INSTITUTO FEDERAL
Sul-rio-grandense | Câmpus Charqueadas

EDUCAÇÃO PÚBLICA 100% GRATUITA

Estruturas definidas pelo programador: struct, enum, union

Estrutura de Dados

Prof. André del Mestre

Agrupando variáveis relacionadas

Structs

Fundamentos

- **Categorias de Variáveis**
  - **Simples** - int float char
  - Arrays **homogêneos** de variáveis simples

```
int matricula;
char status, nome[50];
float n1, n2, rec;
```

Estruturas de Seleção

Structs

Fundamentos

- **Categorias de Variáveis**
  - **Simples** - int float char
  - Arrays **homogêneos** de variáveis simples

```
int matricula;
char status, nome[50];
float n1, n2, rec;

// UM aluno
matricula = 123;
status = 'A';
strcpy(nome, "Fulano da Silva");
n1 = 9.5;
n2 = 9.0;
rec = 10.0;
```

Repare que a relação entre as variáveis

Structs

Fundamentos

- **Categorias de Variáveis**
  - **Simples** - int float char
  - Arrays **homogêneos** de variáveis simples

```
int matricula[30];
char status[30], nome[30][50];
float n1[30], n2[30], rec[30];

// UMA turma de 30 alunos...
for(...){
  matricula[] = ...;
  status[] = ...;
  strcpy(nome[], "...");
  n1[] = ...;
  n2[] = ...;
  rec[] = ...;
}
```

Este código parece necescitar de melhorias....

Structs

Fundamentos

- **Agrupamento de variáveis**
  - Um aluno tem todos estes dados, porque não agrupa-los?
  - Variável composta heterogênea

```
// agrupa vars de UM aluno
struct aluno{
   int matricula;
   char status, nome[50];
   float n1, n2, rec;
}
```

Definindo uma Struct

Structs

Fundamentos

- **Agrupamento de variáveis**
  - Um aluno tem todos estes dados, porque não agrupa-los?
  - Variável composta heterogênea

```
// agrupa vars de UM aluno
struct aluno{
   int matricula;
   char status, nome[50];
   float n1, n2, rec;
}

int main (){
    struct aluno X;
    int i;
    ...
}
```

Declarando uma Struct

Structs

Fundamentos

- **Agrupamento de variáveis**
  - Um aluno tem todos estes dados, porque não agrupa-los?
  - Variável composta heterogênea
- **Comando `typedef`**
  - permite definir os seus próprios tipos com base em outros tipos de dados existentes

```
// agrupa vars de UM aluno
struct aluno{
   int matricula;
   char status, nome[50];
   float n1, n2, rec;
}typedef Aluno;
```

Definindo meu próprio Tipo de Dado

Structs

Fundamentos

- **Agrupamento de variáveis**
  - Um aluno tem todos estes dados, porque não agrupa-los?
  - Variável composta heterogênea
- **Comando `typedef`**
  - permite definir os seus próprios tipos com base em outros tipos de dados existentes
  - Eu prefiro usar sempre para o tipo de var ser apenas uma palavra chave sempre
  - Inicio sempre com letra maiúscula

Declarando uma variável do meu próprio Tipo de Dado

```
// agrupa vars de UM aluno
struct aluno{
   int matricula;
   char status, nome[50];
   float n1, n2, rec;
}typedef Aluno;

int main (){
   Aluno X;
   int i;
   ...
}
```

Structs

Fundamentos

- **Agrupamento de variáveis**
  - Um aluno tem todos estes dados, porque não agrupa-los?
  - Variável composta heterogênea
- **Comando `typedef`**
  - permite definir os seus próprios tipos com base em outros tipos de dados existentes
  - Eu prefiro usar sempre para o tipo de var ser apenas uma palavra chave sempre
  - Inicio sempre com letra maiúscula

