CENTRO PAULA SOUZA

Linguagem de Programação

Estrutura de um Programa:
Estruturas (Structs) ou Registros
Material: LP_Aula09

Estruturas ou Registros

Diz Damas(2007)

- Estruturas em C, permitem colocar, em uma única **entidade**, elementos de **tipos diferentes**.
- Uma estrutura é um conjunto de uma ou mais variáveis agrupadas sob um único nome, de forma a facilitar sua referência.
- As estruturas podem conter elementos com qualquer tipo válido em C(tipos básicos, vetores, ponteiros, strings, ou mesmo outras estruturas).

Exemplo

PESSOA

idade

nome

sexo

salario

Estado civil

dia

ano

mês

Pessoa

Para representarmos uma pessoa, podemos criar uma estrutura e agrupar as variáveis (relacioná-las).

Sintaxe

```
struct [nome_da_estrutura]
 {
     tipo1      campo1, campo2;
          ...
     tipon      campo;
} ;
```

Declarar uma estrutura corresponde unicamente a definição de um novo tipo, e não à declaração de variáveis do tipo estrutura.

Estrutura para suportar datas

Podemos definir como exemplo uma estrutura para suportar datas.

Ex:

```
struct Data
{
   int dia, ano;
   char mes[15];
};
```

Compilador

A definição de uma estrutura indica que, a partir daquele momento o compilador passa a **conhecer um novo tipo**, chamado **struct Data**, que é composto por dois inteiros e um vetor com 15 caracteres.

Ou seja, **Data** não é uma variável, e sim o nome pelo qual é conhecida essa nova denominação de tipo.

Declaração de Variáveis do Tipo Data

Para declarar uma variável do tipo struct Data, basta indicar o tipo seguido do nome da variável.

```
Exs:
struct Data d;
struct Data datas[100];
struct Data *ptr_Data;
```

Em que:

- **d** é uma variável do tipo struct Data,
- **datas** é um vetor de 100 elementos sendo cada um deles do tipo struct Data,
- **ptr_Data** é um ponteiro para o tipo struct Data.

Outra forma de declaração

Pode-se declarar no momento da definição de uma estrutura. Veja:

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

Acesso aos campos membros

Para acessar um campo membro de uma estrutura usa-se o operador ponto [.] desta forma:

– var.membro

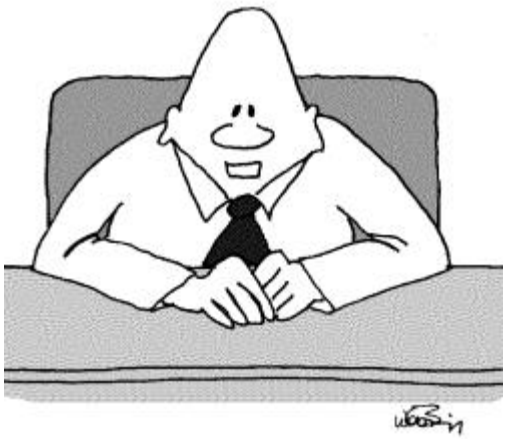
Carga inicial automática de estruturas

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

struct nome_struct var = {v1, v2, .., vn}

Deve colocar na chave os valores dos membros, pela ordem em que foram definidos na estrutura.

Representação Gráfica [Estrutura Empregado]



Empregado

```
struct Empregado
{
   int matr;
   float salario;
   char nome[20];
};
```

Representa Gráfica da Estrutura Empregado

matr	salario	1	2	3….. nome…. 19	20

Observe o Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
struct empregado
{
 int matr;
 float salario;
 char nome[20];
};
void imprimir(struct empregado e)
{
  printf("\n\nDADOS CADASTRADOS\n");
  printf("Matricula.....: %d\n", e.matr);
  printf("Salario.......: %5.2f\n", e.salario);
  printf("Nome..........: %s\n", e.nome);
}
int main(int argc, char *argv[])
{
  struct empregado emp[5];
  printf("\nAlimentando com dados do empregado\n");
  printf("==================================\n");
  printf("Digite a Matricula do Empregado: ");
  scanf("%d", &emp.matr);
  printf("Digite o Salario do Empregado: ");
  scanf("%f", &emp.salario);
  fflush(stdin); // Limpar buffer do teclado.
  printf("Digite o Nome do Empregado: ");
  gets(emp.nome);
  imprimir(emp);
  system("PAUSE");
  return 0;
}
```

Exemplificando o uso de estruturas complexas

Estruturas Heterogêneas com uso de Vetores

Projeto: prjStructHet

Inicie um novo projeto com o nome acima para ilustrarmos o uso de estruturas complexas usando vetor.

Consideremos um exemplo em que desejamos armazenar uma tabela com dados de alunos.

Podemos organizar os dados em um vetor.

Para cada aluno, serão necessárias as seguintes informações...

Informações Necessárias.

Nome: cadeia de 80 caracteres.

Matrícula: número inteiro.

Endereço: cadeia com até 120 caracteres.

Telefone: cadeia com até 20 caracteres.

Nota: número de ponto flutuante.

Estruturar os dados

Vamos definir um tipo que representa dos dados de um aluno.

