LP1

Prof. Luciano Bernardes de Paula

Já vimos:

- Tipos de variáveis;
- Criação de variáveis;
- Utilização de variáveis;
- Criação de variáveis com vários valores do mesmo tipo (vetores e matrizes!!);

Como criar uma única variável que armazene vários valores de **diferentes tipos?**

Exemplo:

- Como armazenar os dados de uma pessoa?

Nome;

Idade;

Altura;

Peso;

Structs

Em programação é comum a necessidade de se agrupar um conjunto de variáveis.

Se é necessário agrupar variáveis do mesmo tipo, é possível utilizar vetores.

Entretanto, se for necessário agrupar variáveis de diferentes tipos, é preciso utilizar estruturas (*structs*).

Uma estrutura (***struct***) agrupa variáveis de diferentes tipos, que podem representar vários dados sobre um mesmo elemento.

Por exemplo: uma estrutura “**Pessoa**” poderia armazenar o nome, idade, peso, altura, etc, de uma pessoa.

Definindo uma struct

```
#include <stdio.h>

typedef struct
{
   char nome[25];
   int idade;
   float peso;
   float altura;
} Pessoa;

int main(){

   Pessoa p1;
   …
}
```

Acessando membros de uma estrutura

```
p1.idade = 25;
```

O operador “.” conecta o nome de uma variável do tipo estrutura a um membro dela.

Inicializando estruturas

```
Pessoa p1 = {“João da Silva”, 25, 75.5, 1.80};
```

Outro exemplo

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

Data nascimento = {25, “dezembro”, 2000};
```

Atribuições entre estruturas

É possível atribuir uma estrutura a outra do mesmo tipo.

```
Data nascimento = {25, “dezembro”, 2000};
Data novaData;

novaData = nascimento;          //CORRETO!
```

Operações entre estruturas

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

Venda v1 = {20, 110.0};
Venda v2 = {3, 16.5};
Venda total;
```

```
total = v1 + v2; // ERRADO!

total.pecas = v1.pecas + v2.pecas; // CORRETO

total.preco = v1.preco + v2.preco;
```

Estruturas aninhadas

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

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

Atribuição:

```
Venda v;

v.pecas = 10;
v.preco = 5.0;
v.diavenda.dia = 10;
strncpy(v.diavenda.mes, “janeiro”, sizeof(v.diavenda.mes)
   - 1);
v.diavenda.mes[7] = ‘\0’;
v.diavenda.ano = 2015;
```

Inicializando estruturas aninhadas

```
Venda A = { 20, 110.0, { 7, “novembro”, 2015 } };
```

É possível declarar um vetor de estruturas

```
Pessoa p[10];
```

Para acessar um membro de uma estrutura em um vetor:

```
p[0].idade = 30;
```

Exemplo:

- Faça um programa que receba os dados de 3 livros (título, nome do autor, número de páginas) e depois os apresente na tela;

Funções

Uma função é um conjunto de instruções criadas para cumprir uma tarefa.

Por exemplo a função **printf** foi escrita para mostrar caracteres na tela.

Não é preciso reescrevê-la toda vez, simplesmente utilizá-la, pois já se encontra implementada.

Uma função simples

```
#include <stdio.h>

float media(float a, float b){
   return (a + b) / 2;
}

int main(){
   float x = 1, y = 2, z;
   z = media(x, y);

   return 0;
}
```

A palavra chave **return** define o valor retornado pela função. Uma limitação é que somente um valor é retornado.

Ao retornar um valor, a função é finalizada.

É possível escrever funções que não retornam valores, sendo o seu tipo **void**.

Em C é possível organizar um programa e reaproveitar funções criando bibliotecas.

Para isso é necessário criar um arquivo de "*headers*" **.h** que contém somente os protótipos das funções e arquivos **.c** que contém a implementação das funções.

O protótipo é necessário para que o compilador entenda tudo que for requerido para que a função seja utilizada corretamente (tipo de retorno e tipo de argumento).

