SCC 503 - ALGORITMOS E ESTRUTURAS DE DADOS II

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

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        ();  /* cria arquivo com os dados */
student_t       readStudentDataFromInput        (); /* lê um registro de dados */
void     writeStudentDataInFile          ( student_t     , FILE    *); /* escreve um registro no                arq    */
void     readStudentFile       (); /* lê o conteúdo do arquivo */
student_t       readStudentDataInFile         (FILE  *); /* lê um registro do arquivo */
void     printStudentData       ( student_t     ); /* imprime um registro na saída */
#endif     //STUDENTSTREAMFILEHANDLER_H
```

StudentStreamFileHandler.c (1/3)

```
#include      " StudentStreamFileHandler.h          “

/* cria o arquivo com os dados: um a um, lê os registros da entrada e
      escreve no arquivo */

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       );
}
```

StudentStreamFileHandler.c (1/3)

```
#include      " StudentStreamFileHandler.h          "

/* lê um registro da entrada padrão e retorna (um apontador para) a variável
      que armazena esse       esse   registro */

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)

```
/* escreve no arquivo o conteúdo do registro, como uma sequência
      de bytes */

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

StudentStreamFileHandler.c (2/3)

```
/*    Lê o conteúdo do arquivo e imprime na saída padrão, registro a registro                    */


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)

```
/* lê um registro          do arquivo e       retorna (um apontador para)
      a variável que        armazena   esse   registro */


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   ;
}
```

StudentStreamFileHandler.c (3/3)

```
/* escreve      na saída padrão o       conteúdo do     registro      */


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: ■   ╕■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 fossem salvos concatenados, sem uma separação
- ➔ 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 A03255509Rio 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

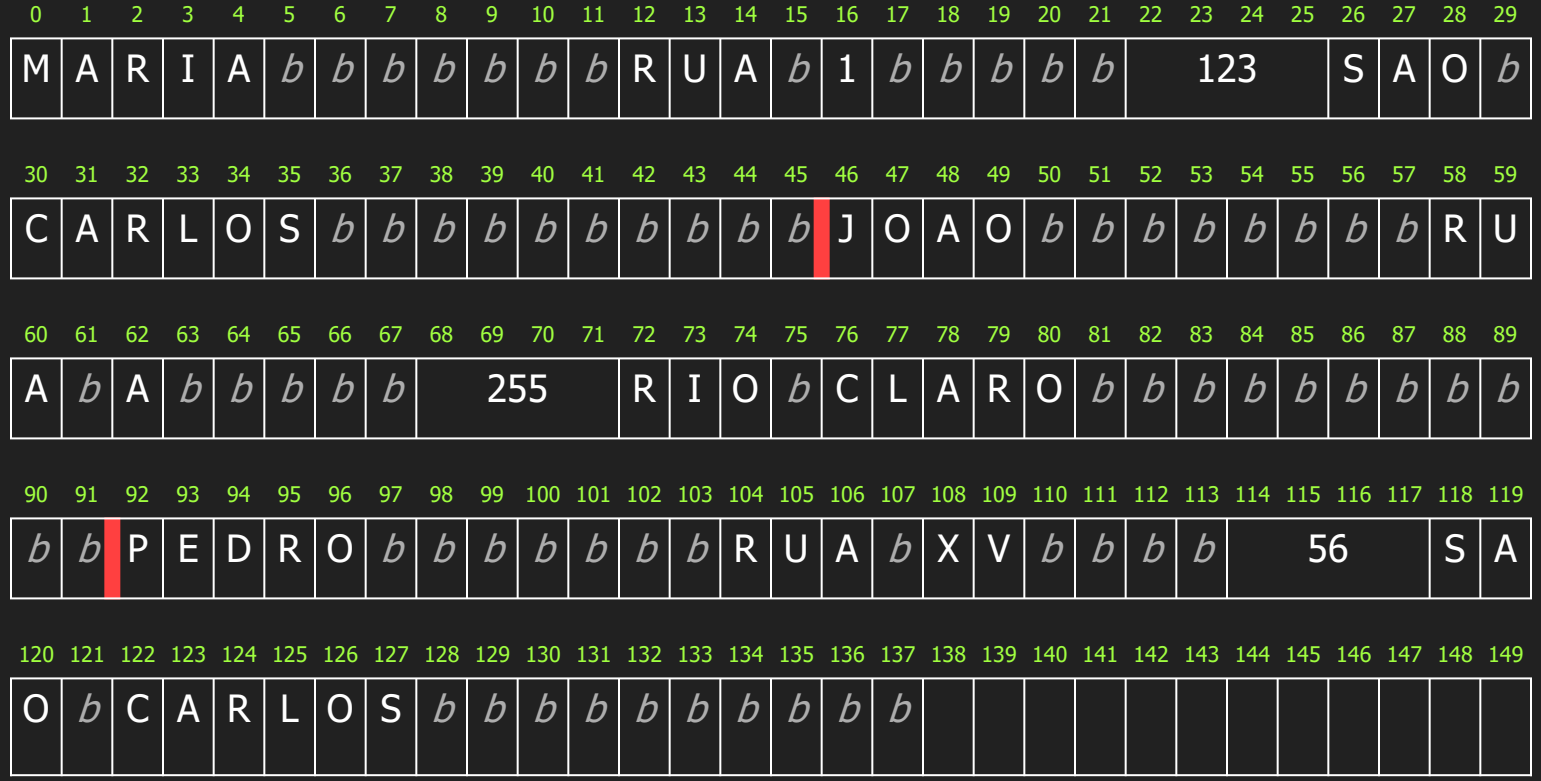
Campos com Tamanho Fixo

- ➔ 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



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 (fragmentação interna)

Campos com Tamanho Fixo

- ➔ Prós e Contras?
  - ➔ 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

Campos com Tamanho Fixo

➔ Vamos revisitar nosso código anterior...

main.c