```
struct aluno{
        char nome[80];
        int mat;
        char endereco[120];
        char fone[20];
        float nota;
};
```

Vetor global

Vamos montar a tabela de alunos usando um vetor global com um número máximo de alunos. Uma forma é declarar um vetor de estruturas. Ex:

```
#define MAX 100
struct aluno tabAlu[MAX];
```

Nota: Exemplo sem uso de ponteiros = mais consumo de memória

Evitando consumo desnecessário de memória

Para contornar esse problema, podemos trabalhar com um vetor de ponteiros.

```
typedef struct aluno *ptr_aluno;
#define MAX 100
ptr_aluno tabAlu[MAX];
```

Notas:

a)Exemplo com uso de ponteiros = menor consumo de memória

b) o typedef é uma palavra reservada da linguagem C, utilizada para definir novos tipos de dados.

Atenção

Uma posição do vetor estará vazia, isto é, disponível para armazenar dados de um novo aluno, **se o valor do seu elemento for nulo**.

Por essa razão, devemos criar uma função para inicializar **NULL** para cada elemento do vetor (tabela tabAlu).

Função de inicialização

```
void inicializa()
{
    int i;
    for(i=0; i<MAX; i++)
    {
     tabAlu[i] = NULL;
    }
}
```

Implementação

Informações dos alunos fornecidas via teclado;

Posição onde os dados serão armazenados será passada pela função.

Se a posição da tabela estiver vazia, aloca-se uma nova estrutura, senão atualiza a estrutura já apontada pelo ponteiro.

A função ***malloc()*,** do programa a seguir, aloca uma quantidade de bytes equivalente ao tamanho da estrutura **aluno**.

Função: void ler(int i)

```
void ler(int i)
{
  if(tabAlu[i]==NULL)
        tabAlu[i] = (ptr_aluno)malloc(sizeof(ptr_aluno));
   fflush(stdin);
   printf("Digite o nome: \n");
   gets(tabAlu[i]->nome);
   printf("Digite a matricula: \n");
   scanf("%ld", &tabAlu[i]->mat);
   fflush(stdin);
   printf("Digite o endereco: \n");
   gets(tabAlu[i]->endereco);
   fflush(stdin);
   printf("Digite o telefone: \n");
   gets(tabAlu[i]->fone);
   printf("Digite a nota: \n");
   scanf("%f", &tabAlu[i]->nota);
}
```

Função para impressão dos dados

```
void imprime(int i)
{
 if(tabAlu[i] != NULL)
 {
  printf("Nome:       %s \n", tabAlu[i]->nome);
  printf("Matricula: %d \n", tabAlu[i]->mat);
  printf("Endereco:  %s \n", tabAlu[i]->endereco);
  printf("Telefone:  %s \n", tabAlu[i]->fone);
  printf("Nota:       %f \n", tabAlu[i]->nota);
  printf("\n");
 }
}
```

Função que imprime todos os dados

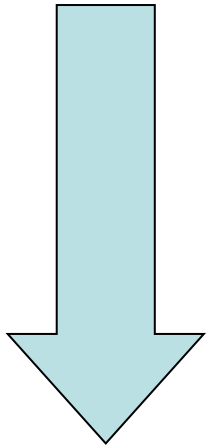
```
void imp_tudo()
{
   int i;
   for(i=0; i<MAX; i++)
   {
      imprime(i);
   }
}
```

Função principal

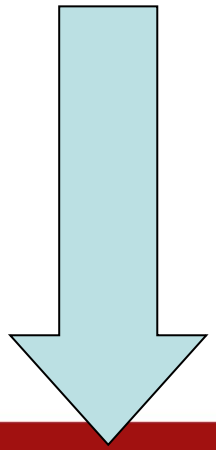
```
int main(int argc, char *argv[])
{
  //Inicializa com null para todas as pos. do vetor
  inicializa();

  int p;
  printf("Qual posicao deseja preencher: \n");
  scanf("%d", &p);
  ler(p);

  printf("Vou imprimir tudo...\n");
  imp_tudo();

