Técnicas de Programação II

Estruturas

Estruturas

- As estruturas em C (que correspondem aos registros em outras linguagens) permitem colocar, em uma única entidade, elementos de tipos diferentes.

- Uma estrutura é um conjunto de uma ou mais variáveis (campos) agrupadas sob um único nome, de forma a facilitar a sua referência.

- As estruturas podem conter elementos com qualquer tipo de dados válidos em C.

Estruturas

- **Suponhamos que pretendíamos representar em uma aplicação os principais dados que constituem um indivíduo.**
- **Uma possível solução seria a declaração de variáveis individuais para cada uma das características a representar.**
- **Exemplo:**

```
int idade;
char sexo, est_civil;
char nome[60];
float salario;
int dia;
char mes[12];
int ano;
```

- **No entanto, essa solução não é mais aconselhável, pois nada nos diz que as variáveis estão de algum modo relacionadas entre si.**

Declaração de Estruturas

- Sintaxe:

```
struct  nome_da_estrutura
{
        tipo campo, campo, ... ;
        ...
        tipo campo, campo, ..., ;
} ;
```

- Como o seu nome indica, a declaração de uma estrutura corresponde unicamente à definição de um novo tipo (isto é, da sua estrutura), e não à declaração de variáveis do tipo estrutura.

Declaração de Estruturas

- Exemplo: definir uma estrutura capaz de suportar datas.

```
struct Data
{
        int Dia, Ano;
        char Mes[12];
};
```

- A definição da estrutura **Data** (struct Data) indica simplesmente que, a partir daquele momento o compilador passa a conhecer um outro tipo, chamado **struct Data**, que é composto por dois inteiros e um vetor com 12 caracteres.

- Neste exemplo, **Data** não é uma variável, mas sim o nome pelo qual é conhecida essa nova definição de tipo de dados.

Declaração de Variáveis do Tipo Estrutura

- Para declarar uma variável do tipo **struct Data**, basta indicar qual o tipo (**struct Data**) seguido do nome das variáveis.

```
struct Data d, datas[100], *ptr_data;
```

Em que:

- **d** é uma variável do tipo *struct Data*
- **datas** é um vetor de 100 elementos (cada um do tipo *struct Data)*
- **ptr_data** é um ponteiro para o tipo *struct Data.*

Declaração de Variáveis do Tipo Estrutura

- A declaração de variáveis pode também ser realizada quando se faz a definição da própria estrutura através da sintaxe:

```
struct  nome_da_estrutura
{
        tipo campo, campo, ... ;
        ...
        tipo campo, campo, ..., ;
} var_1, var_2, ... Var_n;
```

Declaração de Variáveis do Tipo Estrutura

- Exemplo de definição de uma estrutura e declaração de variáveis.

```
struct  Data
{
        int Dia;
        int Ano;
        char Mes[12];
} d, datas[100], *ptr_data;
```

Declaração de Variáveis do Tipo Estrutura

- Defina uma nova estrutura capaz de suportar a definição de um indivíduo chamada **Pessoa**. Declare na definição as variáveis **Paulo e Teresa** desse mesmo tipo.

```
struct Pessoa
{
        int idade;
        char sexo;
        char est_civil;
        char nome[60];
        float salario;
} Paulo, Tereza;
```

Acesso aos membros da estrutura

- Para acessar o membro (campo) **mmm** de uma estrutura **eee** usa-se o operador ponto (**.**) fazendo: **eee.mmm**
- Exemplo:

```
struct Data{
        int dia, ano;
        char mes[12];
} dt_nasc;

dt_nasc.dia = 23;
strcpy(dt_nasc.mes[0], “janeiro”);
dt_nasc.ano = 1966;
printf(“Data: %d/%s/%d\n”, dt_nasc.dia, dt_nasc.mes, dt_nasc.ano);
```

Carga inicial automática de estruturas

- Uma estrutura pode ser iniciada quando é declarada usando-se a sintaxe:

  struct nome_estrutura var = {valor_1, valor_2, ..., valor_n}

- Colocando entre chaves os valores dos membros da estrutura, pela ordem em que esses foram escritos na sua definição.

