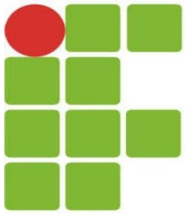
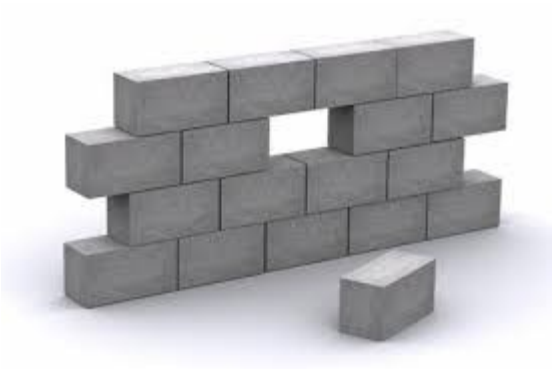


Instituto Federal
Campus Goiânia

Bacharelado em Sistemas de Informação

Estrutura de Dados I



Prof. Dory Gonzaga Rodrigues

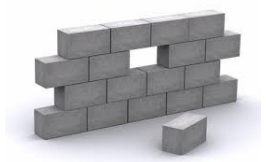
Struct

Conceito

Em C uma estrutura (Struct vem de structure) é uma coleção de variáveis referenciadas por um único nome, também conhecida como um registro, e fornece uma maneira conveniente de agrupar informações relacionadas.

Definição / Sintaxe

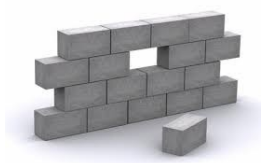
```
struct  <nome_da_coleção> {
        <tipo_de_dados>     nome_campo;
        <tipo_de_dados>     nome_campo;
        . . .
        <tipo_de_dados>     nome_campo;
} ;
```



Struct

Exemplo

```
struct funcionario {
        int       matricula;
        char      nome[50];
        float     salario;
} ;
```



Struct

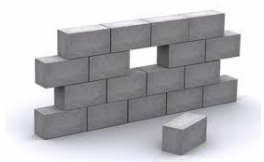
Declaração de variáveis

1) Diretamente na definição

```
struct funcionario {
        int       matricula;
        char      nome[50];
        float     salario;
} <nome_da_variável>;
```

2) Em qualquer lugar do Programa

```
struct funcionario <nome_da_variável>;
```



Struct

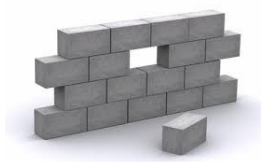
Exemplo

1) Diretamente na definição

```
struct funcionario {
        int         matricula;
        char        nome[50];
        float       salario;
} v_func;
```

2) Em qualquer lugar do Programa

```
struct funcionario v_func;
```



Struct

Manipulação de dados

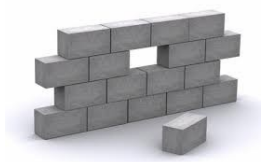
Incluir ou acessar uma informação de um campo de um Struct (registro), escrevemos o nome da variável e o nome do campo separados por um ponto:

Atribuição direta

```
v_func.matricula    = 0001;
v_func.nome         = 'Caroline Maia';
v_func.salario      = 2.530,00;
```

Atribuição indireta

```
printf("Digite o nome: ");
Scanf("%s", v_func.nome);
```



Struct

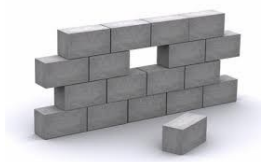
NOVO tipo de dados

Registros podem ser tratados como um novo tipo-de-dados. Depois da seguinte definição, por exemplo, poderemos passar a dizer "funcionario" no lugar de "struct funcionario":

```
typedef struct {
        int     matricula;
        char    nome[50];
        float   salario;
} funcionario;
```

Declaração de variáveis:

```
funcionario  x, y;
```

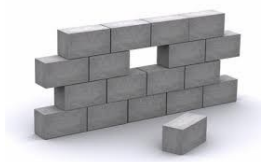


Struct

Registros e Funções

Registros podem ser usados tanto como parâmetros em funções bem como em retorno de funções.

Neste caso o comportamento de registros e similar ao de tipos básicos.



Struct

Funções e Registros

Exemplo.

