

# Arquivos - Organização

## Parte 1 - Campos

Prof.: Leonardo Tórtoro Pereira  
leonardop@usp.br

\*Material baseado em aulas dos professores: Elaine Parros Machado de Souza, Gustavo Batista, Robson Cordeiro, Moacir Ponti Jr., Maria Cristina Oliveira e Cristina Ciferri.

# Organização de Arquivos

- Dados em arquivos são, em geral, organizados de maneira lógica em campos e registros
- Campos e registros são conceitos lógicos
  - ◆ Associados ao arquivo lógico
  - ◆ Não correspondem necessariamente a uma organização física

# Organização de Arquivos

- Por que a organização lógica?
- Dependendo de como os dados são mantidos, campos lógicos sequer podem ser recuperados...

# Stream

# Sequência de Bytes (Stream)

→ Exemplo:

- ◆ Tarefa: armazenar em um arquivo os nomes e endereços de várias pessoas
- ◆ Representação dos dados: sequência simples de bytes
  - Sem delimitadores, contadores, etc

# Vamos Programar



## main.c

```
#include "StudentStreamFileHandler.h"
```

```
int main() {  
    createStudentFile();  
    readStudentFile();  
}
```

# StudentStreamFileHandler.h

```
#ifndef STUDENTSTREAMFILEHANDLER_H
#define STUDENTSTREAMFILEHANDLER_H
#include <stdio.h>
#define TAM 2
#define STUDENTFILENAME "student.txt"
typedef struct student_t
{
    char name[20];
    int age;
    int grade;
} student_t;
void createStudentFile();
student_t readStudentDataFromInput();
void writeStudentDataInFile(student_t, FILE*);
void readStudentFile();
student_t readStudentDataInFile(FILE*);
void printStudentData(student_t);
#endif //STUDENTSTREAMFILEHANDLER_H
```



# StudentStreamFileHandler.c (1/3)

```
#include "StudentStreamFileHandler.h"
void createStudentFile() {
    FILE *filePointer;
    filePointer = fopen(STUDENTFILENAME, "w");
    student_t student;
    printf("Lendo dados\n\n");
    for (int i = 0; i < TAM; ++i) {
        student = readStudentDataFromInput();
        writeStudentDataInFile(student, filePointer);
    }
    fclose(filePointer);
}
student_t readStudentDataFromInput(){
    student_t student;
    printf("nome do estudante: ");
    scanf("%s", student.name);
    printf("idade: ");
    scanf("%d", &student.age);
    printf("nota: ");
    scanf("%d", &student.grade);
    printf("\n");
    return student;
}
```

## StudentStreamFileHandler.c (2/3)

```
void writeStudentDataInFile(student_t student, FILE *filePointer)
{
    fprintf(filePointer, "%s", student.name);
    fprintf(filePointer, "%d", student.age);
    fprintf(filePointer, "%d", student.grade);
}

void readStudentFile()
{
    FILE *filePointer;
    filePointer = fopen(STUDENTFILENAME, "r");
    student_t student;
    printf("Dados no arquivo\n\n");
    for (int i = 0; i < TAM; ++i)
    {
        student = readStudentDataInFile(filePointer);
        printStudentData(student);
    }
    fclose(filePointer);
}
```

## StudentStreamFileHandler.c (3/3)

```
student_t readStudentDataInFile(FILE *filePointer)
{
    student_t student;
    fscanf(filePointer, "%s", student.name);
    fscanf(filePointer, "%d", &student.age);
    fscanf(filePointer, "%d", &student.grade);
    return student;
}

void printStudentData(student_t student)
{
    printf("Nome: %s\n", student.name);
    printf("Idade: %d\n", student.age);
    printf("Nota: %d\n\n", student.grade);
}
```

Qual a saída esperada para a seguinte entrada?

Nome	bruno	adriana
Idade	20	15
Nota	9	7

## Saída Real

Dados no arquivo

Nome: bruno209adriana157

Idade: 6422476

Nota: 1982450880

Nome: ■ 7■a

Idade: 6422476

Nota: 1982450880

# O Arquivo Continha:

bruno209adriana157

O Que Aconteceu?

