Algoritmos e Estruturas de Dados I

Estrutura / Struct / Registro

Prof. Ivre Marjorie

Introdução

- Problema: agrupar em um único nome um conjunto de tipos de dados **não** similares
  - O problema de agrupar dados desiguais em C é resolvido com estruturas
- Estruturas: tipos de variáveis que agrupam dados geralmente desiguais
  - Já as matrizes agrupam dados similares, por isso, são ditas homogêneas
- Os itens de dados da estrutura são chamados de membros, e os da matriz, de elementos

Introdução

- Até o momento, usamos apenas os tipos de dados simples, ou seja, que já estão pré definidos no compilador
- Com uma estrutura é possível criar um novo tipo de dados definido por nós, ou seja, que inclui o que quisermos
- Por exemplo, podemos criar uma estrutura **Folha_de_Pagamento,** que irá incluir: o nome do funcionário (string), o número do departamento (inteiro) e o salário (float)

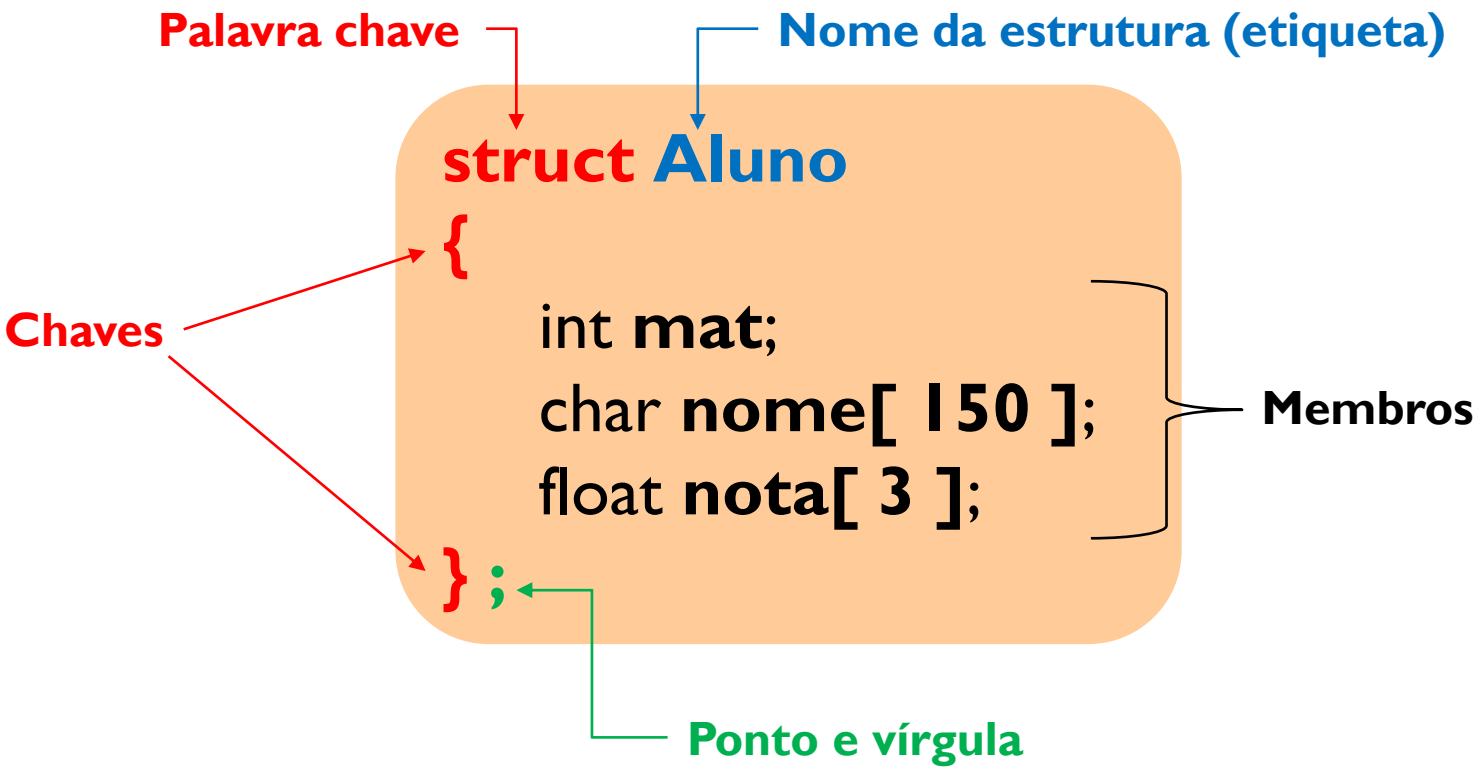
Criando novos tipos de dados

- Por meio da palavra chave **struct** definimos um novo tipo de dado
  - Definir um novo tipo de dado, significa informar ao compilador seu nome, tamanho em bytes e forma como deve ser armazenado e recuperado da memória
- Após ter sido definido, o novo tipo existe e pode ser utilizado para criar variáveis de modo similar a qualquer tipo simples
- Exemplo:

```
struct Aluno
{
    int mat;
    char nome[ 150 ];
    float nota[ 3 ];
} ;
```

Definindo uma estrutura

- A definição da estrutura informa como ela é organizada e quais são seus membros:



Definindo uma estrutura

- A definição de uma estrutura não cria nenhuma variável, somente informa ao compilador as características de um novo tipo de dado
  - **Não** há nenhuma reserva de memória
- A palavra struct indica que um novo tipo de dado está sendo definido Aluno (no exemplo) será o nome da estrutura ou etiqueta
  - O nome do nosso novo tipo de dados é **struct Aluno**
- Vamos definir nossa estrutura antes da **main()**, o que permite um acesso global a todas as funções definidas no programa

Definindo uma estrutura

- A definição ficará antes da main():

```
struct Aluno
{
    ....
} ;

int main()
{
    struct Aluno alu;
    ....
}
```

Declarando uma variável

- Para declarar uma variável do tipo de dado definido, vamos usar a seguinte instrução:

```
int main()
{
   struct NomeEstrutura variável;
}
```

- No nosso exemplo:

```
struct Aluno alu1, alu2;
