Tipos de Dados Definidos Pelo Usuário

- Estruturas
- Uniões
- Enumerações
- O Comando sizeof
- O Comando typedef

Estruturas

Uma estrutura agrupa várias variáveis numa só.

Funciona como uma ficha pessoal que tenha nome, telefone e endereço. A ficha seria uma estrutura.

Estruturas

Criando

Para se criar uma estrutura usa-se o comando **struct**.

Sua forma geral é:

```
struct nome_do_tipo_da_estrutura
{
    tipo_1 nome_1;
    tipo_2 nome_2;
    ...
    tipo_n nome_n;
} variáveis_estrutura;
```

Estruturas

O *nome_do_tipo_da_estrutura* é o nome para a estrutura.

As *variáveis_estrutura* são opcionais e seriam nomes de variáveis que o usuário já estaria declarando e que seriam do *tipo nome_do_tipo_da_estrutura*.

Estruturas

Exemplo

```
struct tipo_endereco
{
        char rua [50];
        int numero;
        char bairro [20];
        char cidade [30];
        char sigla_estado [3];
        long int CEP
};
```

Estruturas

Vamos agora criar uma estrutura chamada ficha com os dados pessoais de uma pessoa:

```
struct ficha_pessoal
{
        char nome [50];
        long int telefone;
        struct tipo_endereco endereco;
};
```

Estruturas

Usando

Vamos utilizar as estruturas declaradas na seção anterior para escrever um programa que preencha uma ficha.

```
#include <stdio.h>
#include <string.h>
struct tipo_endereco
{
        char rua [50];
        int numero;
        char bairro [20];
        char cidade [30];
        char sigla_estado [3];
        long int CEP;
};
```

```
struct ficha_pessoal
{
        char nome [50];
        long int telefone;
        struct tipo_endereco endereco;
};
```

Estruturas

```
main (void)
{
    struct ficha_pessoal ficha;
    strcpy (ficha.nome, "Luiz Osvaldo Silva");
    ficha.telefone = 4921234;
    strcpy (ficha.endereco.rua, "Rua das Flores");
    ficha.endereco.numero = 10;
    strcpy (ficha.endereco.bairro, "Cidade Velha");
    strcpy (ficha.endereco.cidade, "Belo Horizonte");
    strcpy (ficha.endereco.sigla_estado, "MG");
    ficha.endereco.CEP = 31340230;
}
```

O exemplo mostra como podemos acessar um elemento de uma estrutura: basta usar o ponto “.”.

Estruturas

Matrizes de estruturas

Um estrutura é como qualquer outro tipo de dado no C. Podemos, portanto, fazer matrizes de estruturas. Vamos ver como ficaria a declaração de uma matriz de 100 fichas pessoais:

struct ficha_pessoal fichas [100];

Poderíamos então acessar a segunda letra da sigla de estado da décima terceira ficha fazendo:

fichas[12].endereco.sigla_estado[1];

Estruturas

Atribuindo

Podemos atribuir duas estruturas que sejam do mesmo tipo. O C irá, neste caso, copiar uma estrutura na outra.

Exemplo,

```
void main()
{
    struct ficha_pessoal primeira, segunda;
    Le_dados(&primeira);
    segunda = primeira;
    Imprime_dados(segunda);
}
```

Todos os campos de primeira serão copiados na ficha chamada segunda.

Estruturas

Atribuindo

Devemos tomar cuidado com a seguinte declaração:

struct ficha_pessoal fichas [100];

pois neste caso fichas é um ponteiro para a primeira ficha.

Se quisermos a estrutura completa da *n*-ésima ficha devemos usar fichas[n-1].

Estruturas

Passando para funções

No exemplo apresentado no ítem usando, vimos o seguinte comando:

strcpy (ficha.nome,"Luiz Osvaldo Silva");

Neste comando um elemento de uma estrutura é passado para uma função. Este tipo de operação pode ser feita sem maiores considerações.