## O Que Aconteceu?

- Salvar os campos com apenas o `printf` fez com que todos l
- Ao ler o campo `nome` como string, toda a entrada foi lida, pois o `fscanf("%s", ...)` lê tudo que não for espaço.



Ok... Como fazer do jeito certo?

# Organização em Campos!

# Organização em Campos

→ Campo:

- ◆ Menor unidade lógica de informação em arquivo
  - Noção lógica (ferramenta conceitual)
  - Não está associada a um conceito físico

→ Há várias maneiras de organizar um arquivo mantendo a identidade dos campos

- ◆ A organização anterior não proporciona isso...

# Organização em Campos

→ Estruturas de Organização de Campos:

- ◆ Comprimento fixo
- ◆ Indicador de comprimento
- ◆ Delimitadores
- ◆ Uso de tags

# Organização em Campos

(a)	Maria	Rua 1	123	São Carlos
	João	Rua A	255	Rio Claro
	Pedro	Rua 10	56	Rib. Preto

(b) 05Maria05Rua 10312310São Carlos  
04João05Rua A0325509Rio Claro  
05Pedro06Rua 10025610Rib. Preto

(c) Maria|Rua 1|123|São Carlos|  
João|Rua A|255|Rio Claro|  
Pedro|Rua 10|56|Rib. Preto|

(d) Nome=Maria|Endereço=Rua 1|Número=123|Cidade=São Carlos|  
Nome=João|Endereço=Rua A|Número=255|Cidade=Rio Claro|  
Nome=Pedro|Endereço=Rua 10|Número=56|Cidade=Rib. Preto|

# Campos com Tamanho Fixo

## Campos com Tamanho Fixo

→ Cada campo ocupa no arquivo um tamanho fixo, pré-estabelecido

### ◆ Exemplo

- Nome: string de 12 caracteres (12 bytes)
- Rua: string de 10 caracteres (10 bytes)
- Número: inteiro (4 bytes)
- Cidade: string de 20 caracteres (20 bytes)

## Campos com Tamanho Fixo

- O tamanho ser conhecido garante que é possível recuperar cada campo
  - ◆ Como?



# Campos com Tamanho Fixo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
M	A	R	I	A	b	b	b	b	b	b	b	R	U	A	b	1	b	b	b	b	b		123		S	A	O	b	

30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
C	A	R	L	O	S	b	b	b	b	b	b	b	b	b	b	J	O	A	O	b	b	b	b	b	b	b	b	R	U

60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
A	b	A	b	b	b	b	b		255		R	I	O	b	C	L	A	R	O	b	b	b	b	b	b	b	b	b	

90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
b	b	P	E	D	R	O	b	b	b	b	b	b	b	R	U	A	b	X	V	b	b	b	b		56		S	A	

120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149
O	b	C	A	R	L	O	S	b	b	b	b	b	b	b	b	b	b												

# Campos com Tamanho Fixo

→ Prós e Contras?

- ◆ Organização simples para ler e escrever dados
- ◆ O espaço alocado (e não usado) aumenta desnecessariamente o tamanho do arquivo
  - Desperdício de memória secundária

## Campos com Tamanho Fixo

- Dado precisa ser truncado se ultrapassar o tamanho do campo
- Solução inapropriada quando se tem uma grande variabilidade nos tamanhos reais dos campos
- Razoável apenas se o comprimento dos campos é realmente fixo, ou apresenta pouca variação

# Vamos Programar



## main.c

```
#include "StudentFixedSizeFieldFileHandler.h"
```

```
int main() {  
    createStudentFile();  
    readStudentFile();  
}
```

# Student.h

```
#ifndef STUDENT_H_
#define STUDENT_H_
#include <stdio.h>
typedef struct student_t
{
    char name[20];
    int age;
    int grade;
} student_t;
student_t readStudentDataFromInput();
void printStudentData(student_t);

#endif //STUDENT_H_
```

