Universidade Federal do ABC
Centro de Matemática, Computação e Cognição

Linguagem de Programação C++

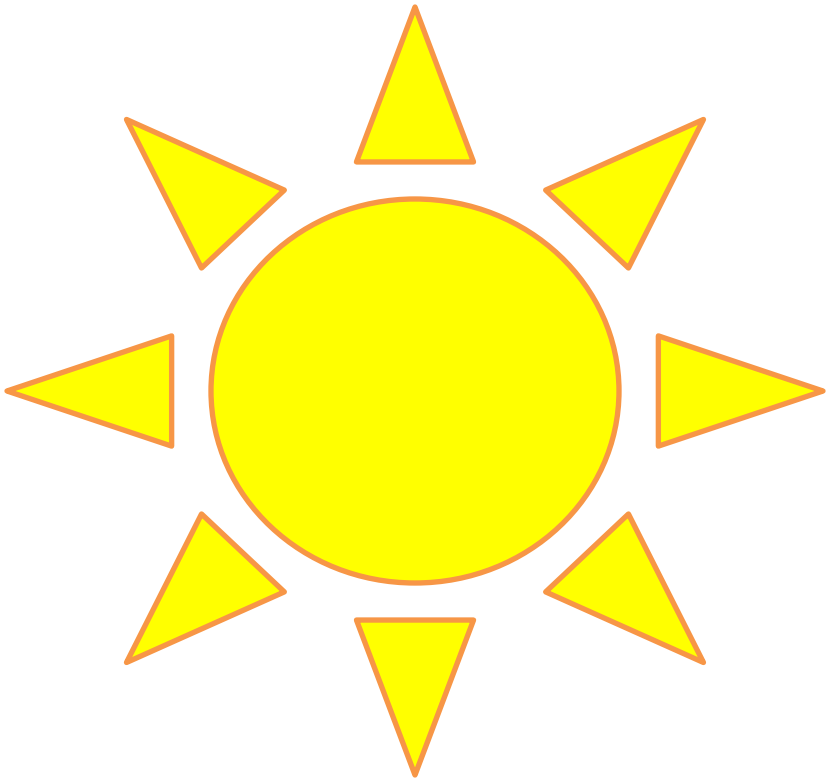
Monael Pinheiro Ribeiro, D.Sc.

Universidade Federal do ABC
Centro de Matemática, Computação e Cognição

Abstração e Tipos Abstratos de Dados

Monael Pinheiro Ribeiro, D.Sc.

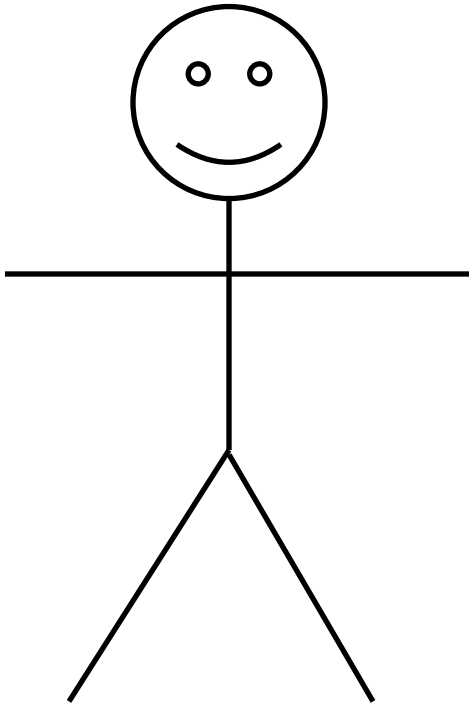
O que é isso?