Declarando um ARRAY do meu próprio Tipo de Dado

```
// agrupa vars de UM aluno
struct aluno{
   int matricula;
   char status, nome[50];
   float n1, n2, rec;
}typedef Aluno;

int main (){
    Aluno turma[30];
    int i;
    ...
}
```

Utilizando Structs

Atribuição

```c
struct aluno{
  int matricula;
  char status, nome[50];
  float n1, n2, rec;
}typedef Aluno;
```

```c
int main (){
  Aluno aluno;

  aluno.matricula = 123;
  strcpy(aluno.nome, "Fulano da Silva");
  aluno.status = 'A';
  aluno.n1 = 9.5;
  aluno.n2 = 9.0;
  aluno.rec = 10.0

}
```

Atencao ao ponto na sintaxe:
var_struct.nome_parametro

Utilizando Structs

Atribuição

```c
struct aluno{
  int matricula;
  char status, nome[50];
  float n1, n2, rec;
}typedef Aluno;
```

```c
int main (){
  Aluno aluno;

  aluno matricula = 123;
  strcpy(aluno.nome, "Fulano da Silva");
  aluno.status = 'A';
  aluno.n1 = 9.5;
  aluno.n2 = 9.0;
  aluno.rec = 10.0

}
```

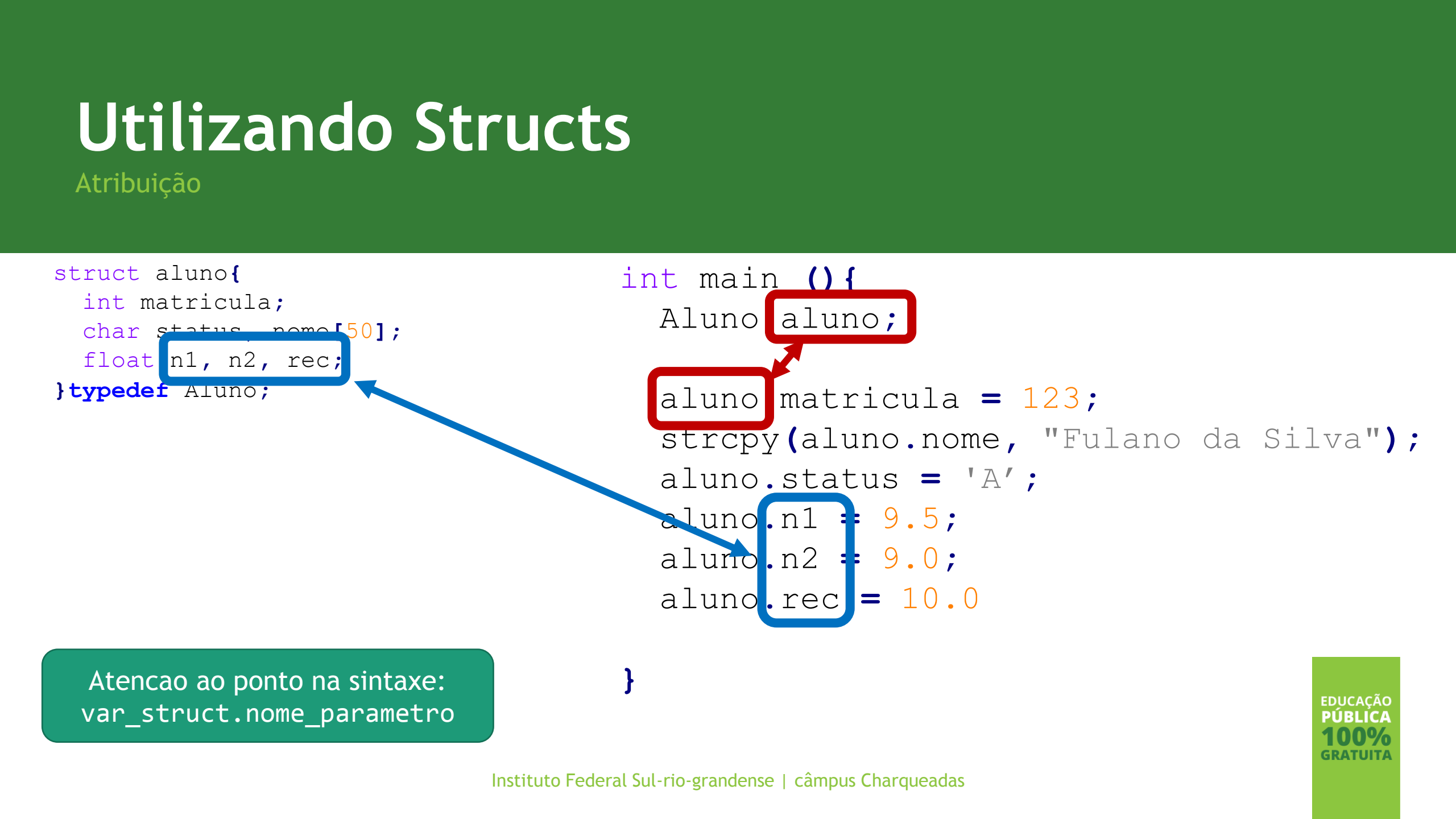
Atencao ao ponto na sintaxe:
var_struct.nome_parametro

