Algoritmos e Programação B

Conjuntos Heterogêneos: estruturas
(continuação)

Conjuntos Heterogêneos

- Conjuntos Heterogêneos constituem tipos de dados definidos pelo usuário:
  - Estruturas (*struct*)
  - Uniões (*union*)
  - Enumeração (*enum*)

- e ainda o uso de
  - *typedef*

Matrizes e estruturas dentro de estruturas

- Um elemento (campo de estrutura) pode ser simples (char, int, float...) ou complexo:
  - vetor,
  - matriz ou
  - estrutura.

Exemplo

```c
struct endereco{
        char rua[40];
        int  nro;
        char complemento[20];
        char CEP[10];
        char cidade[30];
        char estado[3];
};
```

```c
struct pessoa{
        char nome[60];
        struct endereco e;

};
```

Exemplo

```c
int main(){
    struct pessoa a;

    printf("Informe o nome: ");
    fgets(a.nome, 60, stdin);
    printf("Rua: ");
    fgets(a.e.rua, 40, stdin);
    printf("Número: ");
    scanf("%d", &a.e.nro);
    printf("Complemento: ");
    fgets(a.e.complemento, 20, stdin);
    printf("CEP: ");
    fgets(a.e.CEP, 10, stdin);
    printf("Cidade: ");
    fgets(a.e.cidade, 30, stdin);
    printf("Estado: ");
    fgets(a.e.estado, 3, stdin);
```

Exemplo

```
    puts(a.nome);
    puts(a.e.rua);
    printf("numero %d\n", a.e.nro);
    puts(a.e.complemento);
    puts(a.e.CEP);
    puts(a.e.cidade);
    puts(a.e.estado);

    return 0;
}
```

Algoritmos e Programação B

Conjuntos Heterogêneos: Uniões

Uniões (*union*)

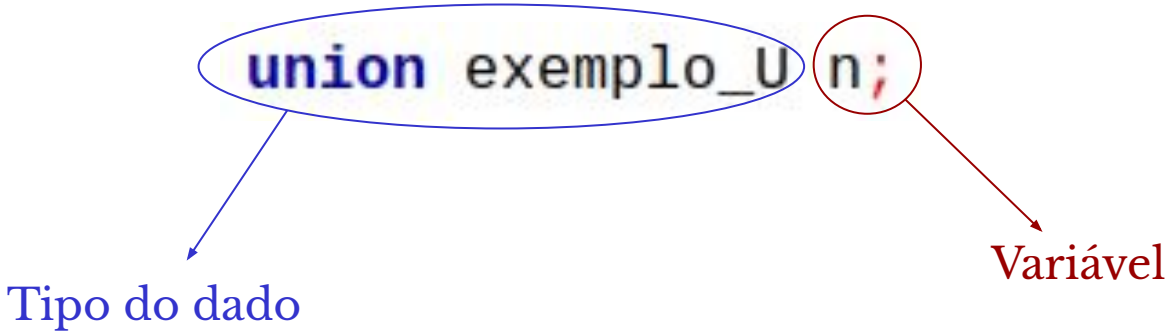
- Uniões definem uma posição de memória compartilhada onde podem ser armazenadas duas ou mais variáveis diferentes, geralmente de tipos diferentes.
- A declaração da União é semelhante a da estrutura
- **Declaração do tipo de dado:**

```
union identificador{
    Tipo1 nome_var1;
    Tipo2 nome_var2;
    Tipo3 nome_var3;
...
};
```

```
union exemplo_U{
    int i;
    char a;
};
```

Uniões (*union*)

- **Declaração de variável do tipo *union*:**



Uniões (*union*)

- Neste exemplo, o compilador cria automaticamente uma variável grande o suficiente para conter o `int` (o tipo de maior tamanho).
- Para acessar as variáveis, a sintaxe é idêntica à usada em estruturas:

```
n.i = 20;
```

- Uso de Union:
  - Para criação de código independente de arquitetura;
  - Para conversão de tipos de dados não padrão.