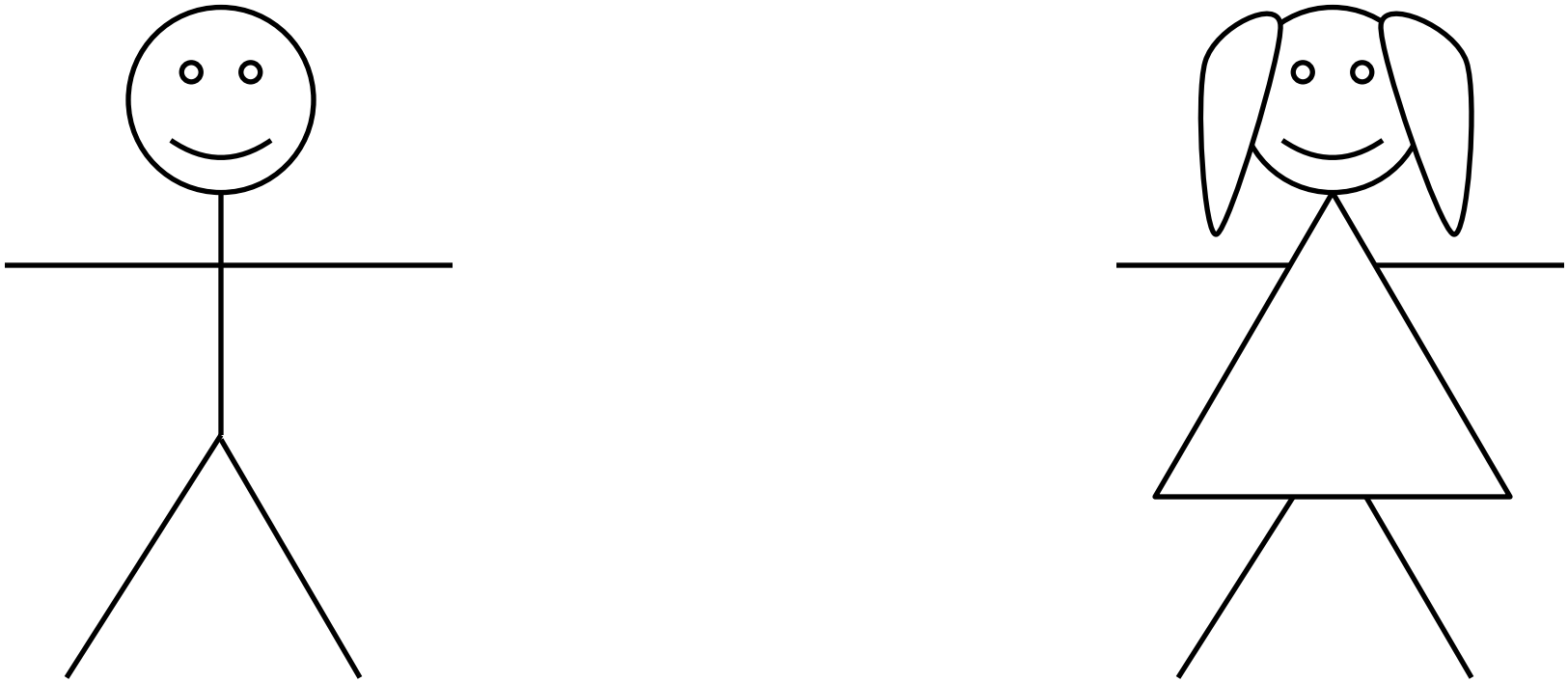
O que é isso?

O que é isso?

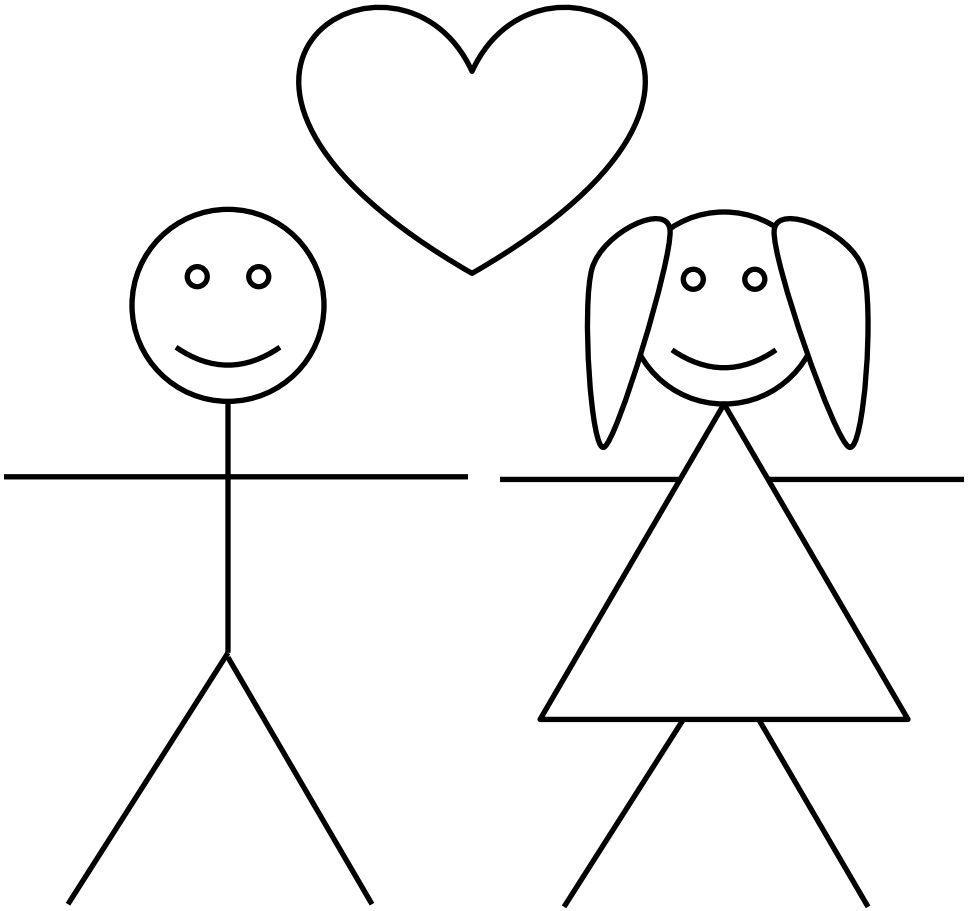
O que é isso?