# Student.c

```
#include "Student.h"
student_t readStudentDataFromInput(){
    student_t student;
    printf("nome do estudante: ");
    scanf("%s", student.name);
    printf("idade: ");
    scanf("%d", &student.age);
    printf("nota: ");
    scanf("%d", &student.grade);
    printf("\n");
    return student;
}
void printStudentData(student_t student){
    printf("Nome: %s\n", student.name);
    printf("Idade: %d\n", student.age);
    printf("Nota: %d\n\n", student.grade);
}
```

# FileHandler.h

```
#ifndef FILEHANDLER_H_
#define FILEHANDLER_H_

#include <stdio.h>
#include "Student.h"
#include "StudentTagDelimiterFileHandler.h"
#define TAM 2
#define STUDENTFILENAME "student.txt"

void createStudentFile();
void readStudentFile();

#endif //FILEHANDLER_H_
```



## FileHandler.c (1/2)

```
#include "FileHandler.h"

void createStudentFile() {
    FILE *filePointer;
    filePointer = fopen(STUDENTFILENAME, "w");
    student_t student;
    printf("Lendo dados\n\n");
    for (int i = 0; i < TAM; ++i)
    {
        student = readStudentDataFromInput();
        writeTagDelimitedStudentDataInFile(student, filePointer);
    }
    fclose(filePointer);
}
```

## FileHandler.c (2/2)

```
void readStudentFile()
{
    FILE *filePointer;
    filePointer = fopen(STUDENTFILENAME, "r");
    student_t student;
    printf("Dados no arquivo\n\n");
    for (int i = 0; i < TAM; ++i)
    {
        student = readTagDelimitedStudentDataInFile(filePointer);
        printStudentData(student);
    }
    fclose(filePointer);
}
```

# StudentFixedSizeFieldFileHandler.h

```
#ifndef STUDENTFIXEDSIZEFIELDFILEHANDLER_H_
#define STUDENTFIXEDSIZEFIELDFILEHANDLER_H_

#include <stdio.h>
#include "Student.h"

void writeFixedStudentDataInFile(student_t, FILE*);
student_t readFixedStudentDataInFile(FILE*);

#endif //STUDENTFIXEDSIZEFIELDFILEHANDLER_H_
```

# StudentFixedSizeFieldFileHandler.c

```
#include "StudentFixedSizeFieldFileHandler.h"

void writeFixedStudentDataInFile(student_t student, FILE *filePointer)
{
    fwrite(&student, sizeof(student_t), 1, filePointer);
}

student_t readFixedStudentDataInFile(FILE *filePointer)
{
    student_t student;
    fread(&student, sizeof(student_t), 1, filePointer);
    return student;
}
```

# Campos com Indicador de Comprimento

## Campos com Indicador de Comprimento

- O tamanho de cada campo é armazenado imediatamente antes da informação
  - ◆ Indicador de comprimento: usualmente um inteiro (4 bytes)
  - ◆ Em arquivos binários: um único byte para armazenamento do indicador de comprimento se o tamanho do campo é inferior a 256 bytes

# Campos com Indicador de Comprimento

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
5				M	A	R	I	A	5				R	U	A	b	1	4				123				10			

30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
S	A	O	b	C	A	R	L	O	S	4				J	O	A	O	5				R	U	A	b	A	4		

60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
255				9				R	I	O	b	C	L	A	R	O	5				P	E	D	R	O	6			

90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
R		U	A	b	X	V	4				56				10				S	A	O	b	C	A	R	L	O	S	

120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149

# Vamos Programar





# StudentFieldLengthIndicatorFileHandler.h

```
#ifndef STUDENTFIELDLenghtIndicatorFileHandler_H_
#define STUDENTFIELDLenghtIndicatorFileHandler_H_
#include <stdio.h>
#include <string.h>
#include "Student.h"
void writeLenghtIndicatorStudentDataInFile(student_t, FILE*);
student_t readLenghtIndicatorStudentDataInFile(FILE*);
#endif //STUDENTFIELDLenghtIndicatorFileHandler_H_
```