- Vamos criar as seguintes funções:
  - `struct Aluno leAluno();`

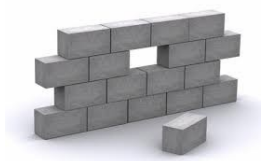
    Esta função faz a leitura dos dados de um registro **Aluno** e devolve o registro lido.

  - `void imprimeAluno(struct Aluno a);`

    Esta função recebe como parâmetro um registro **Aluno** e imprime os dados do registro.

  - `void listarTurma(struct Aluno turma[], int n);`

    Esta função recebe como parâmetros um vetor do tipo **Aluno** representando uma turma, e também um inteiro **n** indicando o tamanho do vetor. A função imprime os dados de todos os alunos.



Struct

Funções e Registros

Implementação das funções:

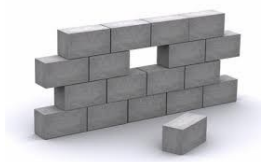
```
struct Aluno leAluno(){
  struct Aluno aux;

  printf("Digite o Nome: ");
  fgets(aux.nome, 80, stdin);
  aux.nome[strlen(aux.nome) -1] ='\0'; //remove '\n'
  printf("Digite a Nota: ");
  scanf("%f",&aux.nota);getchar();

  return aux;
}

void imprimeAluno(struct Aluno a){
  printf("Dados de um aluno --- ");
  printf("Nome: %s. Nota: %.2f\n", a.nome, a.nota);
}

void listarTurma(struct Aluno turma[], int n){
  printf("Imprimindo a turma\n");
  int i;
  for(i=0; i<n; i++)
    imprimeAluno(turma[i]);
}
```

Struct

Funções e Registros

Com as funções implementadas podemos criar o seguinte exemplo de programa.

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

#define MAX 4

struct Aluno{
  char nome[80];
  float nota;
};

struct Aluno leAluno();
void imprimeAluno(struct Aluno a);
void listarTurma(struct Aluno turma[], int n);

int main(){
  int i;
  struct Aluno turma[MAX];
  for(i=0; i<MAX; i++)
    turma[i] = leAluno();

  listarTurma(turma, MAX);
}
```



Struct

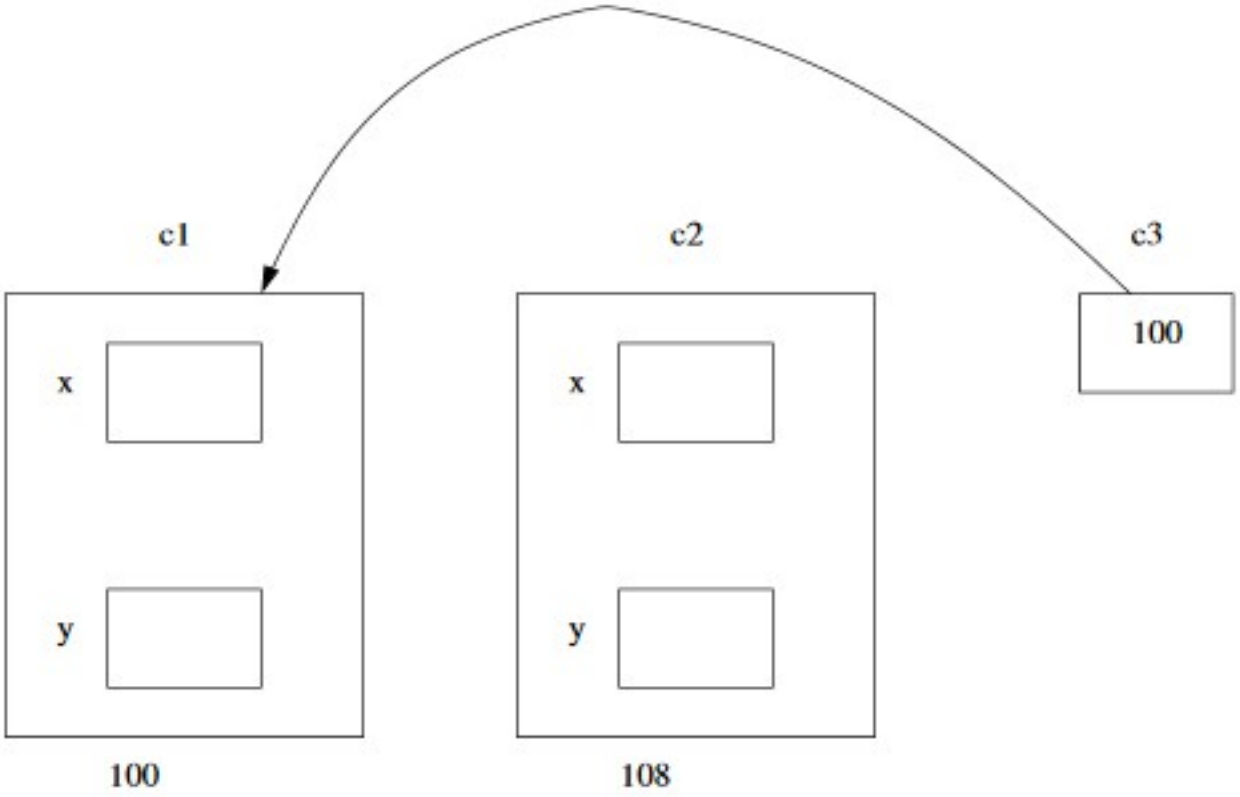
Ponteiros e Registros

- Ao criarmos uma variável de um tipo **struct**, esta é armazenada na memória como qualquer outra variável, e portanto possui um endereço.
- É possível então criar um ponteiro para uma variável de um tipo **struct**!