O que é isso?



O que é isso?



Abstração

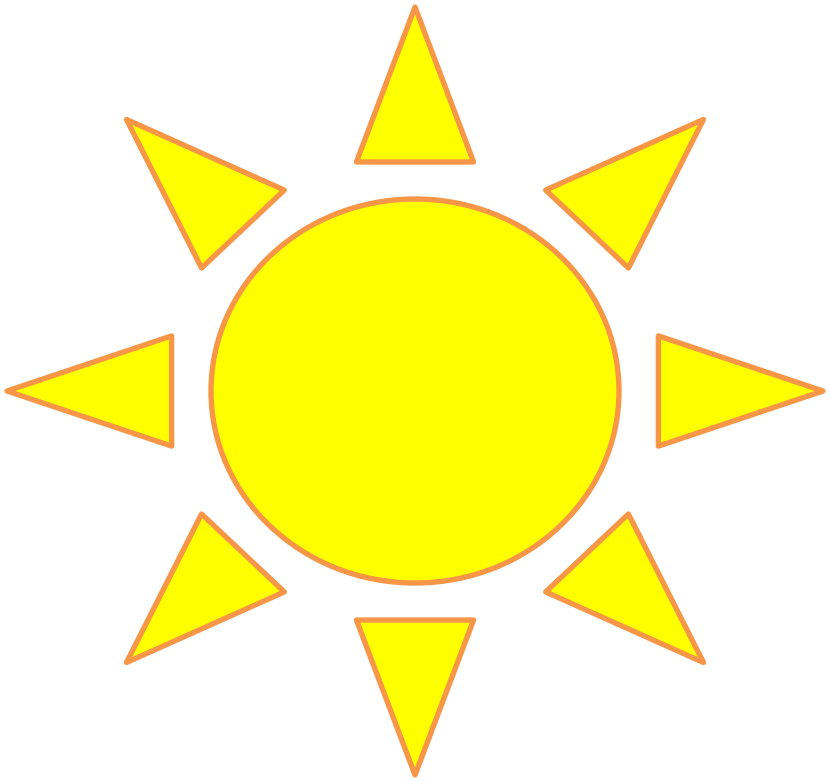
- Abstração é uma visão ou representação de uma entidade que inclui apenas os atributos mais significantes

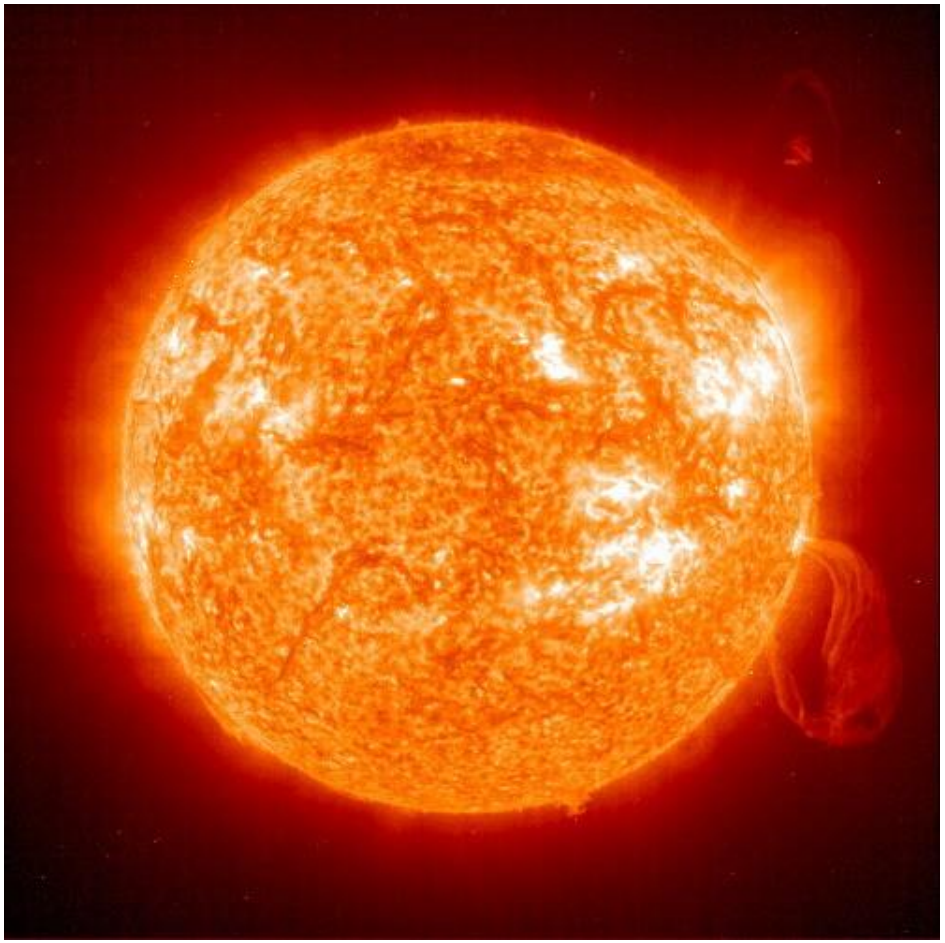
- Abstrair é:

  - Capacidade e habilidade em ater-se aos aspectos essenciais em um contexto.

  - Isolar o que importa e desprezar as características menos importantes.

Abstração

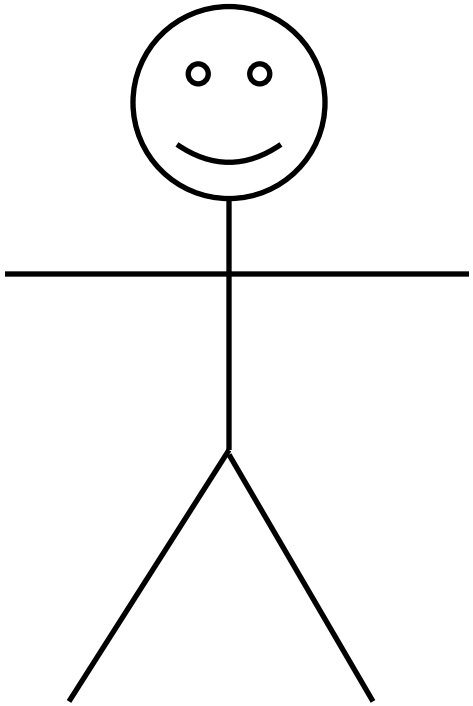




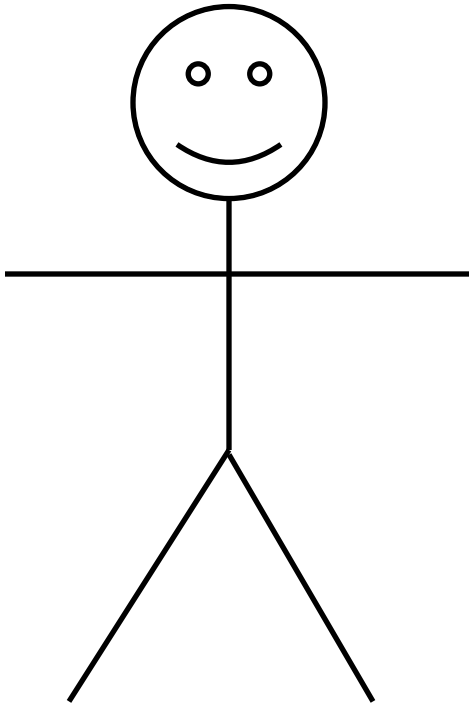
Abstração

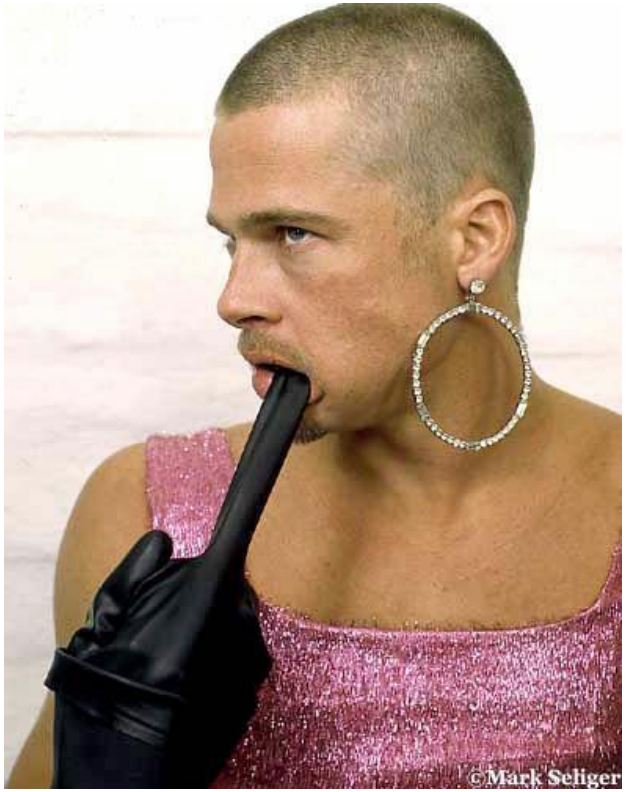
Abstração

Abstração

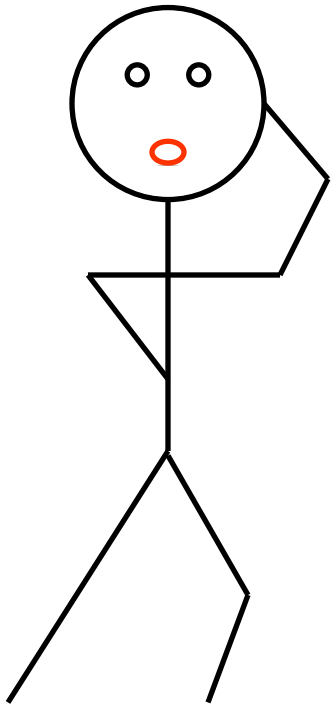


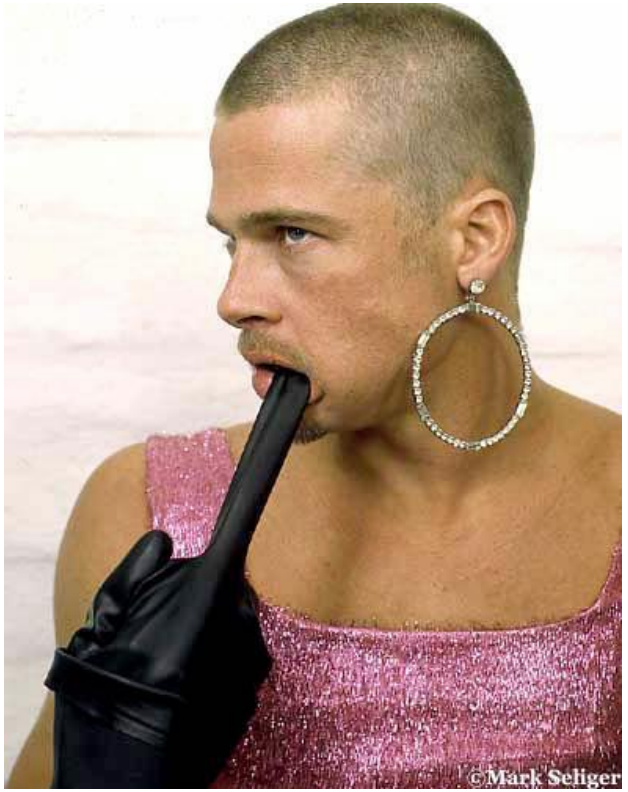
Abstração ?





Abstração





Abstração

- Em um programa de computador a abstração é uma técnica para lidar com a complexidade do problema e de sua codificação.

- Ao abstrair o problema é possível tornar o código menos complicado de ser implementado e principalmente mantido.

Tipos de Abstração em Programação

- Abstração de Processos:
  - Subprogramas (funções, procedimentos, métodos)
    - Isso esconde detalhes da implementação
    - Permite o reuso

Tipos de Abstração em Programação

- Abstração de Dados:
  - Tipos Abstratos de Dados
  - Encapsulamento
  - Instância: Objetos

Tipos Abstratos de Dados

- TADs são especificações de um conjunto de dados que definem um tipo.
- Também faz parte dos TADs as operações sobre esses dados.
- Os TADs propõem abstrair as variáveis corelatas em uma unidade entidade restrita com suas próprias operações.

Tipos Abstratos de Dados

- Exemplo:
  - Faça um programa que receba o nome, matrícula, data de nascimento e 3 notas de 40 alunos. E os mostre o nome e matrícula em ordem crescente de média das notas.

Tipos Abstratos de Dados

- Exemplo:
  - Dados envolvidos e correlatos:

Tipos Abstratos de Dados

- Exemplo:
  - Dados envolvidos e correlatos:
    - Nome
    - Matrícula
    - Data de Nascimento
      - Dia
      - Mês
      - Ano
    - 3 notas

Tipos Abstratos de Dados

- Exemplo:
  - Dados envolvidos e correlatos:
    - Nome
    - Matrícula
    - Data de Nascimento
      - Dia
      - Mês
      - Ano
    - 3 notas

Tipos Abstratos de Dados

- Exemplo:
  - Dados envolvidos e correlatos:
    - Nome
    - Matrícula
    - Data de Nascimento
      - Dia
      - Mês
      - Ano
    - 3 notas

Tipos Abstratos de Dados

- Exemplo:
  - Dados envolvidos e correlatos:
    - Nome (`std::string`)
    - Matrícula (`std::string`)
    - Data de Nascimento (`int, int, int`)[1]
      - Dia
      - Mês
      - Ano
    - 3 notas (`float*`)

Tipos Abstratos de Dados

```
#include <iostream>
int main()
{
    std::string nome1, nome2, nome3, ..., nome40;
    std::string mat1, mat2, mat3, ..., mat40;
    int dia1, dia2, dia3, ..., dia40;
    int mes1, mes2, mes3, ..., mes40;
    int ano1, ano2, ano3, ..., ano40;
    float nota1[3], nota2[3], nota3[3], ..., nota40[3];
    ...
    return 0;
}
```

