C ANSI

# Laboratório de Programação

2

# Estruturas, Uniões e Enumerações

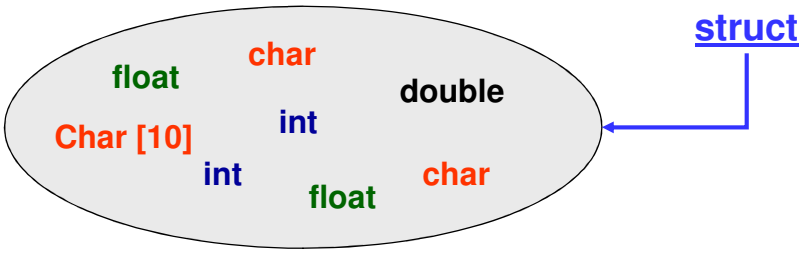


C ANSI

## Estruturas

**Definição:**

- É a definição de um nome para um agrupamento de variáveis, podendo ser de tipos diferentes.
  - informações de tipos diferentes compondo uma informação lógica única



C ANSI

# Estruturas

- Características
  - Algumas linguagens (como o Pascal), referem-se a este tipo de construção como sendo *records* (registros).
  - As variáveis que compõem uma estrutura são denominadas membros ou campos da estrutura.
  - Um exemplo típico de uma estrutura é o caso dos dados de uma lista de endereçamento postal. Cada elemento da lista conteria informações para o nome, endereço (rua, número, etc.), cidade, estado, CEP.

3

C ANSI

# Estruturas

- **Declarando:**
  - **struct rótulo {lista de declarações} var_e;**
  - A definição do rótulo (nome da estrutura) para estrutura é opcional;
  - Os operadores "**.**" e "**->**" dão acesso aos membros da estrutura.
- **Exemplos:**

```
struct Ponto {
  int x;
  int y;
};
```

A

```
struct {
  int x;
  int y;
}ponto;
```

B

```
struct Ponto {
  int x;
  int y;
}p1, p2, p3;
```

C

C ANSI

# Estruturas

```
struct Ponto {
        int x, y;
};
```

A

*É uma estrutura com rótulo definido (Ponto) e dois membros do tipo inteiro (x e y).*

- **Declaração:**
  - **Variável:** **struct Ponto ponto, p1;**
  - **Array:** **struct Ponto pontos[10];**
  - **Apontador:** **struct Ponto *ptPonto;**

5

C ANSI

# Estruturas

```
struct {
        int x, y;
}ponto;
```

B

*É uma estrutura sem rótulo definido, dois membros do tipo inteiro (x e y) e uma variável (ponto).*

- **Declaração:**
  - **Variável:** **Só pode usar ponto;**
  - **Array:** **Não pode ser declarado;**
  - **Apontador:** **Não pode ser declarado.**

6

C ANSI

# Estruturas

```
struct Ponto {
        int x, y;
}p1, p2, p3;
```

C

*É uma estrutura com rótulo definido (Ponto), dois membros do tipo inteiro (x e y) e três variáveis (p1, p2, p3).*

- **Declaração:**
  - **Variável: =A, além de p1, p2 e p3;**
  - **Array: =A: struct Ponto pontos[10];**
  - **Apontador: =A: struct Ponto *ptPonto;**

7

C ANSI

# Estruturas

- **Inicializando:**
  - Coloca-se uma lista de valores, separados por vírgulas, delimitado por { e };
  - Se o número de inicializadores for inferior ao número de membros, os que restarem automaticamente serão inicializados com 0 ou NULL, dependendo do seu tipo;
  - **Não faça comparação direta com estruturas !**
- **Exemplo:**
  - struct Ponto p1 = {5, 6}; **// x = 5 e y = 6**
  - struct Ponto p2 = {5}; **// x = 5 e y = 0**
  - p2 = p1; **// copia dados de p1 em p2**

8

C ANSI

# Estruturas

- **Inicialização de estruturas**
  - Pode-se também inicializar vetores de estruturas. Veja o exemplo abaixo:

```
struct endereco dados[] = {
  {"Júnior", "Rua da Praça, S/N", "João Pessoa", "PB",
  58025},
  {"Marta", "Rua da Matriz, S/N", "João Pessoa", "PB",
  58035},
  {"Daniel", "Praça Boa Vista, S/N", "João Pessoa",
  "PB", 58045}
};
```

  - Este exemplo cria um vetor chamado dados, contendo 3 elementos do tipo `struct endereco`, cujos valores iniciais são aqueles estabelecidos na inicialização.

