Laboratório de Algoritmos I
Novos Tipos de Dados em C: struct e typedef

Introdução

- Problema: agrupar dados relacionados (mas de tipos diferentes) sob um mesmo nome.

- Solução em C: estruturas!

- Estruturas são tipos de variáveis que agrupam dados geralmente desiguais, ao passo que matrizes agrupam dados similares. Os itens de dados da estrutura são chamados de **membros**, e os da matriz, de **elementos**.

struct

- Por meio desta palavra-chave é que criamos uma estrutura em C, ou seja, definimos um novo tipo de dado.

- Definir um tipo de dado significa informar ao compilador seu nome, tamanho em bytes e forma como deve ser armazenado e recuperado da memória.

struct - Exemplo

```
struct Aluno {
  int nmat;
  float nota[3];
  float media;
};
```

struct

- Esta definição pode ser escrita fora de qualquer função (acesso global), como dentro de uma função (acesso local).
- Após ter sido definido, o novo tipo existe e pode ser utilizado para criar variáveis de modo similar a qualquer tipo simples.
- Para acessar os membros de uma estrutura, deve ser utilizado o operador ponto (“.”).

struct – Exemplo de uso

```
int main(){
  struct Aluno Jose;
  Jose.nmat = 456;
  Jose.nota[0] = 7.5;
  Jose.nota[1] = 5.2;
  Jose.nota[2] = 8.4;
  Jose.media = (Jose.nota[0] +
     Jose.nota[1] + Jose.nota[2])/3;
  printf("Matricula: %d\n", Jose.nmat);
  printf("Media: %.2f\n", Jose.media);
  return 0;
}
```

Novos nomes para tipos - typedef

- O comando **typedef** em C é utilizado para dar novos nomes (sinônimos) a um tipo existente.

- Declarações com **typedef** não criam novos tipos. Apenas criam sinônimos para tipos existentes.

- Sintaxe:

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

- Exemplos:

```
typedef unsigned char BYTE;
typedef unsigned int uint;
```

typedef – declarando variáveis

- Exemplos:

```
int main(){
  BYTE ch;
  uint x;
  unsigned char chmm; /* os tipos originais
  continuam disponíveis*/
}
```

Usando typedef com struct – Opção 1

```
struct Aluno
{
  int nmat;
  float nota[3];
  float media;
};
typedef struct Aluno Aluno;
Aluno Jose; //declarando variável
```

Usando typedef com struct – Opção 2

```
typedef struct Aluno
{
  int nmat;
  float nota[3];
  float media;
} Aluno;

Aluno Jose; //declarando variável
```

Usando typedef com struct – Opção 3

```
typedef struct
{
  int nmat;
  float nota[3];
  float media;
} Aluno;

Aluno Jose; //declarando variável
```

Inicializando estruturas

- Assim como vetores, é possível inicializar estruturas em uma só linha, **no momento de sua declaração**!
  - Basta passar o valor dos membros da estrutura, em ordem, entre chaves.

- Por exemplo:

```
typedef struct {
  int dia;
  char mes[10];
  int ano;
} Data;
Data natal = {25, “dezembro”, 2009};
```

Atribuições entre estruturas

- Uma variável estrutura pode ser atribuída a outra do mesmo tipo por meio de uma atribuição simples:

```
Data aniversario = {30, "julho", 2009};
Data Andre;
Andre = aniversario;
```

- Nesse exemplo, a variável Andre passa a ter os mesmos valores da variável aniversario para todos os seus membros.

Operações entre estruturas

- Na linguagem C, operações simples como a soma não estão definidas para tipos criados com a palavra struct. A soma deve ser efetuada membro a membro.

- Por exemplo:

```
typedef struct {
   int x;
   int y;
} Num;
Num x = {1, 2};
Num y = {3, 4};
Num z = x + y; //ERRADO!!
```

- A operação mostrada acima não existe, e não é correta!

Estruturas aninhadas

- Estruturas podem ser aninhadas. Nesses casos, a estrutura a ser usada dentro de outra deve ser declarada primeiro!

```
typedef struct {
  int dia;
  char mes[10];
  int ano;
} Data;

typedef struct {
  int pecas;
  float preco;
  Data diavenda;
} Venda;
```

Estruturas aninhadas

```
int main() {
  Venda A = {20, 110.0, {7, “novembro”,
     2008}};
  printf(“Data: %d de %s de %d\n”,
     A.diavenda.dia, A.diavenda.mes,
     A.diavenda.ano);
  return 0;
}
```

Saída:

```
Data: 7 de novembro de 2008
```

Estruturas e funções

- Na linguagem C, as estruturas podem ser passadas como parâmetros de funções da mesma forma que variáveis dos tipos primitivos.

- Estruturas também podem ser o tipo de retorno de uma função.

Estruturas e funções

- Por exemplo:
  - Dada a seguinte estrutura, faça uma função que receba dois alunos e retorne o aluno de maior média.

```
typedef struct {
  int nmat;
  float nota[3];
  float media;
} Aluno;
```

Estruturas e funções

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int nmat;
  float nota[3];
  float media;
} Aluno;

Aluno maiorMedia(Aluno x, Aluno y) {
  if (x.media > y.media) {
    return x;
  } else {
    return y;
  }
}

int main() {
  Aluno x = {123, {10, 10, 10}, 10.0};
  Aluno y = {456, {9, 9, 9}, 9.0};
  Aluno z = maiorMedia(x, y);
  printf("O aluno de maior media tem numero de matricula: %d.\n", z.nmat);
  return 0;
}
```

Matrizes de estruturas

- Uma matriz de estruturas pode ser declarada e utilizada da mesma forma que uma matriz de tipos primitivos.

```
Venda vendas[50];  /* declara uma matriz
  cujo rótulo é "vendas" com 50 posições
  para armazenar variáveis do tipo "Venda",
  definido anteriormente */
vendas[1].pecas = 10; /* Para acessar os
  membros da estrutura devemos escolher o
  indice do vetor e, então, acessar seus
  membros */
vendas[1].preco = 5.50;
```