Tipos Abstratos de Dados

- Exemplo:
  - Dados envolvidos e correlatos:
    - Nome (`std::string *`)
    - Matrícula (`std::string *`)
    - Data de Nascimento (`int*, int*, int*`)[1]
      - Dia
      - Mês
      - Ano
    - 3 notas (`float**`)

Tipos Abstratos de Dados

```
#include <iostream>
int main()
{
    std::string nomes[40];
    std::string matriculas[40];
    int dias[40]; //*
    int meses[40]; //*
    int anos[40]; //*
    float notas[40][3];
    ...
    return 0;
}
```

Tipos Abstratos de Dados

```
#include <iostream>
int main()
{
    std::string nomes[40];
    std::string matriculas[40];
    int dias[40]; //*
    int meses[40]; //*
    int anos[40]; //*
    float notas[40][3];
    ...
    return 0;
}
```

Tipos Abstratos de Dados

```
#include <iostream>
int main()
{
    std::string nomes[40];
    std::string matriculas[40];
    int dias[40]; //*
    int meses[40]; //*
    int anos[40]; //*
    float notas[40][3];
    ...
    return 0;
}
```

tAluno vetAlu[40];

Tipos Abstratos de Dados

```
#include <iostream>
int main()
{
    tAluno vetAlu[40];
    ...
    return 0;
}
```

Tipos Abstratos de Dados

- Definição de TADs

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
```

Tipos Abstratos de Dados

- Definição de TADs

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
```

Tipos Abstratos de Dados

- Declarando uma variável:

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
```

```
struct tAluno var;
```

Tipos Abstratos de Dados

- Declarando uma variável:

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
```

```
struct tAluno var;
```

Tipos Abstratos de Dados

- Declarando uma variável vetor:

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
```

```
struct tAluno turma[40];
```

Tipos Abstratos de Dados

- Declarando uma variável matriz:

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
```

```
struct tAluno escola[15][40];
```

Tipos Abstratos de Dados

- Acessando os campos (atributos) de TADs
  - Usa-se o operador de acesso (.) ponto

```
var.nome = “Fulano”;
std::cin >> var.matricula;
var.notas[0] = 10f;
var.notas[1] = 9.5f;
var.notas[2] = 7f;
```

Tipos Abstratos de Dados

- Acessando os campos (atributos) de TADs

```
std::cout << ((var.notas[0]+var.notas[1]+var.notas[2])/3f);
```

Tipos Abstratos de Dados

- O TAD trata-se de um tipo definido pelo usuário
  - Por este motivo, tudo que você faz com uma variável de tipo primitivo, pode ser feito com o TAD.

  - Argumento de Funções (métodos):

```
float calcMedia(struct tAluno a)
{ ... }
```

  - Chamada de Função

```
media = calcMedia(var);
```

Tipos Abstratos de Dados

- O TAD trata-se de um tipo definido pelo usuário
  - Por este motivo, tudo que você faz com uma variável de tipo primitivo, pode ser feito com o TAD.

  - Argumento de Funções (métodos) Vetor:

```
float calcMedia(struct tAluno *cl, int n)
{ ... }
```

  - Chamada de Função

```
media = calcMedia(vetAlu, tamanho);
```

Tipos Abstratos de Dados

- O TAD trata-se de um tipo definido pelo usuário
  - Por este motivo, tudo que você faz com uma variável de tipo primitivo, pode ser feito com o TAD.

  - Ponteiro:

    ```
    struct tAluno *ptAluno;
    ```

  - Alocação Dinâmica:

    ```
    ptAluno = new tAluno();
    ```

Tipos Abstratos de Dados

- O TAD trata-se de um tipo definido pelo usuário
  - Por este motivo, tudo que você faz com uma variável de tipo primitivo, pode ser feito com o TAD.

  - Ponteiro:

    ```
    struct tAluno *ptAluno;
    ```

  - Alocação Dinâmica:

    ```
    ptAluno = new tAluno();
    ```

  - Acessando os campos a partir de ponteiro:

    ```
    *(ptAluno).nome;
    ```

Tipos Abstratos de Dados

- O TAD trata-se de um tipo definido pelo usuário
  - Por este motivo, tudo que você faz com uma variável de tipo primitivo, pode ser feito com o TAD.

  - Ponteiro:

    ```
    struct tAluno *ptAluno;
    ```

  - Alocação Dinâmica:

    ```
    ptAluno = new tAluno();
    ```

  - Acessando os campos a partir de ponteiro:
    `*(ptAluno).nome;` ⇔ `ptAluno->nome;`

Tipos Abstratos de Dados

- O TAD trata-se de um tipo definido pelo usuário
  - Por este motivo, tudo que você faz com uma variável de tipo primitivo, pode ser feito com o TAD.