  system("PAUSE");
  return 0;
}
```

Completo

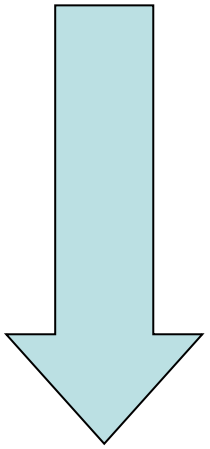
```
#include <stdio.h>
#include <stdlib.h>
struct aluno{
        char nome[80];
        long mat;
        char endereco[120];
        char fone[20];
        float nota;
};
typedef struct aluno *ptr_aluno;
#define MAX 100
ptr_aluno tabAlu[MAX];

void inicializa();
void ler(int i);
void imprime(int i);
void imp_tudo();
```



```
int main(int argc, char *argv[])
{
  //Inicializa com null para todas as pos. do vetor
  inicializa();

  int p;
  printf("Qual posicao deseja preencher: \n");
  scanf("%d", &p);
  ler(p);

  printf("Vou imprimir tudo...\n");
  imp_tudo();

  system("PAUSE");
  return 0;
}
```



```
void inicializa()
{
   int i;
   for(i=0; i<MAX; i++)
   {
    tabAlu[i] = NULL;
   }
}
void ler(int i)
{
  if(tabAlu[i]==NULL)
  {
   tabAlu[i] = (ptr_aluno)malloc(sizeof(struct aluno));
   fflush(stdin);
   printf("Digite o nome: \n");
   gets(tabAlu[i]->nome);
   printf("Digite a matricula: \n");
   scanf("%ld", &tabAlu[i]->mat);
   fflush(stdin);
   printf("Digite o endereco: \n");
   gets(tabAlu[i]->endereco);
   fflush(stdin);
   printf("Digite o telefone: \n");
   gets(tabAlu[i]->fone);
   printf("Digite a nota: \n");
   scanf("%f", &tabAlu[i]->nota);
  }
}
```



```
void imprime(int i)
{
 if(tabAlu[i] != NULL)
 {
  printf("Nome:       %s \n", tabAlu[i]->nome);
  printf("Matricula: %8d \n", tabAlu[i]->mat);
  printf("Endereco:  %s \n", tabAlu[i]->endereco);
  printf("Telefone:  %s \n", tabAlu[i]->fone);
  printf("Nota:       %f \n", tabAlu[i]->nota);
  printf("\n");
 }
}
void imp_tudo()
{
  int i;
  for(i=0; i<MAX; i++)
  {
    imprime(i);
  }
}
```

Fim do programa.

Definição de Tipos

Além da utilização do comando *struct*, vimos como utilizar o *typedef* para definir estruturas heterogêneas como novos tipos de dados.

O uso do *typedef* é muito útil para abreviarmos nomes de tipos e tratarmos tipos complexos. Alguns exemplos:

```
typedef unsigned char Caracter;
typedef int* Inteiro;
typedef float vet[4];
```

Uso

A partir das definições vistas no slide anterior, podemos declarar variáveis usando estes mnemônicos:

```
vet v;
...
V[0] = 3;
V[1] = 4;
....
```

Como visto nesta aula, em geral usamos typedef para definirmos nomes de tipo para as estruturas complexas:

```
struct ponto{
 float x;
 float y;
};
typedef struct ponto Ponto;
```

Ou ponteiros

```
typedef struct ponto *PPonto;
```

Exercícios com Estruturas Homogêneas e Heterogêneas

Desafios:

- 1) Alterar o programa para que o usuário informe primeiro o número de ALUNOS a cadastrar.
- 1a) Somente após cadastrar todos é que se deseja imprimir os ALUNOS cadastrados no vetor.

- 2) Dada uma sequencia de n valores naturais inteiros. Calcular o quadrado do subconjunto de **n que são ímpares**.

- 3) Declarar uma estrutura chamada notas com a seguinte estrutura:
  - **float nota1, nota2;**
  - **Crie um programa para cadastrar as notas (duas) de dez alunos.**
  - **Ao final mostrar a menor e maior média.**

- 4) Definir uma estrutura qualquer com as seguintes variáveis: nome e número. Em seguida receber dados via teclado para os membros da estrutura. Mostrar os valores atribuídos à estrutura na tela.

Prof. Me. Marcos Roberto de Moraes, o Maromo

FIM

Referências Bibliográficas

DAMAS, L. M. D. **Linguagem C**. LTC, 2007.

HERBERT, S. **C completo e total**. 3a. ed. Pearson, 1997.

SILVA, Osmar Q. **Estrutura de Dados e Algoritmos usando C – Fundamentos e Aplicações**. Rio de Janeiro-RJ: Editora Ciência Moderna, 2007, 460p.