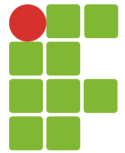
Registros

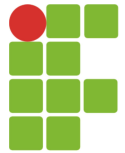
Programação de Computadores 1

Prof. Daniel Saad Nogueira Nunes

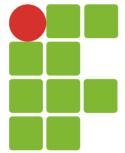
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

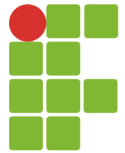


Sumário



Introdução

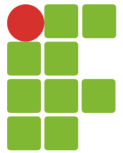
- Registros em C permitem estruturar diversas variáveis, de diferentes tipos, sob um mesmo identificador.
- Normalmente registros são utilizados quando as variáveis fazem sentido juntas, considerando o contexto da aplicação.
- Através de um único identificador, é possível realizar acesso a qualquer variável agrupada.



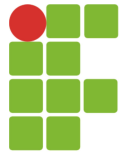
Introdução

Exemplo

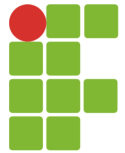
- Registro de alunos, composto de: nome, CPF, data de nascimento, dados de aprovações em disciplinas, ...
- Registros médicos, contendo os dados pessoais dos pacientes, histórico, procedimentos, medicamentos em uso, ...



Sumário



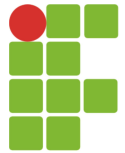
Sumário



Declaração

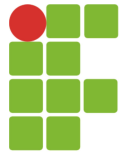
- Para criarmos um tipo de registro, utilizamos seguinte sintaxe:

```
struct nome_tipo_registro{
    tipo_1 nome_campo_1;
    tipo_2 nome_campo_2;
    tipo_3 nome_campo_3;
    ...
    tipo_n nome_campo_n;
};
```



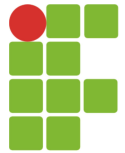
Declaração

```
1  struct pessoa {
2      char nome[30];
3      char cpf[15];
4      double renda_mensal;
5  };
6
7  int main(void) {
8      struct pessoa p; // declaração de uma pessoa 'p'
9      return 0;
10 }
```



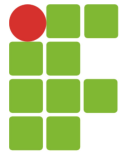
Declaração

- Normalmente colocamos as definições dos tipos dos registros antes do código. Em projetos modularizados em que **se quer exportar** a definição do tipo, incluímos as definições nos arquivos `.h`.



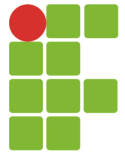
Declaração

```
1  #include <...>
2
3  struct nome_tipo_registro{
4      tipo_1 nome_campo_1;
5      tipo_2 nome_campo_2;
6      tipo_3 nome_campo_3;
7      ...
8      tipo_n nome_campo_n;
9  };
10
11 // ...
12
13 int main(void){
14     // ...
15 }
```

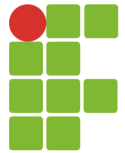


Declaração

```
1  #ifndef NOME_MODULO_H
2  #define NOME_MODULO_H
3
4  struct nome_tipo_registro{
5      tipo_1 nome_campo_1;
6      tipo_2 nome_campo_2;
7      tipo_3 nome_campo_3;
8      ...
9      tipo_n nome_campo_n;
10 };
11
12 // ...
13
14 // protótipos
```

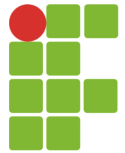


Sumário



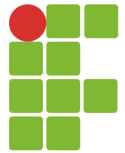
Acesso

- O acesso a cada campo do registro é realizado através do operador . (ponto), seguido do identificado do campo.

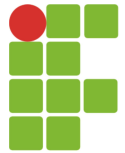


Acesso

```
1  struct pessoa {
2      char nome[30];
3      char cpf[15];
4      double renda_mensal;
5  };
6
7  int main(void) {
8      struct pessoa p1, p2;
9      p1.renda_mensal = 1000.50;
10     p2.renda_mensal = 2 * p2.renda_mensal;
11     return 0;
12 }
```

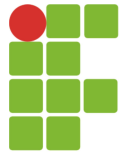


Sumário



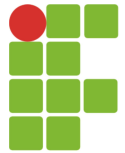
Leitura e escrita

- A leitura e escrita de registros deve ser feita campo a campo.