9

C ANSI

# Estruturas

- Referenciando elementos de uma estrutura
  - **Operador ponto (.)**

**Uso:** `nome da variável estrutura + "." + campo individual`

```
main(){
   struct Ponto p1;
   p1.x = 5;
   p1.y = 4;
   printf("o ponto x tem valor %d",p1.x);
}
```

10

C ANSI

# Estruturas

- Referenciando elementos de uma estrutura
  - Atribuição

```
main() {
  struct Ponto p1, p2;
  p1.x = 5;
  p1.y = 4;
  p2 = p1;
  printf("o ponto x em p2 possui valor %d",
          p2.x);
}
```

11

C ANSI

# Estruturas

- Para referenciar os elementos de um vetor de estruturas, indexa-se o vetor e em seguida adiciona-se o ponto ( . ) e o nome do membro a ser manipulado.

```
struct rua {
  int num;
  char cep[8];
}
struct rua logradouro[10];

...

printf("CEP = %s", logradouro[1].cep);
```

12

C ANSI

# Estruturas

- Operador seta **->**
  - **Operador utilizado para acessar os elementos de uma estrutura através de um ponteiro.**

```
main(){
   struct Ponto p1, *p2;
   p1.x = 5;
   p1.y = 4;
   p2 = &p1;
   p2->x = 10; p2->y = 20;
   printf("p1.x = %d", p1.x);
}
```

13

C ANSI

# Estruturas e funções

- **As estruturas podem ser passadas como parâmetro na chamada das funções:**
  - Cada membro separadamente;
  - A estrutura inteira;
  - Apontador para uma estrutura
- **Atenção!**
  - A passagem da estrutura é por valor!
  - Passagem de array, por default, é feita por referência
    - Array por valor : **struct** com um vetor
  - Uma função pode retornar um tipo estrutura

14

C ANSI

# Estruturas

- **Exemplo 1**

  Faça um programa que leia do usuário as coordenadas x e y de um ponto; depois crie uma função para informar em qual quadrante o ponto se encontra

- Vamos precisar:
  - Estrutura para representar um ponto
  - Função para encontrar o quadrante do ponto
  - Programa principal (main) para testar

15

C ANSI

# Estruturas

- **Exemplo 2**

  Faça um programa que use um procedimento para inicializar as coordenadas de um ponto com os valores "x=10" e "y=10";

- Vamos precisar:
  - Estrutura para representar um ponto
  - Procedimento para iniciar as coordenadas
  - Programa principal (main) para testar

16

C ANSI

# Estruturas

- **Exemplo 3:**
  - Faça um programa em C para ler matricula, nome e salário de no máximo MAXEMP empregados. Devem ser lidos os dados de todos os empregados até ser digitada uma matrícula 0 (zero). Após a leitura dos dados deve ser mostrada a média salarial dos empregados e listados todos os empregados (nome, matrícula e salário), cujo salário está abaixo da média salarial calculada. Use uma macro para definir MAXEMP com valor 100;

17

C ANSI

# Estruturas

- **Outros Exemplos:**

**Funcionário:**

```
struct funcionario{
    char nome[31];
    int matricula;
    float salario;
};
```

**Aluno:**

```
struct aluno{
    char nome[31];
    int matricula;
    float nota1;
    float nota2;
    float nota3;
};
```

18

C ANSI

# Estruturas Aninhadas

- **Definição:**
  - Uma estrutura pode conter como membro uma outra estrutura.
- **Exemplo:**
  - Estrutura circunferência, composta por um ponto e um raio:

```
struct circunferencia{
    struct Ponto centro;
    float raio;
};
```

19

C ANSI

# Estruturas Aninhadas

- **Inicializando:**
  - Inicializa da mesma forma das estruturas simples.

```
struct circunferencia c1 = {1, 1};
```

- **Exemplos:**
  - struct circunferencia c1 = {1, 1, 5.6};
  - struct circunferencia c2 = {{1, 1}, 5.6};
  - struct circunferencia c3 = {1, 1};
  - struct circunferencia c4 = {{1, 1}};
  - struct circunferencia c5 = c1;

20

C ANSI

# Estruturas Aninhadas

- **Acessando os Membros:**

```
struct circunferencia{
        struct Ponto centro;
        float raio;
};
```