Estruturas

Passando para funções

Podemos também passar para uma função uma estrutura inteira. Veja a seguinte função:

```
void PreencheFicha (struct ficha_pessoal ficha)
{
   ...
}
```

Estruturas

Passando para funções

Devemos observar que, como em qualquer outra função no C, a passagem da estrutura é feita por valor. Isto significa que alterações na estrutura dentro da função não terão efeito na variável fora da função. Mais uma vez podemos contornar este pormenor usando ponteiros e passando para a função um ponteiro para a estrutura.

Estruturas

Ponteiros

Podemos ter um ponteiro para uma estrutura. Vamos ver como poderia ser declarado um ponteiro para as estruturas de ficha que estamos usando nestas seções:

*struct ficha_pessoal *p;*

Estruturas

Ponteiros

Os ponteiros para uma estrutura funcionam como os ponteiros para qualquer outro tipo de dados no C. Há, entretanto, um detalhe a ser considerado. Se apontarmos o ponteiro *p* declarado acima para uma estrutura qualquer e quisermos acessar um elemento da estrutura poderíamos fazer:

*(*p).nome*

Este formato raramente é usado.

Estruturas

Ponteiros

O que é comum de se fazer é acessar o elemento nome através do operador seta (->). Assim faremos:

p->nome

A declaração acima é muito mais fácil e concisa. Para acessarmos o elemento *CEP* dentro de *endereco* faríamos:

p->endereco.CEP

Uniões

Em C, uma união é uma posição de memória que é compartilhada por duas ou mais variáveis diferentes, geralmente de tipos diferentes, em momentos diferentes.

Uniões

Declaração Union

A declaração de uma união é semelhante à declaração de uma estrutura:

```
union nome_do_tipo_da_union
{
    tipo_1 nome_1;
    tipo_2 nome_2;
    ...
    tipo_n nome_n;
} variáveis_union;
```

Uniões

Como exemplo, vamos considerar a seguinte união:

```
union angulo
{
    float graus;
    float radianos;
};
```

Nela, temos duas variáveis (graus e radianos) que, apesar de terem nomes diferentes, ocupam o mesmo local da memória. Isto quer dizer que só gastamos o espaço equivalente a um único **float**.

Uniões

Uniões podem ser feitas também com variáveis de diferentes tipos. Neste caso, a memória alocada corresponde ao tamanho da maior variável no union.

Uniões

```
#include <stdio.h>
#define GRAUS  'G'
#define RAD  'R'
union angulo
{
      int graus;
      float radianos;
};
```

```
void main() {
    union angulo ang;
    char op;
    printf("\n Numeros em graus
                          ou radianos? ");
    scanf("%c",&op);
    if (op == GRAUS) {
         ang.graus = 180;
         printf("\nAngulo: %d\n",ang.graus);
    }
   else if (op == RAD) {
        ang.radianos = 3.1415;
        printf("\n Angulo: %f\n",ang.radianos);
   }
   else printf("\n Entrada invalida!!\n");
}
```

Enumerações

Uma enumeração é um conjunto de constantes inteiras que especifica todos os valores legais que uma variavel desse tipo pode ter. Ou seja, numa enumeração podemos dizer ao compilador quais os valores que uma determinada variável pode assumir. Sua forma geral é:

enum nome_do_tipo_da_enumeração
{lista_de_valores} lista_de_variáveis;

Enumerações

Vamos considerar o seguinte exemplo:

enum dias_da_semana {segunda, terca,
quarta, quinta, sexta, sabado, domingo};

O programador diz ao compilador que qualquer variável do tipo dias_da_semana só pode ter os valores enumerados.

Enumerações

```
#include <stdio.h>
enum dias_da_semana {segunda, terca, quarta,
                                    quinta, sexta, sabado, domingo};
main (void)
{
    enum dias_da_semana d1, d2;
    d1 = segunda;
    d2 = sexta;
    if (d1 == d2){
        printf ("O dia e o mesmo.");
    }
    else{
        printf ("São dias diferentes.");
    }
}
```

Enumerações

O compilador pega a lista que você fez de valores e associa, a cada um, um número inteiro. Então, ao primeiro da lista, é associado o número zero, o segundo ao número 1 e assim por diante.