Utilizando Structs

Atribuição

```
struct aluno{
  int matricula;
  char status, nome[50];
  float n1, n2, rec;
}typedef Aluno;
```

```
int main (){
  Aluno aluno;

  aluno.matricula = 123;
  strcpy(aluno.nome, "Fulano da Silva");
  aluno.status = 'A';
  aluno.n1 = 9.5;
  aluno.n2 = 9.0;
  aluno.rec = 10.0

}
```



Atencao ao ponto na sintaxe:
var_struct.nome_parametro

Utilizando Structs

Imprimindo dados

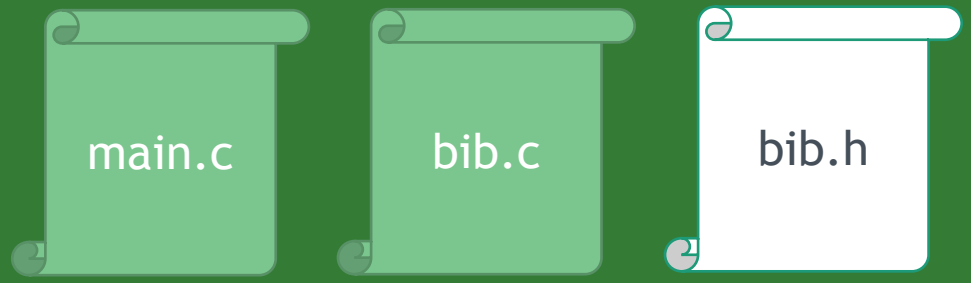
```c
struct aluno{
  int matricula;
  char status, nome[50];
  float n1, n2, rec;
}typedef Aluno;
int main (){
  Aluno aluno;
  aluno.matricula = 123;
  strcpy(aluno.nome, "Fulano da Silva");
  aluno.status = 'A';
  aluno.n1 = 9.5;
  aluno.n2 = 9.0;
  aluno.rec = 10.0;
```

```c
  printf("Matricula: %i\n", aluno.matricula);
  printf("Nome: %s\n", aluno.nome);
  printf("Status: %c\n", aluno.status);
  printf("Notas: %f %f\n", aluno.n1, aluno.n2);
  printf("Recuperacao: %f\n", aluno.rec);
}
```

Struct e funções

Structs e funções

Ponteiros facilitam!



- **1 - Definindo o cabeçalho**
  - Também defina seus tipos de dados em bib.h
  - SIM, ponteiro de struct!!

```
struct aluno{
   int matricula;
   char status, nome[50];
   float n1, n2, rec;
}typedef Aluno;

void init_aluno(Aluno* al, int matricula);
void imprime_aluno(Aluno* al);
void le_aluno(Aluno* al, int matricula);
```

Structs e funções

Ponteiros facilitam!