```
#include <stdio.h>

struct Coordenada{
  double x;
  double y;
};

int main(){
  struct Coordenada c1, c2, *c3;
  c3 = &c1;
   ......
```

Struct

Ponteiros e Registros



Struct

O que será impresso pelo programa abaixo??

```
#include <stdio.h>
struct Coordenada{
  double x;
  double y;
};

int main(){
  struct Coordenada c1, c2, *c3;

  c3 = &c1;
  c1.x = -1;
  c1.y = -1.5;

  c2.x = 2.5;
  c2.y = -5;

  *c3 = c2;

  printf("Coordenadas de c1: (%lf,%lf)\n",c1.x, c1.y);
}
```

Struct

Ponteiros e Registros

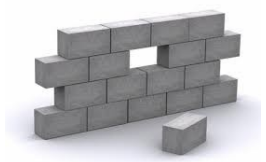
- Para acessarmos os campos de uma variável **struct** via um ponteiro, podemos utilizar o operador * juntamente com o operador . como de costume:

```
Coordenada c1, *c3;
c3 = &c1;
(*c3).x = 1.5;
(*c3).y = 1.5;
```

- Em C também podemos usar o operador ->, que também é usado para acessar campos de uma estrutura via um ponteiro.

```
Coordenada c1, *c3;
c3 = &c1;
c3->x = 1.5;
c3->y = 1.5;
```

- Resumindo: Para acessar campos de estruturas via ponteiros use um dos dois:
  - `ponteiroEstrutura->campo`
  - `(*ponteiroEstrutura).campo`

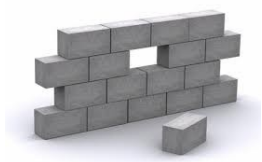


Struct

Ponteiros e Registros

O que será impresso pelo programa abaixo??

```c
#include <stdio.h>

struct Coordenada{
  double x;
  double y;
};

int main(){
  struct Coordenada c1, c2, *c3, *c4;
  c3 = &c1;
  c4 = &c2;

  c1.x = -1;
  c1.y = -1.5;

  c2.x = 2.5;
  c2.y = -5;

  (*c3).x = 1.5;
  (*c3).y = 1.5;

  c4->x = -1;
  c4->y = -1;

  printf("Coordenadas de c1: (%lf,%lf)\n",c1.x, c1.y);
  printf("Coordenadas de c2: (%lf,%lf)\n",c2.x, c2.y);
}
```



Struct

Ponteiros e Registros

Utilizando as funções criadas anteriormente podemos executar o exemplo:

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

struct Aluno{
  char nome[80];
  float nota;
};

struct Aluno leAluno();
void imprimeAluno(struct Aluno a);
void listarTurma(struct Aluno turma[], int n);

int main(){
  struct Aluno *vetAlu;
  int n, i;

  printf("Numero de alunos: ");
  scanf("%d", &n); getchar();

  vetAlu = malloc(n*sizeof(struct Aluno));  //Alocação dinâmica do vetor de registros

  for(i=0; i<n; i++){
    vetAlu[i] = leAluno();
  }
  listarTurma(vetAlu, n);

  free(vetAlu);  //Liberação de memória alocada
}
```