecla para continuar. . .
```

A Figura 18 traz outro exemplo com estrutura enum.  
Figura 18 – Operações com identificadores

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void mostrarRes(int pais);
5
6  //Aqui os valores Italia = 4 e USA = 5
7  enum { FRANÇA = 3, ITALIA, USA };
8
9  int main()
10 {
11     int n = USA;
12     mostrarRes(n);
13
14     system("pause");
15     return 0;
16 }
17
18 void mostrarRes(int pais)
19 {
20     printf(" Para onde irei? :-) \n \n");
21
22     switch (pais)
23     {
24     case USA: printf("New York me aguarde!\n");
25             break;
26     case FRANÇA: printf("Paris cidade apaixonante.\n");
27             break;
28     case ITALIA: printf("Florença teem a melhor pizza.\n");
29             break;
30     default: printf("Preciso viajar.\n");
31
32     }
33 }

```

A Figura 19 mostra a saída do algoritmo acima após a sua execução:  
 Figura 19 – Operações com identificadores

```

Para onde irei? :-)

New York me aguarde!
Pressione qualquer tecla para continuar. . .

```

## TEMA 4 – TYPEDEF

O comando typedef é usado para criar “sinônimo” ou um “alias” para tipos de dados existentes. Na prática podemos dizer que estamos renomeando um tipo de dados. Essa renomeação de tipos facilita a organização e o entendimento do código. Sintaxe:

typedef ;

É importante ressaltar que o comando typedef não cria um novo tipo. Ele apenas permite que um tipo existente seja denominado de uma forma diferente, de acordo com a especificação desejada pelo programador, conforme mostrado no algoritmo da Figura 20:

Figura 20 – Renomeação de tipo de variável com typedef

```
1  #include <stdio.h>
2  #include <conio.h>
3
4  int main()
5  {
6      //redefinição do tipo float para o tipo prova
7      typedef float prova;
8
9      // variáveis usando o tipo prova
10     prova nota1, nota2, media;
11
12     printf("Digite a primeira nota: ");
13     scanf_s("%f", &nota1);
14
15     printf("Digite a segunda nota: ");
16     scanf_s("%f", &nota2);
17
18     media = (nota1 + nota2) / 2;
19
20     printf("Media = %.2f\n", media);
21
22     system("pause");
23     return 0;
24 }
```

A Figura 21 mostra a saída do algoritmo acima após a sua execução:

Figura 21 – Renomeação de tipo de variável com typedef

```
Digite a primeira nota: 9
Digite a segunda nota: 6

Media = 7.50

Pressione qualquer tecla para continuar. . . _
```

## TEMA 5 – TYPEDEF E STRUCT

É muito frequente o uso de typedef para criar apelidos a fim de tornar os nomes mais curtos, desta forma podemos representar uma estrutura usando apenas seu sinônimo (Figura 22).

Figura 22 – Typedef e struct

```

1  # include <stdio.h>
2  # include <stdlib.h>
3
4  typedef float prova; //redefinindo float
5  typedef int RU; //redefinindo int
6
7  struct notasAluno
8  {
9      RU matricula; //apelidos dentro da struct
10     prova nota1;
11     prova nota2;
12 }; typedef struct notasAluno n_aluno; //criando um apelido para a struct
13
14 int main()
15 {
16     n_aluno aluno; // Não é mais necessário escrever "struct n_aluno"
17     prova media = 0;
18
19     printf("Digite a matricula do aluno: ");
20     scanf_s("%d", &aluno.matricula);
21
22     printf("Digite a primeira nota: ");
23     scanf_s("%f", &aluno.nota1);
24
25     printf("Digite a segunda nota: ");
26     scanf_s("%f", &aluno.nota2);
27
28     media = (aluno.nota1 + aluno.nota2) / 2;
29
30     printf("\nMatricula do aluno: %d\n", aluno.matricula);
31     printf("Media das duas notas: %.2f\n\n", media);
32
33     system("pause");
34     return 0;
35 }

```

A Figura 23 mostra a saída do algoritmo acima após a sua execução:  
 Figura 23 – Typedef e struct

```

Digite a matricula do aluno: 8923578
Digite a primeira nota: 9
Digite a segunda nota: 8

Matricula do aluno: 8923578
Media das duas notas: 8.50

Pressione qualquer tecla para continuar. . . _

```

A Figura 24 traz outro exemplo de typedef com struct.  
 Figura 24 – Typedef e struct

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  typedef struct {
6      char nome[30];
7      } p;
8
9  int main(void) {
10     p x, y;
11
12     strcpy_s(x.nome, "Centro Universitario");
13     strcpy_s(y.nome, " UNINTER\n\n");
14
15     printf("%s", x.nome);
16     printf("%s", y.nome);
17
18     system("pause");
19     return 0;
20 }
21

```

A Figura 25 mostra a saída do algoritmo acima após a sua execução:  
 Figura 25 – Typedef e struct

```

Centro Universitario UNINTER
Pressione qualquer tecla para continuar. . .

```

## FINALIZANDO

Nesta aula aprendemos os principais conceitos referente a struct, union, enum e typedef na linguagem de programação C, e também como representá-los facilmente em algoritmos nas diferentes estruturas para resolver problemas computacionais.

Aproveite a disciplina e bons estudos!



# REFERÊNCIAS

MIZRAHI, V. V. Treinamento em Linguagem C. [S.l.]: Edição da autora, 2008