- **2 - Descreva a logica das funções**
  - Prefiro seta (->) que ponto (.)
  - Não precisava usar ponteiro nesta função!

```
void init_aluno(Aluno* al, int matricula){
   al->matricula = matricula;
   strcpy(al->nome, "Fulano da Silva");
   al->status = 'R';
   al->n1 = 0.0;
   al->n2 = 0.0;
   al->rec = 0.0;
}
```

Structs e funções

Ponteiros facilitam!

- **2 - Descreva a logica das funções**
  - Prefiro seta (->) que ponto (.)
  - Não precisava usar ponteiro nesta função!

```
void imprime_aluno(Aluno* al){
  if(al == NULL){
    printf("ERRO - aluno nao encontrado\n");
  }else{
    printf("Matricula: %d\n", al->matricula);
    printf("Nome: %s\n", al->nome);
    printf("Status: %c\n", al->status);
    printf("Notas: %f %f\n", al->n1, al->n2);
    printf("Recuperacao: %f\n", al->rec);
  }
}
```

Structs e funções

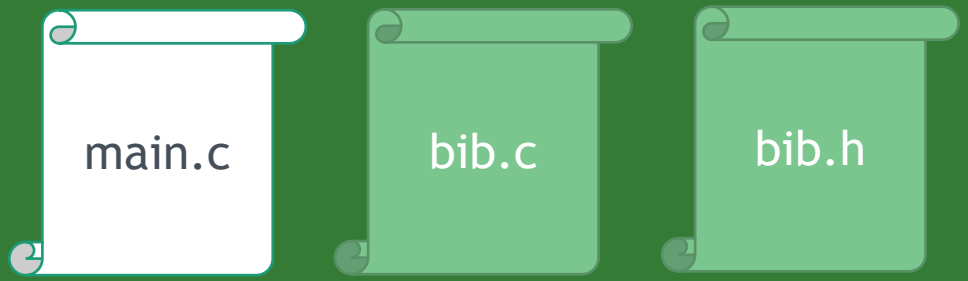
Ponteiros facilitam!

- **2 - Descreva a logica das funções**
  - Prefiro seta (**->**) que ponto (**.**)
  - Na função de leitura, ponteiro eh obrigatório
  - Você usa uma única sintaxe

```
void le_aluno(Aluno* al, int matricula){

  printf("\n\nDigite as notas do aluno %i abaixo\n", matricula);
  al->matricula = matricula;

  printf("Nome do aluno\n");
  gets(al->nome);

  printf("Notas do 1 e 2 semestres\n");
  scanf("%f%f", &al->n1, &al->n2);

  printf("Nota da reavaliacao\n");
  scanf("%f", &al->rec);

  if((al->n1 >= 6 && al->n2 >= 6) || al->rec >= 6){
    al->status = 'A';
  }else{
    al->status = 'R';
  }
}
```

Structs e funções

Ponteiros facilitam!



- 3 - Usando as funções
  - O aluno com matricula 123 será lido do usuário
  - Em seguida os dados digitados serão apresentados na tela

```
int main (){
   Aluno aluno;
   Aluno * aluno_ptr=NULL;

   aluno_ptr = &aluno;
   le_aluno(aluno_ptr, 123);
   imprime_aluno(aluno_ptr);

}
```