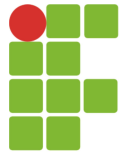


Leitura e escrita

```c
#include <stdio.h>

struct pessoa {
    char nome[30];
    char cpf[15];
    double renda_mensal;
};

int main(void) {
    struct pessoa p1, p2;
    scanf("%s %s %lf", p1.nome, p1.cpf, &p1.renda_mensal);
    scanf("%s %s %lf", p2.nome, p2.cpf, &p2.renda_mensal);
    printf("Imprimindo primeira pessoa\n");
    printf("%s\n%s\n%.2f\n",p1.nome,p1.cpf,p1.renda_mensal);
    printf("Imprimindo segunda pessoa\n");
    printf("%s\n%s\n%.2f\n",p2.nome,p2.cpf,p2.renda_mensal);
    return 0;
}
```

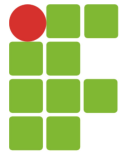


Sumário



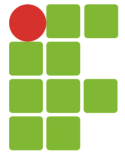
Atribuição

- É possível atribuir um registro a outro registro de mesmo tipo.
- Todas as variáveis são copiadas, inclusive **vetores não dinâmicos**.

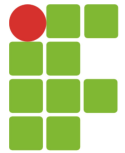


Atribuição

```c
#include <stdio.h>

struct pessoa {
    char nome[30];
    char cpf[15];
    double renda_mensal;
};

int main(void) {
    struct pessoa p1, p2;
    scanf("%s %s %lf", p1.nome, p1.cpf, &p1.renda_mensal);
    printf("Imprimindo primeira pessoa\n");
    printf("%s\n%s\n%.2f\n", p1.nome, p1.cpf, p1.renda_mensal);
    p2 = p1; // atribuição de registros
    printf("Imprimindo segunda pessoa\n");
    printf("%s\n%s\n%.2f\n", p2.nome, p2.cpf, p2.renda_mensal);
    return 0;
}
```

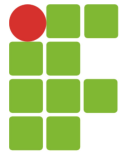


Sumário



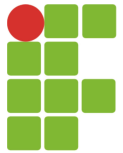
Typedef

- Através do comando `typedef`, é possível criar um novo nome de tipo baseado em outro.
- Em nosso exemplo, para ter que evitar escrever `struct pessoa` ao declarar um registro do tipo pessoa, poderíamos, através do `typedef` chamar `struct pessoa` apenas de `pessoa`



Typedef

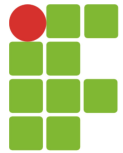
```c
#include <stdio.h>

struct pessoa {
    char nome[30];
    char cpf[15];
    double renda_mensal;
};

typedef struct pessoa pessoa;

int main(void) {
    pessoa p1, p2;
    scanf("%s %s %lf", p1.nome, p1.cpf, &p1.renda_mensal);
    printf("Imprimindo primeira pessoa\n");
    printf("%s\n%s\n%.2f\n", p1.nome, p1.cpf, p1.renda_mensal);
    p2 = p1; // atribuição de registros
    printf("Imprimindo segunda pessoa\n");
    printf("%s\n%s\n%.2f\n", p2.nome, p2.cpf, p2.renda_mensal);
    return 0;
}
```



Typedef

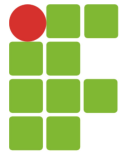
- Ainda é possível unir a declaração do tipo e o `typedef` em uma única estrutura.

```
typedef struct pessoa {
    char nome[30];
    char cpf[15];
    double renda_mensal;
} pessoa;
```

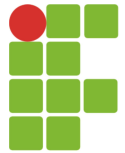


Typedef

```c
#include <stdio.h>

typedef struct pessoa {
    char nome[30];
    char cpf[15];
    double renda_mensal;
} pessoa ;


int main(void) {
    pessoa p1, p2;
    scanf("%s %s %lf", p1.nome, p1.cpf, &p1.renda_mensal);
    printf("Imprimindo primeira pessoa\n");
    printf("%s\n%s\n%.2f\n", p1.nome, p1.cpf, p1.renda_mensal);
    p2 = p1; // atribuição de registros
    printf("Imprimindo segunda pessoa\n");
    printf("%s\n%s\n%.2f\n", p2.nome, p2.cpf, p2.renda_mensal);
    return 0;
}
```

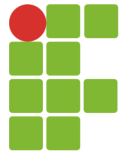


Sumário



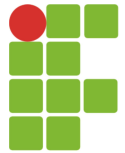
Vetores e registros

- É possível criar um vetor de registros.
  - Declaração: `tipo_registro vet_reg[TAM];` .
  - Acesso: `vet_reg[i].nome_campo` .

