



UNIVERSIDADE  
FEDERAL DO CEARÁ

UNIVERSIDADE FEDERAL DO CEARÁ - CAMPUS CRÁTEUS

CURSOS: CIÊNCIA DA COMPUTAÇÃO E SISTEMA DE INFORMAÇÃO

DISCIPLINA: ESTRUTURAS DE DADOS

PROFS. BRUNO DE CASTRO E WELLINGTON FRANCO

## NOTA DE AULA SOBRE TIPOS ABSTRATOS DE DADOS

### 1. ABSTRAÇÃO

Abstração é o ato de resumir características ou ações similares compartilhadas entre dois ou mais objetos. É importante, pois quando bem-feita reforça o paradigma de virtualizar o mundo real.

Em Computação, esta abordagem serve como premissa para estruturar dados complexos, nos quais as informações são compostas por diversos campos. As linguagens de programação fornecem mecanismos distintos para que um programador possa abstrair e estruturar dados complexos. Em C, o mecanismo mais simples para se estruturar dados complexos vem por meio da diretiva **struct**. Eis a sintaxe da diretiva:

```
struct nome_da_estrutura{  
    tipo campo1;  
    ...  
    tipo campoN;  
};
```

Um exemplo prático seria fazer um programa que calcula notas de uma turma. Uma turma é composta por alunos. Pensamos em aluno como uma entidade do mundo real que precisa ser abstraída computacionalmente. Quais atributos (características) todos os alunos têm em comum? Poderíamos citar: nome, matricula, nota1, nota2, notaFinal. Para evitar desperdício de memória, consideremos também, que o número máximo de caracteres que o atributo nome de aluno pode ter é da ordem de 81.

Programando em C, utilizaríamos então a diretiva **struct** para implementar a entidade aluno em nosso programa:

```

struct aluno {

    char nome[81];
    int matricula;
    float nota1;
    float nota2;
    float notaFinal;

};

int main( ){

    struct aluno aluno1;
    aluno1.nota1 = 7;
    aluno1.nota2 = 5;

    float media = (aluno1.nota1+aluno1.nota2)/2;

    printf("%f",media);

    return 1;

}

```

C também fornece a diretiva **typedef**, muito útil para melhorar a leitura e interpretação do código:

```

typedef struct aluno {

    char nome[81];
    int matricula;
    float nota1;
    float nota2;
    float notaFinal;

} Aluno;

float calcularMedia(Aluno aluno) {

    return (aluno.nota1 + aluno.nota2)/2;

}

int main( ){

    Aluno aluno1;
    aluno1.nota1 = 7;
    aluno1.nota2 = 5;

    float media = calcularMedia(aluno1);

    printf("%f",media);

    return 1;

}

```

## 2. TIPOS ABSTRATOS DE DADOS

A linguagem C possui mecanismos mais sofisticados para se abstrair entidades do mundo real: modularização; e Tipo Abstrato de Dados (TAD). O primeiro foi concebido pelos projetistas para suportar a programação de sistemas em C. Através da modularização é possível subdividir um arquivo de código fonte extenso em arquivos menores, sendo que cada um destes arquivos menores tem um escopo.

Um TAD é um tipo de dados estruturado em termos das operações que elas suportam, e não da maneira como elas são implementadas. Quando criamos um TAD, normalmente utilizamos dois arquivos: **um com extensão .h; e outro com extensão .c. No arquivo .h de um TAD, codificamos apenas definições. Já no arquivo .c de um TAD, codificamos a sua lógica de programação.** Desta forma, entende-se que um TAD estabelece o conceito de tipo de dado divorciado da sua representação.

## 3. EXEMPLO PRÁTICO

Voltemos ao exemplo abordado na seção 1 sobre o programa para calcular notas de um turma. No sentido de deixar seu programa mais elegante, performático e de fácil manutenção, entenda que deve abstrair a entidade aluno utilizando não só **struct**, mas deverá empregar também em seu código os mecanismos de modularização e TAD.