# StudentFieldLengthIndicatorFileHandler.c (1/2)

```
#include "StudentFieldLengthIndicatorFileHandler.h"
void writeLengthIndicatorStudentDataInFile(student_t student, FILE *filePointer)
{
    int fieldSize;
    fieldSize = strlen(student.name);
    fprintf(filePointer, "%d", fieldSize);
    fwrite(student.name, fieldSize, 1, filePointer);
    fieldSize = sizeof(student.age);
    fprintf(filePointer, "%d", fieldSize);
    fwrite(&student.age, fieldSize, 1, filePointer);
    fieldSize = sizeof(student.grade);
    fprintf(filePointer, "%d", fieldSize);
    fwrite(&student.grade, fieldSize, 1, filePointer);
}
```

# StudentFieldLengthIndicatorFileHandler.c (2/2)

```
#include "StudentFieldLengthIndicatorFileHandler.h"
```

```
student_t readLengthIndicatorStudentDataInFile(FILE *filePointer) {  
    student_t student;  
    int fieldSize;  
    fscanf(filePointer, "%d", &fieldSize);  
    fread(student.name, fieldSize, 1, filePointer);  
    student.name[fieldSize] = '\0';  
    fscanf(filePointer, "%d", &fieldSize);  
    fread(&student.age, fieldSize, 1, filePointer);  
    fscanf(filePointer, "%d", &fieldSize);  
    fread(&student.grade, fieldSize, 1, filePointer);  
    return student;  
}
```

# Campos Separados por Delimitadores

# Campos Separados por Delimitadores

- Caractere especial (delimitador) inserido ao final de cada campo
- ◆ Delimitador não pode ser um caractere válido
  - Espaços em branco não serviriam...
  - Ex: para campo nome podemos utilizar “|”, “#”, ...

# Campos Separados por Delimitadores

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
M	A	R	I	A		R	U	A	b	1		123					S	A	O	b	C	A	R	L	O	S		J	O

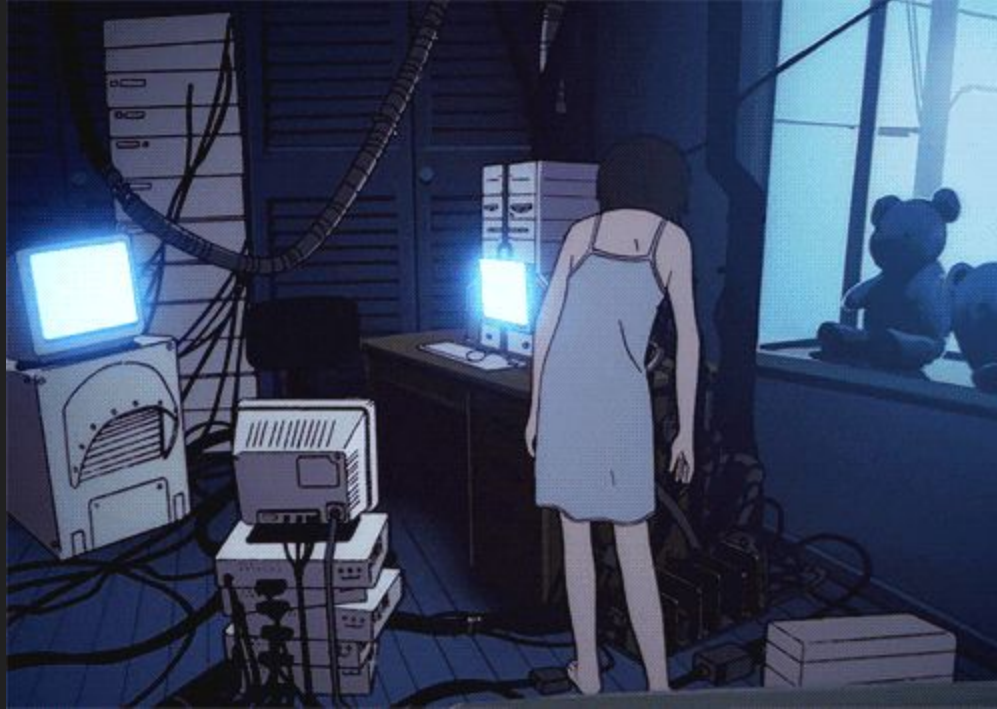
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
A	O		R	U	A	b	A		255				R	I	O	b	C	L	A	R	O		P	E	D	R	O		