  struct Data {int dia, ano; char mês[12];} **dt_nasc = { 23, 1966, "janeiro"};**

  ou

```
struct Data {int dia, ano; char mês[12];}          // declaração da estrutura
struct Data dt_nasc = { 23, 1966, "janeiro" };     //decl. da variável
```

- Se a variável a ser iniciada for um vetor, a carga inicial é feita do mesmo modo, colocando cada um dos elementos dentro de chaves.

```
struct Data v[3] = {{ 23, 1966, "janeiro" }, {2,"marco",1920}, {31, "maio",2000}};
```

Definição de Tipos - typedef

- Uma das desvantagens existentes na utilização de estruturas está na declaração das variáveis, que têm sempre que ser precedidas da palavra reservada **struct**, seguida do nome da estrutura.

- O ideal seria podermos representar uma estrutura unicamente através de uma palavra (um sinônimo), tal como fazemos com os tipos-base da linguagem (int, float,, etc).

- Isso é possível através da palavra reservada **typedef**, cuja sintaxe é:

  typedef tipo_existente sinônimo

- A palavra **typedef** não cria um novo tipo. Permite apenas que um determinado tipo possa ser denominado de forma diferente, de acordo com a necessidade ou vontade do programador.

Definição de Tipos – typedef - exemplos

typedef float Real;

- O tipo **float** passa a ser representado e tratado pela palavra **Real**. Podemos, por isso declarar variáveis usando ambas as palavras:

```
float a, b[10], *ptr;
```

ou

```
Real a, b[10], *ptr;
```

```
typedef char* POINTER;
typedef unsigned long int ULINT;
POINTER string = “Ola”;     // string é do tipo (char*)
ULINT n_total;              // n_total é um inteiro longo sem sinal
```

Definição de Tipos – typedef - Exemplos

```
typedef struct Pessoa
        {
                int idade;
                char sexo;
                char est_civil;
                char nome[60];
                float salario;
        } PESSOA;
```

- Podemos declarar variáveis de duas formas diferentes:

        struct Pessoa Carlos, Serafim;

ou

        PESSOA Carlos, Serafim;

Definição de Tipos – typedef - Exemplos

```
typedef struct                  // a estrutura não tem nome
        {
                int idade;
                char sexo;
                char est_civil;
                char nome[60];
                float salario;
        } PESSOA;
```

- Na definição de um typedef não podem ser declaradas variáveis

```
} PESSOA Paulo, Mario;
```

Definição de Tipos – typedef - Exemplos

```
struct  pessoa
{
        int idade;
        char sexo;
        char est_civil;
        char nome[60];
        float salario;
};
typedef struct pessoa Pessoa;
```

- O tipo que este professor adota e prefere.

Onde definir estruturas e typedef

- Se a definição de uma estrutura for realizada dentro de uma função, apenas essa função conhece essa definição.
- Por isso, as estruturas devem ser definidas de forma a serem visíveis por todo o programa, fazendo a sua definição fora de qualquer função – em geral no início do programa ou num *header file*.

```
#include <.....>
#define ....
struct estrutura {.............};
typedef ...................;
/* protótipos das funções */
int main() {.............}
/* funções */
```

Estruturas dentro de Estruturas

- Uma estrutura pode conter, na sua definição, variáveis simples, vetores, ponteiros ou mesmo outras estruturas.

- A única restrição a que se tem que obedecer é que todas as estruturas ou tipos utilizados na definição de uma nova estrutura tem que estar previamente definidos.

Estruturas dentro de Estruturas

- Declare um novo tipo denominado pessoa que contenha nome, idade, salário e data de nascimento.

```
        typedef struct
        {
                int Dia;
                char Mes[3+1];
                int Ano;
        } DATA;
typedef struct s_pessoa {
                char Nome[100];
                int Idade;
                float Salario;
                DATA dt_Nasc;
        } PESSOA;
```

Estruturas dentro de Estruturas

- Para declarar uma variável simples desse tipo e um vetor com três pessoas:

```
struct s_pessoa homem, mulher[3];
```

ou

```
PESSOA homem, mulher[3];
```

- Qual o código que teria de ser alterado, caso se pretendesse iniciar as variáveis no momento da declaração?

```
PESSOA homem     = {“Ze”, 23, 123.45, (10,”MAI”,1989)},
        mulher[3]  = { {“To”,51,0.0,(15,”JUN”,1975)},
                       {“Alf,17,52.0,(22,”JUL”,1965)},
                       {“NY”,22,25.93,(14,”DEZ”,1956)}  };
```