```

Aqui foram criadas duas variáveis (alu1 e alu2) do tipo de dados Aluno

Typedef

- Declarações com typedef não produzem novos tipos de dados
  - Criam apenas novos nomes (sinônimos) para os tipos de dados existentes, podendo ser uma estrutura
- Sintaxe:

```
typedef  tipo-existente  sinônimo;
```

- Exemplos:

```
typedef  char  Byte;
typedef  int  uint;
```

Foi definido outro nome para o tipo char e para o tipo int

Typedef

- É possível definir outro nome para uma estrutura, ou então definir o mesmo nome usando o typedef

```
typedef struct NomeEstrutura
{
    tipo de dado nomeMembro;
    tipo de dado nomeMembro;
    tipo de dado N nomeMembroN;
} NomeEstruturaNovo ;
```

- Mas porque alterar o nome de uma estrutura?
  - A declaração de uma variável para esse novo tipo ficará mais simplificada

Declarando uma variável usando o typedef

- Definindo um novo nome para a estrutura:

```
typedef struct Aluno
{
    int mat;
    char nome[ 150 ];
    float nota[ 3 ];
} AlunoNovo ;
```

- Declarando a variável:

```
int main()
{
  AlunoNovo variável;
}
```

Observe que aqui basta colocar o novo nome da estrutura, não é necessário usar a palavra chave **struct**

Acessando os membros

- Para acessar os membros de uma estrutura, usamos o operador . (ponto) e a **variável** do tipo da estrutura, já declarada:

```
int main()
{
  NomeEstruturaNovo variável;
  variável. nomeMembro1 = valor;
  scanf("%tipo_de_dado", & variável. nomeMembro2);
}
```

Exemplo 1

```
typedef struct Aluno
{
    int mat;
    char nome[ 250 ];
    float nota[ 3 ], media;
} AlunoNovo;
typedef int novoint;
//usando typedef
```

```
int main()
{
    //struct Aluno alu; é possível usar assim sem typedef
    AlunoNovo alu; // ou assim com typedef
    float soma=0;
    novoint i;
    alu.mat = rand()%100;  //numero aleatorio de 0 a 99 para a matricula
    printf("Digite o nome do aluno: ");
    gets(alu.nome);
    for(i=0;i<3;i++)
    {
        printf("Digite a nota %d: ", i+1);
        scanf("%f", &alu.nota[ i ]);
        soma += alu.nota[ i ];
    }
    alu.media = soma/3;
    printf("O aluno %s matricula %d teve media %.2f!",alu.nome,alu.mat,alu.media);
    return 0;
}
```

Vetor e Matriz de Estruturas

- É possível declarar um vetor e/ou matriz do tipo da estrutura definida
- A declaração é semelhante a usada para os tipos de dados usados até o momento:

```
Estrutura nomeVetor [ tamanho ] ;
```

