Registros

Profa.: Márcia Sampaio Lima

EST - UEA

Registros

- Estrutura de Dados
  - Variáveis compostas homogêneas ?

Registros

- Estrutura de Dados
  - Variáveis compostas homogêneas ?
    - Vetores
    - Matrizes

Registros

- Estrutura de Dados
  - Variáveis compostas heterogêneas
    - Em alguns problemas há necessidade de definirmos conjuntos onde os elementos não sejam do mesmo tipo.
    - Exemplo:
      - Conjunto de informações que caracterizam um aluno: Nome(literal), CPF(literal), Idade(inteiro), coeficiente de rendimento(real), etc..

Registros

- Possível solução: declarar cinco variáveis.

```
String Nome, CPF;
Int idade;
Float coeficiente;
```

- E, se tivéssemos que gerenciar o dado de 100 alunos?

Registros

- Possível solução 2: utilização de cinco vetores.

```
String Nomes[100], CPF[100];
Int idade[100];
Float coeficientes [100];
```

- Porém, manipular de forma adequada os vetores, mantendo seus dados consistentes, se torna trabalhoso.

Registros

- Solução: variáveis compostas heterogêneas.
  - Registro ou estruturas
  - Uma estrutura (**struct**) é uma coleção de uma ou mais variáveis, possivelmente de tipos diferentes, agrupadas sob um único nome.
  - Estruturas são um recurso importante para organizar os dados utilizados por programas, pois trata um grupo de valores como uma única variável.

Registros

- Variáveis compostas heterogêneas.
  - Registro ou estruturas
  - Podem agrupar variáveis de tipos diferentes.
  - Diferentemente de vetores e matrizes que agrupam dados do mesmo tipo.

Registros

- Exemplo:

```
struct estruturaAluno
{
  Char Nome[50], CPF[11];
  int idade;
  float coeficiente;
};
```

Registros

- Registros (agregados heterogêneos): são estruturas de dados que permitem o agrupamento de dados com diferentes tipos de dados.
- Um registro é formado por **campos**.
  - Cada campo é a declaração de uma variável de algum tipo de dado.

```
registro Aluno
    nome: string
    idade: inteiro
    cre: real
fim_registro
```

Registros

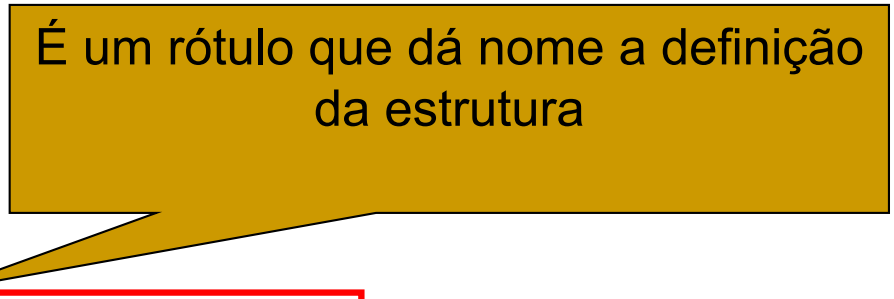
- Exemplo:

Informa ao compilador que um modelo de dados está sendo definido

```
struct estruturaAluno
{
  string Nome, CPF;
  int idade;
  float coeficiente;
};
```

Registros

- Exemplo:




```
struct estruturaAluno
{
  string Nome, CPF;
  int idade;
  float coeficiente;
};
```

Registros

- Exemplo:

```
struct estruturaAluno
{
  string Nome, CPF;
  int idade;
  float coeficiente;
};
```

É um comando, logo deve terminar com ;

Registros

- A definição da estrutura (struct) não reserva espaço de memória.
- **Nenhuma variável é declarada**, apenas a forma dos dados foi definida.
- Dessa definição pode-se criar um novo tipo de dado, que pode ser usado para declarar variáveis.

Registros

- Declaração de uma variável do tipo `estruturaAluno`:

```
struct estruturaAluno
{
      char Nome[50], CPF[17];
      int idade;
      float coeficiente;
};
.....
struct estruturaAluno aluno;
```

ou

```
struct estruturaAluno
{
    Char Nome[50], CPF[17];
    int idade;
    float coeficiente;
}aluno;
```

Registros

- Declaração de uma variável do tipo `estruturaAluno`:

```
struct estruturaAluno
{
  Char Nome[50], CPF[17];
  int idade;
  float coeficiente;
};
.....
struct tipoAluno aluno;
```

Dois comandos:
1. Define a estrutura como um novo tipo.
2. Declara a variável do novo tipo definido.

