

Linguagem de Programação I

Struct

Struct

Uma estrutura é uma coleção de variáveis referenciadas por um nome, fornecendo uma maneira conveniente de se ter informações relacionadas agrupadas.

A palavra-chave `struct` informa ao compilador que um modelo de estrutura está sendo definido.

```
struct cadastro
{
    char nome[50];
    char endereco[100];
    int numero;
    char complemento[30];
    char cidade[30];
    char estado[3];
};
```

→ Nenhuma variável de fato foi declarada! Apenas a forma dos dados foi definida.

Struct

Para declarar uma variável do tipo cadastro escreva:

```
struct cadastro infoCadastro;
```

Isto declara uma variável do tipo struct cadastro chamada infoCadastro.

Quando uma variável de estrutura (como infoCadastro) é declarada, o compilador C aloca automaticamente memória suficiente para acomodar todos os seus membros.

Nome	50 bytes
Endereco	100 bytes
Numero	4 bytes
Complemento	30 bytes
Cidade	30 bytes
estado	3 bytes

Struct

Mais exemplos:

```
struct cadastro
{
    char nome[50];
    char endereco[100];
    int numero;
    char complemento[30];
    char cidade[30];
    char estado[3];
} cadastroInfo1, cadastroInfo2, Cadastroinfo3;
```

Define uma estrutura chamada cadastro e declara as variáveis cadastroInfo1, cadastroInfo2, Cadastroinfo3 desse tipo.

Struct

→ Importante:

- Ao definir uma estrutura, basicamente define-se um tipo complexo de dado, e não uma variável;
- Um tipo estrutura pode ser global ou local, ou seja, o tipo definido pode existir apenas dentro de uma função (struct local) ou pode existir em todo o programa (global). Se a estrutura existir apenas localmente a uma função, apenas esta função poderá ter variáveis do tipo da estrutura declarada.

Struct

Acessando os membros da estrutura

O operador ponto (.) conecta o nome de uma variável estrutura a um membro dela.

Exemplo: cadastroInfo1.numero=198;

A linguagem C trata os membros de uma estrutura como quaisquer outras variáveis simples.

Struct

Acessando os membros da estrutura

Para imprimir o numero na tela:

```
printf("%i",cadastroInfo1.numero);
```

Da mesma forma, podemos usar gets para receber valor de uma string.

```
gets(cadastroInfo.nome)
```

Struct

Atribuição de estruturas

```
int main()
{
    struct {
        int a;
        int b;
    } x, y;
    x.a=10;

    y=x; //atribui uma estrutura a outra
    printf("%d",y.a);
}
```

→ Após a atribuição, y.a conterà o valor 10.

Struct

Passando estruturas para funções:

As estruturas podem ser passadas como argumentos de funções da mesma forma que variáveis simples.

O nome de uma estrutura em C não é um endereço, portanto, pode ser passada por valor.

Struct

```
#include <stdio.h>

typedef struct Venda
{
    int pecas;
    float preco;
} Venda;

// Protótipo da função
Venda TotalVendas(Venda c, Venda d);
```

Struct

```
int main()
{
    Venda a, b, total;

    printf("Venda A\n=====\\n");
    printf("Insira o numero de pecas: ");
    scanf("%i", &a.pecas);
    printf("Insira o preço          :");
    scanf("%f", &a.preco);

    printf("Venda B\n=====\\n");
    printf("Insira o numero de pecas: ");
    scanf("%i", &b.pecas);
    printf("Insira o preço          :");
    scanf("%f", &b.preco);

    total=TotalVendas(a,b); // estruturas como argumento e retorno

    printf("\\n\\n Venda Total\\n=====\\n");
    printf("\\nTotal de Pecas: %i",total.pecas);
    printf("\\nPreco total   : %.2f\\n",total.preco);

    return 0;
}
```

Struct

```
Venda TotalVendas(Venda c, Venda d)
{
    Venda t;
    t.pecas=c.pecas+d.pecas;
    t.preco=c.preco+d.preco;
    return t;
}
```

Matriz de Estruturas

Venda vendas[50]

Declara vendas como sendo uma matriz de 50 elementos. Cada elemento da matriz é uma estrutura do tipo Venda.

O nome vendas é o nome da matriz (portanto um endereço) em que os elementos são estruturas.

O compilador, então, providencia espaço contínuo de memória para armazenar 50 estruturas do tipo Venda.

Uso:

Vendas[0].preco;

Vendas[1].pecas;

Ponteiros para Estruturas

```
struct cadastro
{
    char nome[50];
    char endereco[100];
    int numero;
    char complemento[30];
    char cidade[30];
    char estado[3];
};
```

Declaração:

struct cadastro *pcadastro;

Declara pcadastro como um ponteiro para dados daquele tipo.

Ponteiros para Estruturas

Há dois usos primários para ponteiros de estruturas:

- Gerar uma chamada por referência para uma função.
- Criar listas encadeadas e outras estruturas de dados dinâmicas

Ponteiros para Estruturas

Para encontrar o endereço de uma variável estrutura, deve-se colocar o operador & antes do nome da estrutura.

Ex.:

```
struct bal
{
    float balance;
    char name[80];
} person;
```

```
struct bal *p; // declara um ponteiro para a estrutura
```


Ponteiros para Estruturas

// coloca o endereço da estrutura person no ponteiro p.

```
p=&person;
```

Para acessar os elementos de uma estrutura usando um ponteiro para a estrutura, você deve usar o operador ->.

Operador seta

```
p->balance
```

A seta é usado no lugar do ponto(.) quando se está acessando um elemento de estrutura por meio de um ponteiro para a estrutura.

Ponteiros para Estruturas

```
struct meuTempo
{
    int horas;
    int minutos;
    int segundos;
};
void mostra(struct meuTempo *t)
{
    printf("\n%dh ", t->horas);
    printf("%dmin ", t->minutos);
    printf("%d s \n", t->segundos);
    //printf("\n%p", t);
}
void espera()
{
    long int t;
    for (t=0; t<100000; t++) { }
}
```

Ponteiros para Estruturas

```
void atualiza(struct meuTempo *t)
{
    t->segundos++;
    if (t->segundos==60)
    {
        t->segundos=0;
        t->minutos++;
    }
    if (t->minutos==60)
    {
        t->minutos=0;
        t->horas++;
    }
    if (t->horas==24)
        t->horas=0;
    espera();
}
```

Ponteiros para Estruturas

```
int main()
{
    struct meuTempo tempo;
    int i;
    tempo.horas=0;
    tempo.minutos=0;
    tempo.segundos=0;

    for (i=0;i<=100;i++)
    {
        atualiza(&tempo);
        mostra(&tempo);
    }
    return 0;
}
```



Referência Bibliográfica

Mizrahi, Victorine Viviane. Treinamento em Linguagem C. São Paulo: Editora Pearson, 2008, 2ª edição.

Schildt, Hebert. C completo e Total. São Paulo: Makron Books, 1996.