- Para colocar o ano de 1999 na segunda mulher do vetor:

```
mulhr[1].dt_nasc.Ano = 1999;
```

Estruturas dentro de Estruturas

- Para alterar o mês da última mulher para “NOV:

  strcpy(mulher[2].dt_nasc.Mes, “NOV”);

- Para mostra na tela a primeira letra de cada um dos meses das mulheres:

  for(i=0;i<3;i++) putchar(mulher[i].dt_Nasc.Mes[0]);

- Então:
  - mulher - é um vetor com três estruturas do tipo PESSOA..
  - mulher[i] - representa a i-ésima estrutura do tipo PESSOA.
  - mulher[i].dt_Nasc - o mês que queremos acessar faz parte da estrutura dt_Nasc.
  - mulher[i].dt_Nasc.Mes - nesta apenas queremos utilizar o campo Mês.
  - mulher[i].dt_Nasc.Mes[0] – do qual queremos obter apenas o primeiro caractere.

Passagem de Estruturas para Funções

- A passagem se faz indicando no parâmetro o tipo associado à estrutura (ou o typedef).
- Exemplo: escrever uma função que permita escrever na tela os valores existentes em um estrutura recebida como argumento. Sendo as estruturas:

```
struct data {
        int Dia;
        int Mes;
        int Ano;
};
typedef struct data Data;
```

```
struct pessoa {
        char Nome[100];
        int Idade;
        float salario;
        Data Nasc;
}
typedef struct pessoa Pessoa;
```

Passagem de Estruturas para Funções

```
int main()
{
   Pessoa p;   /* cria uma estrutura com o nome p */
/* entrada de dados */
   printf("Nome: ");
   gets(p.Nome);
   printf("Idade: ");
   scanf("%d",&p.Idade);
   printf("Salario: ");
   scanf("%f",&p.salario);
   printf("Dia Nascimento: ");
   scanf("%d",&p.Nasc.Dia);
   printf("Mes Nascimento: ");
   scanf("%d",&p.Nasc.Mes);
   printf("Ano Nascimento: ");
   scanf("%d",&p.Nasc.Ano);

/* apresentacao dos dados */
   printf("\nNome: %s\n", p.Nome);
   printf("Idade: %d anos\n", p.Idade);
   printf("Salario: %.2f\n", p.salario);
   printf("Nascimento: %02d/%02d/%04d\n", p.Nasc.Dia, p.Nasc.Mes, p.Nasc.Ano);
   system("pause>nul");
}
```

Passagem de Estruturas para Funções V1

```
void mostra(Pessoa); /* ou struct pessoa */
int main(){
   Pessoa p;   /* cria uma estrutura com o nome p */
   printf("Nome: ");
   gets(p.Nome);
   printf("Idade: ");
   scanf("%d",&p.Idade);
   printf("Salario: ");
   scanf("%f",&p.salario);
   printf("Dia Nascimento: ");
   scanf("%d",&p.Nasc.Dia);
   printf("Mes Nascimento: ");
   scanf("%d",&p.Nasc.Mes);
   printf("Ano Nascimento: ");
   scanf("%d",&p.Nasc.Ano);
   mostra(p);
   system("pause>nul");
}
void mostra(Pessoa p)
{
   printf("\nNome: %s\n", p.Nome);
   printf("Idade: %d anos\n", p.Idade);
   printf("Salario: %.2f\n", p.salario);
   printf("Nascimento: %02d/%02d/%04d\n", p.Nasc.Dia, p.Nasc.Mes, p.Nasc.Ano);
}
```