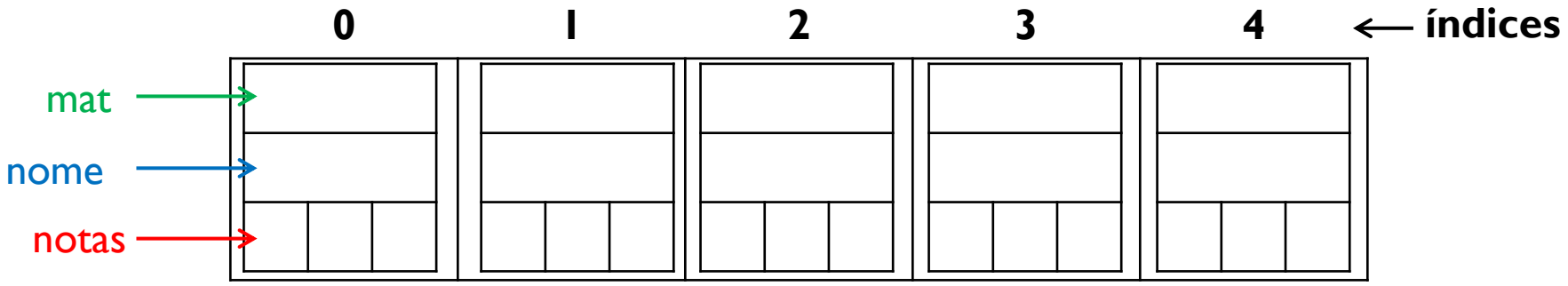
- E para o caso de matrizes:

```
Estrutura nomeMatriz [ linhas ] [ colunas] ;
```

Vetor e Matriz de Estruturas

- Cada posição do vetor ou da matriz irá conter todos os membros definidos na estrutura, observe o exemplo, considerando um vetor:

```
int main()
{
  AlunoNovo vet[ 5 ];
  ...
}
```



Vetor e Matriz de Estruturas

- Para acessar os membros da estrutura com vetor também usaremos o operador ponto (.)
- A principal diferença é que precisamos indicar em qual posição do vetor (índice) será armazenado cada dado, isso no caso de entrada de dados
- Continuamos precisando de repetição para percorrer o vetor e ir armazenando os dados em cada índice

```
for ( i = 0; i < tamanho; i ++)
 {
   scanf( "%tipoDeDadoDoMembro", &nomeVetor[ i ].membroEstrutura);
 }
```

Exemplo 2 – Parte 1

```
typedef struct Aluno{
 int mat;
 char nome[150];
 float nota[3];
}AlunoNovo;
#define tamanho 3
int main()
{
    AlunoNovo alu[tamanho];
     int i, j;
     for(i=0; i <tamanho; i++)
     {
       alu[i].mat=rand()%100;
       printf("Digite o nome:");
       gets(alu[i].nome);
       gets(alu[i].nome);

// continua ...
```

Exemplo 2 – Parte 2

```
for(j=0; j<3; j++)
      {
        printf("Digite a nota:");
        scanf("%f", &alu[i].nota[j]);
      }
    }
    printf("\n");
    printf("******************** Cadastro de Aluno ********************");
    for(i=0; i <tamanho; i++)
     {
       printf("\nMatricula: %d",alu[i].mat);
       printf("\nNome: %s",alu[i].nome);
       for(j=0; j<3; j++)
        {
         printf("\nNota %d = ",(j+1),alu[i].nota[j]);
        }
     }
    return 0;
}
```

Exemplo 2 - Saída

```
******************** Cadastro de Aluno ********************
Matricula: 41
Nome:
Nota 1 =
Nota 2 =
Nota 3 =
Matricula: 67
Nome: Joana
Nota 1 =
Nota 2 =
Nota 3 =
Matricula: 34
Nome: Pedro
Nota 1 =
Nota 2 =
Nota 3 =
Process returned 0 (0x0)   execution time : 17.207 s
Press any key to continue.
```

Função e estrutura

- A linguagem C permite que as funções retornem uma estrutura completa para outra função
  - Para isso, basta criar uma estrutura do tipo da estrutura e não esquecer de incluir o return (retornando uma estrutura)
- Além disso, é possível que os parâmetros de uma função sejam de um tipo de dados criado, ou seja, do tipo de uma estrutura
  - Para isso, basta indicar na criação o tipo de dado do parâmetro, assim como fazemos com os tipos de dados simples
- Observe o exemplo 2