```
#include      " StudentFixedSizeFieldFileHandler.h                "
/* criar o mesmo arquivo, mas com campos de tamanho fixo */
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      " StudentFixedSizeFieldFileHandler.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       ();
                writeFixedSizeFieldStudentDataInFile              ( 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      =  readFixedSizeFieldStudentDataInFile              ( 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

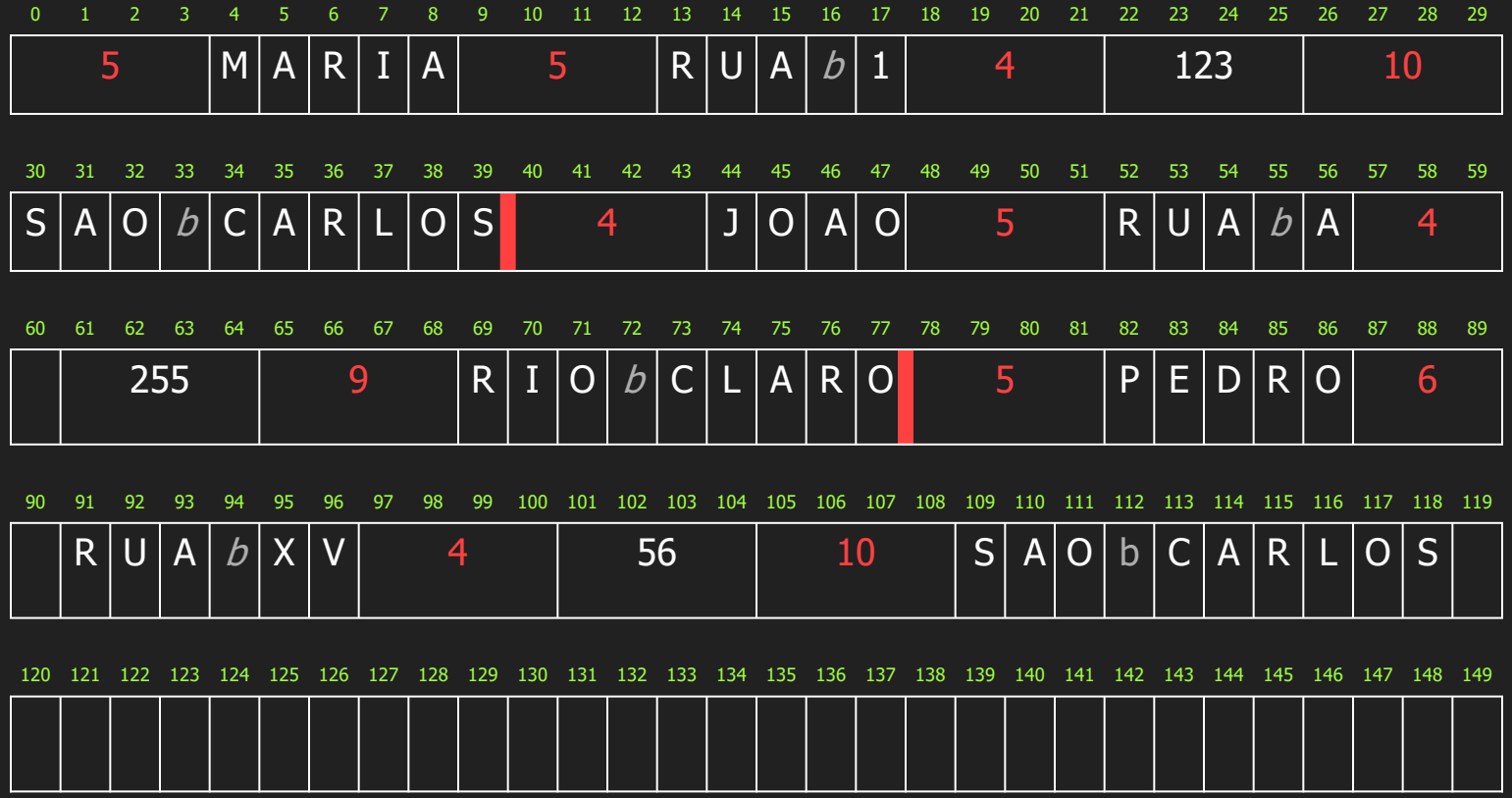
```
#include      "StudentFixedSizeFieldFileHandler.h"

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



StudentFieldLenghtIndicatorFileHandler.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_
```