  - Ponteiro:

```
struct tAluno *ptAluno;
```

  - Alocação Dinâmica:

```
ptAluno = new tAluno();
```

  - Acessando os campos a partir de ponteiro:

```
*(ptAluno).nome; ⇔ ptAluno->nome;
```

Operador -> : Opera sobre um ponteiro para uma TAD.
Resolve o endereço de memória e posteriormente acessa o campo.

Tipos Abstratos de Dados

- O TAD trata-se de um tipo definido pelo usuário
  - Por este motivo, tudo que você faz com uma variável de tipo primitivo, pode ser feito com o TAD.

  - Ponteiro:

    ```
    struct tAluno *ptAluno;
    ```

  - Alocação Dinâmica:

    ```
    ptAluno = new tAluno();
    ```

  - Acessando os campos a partir de ponteiro:

    ```
    *(ptAluno).notas[i];
    ```

Tipos Abstratos de Dados

- O TAD trata-se de um tipo definido pelo usuário
  - Por este motivo, tudo que você faz com uma variável de tipo primitivo, pode ser feito com o TAD.

  - Ponteiro:

    ```
    struct tAluno *ptAluno;
    ```

  - Alocação Dinâmica:

    ```
    ptAluno = new tAluno();
    ```

  - Acessando os campos a partir de ponteiro:
    `*(ptAluno).notas[i];` ⇔ `ptAluno->notas[i];`

Tipos Abstratos de Dados

- O TAD trata-se de um tipo definido pelo usuário
  - Por este motivo, tudo que você faz com uma variável de tipo primitivo, pode ser feito com o TAD.

  - Ponteiro:

```
struct tAluno *ptTurma;
```

  - Alocação Dinâmica de um vetor de tamanho N:

```
ptTurma = new tAluno[n];
```

Tipos Abstratos de Dados

- O TAD trata-se de um tipo definido pelo usuário
  - Por este motivo, tudo que você faz com uma variável de tipo primitivo, pode ser feito com o TAD.

  - Atribuição direta (na declaração):

```
struct tAluno alu = {“Fulano”, “12345678”,
                       15, 11, 1990,
                       {5.3f, 9.1f, 10f} };
```

Tipos Abstratos de Dados

- Acessando os campos (atributos) de TADs

```
int main() {
        struct tAluno alu;
        std::cin >> alu.nome;
        std::cin >> alu.matricula;
        std::cin >> alu.dia;
        std::cin >> alu.mes;
        std::cin >> alu.ano;
        std::cin >> alu.notas[0];
        std::cin >> alu.notas[1];
        std::cin >> alu.notas[2];
        std::cout << "Nome .......: " << alu.nome << std::endl;
        std::cout << "Matricula ..: " << alu.matricula << std::endl;
        std::cout << "Data Nacto .: " << alu.dia << "/" << alu.mes << "/"
                                                  << alu.ano << std::endl;
        std::cout << "Notas ......: " << alu.notas[0] << " " << alu.notas[1]
                                             << " " << alu.notas[2] << std:endl;
        std::cout << "Media ......: " << (alu.notas[0] + alu.notas[1] +
                                               alu.notas[2])/3f) << std::endl;
        return 0;
}
```

Tipos Abstratos de Dados

- Exemplo:
  - Faça um programa que receba o nome, matrícula, data de nascimento e 3 notas de N alunos. E os mostre o nome e matricula em ordem crescente de média das notas.

Parte da Solução

```
struct tAluno {
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
int main() {
   struct tAluno *vetAlu;
   int i, n;
   std::cin >> n;
   vetAlu = new tAluno[n];
   for(i=0; i<n; i++) {
        std::cin >> vetAlu[i].nome;
        std::cin >> vetAlu[i].matricula;
        ...
   }
   ordenaAlunos(vetAlu, n);
   for(i=0; i<n; i++) {
        ...
   }
   return 0;
}
```

Parte da Solução

```
struct tAluno {
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
int main() {
   struct tAluno *vetAlu;
   int i, n;
   std::cin >> n;
   vetAlu = new tAluno[n];
   for(i=0; i<n; i++) {
        std::cin >> vetAlu[i].nome;
        std::cin >> vetAlu[i].matricula;
        ...
   }
   ordenaAlunos(vetAlu, n);
   for(i=0; i<n; i++) {
        ...
   }
   return 0;
}
```

Parte da Solução

```
struct tAluno {
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
int main() {
   struct tAluno *vetAlu;
   int i, n;
   std::cin >> n;
   vetAlu = new tAluno[n];
   for(i=0; i<n; i++) {
        std::cin >> vetAlu[i].nome;
        std::cin >> vetAlu[i].matricula;
        ...
   }
   ordenaAlunos(vetAlu, n);
   for(i=0; i<n; i++) {
        ...
   }
   return 0;
}
```