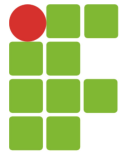


Vetores e registros

```c
#include <stdio.h>

typedef struct pessoa {
    char nome[30];
    char cpf[15];
    double renda_mensal;
} pessoa;

int main(void) {
    pessoa familia[5];
    for (int i = 0; i < 5; i++) {
        printf("Digite o nome do integrante %d: ", i + 1);
        scanf("%s", familia[i].nome);
        printf("Digite o CPF do integrante %d: ", i + 1);
        scanf("%s", familia[i].cpf);
        printf("Digite a renda mensal do integrante %d: ", i + 1);
        scanf("%lf", &familia[i].renda_mensal);
    }

    printf("Imprimindo os integrantes da família\n");
    for (int i = 0; i < 5; i++){
        printf("Nome do integrante %d: %s\n",i+1,familia[i].nome);
        printf("CPF do integrante %d: %s\n",i+1,familia[i].cpf);
        printf("Renda mensal do integrante %d: %.2f\n",i+1,familia[i].renda_mensal);
    }
    return 0;
}
```

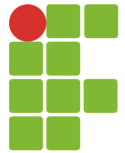


Sumário



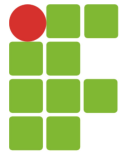
Funções e registros

- Funções podem receber como parâmetros variáveis do tipo registro.
- Também podem retornar um registro.
- Uma das formas de uma função retornar mais de um parâmetro em C é fazendo com que ela retorne um registro.



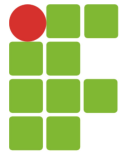
Funções e registros

- Para exemplificar os conceitos, iremos criar as seguintes funções:
  - `pessoa le_pessoa(void);`
  - `void imprime_pessoa(pessoa p);`
  - `void listar_membros(pessoa* v_pessoa, int n);`



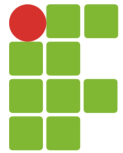
Funções e registros

```
typedef struct pessoa {
    char nome[30];
    char cpf[15];
    double renda_mensal;
} pessoa;
```



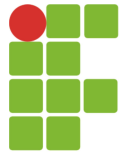
Funções e registros

```
pessoa le_pessoa(void) {
    pessoa p;
    printf("Digite o nome da pessoa: ");
    scanf("%s", p.nome);
    printf("Digite o CPF da pessoa: ");
    scanf("%s", p.cpf);
    printf("Digite a renda mensal: ");
    scanf("%lf", &p.renda_mensal);
    return p;
}
```



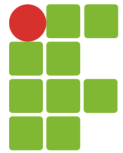
Funções e registros

```
void imprime_pessoa(pessoa p) {
    printf("Nome: %s\n", p.nome);
    printf("CPF: %s\n", p.cpf);
    printf("Renda Mensal: %.2f\n", p.renda_mensal);
}
```



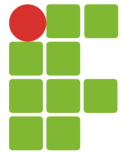
Funções e registros

```
void listar_membros(pessoa *v_pessoa, int n) {
    for (int i = 0; i < n; i++) {
        imprime_pessoa(v_pessoa[i]);
    }
}
```



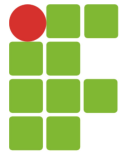
Funções e registros

```c
#include <stdio.h>

typedef struct pessoa {
    char nome[30];
    char cpf[15];
    double renda_mensal;
} pessoa;

pessoa le_pessoa(void) {
    pessoa p;
    printf("Digite o nome da pessoa: ");
    scanf("%s", p.nome);
    printf("Digite o CPF da pessoa: ");
    scanf("%s", p.cpf);
    printf("Digite a renda mensal: ");
    scanf("%lf", &p.renda_mensal);
    return p;
}

void imprime_pessoa(pessoa p) {
    printf("Nome: %s\n", p.nome);
    printf("CPF: %s\n", p.cpf);
    printf("Renda Mensal: %.2f\n", p.renda_mensal);
}
```

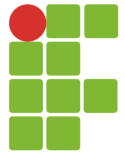


Funções e registros

```
25
26  void listar_membros(pessoa *v_pessoa, int n) {
27      for (int i = 0; i < n; i++) {
28          imprime_pessoa(v_pessoa[i]);
29      }
30  }
31
32  int main(void) {
33      pessoa v_pessoa[5];
34      printf("Digite os membros da família.\n");
35      for (int i = 0; i < 5; i++)
36          v_pessoa[i] = le_pessoa();
37
38      printf("Imprimindo os membros da família.\n");
39      listar_membros(v_pessoa, 5);
40      return 0;
41  }
```