- **Variável: struct cincunferencia c;**
  - Coordenada x do centro (Ponto): **c.centro.x;**
  - Coordenada y do centro (Ponto): **c.centro.y;**
  - Raio da circunferencia: **c.raio;**

21

C ANSI

# Estruturas Aninhadas

- **Dada a seguinte situação:**
  - Seja a seguinte estrutura utilizada para descrever os nomes dos funcionários:

```
struct Nome {
     char nome[15];
    char sobreNome[25];
};
```

  - Seja a seguinte estrutura utilizada para descrever os empregados:

```
struct Empregado{
    struct Nome nome;
     int matricula;
};
```

22

C ANSI

# Estruturas Aninhadas

- **Exemplo 4:**
  - Faça um programa que leia, do usuário, os dados de 5 funcionários. No final imprima os nomes (nome e sobrenome) e matrículas desses funcionários:

- **Vamos Precisar:**
  - Armazenar os dados dos funcionários em um vetor;
  - Programa principal (main) para testar.

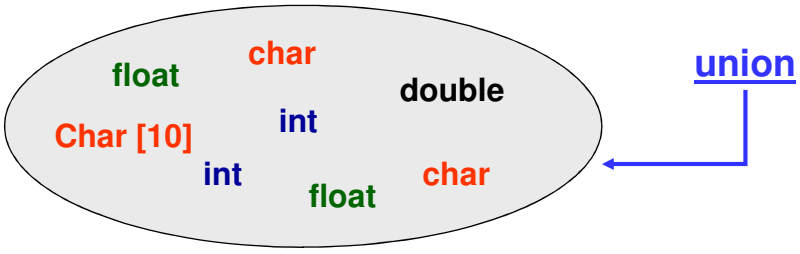
23

C ANSI

# Uniões

**Definição:**

- Semelhante às estruturas, com a diferença que os membros compartilham o mesmo espaço de armazenamento.



C ANSI

# Uniões

- **Declarando:**

```
union rótulo {
    lista de declarações
} lista_variáveis;
```

  - Obedecem regras sintáticas semelhantes às das estruturas.
    - Os operadores "**.**" e "**->**" dão acesso aos membros da union.

- **Exemplos:**

```
union Num{
   int x, float y;
};
```
A

```
union {
   int x, float y;
}num;
```
B

```
union Num{
   int x, float y;
}n1, n2, n3;
```
C

C ANSI

# Uniões

- Considerações:
  - O compilador aloca espaço o suficiente para conter o membro de maior tamanho da união
  - Apenas um dos campos pode ser considerado válido em um momento

```
num.x = 5;
num.y = 4.5;
```

    - ao final da segunda atribuição, o valor de **x** será perdido.
    - Não tem sentido acessar outros campos da union
    - **O programador é responsável por esse controle!**

26

C ANSI

# Uniões

- **Inicializando:**
  - Só pode ser inicializada com um valor do mesmo tipo que o do primeiro membro da união;
  - **Não se pode comparar duas estruturas do tipo union!**
    - **(u1 == u2) – Errado!**
- **Exemplo:**
  - union Num n1 = {5}; **// x = 5**
  - union Num n2 = {6.5}; **// Erro**
  - n2 = n1; **// copia dados de n1 em n2**

27

C ANSI

# Uniões

- Onde eu uso isso???
  - Implementação de registros variantes
    - Composto por uma **parte fixa** e outra **variável**

```
Endereco
  char rua[31]
  int numero
```

```
EstadoCivil
  SOLTEIRO
  CASADO
  DIVORCIADO
```

*enumeration*

```
struct Empregado {
   char nome[31];
   int matricula;
   Endereco  end;
   EstadoCivil estCivil;
   // int flag;
   union {
      char nomeConj[41];
      short int idade;
      char dataDivorcio[11];
   }
}emp;
```

28

C ANSI

# Uniões

- Exemplo de código:

```
...
if (emp.estCivil == CASADO) {
  printf("%s", emp.nomeConj);
} else if (emp.estCivil == SOLTEIRO) {
  printf("Idade: %d", emp.idade);
} else if (emp.estCivil == DIVORCIADO)
  printf("Dava do divorcio: %s",
          emp.dataDivorcio);
}
...
```

**Importante:** Se um registro variante não possui campo indicador, não existe nenhuma forma de se determinar qual campo variante está correntemente em uso.