StudentFieldLenghtIndicatorFileHandler.c (1/2)

```
#include    " StudentFieldLenghtIndicatorFileHandler.h                "

void    writeLenghtIndicatorStudentDataInFile              ( 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       );
}
```

StudentFieldLenghtIndicatorFileHandler.c (2/2)

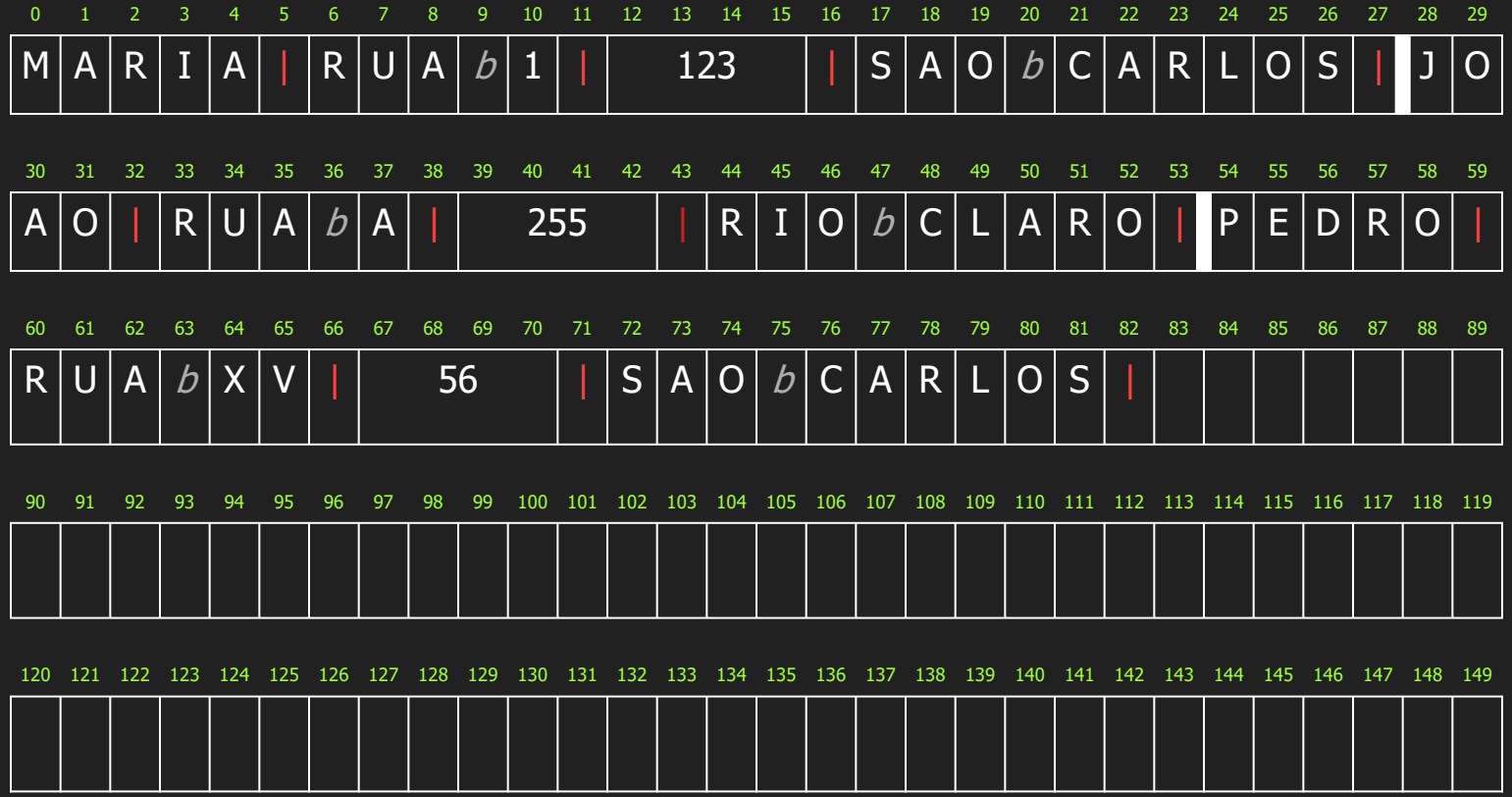
```
#include " StudentFieldLenghtIndicatorFileHandler.h "

student_t readLenghtIndicatorStudentDataInFile (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



