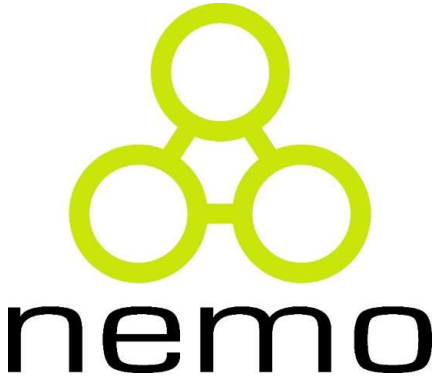


Programação II

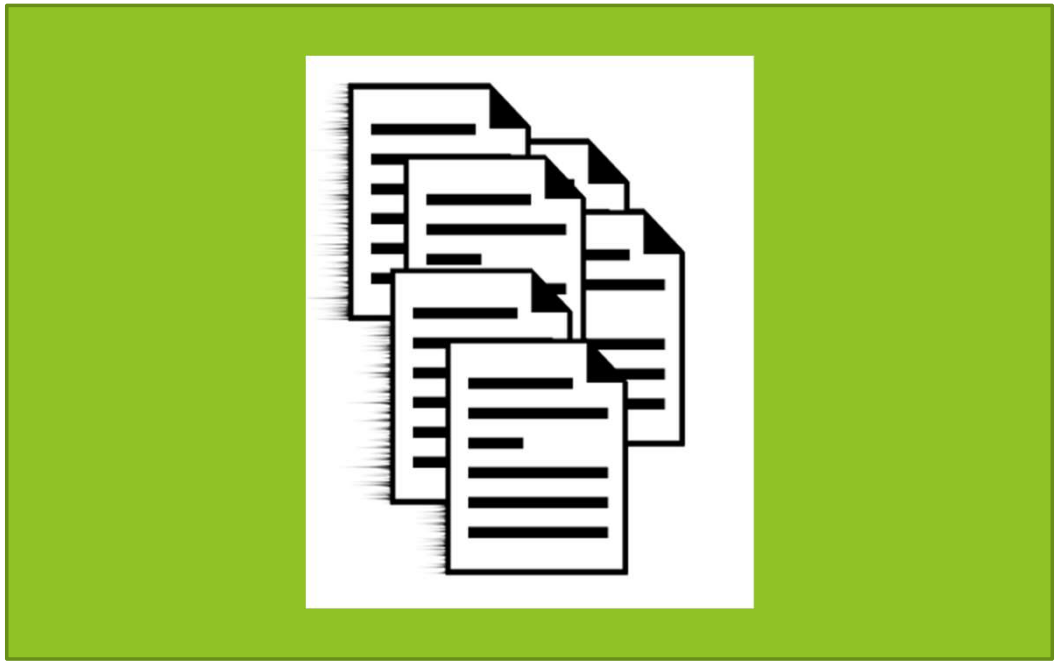
Jordana S. Salamon

jssalamon@inf.ufes.br

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO



Estruturas

nemo

Introdução

- As variáveis estudadas até o momento podem armazenar somente dados simples. Ex: int, float, char;
- Contudo, muitas vezes precisamos armazenar dados compostos, ou seja, várias variáveis que se referem ao mesmo conceito do mundo real.
- Exemplos:
  - Uma **pessoa** que possui nome, idade, peso e altura.
  - Um **aluno** que possui nome, número de matrícula e curso.
  - Um **livro** que possui título, gênero, autor(es), ISBN, etc.
  - Um **jogo** que possui nome, gênero, valor, empresa desenvolvedora.

Estruturas de Dados

- Relembre o exercício em que eram lidas informações de vários pacientes e era calculado o IMC para ver se os pacientes estavam no peso ideal ou acima do peso.
- Para representar um paciente, utilizaríamos as seguintes variáveis para armazenar as informações:
  - **char nome[30];**
  - **int idade;**
  - **float peso, altura;**

Estruturas de Dados

- Agora imagine se tivéssemos que armazenar vários pacientes na memória (ex:5) para depois calcular todos os IMCs de uma vez.
- Para representar um conjunto de pacientes, utilizaríamos as seguintes variáveis para armazenar as informações:
  - **char nome[5][30];**
  - **int idade[5];**
  - **float peso[5], altura[5];**
- O controle das variáveis nesse tipo de programa se torna difícil, uma vez que é necessário controlar vários vetores simultaneamente.

Estruturas de Dados

- Se pensarmos no mundo real, não temos conjuntos de informações separadas, elas fazem parte de um todo, que nesse exemplo é o Paciente.
- Desta forma seria melhor termos um “vetor de pacientes” ao invés de quatro vetores separados.
- Isso é possível utilizando as **estruturas de dados**.
- Para definir uma estrutura de dados utilizamos a palavra-chave **struct**.