Enumerações

Você pode especificar o valor de um ou mais dos símbolos usando um inicializador. Isso é feito colocando-se um sinal de igual e um valor inteiro após o símbolo.

```
enum coin {penny, nickel, dime, quarter = 100,
                                    half_dollar, dollar};
```

Agora, os valores destes símbolos são:

penny	0	nickel	1	dime	2
quarter	100	half_dollar	101	dollar	102

Enumerações

Os símbolos de enumerações são constantes de números inteiros. Portanto,

```
/* isso não funcionará */
enum coin money;
money = dollar;
printf("%s", money);
```

```
/* esse código está errado */
enum coin money;
strcpy(money, "dime");
```

Enumerações

```
#include <stdio.h>
enum coin {penny, nickel, dime, quarter,
                                        half_dollar, dollar};
main (void)
{
    enum coin money;
    switch(money) {
       case penny : printf("penny"); break;
       case nickel : printf("nickel"); break;
       case dime: printf("dime"); break;
       case quarter: printf("quarter"); break;
       case half_dollar: printf("half_dollar"); break;
       case dollar: printf("dollar"); break;
}
```

O Comando sizeof

O operador **sizeof** é usado para se saber o tamanho de variáveis ou de tipos. Ele retorna o tamanho do tipo ou variável em bytes. Mas porque usá-lo se sabemos, por exemplo, que um inteiro ocupa 2 bytes? Devemos usá-lo para garantir portabilidade. O tamanho de um inteiro pode depender do sistema para o qual se está compilando. O **sizeof** é chamado um operador porque ele é substituído pelo tamanho do tipo ou variável no momento da compilação. Ele não é uma função.

O Comando sizeof

O **sizeof** admite duas formas:

sizeof *(nome_da_variável);*

sizeof *(nome_do_tipo);*

O Comando sizeof

Se quisermos então saber o tamanho de um **float** fazemos **sizeof**(**float**). Se declararmos a variável *f* como **float** e quisermos saber o seu tamanho faremos **sizeof** (*f*).

O operador **sizeof** também funciona com estruturas, campos bit, uniões e enumerações para saber o tamanho de tipos definidos pelo usuário. Seria, por exemplo, uma tarefa um tanto complicada a de alocar a memória para um ponteiro para a estrutura ficha_pessoal, se não fosse o uso de **sizeof**.

```
#include <stdio.h>
struct tipo_endereco
{
        char rua [50];
        int numero;
        char bairro [20];
        char cidade [30];
        char sigla_estado [3];
        long int CEP;
};
```

```
struct ficha_pessoal
{
   char nome [50];
   long int telefone;
   struct tipo_endereco endereco;
};
```

```
void main(void) {
    struct ficha_pessoal *ex;
    ex = (struct ficha_pessoal *) malloc(sizeof(struct ficha_pessoal));
    ...
    free(ex);
}
```

O Comando typedef

O comando **typedef** permite ao programador definir um novo nome para um determinado tipo. Sua forma geral é:

typedef antigo_nome novo_nome;

Como exemplo vamos dar o nome de inteiro para o tipo **int**:

typedef int inteiro;

Agora podemos declarar o tipo inteiro.

O Comando typedef

O comando **typedef** também pode ser utilizado para dar nome a tipos complexos, como as estruturas.

```
#include <stdio.h>
typedef struct tipo_endereco
{
        char rua [50];
        int numero;
        char bairro [20];
        char cidade [30];
        char sigla_estado [3];
        long int CEP;
} TEndereco;
```

```
typedef struct ficha_pessoal
{
        char nome [50];
        long int telefone;
        TEndereco endereco;
}TFicha;
void main(void)
{
    TFicha *ex;
...
}
```