Exemplo 3

```
typedef struct Venda
{
 int pecas;
 float preco;
} Venda;
Venda TotalVendas(Venda C, Venda D);

int main()
{
    Venda A, B, Total;
    printf("\nVenda A");
    printf("\nDigite a quantidade de pecas: ");
    scanf("%d", &A.pecas);
    printf("\nDigite o preco: ");
    scanf("%f", &A.preco);
    printf("\nVenda B");
    //continua ...
```

Exemplo 3

```
    //...continuação
    printf("\nDigite a quantidade de pecas: ");
    scanf("%d", &B.pecas);
    printf("\nDigite o preco: ");
    scanf("%f", &B.preco);
    Total = TotalVendas(A, B);
    printf("\n\nTotal das vendas: \nPecas: %d \nPreco: %.2f", Total.pecas, Total.preco);
    return 0;
}

Venda TotalVendas(Venda C, Venda D)
{
    Venda T;
    T.pecas = C.pecas + D.pecas;
    T.preco = C.preco + D.preco;
    return T;
}
```

Estrutura de estrutura

- É possível definir estruturas com membros que sejam outras estruturas

```
typedef struct NomeEstrutura1
{
    tipo de dado nomeMembro;
    tipo de dado N nomeMembroN;
} NomeEstruturaNovo1 ;
```

```
typedef struct NomeEstrutura2
{
    tipo de dado nomeMembro1;
    NomeEstruturaNovo1 nomeMembro2;
} NomeEstruturaNovo2 ;
```

Observe que o membro2 dessa estrutura é do tipo da estrutura anterior

Exemplo 4

```
typedef struct Data
{
  int dia;
  int mes;
  int ano;
}Data;

typedef struct Aluno
{
  Data dtanasc;
  int mat;
  char nome[250];
}Aluno;

//continua ...
```

Exemplo 4

```
int main()
{
   Aluno A;
  printf("Cadastro Aluno: ");
  printf("\nData de nascimento - dia: ");
  scanf("%d", &A.dtanasc.dia);
  printf("\nData de nascimento - mes: ");
  scanf("%d", &A.dtanasc.mes);
  printf("\nData de nascimento - ano: ");
  scanf("%d", &A.dtanasc.ano);
  printf("\nDigite a matricula: ");
  scanf("%d", &A.mat);
  printf("\nDigite o nome: ");
  gets( A.nome);
  gets( A.nome);
  printf("\n\n**********************");
  printf("\nAluno: %s Matricula:%d",A.nome,A.mat);
  printf("\nData de nascimento: %d / %d / %d ",A.dtanasc.dia,A.dtanasc.mes,A.dtanasc.ano);
  return 0;
}
```

Exemplo 5

```
typedef struct Data
{
  int dia;
  int mes;
  int ano;
}Data;

typedef struct Aluno
{
  Data dtanasc;
  int mat;
  char nome[250];
}Aluno;

Aluno CadastraAluno(Aluno A);
//continua ....
```

Exemplo 5

```
int main()
{
  Aluno Alu, Alu1;
  Alu1 = CadastraAluno(Alu);
  printf("\n\n*****************");
  printf("\nAluno: %s Matricula:%d",
Alu1.nome, Alu1.mat);
  printf("\nData de nascimento: %d / %d /
%d ",Alu1.dtanasc.dia, Alu1.dtanasc.mes,
Alu1.dtanasc.ano);
  return 0;
}
```

```
Aluno CadastraAluno(Aluno A)
{
  printf("Cadastro Aluno: ");
  printf("\nData de nascimento - dia: ");
  scanf("%d", &A.dtanasc.dia);
  printf("\nData de nascimento - mes: ");
  scanf("%d", &A.dtanasc.mes);
  printf("\nData de nascimento - ano: ");
  scanf("%d", &A.dtanasc.ano);
  printf("\nDigite a matricula: ");
  scanf("%d", &A.mat);
  printf("\nDigite o nome: ");
  gets(A.nome);
  gets(A.nome);
  return A;
}
```

Exemplo 6

```
#include <stdio.h>
#include <stdlib.h>
#define TAM 3
typedef struct Data
{
  int dia;
  int mes;
  int ano;
}Data;

typedef struct Empregado
{
  Data dtanasc;
  int mat;
  char nome[250];
  float salario;
}Empregado;
```