Registros

- Declaração de uma variável do tipo `estruturaAluno`:

```
struct estruturaAluno
{
      string Nome, CPF;
      int idade;
      float coeficiente;
};

struct estruturaAluno aluno;
```

Dois comandos:
1. Define a estrutura como um novo tipo.
2. Declara a variável do novo tipo definido.

```
....
struct strAluno{
   char nome[50], end[60];
   int matricula;
   double notaAED;
   };


   int main() {
      strAluno alu1;
      return (0);
   }
```

Registros

- Declaração de uma variável do tipo `tipoAluno`:

```
struct estruturaAluno
{
        char Nome[50], CPF[17];
        int idade;
        float coeficiente;
}aluno;
```

Um comandos:
1. Define a estrutura e declara a variável do novo tipo.

```
....
struct strAluno{
  char nome[50], end[60];
  int matricula;
  double notaAED;
  }alu1;


  int main() {
     ....
     return (0);
  }
```

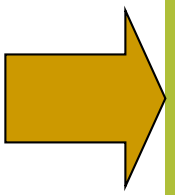
Registros

- Uma estrutura é composta por campos.
- Os campos de uma estrutura podem ser de qualquer tipo, inclusive outra estrutura previamente definida.
- Os campos não podem ser do próprio tipo da estrutura que está sendo definida.

Registros

- Exemplo:

```
struct estruturaAluno
{
char Nome[50], CPF[17];
int idade;
float coeficiente;
};
```



```
struct estrMatDisciplina
{
      estruturaAluno aluno;
      char disicplina[50];
};
```

```
struct strEndAlun{
   char rua[20];
   int numero;
   };


   struct strAluno{
     char nome[50];
     struct strEndAlun end;
     int matricula;
     double notaAED;
   }alu1;
```

Registros

- A definição de um formato de uma estrutura pode ser feita dentro ou fora da `main()`.

- Por questões de visibilidade da estrutura, normalmente declara-se fora da `main()`.

Registros

- Sintaxe da definição de um registro:

```
struct <rótulo>
{
    <tipo> campo_1;
    <tipo> campo_2;
        .....
    <tipo> campo_3;

}<variáveis>;
```

Registros

- Definição de novos TIPOS:
  - Usando a palavra reservada `typedef`.

```
typedef struct nome-da-estrutura
{
        <tipo> campo_1;
        <tipo> campo_2;
        .....
        <tipo> campo_3;

} nome-do-tipo;
```

Registros

- Exemplo:

```
typedef struct estruturaAluno
{
        char Nome[50], CPF[17];
        int idade;
        float coeficiente;

}tipoAluno;
```

Registros

- O uso mais comum de `typedef` é com estruturas de dados, pois evita que a palavra-chave `struct` tenha de ser colocada toda vez que uma estrutura é declarada.

```
struct estruturaAluno aluno;
          X
tipoAluno aluno;
```

```
typedef struct {
   char rua[20];
   int numero;
}tipoEnd;
typedef struct {
   char nome[50];
   tipoEnd end;
   int matricula;
   double notaAED;
}tipoAluno;
int main() {
      tipoAluno alu1;
      .....
}
```

Registros

- Acessando campos de um registro:
  - Podemos acessar individualmente os campos de uma determinada estrutura como se fossem variáveis comuns.
  - A sintaxe para acessar e manipular campos de estruturas é a seguinte:

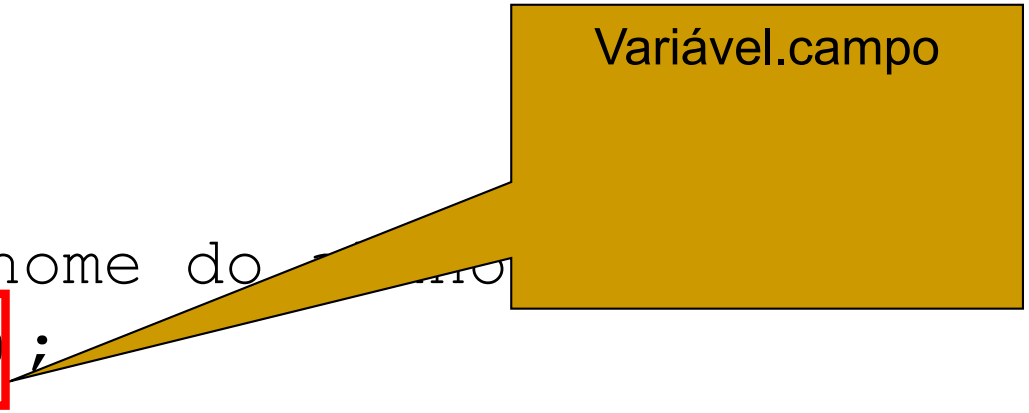
```
<nome_da_variável>.<campo>
```

Registros

- A leitura dos campos de uma estrutura a partir do teclado deve ser feita campo a campo, como se fosse variáveis independentes.

Registros

- Exemplo:




```
cout<<"Digite o nome do aluno: ";
gets (aluno.nome);

cout << "Digite o cpf do aluno: ";
gets (aluno.cpf);

cout << "Digite a idade do aluno: ";
cin >> aluno.idade;

cout << "Digite o coeficiente do aluno: ";
cin >> aluno.coeficiente;
```

Registros

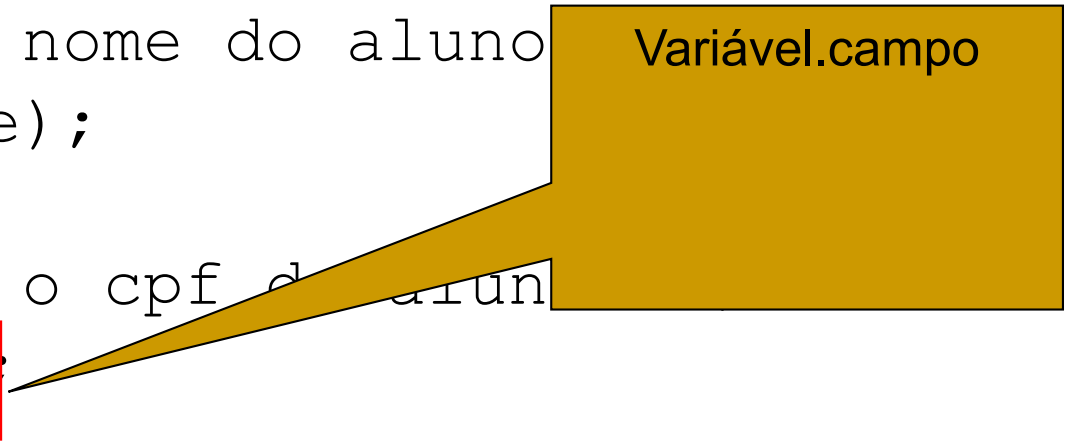
- Exemplo:

```
cout<<"Digite o nome do aluno
gets (aluno.nome);

cout << "Digite o cpf d  alun
gets(aluno.cpf);

cout << "Digite a idade do aluno: ";
cin >> aluno.idade;

cout << "Digite o coeficiente do aluno: ";
cin >> aluno.coeficiente;
```



Registros

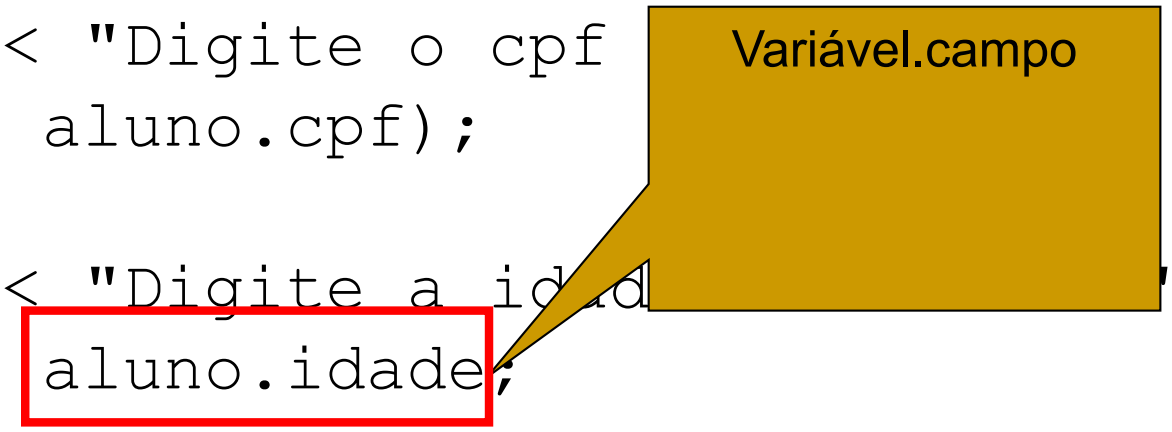
- Exemplo:

```
cout<<"Digite o nome do aluno: ";
gets (aluno.nome);

cout << "Digite o cpf
gets ( aluno.cpf);

cout << "Digite a idade
cin >> aluno.idade;

cout << "Digite o coeficiente do aluno: ";
cin >> aluno.coeficiente;
```