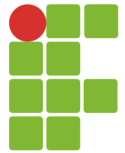


Sumário



Ponteiros e registros

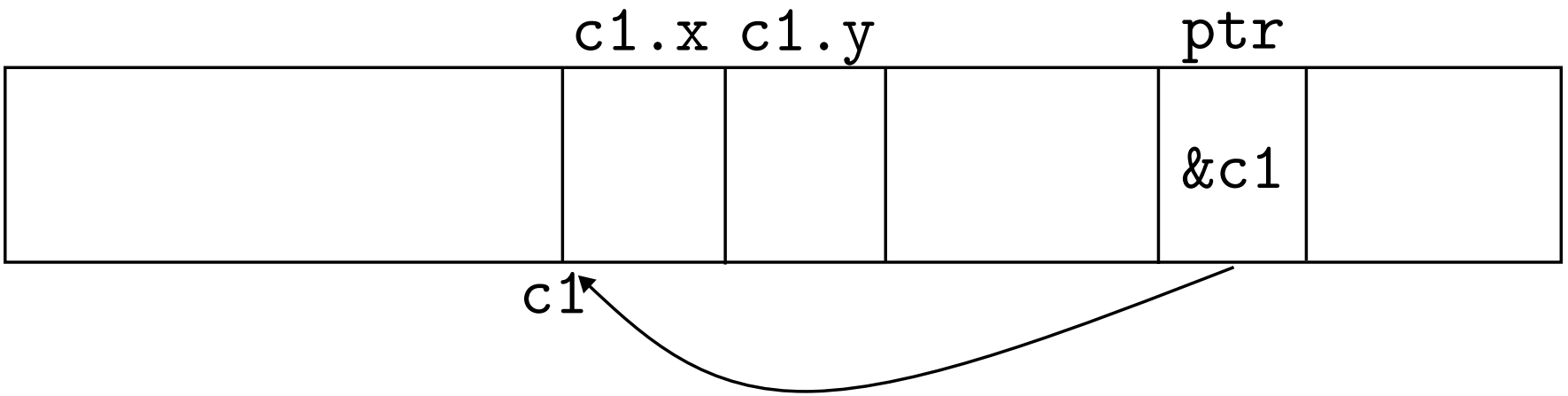
- Um registro também ocupa um endereço da memória.
- Logo, podemos ter ponteiros para registros.

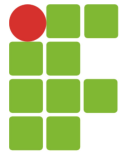


Ponteiros e registros

```c
#include <stdio.h>

typedef struct coordenada {
    double x;
    double y;
} coordenada;

int main(void) {
    coordenada c1;
    coordenada *ptr;
    ptr = &c1;
    return 0;
}
```

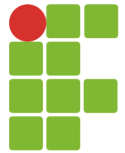
Ponteiros e registros





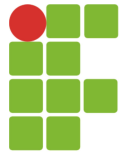
Ponteiros e registros

- Dado que `ptr` aponta para `c1`, como podemos mudar os valores de x e y do registro através do ponteiro?
- Primeiro aplicamos o operador de desreferência seguido do operador ponto.
- `(*ptr).x`
- Os parênteses são necessários, pois o ponto tem precedência sobre o asterisco.



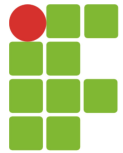
Ponteiros e registros

```c
#include <stdio.h>

typedef struct coordenada {
    double x;
    double y;
} coordenada;

int main(void) {
    coordenada c1;
    coordenada *ptr;
    ptr = &c1;
    (*ptr).x = 1.5;
    (*ptr).y = -2.3;
    printf("(%.2f %.2f)\n", c1.x, c1.y);
    return 0;
}
```



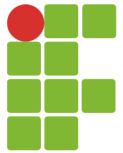
Operador seta

- Aplicar o operador de desreferência seguido do operador de acesso a um campo de um registro é uma operação muito comum em C.
- Para simplificar, o Cdisponibiliza um operador extra: seta.
- `ptr->x` $\equiv$ `(*ptr).x`



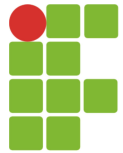
Operador seta

```c
#include <stdio.h>

typedef struct coordenada {
    double x;
    double y;
} coordenada;

int main(void) {
    coordenada c1;
    coordenada *ptr;
    ptr = &c1;
    ptr->x = 1.5;
    ptr->y = -2.3;
    printf("(%.2f %.2f)\n", c1.x, c1.y);
    return 0;
}
```



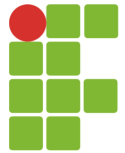
Desempenho

- Uma questão importante na linguagem C é a passagem, ou retorno,de registros para, ou de, funções.
- Como sabemos, o C utiliza passagem por valor, fazendo com que os registros sejam efetivamente copiados.
- Dependendo do tamanho do tipo registro, isso pode representar uma penalidade de tempo muito alta.
- Deve ser considerada a passagem por "referência" através de ponteiros para evitar a cópia do registro nesses casos.