Exemplo 6

```
int main()
{
  Empregado Emp[TAM];
  int i;
  for(i=0;i<TAM; i++)
  {
    printf("\nCadastro Empregado: ");
    printf("\nData de nascimento - dia: ");
    scanf("%d", &Emp[i].dtanasc.dia);
    printf("\nData de nascimento - mes: ");
    scanf("%d", &Emp[i].dtanasc.mes);
    printf("\nData de nascimento - ano: ");
    scanf("%d", &Emp[i].dtanasc.ano);
    Emp[i].mat = (i+1);
    printf("\nDigite o nome: ");
    gets(Emp[i].nome);
    gets(Emp[i].nome);
    printf("\nDigite o salario: ");
    scanf("%f", &Emp[i].salario);
  }
  //continua …
```

Exemplo 6

```
//…continuação
  for(i=0;i<TAM; i++)
  {
  printf("\n\n*****************");
  printf("\nEmpregado: %s Matricula:%d", Emp[i].nome, Emp[i].mat);
  printf("\nData de nascimento: %d / %d / %d ",Emp[i].dtanasc.dia, Emp[i].dtanasc.mes,
Emp[i].dtanasc.ano);
  printf("\nSalario: %.2f", Emp[i].salario);
  }
  return 0;
}
```

Exercícios

1. Faça download do arquivo texto disponível no SGA (**empregados.txt**), e do programa, **exercicio1.c**.

Crie as seguintes funções:

a) função para calcular a média salarial (função com retorno)
b) função para identificar o empregado com o maior salário
c) função para identificar o empregado com o menor salário
d) função que identifica o empregado com salário acima da média (chame dentro dessa função a função para calcular a média salarial)

Exercícios

2. Uma empresa possui 18 funcionários, sobre os quais se tem estas informações: nome, número de horas trabalhadas no mês, turno de trabalho (M – matutino, V vespertino ou N – noturno), categoria (O – operário, G- Gerente) e valor da hora trabalhada. Sabendo-se que essa empresa deseja informatizar sua folha de pagamento, faça um programa que leia o nome, número de horas trabalhadas no mês, o turno e a categoria dos funcionários, não permitindo que sejam informados turnos e categorias inexistentes. O programa deverá calcular o valor da hora trabalhada, conforme a tabela a seguir, adotando o valor de R$ 950,00 para o salário mínimo.

Exercícios

Categoria	Turno	Valor da hora trabalhada
G	N	18% do salário mínimo
G	M ou V	15% do salário mínimo
O	N	13% do salário mínimo
O	M ou V	10% do salário mínimo

- O programa deverá calcular o salário inicial dos funcionários com base no valor da hora e no número de horas trabalhadas. Todos recebem um auxílio alimentação, de acordo com o seu salário inicial, conforme tabela a seguir:

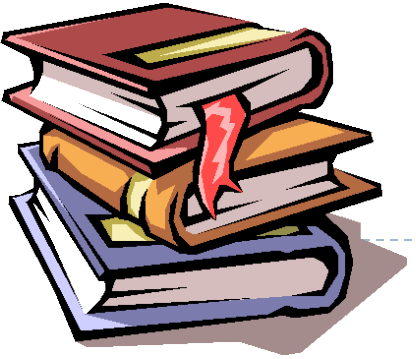
Exercícios

Salário inicial	Auxílio alimentação
Até R$ 950,00	20% do salário inicial
De R$ 950,00 a R$ 1500,00	15% do salário inicial
Acima de R$ 1500,00	5% do salário inicial

- O programa deverá mostrar o nome, o número de horas trabalhadas e o valor da hora trabalhada, o salário inicial, o auxilio alimentação e o salário final (salário inicial + auxilio alimentação) de todos os funcionários. Use funções e estruturas em seu programa. Caso ache melhor inclua um menu de opções.

Exercícios

Categoria	Turno	Valor da hora trabalhada
G	N	18% do salário mínimo
G	M ou V	15% do salário mínimo
O	N	13% do salário mínimo
O	M ou V	10% do salário mínimo



Referência Bibliográfica

- MIZRAHI, Victorine Viviane. **Treinamento em linguagem C.** São Paulo:Pearson Prentice Hall, 2008. 2ª edição. Curso Completo. Capítulo 8.
- ASCENCIO, Ana Fernanda Gomes e CAMPOS, Edilene A. Veneruchi. **Fundamentos da Programação de Computadores – Algoritmos, Pascal, C/C++ e Java**. São Paulo: Pearson Prentice Hall, 2012. 3ª Edição.