Registros

- Exemplo:

```
cout<<"Digite o nome do aluno: ";
gets (aluno.nome);

cout << "Digite o cpf do aluno: ";
gets (aluno.cpf);

cout << "Digite a idade do aluno: ";
cin >> aluno.idade;

cout << "Digite o coeficiente do aluno: ";
cin >> aluno.coeficiente;
```

Variável.campo

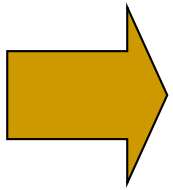
Registros

- Um campo de uma estrutura pode ser uma outra estrutura.
- Quando isso ocorre, temos uma estrutura aninhada.
- O padrão ANSI C especifica que as estruturas podem ser aninhadas até 15 níveis, mas a maioria dos compiladores permite mais.

Registros

- Exemplo:

```
typedef struct estruturaAluno{
      char nome[50];
      char cpf[17];
      int idade;
      float coeficiente;
}tipoAluno;
```

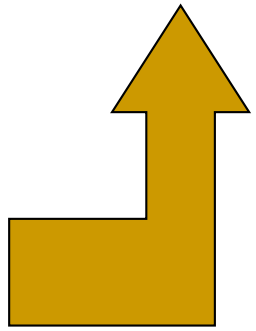


```
typedef struct estrMatDisciplinas{
      tipoAluno aluno;
      char disciplina[30];
}tipoMatDisciplinas;
```

Registros

- Exemplo:

```
typedef struct estrMatDisciplinas{
        tipoAluno aluno;
        char disciplina[30];
}tipoMatDisciplinas;
```



```
tipoMatDisciplinas matricula;

cout<<"Digite o nome do aluno: ";
getline (cin, matricula.aluno.nome);

cout << "Digite a disciplina do aluno: ";
getline (cin, matricula.disciplina);
```

getline() --> Aplicável ao tipo String do c++

Registros

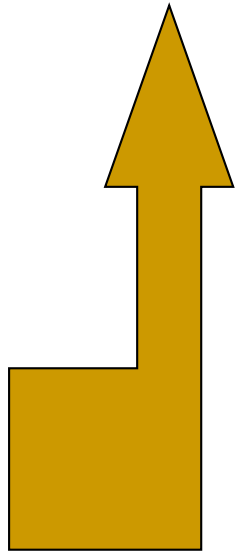
- Exemplo:

```
typedef struct estrMatDisciplinas{
        tipoAluno aluno;
        string disciplina;
}tipoMatDisciplinas;
```

```
tipoMatDisciplinas matricula;

cout<<"Digite o nome do aluno: ";
getline (cin, matricula.aluno.nome);

cout << "Digite a disciplina do aluno: ";
getline (cin, matricula.disciplina);
```



getline() --> Aplicável ao tipo String do c++

Registros

- Vetor de Estruturas:
  - Usado quando precisamos de diversas cópias de uma estrutura.
  - Por exemplo, cada aluno de AED constitui um elemento de um vetor, cujo tipo é uma estrutura de dados que define as características de cada aluno e a matéria.

```
struct estruturaAluno vetAlunos[50];
```

X

```
tipoAluno vetAlunos[50];
```

Registros

- Exercício:
  - Faça um programa que:
    - Leia um vetor “turma” de 5 alunos;
    - Cada registro/struct de aluno deve ter o numero de matrícula do aluno e suas notas de quatro bimestres.
    - Imprima a lista de matrícula e notas de cada aluno.
    - E calcule a média das quatro notas e imprima a matrícula e a média calcula para cada aluno.