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)

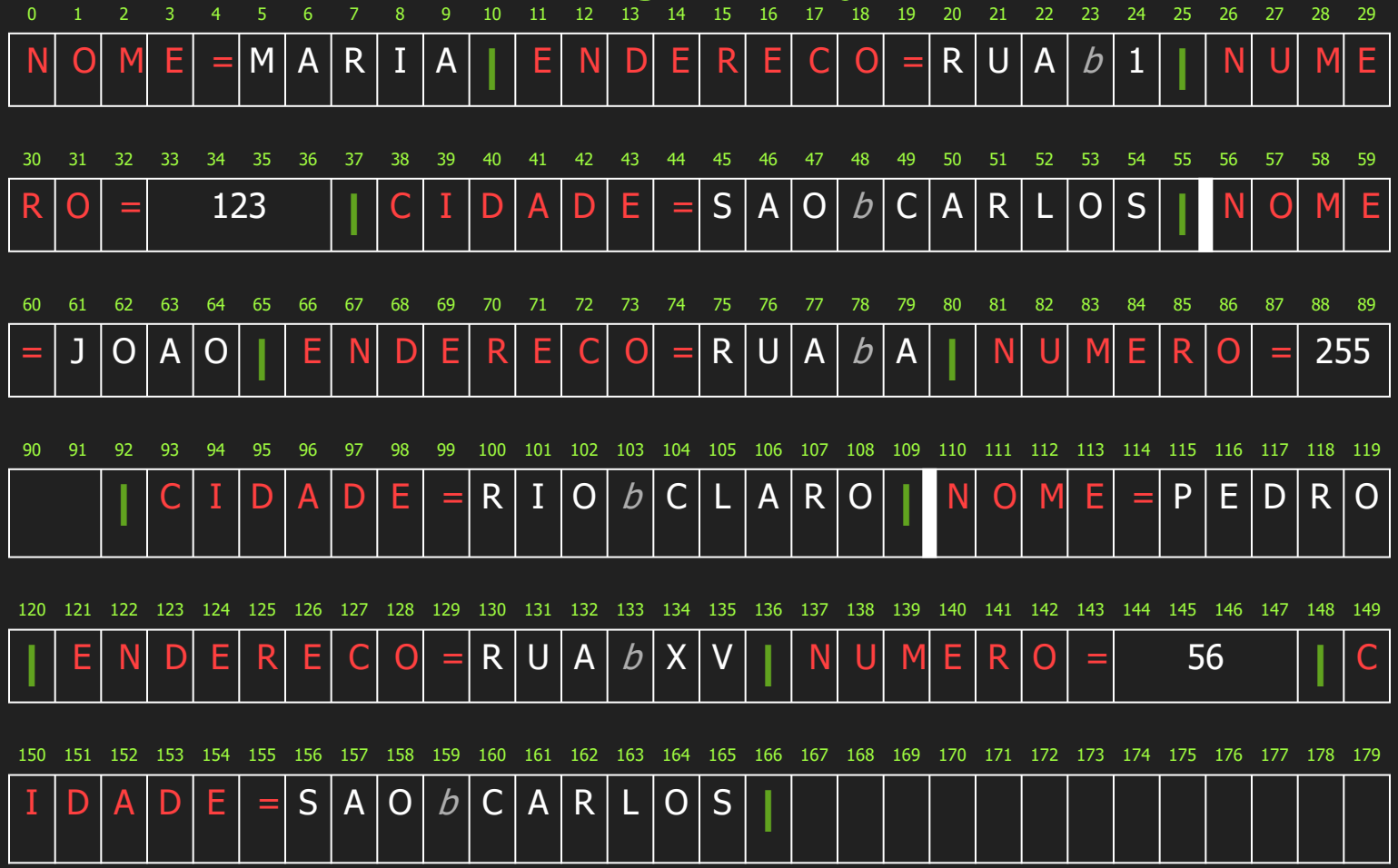
```
student_t      readDelimitedStudentDataInFile          (FILE *    filePointer       )
{
       student_t      student   ;
       char    currentInput      [ 50];

       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”



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

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