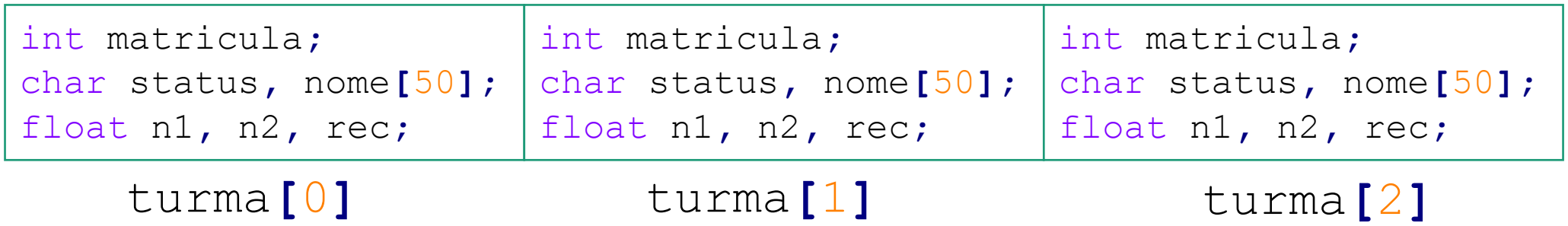
Vetor de Struct

Vetor de Structs

Ponteiros facilitam!

- **Retomando**
  - **`struct`**: define um "conjunto" de variáveis que podem ser de tipos diferentes;
  - **`array`**: é uma "lista" de elementos de mesmo tipo.

```c
struct aluno{
  int matricula;
  char status, nome[50];
  float n1, n2, rec;
}typedef Aluno;

int main (){
  Aluno turma[30];
...
}
```



Vetor de Structs

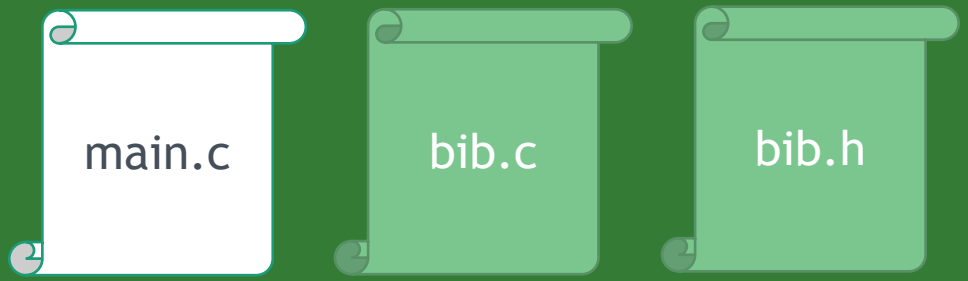
Ponteiros facilitam!

- **Modo de usar**
  - Trabalhar vetores de structs com funções eh muito mais fácil!

```c
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
    float media;
};


int main(){
    struct aluno a[10];
    int i;
    for(i=0;i<10;i++){
        scanf("%d",&a[i].num_aluno);
        scanf("%f",&a[i].nota1);
        scanf("%f",&a[i].nota2);
        scanf("%f",&a[i].nota3);
        a[i].media = (a[i].nota1 + a[i].nota2 + a[i].nota3)/3.0
    }
}
```

Vetor de Structs

Ponteiros facilitam!



- **3 - Usando as funções**
  - Trabalhar vetores de structs com funções eh muito mais fácil!

```
int main (){
  Aluno turma[10];
  Aluno * aluno_ptr = turma;

  for(i=0; i<8; i++)
    init_aluno(aluno_ptr+i, i);

  for(i=8; i<10; i++)
    le_aluno(aluno_ptr+i, i);

  printf("Imprime Turma --------\n");
  for(i=0; i<10; i++)
    imprime_aluno(aluno_ptr+i);
}
```

Struct, Funções e Vetores

Mais funções com Structs

Ponteiros facilitam!

- **1 - Definindo o cabeçalho**
  - Também defina seus tipos de dados em bib.h
  - SIM, ponteiro de struct!!

```
struct aluno{
   int matricula;
   char status, nome[50];
   float n1, n2, rec;
}typedef Aluno;

float media_aluno(Aluno * al);
float media_turma(Aluno * turma, int tamanho);

int conta_status(Aluno * turma, int tamanho, char status);

Aluno * consulta_aluno_por_matricula(Aluno * al, int
tamanho, int matricula);
```

MUITO OBRIGADO

Prof. André del Mestre

www.ifsul.edu.br
andremartins@ifsul.edu.br