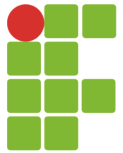
Desempenho

```
pessoa le_pessoa(void) {
    pessoa p;
    printf("Digite o nome da pessoa: ");
    scanf("%s", p.nome);
    printf("Digite o CPF da pessoa: ");
    scanf("%s", p.cpf);
    printf("Digite a renda mensal: ");
    scanf("%lf", &p.renda_mensal);
    return p;
}
```



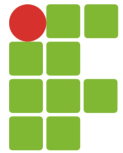
Desempenho

- Podemos substituir a função `le_pessoa` que retorna um registro com as informações da pessoa preenchida por outra versão, que modifica, "por referência", uma pessoa.
- O tempo gasto é muito menor, visto que apenas o endereço de uma variável do tipo pessoa é copiado.



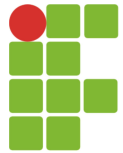
Desempenho

```
void le_pessoa(pessoa* p){
    printf("Digite o nome da pessoa: ");
    scanf("%s", p->nome);
    printf("Digite o CPF da pessoa: ");
    scanf("%s", p->cpf);
    printf("Digite a renda mensal: ");
    scanf("%lf", &p->renda_mensal);
}
```



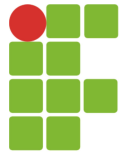
Desempenho

- Caso a função não altere o valor de um registro, podemos passar um ponteiro para um registro constante, ao utilizar o modificador `const` nos parâmetros da função.

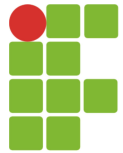


Desempenho

```
void imprime_pessoa(const pessoa* p) {
    printf("Nome: %s\n", p->nome);
    printf("CPF: %s\n", p->cpf);
    printf("Renda Mensal: %.2f\n", p->renda_mensal);
}
```

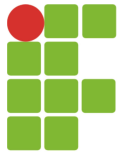


Sumário



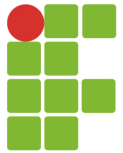
Alocação dinâmica e registros

- Como podemos ter ponteiros para registros, é possível criar vetores ou matrizes dinâmicas de registros.
- `pessoa* v_pessoa = malloc(sizeof(pessoa)*n);`



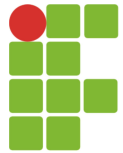
Desempenho

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

typedef struct pessoa {
    char nome[30];
    char cpf[15];
    double renda_mensal;
} pessoa;
```



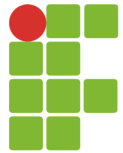
Desempenho

```
void le_pessoa(pessoa *p) {
    printf("Digite o nome da pessoa: ");
    scanf("%s", p->nome);
    printf("Digite o CPF da pessoa: ");
    scanf("%s", p->cpf);
    printf("Digite a renda mensal: ");
    scanf("%lf", &p->renda_mensal);
}
```



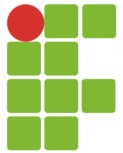
Desempenho

```c
void imprime_pessoa(const pessoa *p) {
    printf("Nome: %s\n", p->nome);
    printf("CPF: %s\n", p->cpf);
    printf("Renda Mensal: %.2f\n", p->renda_mensal);
}
```



Desempenho

```
void listar_membros(pessoa *v_pessoa, int n) {
    for (int i = 0; i < n; i++) {
        imprime_pessoa(&v_pessoa[i]);
    }
}
```



Desempenho

```c
int main(void) {
    int n;
    scanf("Digite o número de pessoas na família: ");
    scanf("%d", &n);
    pessoa *v_pessoa = malloc(sizeof(pessoa) * n);
    printf("Digite os membros da família.\n");
    for (int i = 0; i < n; i++)
        le_pessoa(&v_pessoa[i]);

    printf("Imprimindo os membros da família.\n");
    listar_membros(v_pessoa, n);
    free(v_pessoa);
    return 0;
}
```