O arquivo .h deve conter os protótipos das funções.

```
tipo_de_retorno nomeDaFunção(tipo argumentos);
```

Exemplo - Arquivo operacoes.h

```
float media(float a, float b);
int soma(int a, int b);
int sub(int a, int b);
```

Exemplo:

Arquivo operacoes.c:

```
-----
#include <stdio.h>
#include “operacoes.h"

int soma(int a, int b){
    return a + b;
}

int sub(int a, int b){
   return a - b;
}

float media(float a, float b){
   return (a + b) / 2;
}

-----
```

Em um programa qualquer, é possível incluir a biblioteca e usar as funções:

```
#include <stdio.h>
#include “operacoes.h"

int main(){

   int a = 5, b = 10, c, d;
   float f = 5.25, g = 3.75, h;

    c = soma(a, b);
    d = sub(a, b);
    h = media(f, g);
   …
}
```

Para que seja feita a “linkagem” entre o fonte do programa e os arquivos de headers e os fontes das funções, é preciso indicar ao compilador no início da compilação do programa.

No Dev-C++, basta vc criar um “Projeto” e colocar todos esses arquivos na pasta do projeto criado.

O Dev-C++ possui também um diretório de includes, que ele usa para todos os projetos.

Chamando funções

a = f1(x, y);

A função *f1* recebe dois valores *x* e *y* e retorna um valor que será armazenado em *a*.

b = f2();

A função *f2* não recebe nenhum parâmetro e retorna um valor que será armazenado em *b*.

f3();

A função *f3* não recebe nenhum parâmetro e não retorna nenhum valor.

f4(i, j, k);

A função f4 recebe três parâmetros e não retorna nada.

Uso de vetores como parâmetro

Protótipo

```
int somaVet(int vetor[], int tamanho);
```

Código

```
int somaVet(int vetor[], int tamanho){
 int i, soma = 0;
 for(i = 0; i < tamanho; i++) soma = soma +
  vetor[i];
 return soma;
}
```

Protótipo

```
float mediaVet(int vetor[], int tamanho);
```

Código

```
float mediaVet(int vetor[], int tamanho){
 int i, soma = 0;
 for(i = 0; i < tamanho; i++) soma = soma +
  vetor[i];
 return soma / tamanho;
}
```

Passagem de argumentos por valor

Quando uma variável é passada para uma função por valor, seu valor é copiado em um novo espaço de memória e assim é utilizado.

A variável original não é alterada.

Passagem de argumento por referência

Quando uma variável é passada para uma função por referência, seu endereço é passado para a função, dessa forma a variável original é alterada.

É possível em C, mas deve se usar ponteiros (veremos nas adiante na disciplina).

Variáveis são sempre passadas por valor (a não ser que passamos um ponteiro, que veremos nas próximas aulas).

Porém, vetores e matrizes são sempre passadas por referência.

Exemplo 1

```
void maismais(int a){
     a++;
}

Se fizermos o seguinte uso:
...
aux = 10;
maismais(aux);
printf(“%d”, aux);
...
```

O valor de aux ainda será 10, pois foi feita uma passagem por valor.

Exemplo 2

```
void zeraVetor(int vetor[], int tamanho){
  int i;
  for(i = 0; i < tamanho; i++) vetor[i] = 0;
}

Se for usado
...
int vetor[5];
zeraVetor(vetor, 5);
...
```

Nesse caso os valores em vetor serão alterados.

Passagem de matriz para função

Para passar uma matriz com 2 ou mais dimensões, deve ser feito da seguinte forma:

Protótipo

```
void zeraMatriz(int mat[][COL], int linha, int col);
```

Código

```
void zeraMatriz(int mat[][COL], int linha, int col){
   int i, j;

   for(i = 0; i < linha; i++){
      for(j = 0; j < col; j++){
        mat[i][j] = 0;
      }
   }
}
```

E para usar a função em um programa:

```
int matriz[5][5];

zeraMatriz(matriz, 5, 5);
```

Para matrizes de diferentes tamanhos é preciso ajustar os valores dos índices.

Exemplo:

Programa dos livros com funções.

Exercícios.