29

C ANSI

# Enumerações

- Conceito
  - É um conjunto de constantes inteiras (valores) que uma variável pode assumir;
- Características
  - Cada constante inteira é representada por um identificador (uso de constantes simbólicas);
  - Os valores (inteiro) são atribuídos automaticamente às constantes;
  - Se não for especificado, os valores da enumeração começam com 0 (zero)

30

C ANSI

# Enumerações

- Definição de uma enumeração
  - Semelhantemente à definição de uma estrutura

```
enum <identificador> {
    lista de enumeração
} <lista de variáveis>;
```

- Exemplos:

```
/* definicao da enumeracao */
/* OBS: O valor do primeiro simbolo é ZERO */
enum meses {JAN,FEV,MAR,ABR,MAI,JUN,JUL,
                AGO,SET,OUT,NOV,DEZ};
/* declaracao de uma variavel do tipo "meses" */
enum meses mes;
```

31

C ANSI

# Enumerações

- Tipos de comandos válidos:

```
…
mes = JAN;
If (mes == JAN ) // JAN possui valor zero
   printf("Mes de Janeiro");
else if (mes == FEV ) // FEV possui valor 1
   printf("Mes de Fevereiro")
else if (… ) // E assim por diante
}
printf("%d e %d",MAR, JUN); // o que vai ser impresso?
…
```

**Obs:** cada símbolo representa um valor inteiro! Podem ser usados em qualquer lugar como um inteiro normal

32

C ANSI

# Enumerações

- Outros exemplos:

```
enum dias_da_semana {SEGUNDA = 1, TERCA,
  QUARTA, QUINTA, SEXTA, SABADO,
  DOMINGO} dia; /* Valores: 1,2,3,4,5,6 e 7,
  respectivamente */
enum boolean {FALSO, VERDADEIRO}; /*
  Valores: 0 e 1, respectivamente */
enum estacoes {VERAO, INVERNO, PRIMAVERA
  = 100 , OUTONO}; /* Valores: 0, 1, 100, 101,
  respectivamente */
```

- Não faça:

```
dia = SEGUNDA;
printf("%s", dia); /* Não vai imprimir "SEGUNDA" */
```

33

C ANSI

# Enumerações

- **Exemplo 5:**
  - Faça um programa que imprima os meses do ano. Use enumeração.

```
#include <stdio.h>

enum meses {JAN, FEV, MAR, ABR, MAI, JUN, JUL, AGO, SET, OUT, NOV, DEZ};

int main(void){
   char *nomeMes[] = {"Janeiro", "Fevereiro", "Março", "Abril", "Maio",
                      "Junho", "Julho", "Agosto", "Setembro", "Outubro",
                      "Novembro", "Dezembro"};
   enum meses mes;

   for(mes = JAN; mes <= DEZ; ++mes)
      printf("[%d] - %s\n", mes + 1, nomeMes[mes]);
}
```

34

C ANSI

# Typedef

- Permite ao programador definir novos nomes aos tipos de dados (**sinônimos**);
  - Definição de um novo nome a um tipo já existente
- **Sintaxe**

  **typedef** <tipo> *novo_nome*;
- **Exemplos:**
  - typedef int inteiro;
  - typedef float real;
  - typedef struct Ponto ponto;
  - typedef enum Boolean boolean;

35

C ANSI

# Typedef

- Exemplo com estrutura:
  - Definindo um "novo tipo de dado", para guardar as informações de um empregado:

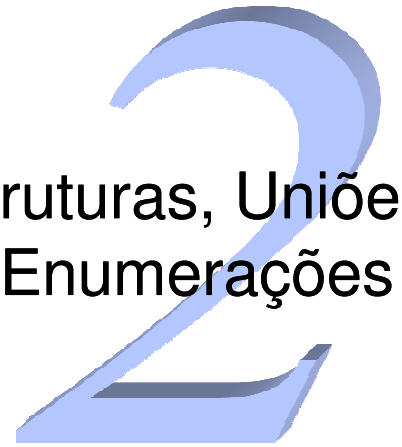
```
typedef struct {
        char nome[31];
        int matricula;
        float salario;
        char cargo[20];
} Tempregado; // o "T" é de Tipo
// Declaração (note que não tem mais o "struct")
TEmpregado empregado, empregados[10];
Estadocivil estcivil = SOLTEIRO;
```

```
typedef enum {SOLTEIRO,
CASADO, DIVORCIADO}
Estadocivil;
```

36

C ANSI

# *Laboratório de Programação*



## Estruturas, Uniões e Enumerações

37


19