**A composição da estrutura Aluno (struct aluno) não é exportada pelo módulo. Dessa forma, os demais módulos que usarem esse TAD não poderão acessar diretamente os campos dessa estrutura. Os arquivos que usarem esse TAD só terão acesso às informações que possam ser obtidas através das funções exportadas pelo arquivo aluno.h.**

Nas páginas que se seguem, é exposto do código fonte do TAD de Aluno e como se deve usá-lo no arquivo **main.c**.

**//Fonte do arquivo main.c:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include "aluno.h"**

**int main(){**

**Aluno\* aluno1 = criaAluno();**

**setNome(aluno1, "Joao");**

**setNota1(aluno1, 5);**

**setNota2(aluno1, 8);**

**float media = calcularMedia(aluno1);**

**char \*nome = getNome(aluno1);**

**printf("A media de %s foi: %f",nome, media);**

**return 1;**

**}**

**//Fonte do arquivo aluno.h:**

**/\* TAD: Aluno\*/**

**/\* Tipo exportado\*/**

**typedef struct aluno Aluno;**

**/\* A composição da estrutura Aluno (struct aluno) não é exportada pelo módulo. Dessa forma, os demais módulos que usarem esse TAD não poderão acessar**

**diretamente os campos dessa estrutura. Os arquivos que usarem esse TAD só terão acesso às**

**informações que possam ser obtidas através das funções exportadas pelo arquivo**

**aluno.h. \*/**

**/\*\* Dessa forma se faz necessário termos uma função para criar o nosso TAD \*/**

**/\* Função construtora de Aluno na memória do computador\*/**

**Aluno\* criaAluno();**

**/\*\* E também, funções para acessar os atributos do nosso TAD e passar valores para estes atributos \*/**

**char\* getNome(Aluno\* aluno);**

**void setNome(Aluno\* aluno, char \*nome);**

**int getMatricula(Aluno\* aluno);**

**void setMatricula(Aluno\* aluno, int matricula);**

**float getNota1(Aluno\* aluno);**