Passagem de Estruturas para Funções por valor

```
struct pessoa ler(struct pessoa p);
void mostra(struct pessoa p);
int main(){
   Pessoa p;
   mostra(ler(p));
   system("pause>nul");
}
struct pessoa ler(struct pessoa p){
   printf("Nome: ");
   gets(p.Nome);
   printf("Idade: ");
   scanf("%d",&p.Idade);
   printf("Salario: ");
   scanf("%f",&p.salario);
   printf("Dia Nascimento: ");
   scanf("%d",&p.Nasc.Dia);
   printf("Mes Nascimento: ");
   scanf("%d",&p.Nasc.Mes);
   printf("Ano Nascimento: ");
   scanf("%d",&p.Nasc.Ano);
   return p;
}
```

```
void mostra(struct pessoa p)
{
   printf("\nNome: %s\n", p.Nome);
   printf("Idade: %d anos\n", p.Idade);
   printf("Salario: %.2f\n", p.salario);
   printf("Nascimento:%02d/%02d/%04d\n",
p.Nasc.Dia, p.Nasc.Mes, p.Nasc.Ano);
}
```

Passagem de Estruturas para Funções por referência

```
void ler(Pessoa *p);
void mostra(struct pessoa p);
int main()
{
   Pessoa p;
   ler(&p);
   mostra(p);
   system("pause>nul");
}
void ler(Pessoa *p)
{
   printf("Nome                : ");   gets(p->Nome);
   printf("Idade               : ");   scanf("%d",&p->Idade);
   printf("Salario             : ");   scanf("%f",&p->salario);
   printf("Dia Nascimento      : ");   scanf("%d",&p->Nasc.Dia);
   printf("Mes Nascimento      : ");   scanf("%d",&p->Nasc.Mes);
   printf("Ano Nascimento      : ");   scanf("%d",&p->Nasc.Ano);
}

void mostra(struct pessoa p)
{
   printf("\nNome: %s\n", p.Nome);
   printf("Idade: %d anos\n", p.Idade);
   printf("Salario: %.2f\n", p.salario);
   printf("Nascimento: %02d/%02d/%04d\n", p.Nasc.Dia, p.Nasc.Mes, p.Nasc.Ano);
}
```

Operações sobre Estruturas

- Se **x** é uma estrutura e **m** é um membro dessa estrutura, então o operador ponto (.) permite obter o valor do membro de **m** de **x** através de **x.m**

- Se **p** é um ponteiro para uma estrutura e **m** é um membro dessa estrutura, então o operador **(*p)** permite obter o valor do membro **m** de **x** por meio de **(*p).m**

- Se **p** é um ponteiro para uma estrutura e **m** é um membro dessa estrutura, então o operador **->** permite obter o valor do membro m de **x** por meio de **p->m**

- Se **x** e **y** forem duas variáveis com a mesma estrutura, então, para copiar todos os membros de **x** para **y** basta fazer **y = x**, isto é, pode-se fazer a atribuição de estruturas

Operações sobre Estruturas

- Se **x** é uma estrutura, então **&x** devolve o endereço da estrutura em memória, isto é, o menor dos endereços ocupados pela estrutura em memória

- Se **x** é uma estrutura e **m** um campo dessa estrutura, então **&x.m** devolve o endereço de memória do membro **m** de **x**

- Não se pode fazer comparações diretas entre estruturas através dos operadores **<, <=, >, >=, == ou !=**. O programador deverá estabelecer qual a relação entre duas variáveis do tipo estrutura a partir de comparações entre seus campos.

MAIS DETALHES EM MEMÓRIA DINÂMICA

Exercícios

- Defina em C um novo tipo denominado Pessoa que contenha as seguintes características: Nome, Idade, Salário e um Indicador que mostre se o registro está apagado ou não.

- Implemente as funções Ler_Pessoa e Mostrar_Pessoa, que permitem ler e mostrar todos os dados relativos a uma determinada pessoa.

Exercício Proposto

- Implemente uma aplicação completa baseada no exercício anterior que contenha um Menu com as seguintes opções:

  1. Inserir Registro
  2. Alterar Registro
  3. Apagar Registro
  4. Lista Registros
  5. Pesquisas

  0 Sair

  A opção 5 deverá apresentar um novo SubMenu com as seguintes opções:

  1. Pesquisar por Intervalo de Idades
  2. Pesquisar por Nome

  0. Voltar

  Implemente todas as funcionalidades presentes nas opções de cada um dos Menus. A questão não poderá ter mais que **MAX** registros (valor definido pelo programador). Portanto, é um vetor de estruturas.