60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
R	U	A	b	X	V		56				S	A	O	b	C	A	R	L	O	S									

90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119

120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149

# Vamos Programar



# StudentDelimiterFileHandler.h

```
#ifndef STUDENTDELIMITERFILEHANDLER_H_
#define STUDENTDELIMITERFILEHANDLER_H_
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "Student.h"
#define DELIMITERCHAR '|'
void writeDelimitedStudentDataInFile(student_t, FILE*);
student_t readDelimitedStudentDataInFile(FILE*);

#endif //STUDENTDELIMITERFILEHANDLER_H_
```



# StudentDelimiterFileHandler.c (1/3)

```
#include "StudentDelimiterFileHandler.h"
static char readCurrentInput(FILE *filePointer, char *currentInput)
{
    char currentChar;
    int currentInputIndex;
    currentInputIndex = 0;
    currentChar = fgetc(filePointer);
    while((currentChar != EOF) && (currentChar != DELIMITERCHAR))
    {
        currentInput[currentInputIndex++] = currentChar;
        currentChar = fgetc(filePointer);
    }
    currentInput[currentInputIndex] = '\0';
    return currentChar;
}
```

## StudentDelimiterFileHandler.c (2/3)

```
void writeDelimitedStudentDataInFile(student_t student, FILE *filePointer)
{
    fprintf(filePointer, "%s", student.name);
    fputc(DELIMITERCHAR, filePointer);
    fprintf(filePointer, "%d", student.age);
    fputc(DELIMITERCHAR, filePointer);
    fprintf(filePointer, "%d", student.grade);
    fputc(DELIMITERCHAR, filePointer);
}
```

## StudentDelimiterFileHandler.c (3/3)

```
student_t readDelimitedStudentDataInFile(FILE *filePointer)
{
    student_t student;
    char currentInput[50];
    readCurrentInput(filePointer, currentInput);
    strcpy(student.name, currentInput);
    readCurrentInput(filePointer, currentInput);
    student.age = atoi(currentInput);
    readCurrentInput(filePointer, currentInput);
    student.grade = atoi(currentInput);
    return student;
}
```

Campos com Tags “Keyword = Value”

# Campos com Tags “Keyword = Value”

- Expressão “keyword=value”
  - ◆ Colocada imediatamente antes do campo
  - ◆ Possui semântica que explica o significado do campo
- Geralmente usada em conjunto com outro método para campos (ex. delimitador, indicador de tamanho)

# Campos com Tags "Keyword = Value"

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
N	O	M	E	=	M	A	R	I	A		E	N	D	E	R	E	C	O	=	R	U	A	b	1		N	U	M	E

30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
R	O	=		123			C	I	D	A	D	E	=	S	A	O	b	C	A	R	L	O	S		N	O	M	E	

60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
=	J	O	A	O		E	N	D	E	R	E	C	O	=	R	U	A	b	A		N	U	M	E	R	O	=	255	

90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
		C	I	D	A	D	E	=	R	I	O	b	C	L	A	R	O		N	O	M	E	=	P	E	D	R	O	

120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149
	E	N	D	E	R	E	C	O	=	R	U	A	b	X	V		N	U	M	E	R	O	=		56			C	

150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
I	D	A	D	E	=	S	A	O	b	C	A	R	L	O	S														

# Campos com Tags “Keyword = Value”

## → Vantagens:

- ◆ Possui semântica local
  - Campo fornece informação sobre si próprio
- ◆ Permite identificar localmente o conteúdo do arquivo
- ◆ Permite campos não obrigatórios