**void setNota1(Aluno\* aluno, float nota1);**

**float getNota2(Aluno\* aluno);**

**void setNota2(Aluno\* aluno, float nota2);**

**float getNotaFinal(Aluno\* aluno) ;**

**void setNotaFinal(Aluno\* aluno, float notaFinal);**

**int getQuantidadeFaltas(Aluno\* aluno);**

**void setQuantidadeFaltas(Aluno\* aluno, int quantidadeFaltas);**

**/\*Demais funcionalidades exportadas\*/**

**float calcularMedia(Aluno\* aluno);**

```
//Fonte do arquivo aluno.c:
#include <stdio.h>
#include <stdlib.h>
#include "aluno.h"

struct aluno {
    char *nome;
    int matricula;
    float nota1;
    float nota2;
    float notaFinal;
    int quantidadeFaltas;
};

Aluno* criaAluno(){
    return (Aluno*) malloc(sizeof(Aluno));
}

char* getNome(Aluno* aluno){
    return aluno->nome;
}

void setNome(Aluno* aluno, char *nome){
    aluno->nome = nome;
}

int getMatricula(Aluno* aluno){
    return aluno->matricula;
}

void setMatricula(Aluno* aluno, int matricula){
    aluno->matricula = matricula;
}

float getNota1(Aluno* aluno){
    return aluno->nota1;
}

void setNota1(Aluno* aluno, float nota1){
    aluno->nota1 = nota1;
}

float getNota2(Aluno* aluno){
    return aluno->nota2;
}

void setNota2(Aluno* aluno, float nota2){
    aluno->nota2 = nota2;
}
```

```
float getNotaFinal(Aluno* aluno){
    return aluno->notaFinal;
}

void setNotaFinal(Aluno* aluno, float notaFinal){
    aluno->notaFinal = notaFinal;
}

int getQuantidadeFaltas(Aluno* aluno){
    return aluno->quantidadeFaltas;
}

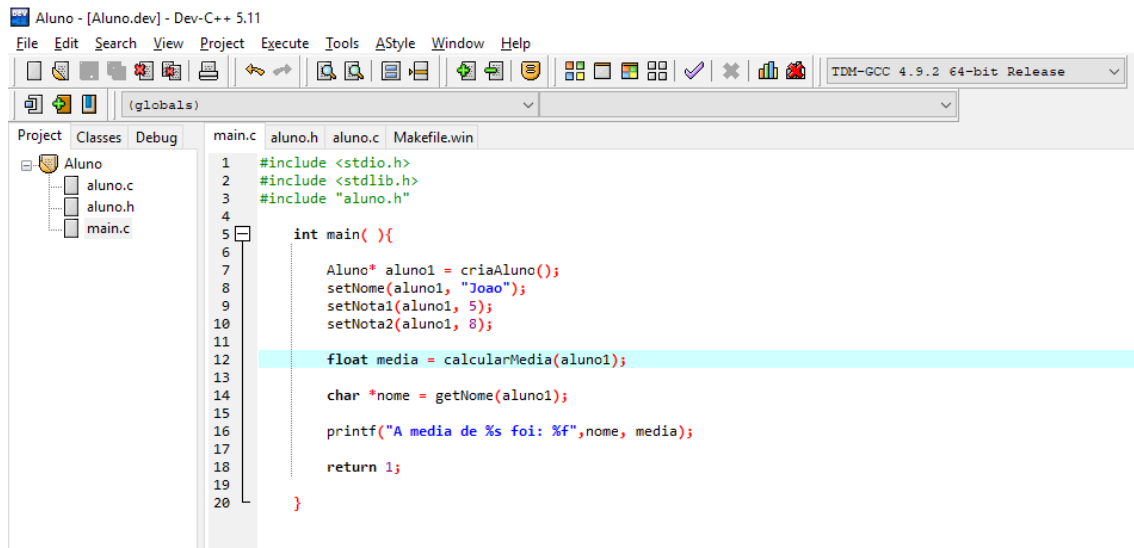
void setQuantidadeFaltas(Aluno* aluno, int quantidadeFaltas){
    aluno->quantidadeFaltas = quantidadeFaltas;
}

float calcularMedia(Aluno* aluno){

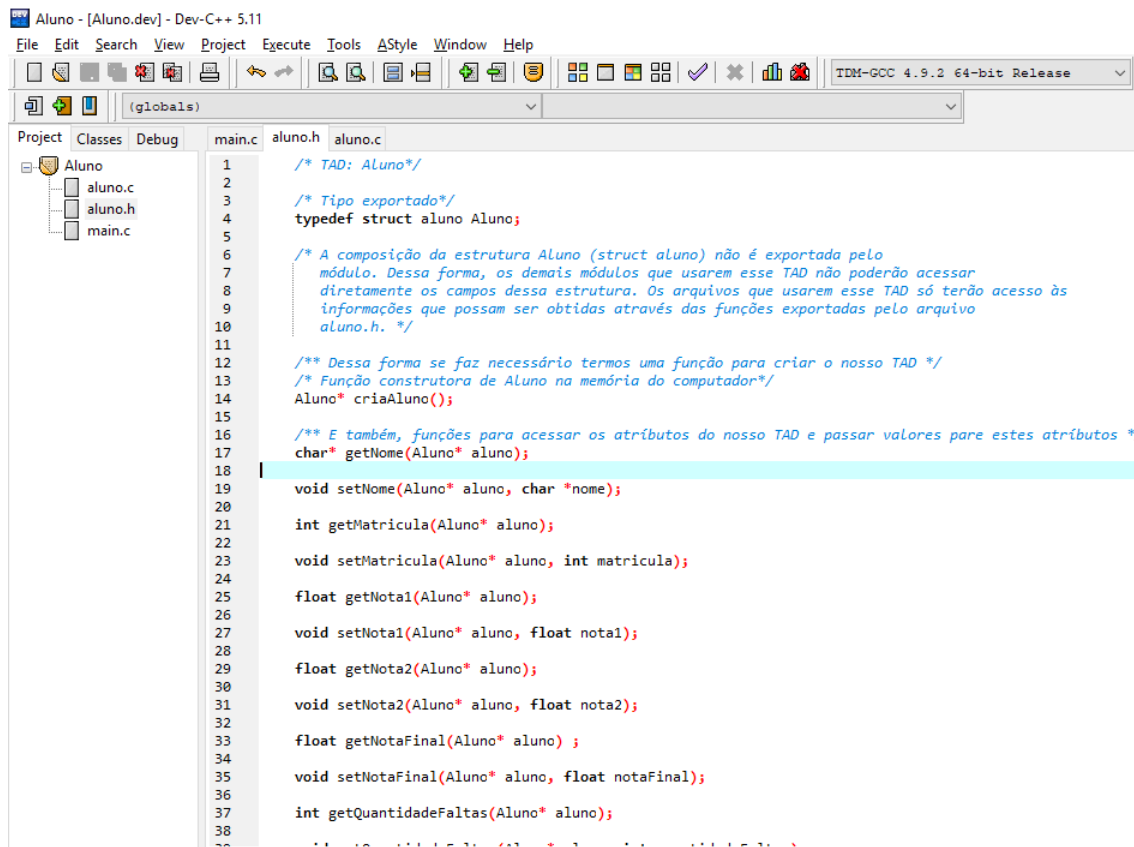
    return (aluno->nota1 + aluno->nota2)/2;

}
```