Uniões (*union*): exemplo

```c
#include <stdio.h>

#define Pi 3.14159

union angulo{
    float graus;
    float radianos;
};

int main() {
   union angulo ang;
   float auxGrau;
   char op;

   printf("\nNumeros em graus (G) ou radianos (R)? ");
   scanf("%c",&op);

   printf("Digite o angulo: ");
   scanf("%f",&auxGrau);
```

Uniões (*union*): exemplo

```
    if (op == 'G' || op == 'g') {
       printf("Angulo digitado em graus %f\n", auxGrau);
       ang.radianos = auxGrau/180*Pi;
       printf("\nAngulo em radianos: %f\n",ang.radianos);
    } else if (op == 'R' || op == 'r') {
              printf("Angulo digitado em radianos %f\n", auxGrau);
              ang.graus = auxGrau*180/Pi;
              printf("\nAngulo em graus: %f\n",ang.graus);
           } else printf("\nEntrada invalida!!\n");

     return 0;
}
```

Algoritmos e Programação B

Conjuntos Heterogêneos: Enumerações

Enumerações (*enum*)

- Enumerações definem um conjunto de constantes inteiras que especificam todos os valores legais que uma variável desse tipo pode ter.
- **Exemplo:**

```
enum Semana {seg, ter, qua, qui, sex, sab, dom};
enum Semana diaSemana;

diaSemana=seg;
if(diaSemana==dom) printf("O Dia é domingo\n");
```

Enumerações (*enum*)

```
enum Semana {seg, ter, qua, qui, sex, sab, dom};
enum Semana diaSemana;

diaSemana=seg;
if(diaSemana==dom) printf("O Dia é domingo\n");
```

- Cada símbolo representa um valor inteiro e, assim, pode ser usado onde um número inteiro poderia ser usado.
- No exemplo acima, `seg` é representada pelo valor `0 (zero)`.

Enumerações (*enum*)

- As enumerações são usadas para definir a tabela de símbolos de um compilador.
- Para mudar a sucessão de valores, basta igualar o primeiro elemento de `enum` a um valor.
- Isto pode ser usado no lugar de serem declaradas várias constantes, por exemplo:

```
enum months {jan=1, fev, mar, abr, mai, jun, jul, ago, set, out, nov, dez};
enum months mes;
…
if(mes==1) printf(“Janeiro”);
```

Enumerações (*enum*) - exemplo

```c
#include <stdio.h>

int main (void) {
      enum estacoes {primavera, verao, outono, inverno };
      enum estacoes est_ano;

      est_ano = inverno;
      printf ("%d ",est_ano);

      if (est_ano==2){
                        printf("outono");
                        }
      if (outono>verao){
         printf ("Frio");
      } else prinft("Quente");

      return 0;
}
```

Enumerações (*enum*)

- Lembre-se que `enum` trabalha com **constantes!!!**

Algoritmos e Programação B

Typedef

Typedef

- `Typedef` define um novo nome a um tipo de dado existente.
- É útil quando há grande quantidade de estruturas (`struct`).
- É aplicado também à uniões e enumerações.
- Torna o código mais fácil de ler.
- Com o uso da palavra typedef, na declaração de variáveis de conjuntos heterogêneos, é dispensado o uso das palavras reservadas: `struct`, `union` e `enum`.

Typedef - exemplo

```c
//Declaração do tipo de dado:
typedef struct {
    char matricula[11];
    char nome[30];
    int anoIngresso;
    float mediaVestibular;
} Taluno;


//Declaração da variável do tipo de dado criado:
Taluno aluno;
```

Algoritmos e Programação B

Outras funções da Linguagem C

Outras funções da Linguagem C

A Linguagem C oferece funções para realizar operações de conversão de tipos. Algumas funções são:

- **`atoi`** – converte string para inteiro. Recebe uma string como parâmetro e retorna o valor inteiro da representação.
- **`itoa`** - converte um inteiro para string. Recebe como parâmetro um valor inteiro, uma string (onde será armazenado o resultado) e um valor inteiro que indica a base.
- **`atof`** – converte string para um número em ponto flutuante. Recebe uma string como parâmetro e retorna o valor ponto flutuante da representação.