## → Desvantagem:

- ◆ As keywords podem ocupar uma porção significativa do arquivo

# Vamos Programar





# StudentTagDelimiterFileHandler.h

```
#ifndef STUDENTTAGDELIMITERFILEHANDLER_H_
#define STUDENTTAGDELIMITERFILEHANDLER_H_

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "Student.h"
#define DELIMITERCHAR '|'
#define TAGDELIMITERCHAR '='
#define NAMETAG "NAME"
#define AGETAG "AGE"
#define GRADETAG "GRADE"
#define TOTALFIELDS 3
void writeTagDelimitedStudentDataInFile(student_t, FILE*);
student_t readTagDelimitedStudentDataInFile(FILE*);

#endif //STUDENTTAGDELIMITERFILEHANDLER_H_
```

# StudentTagDelimiterFileHandler.c (1/3)

```
#include "StudentTagDelimiterFileHandler.h"
static char readCurrentInput(FILE *filePointer, char *currentInput, char delimiter)
{
    char currentChar;
    int currentInputIndex;
    currentInputIndex = 0;
    currentChar = fgetc(filePointer);
    while((currentChar != EOF) && (currentChar != delimiter))
    {
        currentInput[currentInputIndex++] = currentChar;
        currentChar = fgetc(filePointer);
    }
    currentInput[currentInputIndex] = '\0';
    return currentChar;
}
```

## StudentTagDelimiterFileHandler.c (2/3)

```
void writeTagDelimitedStudentDataInFile(student_t student, FILE *filePointer)
{
    fprintf(filePointer, "%s", NAMETAG);
    fputc(TAGDELIMITERCHAR, filePointer);
    fprintf(filePointer, "%s", student.name);
    fputc(DELIMITERCHAR, filePointer);
    fprintf(filePointer, "%s", AGETAG);
    fputc(TAGDELIMITERCHAR, filePointer);
    fprintf(filePointer, "%d", student.age);
    fputc(DELIMITERCHAR, filePointer);
    fprintf(filePointer, "%s", GRADETAG);
    fputc(TAGDELIMITERCHAR, filePointer);
    fprintf(filePointer, "%d", student.grade);
    fputc(DELIMITERCHAR, filePointer);
}
```

# StudentTagDelimiterFileHandler.c (3/3)

```
student_t readTagDelimitedStudentDataInFile(FILE *filePointer)
{
    student_t student;
    int nFields = 0;
    char currentInput[50], currentTag[50];
    do {
        readCurrentInput(filePointer, currentTag, TAGDELIMITERCHAR);
        readCurrentInput(filePointer, currentInput, DELIMITERCHAR);
        if(strcmp(currentTag, NAMETAG) == 0)
            strcpy(student.name, currentInput);
        else if(strcmp(currentTag, AGETAG) == 0)
            student.age = atoi(currentInput);
        else if(strcmp(currentTag, GRADETAG) == 0)
            student.grade = atoi(currentInput);
        nFields++;
    }while(nFields < TOTALFIELDS);
    return student;
}
```

Como escalar isso para vários campos?

# Organização em Registros!

# Organização em Registros

- Registro:
  - ◆ Um conjunto de campos agrupados
- Arquivo organizado em registros
  - ◆ Nível de organização mais alto

# Organização em Registros

- Assim como os campos, um registro é uma ferramenta conceitual
  - ◆ Está associado ao arquivo lógico e não à organização física
  - ◆ Outro nível de organização imposto aos dados para preservar significado semântico



# Organização em Registros

## → Estruturas de Organização de Registros:

### ◆ Tamanho fixo

- Campos de tamanho fixo
- Campos de tamanho variável

### ◆ Tamanho variável

- Número pré-determinado de campos
- Uso de delimitadores
- Indicador de tamanho
- Uso de índice

# Organização em Registros

→ Veremos na próxima aula :)

## Referências

- M. J. Folk, B. Zoellick and G. Riccardi. File Structures: An object-oriented approach with C++, Addison Wesley, 1998.
- Códigos da aula estão no repositório <https://github.com/LeonardoTPereira/Alg2/tree/main/Aula%2003>