## Prints do projeto para elucidar a disposição dos arquivos no projeto:



```
Aluno - [Aluno.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug main.c aluno.h aluno.c Makefile.win
Aluno
├── aluno.c
├── aluno.h
└── main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "aluno.h"
4
5  int main( ){
6
7      Aluno* aluno1 = criaAluno();
8      setNome(aluno1, "Joao");
9      setNota1(aluno1, 5);
10     setNota2(aluno1, 8);
11
12     float media = calcularMedia(aluno1);
13
14     char *nome = getNome(aluno1);
15
16     printf("A media de %s foi: %f", nome, media);
17
18     return 1;
19
20 }
```



```
Aluno - [Aluno.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug main.c aluno.h aluno.c
1  /* TAD: Aluno*/
2
3  /* Tipo exportado*/
4  typedef struct aluno Aluno;
5
6  /* A composição da estrutura Aluno (struct aluno) não é exportada pelo
7  módulo. Dessa forma, os demais módulos que usarem esse TAD não poderão acessar
8  diretamente os campos dessa estrutura. Os arquivos que usarem esse TAD só terão acesso às
9  informações que possam ser obtidas através das funções exportadas pelo arquivo
10 aluno.h. */
11
12 /** Dessa forma se faz necessário termos uma função para criar o nosso TAD */
13 /* Função construtora de Aluno na memória do computador*/
14 Aluno* criaAluno();
15
16 /** E também, funções para acessar os atributos do nosso TAD e passar valores para estes atributos */
17 char* getNome(Aluno* aluno);
18
19 void setNome(Aluno* aluno, char *nome);
20
21 int getMatricula(Aluno* aluno);
22
23 void setMatricula(Aluno* aluno, int matricula);
24
25 float getNota1(Aluno* aluno);
26
27 void setNota1(Aluno* aluno, float nota1);
28
29 float getNota2(Aluno* aluno);
30
31 void setNota2(Aluno* aluno, float nota2);
32
33 float getNotaFinal(Aluno* aluno);
34
35 void setNotaFinal(Aluno* aluno, float notaFinal);
36
37 int getQuantidadeFaltas(Aluno* aluno);
38
```



Aluno - [Aluno.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help



(globals)

Project Classes Debug



```
1      #include <stdio.h>
2      #include <stdlib.h>
3      #include "aluno.h"
4
5      struct aluno {
6          char *nome;
7          int matricula;
8          float nota1;
9          float nota2;
10         float notaFinal;
11         int quantidadeFaltas;
12     };
13
14     Aluno* criaAluno(){
15         return (Aluno*) malloc(sizeof(Aluno));
16     }
17
18     char* getNome(Aluno* aluno){
19         return aluno->nome;
20     }
21
22     void setNome(Aluno* aluno, char *nome){
23         aluno->nome = nome;
24     }
25
26     int getMatricula(Aluno* aluno){
27         return aluno->matricula;
28     }
29
30     void setMatricula(Aluno* aluno, int matricula){
31         aluno->matricula = matricula;
32     }
33
34     float getNota1(Aluno* aluno){
35         return aluno->nota1;
36     }
37
38     void setNota1(Aluno* aluno, float nota1){
```