Abstração

Abstração

Tipos Abstratos de Dados

- Definição de TADs

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
```

Tipos Abstratos de Dados

- Definição de TADs

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    int dia, mes, ano; //*
    float notas[3];
};
```

Tipos Abstratos de Dados

- Exemplo:
  - Dados envolvidos e correlatos:
    - Nome
    - Matrícula
    - Data de Nascimento
      - Dia
      - Mês
      - Ano
    - 3 notas

Tipos Abstratos de Dados

- Exemplo:
  - Dados envolvidos e correlatos:
    - Nome
    - Matrícula
    - Data de Nascimento
      - Dia
      - Mês
      - Ano
    - 3 notas

Tipos Abstratos de Dados

- Exemplo:
  - Dados envolvidos e correlatos:
    - Nome
    - Matrícula
    - Data de Nascimento
      - Dia
      - Mês
      - Ano
    - 3 notas

Tipos Abstratos de Dados

- Exemplo:
  - Dados envolvidos e correlatos:
    - Nome
    - Matrícula
    - Data de Nascimento
      - Dia
      - Mês
      - Ano

      `Data aniversario;`
    - 3 notas

Tipos Abstratos de Dados

- Composição de TADs
  - TADs são especificações de um conjunto de dados que definem um tipo.
  - Tais dados podem inclusive estar em outros TADs.
  - Por exemplo:
    - TAD Data
    - TAD Aluno

Tipos Abstratos de Dados

- TAD Data:
  - Dados envolvidos e correlatos:
    - Dia
    - Mês
    - Ano

Tipos Abstratos de Dados

- TAD Data:
  - Dados envolvidos e correlatos:
    - Dia
    - Mês
    - Ano

Abstração

Tipos Abstratos de Dados

- Definição da TAD Data

```
struct tData
{
    int dia;
    int mes;
    int ano;
};
```

Tipos Abstratos de Dados

- Definição da TAD Data

```
struct tData
{
    int dia;
    int mes;
    int ano;
};
```

Abstração

Tipos Abstratos de Dados

- TADs da Solução

```
struct tData
{
   int dia;
   int mes;
   int ano;
};

struct tAluno
{
    std::string nome;
    std::string matricula;
    struct tData dtNascimento;
    float[] notas;
};
```

Tipos Abstratos de Dados

- TADs da Solução

```
struct tData
{
    int dia;
    int mes;
    int ano;
};

struct tAluno
{
     std::string nome;
     std::string matricula;
     struct tData dtNascimento;
     float[] notas;
};
```

Abstração

Abstração

Tipos Abstratos de Dados

- TADs da Solução

```
struct tData
{
    int dia;
    int mes;
    int ano;
};

struct tAluno
{
    std::string nome;
    std::string matricula;
    struct tData dtNascimento;
    float[] notas;
};
```

Abstração

Abstração

Parte da Solução

```
struct tData {
int dia, mes, ano;
};
struct tAluno {
    std::string nome;
    std::string matricula;
    struct tData dtNascimento;
    float[] notas;
};
int main() {
   struct tAluno *vetAlu;
   int i, n;
   std::cin >> n;
   vetAlu = new Aluno[n];
   for(i=0; i<n; i++) {
      ...
      std::cin >> vetAlu[i].dtNascimento.dia;
      std::cin >> vetAlu[i].dtNascimento.mes;
      std::cin >> vetAlu[i].dtNascimento.ano;
      ...
```

Auto-referências

- Um TAD pode se auto referenciar:
  - Exemplo:

Auto-referências

- Um TAD pode se auto referenciar:

```
1.  struct tElemento
2.  {
3.          int x;
4.          int y;
5.          struct tElemento *proximo;
6.  };
```

Auto-referências

- Um TAD pode se auto referenciar:

```
1.  struct tElemento
2.  {
3.          int x;
4.          int y;
5.          struct tElemento *proximo;
6.  };
```

- Utilidade: Uma lista dinâmica de elementos.

Lista Dinâmica

- TAD Data

```
struct tData
{
   int dia;
   int mes;
   int ano;
};
```

- TAD Aluno

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    struct tData dtNascimento;
    float[] notas;
};
```

Lista Dinâmica

- TAD Data

```
struct tData
{
   int dia;
   int mes;
   int ano;
};
```

- TAD Aluno

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    struct tData dtNascimento;
    float[] notas;
};
```

```
struct tElemento
{
    struct tAluno *item;
    struct tElemento *proximo;
};
```

Lista Dinâmica

- TAD Data

```
struct tData
{
   int dia;
   int mes;
   int ano;
};
```

- TAD Aluno

```
struct tAluno
{
    std::string nome;
    std::string matricula;
    struct tData dtNascimento;
    float[] notas;
};
```

- TAD Container

```
struct tElemento
{
    struct tAluno *item;
    struct tElemento *proximo;
};
```