Sintaxe - Estruturas de Dados

```
struct <nome> {
    <declaração de variáveis>
    ...
    <declaração de variáveis>
};

<declaração de variáveis> =
    <tipo> <nome>, <nome>, ...;
```

```
struct paciente {
    char nome[20];
    int idade;
    float altura, peso;
};

struct ponto {
    int x,y;
};
```

Declarando e Manipulando uma Estrutura

```c
struct paciente{
    char nome[20];
    int idade;
    float altura, peso;
};

int main(){
    struct paciente p;
    strcpy(p.nome,"Joao");
    p.idade = 28;
    p.altura = 1.91;
    p.peso = 88.0;
}
```

Typedef

```c
struct paciente{
    char nome[20];
    int idade;
    float altura, peso;
};

typedef struct paciente Paciente;

int main(){
    Paciente p;
    strcpy(p.nome,"Joao");
    p.idade = 28;
    p.altura = 1.91;
    p.peso = 88.0;
}
```

Passando uma estrutura por parâmetro

```
void imprimePaciente(Paciente pac){
    printf("Dados de Pessoa\n");
    printf("Nome = %s\n", pac.nome);
    printf("Idade = %d\n", pac.idade);
    printf("Altura = %.2f\n", pac.altura);
    printf("Peso = %.2f\n", pac.peso);
}

int main(){
    Paciente p;
    strcpy(p.nome,"Joao");
    p.idade = 28;
    p.altura = 1.91;
    p.peso = 88.0;
    imprimePaciente(p);
}
```

nemo

Retornando uma estrutura

```
Paciente criaPaciente(){
    Paciente p;
    printf("Nome paciente: ");
    scanf("%s", p.nome);
    printf("Idade paciente: ");
    scanf("%d", &p.idade);
    printf("Altura paciente: ");
    scanf("%f", &p.altura);
    printf("Peso paciente: ");
    scanf("%f", &p.peso);
    return p;
}

int main(){
    Paciente p;
    p = criaPaciente();
}
```

Vetores de Estruturas

```
int main(){
    int i;
    Paciente vetor[10];
    for(i=0;i<10;i++){
        printf("Nome paciente: ");
        scanf("%s", vetor[i].nome);
        printf("Idade paciente: ");
        scanf("%d", &vetor[i].idade);
        printf("Altura paciente: ");
        scanf("%f", &vetor[i].altura);
        printf("Peso paciente: ");
        scanf("%f", &vetor[i].peso);
    }
    for(i=0;i<10;i++){
        imprimePaciente(vetor[i]);
    }
}
```

Vetores dentro das Estruturas

```c
struct paciente{
    char nome[20];
    int idade;
    float altura, peso;
    char telefones[3][15];
};
typedef struct paciente Paciente;
int main(){
    int i;
    Paciente vetor[2];
    for(i=0;i<2;i++){
        printf("Telefone 1: ");
        scanf("%s", vetor[i].telefones[0]);
        printf("Telefone 2: ");
        scanf("%s", vetor[i].telefones[1]);
        printf("Telefone 3: ");
        scanf("%s", vetor[i].telefones[2]);
    }
}
```

Estruturas dentro das Estruturas

```c
struct endereco{
    char cidade[20];
    char bairro[20];
    int numero;
};

typedef struct endereco Endereco;

struct paciente{
    char nome[20];
    int idade;
    float altura, peso;
    Endereco endereco;
};

typedef struct paciente Paciente;
```

Estruturas dentro das Estruturas

- Podemos representar um círculo como

```
struct circulo {
  float x, y;  //centro do círculo
  float r;  //raio
}
```

- Como já temos o tipo Ponto definido:

```
 struct circulo {
  Ponto p;
  float r;
}

typedef struct circulo Circulo;
```

Estruturas dentro das Estruturas

- Para implementar uma função que determinar se um dado ponto está dentro de um círculo
  - Podemos usar a função da distância, visto que usamos o tipo ponto na definição do círculo

```
int interior (Circulo c, Ponto p)
{
   float d = distancia (c.p, p);
   return (d<c.r);
}
```

Definição de novos tipos

- Podemos criar nomes de tipos em C
  - typedef float Real;
  - Real pode ser usado como mnemônico de float

```
typedef unsigned char UChar;
typedef int PInt;
typedef float Vetor[4];
```

- Podemos declarar as seguintes variáveis:

```
Vetor v;
...
v[0] = 3;
...
```

That’s all Folks!